

# *Lecture 9: Visualizing CNNs and Recurrent Neural Networks*

*Tuesday February 28, 2017*

# Announcements!

- HW #3 is out
- Final Project proposals due this **Thursday March 2**
- Papers to read: Students should read all papers on the **Schedule** tab, and are encouraged to read as many papers as possible from the **Papers** tab.
- Next paper: **March 7** *You Only Look Once: Unified, Real-Time Object Detection*. If this paper seems too deep or confusing, look at *Fast R-CNN*, *Faster R-CNN*

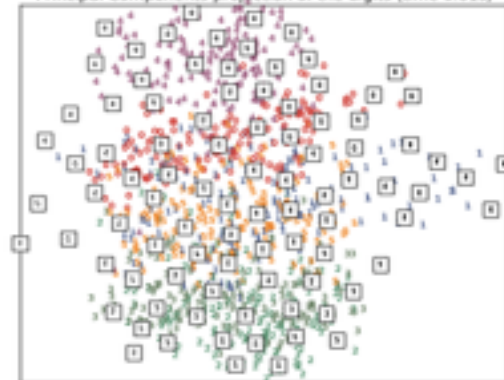
# Python/Numpy of the Day

- t-SNE (t-Distributed Stochastic Nearest Neighbor Embedding)
  - [Scikit-Learn t-SNE](#)
  - [Examples of 2D Embedding Visualizations of MNIST dataset](#)
  - [Other Embedding functions in Scikit-Learn](#)

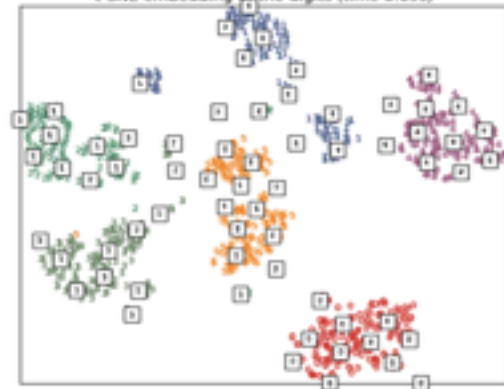
A selection from the 64-dimensional digits dataset



Principal Components projection of the digits (time 0.01s)

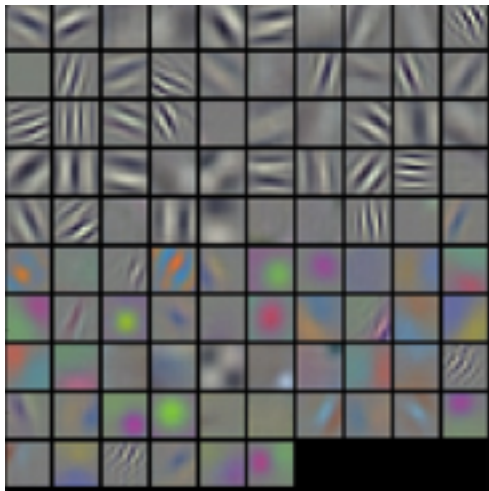


t-SNE embedding of the digits (time 5.69s)



# Visualizing CNN Behavior

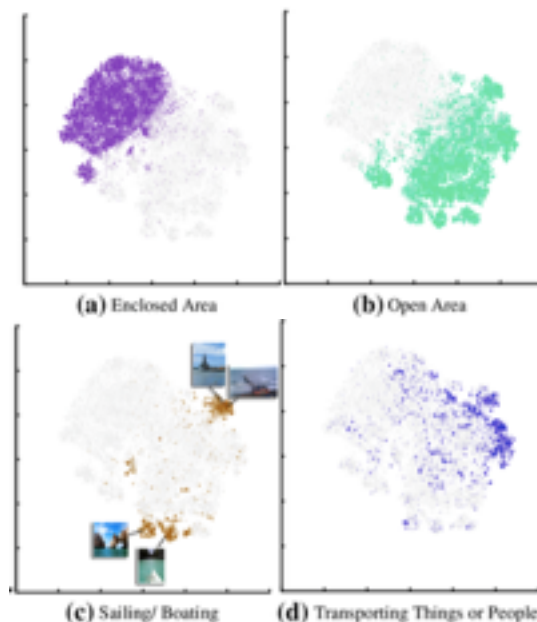
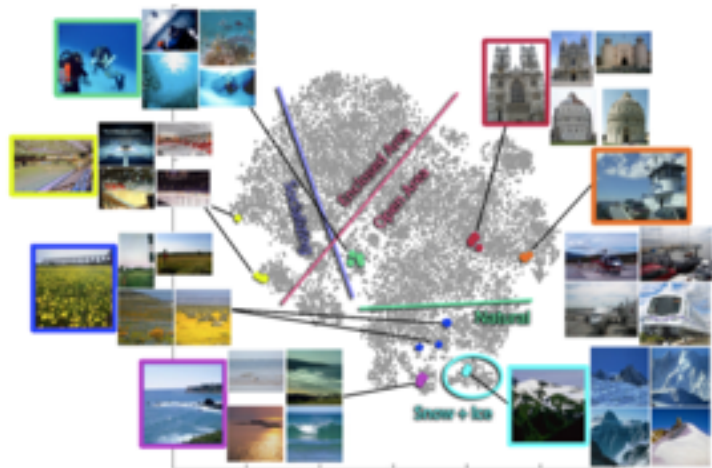
- How can we see what's going on in a CNN?
- *Stuff we've already done:*
  - Visualize the weights
  - Occlusion experiments — ex. Jason and Lisa's AlexNet Occlusion Tests





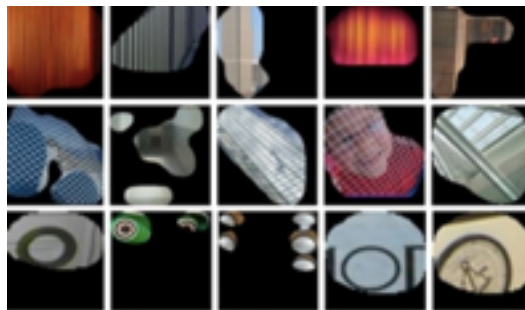
# Visualizing CNN Behavior

- How can we see what's going on in a CNN?
- *Straightforward stuff to try in the future:*
  - Visualize the representation space (e.g. with t-SNE)
  - Human experiment comparisons



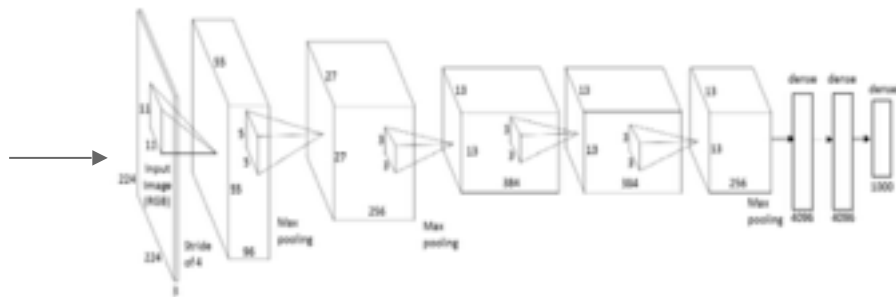
# Visualizing CNN Behavior

- How can we see what's going on in a CNN?
  - *More sophisticated approaches (HW #4)*
    - Visualize patches that maximally activate neurons
    - Optimization over image approaches (optimization)
    - Deconv approaches (single backward pass)



# Deconv approaches - projecting backward from one neuron to see what is activating it

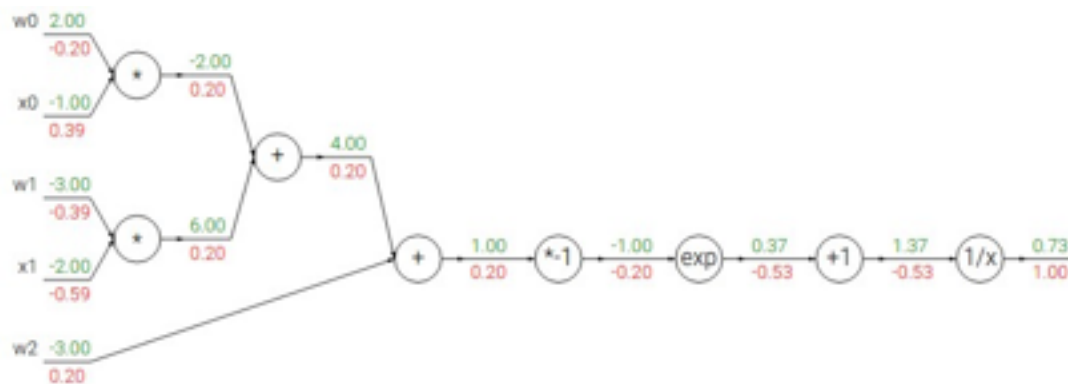
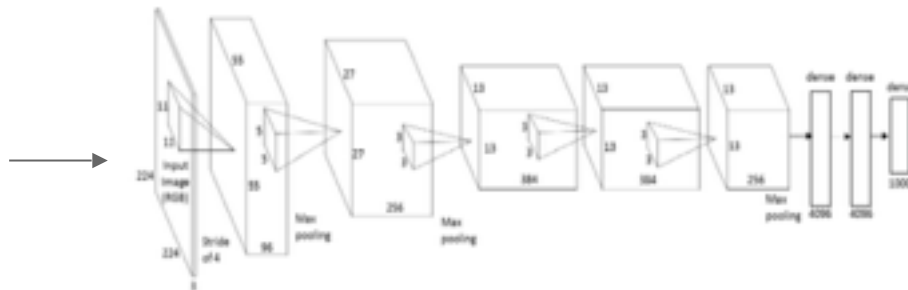
## 1. *Feed image into net*



*Q: how can we compute the gradient of any arbitrary neuron in the network w.r.t. the image?*

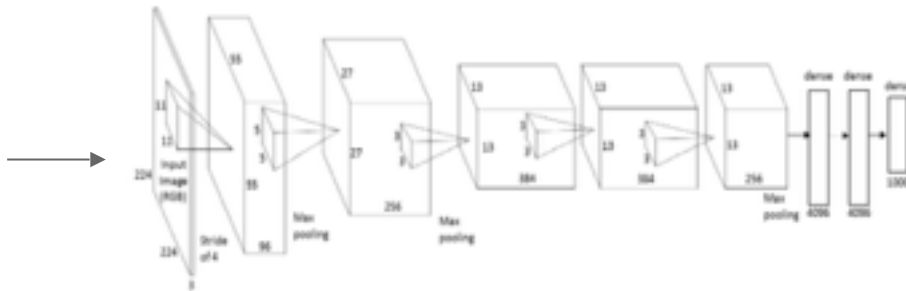
# Deconv approaches

## 1. Feed image into net



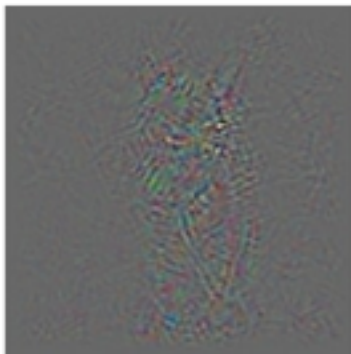
# Deconv approaches

## 1. Feed image into net



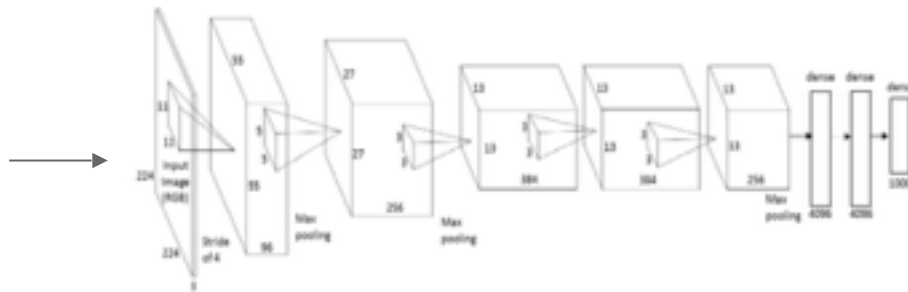
## 2. Pick a layer, set the gradient there to be all zero except for one 1 for some neuron of interest

## 3. Backprop to image:

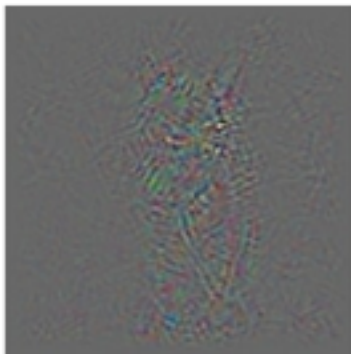


# Deconv approaches

## 1. Feed image into net



2. Pick a layer, set the gradient there to be all zero except for one 1 for some neuron of interest
3. Backprop to image:



**“Guided  
backpropagation:”**  
only propagate  
positive gradients

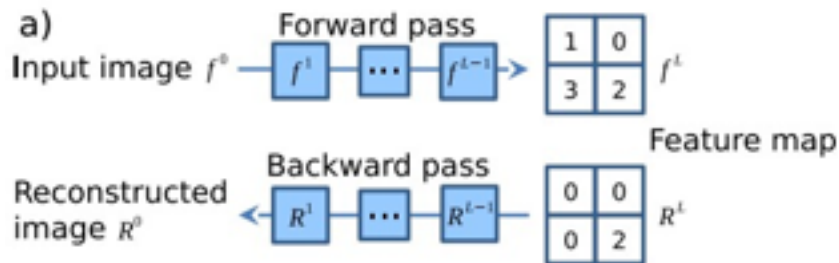


# Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]

[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]

[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]

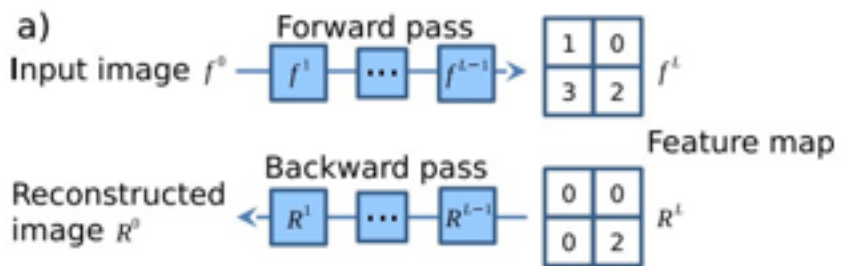


# Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]

[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]

[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



c) activation:  $f_i^{l+1} = \text{relu}(f_i^l) = \max(f_i^l, 0)$

backpropagation:  $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$ , where  $R_i^{l+1} = \frac{\partial f^{\text{out}}}{\partial f_i^{l+1}}$

Backward pass for a ReLU (will be changed in Guided Backprop)

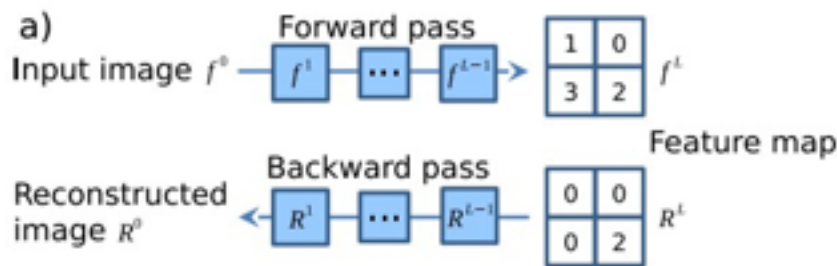


# Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]

[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]

[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



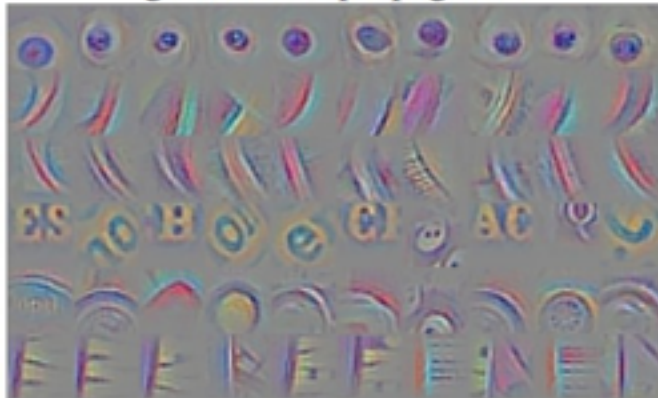
c) activation:  $f_i^{l+1} = \text{relu}(f_i^l) = \max(f_i^l, 0)$

backpropagation:  $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$ , where  $R_i^{l+1} = \frac{\partial f^{\text{out}}}{\partial f_i^{l+1}}$

guided backpropagation:  $R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$



guided backpropagation



corresponding image crops

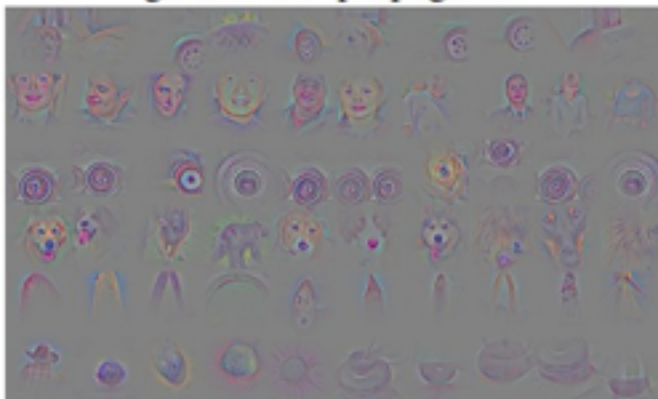


Visualization of patterns learned by the layer **conv6** (top) and layer **conv9** (bottom) of the network trained on ImageNet.

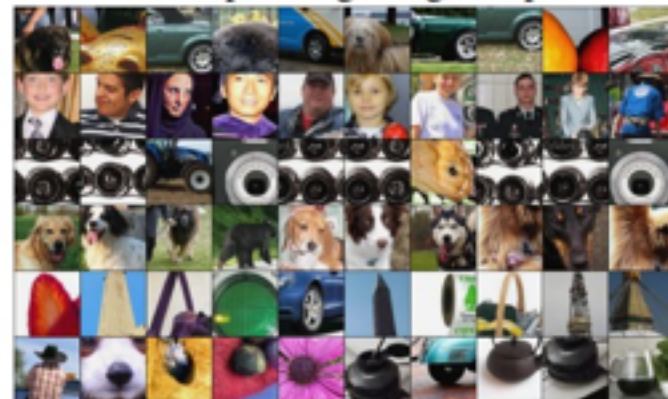
Each row corresponds to one filter.

The visualization using “guided backpropagation” is based on the top 10 image patches activating this filter taken from the ImageNet dataset.

guided backpropagation



corresponding image crops



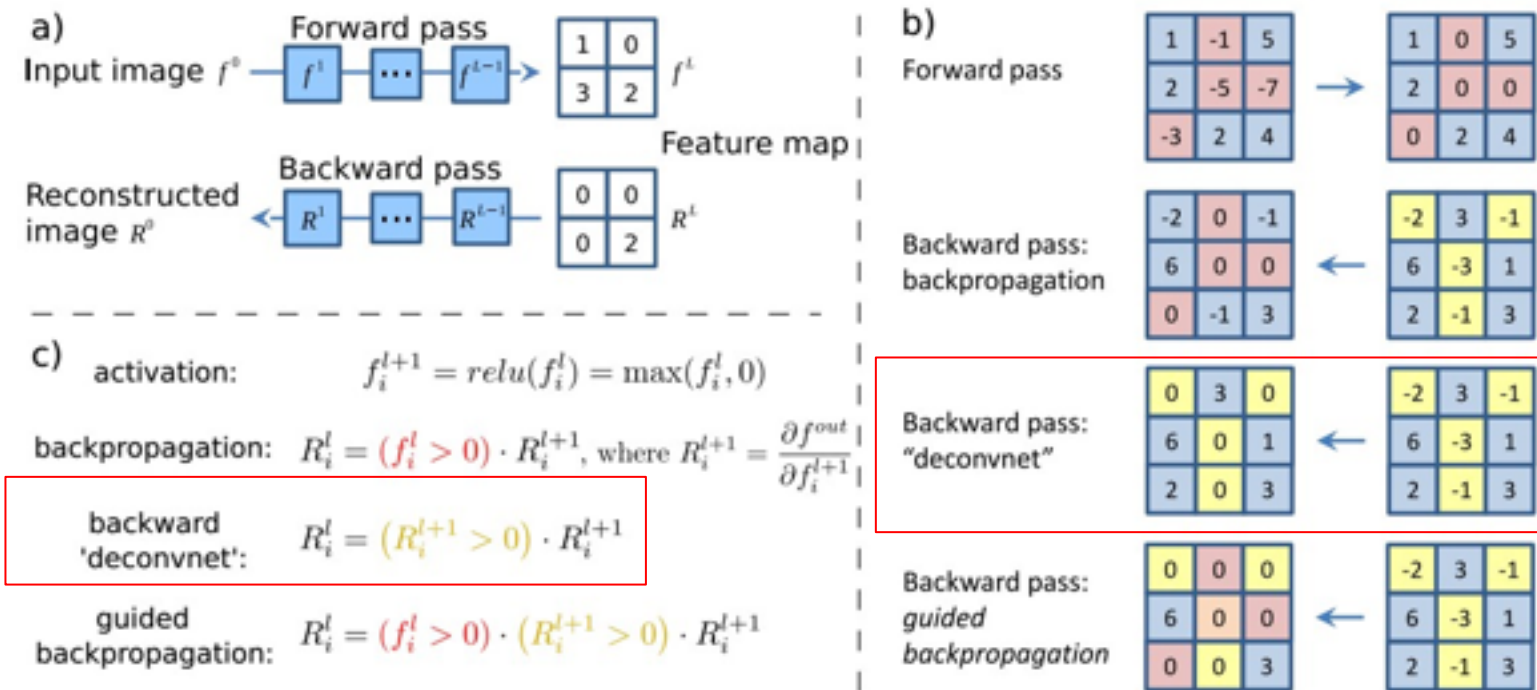
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]

# Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]

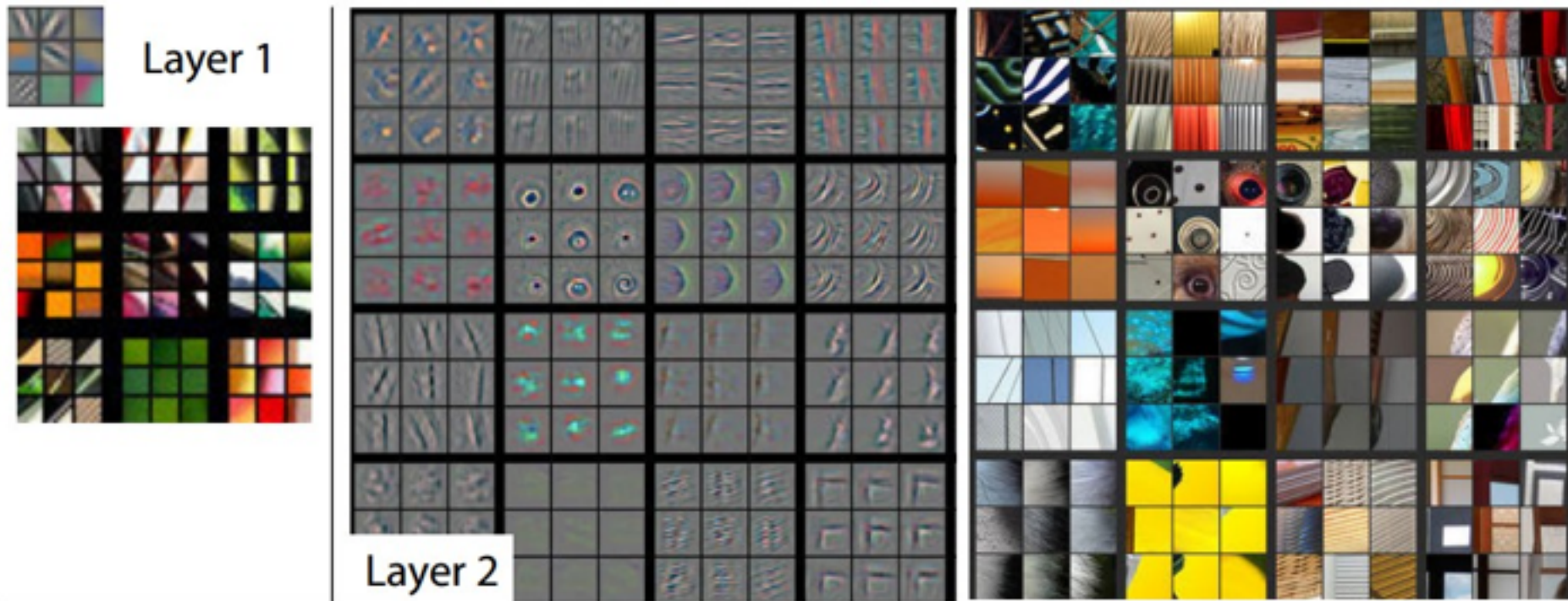
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]

[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



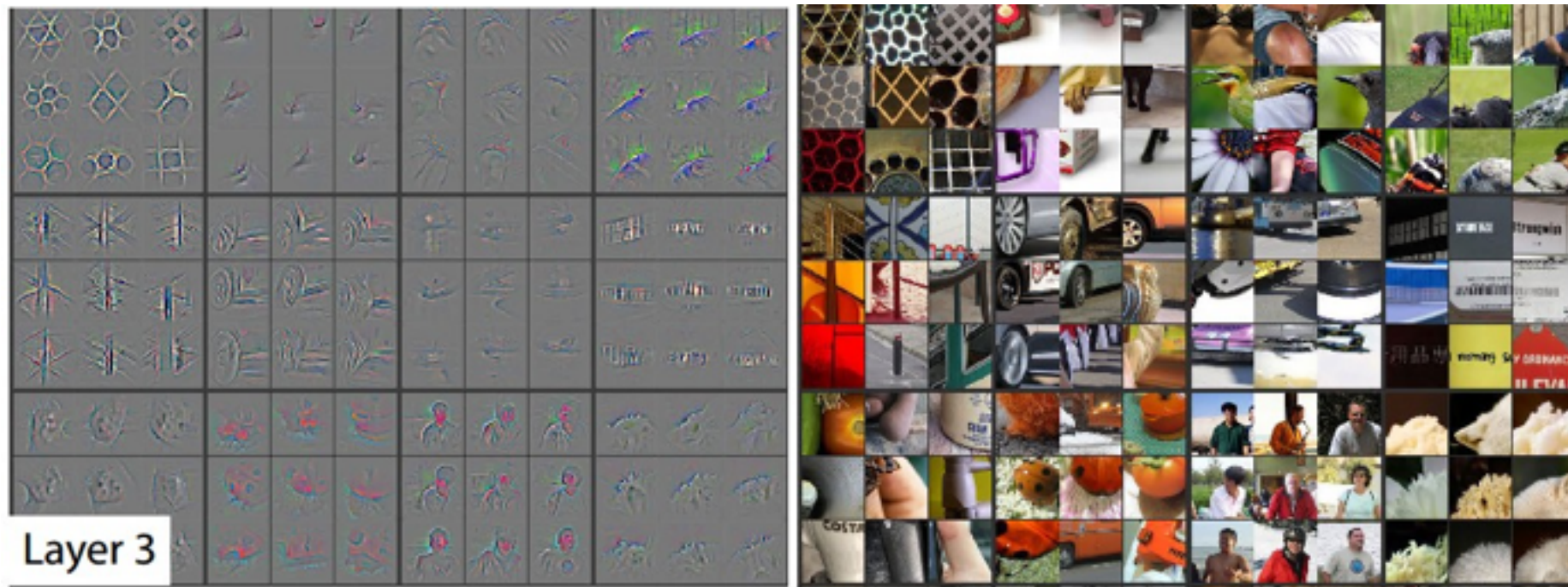
*backprops  
to weights  
that were  
zero-d out  
by ReLu*

## *Visualizing arbitrary neurons along the way to the top...*

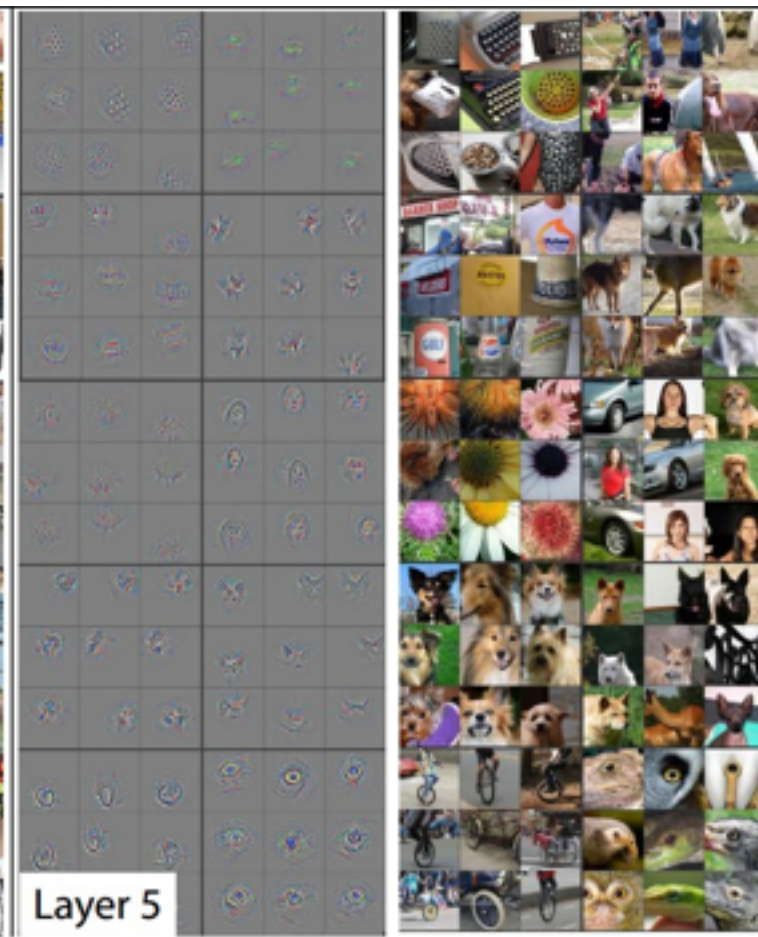
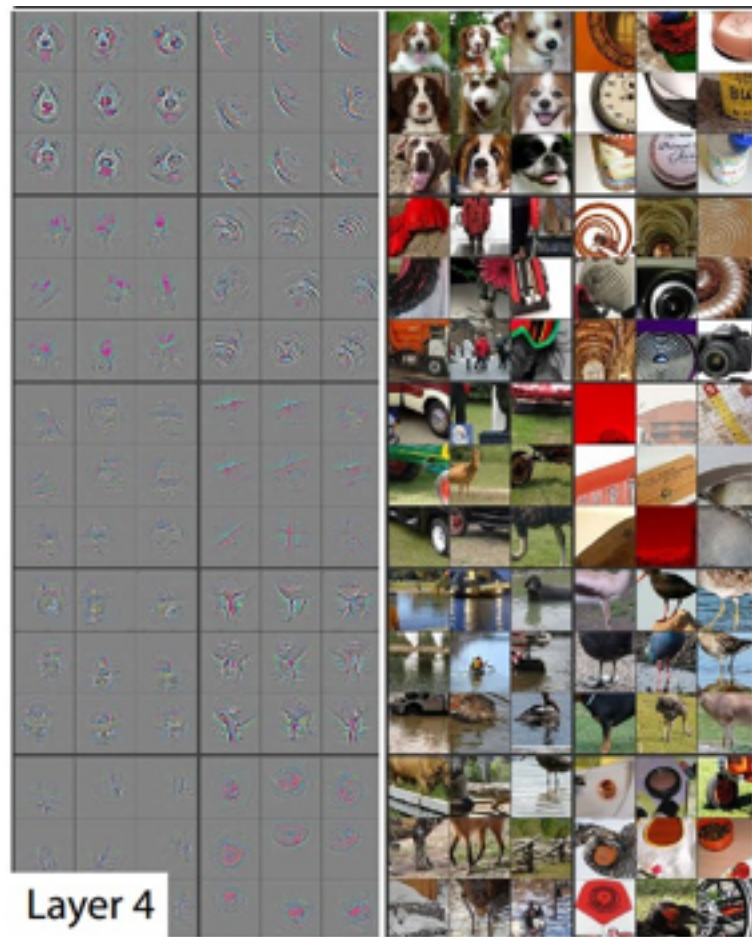




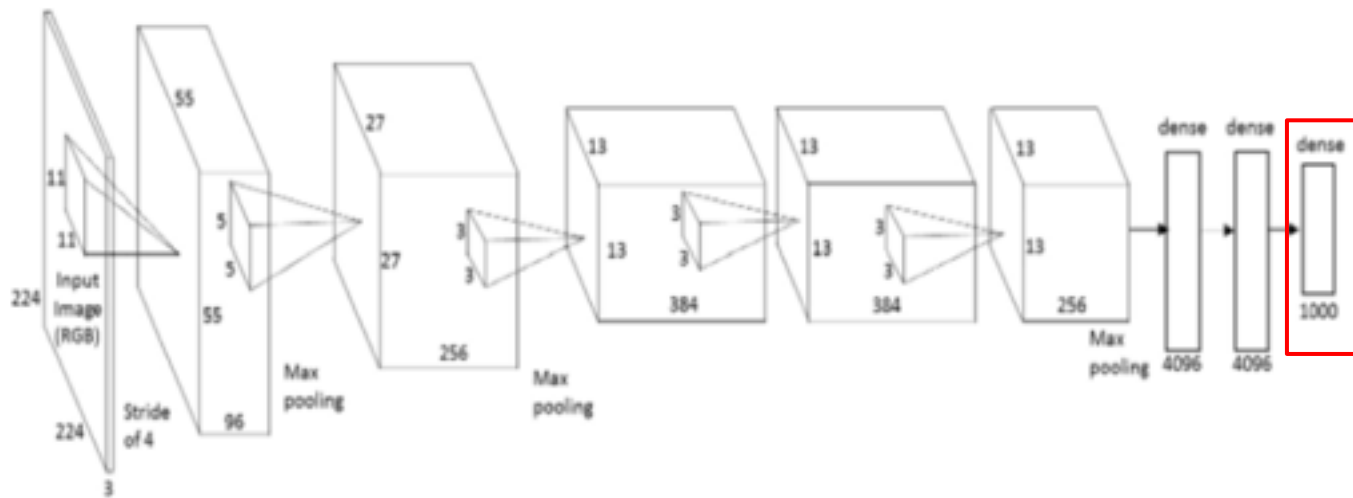
# Visualizing arbitrary neurons along the way to the top...



*Visualizing  
arbitrary  
neurons along  
the way to the  
top...*



# Optimization to Image

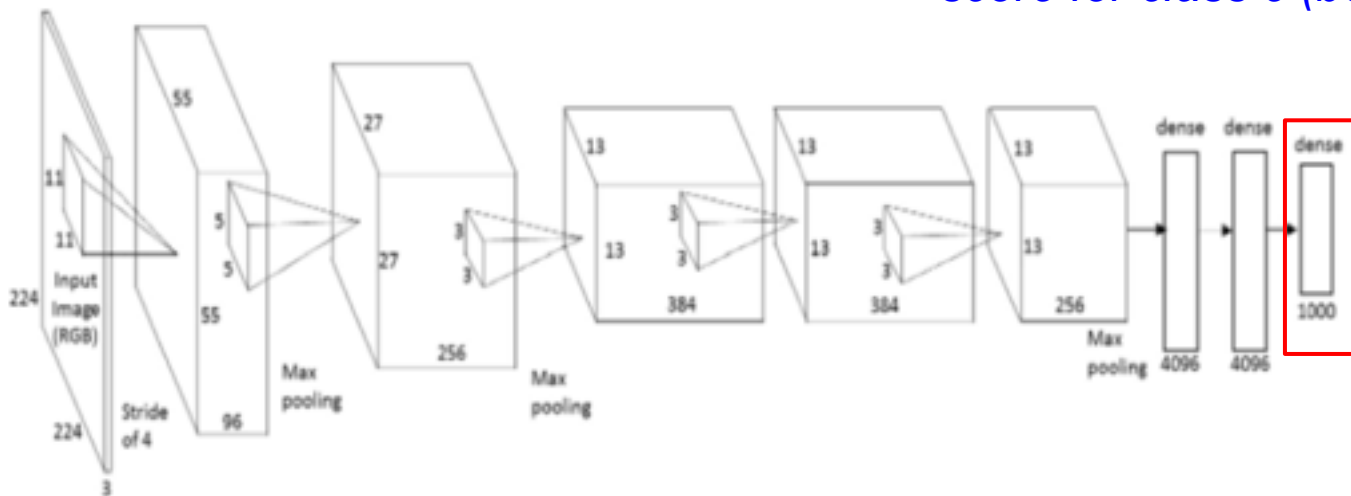


*Q: can we find an image that maximizes some class score?*

# Optimization to Image

$$\arg \max_I \boxed{S_c(I)} - \lambda \|I\|_2^2$$

*score for class c (before Softmax)*

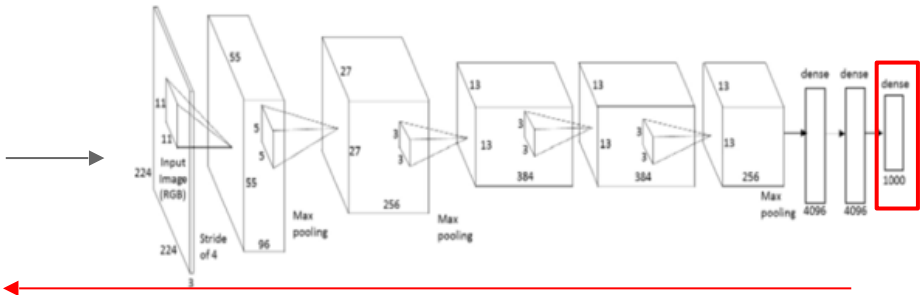
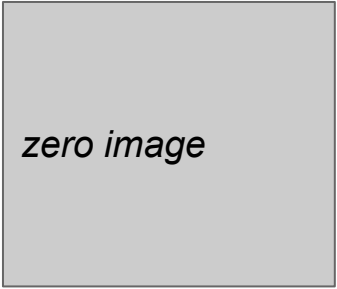


*Q: can we find an image that maximizes some class score?*



# Optimization to Image

1. feed in zeros.

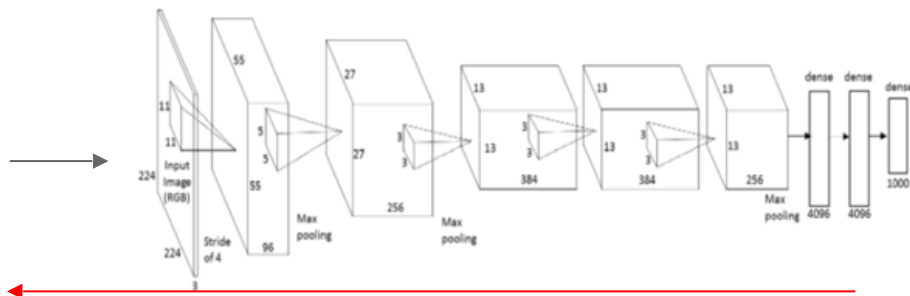


2. set the gradient of the scores vector to be  $[0, 0, \dots, 1, \dots, 0]$ , then backprop to image

\* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

# Optimization to Image

1. feed in zeros.



2. set the gradient of the scores vector to be  $[0, 0, \dots, 1, \dots, 0]$ , then backprop to image

3. do a small “image update”

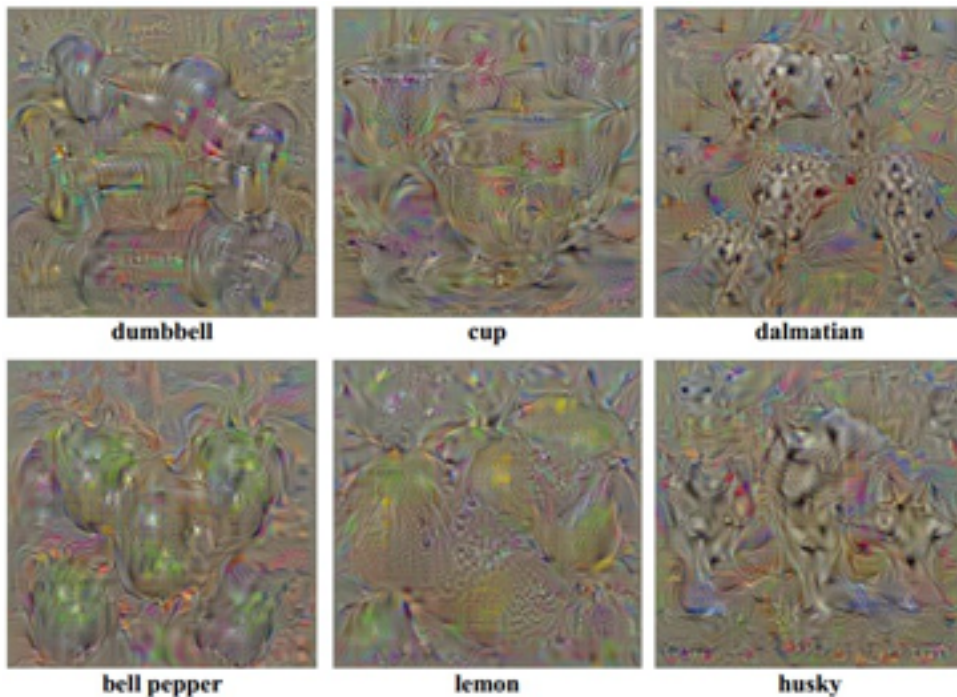
4. forward the image through the network.

5. go back to 2.

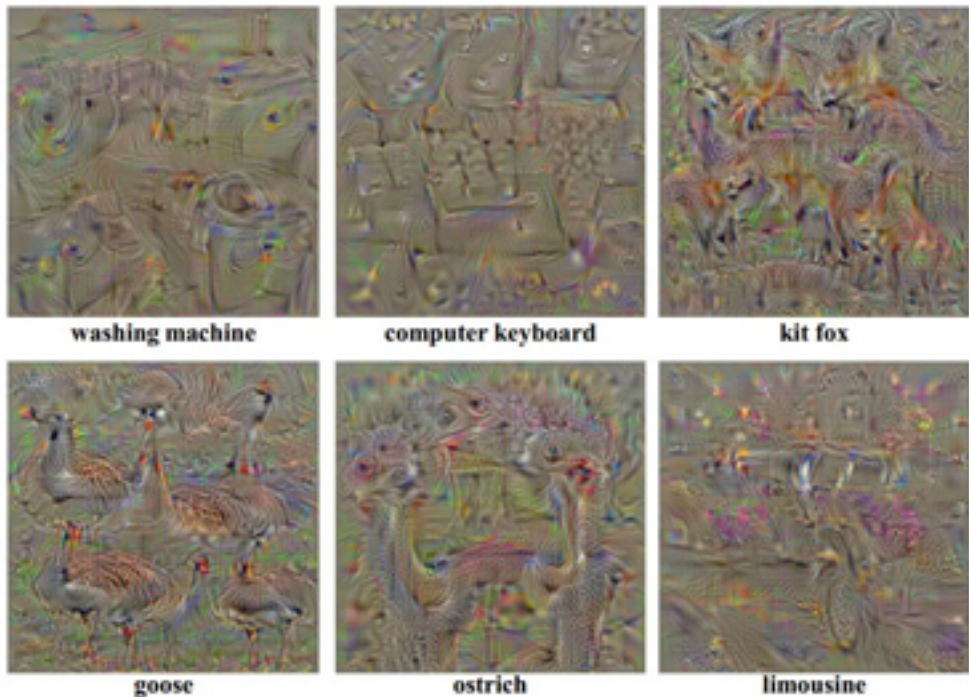
$$\arg \max_I \boxed{S_c(I)} - \lambda \|I\|_2^2$$

score for class  $c$  (before Softmax)

# *1. Find images that maximize some class score:*



# *1. Find images that maximize some class score:*



## 2. Visualize the Data gradient:

(note that the gradient on data has three channels.  
Here they visualize  $M$ , s.t.:

$$M_{ij} = \max_c |w_{h(i,j,c)}|$$

(at each pixel take abs val, and max over channels)



$$M = ?$$

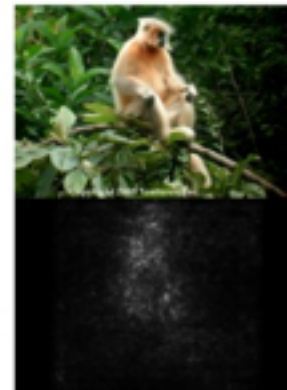
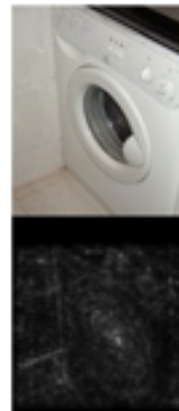
## 2. Visualize the Data gradient:

(note that the gradient on data has three channels.

Here they visualize  $M$ , s.t.:

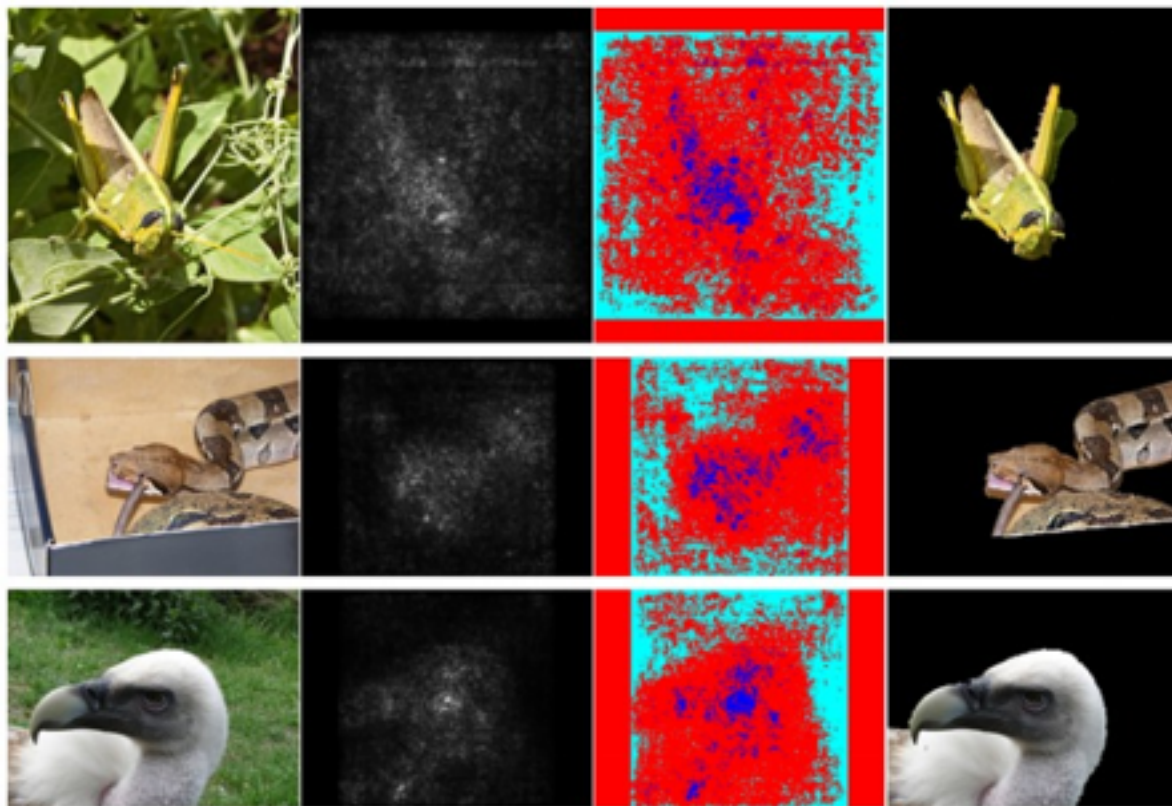
$$M_{ij} = \max_c |w_{h(i,j,c)}|$$

(at each pixel take abs val, and max over channels)

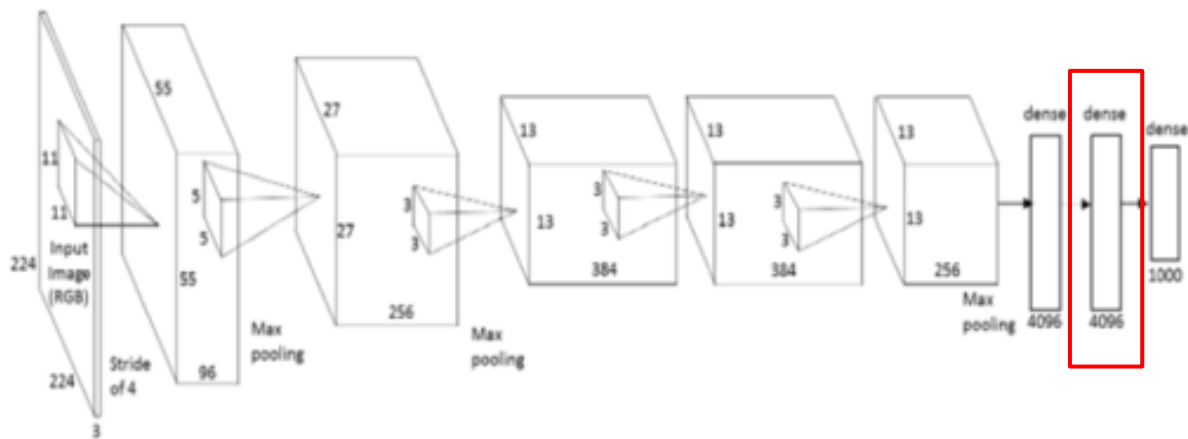




- Use **grabcut** for segmentation
- This optimization can be done for arbitrary neurons in the CNN



Question: Given a CNN **code**, is it possible to reconstruct the original image?



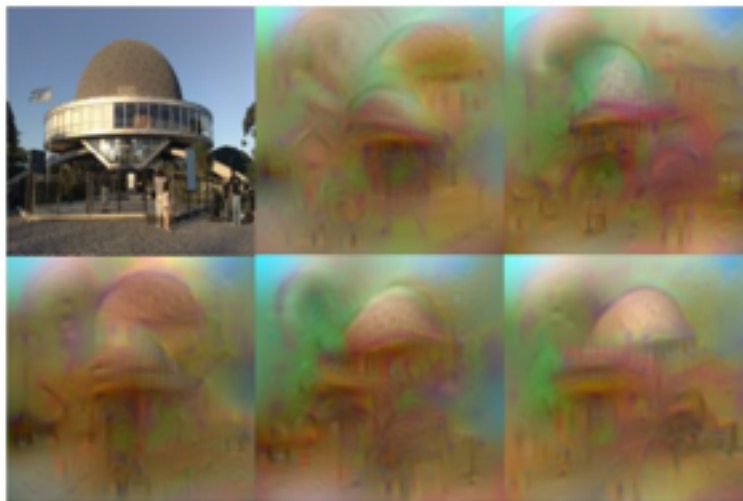


# *Understanding Deep Image Representations by Inverting Them* *[Mahendran and Vedaldi, 2014]*

Find an image such that:

- Its code is similar to a given code
- It “looks natural” (image prior regularization)

*original image*



*reconstructions  
from the 1000  
log probabilities  
for ImageNet  
(ILSVRC)  
classes*



DeepDream <https://github.com/google/deepdream>

```

def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
             jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst) # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)

```



```
def objective_L2(dst):  
    dst.diff[:] = dst.data
```

*DeepDream: set dx = x :)*

```
def make_step(net, step_size=1.5, end='inception_4c/output',  
             jitter=32, clip=True, objective=objective_L2):  
    '''Basic gradient ascent step.'''
```

```
    src = net.blobs['data'] # input image is stored in Net's 'data' blob  
    dst = net.blobs[end]
```

```
    ox, oy = np.random.randint(-jitter, jitter+1, 2)  
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift
```

```
    net.forward(end=end)  
    objective(dst) # specify the optimization objective  
    net.backward(start=end)
```

```
    g = src.diff[0]  
    # apply normalized ascent step to the input image  
    src.data[:] += step_size/np.abs(g).mean() * g
```

*“image update”*

```
    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image
```

```
    if clip:  
        bias = net.transformer.mean['data']  
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

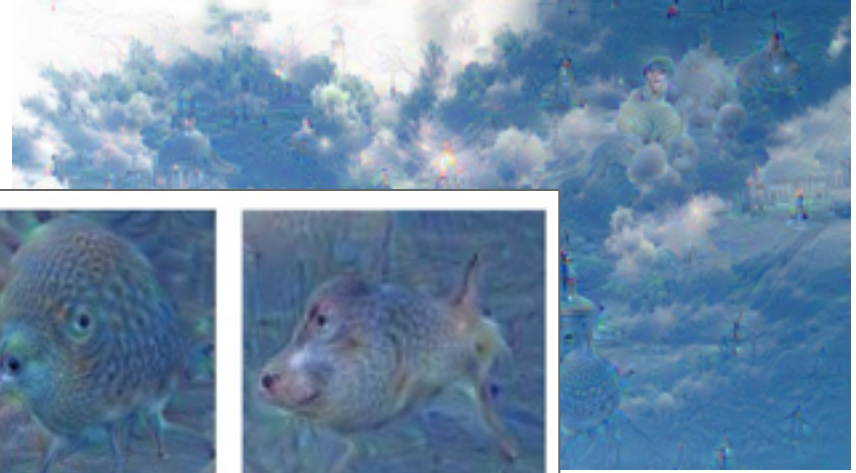
*jitter regularizer*

*inception\_4c/output*



DeepDream modifies the image in a way that “boosts” all activations, at any layer  
this creates a feedback loop: e.g. any slightly detected dog face will be made more  
and more dog like over time

*inception\_4c/output*



"Admiral Dog!"



"The Pig-Snail"



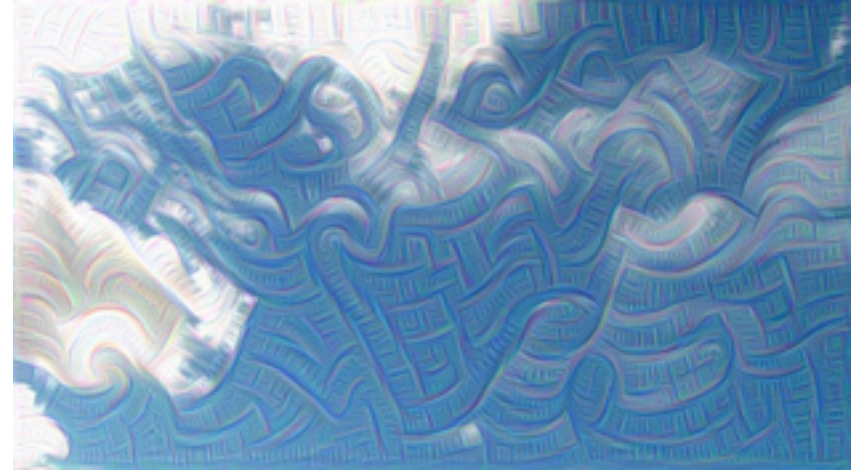
"The Camel-Bird"



"The Dog-Fish"

*DeepDream modifies the image in a way that boosts all activations, at any layer*

*inception\_3b/5x5\_reduce*



*DeepDream modifies the image in a way that “boosts” all activations, at any layer*



# NeuralStyle

[ A Neural Algorithm of Artistic Style by Leon A. Gatys,  
Alexander S. Ecker, and Matthias Bethge, 2015]

**good implementation by Justin in Torch:**  
<https://github.com/jcjohnson/neural-style>

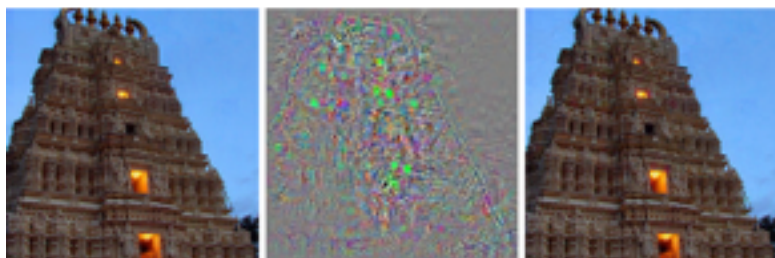
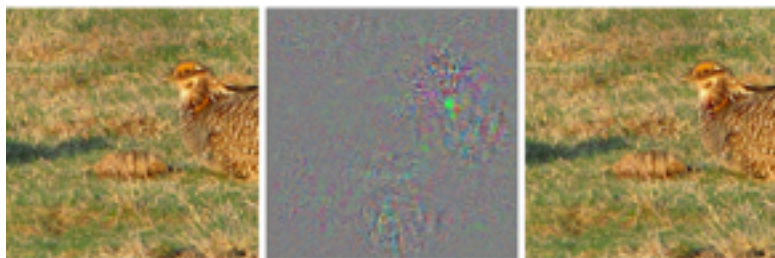
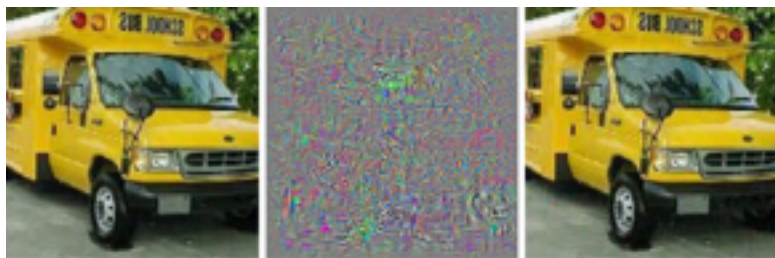




We can pose an optimization over the input image to maximize any class score.  
That seems useful.

Question: Can we use this to “fool” ConvNets?

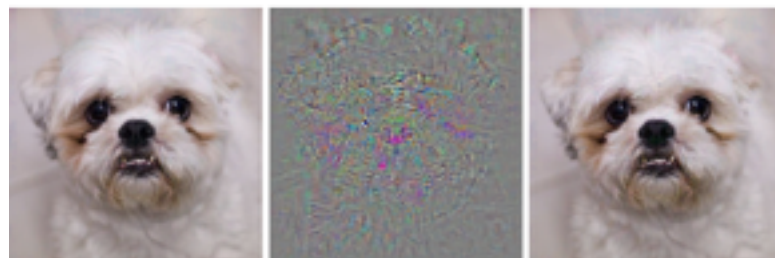
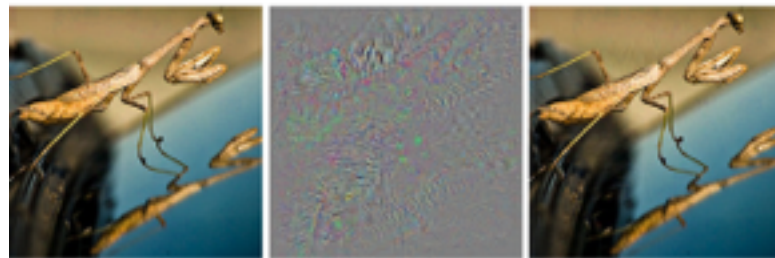
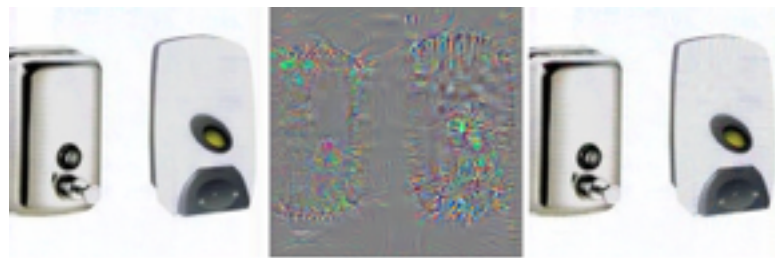
[Intriguing properties of neural networks, Szegedy et al., 2013]



*correct*

*+distort*

*ostrich*



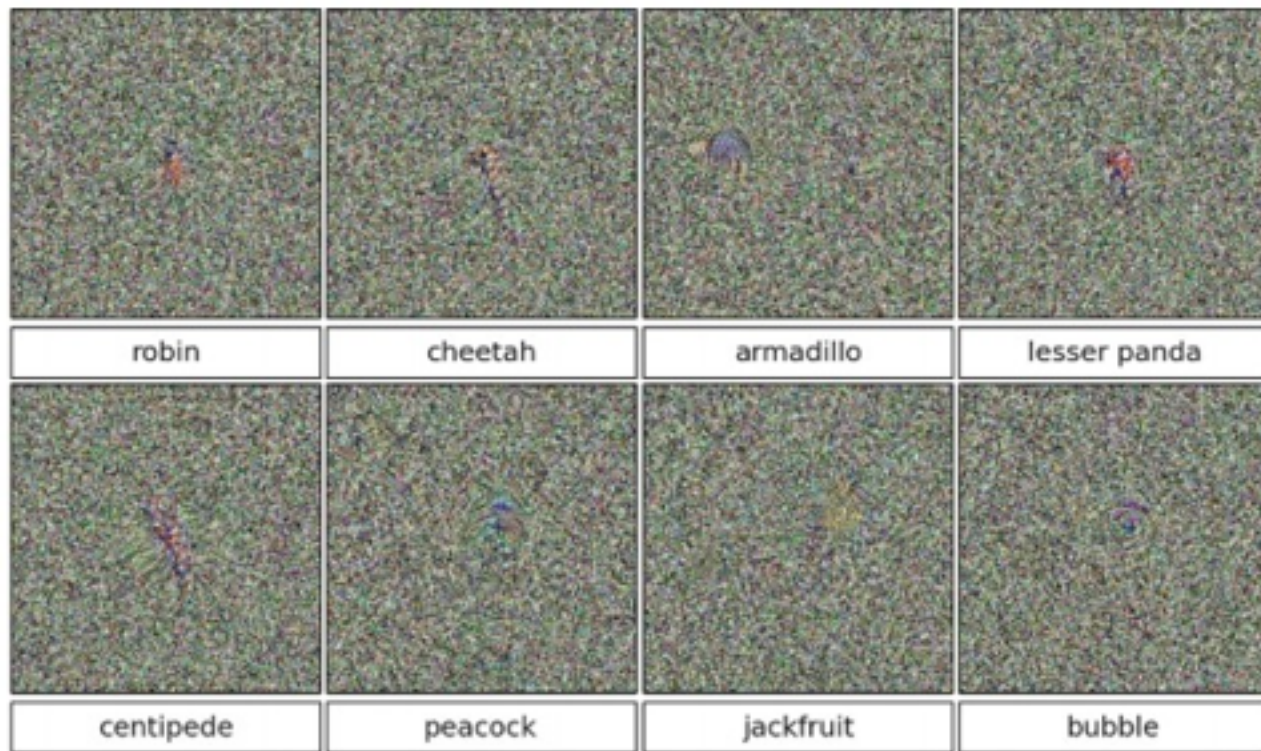
*correct*

*+distort*

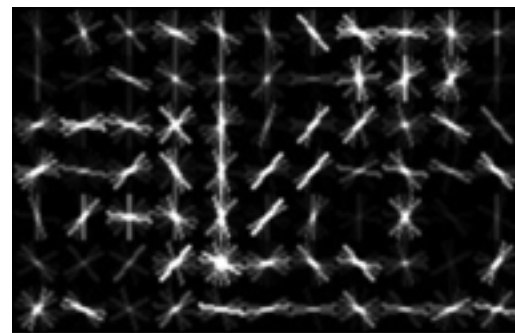
*ostrich*

*[Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images  
Nguyen, Yosinski, Clune, 2014]*

>99.6%  
confidences



*These kinds of results were around even before ConvNets...*  
*[Exploring the Representation Capabilities of the HOG Descriptor, Tatu et al., 2011]*

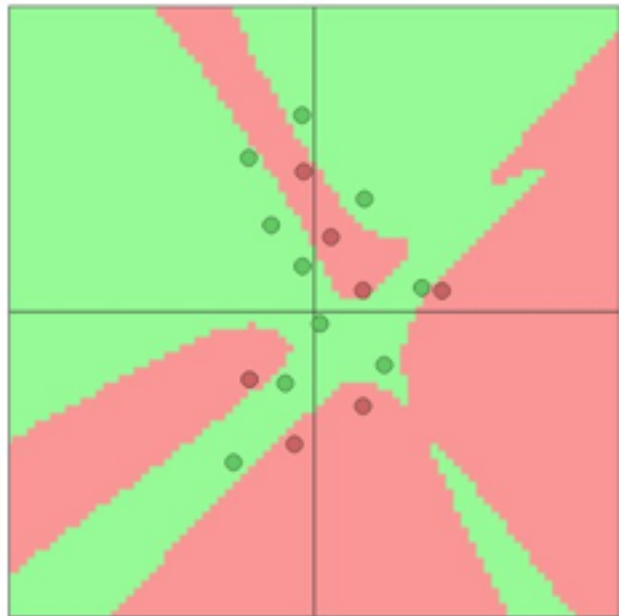


*Identical HOG representation*

# EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

[Goodfellow, Shlens & Szegedy, 2014]

*“primary cause of neural networks’ vulnerability to adversarial perturbation is their **linear nature**”*



## Lets fool a binary linear classifier:

$x$	2	-1	3	-2	2	2	1	-4	5	1	← <i>input example</i>
$w$	-1	-1	1	-1	1	-1	1	1	-1	1	← <i>weights</i>

$$P(y = 1 \mid x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$



## Lets fool a binary linear classifier:

$x$	2	-1	3	-2	2	2	1	-4	5	1	← <i>input example</i>
$w$	-1	-1	1	-1	1	-1	1	1	-1	1	← <i>weights</i>

*class 1 score = dot product:*

$$= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

*=> probability of class 1 is  $1/(1+e^{(-(-3))}) = 0.0474$*

*i.e. the classifier is **95%** certain that this is class 0 example.*

$$P(y = 1 \mid x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

## Lets fool a binary linear classifier:

$x$	2	-1	3	-2	2	2	1	-4	5	1	← <i>input example</i>
$w$	-1	-1	1	-1	1	-1	1	1	-1	1	← <i>weights</i>
<i>adversarial x</i>	?	?	?	?	?	?	?	?	?	?	

*class 1 score = dot product:*

$$= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

*=> probability of class 1 is  $1/(1+e^{(-(-3))}) = 0.0474$*

*i.e. the classifier is **95%** certain that this is class 0 example.*

$$P(y = 1 \mid x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

# Lets fool a binary linear classifier:

$x$	2	-1	3	-2	2	2	1	-4	5	1	← <i>input example</i>
$w$	-1	-1	1	-1	1	-1	1	1	-1	1	← <i>weights</i>
<i>adversarial x</i>	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5	

*class 1 score before:*

$$-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

$$\Rightarrow \text{probability of class 1 is } 1/(1+e^{(-(-3))}) = 0.0474$$

$$\textcolor{red}{-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2}$$

$$\Rightarrow \text{probability of class 1 is now } 1/(1+e^{(-(2))}) = 0.88$$

***i.e. we improved the class 1 probability from 5% to 88%***

$$P(y = 1 \mid x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

## Lets fool a binary linear classifier:

$X$	2	-1	3	-2	2	2	1	-4	5	1	← <i>input example</i>
$W$	-1	-1	1	-1	1	-1	1	1	-1	1	← <i>weights</i>
<i>adversarial x</i>	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5	

*class 1 score before:*

$$-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3$$

$$\Rightarrow \text{probability of class 1 is } 1/(1+e^{(-(-3))}) = 0.0474$$

$$\textcolor{red}{-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2}$$

$$\Rightarrow \text{probability of class 1 is now } 1/(1+e^{(-(-2))}) = 0.88$$

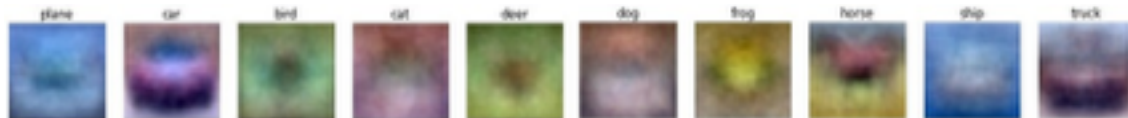
***i.e. we improved the class 1 probability from 5% to 88%***

*This was only with 10 input dimensions. A 224x224 input image has 150,528.*

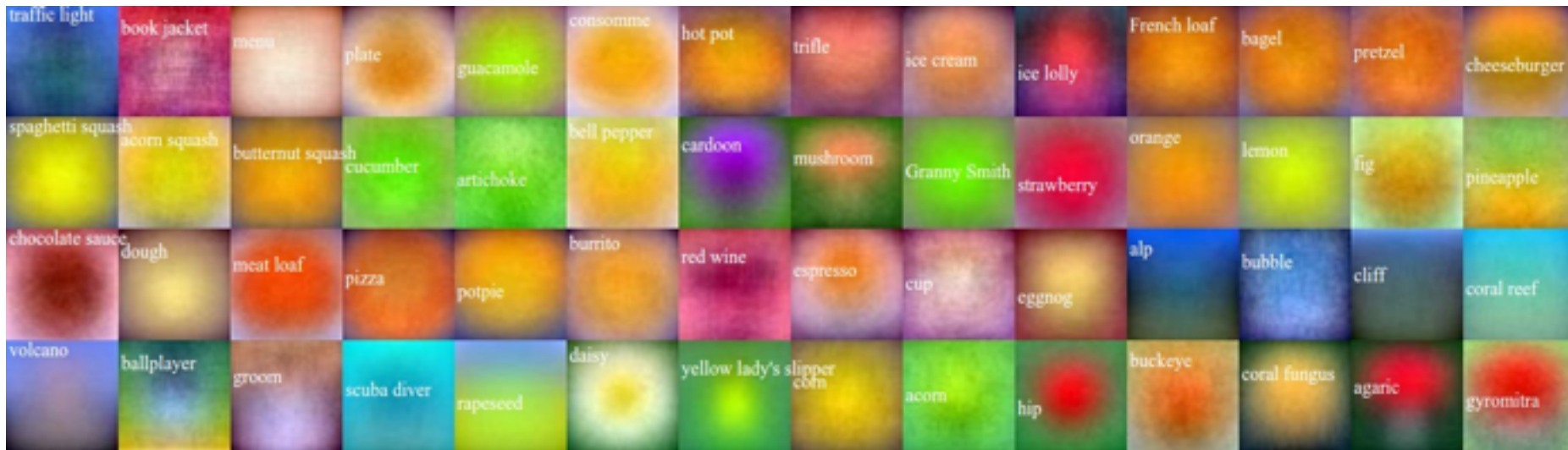
*(It's significantly easier with more numbers, need smaller nudge for each)*

# Andrej Karpathy Blog post: Breaking Linear Classifiers on ImageNet

Recall CIFAR-10 linear classifiers:

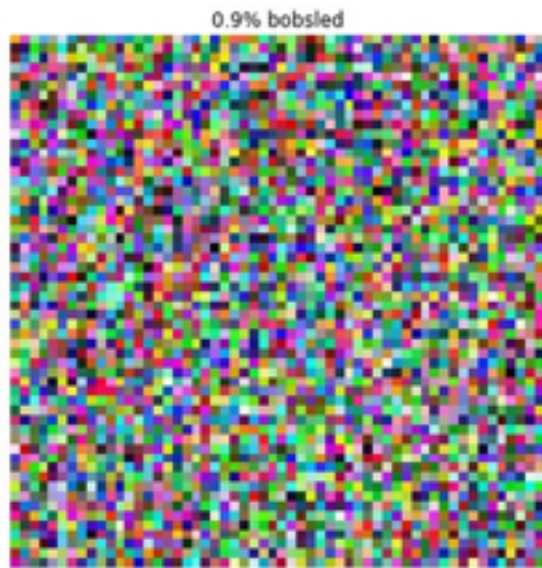


ImageNet classifiers:

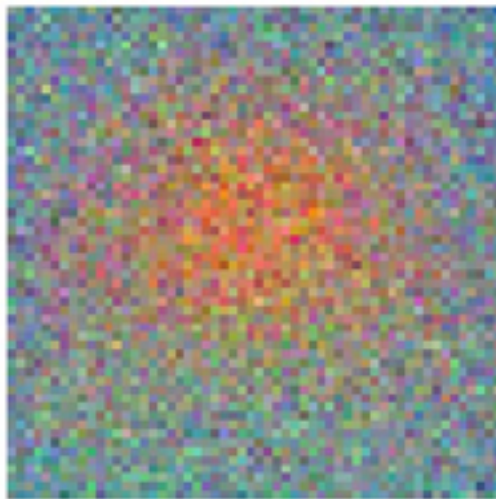




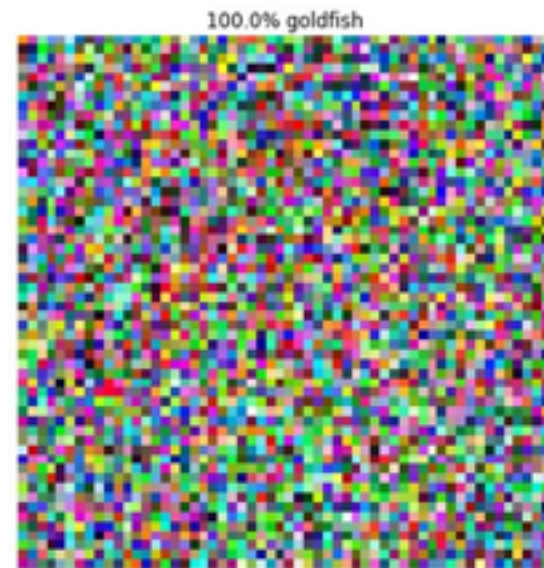
*mix in a tiny bit of  
Goldfish classifier weights*



+



=

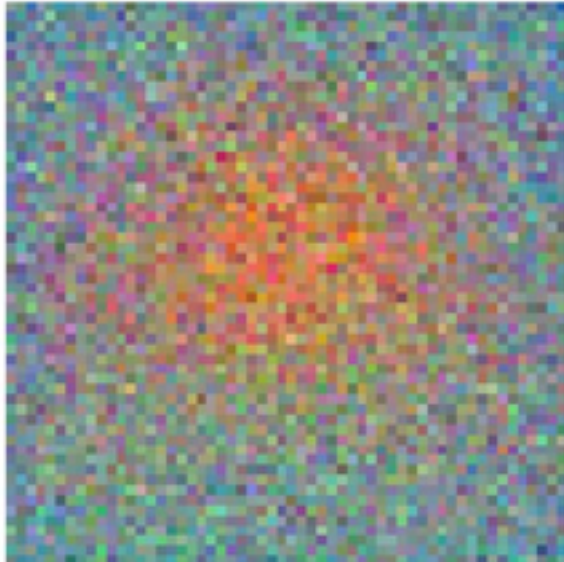


**100% Goldfish**

1.0% kit fox

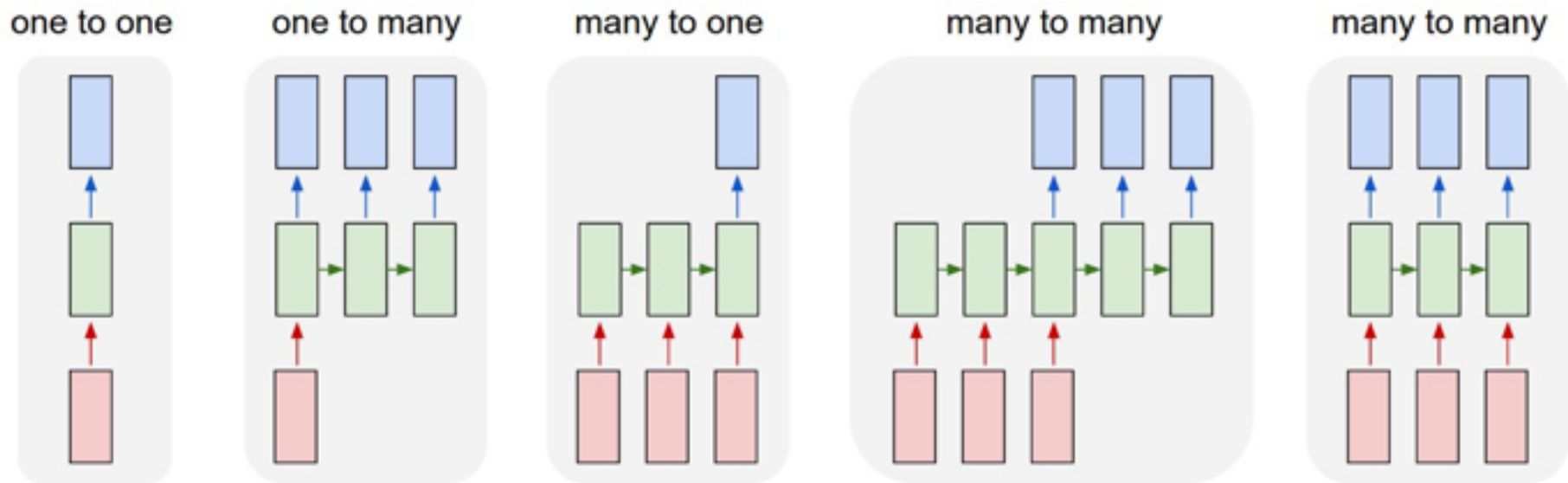


8.0% goldfish



# Recurrent Neural Networks

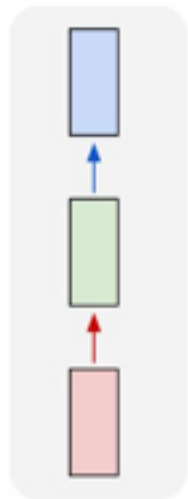
# Recurrent Networks offer a lot of flexibility:



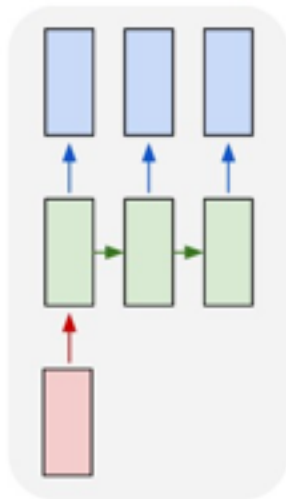
↖ **Vanilla Neural Networks**

# Recurrent Networks offer a lot of flexibility:

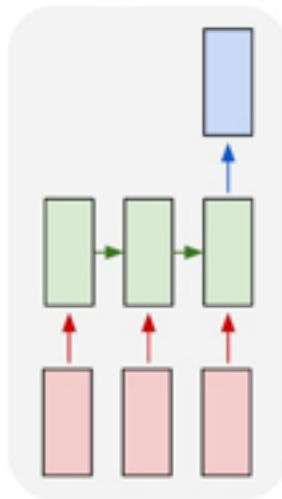
one to one



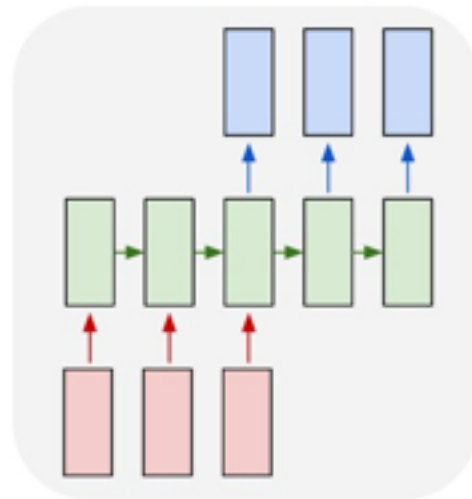
one to many



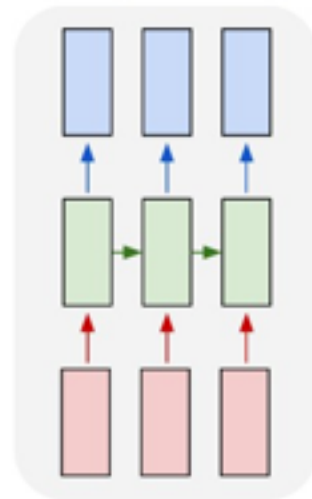
many to one



many to many



many to many

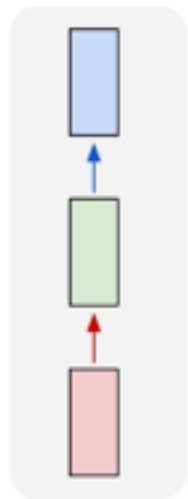


**e.g. Image Captioning**  
image -> sequence of words

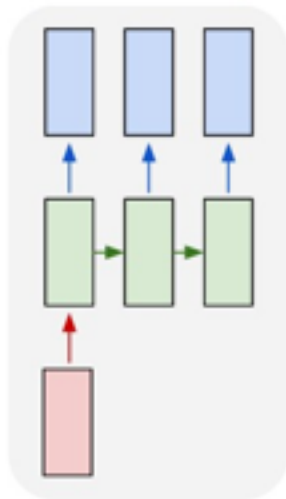


# Recurrent Networks offer a lot of flexibility:

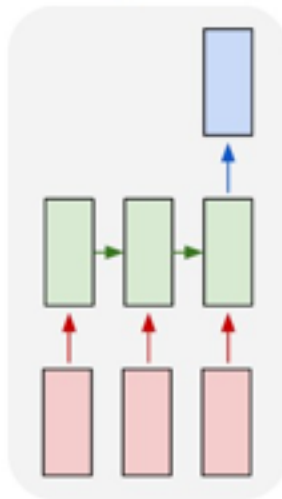
one to one



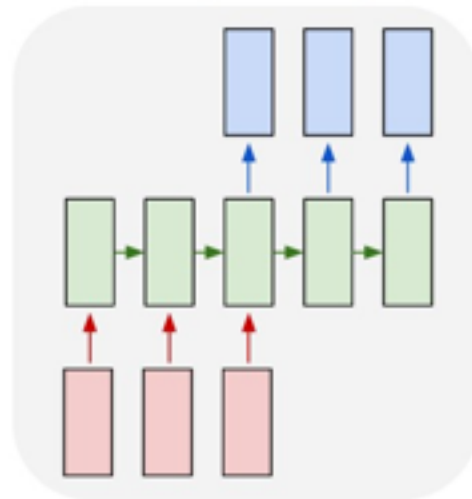
one to many



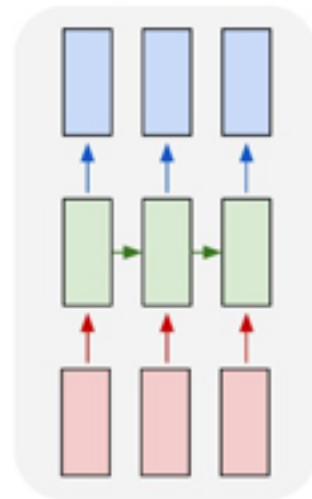
many to one



many to many



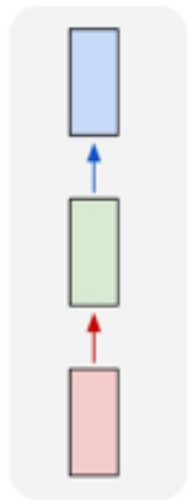
many to many



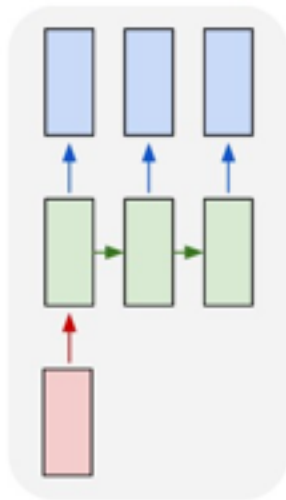
e.g. **Sentiment Classification**  
sequence of words -> sentiment

# Recurrent Networks offer a lot of flexibility:

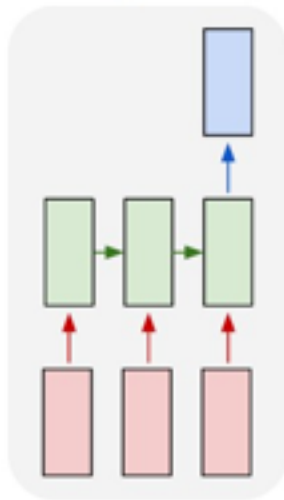
one to one



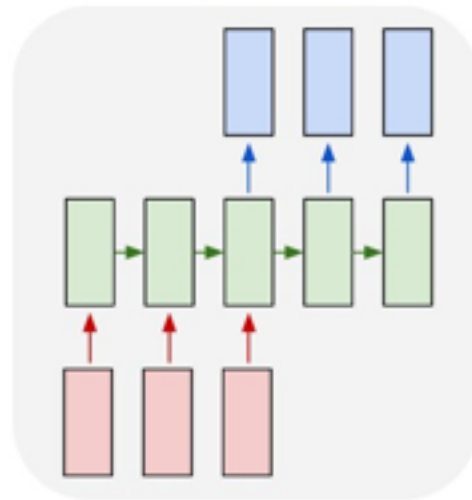
one to many



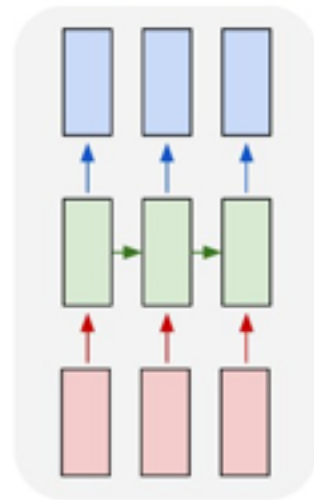
many to one



many to many



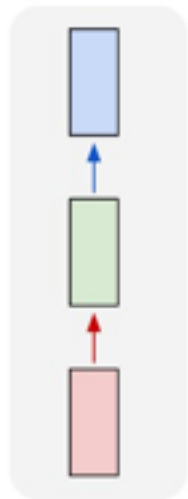
many to many



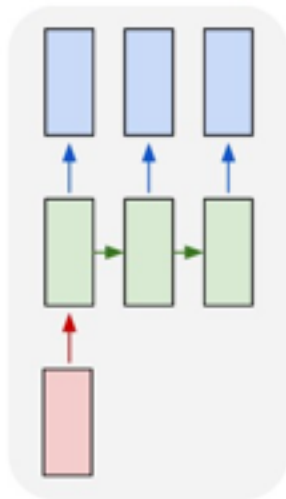
e.g. **Machine Translation**  
seq of words  $\rightarrow$  seq of words

# Recurrent Networks offer a lot of flexibility:

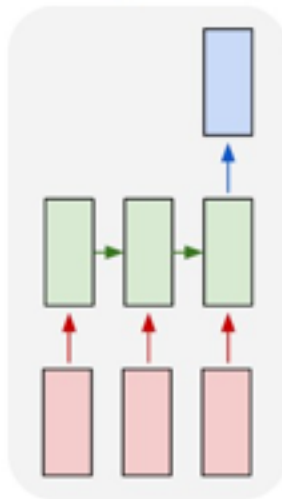
one to one



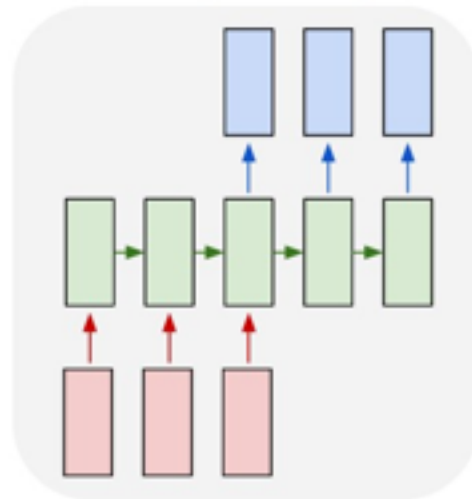
one to many



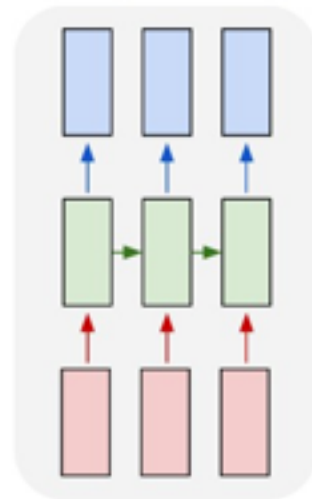
many to one



many to many



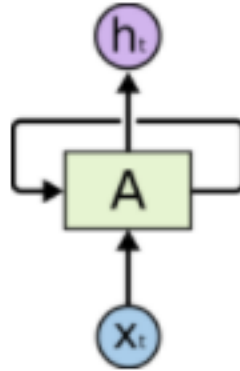
many to many



e.g. Video classification on frame level

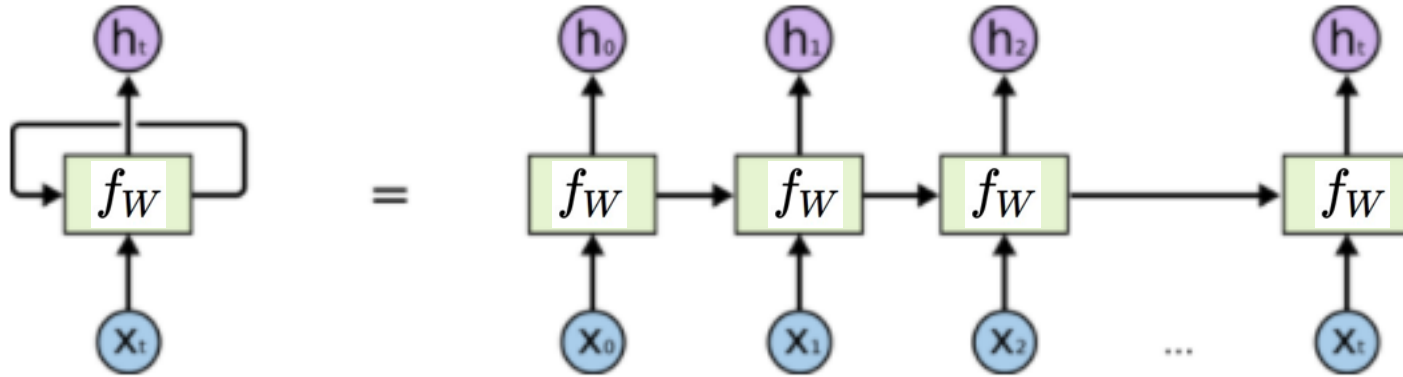


# Recurrent Networks



**Recurrent Neural Networks have loops.**

# RNN - at each time step



An unrolled recurrent neural network.

$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

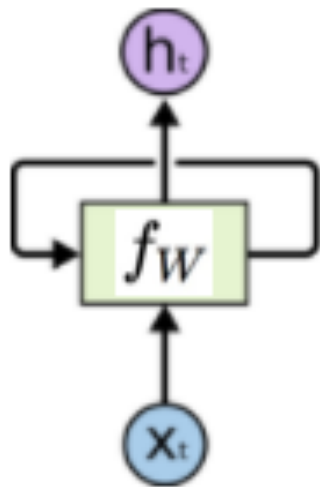
some function  
with parameters  $W$

Notice: the same function and the same set of parameters are used at every time step.



# (Vanilla) Recurrent Neural Network

The state consists of a single “hidden” vector  $\mathbf{h}$ :



$$h_t = f_W(h_{t-1}, x_t)$$



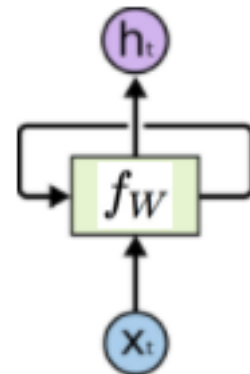
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

# Character-level language model example

Vocabulary:  
[h,e,l,o]

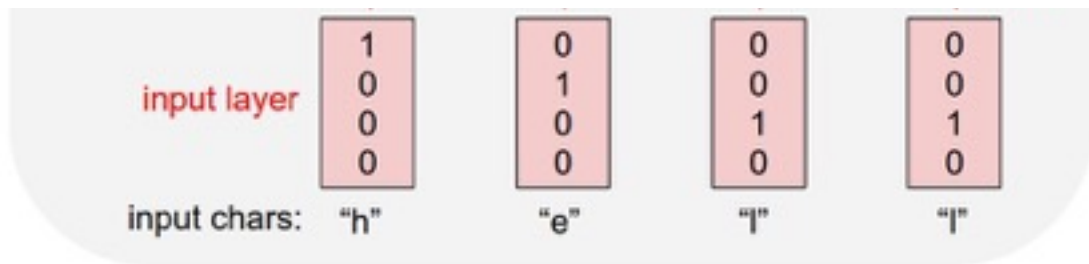
Example training  
sequence:  
**“hello”**



# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”

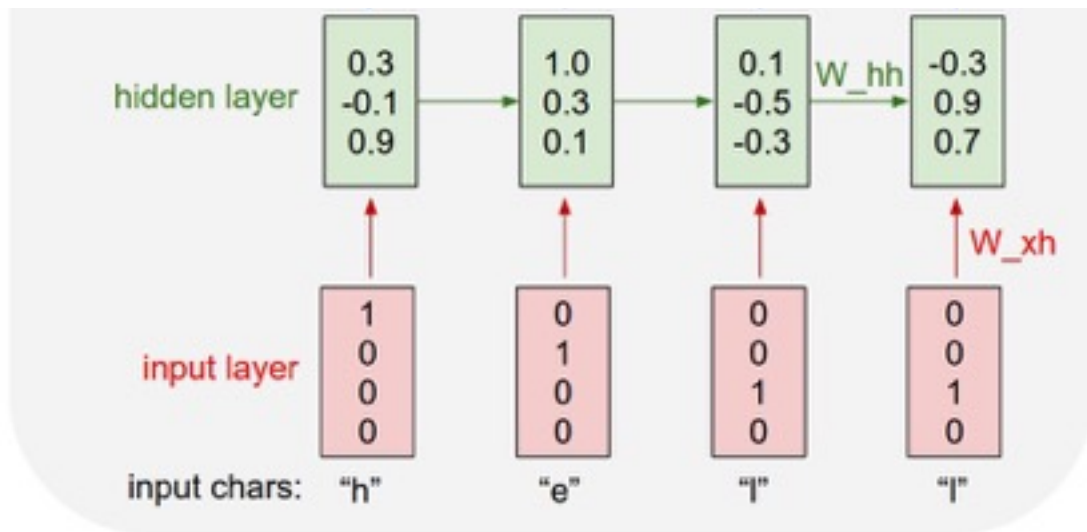


# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”

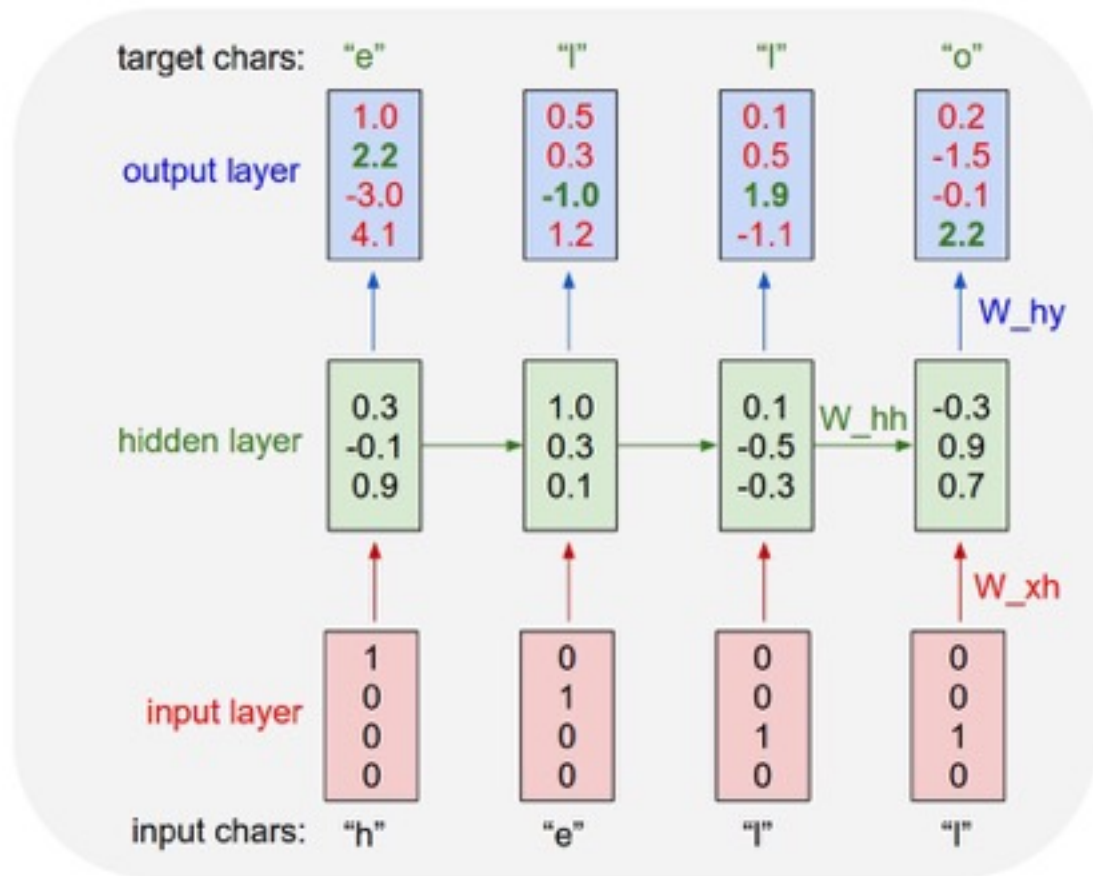
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



# Character-level language model example

Vocabulary:  
[h,e,l,o]

Example training  
sequence:  
“hello”



# min-char-rnn.py gist: 112 lines of Python

```
1 """
2 MiniChar RNN (min-char-rnn) vanilla rnn model, written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # Data I/O
8 data = open("input.txt", "r").read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print "data has %d characters, %d unique." % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { ix:i for i,ix in enumerate(chars) }
14
15 # Hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 28 # number of steps to unroll the rnn for
18 learning_rate = 0.1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bi = np.zeros(hidden_size, 1) # hidden bias
25 bo = np.zeros(vocab_size, 1) # output bias
26
27 def lossfun(inputs, targets, hprev):
28     """
29     Inputs, targets are both list of integers.
30     hprev is Numpy array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xi, yi, yf, oh = [], [], [], []
34     h0 = xi + np.empty(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xi[t] = np.zeros(vocab_size, 1) # encode in 1-of-N representation
39         xi[t][inputs[t]] = 1
40         h0[t] = np.tanh(np.dot(wih, xi[t]) + np.dot(whh, h0[t-1]) + bi) # hidden state
41         h0[t] = np.dot(why, h0[t]) # softmax weights for next steps
42         oh[t] = np.exp(h0[t]) / np.sum(np.exp(h0[t])) # probabilities for next chars
43         loss += np.log(oh[t][targets[t]]) # softmax (cross-entropy) loss
44     # backward pass: compute gradients going backwards
45     dwh, dwhh, dwhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46     dxi, dhy = np.zeros_like(h0), np.zeros_like(h0)
47     for t in xrange(len(inputs)-1):
48         dy = np.zeros(vocab_size)
49         dxi[targets[t]] += 1 # backprop into y
50         dhy += np.dot(dy, h0[t+1])
51         dy *= dy
52         dh = np.dot(why.T, dy) + dxi # backprop into h
53         dwhh = [1 - h0[t]**2] * dh # backprop through each nonlinearity
54         dwh += np.dot(dwhh, xi[t+1])
55         dwhh += np.dot(dwhh, h0[t+1])
56         dxi += np.dot(dwhh.T, dwhh)
57         dxi += np.dot(dwhh.T, dwhh)
58     for dparam in [dwh, dwhh, dwhy, dxi, dhy]:
59         np.clip(dparam, -1, 1, out=dparam) # clip to mitigate exploding gradients
60     return loss, dwh, dwhh, dwhy, dxi, dhy, h0[len(inputs)-1]
```

```
61 def sample(h, seed_ix, n):
62     """
63     sample a sequence of integers from the model
64     h is memory state, seed_ix is seed letter for first time step
65     """
66     x = np.zeros(vocab_size, 1)
67     h[seed_ix] = 1
68     ixes = []
69     for i in xrange(n):
70         h = np.tanh(np.dot(wih, x) + np.dot(whh, h) + bi)
71         y = np.dot(why, h) + bo
72         p = np.exp(y) / np.sum(np.exp(y))
73         ix = np.random.choice(vocab_size, p=p, replace=True)
74         x[ix] = 1
75         ixes.append(ix)
76     return ixes
77
78 N, D = 0, 0
79 dwh, dwhh, dwhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
80 dxi, dhy = np.zeros_like(h0), np.zeros_like(h0) # memory variables for adapted
81 smooth_loss = -np.log(0.5*vocab_size)/seq_length # loss at iteration 0
82 while True:
83     # prepare inputs (we're sweeping from left to right in steps seq_length long)
84     if seq_length+1 > len(data) or n % D:
85         hprev = np.zeros_like(h0) # reset rnn memory
86         p = 0 # go from start of data
87         inputs = [char_to_ix[ch] for ch in data[p:seq_length]]
88         targets = [char_to_ix[ch] for ch in data[p+seq_length:]]
89
90     # sample from the model now and then
91     if n % 100 == 0:
92         sample_ix = sample(hprev, inputs[0], 100)
93         ix = ''.join(ix_to_char[ix] for ix in sample_ix)
94         print "---->%s\n" % ix,
95
96     # forward seq_length characters through the net and fetch gradient
97     loss, dwh, dwhh, dwhy, dxi, dhy, hprev = lossfun(inputs, targets, hprev)
98     smooth_loss = smooth_loss + 0.999 * loss + 0.001 * loss
99     if n % 100 == 0: print "iter %d, loss: %f" % (n, smooth_loss) # print progress
100
101     # perform parameter update with adapted
102     for dparam, dparam_adapt in zip([dwh, dwhh, dwhy, dxi, dhy],
103                                     [dwh, dwhh, dwhy, dxi, dhy]):
104         dparam += dparam_adapt
105     dparam = -learning_rate * dparam / np.sqrt(10000 + 1e-5) # adapted update
106
107     p = seq_length + next data pointer
108     n += 1 # iteration counter
```

(<https://gist.github.com/karpathy/d4dee566867f8291f086>)



[n-char-rnn.py gist](#)

[illegible]

## Data I/O

[illegible]

```
1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
```

n Andrej Karpathy  
1n

comp150dl



```

1 #minid character-level vanilla rnn model, written by Andrej Karpathy (@karpathy)
2
3 import numpy as np
4
5 # data
6 data = open('input.txt', 'r').read() # should be simple plain text file
7 chars = list(data)
8 vocab_size, vocab_size = len(chars), len(chars)
9 print('data has %d characters, %d unique.' % (len(data), vocab_size))
10 char_to_ix = {c:i for i,c in enumerate(chars)}
11 ix_to_char = {i:c for i,c in enumerate(chars)}
12
13 # hyperparameters
14 hidden_size = 100 # size of hidden layer of neurons
15 seq_length = 25 # number of steps to unroll the rnn for
16 learning_rate = 0.1
17
18 # model parameters
19 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
20 Wwh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
21 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
22 bx = np.zeros((hidden_size, 1)) # hidden bias
23 by = np.zeros((vocab_size, 1)) # output bias
24
25 def forward(inputs, targets, hprev):
26     loss = 0
27     inputs, targets are both list of integers
28     hprev is the array of initial hidden state
29     requires the loss, gradients to model parameters, and last hidden state
30     #
31     #
32     #
33     #
34     #
35     #
36     #
37     #
38     #
39     #
40     #
41     #
42     #
43     #
44     #
45     #
46     #
47     #
48     #
49     #
50     #
51     #
52     #
53     #
54     #
55     #
56     #
57     #
58     #
59     #
60     #
61     #
62     #
63     #
64     #
65     #
66     #
67     #
68     #
69     #
70     #
71     #
72     #
73     #
74     #
75     #
76     #
77     #
78     #
79     #
80     #
81     #
82     #
83     #
84     #
85     #
86     #
87     #
88     #
89     #
90     #
91     #
92     #
93     #
94     #
95     #
96     #
97     #
98     #
99     #
100    #
101    #
102    #
103    #
104    #
105    #
106    #
107    #
108    #
109    #
110    #
111    #
112    #
113    #
114    #
115    #
116    #
117    #
118    #
119    #
120    #
121    #
122    #
123    #
124    #
125    #
126    #
127    #
128    #
129    #
130    #
131    #
132    #
133    #
134    #
135    #
136    #
137    #
138    #
139    #
140    #
141    #
142    #
143    #
144    #
145    #
146    #
147    #
148    #
149    #
150    #
151    #
152    #
153    #
154    #
155    #
156    #
157    #
158    #
159    #
160    #
161    #
162    #
163    #
164    #
165    #
166    #
167    #
168    #
169    #
170    #
171    #
172    #
173    #
174    #
175    #
176    #
177    #
178    #
179    #
180    #
181    #
182    #
183    #
184    #
185    #
186    #
187    #
188    #
189    #
190    #
191    #
192    #
193    #
194    #
195    #
196    #
197    #
198    #
199    #
200    #
201    #
202    #
203    #
204    #
205    #
206    #
207    #
208    #
209    #
210    #
211    #
212    #
213    #
214    #
215    #
216    #
217    #
218    #
219    #
220    #
221    #
222    #
223    #
224    #
225    #
226    #
227    #
228    #
229    #
230    #
231    #
232    #
233    #
234    #
235    #
236    #
237    #
238    #
239    #
240    #
241    #
242    #
243    #
244    #
245    #
246    #
247    #
248    #
249    #
250    #
251    #
252    #
253    #
254    #
255    #
256    #
257    #
258    #
259    #
260    #
261    #
262    #
263    #
264    #
265    #
266    #
267    #
268    #
269    #
270    #
271    #
272    #
273    #
274    #
275    #
276    #
277    #
278    #
279    #
280    #
281    #
282    #
283    #
284    #
285    #
286    #
287    #
288    #
289    #
290    #
291    #
292    #
293    #
294    #
295    #
296    #
297    #
298    #
299    #
300    #
301    #
302    #
303    #
304    #
305    #
306    #
307    #
308    #
309    #
310    #
311    #
312    #
313    #
314    #
315    #
316    #
317    #
318    #
319    #
320    #
321    #
322    #
323    #
324    #
325    #
326    #
327    #
328    #
329    #
330    #
331    #
332    #
333    #
334    #
335    #
336    #
337    #
338    #
339    #
340    #
341    #
342    #
343    #
344    #
345    #
346    #
347    #
348    #
349    #
350    #
351    #
352    #
353    #
354    #
355    #
356    #
357    #
358    #
359    #
360    #
361    #
362    #
363    #
364    #
365    #
366    #
367    #
368    #
369    #
370    #
371    #
372    #
373    #
374    #
375    #
376    #
377    #
378    #
379    #
380    #
381    #
382    #
383    #
384    #
385    #
386    #
387    #
388    #
389    #
390    #
391    #
392    #
393    #
394    #
395    #
396    #
397    #
398    #
399    #
400    #
401    #
402    #
403    #
404    #
405    #
406    #
407    #
408    #
409    #
410    #
411    #
412    #
413    #
414    #
415    #
416    #
417    #
418    #
419    #
420    #
421    #
422    #
423    #
424    #
425    #
426    #
427    #
428    #
429    #
430    #
431    #
432    #
433    #
434    #
435    #
436    #
437    #
438    #
439    #
440    #
441    #
442    #
443    #
444    #
445    #
446    #
447    #
448    #
449    #
450    #
451    #
452    #
453    #
454    #
455    #
456    #
457    #
458    #
459    #
460    #
461    #
462    #
463    #
464    #
465    #
466    #
467    #
468    #
469    #
470    #
471    #
472    #
473    #
474    #
475    #
476    #
477    #
478    #
479    #
480    #
481    #
482    #
483    #
484    #
485    #
486    #
487    #
488    #
489    #
490    #
491    #
492    #
493    #
494    #
495    #
496    #
497    #
498    #
499    #
500    #
501    #
502    #
503    #
504    #
505    #
506    #
507    #
508    #
509    #
510    #
511    #
512    #
513    #
514    #
515    #
516    #
517    #
518    #
519    #
520    #
521    #
522    #
523    #
524    #
525    #
526    #
527    #
528    #
529    #
530    #
531    #
532    #
533    #
534    #
535    #
536    #
537    #
538    #
539    #
540    #
541    #
542    #
543    #
544    #
545    #
546    #
547    #
548    #
549    #
550    #
551    #
552    #
553    #
554    #
555    #
556    #
557    #
558    #
559    #
560    #
561    #
562    #
563    #
564    #
565    #
566    #
567    #
568    #
569    #
570    #
571    #
572    #
573    #
574    #
575    #
576    #
577    #
578    #
579    #
580    #
581    #
582    #
583    #
584    #
585    #
586    #
587    #
588    #
589    #
590    #
591    #
592    #
593    #
594    #
595    #
596    #
597    #
598    #
599    #
600    #
601    #
602    #
603    #
604    #
605    #
606    #
607    #
608    #
609    #
610    #
611    #
612    #
613    #
614    #
615    #
616    #
617    #
618    #
619    #
620    #
621    #
622    #
623    #
624    #
625    #
626    #
627    #
628    #
629    #
630    #
631    #
632    #
633    #
634    #
635    #
636    #
637    #
638    #
639    #
640    #
641    #
642    #
643    #
644    #
645    #
646    #
647    #
648    #
649    #
650    #
651    #
652    #
653    #
654    #
655    #
656    #
657    #
658    #
659    #
660    #
661    #
662    #
663    #
664    #
665    #
666    #
667    #
668    #
669    #
670    #
671    #
672    #
673    #
674    #
675    #
676    #
677    #
678    #
679    #
680    #
681    #
682    #
683    #
684    #
685    #
686    #
687    #
688    #
689    #
690    #
691    #
692    #
693    #
694    #
695    #
696    #
697    #
698    #
699    #
700    #
701    #
702    #
703    #
704    #
705    #
706    #
707    #
708    #
709    #
710    #
711    #
712    #
713    #
714    #
715    #
716    #
717    #
718    #
719    #
720    #
721    #
722    #
723    #
724    #
725    #
726    #
727    #
728    #
729    #
730    #
731    #
732    #
733    #
734    #
735    #
736    #
737    #
738    #
739    #
740    #
741    #
742    #
743    #
744    #
745    #
746    #
747    #
748    #
749    #
750    #
751    #
752    #
753    #
754    #
755    #
756    #
757    #
758    #
759    #
760    #
761    #
762    #
763    #
764    #
765    #
766    #
767    #
768    #
769    #
770    #
771    #
772    #
773    #
774    #
775    #
776    #
777    #
778    #
779    #
780    #
781    #
782    #
783    #
784    #
785    #
786    #
787    #
788    #
789    #
790    #
791    #
792    #
793    #
794    #
795    #
796    #
797    #
798    #
799    #
800    #
801    #
802    #
803    #
804    #
805    #
806    #
807    #
808    #
809    #
810    #
811    #
812    #
813    #
814    #
815    #
816    #
817    #
818    #
819    #
820    #
821    #
822    #
823    #
824    #
825    #
826    #
827    #
828    #
829    #
830    #
831    #
832    #
833    #
834    #
835    #
836    #
837    #
838    #
839    #
840    #
841    #
842    #
843    #
844    #
845    #
846    #
847    #
848    #
849    #
850    #
851    #
852    #
853    #
854    #
855    #
856    #
857    #
858    #
859    #
860    #
861    #
862    #
863    #
864    #
865    #
866    #
867    #
868    #
869    #
870    #
871    #
872    #
873    #
874    #
875    #
876    #
877    #
878    #
879    #
880    #
881    #
882    #
883    #
884    #
885    #
886    #
887    #
888    #
889    #
890    #
891    #
892    #
893    #
894    #
895    #
896    #
897    #
898    #
899    #
900    #
901    #
902    #
903    #
904    #
905    #
906    #
907    #
908    #
909    #
910    #
911    #
912    #
913    #
914    #
915    #
916    #
917    #
918    #
919    #
920    #
921    #
922    #
923    #
924    #
925    #
926    #
927    #
928    #
929    #
930    #
931    #
932    #
933    #
934    #
935    #
936    #
937    #
938    #
939    #
940    #
941    #
942    #
943    #
944    #
945    #
946    #
947    #
948    #
949    #
950    #
951    #
952    #
953    #
954    #
955    #
956    #
957    #
958    #
959    #
960    #
961    #
962    #
963    #
964    #
965    #
966    #
967    #
968    #
969    #
970    #
971    #
972    #
973    #
974    #
975    #
976    #
977    #
978    #
979    #
980    #
981    #
982    #
983    #
984    #
985    #
986    #
987    #
988    #
989    #
990    #
991    #
992    #
993    #
994    #
995    #
996    #
997    #
998    #
999    #
1000   #

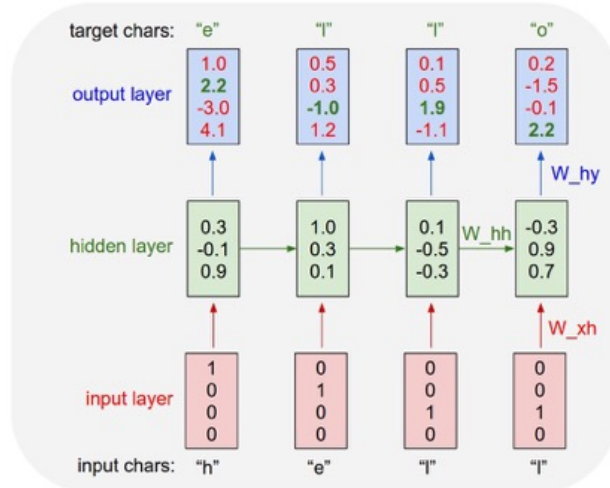
```

```

15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 Wwh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias

```

recall:



[n-char-rnn.py gist](#)

[illegible]

```

1  # @param {String} name - name of the model
2  # @param {Object} data - data to be saved
3  # @param {Object} options - options for the save method
4  # @param {Function} callback - callback function
5  # @return {Object} - the saved model
6  # @example
7  #   model.save({name: 'John', age: 30}, function(err, model) {
8  #     if (err) {
9  #       // handle error
10    }
11    // model is the saved model
12  })
13
14  // Save a new model
15  model.save({name: 'John', age: 30}, function(err, model) {
16    if (err) {
17      // handle error
18    }
19    // model is the saved model
20  })
21
22  // Save an existing model
23  model.findById('507f1f77bcf86cd791e40133').save(function(err, model) {
24    if (err) {
25      // handle error
26    }
27    // model is the saved model
28  })
29
30  // Save a new model with options
31  model.save({name: 'John', age: 30}, {validate: true}, function(err, model) {
32    if (err) {
33      // handle error
34    }
35    // model is the saved model
36  })
37
38  // Save a new model with a callback
39  model.save({name: 'John', age: 30}, function(err, model) {
40    if (err) {
41      // handle error
42    }
43    // model is the saved model
44  })
45
46  // Save a new model with a callback and options
47  model.save({name: 'John', age: 30}, {validate: true}, function(err, model) {
48    if (err) {
49      // handle error
50    }
51    // model is the saved model
52  })
53
54  // Save a new model with a callback and options
55  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
56    validate: true
57  }}, function(err, model) {
58    if (err) {
59      // handle error
60    }
61    // model is the saved model
62  })
63
64  // Save a new model with a callback and options
65  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
66    validate: true,
67    saveOptions: {
68      validate: true
69    }
70  }}, function(err, model) {
71    if (err) {
72      // handle error
73    }
74    // model is the saved model
75  })
76
77  // Save a new model with a callback and options
78  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
79    validate: true,
80    saveOptions: {
81      validate: true,
82      saveOptions: {
83        validate: true
84      }
85    }
86  }}, function(err, model) {
87    if (err) {
88      // handle error
89    }
90    // model is the saved model
91  })
92
93  // Save a new model with a callback and options
94  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
95    validate: true,
96    saveOptions: {
97      validate: true,
98      saveOptions: {
99        validate: true,
100       saveOptions: {
101         validate: true
102       }
103     }
104   }}, function(err, model) {
105     if (err) {
106       // handle error
107     }
108     // model is the saved model
109   })
110
111  // Save a new model with a callback and options
112  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
113    validate: true,
114    saveOptions: {
115      validate: true,
116      saveOptions: {
117        validate: true,
118        saveOptions: {
119          validate: true
120        }
121      }
122    }
123  }}, function(err, model) {
124    if (err) {
125      // handle error
126    }
127    // model is the saved model
128  })
129
130  // Save a new model with a callback and options
131  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
132    validate: true,
133    saveOptions: {
134      validate: true,
135      saveOptions: {
136        validate: true,
137        saveOptions: {
138          validate: true,
139          saveOptions: {
140            validate: true
141          }
142        }
143      }
144    }
145  }}, function(err, model) {
146    if (err) {
147      // handle error
148    }
149    // model is the saved model
150  })
151
152  // Save a new model with a callback and options
153  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
154    validate: true,
155    saveOptions: {
156      validate: true,
157      saveOptions: {
158        validate: true,
159        saveOptions: {
160          validate: true,
161          saveOptions: {
162            validate: true,
163            saveOptions: {
164              validate: true
165            }
166          }
167        }
168      }
169    }
170  }}, function(err, model) {
171    if (err) {
172      // handle error
173    }
174    // model is the saved model
175  })
176
177  // Save a new model with a callback and options
178  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
179    validate: true,
180    saveOptions: {
181      validate: true,
182      saveOptions: {
183        validate: true,
184        saveOptions: {
185          validate: true,
186          saveOptions: {
187            validate: true,
188            saveOptions: {
189              validate: true,
190              saveOptions: {
191                validate: true
192              }
193            }
194          }
195        }
196      }
197    }
198  }}, function(err, model) {
199    if (err) {
200      // handle error
201    }
202    // model is the saved model
203  })
204
205  // Save a new model with a callback and options
206  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
207    validate: true,
208    saveOptions: {
209      validate: true,
210      saveOptions: {
211        validate: true,
212        saveOptions: {
213          validate: true,
214          saveOptions: {
215            validate: true,
216            saveOptions: {
217              validate: true,
218              saveOptions: {
219                validate: true,
220                saveOptions: {
221                  validate: true
222                }
223              }
224            }
225          }
226        }
227      }
228    }
229  }}, function(err, model) {
230    if (err) {
231      // handle error
232    }
233    // model is the saved model
234  })
235
236  // Save a new model with a callback and options
237  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
238    validate: true,
239    saveOptions: {
240      validate: true,
241      saveOptions: {
242        validate: true,
243        saveOptions: {
244          validate: true,
245          saveOptions: {
246            validate: true,
247            saveOptions: {
248              validate: true,
249              saveOptions: {
250                validate: true,
251                saveOptions: {
252                  validate: true,
253                  saveOptions: {
254                    validate: true
255                  }
256                }
257              }
258            }
259          }
260        }
261      }
262    }
263  }}, function(err, model) {
264    if (err) {
265      // handle error
266    }
267    // model is the saved model
268  })
269
270  // Save a new model with a callback and options
271  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
272    validate: true,
273    saveOptions: {
274      validate: true,
275      saveOptions: {
276        validate: true,
277        saveOptions: {
278          validate: true,
279          saveOptions: {
280            validate: true,
281            saveOptions: {
282              validate: true,
283              saveOptions: {
284                validate: true,
285                saveOptions: {
286                  validate: true,
287                  saveOptions: {
288                    validate: true,
289                    saveOptions: {
290                      validate: true
291                    }
292                  }
293                }
294              }
295            }
296          }
297        }
298      }
299    }
300  }}, function(err, model) {
301    if (err) {
302      // handle error
303    }
304    // model is the saved model
305  })
306
307  // Save a new model with a callback and options
308  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
309    validate: true,
310    saveOptions: {
311      validate: true,
312      saveOptions: {
313        validate: true,
314        saveOptions: {
315          validate: true,
316          saveOptions: {
317            validate: true,
318            saveOptions: {
319              validate: true,
320              saveOptions: {
321                validate: true,
322                saveOptions: {
323                  validate: true,
324                  saveOptions: {
325                    validate: true,
326                    saveOptions: {
327                      validate: true,
328                      saveOptions: {
329                        validate: true
330                      }
331                    }
332                  }
333                }
334              }
335            }
336          }
337        }
338      }
339    }
340  }}, function(err, model) {
341    if (err) {
342      // handle error
343    }
344    // model is the saved model
345  })
346
347  // Save a new model with a callback and options
348  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
349    validate: true,
350    saveOptions: {
351      validate: true,
352      saveOptions: {
353        validate: true,
354        saveOptions: {
355          validate: true,
356          saveOptions: {
357            validate: true,
358            saveOptions: {
359              validate: true,
360              saveOptions: {
361                validate: true,
362                saveOptions: {
363                  validate: true,
364                  saveOptions: {
365                    validate: true,
366                    saveOptions: {
367                      validate: true,
368                      saveOptions: {
369                        validate: true,
370                        saveOptions: {
371                          validate: true
372                        }
373                      }
374                    }
375                  }
376                }
377              }
378            }
379          }
380        }
381      }
382    }
383  }}, function(err, model) {
384    if (err) {
385      // handle error
386    }
387    // model is the saved model
388  })
389
390  // Save a new model with a callback and options
391  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
392    validate: true,
393    saveOptions: {
394      validate: true,
395      saveOptions: {
396        validate: true,
397        saveOptions: {
398          validate: true,
399          saveOptions: {
400            validate: true,
401            saveOptions: {
402              validate: true,
403              saveOptions: {
404                validate: true,
405                saveOptions: {
406                  validate: true,
407                  saveOptions: {
408                    validate: true,
409                    saveOptions: {
410                      validate: true,
411                      saveOptions: {
412                        validate: true,
413                        saveOptions: {
414                          validate: true,
415                          saveOptions: {
416                            validate: true
417                          }
418                        }
419                      }
420                    }
421                  }
422                }
423              }
424            }
425          }
426        }
427      }
428    }
429  }}, function(err, model) {
430    if (err) {
431      // handle error
432    }
433    // model is the saved model
434  })
435
436  // Save a new model with a callback and options
437  model.save({name: 'John', age: 30}, {validate: true, saveOptions: {
438    validate: true,
439    saveOptions: {
440      validate: true,
441      saveOptions: {
442        validate: true,
443        saveOptions: {
444          validate: true,
445          saveOptions: {
446            validate: true,
447            saveOptions: {
448              validate
```

n Andrej Karpathy  
1n

## Main loop

```

01 n, p = 0, 0
02 dwxh, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
03 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
04 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
05 while True:
06     # prepare inputs (we're sweeping from left to right in steps seq_length long)
07     if p+seq_length+1 >= len(data) or n == 0:
08         hprev = np.zeros((hidden_size,1)) # reset RNN memory
09         p = 0 # go from start of data
10     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
11     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
12
13     # sample from the model now and then
14     if n % 100 == 0:
15         sample_ix = sample(hprev, inputs[0], 200)
16         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
17         print '----\n %s \n----' % (txt, )
18
19     # forward seq_length characters through the net and fetch gradient
20     loss, dwxh, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
21     smooth_loss = smooth_loss * 0.999 + loss * 0.001
22     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
23
24     # perform parameter update with Adagrad
25     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
26                                   [dwxh, dwhh, dwhy, dbh, dby],
27                                   [mwxh, mwhh, mwhy, mbh, mby]):
28         mem += dparam * dparam
29         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
30
31     p += seq_length # move data pointer
32     n += 1 # iteration counter

```

## [n-char-rnn.py gist](#)

[illegible][illegible]

n Andrej Karpathy

## Main loop

```

81 n, p = 0, 0
82 dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n %s \n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                   [dWxh, dWhh, dWhy, dbh, dby],
107                                   [mWxh, mWhh, mWhy, mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter

```

## [n-char-rnn.py gist](#)

[illegible][illegible]

n Andrej Karpathy

## Main loop

```

81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93         # sample from the model now and then
94         if n % 100 == 0:
95             sample_ix = sample(hprev, inputs[0], 200)
96             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97             print '----\n %s \n----' % (txt, )
98
99         # forward seq_length characters through the net and fetch gradient
100         loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101         smooth_loss = smooth_loss * 0.999 + loss * 0.001
102         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104         # perform parameter update with Adagrad
105         for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                     [dWxh, dWhh, dWhy, dbh, dby],
107                                     [mWxh, mWhh, mWhy, mbh, mby]):
108             mem += dparam * dparam
109             param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111         p += seq_length # move data pointer
112         n += 1 # iteration counter

```



[n-char-rnn.py gist](#)

[illegible]

## Main loop

```

61 n, p = 0, 0
62 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
63 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
64 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
65 while True:
66     # prepare inputs (we're sweeping from left to right in steps seq_length long)
67     if p+seq_length+1 >= len(data) or n == 0:
68         hprev = np.zeros((hidden_size,1)) # reset RNN memory
69         p = 0 # go from start of data
70     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
71     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
72
73     # sample from the model now and then
74     if n % 100 == 0:
75         sample_ix = sample(hprev, inputs[0], 200)
76         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
77         print '----\n %s \n----' % (txt, )
78
79     # forward seq_length characters through the net and fetch gradient
80     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
81     smooth_loss = smooth_loss * 0.999 + loss * 0.001
82     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
83
84     # perform parameter update with Adagrad
85     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
86                                   [dWxh, dWhh, dWhy, dbh, dby],
87                                   [mWxh, mWhh, mWhy, mbh, mby]):
88         mem += dparam * dparam
89         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
90
91     p += seq_length # move data pointer
92     n += 1 # iteration counter

```



```

1 #minid character-level vanilla rnn model, written by Andrej Karpathy (@karpathy)
2
3 import numpy as np
4
5 # data size
6 data = open('input.txt', 'r').read() # should be simple plain text file
7 chars = list(data)
8 data_size, vocab_size = len(data), len(chars)
9 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
10 char_to_ix = {c:i for i,c in enumerate(chars)}
11 ix_to_char = {i:c for i,c in enumerate(chars)}
12
13 # hyperparameters
14 hidden_size = 100 # size of hidden layer of neurons
15 seq_length = 20 # number of steps to unroll the rnn for
16 learning_rate = 0.1
17
18 # model parameters
19 w = np.random.randn(hidden_size, hidden_size)*0.01 # input to hidden
20 wh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
21 wb = np.random.randn(hidden_size, vocab_size)*0.01 # hidden to output
22 b = np.zeros((hidden_size, 1)) # hidden bias
23 by = np.zeros((vocab_size, 1)) # output bias
24
25 def forward(inputs, targets, hprev):
26     loss = 0
27     inputs, targets are both list of integers
28     hprev is the array of initial hidden state
29     requires the loss, gradients on model parameters, and last hidden state
30
31     # unrolled inputs, targets, and last hidden state
32     n, m, yk, yk_1, yk_2, yk_3, yk_4, yk_5, yk_6, yk_7, yk_8, yk_9, yk_10, yk_11, yk_12, yk_13, yk_14, yk_15, yk_16, yk_17, yk_18, yk_19, yk_20, yk_21, yk_22, yk_23, yk_24, yk_25, yk_26, yk_27, yk_28, yk_29, yk_30, yk_31, yk_32, yk_33, yk_34, yk_35, yk_36, yk_37, yk_38, yk_39, yk_40, yk_41, yk_42, yk_43, yk_44, yk_45, yk_46, yk_47, yk_48, yk_49, yk_50, yk_51, yk_52, yk_53, yk_54, yk_55, yk_56, yk_57, yk_58, yk_59, yk_60, yk_61, yk_62, yk_63, yk_64, yk_65, yk_66, yk_67, yk_68, yk_69, yk_70, yk_71, yk_72, yk_73, yk_74, yk_75, yk_76, yk_77, yk_78, yk_79, yk_80, yk_81, yk_82, yk_83, yk_84, yk_85, yk_86, yk_87, yk_88, yk_89, yk_90, yk_91, yk_92, yk_93, yk_94, yk_95, yk_96, yk_97, yk_98, yk_99, yk_100, yk_101, yk_102, yk_103, yk_104, yk_105, yk_106, yk_107, yk_108, yk_109, yk_110, yk_111, yk_112, yk_113, yk_114, yk_115, yk_116, yk_117, yk_118, yk_119, yk_120, yk_121, yk_122, yk_123, yk_124, yk_125, yk_126, yk_127, yk_128, yk_129, yk_130, yk_131, yk_132, yk_133, yk_134, yk_135, yk_136, yk_137, yk_138, yk_139, yk_140, yk_141, yk_142, yk_143, yk_144, yk_145, yk_146, yk_147, yk_148, yk_149, yk_150, yk_151, yk_152, yk_153, yk_154, yk_155, yk_156, yk_157, yk_158, yk_159, yk_160, yk_161, yk_162, yk_163, yk_164, yk_165, yk_166, yk_167, yk_168, yk_169, yk_170, yk_171, yk_172, yk_173, yk_174, yk_175, yk_176, yk_177, yk_178, yk_179, yk_180, yk_181, yk_182, yk_183, yk_184, yk_185, yk_186, yk_187, yk_188, yk_189, yk_190, yk_191, yk_192, yk_193, yk_194, yk_195, yk_196, yk_197, yk_198, yk_199, yk_200, yk_201, yk_202, yk_203, yk_204, yk_205, yk_206, yk_207, yk_208, yk_209, yk_210, yk_211, yk_212, yk_213, yk_214, yk_215, yk_216, yk_217, yk_218, yk_219, yk_220, yk_221, yk_222, yk_223, yk_224, yk_225, yk_226, yk_227, yk_228, yk_229, yk_230, yk_231, yk_232, yk_233, yk_234, yk_235, yk_236, yk_237, yk_238, yk_239, yk_240, yk_241, yk_242, yk_243, yk_244, yk_245, yk_246, yk_247, yk_248, yk_249, yk_250, yk_251, yk_252, yk_253, yk_254, yk_255, yk_256, yk_257, yk_258, yk_259, yk_260, yk_261, yk_262, yk_263, yk_264, yk_265, yk_266, yk_267, yk_268, yk_269, yk_270, yk_271, yk_272, yk_273, yk_274, yk_275, yk_276, yk_277, yk_278, yk_279, yk_280, yk_281, yk_282, yk_283, yk_284, yk_285, yk_286, yk_287, yk_288, yk_289, yk_290, yk_291, yk_292, yk_293, yk_294, yk_295, yk_296, yk_297, yk_298, yk_299, yk_300, yk_301, yk_302, yk_303, yk_304, yk_305, yk_306, yk_307, yk_308, yk_309, yk_310, yk_311, yk_312, yk_313, yk_314, yk_315, yk_316, yk_317, yk_318, yk_319, yk_320, yk_321, yk_322, yk_323, yk_324, yk_325, yk_326, yk_327, yk_328, yk_329, yk_330, yk_331, yk_332, yk_333, yk_334, yk_335, yk_336, yk_337, yk_338, yk_339, yk_340, yk_341, yk_342, yk_343, yk_344, yk_345, yk_346, yk_347, yk_348, yk_349, yk_350, yk_351, yk_352, yk_353, yk_354, yk_355, yk_356, yk_357, yk_358, yk_359, yk_360, yk_361, yk_362, yk_363, yk_364, yk_365, yk_366, yk_367, yk_368, yk_369, yk_370, yk_371, yk_372, yk_373, yk_374, yk_375, yk_376, yk_377, yk_378, yk_379, yk_380, yk_381, yk_382, yk_383, yk_384, yk_385, yk_386, yk_387, yk_388, yk_389, yk_390, yk_391, yk_392, yk_393, yk_394, yk_395, yk_396, yk_397, yk_398, yk_399, yk_400, yk_401, yk_402, yk_403, yk_404, yk_405, yk_406, yk_407, yk_408, yk_409, yk_410, yk_411, yk_412, yk_413, yk_414, yk_415, yk_416, yk_417, yk_418, yk_419, yk_420, yk_421, yk_422, yk_423, yk_424, yk_425, yk_426, yk_427, yk_428, yk_429, yk_430, yk_431, yk_432, yk_433, yk_434, yk_435, yk_436, yk_437, yk_438, yk_439, yk_440, yk_441, yk_442, yk_443, yk_444, yk_445, yk_446, yk_447, yk_448, yk_449, yk_450, yk_451, yk_452, yk_453, yk_454, yk_455, yk_456, yk_457, yk_458, yk_459, yk_460, yk_461, yk_462, yk_463, yk_464, yk_465, yk_466, yk_467, yk_468, yk_469, yk_470, yk_471, yk_472, yk_473, yk_474, yk_475, yk_476, yk_477, yk_478, yk_479, yk_480, yk_481, yk_482, yk_483, yk_484, yk_485, yk_486, yk_487, yk_488, yk_489, yk_490, yk_491, yk_492, yk_493, yk_494, yk_495, yk_496, yk_497, yk_498, yk_499, yk_500, yk_501, yk_502, yk_503, yk_504, yk_505, yk_506, yk_507, yk_508, yk_509, yk_510, yk_511, yk_512, yk_513, yk_514, yk_515, yk_516, yk_517, yk_518, yk_519, yk_520, yk_521, yk_522, yk_523, yk_524, yk_525, yk_526, yk_527, yk_528, yk_529, yk_530, yk_531, yk_532, yk_533, yk_534, yk_535, yk_536, yk_537, yk_538, yk_539, yk_540, yk_541, yk_542, yk_543, yk_544, yk_545, yk_546, yk_547, yk_548, yk_549, yk_550, yk_551, yk_552, yk_553, yk_554, yk_555, yk_556, yk_557, yk_558, yk_559, yk_560, yk_561, yk_562, yk_563, yk_564, yk_565, yk_566, yk_567, yk_568, yk_569, yk_570, yk_571, yk_572, yk_573, yk_574, yk_575, yk_576, yk_577, yk_578, yk_579, yk_580, yk_581, yk_582, yk_583, yk_584, yk_585, yk_586, yk_587, yk_588, yk_589, yk_590, yk_591, yk_592, yk_593, yk_594, yk_595, yk_596, yk_597, yk_598, yk_599, yk_600, yk_601, yk_602, yk_603, yk_604, yk_605, yk_606, yk_607, yk_608, yk_609, yk_610, yk_611, yk_612, yk_613, yk_614, yk_615, yk_616, yk_617, yk_618, yk_619, yk_620, yk_621, yk_622, yk_623, yk_624, yk_625, yk_626, yk_627, yk_628, yk_629, yk_630, yk_631, yk_632, yk_633, yk_634, yk_635, yk_636, yk_637, yk_638, yk_639, yk_640, yk_641, yk_642, yk_643, yk_644, yk_645, yk_646, yk_647, yk_648, yk_649, yk_650, yk_651, yk_652, yk_653, yk_654, yk_655, yk_656, yk_657, yk_658, yk_659, yk_660, yk_661, yk_662, yk_663, yk_664, yk_665, yk_666, yk_667, yk_668, yk_669, yk_670, yk_671, yk_672, yk_673, yk_674, yk_675, yk_676, yk_677, yk_678, yk_679, yk_680, yk_681, yk_682, yk_683, yk_684, yk_685, yk_686, yk_687, yk_688, yk_689, yk_690, yk_691, yk_692, yk_693, yk_694, yk_695, yk_696, yk_697, yk_698, yk_699, yk_700, yk_701, yk_702, yk_703, yk_704, yk_705, yk_706, yk_707, yk_708, yk_709, yk_710, yk_711, yk_712, yk_713, yk_714, yk_715, yk_716, yk_717, yk_718, yk_719, yk_720, yk_721, yk_722, yk_723, yk_724, yk_725, yk_726, yk_727, yk_728, yk_729, yk_730, yk_731, yk_732, yk_733, yk_734, yk_735, yk_736, yk_737, yk_738, yk_739, yk_740, yk_741, yk_742, yk_743, yk_744, yk_745, yk_746, yk_747, yk_748, yk_749, yk_750, yk_751, yk_752, yk_753, yk_754, yk_755, yk_756, yk_757, yk_758, yk_759, yk_760, yk_761, yk_762, yk_763, yk_764, yk_765, yk_766, yk_767, yk_768, yk_769, yk_770, yk_771, yk_772, yk_773, yk_774, yk_775, yk_776, yk_777, yk_778, yk_779, yk_780, yk_781, yk_782, yk_783, yk_784, yk_785, yk_786, yk_787, yk_788, yk_789, yk_790, yk_791, yk_792, yk_793, yk_794, yk_795, yk_796, yk_797, yk_798, yk_799, yk_800, yk_801, yk_802, yk_803, yk_804, yk_805, yk_806, yk_807, yk_808, yk_809, yk_810, yk_811, yk_812, yk_813, yk_814, yk_815, yk_816, yk_817, yk_818, yk_819, yk_820, yk_821, yk_822, yk_823, yk_824, yk_825, yk_826, yk_827, yk_828, yk_829, yk_830, yk_831, yk_832, yk_833, yk_834, yk_835, yk_836, yk_837, yk_838, yk_839, yk_840, yk_841, yk_842, yk_843, yk_844, yk_845, yk_846, yk_847, yk_848, yk_849, yk_850, yk_851, yk_852, yk_853, yk_854, yk_855, yk_856, yk_857, yk_858, yk_859, yk_860, yk_861, yk_862, yk_863, yk_864, yk_865, yk_866, yk_867, yk_868, yk_869, yk_870, yk_871, yk_872, yk_873, yk_874, yk_875, yk_876, yk_877, yk_878, yk_879, yk_880, yk_881, yk_882, yk_883, yk_884, yk_885, yk_886, yk_887, yk_888, yk_889, yk_890, yk_891, yk_892, yk_893, yk_894, yk_895, yk_896, yk_897, yk_898, yk_899, yk_900, yk_901, yk_902, yk_903, yk_904, yk_905, yk_906, yk_907, yk_908, yk_909, yk_910, yk_911, yk_912, yk_913, yk_914, yk_915, yk_916, yk_917, yk_918, yk_919, yk_920, yk_921, yk_922, yk_923, yk_924, yk_925, yk_926, yk_927, yk_928, yk_929, yk_930, yk_931, yk_932, yk_933, yk_934, yk_935, yk_936, yk_937, yk_938, yk_939, yk_940, yk_941, yk_942, yk_943, yk_944, yk_945, yk_946, yk_947, yk_948, yk_949, yk_950, yk_951, yk_952, yk_953, yk_954, yk_955, yk_956, yk_957, yk_958, yk_959, yk_960, yk_961, yk_962, yk_963, yk_964, yk_965, yk_966, yk_967, yk_968, yk_969, yk_970, yk_971, yk_972, yk_973, yk_974, yk_975, yk_976, yk_977, yk_978, yk_979, yk_980, yk_981, yk_982, yk_983, yk_984, yk_985, yk_986, yk_987, yk_988, yk_989, yk_990, yk_991, yk_992, yk_993, yk_994, yk_995, yk_996, yk_997, yk_998, yk_999, yk_1000, yk_1001, yk_1002, yk_1003, yk_1004, yk_1005, yk_1006, yk_1007, yk_1008, yk_1009, yk_1010, yk_1011, yk_1012, yk_1013, yk_1014, yk_1015, yk_1016, yk_1017, yk_1018, yk_1019, yk_1020, yk_1021, yk_1022, yk_1023, yk_1024, yk_1025, yk_1026, yk_1027, yk_1028, yk_1029, yk_1030, yk_1031, yk_1032, yk_1033, yk_1034, yk_1035, yk_1036, yk_1037, yk_1038, yk_1039, yk_1040, yk_1041, yk_1042, yk_1043, yk_1044, yk_1045, yk_1046, yk_1047, yk_1048, yk_1049, yk_1050, yk_1051, yk_1052, yk_1053, yk_1054, yk_1055, yk_1056, yk_1057, yk_1058, yk_1059, yk_1060, yk_1061, yk_1062, yk_1063, yk_1064, yk_1065, yk_1066, yk_1067, yk_1068, yk_1069, yk_1070, yk_1071, yk_1072, yk_1073, yk_1074, yk_1075, yk_1076, yk_1077, yk_1078, yk_1079, yk_1080, yk_1081, yk_1082, yk_1083, yk_1084, yk_1085, yk_1086, yk_1087, yk_1088, yk_1089, yk_1090, yk_1091, yk_1092, yk_1093, yk_1094, yk_1095, yk_1096, yk_1097, yk_1098, yk_1099, yk_1100, yk_1101, yk_1102, yk_1103, yk_1104, yk_1105, yk_1106, yk_1107, yk_1108, yk_1109, yk_1110, yk_1111, yk_1112, yk_1113, yk_1114, yk_1115, yk_1116, yk_1117, yk_1118, yk_1119, yk_1120, yk_1121, yk_1122, yk_1123, yk_1124, yk_1125, yk_1126, yk_1127, yk_1128, yk_1129, yk_1130, yk_1131, yk_1132, yk_1133, yk_1134, yk_1135, yk_1136, yk_1137, yk_1138, yk_1139, yk_1140, yk_1141, yk_1142, yk_1143, yk_1144, yk_1145, yk_1146, yk_1147, yk_1148, yk_1149, yk_1150, yk_1151, yk_1152, yk_1153, yk_1154, yk_1155, yk_1156, yk_1157, yk_1158, yk_1159, yk_1160, yk_1161, yk_1162, yk_1163, yk_1164, yk_1165, yk_1166, yk_1167, yk_1168, yk_1169, yk_1170, yk_1171, yk_1172, yk_1173, yk_1174, yk_1175, yk_1176, yk_1177, yk_1178, yk_1179, yk_1180, yk_1181, yk_1182, yk_1183, yk_1184, yk_1185, yk_1186, yk_1187, yk_1188, yk_1189, yk_1190, yk_1191, yk_1192, yk_1193, yk_1194, yk_1195, yk_1196, yk_1197, yk_1198, yk_1199, yk_1200, yk_1201, yk_1202, yk_1203, yk_1204, yk_1205, yk_1206, yk_1207, yk_1208, yk_1209, yk_1210, yk_1211, yk_1212, yk_1213, yk_1214, yk_1215, yk_1216, yk_1217, yk_1218, yk_1219, yk_1220, yk_1221, yk_1222, yk_1223, yk_1224, yk_1225, yk_1226, yk_1227, yk_1228, yk_1229, yk_1230, yk_1231, yk_1232, yk_1233, yk_1234, yk_1235, yk_1236, yk_1237, yk_1238, yk_1239, yk_1240, yk_1241, yk_1242, yk_1243, yk_1244, yk_1245, yk_1246, yk_1247, yk_1248, yk_1249, yk_1250, yk_1251, yk_1252, yk_1253, yk_1254, yk_1255, yk_1256, yk_1257, yk_1258, yk_1259, yk_1260, yk_1261, yk_1262, yk_1263, yk_1264, yk_1265, yk_1266, yk_1267, yk_1268, yk_1269, yk_1270, yk_1271, yk_1272, yk_1273, yk_1274, yk_1275, yk_1276, yk_1277, yk_1278, yk_1279, yk_1280, yk_1281, yk_1282, yk_1283, yk_1284, yk_1285, yk_1286, yk_1287, yk_1288, yk_1289, yk_1290, yk_1291, yk_1292, yk_1293, yk_1294, yk_1295, yk_1296, yk_1297, yk_1298, yk_1299, yk_1300, yk_1301, yk_1302, yk_1303, yk_1304, yk_1305, yk_1306, yk_1307, yk_1308, yk_1309, yk_1310, yk_1311, yk_1312, yk_1313, yk_1314, yk_1315, yk_1316, yk_1317, yk_1318, yk_1319, yk_1320, yk_1321, yk_1322, yk_1323, yk_1324, yk_1325, yk_1326, yk_1327, yk_1328, yk_1329, yk_1330, yk_1331, yk_1332, yk_1333, yk_1334, yk_1335, yk_1336, yk_1337, yk_1338, yk_1339, yk_1340, yk_1341, yk_1342, yk_1343, yk_1344, yk_1345, yk_1346, yk_1347, yk_1348, yk_1349, yk_1350, yk_1351, yk_1352, yk_1353, yk_1354, yk_1355, yk_1356, yk_1357, yk_1358, yk_1359, yk_1360, yk_1361, yk_1362, yk_1363, yk_1364, yk_1365, yk_1366, yk_1367, yk_1368, yk_1369, yk_1370, yk_1371, yk_1372, yk_1373, yk_1374, yk_1375, yk_1376, yk_1377, yk_1378, yk_1379, yk_1380, yk_1381, yk_1382, yk_1383, yk_1384, yk_1385, yk_1386, yk_1387, yk_1388, yk_1389, yk_1390, yk_1391, yk_1392, yk_1393, yk_1394, yk_1395, yk_1396, yk_1397, yk_1398, yk_1399, yk_1400, yk_1401, yk_1402, yk_1403, yk_1404, yk_1405, yk_1406, yk_1407, yk_1408, yk_1409, yk_1410, yk_1411, yk_1412, yk_1413, yk_1414, yk_1415, yk_1416, yk_1417, yk_1418, yk_1419, yk_1420, yk_1421, yk_1422, yk_1423, yk_1424, yk_1425, yk_1426, yk_1427, yk_1428, yk_1429, yk_1430, yk_1431, yk_1432, yk_1433, yk_1434, yk_1435, yk_1436, yk_1437, yk_1438, yk_1439, yk_1440, yk_1441, yk_1442, yk_1443, yk_1444, yk_1445, yk_1446, yk_1447, yk_1448, yk_1449, yk_1450, yk_1451, yk_1452, yk_1453, yk_1454, yk_1455, yk_1456, yk_1457, yk_1458, yk_1459, yk_1460, yk_1461, yk_1462, yk_1463, yk_1464, yk_1465, yk_1466, yk_1467, yk_1468, yk_1469, yk_1470, yk_1471, yk_1472, yk_1473, yk_1474, yk_1475, yk_1476, yk_1477, yk_1478, yk_1479, yk_1480, yk_1481, yk_1482, yk_1483, yk_1484, yk_1485, yk_1486, yk_1487, yk_1488, yk_1489, yk_1490, yk_1491, yk_1492, yk_1493, yk_1494, yk_1495, yk_1496, yk_1497, yk_1498, yk_1499, yk_1500, yk_1501, yk_1502, yk_1503, yk_1504, yk_1505, yk_1506, yk_1507, yk_1508, yk_1509, yk_1510, yk_1511, yk_1512, yk_1513, yk_1514, yk_1515, yk_1516, yk_1517, yk_1518, yk_1519, yk_1520, yk_1521, yk_1522, yk_1523, yk_1524, yk_1525, yk_1526, yk_1527, yk_1528, yk_1529, yk_1530, yk_1531, yk_1532, yk_1533, yk_1534, yk_1535, yk_1536, yk_1537, yk_1538, yk_1539, yk_1540, yk_1541, yk_1542, yk_1543, yk_1544, yk_1545, yk_1546, yk_1547, yk_1548, yk_1549, yk_1550, yk_1551, yk_1552, yk_1553, yk_1554, yk_1555, yk_1556, yk_1557, yk_1558, yk_1559, yk_1560, yk_1561, yk_1562, yk_1563, yk_1564, yk_1565, yk_1566, yk_1567, yk_1568, yk_1569, yk_1570, yk_1571, yk_1572, yk_1573, yk_1574, yk_1575, yk_1576, yk_1577, yk_1578, yk_1579, yk_1580, yk_1581, yk_1582, yk_1583, yk_1584, yk_1585, yk_1586, yk_1587, yk_1588, yk_1589, yk_1590, yk_1591, yk_1592, yk_1593, yk_1594, yk_1595, yk_1596, yk_1597, yk_1598, yk_1599, yk_1600, yk_1601, yk_1602, yk_1603, yk_1604, yk_1605, yk_1606, yk_1607, yk_1608, yk_1609, yk_1610, yk_1611, yk_1612, yk_1613, yk_1614, yk_1615, yk_1616, yk_1617, yk_1618, yk_1619, yk_1620, yk_1621, yk_1622, yk_1623, yk_1624, yk_1625, yk_1
```

# Loss function

- forward pass (compute loss)
- backward pass (compute param gradient)

```

1 # Import necessary modules
2 import numpy as np
3
4 # Parameters
5 vocab_size = 27
6 num_hidden = 100
7 num_layers = 1
8
9 # Input and target data
10 data_loader = DataLoader('data_loader.py')
11 data_loader.load_data()
12 data_loader.get_data_loader()
13 data_loader.get_data_loader()
14 data_loader.get_data_loader()
15 data_loader.get_data_loader()
16 data_loader.get_data_loader()
17 data_loader.get_data_loader()
18 data_loader.get_data_loader()
19 data_loader.get_data_loader()
20 data_loader.get_data_loader()
21 data_loader.get_data_loader()
22 data_loader.get_data_loader()
23 data_loader.get_data_loader()
24 data_loader.get_data_loader()
25 data_loader.get_data_loader()
26 data_loader.get_data_loader()
27 data_loader.get_data_loader()
28 data_loader.get_data_loader()
29 data_loader.get_data_loader()
30 data_loader.get_data_loader()
31 data_loader.get_data_loader()
32 data_loader.get_data_loader()
33 data_loader.get_data_loader()
34 data_loader.get_data_loader()
35 data_loader.get_data_loader()
36 data_loader.get_data_loader()
37 data_loader.get_data_loader()
38 data_loader.get_data_loader()
39 data_loader.get_data_loader()
40 data_loader.get_data_loader()
41 data_loader.get_data_loader()
42 data_loader.get_data_loader()
43 data_loader.get_data_loader()
44 data_loader.get_data_loader()
45 data_loader.get_data_loader()
46 data_loader.get_data_loader()
47 data_loader.get_data_loader()
48 data_loader.get_data_loader()
49 data_loader.get_data_loader()
50 data_loader.get_data_loader()
51 data_loader.get_data_loader()
52 data_loader.get_data_loader()
53 data_loader.get_data_loader()
54 data_loader.get_data_loader()
55 data_loader.get_data_loader()
56 data_loader.get_data_loader()
57 data_loader.get_data_loader()
58 data_loader.get_data_loader()
59 data_loader.get_data_loader()
60 data_loader.get_data_loader()
61 data_loader.get_data_loader()
62 data_loader.get_data_loader()
63 data_loader.get_data_loader()
64 data_loader.get_data_loader()
65 data_loader.get_data_loader()
66 data_loader.get_data_loader()
67 data_loader.get_data_loader()
68 data_loader.get_data_loader()
69 data_loader.get_data_loader()
70 data_loader.get_data_loader()
71 data_loader.get_data_loader()
72 data_loader.get_data_loader()
73 data_loader.get_data_loader()
74 data_loader.get_data_loader()
75 data_loader.get_data_loader()
76 data_loader.get_data_loader()
77 data_loader.get_data_loader()
78 data_loader.get_data_loader()
79 data_loader.get_data_loader()
80 data_loader.get_data_loader()
81 data_loader.get_data_loader()
82 data_loader.get_data_loader()
83 data_loader.get_data_loader()
84 data_loader.get_data_loader()
85 data_loader.get_data_loader()
86 data_loader.get_data_loader()
87 data_loader.get_data_loader()
88 data_loader.get_data_loader()
89 data_loader.get_data_loader()
90 data_loader.get_data_loader()
91 data_loader.get_data_loader()
92 data_loader.get_data_loader()
93 data_loader.get_data_loader()
94 data_loader.get_data_loader()
95 data_loader.get_data_loader()
96 data_loader.get_data_loader()
97 data_loader.get_data_loader()
98 data_loader.get_data_loader()
99 data_loader.get_data_loader()
100 data_loader.get_data_loader()

```

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36
37     # Forward pass
38     for t in xrange(len(inputs)):
39         xs[t] = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
40         xs[t][inputs[t]] = 1
41         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
42         ys[t] = np.dot(Wyh, hs[t]) + by # unnormalized log probabilities for next chars
43         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
44         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
45
46     # Backward pass: compute gradients going backwards
47     dhdh, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Wyh)
48     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
49     dhnext = np.zeros_like(hs[0])
50     for t in reversed(xrange(len(inputs))):
51         dy = np.copy(ps[t])
52         dy[targets[t]] -= 1 # backprop into y
53         dwhy += np.dot(dy, hs[t].T)
54         dby += dy
55         dh = np.dot(Wyh.T, dy) + dhnext # backprop into h
56         dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
57         dbh += dhraw
58         dwhh += np.dot(dhraw, xs[t].T)
59         dwhh += np.dot(dhraw, hs[t-1].T)
60         dhnext = np.dot(Whh.T, dhraw)
61
62     for dparam in [dwhh, dwhh, dwhy, dbh, dby]:
63         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
64     return loss, dwhh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

n Andrej Karpathy  
1n

cor

## n-char-rnn.py gist

```

1 #!/usr/bin/perl -w
2 #
3 # This script is used to generate a random string of a given length.
4 #
5 # Usage: perl random_string.pl [length]
6 #
7 # Example: perl random_string.pl 10
8 #
9 # Output: 0x0000000000000000
10
11 #
12 # This script is used to generate a random string of a given length.
13 #
14 # Usage: perl random_string.pl [length]
15 #
16 # Example: perl random_string.pl 10
17 #
18 # Output: 0x0000000000000000
19
20 #
21 # This script is used to generate a random string of a given length.
22 #
23 # Usage: perl random_string.pl [length]
24 #
25 # Example: perl random_string.pl 10
26 #
27 # Output: 0x0000000000000000
28
29 #
30 # This script is used to generate a random string of a given length.
31 #
32 # Usage: perl random_string.pl [length]
33 #
34 # Example: perl random_string.pl 10
35 #
36 # Output: 0x0000000000000000
37
38 #
39 # This script is used to generate a random string of a given length.
40 #
41 # Usage: perl random_string.pl [length]
42 #
43 # Example: perl random_string.pl 10
44 #
45 # Output: 0x0000000000000000
46
47 #
48 # This script is used to generate a random string of a given length.
49 #
50 # Usage: perl random_string.pl [length]
51 #
52 # Example: perl random_string.pl 10
53 #
54 # Output: 0x0000000000000000
55
56 #
57 # This script is used to generate a random string of a given length.
58 #
59 # Usage: perl random_string.pl [length]
60 #
61 # Example: perl random_string.pl 10
62 #
63 # Output: 0x0000000000000000
64
65 #
66 # This script is used to generate a random string of a given length.
67 #
68 # Usage: perl random_string.pl [length]
69 #
70 # Example: perl random_string.pl 10
71 #
72 # Output: 0x0000000000000000
73
74 #
75 # This script is used to generate a random string of a given length.
76 #
77 # Usage: perl random_string.pl [length]
78 #
79 # Example: perl random_string.pl 10
80 #
81 # Output: 0x0000000000000000
82
83 #
84 # This script is used to generate a random string of a given length.
85 #
86 # Usage: perl random_string.pl [length]
87 #
88 # Example: perl random_string.pl 10
89 #
90 # Output: 0x0000000000000000
91
92 #
93 # This script is used to generate a random string of a given length.
94 #
95 # Usage: perl random_string.pl [length]
96 #
97 # Example: perl random_string.pl 10
98 #
99 # Output: 0x0000000000000000
100

```

[illegible][illegible]

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(Wyh, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)

```

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned}$$

# Softmax classifier

# n-char-rnn.py gist

```

1 # n-char-rnn.py: vanilla rnn model, written by Andrej Karpathy (@karpathy)
2
3 # imports
4
5 # imports numpy as np
6
7 # main fn
8
9 data = open('input.txt', 'r').read() # should be simple plain text file
10 chars = list(data)
11 data_size, vocab_size = len(data), len(chars)
12 print "data has %d characters, of which %d unique" % (data_size, vocab_size)
13 char_to_ix = {c: ix for ix, c in enumerate(chars)}
14 ix_to_char = {ix: c for ix, c in enumerate(chars)}
15
16 # hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 seq_length = 20 # number of steps to unroll the rnn for
19 learning_rate = 0.001
20
21 # model parameters
22 wih = np.random.randn(hidden_size, data_size) # input to hidden
23 whh = np.random.randn(hidden_size, hidden_size) # hidden to hidden
24 why = np.random.randn(hidden_size, vocab_size) # hidden to output
25 wbi = np.random.randn(data_size, 1) # bias to input
26 wby = np.random.randn(vocab_size, 1) # bias to output

```

```

27 def forward(inputs, targets, hprev):
28     loss = 0
29     inputs, targets are both list of integers
30     hprev is the array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32
33     # init, h0, y0, x0 = 0, 0, 0, 0
34     h0 = 0 # no previous hidden
35     y0 = 0 # no previous output
36     x0 = 0 # no previous input
37
38     for t in xrange(len(inputs)):
39         xi = np.random.randn(data_size) # sample in 0..1 of a representation
40         xi = xi * inputs[t] + 1
41         h0 = np.tanh(dot(hprev, whh) + dot(xi, wih) + wbi) # h0 = hidden state
42         y0 = np.dot(h0, why) # h0 = current hidden state, y0 = output
43         y0 = np.exp(y0) / np.sum(np.exp(y0)) # probabilities for next char
44         loss += -np.log(y0[targets[t]]) # softmax cross-entropy loss
45         # backward pass: compute gradients going backwards

```

```

46     dh0, dxi, dhy = np.zeros_like(hprev), np.zeros_like(xi), np.zeros_like(y0)
47     dprev_h, dprev_x, dprev_y = np.zeros_like(hprev), np.zeros_like(xi), np.zeros_like(y0)
48     for t in reversed(xrange(len(inputs))):
49         # h_t = hidden state
50         dxi = np.zeros_like(xi)
51         dhy = np.zeros_like(y0)
52         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
53         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
54         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
55         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
56         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
57         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
58         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
59         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity
60         dxi = dxi + dhy * wby # dxi = backward through bias nonlinearity

```

```

61     # sample from the model
62     sample = np.argmax(y0)
63     h = np.dot(h0, whh) + dot(xi, wih) + wbi
64     xi = np.random.randn(data_size, 1)
65     y = np.dot(h, why)
66     y = np.exp(y) / np.sum(np.exp(y))
67     loss += -np.log(y[sample])
68     hprev = h
69     return loss, dprev_h, dprev_x, dprev_y

```

```

70 # main fn
71
72 # init, h0, y0, x0 = 0, 0, 0, 0
73 h0, y0, x0 = np.zeros_like(hprev), np.zeros_like(xi), np.zeros_like(y0)
74 hprev, xprev, yprev = np.zeros_like(hprev), np.zeros_like(xi), np.zeros_like(y0)
75 seq_length = np.random.randint(10, 20) # sample seq length
76
77 # unroll the rnn
78 for t in xrange(seq_length):
79     # sample from the model
80     sample = np.argmax(y0)
81     h = np.dot(h0, whh) + dot(xi, wih) + wbi
82     xi = np.random.randn(data_size, 1)
83     y = np.dot(h, why)
84     y = np.exp(y) / np.sum(np.exp(y))
85     loss += -np.log(y[sample])
86     hprev = h
87     return loss, dprev_h, dprev_x, dprev_y

```

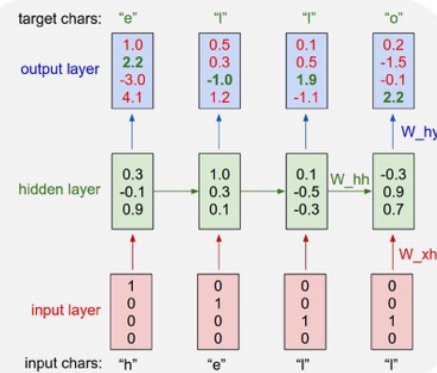
n Andrej Karpathy  
1n

```

44 # backward pass: compute gradients going backwards
45 dwhx, dwhh, dwhy = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(why)
46 dbh, dby = np.zeros_like(bi), np.zeros_like(by)
47 dhnext = np.zeros_like(hs[0])
48 for t in reversed(xrange(len(inputs))):
49     dy = np.copy(ps[t])
50     dy[targets[t]] -= 1 # backprop into y
51     dwhy += np.dot(dy, hs[t].T)
52     dby += dy
53     dh = np.dot(why.T, dy) + dhnext # backprop into h
54     dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55     dbh += dhraw
56     dwhx += np.dot(dhraw, xs[t].T)
57     dwhh += np.dot(dhraw, hs[t-1].T)
58     dhnext = np.dot(whh.T, dhraw)
59 for dparam in [dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]]:
60     np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61 return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

recall:



## [n-char-rnn.py gist](#)

[illegible]

```

63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
73         y = np.dot(Why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79     return ixes

```

n Andrej Karpathy  
1n

comp150dl







## Sonnet 116 – Let me not ...

*by William Shakespeare*

Let me not to the marriage of true minds  
Admit impediments. Love is not love  
Which alters when it alteration finds,  
Or bends with the remover to remove:  
O no! it is an ever-fixed mark  
That looks on tempests and is never shaken;  
It is the star to every wandering bark,  
Whose worth's unknown, although his height be taken.  
Love's not Time's fool, though rosy lips and cheeks  
Within his bending sickle's compass come:  
Love alters not with his brief hours and weeks,  
But bears it out even to the edge of doom.  
If this be error and upon me proved,  
I never writ, nor no man ever loved.

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e  
plia tklrgrd t o idoe ns,smtt h ne etie h,hregtrs nigtkie,aoaenns lng



train more

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y fil on aseterlome  
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."



train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of  
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and ofter.



train more

"Why do what that day," replied Natasha, and wishing to himself the fact the  
princess, Princess Mary was easier, fed in had oftended him.  
Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day  
When little strain would be attain'd into being never fed,  
And who is but a chain and subjects of his death,  
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,  
Breaking and strongly should be buried, when I perish  
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and  
my fair nudes begun out of the fact, to be conveyed,  
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

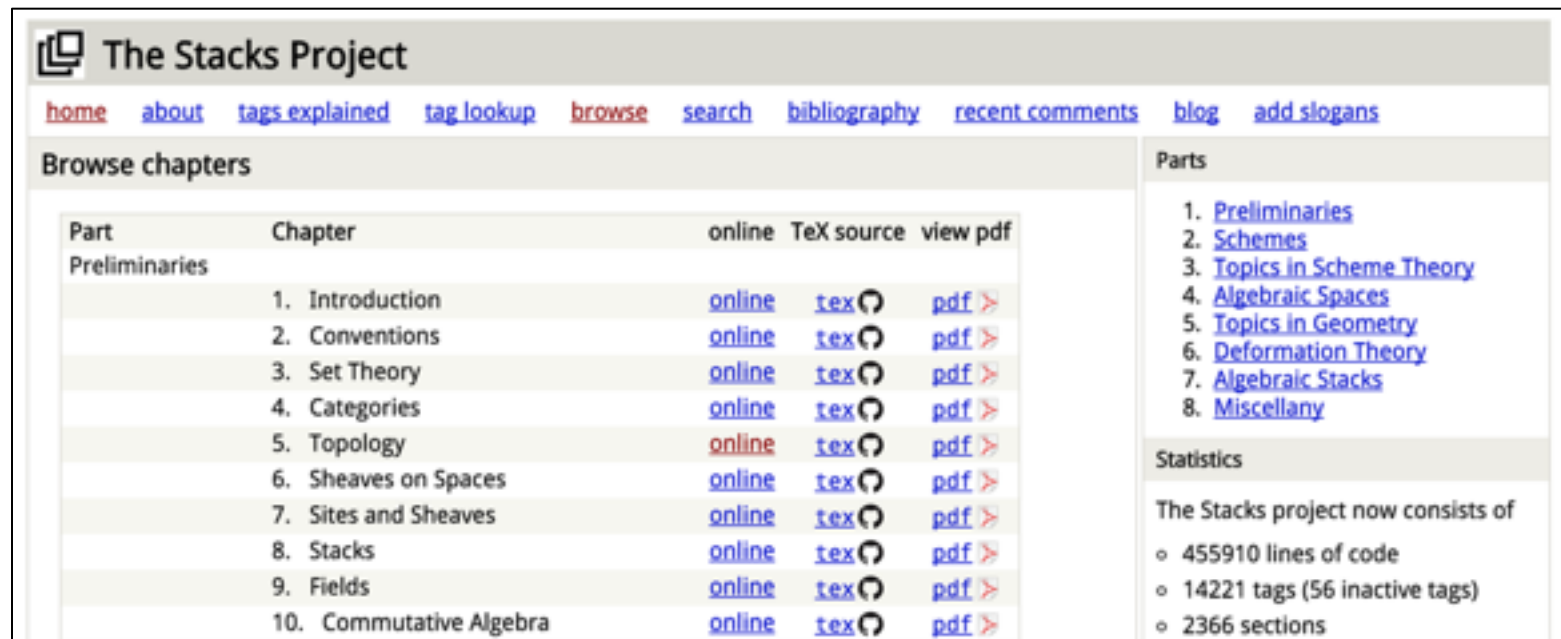
VIOLA:

Why, Salisbury must find his flesh and thought  
That which I am not apt, not a man and in fire,  
To show the reining of the raven and the wars  
To grace my hand reproach within, and not a fair are hand,  
That Caesar and my goodly father's world;  
When I was heaven of presence and our fleets,  
We spare with hours, but cut thy council I am great,  
Murdered and by thy master's ready there  
My power to give thee but so much as hell:  
Some service in the noble bondman here,  
Would show him to her wine.





















KING LEAR:

O, if you were a feeble sight, the courtesy of your law,  
Your sight and several breath, will wear the gods  
With his heads, and my hands are wonder'd at the deeds,  
So drop upon your lordship's head, and your opinion  
Shall be against your honour.

# open source textbook on algebraic geometry



The screenshot shows the homepage of The Stacks Project. At the top is the logo and title 'The Stacks Project'. Below it is a navigation bar with links: home, about, tags explained, tag lookup, browse, search, bibliography, recent comments, blog, and add slogans. The main content area is divided into two sections. The left section, 'Browse chapters', contains a table with 10 rows. The first row is 'Preliminaries'. The subsequent rows are numbered 1 through 10, corresponding to the chapters in the right sidebar. Each row has three links: 'online', 'TeX source', and 'view pdf'. The right section, 'Parts', contains a list of 8 items, each with a number and a link. The first item is '1. Preliminaries'. The second item is '2. Schemes'. The third item is '3. Topics in Scheme Theory'. The fourth item is '4. Algebraic Spaces'. The fifth item is '5. Topics in Geometry'. The sixth item is '6. Deformation Theory'. The seventh item is '7. Algebraic Stacks'. The eighth item is '8. Miscellany'. Below this list is a section titled 'Statistics' which states 'The Stacks project now consists of' followed by three bullet points: '455910 lines of code', '14221 tags (56 inactive tags)', and '2366 sections'.

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	2. Conventions	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	3. Set Theory	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	4. Categories	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	5. Topology	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	6. Sheaves on Spaces	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	7. Sites and Sheaves	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	8. Stacks	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	9. Fields	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 
	10. Commutative Algebra	<a href="#">online</a>	<a href="#">tex</a> 	<a href="#">pdf</a> 

Parts

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

For  $\bigoplus_{i=1,\dots,m} \mathcal{L}_{\mathcal{M}_i} = 0$ , hence we can find a closed subset  $\mathcal{H}$  in  $\mathcal{H}$  and any sets  $\mathcal{F}$  on  $X$ ,  $U$  is a closed immersion of  $S$ , then  $U \rightarrow T$  is a separated algebraic space.

*Proof.* Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparico in the fibre product covering we have to prove the lemma generated by  $\coprod Z \times_U U \rightarrow V$ . Consider the maps  $M$  along the set of points  $\mathrm{Sch}_{\mathrm{fppf}}$  and  $U \rightarrow U$  is the fibre category of  $S$  in  $U$  in Section, ?? and the fact that any  $U$  affine, see Morphisms, Lemma ?? . Hence we obtain a scheme  $S$  and any open subset  $W \subset U$  in  $\mathrm{Sh}(G)$  such that  $\mathrm{Spec}(R') \rightarrow S$  is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that  $f_i$  is of finite presentation over  $S$ . We claim that  $\mathcal{O}_{X,s}$  is a scheme where  $x, x', s'' \in S'$  such that  $\mathcal{O}_{X,s'} \rightarrow \mathcal{O}'_{X',s'}$  is separated. By Algebra, Lemma ?? we can define a map of complexes  $\mathrm{GL}_{S'}(x'/S'')$  and we win.  $\square$

To prove study we see that  $\mathcal{F}|_U$  is a covering of  $\mathcal{X}'$ , and  $\mathcal{T}_i$  is an object of  $\mathcal{F}_{X/S}$  for  $i > 0$  and  $\mathcal{F}_p$  exists and let  $\mathcal{F}_i$  be a presheaf of  $\mathcal{O}_X$ -modules on  $\mathcal{C}$  as a  $\mathcal{F}$ -module. In particular  $\mathcal{F} = U/\mathcal{F}$  we have to show that

$$\tilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} = i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (\mathrm{Sch}/S)_{\mathrm{fppf}}^{\mathrm{opp}}, (\mathrm{Sch}/S)_{\mathrm{fppf}}$$

and

$$V = \Gamma(S, \mathcal{O}) \hookrightarrow (U, \mathrm{Spec}(A))$$

is an open subset of  $X$ . Thus  $U$  is affine. This is a continuous map of  $X$  is the inverse, the groupoid scheme  $S$ .

*Proof.* See discussion of sheaves of sets.  $\square$

The result for prove any open covering follows from the less of Example ?? . It may replace  $S$  by  $X_{\mathrm{spaces, \acute{e}tale}}$  which gives an open subspace of  $X$  and  $T$  equal to  $S_{\mathrm{var}}$ , see Descent, Lemma ?? . Namely, by Lemma ?? we see that  $R$  is geometrically regular over  $S$ .

**Lemma 0.1.** Assume (3) and (3) by the construction in the description.

Suppose  $X = \lim |X|$  (by the formal open covering  $X$  and a single map  $\mathrm{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$  over  $U$  compatible with the complex

$$\mathrm{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X, \mathcal{O}_X}).$$

When in this case of to show that  $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$  is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If  $T$  is surjective we may assume that  $T$  is connected with residue fields of  $S$ . Moreover there exists a closed subspace  $Z \subset X$  of  $X$  where  $U$  in  $X'$  is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1)  $f$  is locally of finite type. Since  $S = \mathrm{Spec}(R)$  and  $Y = \mathrm{Spec}(R)$ .

*Proof.* This is form all sheaves of sheaves on  $X$ . But given a scheme  $U$  and a surjective étale morphism  $U \rightarrow X$ . Let  $U \cap U = \coprod_{i=1,\dots,n} U_i$  be the scheme  $X$  over  $S$  at the schemes  $X_i \rightarrow X$  and  $U = \lim_i X_i$ .  $\square$

The following lemma surjective restocomposes of this implies that  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\dots,0}$ .

**Lemma 0.2.** Let  $X$  be a locally Noetherian scheme over  $S$ ,  $E = \mathcal{F}_{X/S}$ . Set  $\mathcal{I} = \mathcal{I}_1 \subset \mathcal{I}_n$ . Since  $\mathcal{I}^\bullet \subset \mathcal{I}^\bullet$  are nonzero over  $i_0 \leq p$  is a subset of  $\mathcal{I}_{n,0} \circ \bar{A}_2$  works.

**Lemma 0.3.** In Situation ?? . Hence we may assume  $q' = 0$ .

*Proof.* We will use the property we see that  $\mathfrak{p}$  is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where  $K$  is an  $F$ -algebra where  $\delta_{n+1}$  is a scheme over  $S$ .  $\square$



Proof. Omitted. □

**Lemma 0.1.** Let  $\mathcal{C}$  be a set of the construction.

Let  $\mathcal{C}$  be a gerber covering. Let  $\mathcal{F}$  be a quasi-coherent sheaves of  $\mathcal{O}$ -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves  $\mathcal{F}$  on  $X_{\text{étale}}$  we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where  $\mathcal{G}$  defines an isomorphism  $\mathcal{F} \rightarrow \mathcal{F}$  of  $\mathcal{O}$ -modules. □

**Lemma 0.2.** This is an integer  $\mathbb{Z}$  is injective.

Proof. See Spaces, Lemma ?? □

**Lemma 0.3.** Let  $S$  be a scheme. Let  $X$  be a scheme and  $X$  is an affine open covering. Let  $\mathcal{U} \subset X$  be a canonical and locally of finite type. Let  $X$  be a scheme. Let  $X$  be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let

$$b: X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

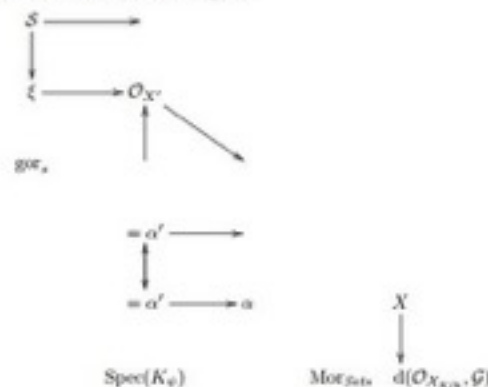
be a morphism of algebraic spaces over  $S$  and  $Y$ .

Proof. Let  $X$  be a nonzero scheme of  $X$ . Let  $X$  be an algebraic space. Let  $\mathcal{F}$  be a quasi-coherent sheaf of  $\mathcal{O}_X$ -modules. The following are equivalent

- (1)  $\mathcal{F}$  is an algebraic space over  $S$ .
- (2) If  $X$  is an affine open covering.

Consider a common structure on  $X$  and  $X$  the functor  $\mathcal{O}_X(U)$  which is locally of finite type. □

This since  $\mathcal{F} \in \mathcal{F}$  and  $x \in \mathcal{G}$  the diagram



is a limit. Then  $\mathcal{G}$  is a finite type and assume  $S$  is a flat and  $\mathcal{F}$  and  $\mathcal{G}$  is a finite type  $f_*$ . This is of finite type diagrams, and

- the composition of  $\mathcal{G}$  is a regular sequence,
- $\mathcal{O}_{X'}$  is a sheaf of rings.

□

Proof. We have see that  $X = \text{Spec}(R)$  and  $\mathcal{F}$  is a finite type representable by algebraic space. The property  $\mathcal{F}$  is a finite morphism of algebraic stacks. Then the cohomology of  $X$  is an open neighbourhood of  $U$ . □

Proof. This is clear that  $\mathcal{G}$  is a finite presentation, see Lemmas ??.

A reduced above we conclude that  $U$  is an open covering of  $\mathcal{C}$ . The functor  $\mathcal{F}$  is a "field"

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_x \rightarrow \mathcal{O}_{X,x} \rightarrow \mathcal{O}_{X,x} \rightarrow \mathcal{O}_{X,x} \rightarrow \mathcal{O}_{X,x}$$

is an isomorphism of covering of  $\mathcal{O}_{X_1}$ . If  $\mathcal{F}$  is the unique element of  $\mathcal{F}$  such that  $X$  is an isomorphism.

The property  $\mathcal{F}$  is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme  $\mathcal{O}_X$ -algebra with  $\mathcal{F}$  are opens of finite type over  $S$ . If  $\mathcal{F}$  is a scheme theoretic image points. □

If  $\mathcal{F}$  is a finite direct sum  $\mathcal{O}_{X_1}$  is a closed immersion, see Lemma ?? . This is a sequence of  $\mathcal{F}$  is a similar morphism.



## torvalds / linux

👁 Watch - 3,711

★ Star 23,054

🍴 Fork 9,141

### Linux kernel source tree

📄 520,037 commits

🌿 1 branch

📦 420 releases

👤 5,039 contributors

🌿 branch: master - linux / +

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux



**torvalds** authored 9 hours ago

latest commit 4b1706927d

Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/hab/target-pending	6 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l...	a day ago
block	block: discard bdi_unregister() in favour of bdi_destroy()	9 days ago
crypto	Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6	10 days ago
drivers	Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	9 hours ago
firmware	firmware/hex2fw.c: restore missing default in switch statement	2 months ago
fs	vfs: read file_handle only once in handle_to_path	4 days ago
include	Merge branch 'perl-urgent-for-linus' of git://git.kernel.org/pub/scm/...	a day ago
init	init: fix regression by supporting devices with major:minor:offset fo...	a month ago
io	Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel...	a month ago

<> Code

🔗 Pull requests 74

📡 Pulse

📊 Graphs

HTTPS clone URL

https://github.com/torvalds/linux

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).

📁 Clone in Desktop

📄 Download ZIP

# Generated C code

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

```

/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 *
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

```

```

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP AFSR(0, load)
#define STACK_DDR(type)      (func)

#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs() arch_get_unaligned_child()
#define access_rw(TST) asm volatile("movd %!esp, %0, %!3" : : "r" (0)); \
    if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
        (unsigned long)-1->lr_full; low;
}

```

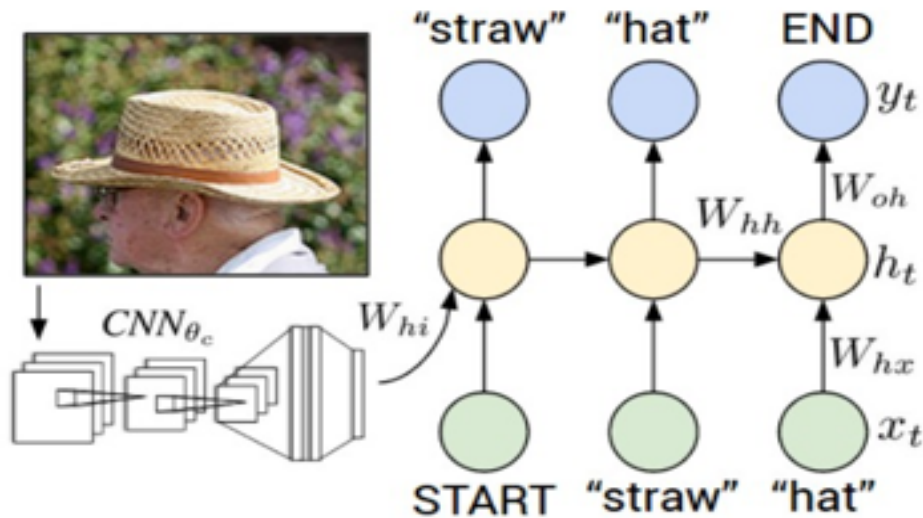
# Recommended Reading:

## *Visualizing and Understanding Recurrent Networks*

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
     */
}
```

*[Visualizing and Understanding Recurrent Networks, Andrej Karpathy\*, Justin Johnson\*, Li Fei-Fei]*

# Image Captioning



*Explain Images with Multimodal Recurrent Neural Networks, Mao et al.*

*Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei*

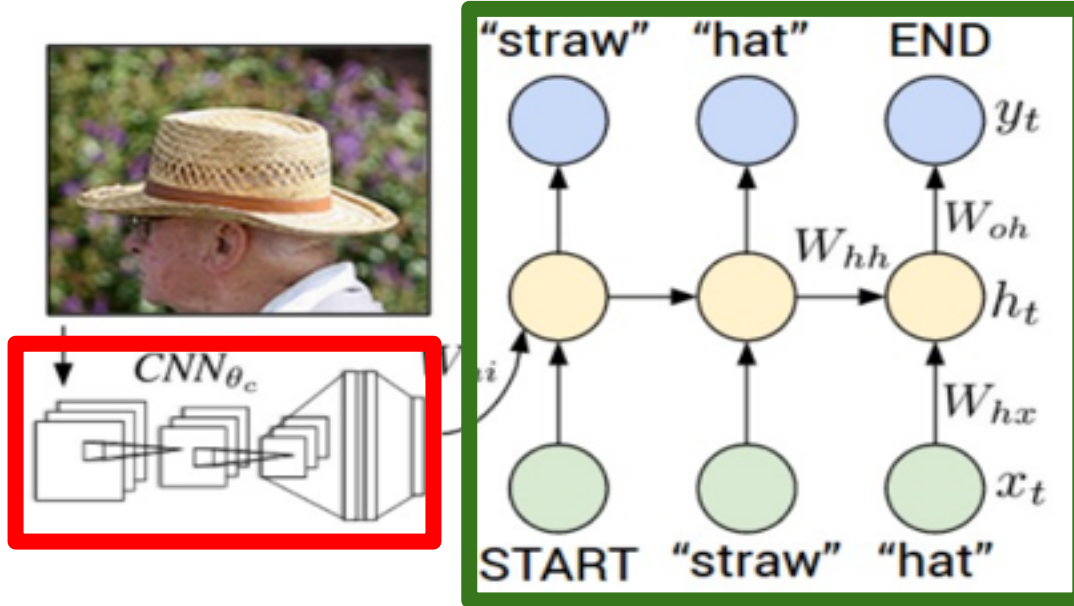
*Show and Tell: A Neural Image Caption Generator, Vinyals et al.*

*Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.*

*Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick*



# Recurrent Neural Network



# Convolutional Neural Network



test image

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

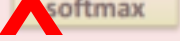
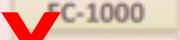
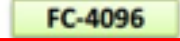
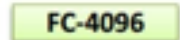
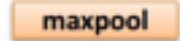
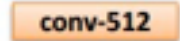
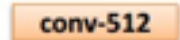
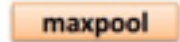
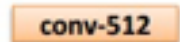
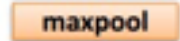
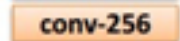
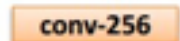
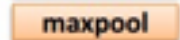
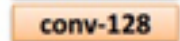
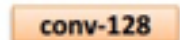
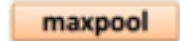
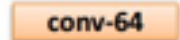
FC-4096

FC-1000

softmax



test image



test image

\* softmax

comp150dl  **Tufts**  
UNIVERSITY

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

x0  
<STA  
RT>

\* (ar

ved from Andrej Karpathy  
rd cs231n

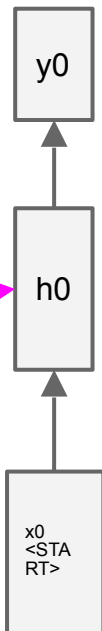
<START>

comp150dl





**W<sub>ih</sub>**



**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$



test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



test image

y0

h0

x0  
<START>  
RT>

straw

sample!

ved from Andrej Karpathy  
rd cs231n

comp150dl



\* (ar

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

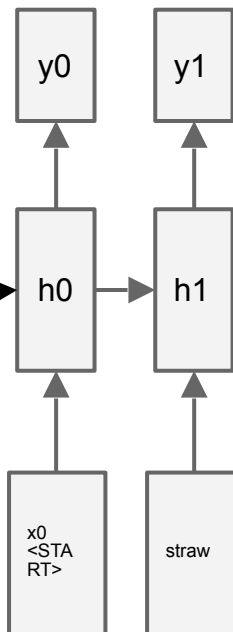
maxpool

FC-4096

FC-4096



test image



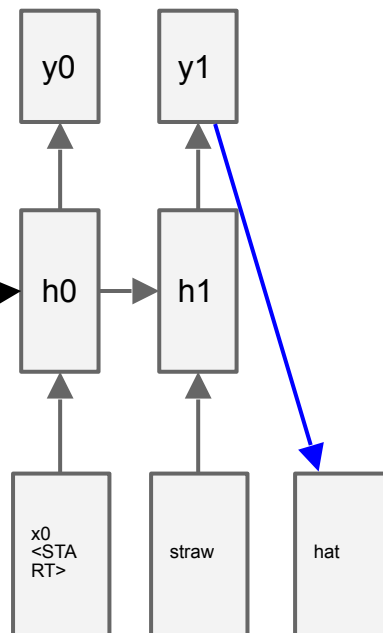
ved from Andrej Karpathy  
rd cs231n

comp150dl





test image



sample!

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

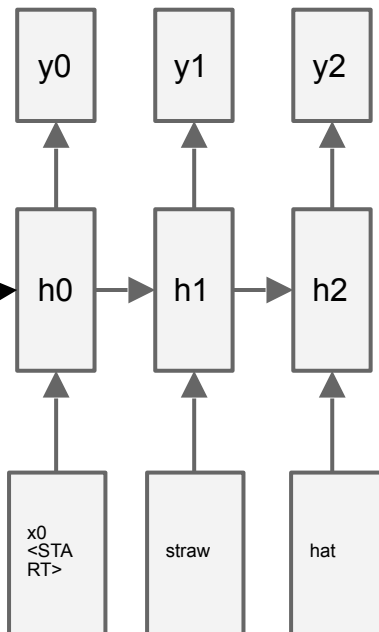
maxpool

FC-4096

FC-4096



test image



ved from Andrej Karpathy  
rd cs231n

comp150dl



\* (ar

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

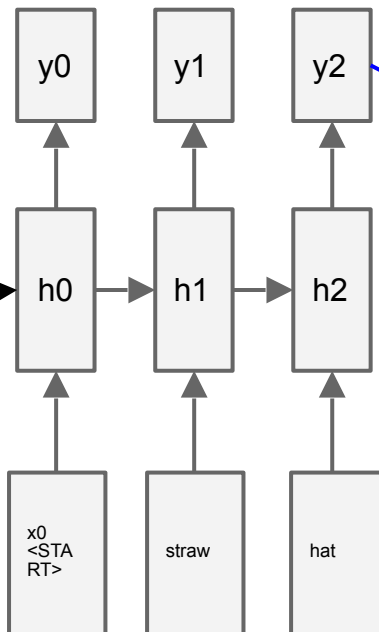
maxpool

FC-4096

FC-4096



test image



sample  
<END> token  
=> finish.

# Image Sentence Datasets

a man riding a bike on a dirt path through a forest.  
bicyclist raises his fist as he rides on desert dirt trail.  
this dirt bike rider is smiling and raising his fist in triumph.  
a man riding a bicycle while pumping his fist in the air.  
a mountain biker pumps his fist in celebration.



## Microsoft COCO

[Tsung-Yi Lin et al. 2014]

[mscoco.org](http://mscoco.org)

currently:

~120K images

~5 sentences each





"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."



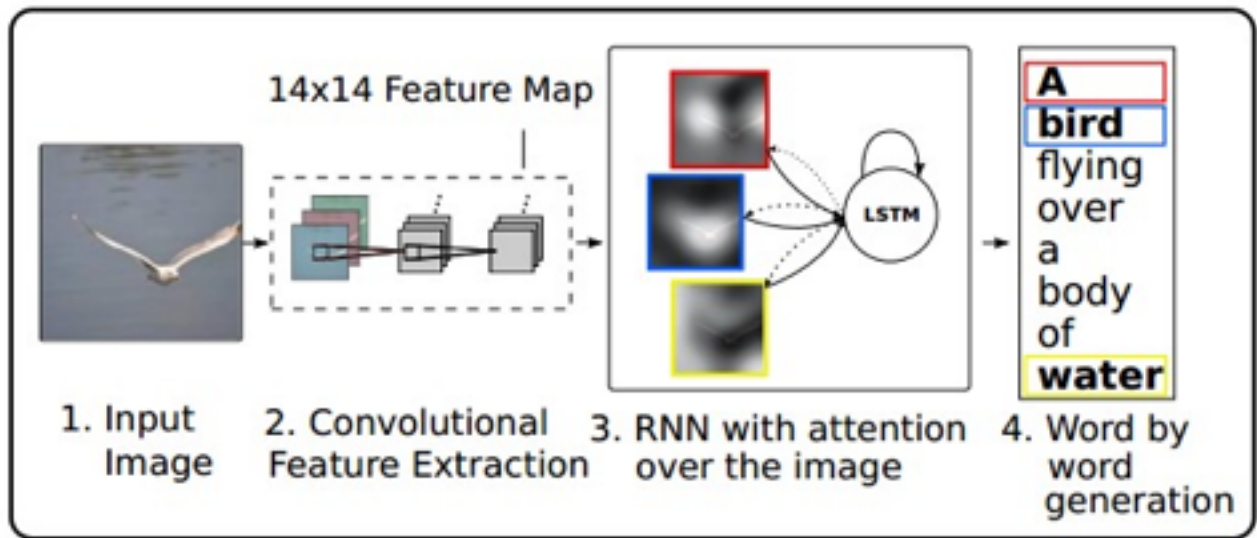
"a woman holding a teddy bear in front of a mirror."



"a horse is standing in the middle of a road."

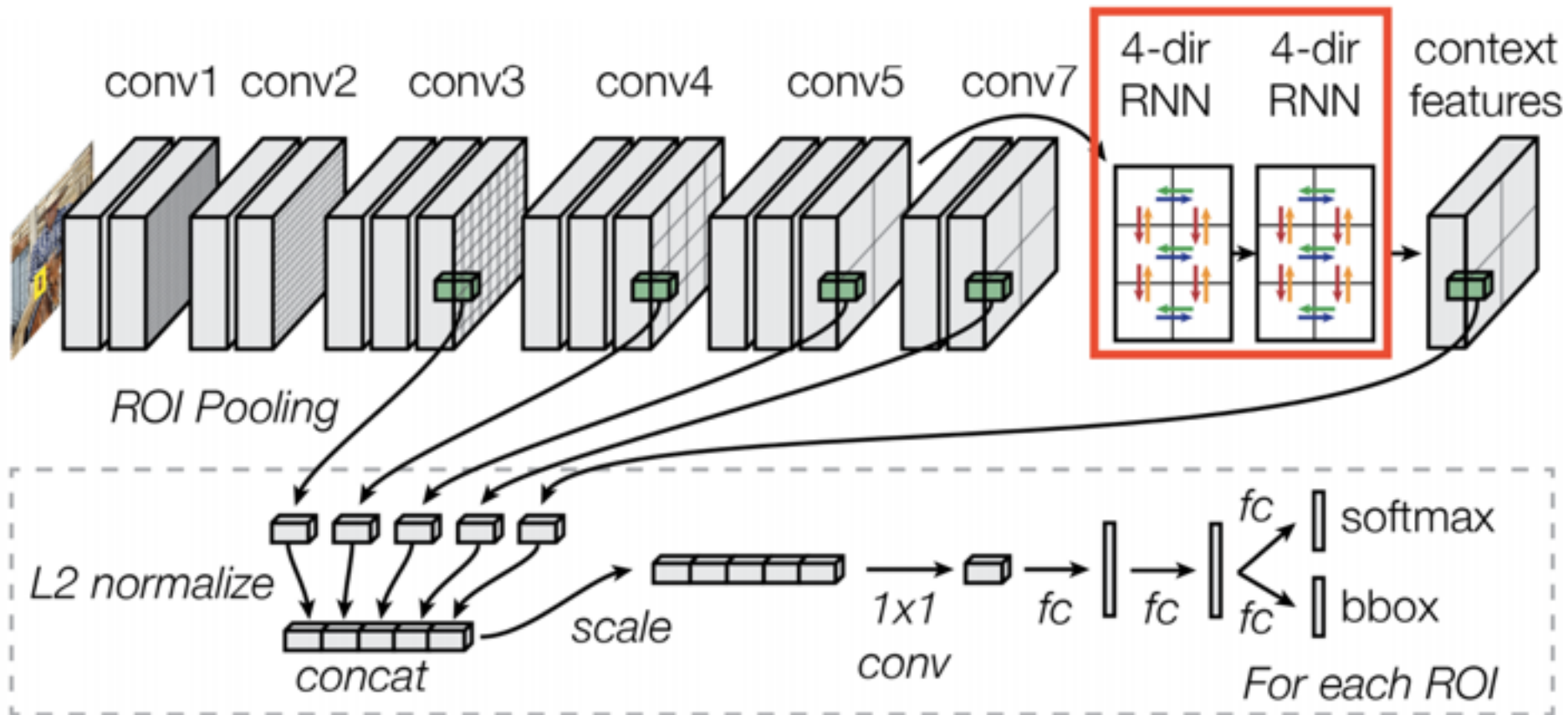
# Preview of fancier architectures

RNN attends spatially to different parts of images while generating each word of the sentence:



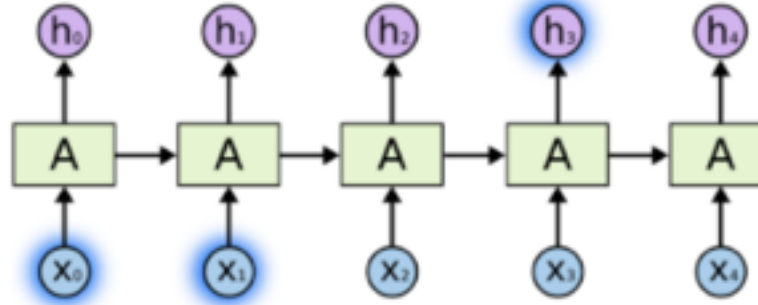
*Show Attend and Tell, Xu et al., 2015*

# ION: INSIDE-OUTSIDE NET

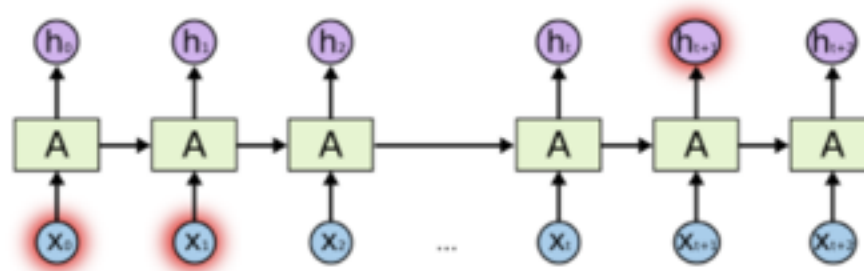




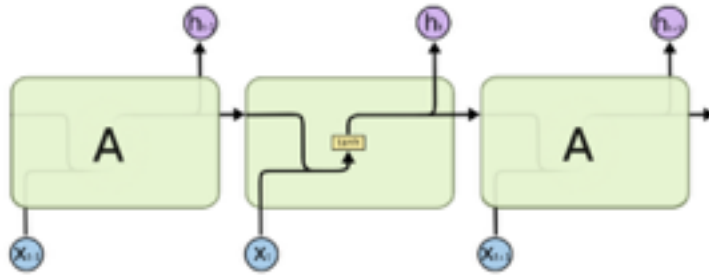
# Limitations of RNNs



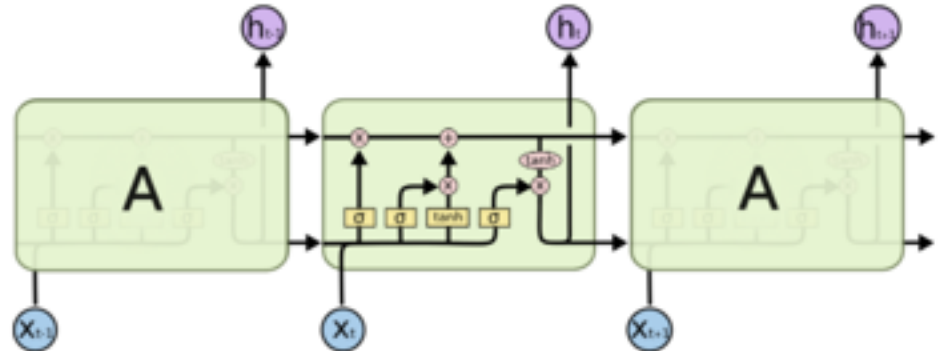
“I grew up in France... I speak fluent ***French***.”



# Long Short Term Memory Networks



The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.



# RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$h \in \mathbb{R}^n, \quad W^l [n \times 2n]$$

# LSTM:

$$W^l [4n \times 2n]$$

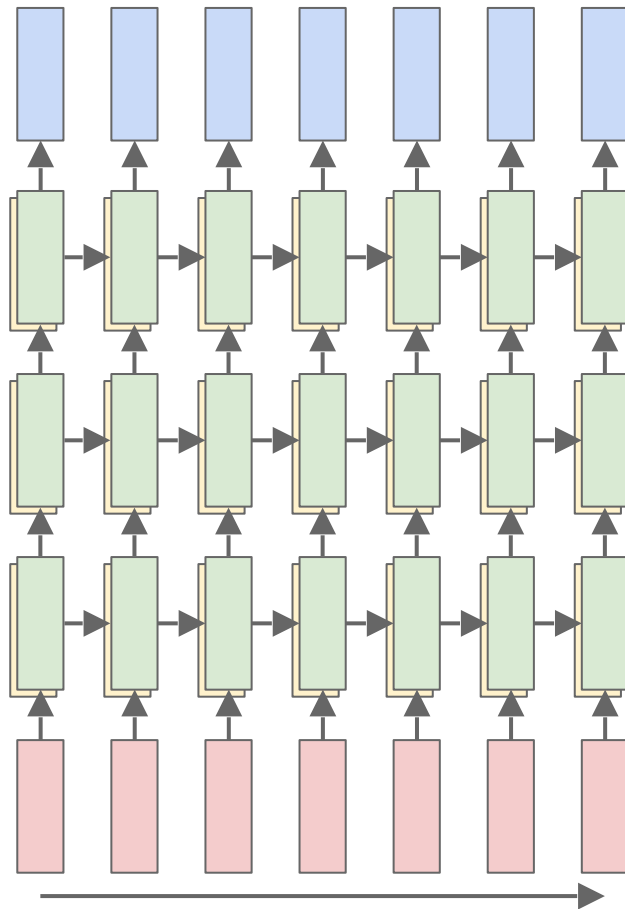
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

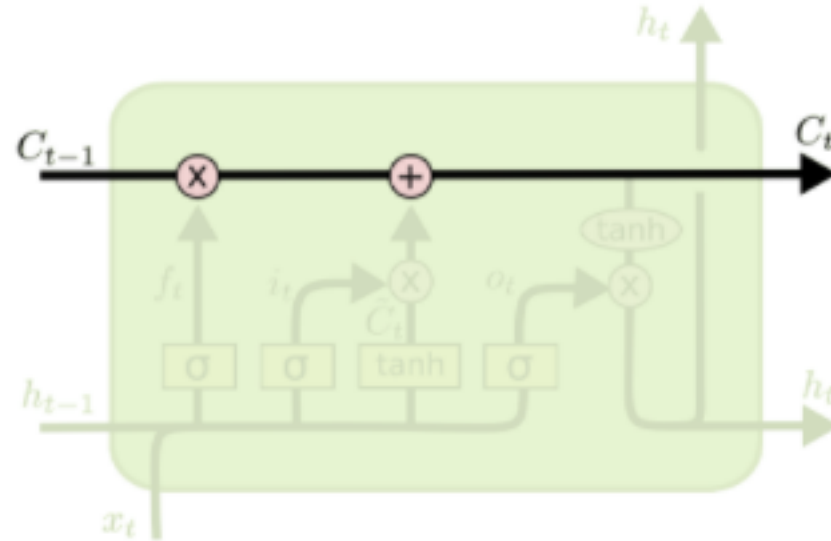
depth

time



# LSTM: Cell State

long running memory of the network

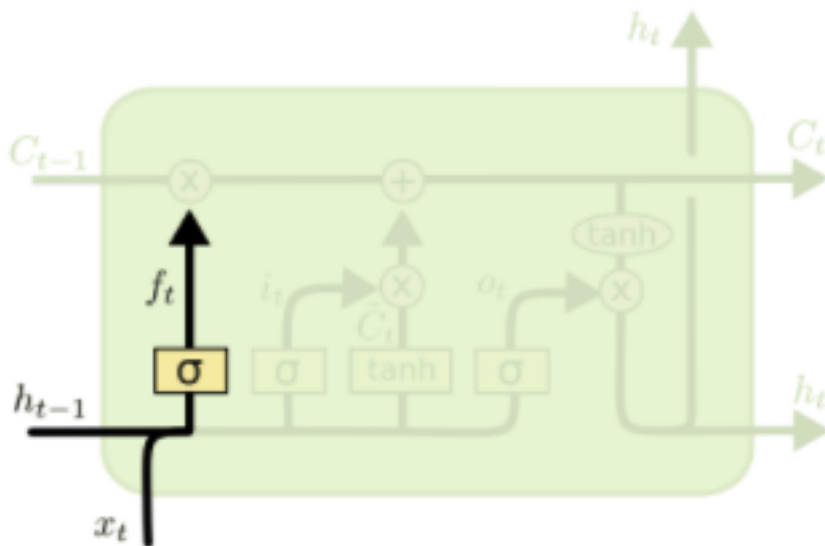


# LSTM: Forget Gate $f$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$



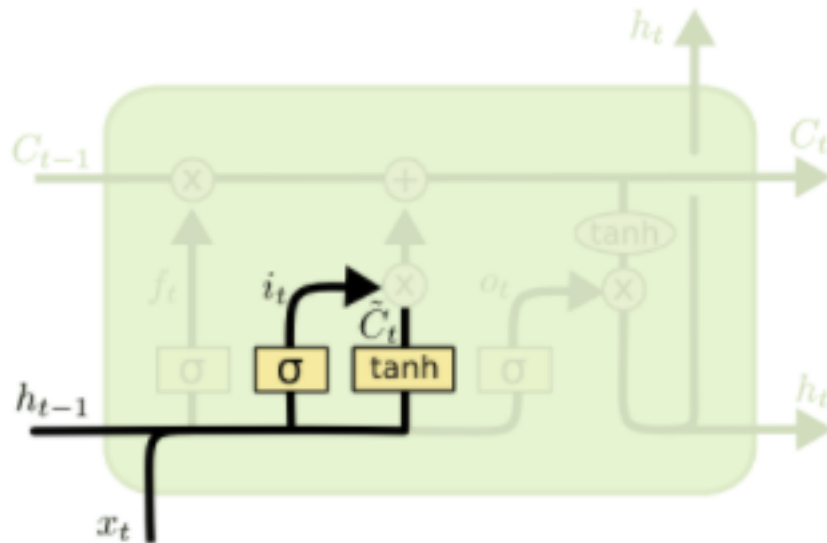
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

# LSTM: Ignore Gate $i$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_{t-1}^l \\ x_t \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

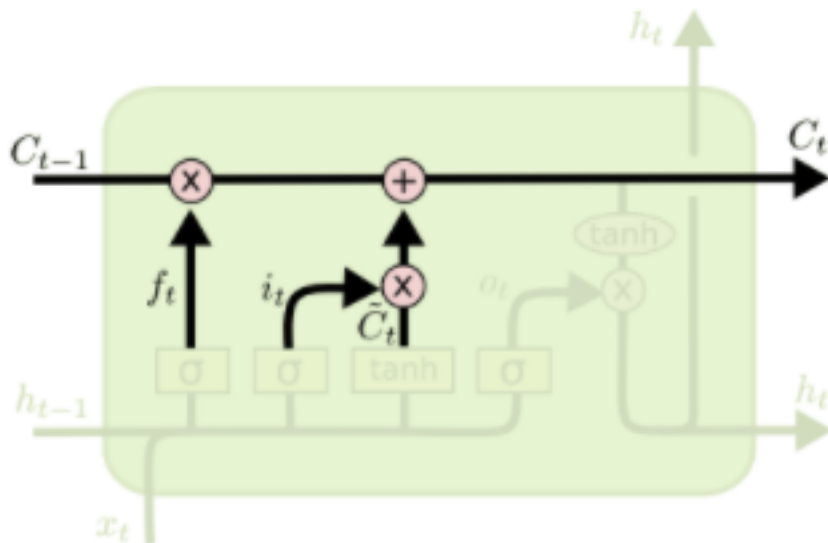
# LSTM: Block Gate $g$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



# LSTM: Output Gate $o$

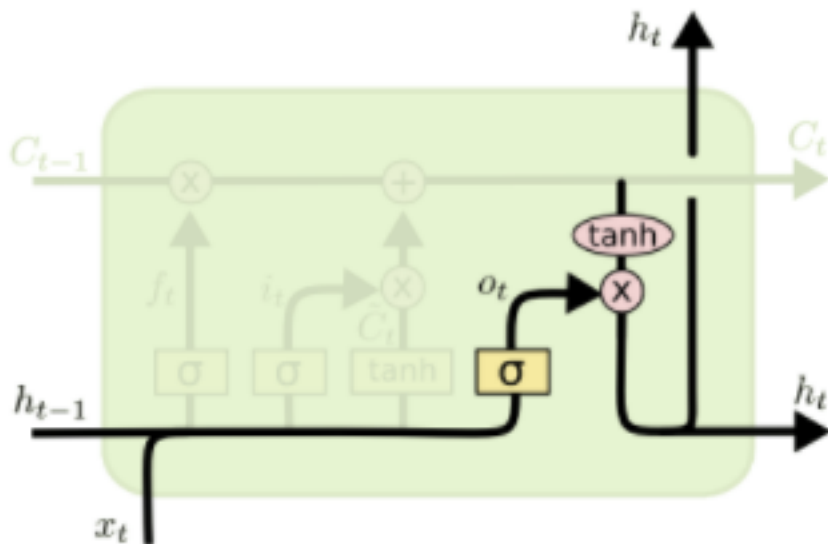
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$





# Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Additional resource for RNNs and LSTMs for Deep NLP:  
[cs224d.stanford.edu](https://cs224d.stanford.edu)