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
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

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

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]
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.

[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]

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

backprops to weights that were zero-d out by ReLu
Visualizing and Understanding Convolutional Networks
Zeiler & Fergus, 2013

**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...

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Q: can we find an image that maximizes some class score?
Optimization to Image

\[
\arg \max_I \left[ S_c(I) - \lambda \| I \|_2^2 \right]
\]

Q: can we find an image that maximizes some class score?
1. feed in zeros.

2. set the gradient of the scores vector to be [0,0,...,1,...,0], then backprop to image
Optimization to Image

1. feed in zeros.

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

3. do a small “image update”

4. forward the image through the network.

5. go back to 2.

\[
\text{arg max}_I \left[ S_c(I) - \lambda \left\| I \right\|_2^2 \right]
\]

score for class c (before Softmax)
1. Find images that maximize some class score:
1. Find images that maximize some class score:
Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

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)
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?
Find an image such that:

- Its code is similar to a given code
- It “looks natural” (image prior regularization)
DeepDream  https://github.com/google/deepdream

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
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)
DeepDream: set $dx = x$ :)
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.
DeepDream modifies the image in a way that "boosts" all activations, at any layer.
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

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

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

<table>
<thead>
<tr>
<th>(x)</th>
<th>2</th>
<th>-1</th>
<th>3</th>
<th>-2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>-4</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(w)</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n*
**Let's fool a binary linear classifier:**

\[
\begin{array}{cccccccccc}
X & 2 & -1 & 3 & -2 & 2 & 2 & 1 & -4 & 5 & 1 \\
W & -1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 \\
\end{array}
\]

- **input example**
- **weights**

**class 1 score = dot product:**

\[\text{score} = -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3\]

\[\Rightarrow \text{probability of class 1 is } \frac{1}{1+e^{\frac{-(-3)}}} = 0.0474\]

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

\[
P(y = 1 \mid x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)
\]
**Let's fool a binary linear classifier:**

<table>
<thead>
<tr>
<th>X</th>
<th>2</th>
<th>-1</th>
<th>3</th>
<th>-2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>-4</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

**class 1 score = dot product:**

\[ = -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3 \]

\[ \Rightarrow \text{probability of class 1 is } \frac{1}{1+e^{(-(-3))}} = 0.0474 \]

i.e. the classifier is **95%** certain that this is class 0 example.
**Let's fool a binary linear classifier:**

<table>
<thead>
<tr>
<th>$x$</th>
<th>2</th>
<th>-1</th>
<th>3</th>
<th>-2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>-4</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>adversarial $x$</td>
<td>1.5</td>
<td>-1.5</td>
<td>3.5</td>
<td>-2.5</td>
<td>2.5</td>
<td>1.5</td>
<td>1.5</td>
<td>-3.5</td>
<td>4.5</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Class 1 score before:

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

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

Adversarial $x$:

$$-1.5 + 1.5 + 3.5 + 2.5 + 2.5 - 1.5 + 1.5 - 3.5 - 4.5 + 1.5 = 2$$

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

i.e. we improved the class 1 probability from 5% to 88%
Let's fool a binary linear classifier:

\[
\begin{array}{cccccccccc}
X & 2 & -1 & 3 & -2 & 2 & 2 & 1 & -4 & 5 & 1 \\
W & -1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 \\
\text{adversarial } x & 1.5 & -1.5 & 3.5 & -2.5 & 2.5 & 1.5 & 1.5 & -3.5 & 4.5 & 1.5 \\
\end{array}
\]

Input example:
weights

Class 1 score before:
\[-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3\]
=> probability of class 1 is \(1/(1+e^{-(3)})\) = 0.0474

\[-1.5 + 1.5 + 3.5 + 2.5 + 2.5 - 1.5 + 1.5 - 3.5 - 4.5 + 1.5 = 2\]
=> probability of class 1 is now \(1/(1+e^{-(2)})\) = 0.88

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

0.9% bobsled + = 100% goldfish

100% Goldfish
1.0% kit fox

8.0% goldfish

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

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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Recurrent Networks offer a lot of flexibility:

- one to one
- one to many
- many to one
- 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**

E.g., **Machine Translation**

seq of words -> seq of words
Recurrent Networks offer a lot of flexibility:

- One to one
- One to many
- Many to one
- Many to many

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

---

e.g. Video classification on frame level
Recurrent Networks

Recurrent Neural Networks have loops.

* figure courtesy Chris Olah
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 $h$:

\[
\begin{align*}
  h_t &= f_W(h_{t-1}, x_t) \\
  h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
  y_t &= W_{hy}h_t
\end{align*}
\]
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”
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
min-char-rnn.py gist: 112 lines of Python

(https://gist.github.com/karpathy/d4dee566867f8291f086)
```python
###

Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
 BSD License
###

```
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }
```
# hyperparameters

```
hidden_size = 100  # size of hidden layer of neurons
seq_length = 25   # number of steps to unroll the RNN for
learning_rate = 1e-1
```

# model parameters

```
Wxh = np.random.randn(hidden_size, vocab_size)*0.01  # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01  # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01  # hidden to output
bh = np.zeros((hidden_size, 1))  # hidden bias
by = np.zeros((vocab_size, 1))   # output bias
```
Main loop

```python
n, p = 0, 0
mWxh, mwWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mbY = np.zeros_like(bh), np.zeros_like(by)  # memory variables for Adagrad
smooth_loss = -np.log(1.0 / vocab_size) * seq_length  # loss at iteration 0
while True:
    # prepare inputs (we’re sweeping from left to right in steps seq_length long)
    if p+seq_length > len(data) or n == 0:
        hprev = np.zeros((hidden_size, 1))  # reset RNN memory
        p = 0  # go from start of data
    inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
    targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print('-----%n %s
----' % (txt, ))
    # forward seq_length characters through the net and fetch gradient
    loss, dwxh, dhWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss += smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss)  # print progress
    # perform parameter update with Adagrad
    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dwxh, dhWhh, dWhy, dbh, dby],
                                [mWxh, mwWhh, mWhy, mbh, mbY]):
        mem += dparam * dparam
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8)  # adagrad update
    p += seq_length  # move data pointer
    n += 1  # iteration counter
```
n, p = 0, 0
mwkh, mwhh, mwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by)  # memory variables for Adagrad
smooth_loss = -np.log(1.0 / vocab_size) ** seq_length  # loss at iteration 0
while True:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p + seq_length + 1 >= len(data) or n == 0:
        hprev = np.zeros((hidden_size, 1))  # reset RNN memory
        p = 0  # go from start of data
        inputs = [char_to_ix[ch] for ch in data[p:p + seq_length + 1]]
        targets = [char_to_ix[ch] for ch in data[p + 1:p + seq_length + 1]]
    else:
        inputs = [char_to_ix[ch] for ch in data[p:p + seq_length + 1]]
        targets = [char_to_ix[ch] for ch in data[p + 1:p + seq_length + 1]]

    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print '%d %s
----' % (txt, )

    # forward seq_length characters through the net and fetch gradient
    loss, dwwxh, dwwhh, dwwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss += smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss)  # print progress

    # perform parameter update with Adagrad
    for param, dparam, mem in zip([wkh, whh, why, bh, by],
                                   [dwwxh, dwwhh, dwwhy, dbh, dby],
                                   [mwkh, mwhh, mwhy, mbh, mby]):
        mem += dparam * dparam
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8)  # adagrad update

    p += seq_length  # move data pointer
    n += 1  # iteration counter

Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n comp150dl
```python
n, p = 0, 0
Wxh, Whh, Why = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
bh, bby = np.zeros_like(bh), np.zeros_like(bby)  # memory variables for Adagrad
smooth_loss = -np.log(1.0 / vocab_size)**seq_length  # loss at iteration 0
while True:
    # prepare inputs (we’re sweeping from left to right in steps seq_length long)
    if p+seq_length+1 >= len(data) or n == 0:
        hprev = np.zeros((hidden_size, 1))  # reset RNN memory
        p = 0  # go from start of data
        inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
        targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print('-----

%.2f%%
-----' % (txt, ))

    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy, dbh, dbby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss = smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print('iter %d, loss: %f' % (n, smooth_loss))  # print progress

    # perform parameter update with Adagrad
    for param, dparam, mem in zip([Wxh, Whh, Why, bh, bby],
                           [dWxh, dWhh, dWhy, dbh, dbby],
                           [mwxh, mwhh, mwhy, mbh, mbyb]):
        mem += dparam ** 2
        param -= learning_rate * dparam / np.sqrt(mem + 1e-8)  # adagrad update

    p += seq_length  # # iteration counter
    n += 1
```

Andrei Karpathy
Main loop

```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p+seq_length >= len(data) or n == 0:
        hprev = np.zeros((hidden_size, 1)) # reset RNN memory
        p = 0 # go from start of data
    inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
    targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        sample_txt = ''.join([ix_to_char[ix] for ix in sample_ix])
        print('...\n
    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss = smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

    # perform parameter update with Adagrad
    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                   [dWxh, dWhh, dWhy, dbh, dby],
                                   [mWxh, mWhh, mWhy, mbh, mby]):
        mem += dparam * dparam
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
    p += seq_length # move data pointer
    n += 1 # iteration counter
```
Main loop

```python
n, p = 0, 0
mWxh, mwWh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p+seq_length-1 >= len(data) or n == 0:
        hprev = np.zeros((hidden_size,1)) # reset RNN memory
        p = 0 # go from start of data
        inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
        targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print '--- %s ---' % (txt, )
    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss = smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                   [dWxh, dWhh, dWhy, dbh, dby],
                                   [mWxh, mwWh, mWhy, mh, mby]):
        mem += dparam * dparam
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
    p += seq_length # move data pointer
    n += 1 # iteration counter
```

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n comp150dl

n Andrej Karpathy

min-char-rnn.py gist
Loss function
- forward pass (compute loss)
- backward pass (compute param gradient)
def lossFun(inputs, targets, hprev):
    
    inputs, targets are both list of integers.
    hprev is Hx1 array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state
    
    xs, hs, ys, ps = [], [], [], []
    hs[-1] = np.copy(hprev)
    loss = 0
    
    # forward pass
    for t in xrange(len(inputs)):
        xs[t] = np.zeros((vocab_size, 1))  # encode in 1-of-k representation
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh)  # hidden state
        ys[t] = np.dot(Why, hs[t])  # unnormalized log probabilities for next chars
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))  # probabilities for next chars
        loss += -np.log(ps[t][targets[t], 0])  # softmax (cross-entropy loss)

    return loss, ds, dxh, dWhh, dWhy, hprev

# backwards pass
for t in xrange(len(inputs) - 1, 0, -1):
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t]))  # probabilities for next chars
    ds[t] = np.dot(Why.T, ps[t])
    ds[t] -= np.dot(Why.T, ps[t][targets[t], np.newaxis])
    ds[t] *= (1 - np.power(np.tanh(hs[t]), 2))
    dWxh += np.dot(xs[t].T, ds[t])
    dWhh += np.dot(hs[t-1].T, ds[t])
    dWhy += np.dot(hs[t].T, ds[t])

# initial hidden state
hs[0] = np.zeros(1, H)
# backward pass: compute gradients going backwards

dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
dbh, dby = np.zeros_like(bh), np.zeros_like(by)
dhnext = np.zeros_like(hs[0])

for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dwhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)

for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients

return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```python
def sample(h, seed_ix, n):
    """
    sample a sequence of integers from the model
    h is memory state, seed_ix is seed letter for first time step
    """
    x = np.zeros((vocab_size, 1))
    x[seed_ix] = 1
    ixes = []
    for t in xrange(n):
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
        y = np.dot(Why, h) + by
        p = np.exp(y) / np.sum(np.exp(y))
        ix = np.random.choice(range(vocab_size), p=p.ravel())
        x = np.zeros((vocab_size, 1))
        x[ix] = 1
        ixes.append(ix)
    return ixes
```
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:

train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorton 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 oftened 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 sain 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 nues 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 aps, 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

Latex source
Lemma 0.1. Assume (3) and (4) by the construction in the description.
Suppose $X = \lim X$ by the formal open covering $X$ and a single map $	ext{Proj}_X(A) = \text{Spec}(B)$ over $U$ compatible with the complex

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

When in this case of to show that $\mathcal{Q} \rightarrow \mathcal{C}_{\mathcal{L}}$ 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 etale. 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 $S = \text{Spec}(R)$ and $Y = \text{Spec}(S)$.

Proof. This is form all sheaves of sheaves on $X$. But given a scheme $U$ and a surjective étale morphism $U \to X$. Let $U \cap U' = \bigcap_i U_i$ the scheme $X$ over $S$ at the schemes $X_i \to X$ and $U = \lim X_i$.

The following lemma surjective restrecombines of this implies that $\mathcal{F}_{X_0} = \mathcal{F}_{X_1} = \mathcal{F}_{X_2, \ldots}$. $\mathcal{F}_{X_0}$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_{X_0}$-modules on $C$ as a $\mathcal{F}$-module.

In particular $\mathcal{F} = \mathcal{F} / \mathcal{F}$ we have to show that

$$\overline{\mathcal{M}}^* = \mathcal{T}^* \otimes_{\text{Spec}(k)} \mathcal{O}_{S, \mathcal{O}} = \mathcal{F}_0^* \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = \{\text{Sch(S)}^{\mathcal{F}}_{\mathcal{F}^*_{\mathcal{F}}}, \mathcal{O}_{\mathcal{S}, \mathcal{O}}\}$$

and

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

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.

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

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

$$\mathcal{O}_{\mathcal{C}} = \mathcal{O}_{\mathcal{F}}(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\text{etale}}$ 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 $Z$ 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 $U \subseteq X$ 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

$$\begin{array}{c}
S \\
\downarrow \\
\xi \\
\downarrow \\
\alpha' \\
\downarrow \\
\alpha \\
\downarrow \\
X
\end{array}$$

is a limit. Then $\mathcal{G}$ a finite type and assume $S$ is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $L$. 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"

$$\mathcal{O}_{X, x} \rightarrow \mathcal{F}_{x} \rightarrow \mathcal{O}_{X, x} \rightarrow \mathcal{O}_{X, x}^{-1} \mathcal{O}_{X, x} \rightarrow \mathcal{O}_{X, x}^{-1} \mathcal{O}_{X, x}$$

is an isomorphism of covering of $\mathcal{O}_{X}$. 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 $\mathcal{O}_{X}$, is a closed immersion, see Lemma ??.

This is a sequence of $\mathcal{F}$ is a similar morphism.
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
- arch
- block
- crypto
- drivers
- firmware
- fs
- include
- init
- iso

- Merge git://git.kernel.org/pub/svcs/linux/kernel/git/hab/target-pending
- Merge branch 'x86-urgent-for-linux' of git://git.kernel.org/pub/svcs/l...
- block: discard bdi_unregister() in favour of bdi_destroy()
- Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6
- Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
- firmware/hex2fw.c: restore missing default in switch statement
- vifs: read file_handle only once in handle_to_path
- Merge branch 'perf-urgent-for-linux' of git://git.kernel.org/pub/scm/...
- init: fix regression by supporting devices with major:minor:offset fo...
- Merge branch 'linux' of git://git.kernel.org/pub/scm/linux/kernel

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

comp150dl Tufts University
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 << 1))
            pipe = (in_use & UMXTTHREAD_UNCCA) +
                ((count & 0x00000000fffff8) & 0x000000f) << 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/kernvod.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multio.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setev.h>
#include <asm/pgproto.h>
```c
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setev.h>
#include <asm/pgproto.h>

#define REG_PG  vesa_slot_addr_pack
#define PPM_NOCOMP  APSR(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));

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)-l->lr_full; low;
```
Recommended Reading:
Visualizing and Understanding Recurrent Networks

/* Unpack a filter field's string representation from user-space */
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]
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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
test image
* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
before:
\[ h = \tanh(W_{xh} \ast x + W_{hh} \ast h) \]

now:
\[ h = \tanh(W_{xh} \ast x + W_{hh} \ast h + W_{ih} \ast v) \]
`test image`

`sample!`
* (ar

wed from Andrej Karpathy
rd cs231n

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

<START>

h0

x0

<START>

straw

Comp150dl

h1

hat

y0

y1

test image

sample!
test image
test image

<START>

sample

<END> token

=> finish.
Image Sentence Datasets

Microsoft COCO

[Tsung-Yi Lin et al. 2014]

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.”

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

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
ION: INSIDE-OUTSIDE NET

conv1 conv2 conv3 conv4 conv5 conv7

ROI Pooling

L2 normalize
concat
scale

1x1 conv
fc
fc
fc

softmax bbox

For each ROI

4-dir RNN 4-dir RNN

context features

* slide courtesy Sean Bell

Base ConvNet: VGG16 [Simonyan 2014]
Limitations of RNNs

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

* figures courtesy Chris Olah
Long Short Term Memory Networks

* figures courtesy Chris Olah
RNN:

\[ h^l_t = \tanh W^l \left( h^{l-1}_t \right) \]

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

LSTM:

\begin{pmatrix}
  i \\
  f \\
  o \\
  g
\end{pmatrix} =
\begin{pmatrix}
  \text{sigm} \\
  \text{sigm} \\
  \text{sigm} \\
  \tanh
\end{pmatrix}
W^l \left( h^{l-1}_t \right)

\[ c^l_t = f \odot c_{t-1}^l + i \odot g \]

\[ h^l_t = o \odot \tanh(c^l_t) \]

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
LSTM: Cell State
long running memory of the network

* figures courtesy Chris Olah
LSTM: Forget Gate $f$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h^{l-1}_t \\ h^l_t \\ h^{l-1}_{t-1} \end{pmatrix}$$

$$c^l_t = f \odot c^l_{t-1} + i \odot g$$

$$h^l_t = o \odot \text{tanh}(c^l_t)$$

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

* figures courtesy Chris Olah
LSTM: Ignore Gate $i$

\[
\begin{align*}
\begin{pmatrix}
  i \\
  f \\
  o \\
  g
\end{pmatrix} &= \begin{pmatrix}
  \text{sigm} \\
  \text{sigm} \\
  \text{sigm} \\
  \text{tanh}
\end{pmatrix} W^l \begin{pmatrix}
  h_{t-1}^l \\
  h_t^l
\end{pmatrix} \\
\end{align*}
\]

\[
\begin{align*}
  c_t^l &= f \odot c_{t-1}^l + i \odot g \\
  h_t^l &= o \odot \tanh(c_t^l)
\end{align*}
\]

\[
\begin{align*}
  i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
  \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)
\end{align*}
\]

* figures courtesy Chris Olah
LSTM: Block Gate $g$

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

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

\[
h_t^l = o \odot \tanh(c_t^l)
\]

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

* figures courtesy Chris Olah
LSTM: Output Gate $o$

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

$$c^l_t = f \odot c^l_{t-1} + i \odot g$$

$$h^l_t = o \odot \tanh(c^l_t)$$

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

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

* figures courtesy Chris Olah
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