Lecture 4: Backpropagation and Neural Networks (part 1)

Tuesday January 31, 2017
Announcements!

- If you are adversely affected by immigration ban, please talk to me about accommodations

- Send in paper choices by **tonight**

- Should be able to run Jupyter server on Tufts was and network machines now
  -  (deep-venv)> pip install --upgrade jupyter

- hw1 deadline in two days — Thurs Feb 2: Don’t forget to read the course notes.

- Redo calculation of dL/dW for hinge loss
Python/Numpy of the Day

- `y_pred = scores.argmax(axis=1)`
- `inds = np.random.choice(X.shape[0], batch_size)`
  - randomly select N numbers in a range,
  - useful for subsampling
- `[:, np.newaxis]`
  - reshapes matrices of size (N,) to size (N,1)
Where we are...

\[ s = f(x; W) = Wx \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_{k} W_k^2 \]

scores function

SVM loss

data loss + regularization

want \( \nabla_W L \)
Optimization

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad  # perform parameter update
```

(image credits to Alec Radford)
Gradient Descent

\[ \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

Numerical gradient: slow :, approximate :, easy to write :
Analytic gradient: fast :), exact :), error-prone :

In practice: Derive analytic gradient, check your implementation with numerical gradient

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Hinge Loss Gradient wrt Weights $W$

$$L_i = \sum_{j \neq y_i} \left[ \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

- We want the Jacobian Matrix of all gradients
  - partial derivatives of all output dimensions by all input dimensions

For all rows of $dW$ where the row corresponds to the GT value for that training instance, i.e. $j = y_i$

$$\nabla_{w_{y_i}} L_i = -\left( \sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right)x_i$$

For all rows of $dW$ where $j \neq y_i$

$$\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)x_i$$

http://cs231n.github.io/optimization-1/#analytic
Softmax Loss Gradient wrt Score $S$

\[ a_j = w_j^T x_j \]

\[ S_j = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \quad \forall j \in 1..N \]

\[ \frac{\partial S_i}{\partial a_j} = \frac{\partial}{\partial a_j} \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} = \frac{e^{a_i} \sum - e^{a_j} e^{a_i}}{\sum^2} = \frac{e^{a_i} \sum - e^{a_j}}{\sum} = S_i(1 - S_j) \]

\[ \nabla a_j S_i, \text{ when } i = j \]

\[ \nabla a_j S_i, \text{ when } i \neq j \]

\[ \frac{\partial}{\partial a_j} \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} = \frac{0 - e^{a_j} e^{a_i}}{\sum^2} = -S_j S_i \]

\[ \nabla a_j S_i = S_i (I(i = j) - S_j) \]

* note change of subscripts from last slide

Skipping some steps for space, please see original notes.

[eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/](eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/)
Softmax Loss Gradient wrt Score $S$

\[ a_j = w^T_j x_j \]

\[ S_j = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \quad \forall j \in 1..N \]

\[ \nabla a_j S_i = S_i (1(i = j) - S_j) \]

\[ \nabla S_i L = \frac{\partial}{\partial S_i} - \log(S_i) = S_j - 1(i = j) \]

\[ \nabla W_j L = \frac{\partial L}{\partial S_i} \ast \frac{\partial S_i}{\partial W_j} = (S_j - 1(i = j)) x_i \]

Skipping some steps for space, please see original notes.
**Computational Graph**

\[ f = W x \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]

\[ R(W) \]

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n*
Convolutional Network (AlexNet)

input image
weights
loss

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \(x = -2, y = 5, z = -4\)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

*Example: x = -2, y = 5, z = -4*

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]
\[ \text{e.g. } x = -2, y = 5, z = -4 \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \]

Want:

\[ \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want:

\[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]
activations

\[ f \]

\[ x \]

\[ y \]

\[ z \]
activations

\[
\frac{\partial z}{\partial x} \quad \frac{\partial z}{\partial y}
\]

"local gradient"
activations

\[ \frac{\partial z}{\partial x} \]

\[ \frac{\partial z}{\partial y} \]

"local gradient"

\[ \frac{\partial L}{\partial z} \]

ggradients

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

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

"local gradient"

\[ \frac{\partial z}{\partial x} \]

\[ \frac{\partial z}{\partial y} \]

\[ \frac{\partial L}{\partial z} \]

gradients

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

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]

“local gradient”

\[ \frac{\partial L}{\partial z} \]

gradients

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

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]

"local gradient"

gradients
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]

\[ \left( -\frac{1}{1.37^2} \right)(1.00) = -0.53 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ (1)(-0.53) = -0.53 \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
f(x) &= e^x \\
f_a(x) &= ax \\
f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
\frac{df}{dx} &= e^x \\
\frac{df}{dx} &= a \\
\frac{df}{dx} &= 1
\end{align*}
\]

\[
\begin{align*}
f(x) &= \frac{1}{x} \\
f(x) &= \frac{1}{x^2}
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

\[(e^{-1})(-0.53) = -0.20\]

- \[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]
- \[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]
- \[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2 \]
- \[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]

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

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
f(x) &= e^x & \rightarrow & & \frac{df}{dx} &= e^x \\
f_a(x) &= ax & \rightarrow & & \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
f(x) &= \frac{1}{x} & \rightarrow & & \frac{df}{dx} &= -\frac{1}{x^2} \\
f_c(x) &= c + x & \rightarrow & & \frac{df}{dx} &= 1
\end{align*}
\]

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

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

\((-1) \times (-0.20) = 0.20\)

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ [\text{local gradient}] \times [\text{its gradient}] \]

\[ [1] \times [0.2] = 0.2 \]

\[ [1] \times [0.2] = 0.2 \quad (\text{both inputs!}) \]
Another example:

\[
f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}
\]

\[
f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x
\]

\[
f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a
\]

\[
f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}
\]

\[
f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

[local gradient] x [its gradient]

x0: \([2] \times [0.2] = 0.4\)  
w0: \([-1] \times [0.2] = -0.2\)
The sigmoid function is defined as:

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

The derivative of the sigmoid function is:

\[ \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x) \]
\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

\[ \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x) \]

sigmoid function

(0.73) * (1 - 0.73) = 0.2
Patterns in backward flow

**add** gate: gradient distributor
**max** gate: gradient router
**mul** gate: gradient… “switcher”?

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Gradients add at branches
Implementation: forward/backward API

Graph (or Net) object. (Rough psuedo code)

```python
class ComputationalGraph(object):
    # ...

    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss

    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```
Implementation: forward/backward API

(x, y, z are scalars)
Implementation: forward/backward API

class MultiplyGate(object):
    def forward(x, y):
        z = x * y
        self.x = x  # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz  # [dz/dx * dL/dz]
        dy = self.x * dz  # [dz/dy * dL/dz]
        return [dx, dy]

(x, y, z are scalars)
Example: Torch Layers

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Example: Torch Layers
Example: Torch MulConstant

\[ f(X) = aX \]

- **Initialization**
- **forward()**
- **backward()**
Example: Caffe Layers
Caffe Sigmoid Layer

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

*(top_diff) (chain rule)*
#include <vector>
#include <random>
#include "iterator"
#include "caffe/layers/shuffle_layer.hpp"
#include "caffe/util/math_functions.hpp"

namespace caffe {

template <typename Dtype>
void ShuffleLayer<Dtype>::LayerSetUp(const vector<Blob<Dtype> *> & bottom,
                                      const vector<Blob<Dtype> *> & top) {
  const int count = top[0]->num();
  std::cout << bottom[0]->shape(0) << std::endl;
  std::cout << bottom[0]->shape(1) << std::endl;
  vector<int> orig_shape = bottom[0]->shape();
  vector<int> new_shape = new_shape.push_back(count);
  new_shape.push_back(bottom[0]->shape(1));
  top[0]->reshape(new_shape);
  bottom[0]->reshape(orig_shape);
  Dtype* bottom_data = bottom[0]->mutable_cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int shuffle_order = shuffle_order_.cpu_data();
  bool forward = true;
  Shuffle(bottom_data, top_data, batch_item_size_, forward, shuffle_order,
          count);
  top[0]->reshape(orig_shape);
  bottom[0]->reshape(orig_shape);
}

}  // namespace caffe

// namespace caffe
Gradients for vectorized code

This is now the Jacobian matrix (derivative of each element of \(z\) w.r.t. each element of \(x\))

(x,y,z are now vectors)

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

\[ f(x) = \max(0, x) \]  
\text{(elementwise)}

4096-d input vector \quad \Rightarrow \quad 4096-d output vector
Vectorized operations

\[ f(x) = \max(0,x) \] (elementwise)

Q: what is the size of the Jacobian matrix?

Jacobian matrix

4096-d input vector

\[ \frac{\partial L}{\partial x} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial L}{\partial f_1} \\ \cdot & \cdot \\ \frac{\partial f}{\partial x_n} & \frac{\partial L}{\partial f_n} \end{bmatrix} \]

4096-d output vector
Vectorized operations

f(x) = max(0, x) (elementwise)

4096-d input vector

4096-d output vector

Q: what is the size of the Jacobian matrix? [4096 x 4096!]

Q2: what does it look like?

\[
\frac{\partial L}{\partial x} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial L}{\partial f} \end{bmatrix}
\]

Jacobian matrix

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

in practice we process an entire minibatch (e.g. 100) of examples at one time:

\[
f(x) = \max(0,x) \quad (elementwise)
\]

i.e. Jacobian would technically be a \([409,600 \times 409,600]\) matrix.
Assignment: Writing SVM/Softmax
Stage your forward/backward computation!

E.g. for the SVM:

```python
# receive W (weights), X (data)
# forward pass (we have 8 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```
Summary so far

- neural nets will be very large: no hope of writing down gradient formula by hand for all parameters
- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates
- implementations maintain a graph structure, where the nodes implement the `forward()` / `backward()`.
- **forward**: compute result of an operation and save any intermediates needed for gradient computation in memory
- **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs.
Neural Network so far:

(Before) Linear score function: \[ f = Wx \]
Neural Network so far:

(Before) Linear score function:

\[ f = Wx \]

(Now) 2-layer Neural Network

\[ f = W_2 \max(0, W_1x) \]
Neural Network so far:

(Before) Linear score function:

$$f = Wx$$

(Now) 2-layer Neural Network

$$f = W_2 \max(0, W_1x)$$

Neural Network so far:
Neural Network so far:

(Before) Linear score function:

\[ f = W x \]

(Now) 2-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]
Neural Network so far:

(Before) Linear score function:

$$f = Wx$$

(Now) 2-layer Neural Network or 3-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$
Full implementation of training a 2-layer Neural Network needs ~11 lines:

```
X = np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
y = np.array([[0,1,1,0]]).T
syn0 = 2*np.random.random((3,4)) - 1
syn1 = 2*np.random.random((4,1)) - 1
for j in xrange(60000):
    l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
    l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
    l2_delta = (y - l2)*(l2*(1-l2))
    l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
    syn1 += l1.T.dot(l2_delta)
    syn0 += X.T.dot(l1_delta)
```

from @iamtrask, http://iamtrask.github.io/2015/07/12/basic-python-network/
Assignment: Writing 2layer Net
Stage your forward/backward computation!

```python
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
hi = #... function of X,W1,b1
scores = #... function of hi,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dhi,dW2,db2 = #...
dW1,db1 = #...
```
sigmoid activation function

\[ f(x) = \frac{1}{1 + e^{-x}} \]
impulses carried toward cell body

impulses carried away from cell body

dendrites

branches of axon

axon terminals

axon

nucleus

cell body

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n
Be very careful with your Brain analogies:

**Biological Neurons:**
- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Rate code may not be adequate

[Dendritic Computation. London and Hausser]
Activation Functions

Sigmoid

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

\( \text{tanh} \quad \text{tanh}(x) \)

ReLU

\( \text{max}(0, x) \)

Leaky ReLU

\( \text{max}(0.1x, x) \)

Maxout

\( \text{max}(w_1^T x + b_1, w_2^T x + b_2) \)

ELU

\[
\begin{cases} 
  x & \text{if } x > 0 \\
  \alpha (\exp(x) - 1) & \text{if } x \leq 0 
\end{cases}
\]
Neural Networks: Architectures

“2-layer Neural Net”, or “1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or “2-hidden-layer Neural Net”
Example Feed-forward computation of a Neural Network

```python
class Neuron:
    # ...
    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum))  # sigmoid activation function
        return firing_rate
```

We can efficiently evaluate an entire layer of neurons.
Example Feed-forward computation of a Neural Network

# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
Setting the number of layers and their sizes

more neurons = more capacity
Do not use size of neural network as a regularizer. Use stronger regularization instead:

(you can play with this demo over at ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

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

- we arrange neurons into fully-connected layers
- the abstraction of a layer has the nice property that it allows us to use efficient vectorized code (e.g. matrix multiplies)
- neural networks are not really neural
- neural networks: bigger = better (but might have to regularize more strongly)
Next Lecture:

More than you ever wanted to know about Neural Networks and how to train them.