Lecture 12:
- Computational Graph
- Backpropagation
Last time…

**Multilayer Perceptron**

- **Layer Representation**
  \[
  y_i = W_i x_i \\
  x_{i+1} = \sigma(y_i)
  \]

- (typically) iterate between linear mapping $Wx$ and nonlinear function

- **Loss function** $l(y, y_i)$ to measure quality of estimate so far
Last time... Forward Pass

- Output of the network can be written as:

\[ h_j(x) = f(v_{j0} + \sum_{i=1}^{D} x_i v_{ji}) \]

\[ o_k(x) = g(w_{k0} + \sum_{j=1}^{J} h_j(x) w_{kj}) \]

(j indexing hidden units, k indexing the output units, D number of inputs)

- Activation functions \( f, g \): sigmoid/logistic, tanh, or rectified linear (ReLU)

\[ \sigma(z) = \frac{1}{1 + \exp(-z)}, \quad \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}, \quad \text{ReLU}(z) = \max(0, z) \]
Last time... **Forward Pass in Python**

- Example code for a forward pass for a 3-layer network in Python:

  ```python
  # 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)
  ```

- Can be implemented efficiently using matrix operations
- Example above: $W_1$ is matrix of size $4 \times 3$, $W_2$ is $4 \times 4$. What about biases and $W_3$?

[http://cs231n.github.io/neural-networks-1/]
Backpropagation
Recap: Loss function/Optimization

<table>
<thead>
<tr>
<th>Object</th>
<th>Score 1</th>
<th>Score 2</th>
<th>Score 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
<td>-0.51</td>
<td>3.42</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
<td>6.04</td>
<td>4.64</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
<td>5.31</td>
<td>2.65</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
<td>-4.22</td>
<td>5.1</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
<td>-4.19</td>
<td>2.64</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
<td>3.58</td>
<td>5.55</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
<td>4.49</td>
<td>-4.34</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
<td>-4.37</td>
<td>-1.5</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
<td>-2.09</td>
<td>-4.79</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
<td>-2.93</td>
<td>6.14</td>
</tr>
</tbody>
</table>

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. *(optimization)*

We defined a (linear) **score function**:

\[ f(x_i, W, b) = W x_i + b \]
# Softmax Classifier (Multinomial Logistic Regression)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[ s = f(x_i; W) \]

cat \hspace{1cm} 3.2

car \hspace{1cm} 5.1

frog \hspace{1cm} -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[
P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

where \( s = f(x_i; W) \)

cat \quad 3.2

car \quad 5.1

frog \quad -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

where \( s = f(x_i; W) \)

- cat: 3.2
- car: 5.1
- frog: -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[
P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

where

\[
s = f(x_i; W)
\]

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

\[
L_i = -\log P(Y = y_i | X = x_i)
\]

cat 3.2
car 5.1
frog -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[
P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

where

\[
s = f(x_i; W)
\]

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

\[
L_i = - \log P(Y = y_i | X = x_i)
\]

in summary:

\[
L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right)
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td></td>
</tr>
</tbody>
</table>
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log \left( \frac{e^{sy_i}}{\sum_j e^{sj}} \right) \]

<table>
<thead>
<tr>
<th>cat</th>
<th>3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

unnormalized log probabilities
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log \left( \frac{e^{s_{yi}}}{\sum_j e^{s_j}} \right) \]

unnormalized probabilities

\[
\begin{array}{c|c|c}
\text{cat} & 3.2 & 24.5 \\
\text{car} & 5.1 & 164.0 \\
\text{frog} & -1.7 & 0.18 \\
\end{array}
\]

unnormalized log probabilities
Softmax Classifier (Multinomial Logistic Regression)

\[
L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right)
\]

unnormalized probabilities

<table>
<thead>
<tr>
<th>Unnormalized log probabilities</th>
<th>Exp</th>
<th>Normalized probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat 3.2</td>
<td>24.5</td>
<td>0.13</td>
</tr>
<tr>
<td>car 5.1</td>
<td>164.0</td>
<td>0.87</td>
</tr>
<tr>
<td>frog -1.7</td>
<td>0.18</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log \left( \frac{e^{s y_i}}{\sum_j e^{s_j}} \right) \]

unnormalized probabilities

<table>
<thead>
<tr>
<th>cat</th>
<th>3.2</th>
<th>24.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.0</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
</tr>
</tbody>
</table>

unnormalized log probabilities

L_i = -log(0.13) = 0.89

probabilities

0.13
0.87
0.00
Optimization
Gradient Descent

# Vanilla Gradient Descent

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

In 1-dimension, the derivative of a function:

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

In multiple dimensions, the gradient is the vector of (partial derivatives).
Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient

```python
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256)  # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad  # perform parameter update
```
Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient

```python
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256)  # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad  # perform parameter update
```

there are also more fancy update formulas (momentum, Adagrad, RMSProp, Adam, …)
The effects of different update form formulas
Example of optimization progress while training a neural network.

(Loss over mini-batches goes down over time.)
The effects of step size (or “learning rate”)

- **High learning rate**: leads to overshooting and may cause the model to diverge.
- **Low learning rate**: convergence is slow.
- **Good learning rate**: achieves a good balance between convergence speed and accuracy.

The plot shows the loss over epochs for different learning rates, demonstrating the impact on the convergence of the model.
Computational Graph

\[ f = Wx \]

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

input image
weights
loss
Neural Turing Machine

input tape

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

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

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

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

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

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

\[ \text{e.g. } x = -2, \ y = 5, \ z = -4 \]

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

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

Want:

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

Chain rule:

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

\[
\begin{array}{c}
x \quad -2 \\
\downarrow \\
-4
\end{array}
\]

\[
\begin{array}{c}
y \quad 5 \\
\downarrow \\
-4
\end{array}
\]

\[
\begin{array}{c}
z \quad -4 \\
\downarrow \\
3
\end{array}
\]

\[
\begin{array}{c}
+ \quad 3 \\
\downarrow \\
-4
\end{array}
\]

\[
\begin{array}{c}
* \\
\downarrow \\
1
\end{array}
\]

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

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

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

"local gradient"
activations

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

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

gradients

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

"local gradient"
\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

activations

"local gradient"

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

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

gradients

f

\[
\frac{\partial L}{\partial y}
\]
activations

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

"local gradient"

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

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

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

gradients
activations

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

“local gradient”

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

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

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

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

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

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

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

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

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

\[
\begin{align*}
  f(x) &= \frac{1}{x} \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= -\frac{1}{x^2} \\
  \frac{df}{dx} &= 1
\end{align*}
\]
Another example: 

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

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

\[
\begin{align*}
\frac{d f}{d x} &= e^x \\
\frac{d f}{d x} &= a \\
\frac{d f}{d x} &= -1/x^2 \\
\frac{d f}{d x} &= 1
\end{align*}
\]
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:

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

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

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

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

\[ (1)(-0.53) = -0.53 \]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

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

\[
\begin{align*}
  \frac{df}{dx} &= e^x \\
  \frac{df}{dx} &= a \\
  \frac{df}{dx} &= 1
\end{align*}
\]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

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

\[
\begin{align*}
\frac{df}{dx} &= e^x \\
\frac{df}{dx} &= \frac{1}{x} \\
\frac{df}{dx} &= -\frac{1}{x^2} \\
\frac{df}{dx} &= 1
\end{align*}
\]
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: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
  (-1) \times (-0.20) &= 0.20 \\
  f(x) &= e^x \\
  f_a(x) &= ax \\
  f_c(x) &= c + x
\end{align*}
\]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
f(x) &= e^x & \rightarrow & & \frac{df}{dx} &= e^x \\
f_a(x) &= ax & \rightarrow & & \frac{df}{dx} &= a \\
f_c(x) &= c + x & \rightarrow & & \frac{df}{dx} &= 1
\end{align*}
\]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

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

\[
\begin{align*}
[\text{local gradient}] \times [\text{its gradient}] \\
[1] \times [0.2] &= 0.2 \\
[1] \times [0.2] &= 0.2 \text{ (both inputs!)}
\end{align*}
\]
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:

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

[local gradient] x [its gradient]

\[ x_0: [2] \times [0.2] = 0.4 \]
\[ w_0: [-1] \times [0.2] = -0.2 \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]
\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]
\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
\[
f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}
\]

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

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

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

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

\((0.73) \times (1 - 0.73) = 0.2\)
Patterns in backward flow

- **add** gate: gradient distributor
- **max** gate: gradient router
- **mul** gate: gradient… “switcher”?
Gradients add at branches
Implementation: forward/backward API

Graph (or Net) object. *(Rough pseudo 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

(x, y, z are scalars)

```python
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]
```

by Fei-Fei Li & Andrej Karpathy & Justin Johnson
Summary

• 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()` API.

• **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.
Where are we now...

Mini-batch SGD

Loop:
1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient
Next Lecture:

Convolutional Neural Networks