Optimization and Backpropagation
Lecture 3 Recap
Neural Network

• Linear score function $f = Wx$
Neural Network

• Linear score function $f = Wx$

• Neural network is a nesting of ‘functions’
  - 2-layers: $f = W_2 \max(0, W_1x)$
  - 3-layers: $f = W_3 \max(0, W_2 \max(0, W_1x))$
  - 4-layers: $f = W_4 \tanh(W_3, \max(0, W_2 \max(0, W_1x)))$
  - 5-layers: $f = W_5 \sigma(W_4 \tanh(W_3, \max(0, W_2 \max(0, W_1x))))$
  - ... up to hundreds of layers
Neural Network

Input layer

Hidden layer

Output layer

Credit: Li/Karpathy/Johnson
Activation Functions

Sigmoid: \( \sigma(x) = \frac{1}{1 + e^{-x}} \)

\[ \text{Tanh}: \tanh(x) \]

ReLU: \( \max(0, x) \)

Leaky ReLU: \( \max(0.1x, x) \)

Maxout \( \max(w_1^T x + b_1, w_2^T x + b_2) \)

ELU \( f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \)
Loss Functions

• Measure the goodness of the predictions (or equivalently, the network's performance)

• Regression loss
  – L1 loss \( L(y, \hat{y}; \theta) = \frac{1}{n} \sum_{i}^{n} |y_i - \hat{y}_i|_1 \)
  – MSE loss \( L(y, \hat{y}; \theta) = \frac{1}{n} \sum_{i}^{n} |y_i - \hat{y}_i|_2^2 \)

• Classification loss (for multi-class classification)
  – Cross Entropy loss \( E(y, \hat{y}; \theta) = - \sum_{i=1}^{n} \sum_{k=1}^{k} (y_{ik} \cdot \log \hat{y}_{ik}) \)
Computational Graphs

• Neural network is a computational graph
  – It has compute nodes
  – It has edges that connect nodes
  – It is directional
  – It is organized in ‘layers’
Backprop
The Importance of Gradients

• Our optimization schemes are based on computing gradients
  \[ \nabla_{\theta} L(\theta) \]

• One can compute gradients analytically but what if our function is too complex?

• Break down gradient computation

Backpropagation

Rumelhart 1986
Backprop: Forward Pass

- $f(x, y, z) = (x + y) \cdot z$

Initialization $x = 1, y = -3, z = 4$

$\begin{align*}
1 & \rightarrow x \\
-3 & \rightarrow y \\
4 & \rightarrow z \\
\end{align*}$

$\begin{align*}
\text{sum} & \rightarrow + \\
\text{mult} & \rightarrow \times \\
f & = -8 \\
d & = -2
\end{align*}$
\[ f(x, y, z) = (x + y) \cdot z \]

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?
Backprop: Backward Pass

\[ f(x, y, z) = (x + y) \cdot z \]

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

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

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

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

with \( x = 1, y = -3, z = 4 \)

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

\[
f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d
\]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?
Backprop: Backward Pass

\[ f(x, y, z) = (x + y) \cdot z \]

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?

Chain Rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial d} \cdot \frac{\partial d}{\partial y} \]

\[ \frac{\partial f}{\partial y} = 4 \cdot 1 = 4 \]
Backprop: Backward Pass

\[ f(x, y, z) = (x + y) \cdot z \]
with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?

**Chain Rule:**

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial d} \cdot \frac{\partial d}{\partial x} \]

\[ \frac{\partial f}{\partial x} = 4 \cdot 1 = 4 \]
Compute Graphs -> Neural Networks

- $x_k$ input variables
- $w_{l,m,n}$ network weights (note 3 indices)
  - $l$ which layer
  - $m$ which neuron in layer
  - $n$ which weight in neuron
- $\hat{y}_i$ computed output ($i$ output dim; $n_{out}$)
- $y_i$ ground truth targets
- $L$ loss function
Compute Graphs -> Neural Networks

- **Input layer**: $x_0$, $x_1$
- **Output layer**: $\hat{y}_0$, $y_0$

  - $\hat{y}_0 = x_0 \cdot w_0 + x_1 \cdot w_1$
  - $y_0 = \text{L2 Loss function}$

  - **Input**: $x_0$, $x_1$
  - **Weights** (unknowns!): $w_0$, $w_1$
  - **Loss/Cost**: $\|y_0 - x \cdot x\|^2$

  e.g., class label/regression target
We want to compute gradients w.r.t. all weights $W$. 

Input layer: $x_0, x_1$

Output layer: $\hat{y}_0, y_0$

e.g., class label/ regression target

Input: $x_0, x_1$

Weights (unknowns!): $w_0, w_1$

ReLU Activation (btw. I’m not arguing this is the right choice here)

L2 Loss function:

$\text{Loss/cost} = \max(0, x) - y_0 + x^*x$

We want to compute gradients w.r.t. all weights $W$. 

Compute Graphs -> Neural Networks
Compute Graphs -> Neural Networks

We want to compute gradients w.r.t. all weights $W$. 

I2DL: Prof. Niessner, Prof. Leal-Taixé
Compute Graphs -> Neural Networks

**Goal:** We want to compute gradients of the loss function $L$ w.r.t. all weights $W$

$$L = \sum_i L_i$$

$L$: sum over loss per sample, e.g. L2 loss $\rightarrow$ simply sum up squares:

$$L_i = (\hat{y}_i - y_i)^2$$

$\rightarrow$ use chain rule to compute partials

$$\frac{\partial L_i}{\partial w_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_{i,k}}$$

We want to compute gradients w.r.t. all weights $W$ AND all biases $b$
NNs as Computational Graphs

- We can express any kind of functions in a computational graph, e.g. \( f(w, x) = \frac{1}{1+e^{-\left(b+w_0 x_0 +w_1 x_1\right)}} \)

Sigmoid function

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

\[ f(w, x) = \frac{1}{1 + e^{-(b + w_0 x_0 + w_1 x_1)}} \]
NNs as Computational Graphs

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

\[
\begin{align*}
g(x) &= \frac{1}{x} \quad \Rightarrow \quad \frac{\partial g}{\partial x} = -\frac{1}{x^2} \\
g_\alpha(x) &= \alpha + x \quad \Rightarrow \quad \frac{\partial g}{\partial x} = 1 \\
g(x) &= e^x \quad \Rightarrow \quad \frac{\partial g}{\partial x} = e^x \\
g_\alpha(x) &= \alpha x \quad \Rightarrow \quad \frac{\partial g}{\partial x} = \alpha
\end{align*}
\]

\[ 1 \cdot -\frac{1}{1.37^2} = -0.53 \]
NNs as Computational Graphs

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

- \( g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2} \)
- \( g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x \)
- \( g(\alpha x) = \alpha x \Rightarrow \frac{\partial g}{\partial x} = \alpha \)

\[ \exp(\cdot) \]

\[ 0.37 \cdot 1 = 0.37 \]

\[ -0.53 \cdot 1 = -0.53 \]

\[ 0.73 \]

\[ 1 \]
NNs as Computational Graphs

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

- \( g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2} \)
- \( g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x \)
- \( g(x) = ax \Rightarrow \frac{\partial g}{\partial x} = a \)

\(-0.53 \cdot e^{-1} = -0.2\)
NNs as Computational Graphs

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

\[ g(x) = \frac{1}{x} \Rightarrow \frac{dg}{dx} = -\frac{1}{x^2} \]

\[ g_\alpha(x) = \alpha + x \Rightarrow \frac{dg}{dx} = 1 \]

\[ g(x) = e^x \Rightarrow \frac{dg}{dx} = e^x \]

\[ g_\alpha(x) = \alpha x \Rightarrow \frac{dg}{dx} = \alpha \]
NNs as Computational Graphs

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

\[ g(x) = \frac{1}{x} \Rightarrow \frac{\partial g}{\partial x} = -\frac{1}{x^2} \]
\[ g_\alpha(x) = \alpha + x \Rightarrow \frac{\partial g_\alpha}{\partial x} = 1 \]
\[ g(x) = e^x \Rightarrow \frac{\partial g}{\partial x} = e^x \]
\[ g_\alpha(x) = \alpha x \Rightarrow \frac{\partial g_\alpha}{\partial x} = \alpha \]
Gradient Descent
Gradient Descent

\[ x^* = \arg \min f(x) \]
Gradient Descent

- From derivative to gradient
  \[ \frac{df(x)}{dx} \rightarrow \nabla_x f(x) \]

- Gradient steps in direction of negative gradient
  \[ x' = x - \alpha \nabla_x f(x) \]
Gradient Descent for Neural Networks

Input Layer

Hidden Layer 1
Hidden Layer 2
Hidden Layer 3

Output Layer

\( m \) Neurons

\( l \) Layers
Gradient Descent for Neural Networks

For a given training pair \( \{x, y\} \), we want to update all weights, i.e., we need to compute the derivatives w.r.t. to all weights:

\[
\nabla_w f_{\{x, y\}}(W) = \begin{bmatrix}
\frac{\partial f}{\partial w_{0,0,0}} \\
\vdots \\
\frac{\partial f}{\partial w_{l,m,n}}
\end{bmatrix}
\]

Gradient step:

\[
W' = W - \alpha \nabla_w f_{\{x, y\}}(W)
\]
NNs can Become Quite Complex...

• These graphs can be huge!

[Szegedy et al., CVPR’15] Going Deeper with Convolutions
The Flow of the Gradients

• Many many many many of these nodes form a neural network

NEURONS

• Each one has its own work to do

FORWARD AND BACKWARD PASS
The Flow of the Gradients

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

\[ z = f(x, y) \]

Activations

Activation function
Gradient Descent for Neural Networks

Loss function

\[ L_i = (\hat{y}_i - y_i)^2 \]

\[ \hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j}) \]

\[ h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k}) \]

Just simple:

\[ A(x) = \max(0, x) \]
Gradient Descent for Neural Networks

```
h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})

\hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})

L_i = (\hat{y}_i - y_i)^2
```

**Backpropagation**

\[
\frac{\partial L_i}{\partial w_{1,i,j}} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_{1,i,j}}
\]

\[
\frac{\partial L_i}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i)
\]

\[
\frac{\partial \hat{y}_i}{\partial w_{1,i,j}} = h_j \text{ if } > 0, \text{ else } 0
\]

\[
\frac{\partial L_i}{\partial w_{0,j,k}} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{0,j,k}}
\]

...
Gradient Descent for Neural Networks

How many unknown weights?

- Output layer: \(2 \cdot 4 + 2\)
- Hidden Layer: \(4 \cdot 3 + 4\)

\[
\begin{align*}
    h_j &= A(b_{0,j} + \sum_k x_k w_{0,j,k}) \\
    \hat{y}_i &= A(b_{1,i} + \sum_j h_j w_{1,i,j}) \\
    L_i &= (\hat{y}_i - y_i)^2
\end{align*}
\]

Note that some activations have also weights
Derivatives of Cross Entropy Loss

Binary Cross Entropy loss

\[ L = - \sum_{i=1}^{n_{out}} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \]

\[ \hat{y}_i = \frac{1}{1 + e^{-s_i}} \]

\[ s_i = \sum_j h_j w_{ji} \]

\[ y_i = \frac{1}{1 + e^{-s_i}} \]

Gradients of weights of last layer:

\[ \frac{\partial L_i}{\partial w_{ji}} = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial s_i} \cdot \frac{\partial s_i}{\partial w_{ji}} \]

\[ \frac{\partial L_i}{\partial \hat{y}_i} = -y_i + 1 - y_i = \frac{\hat{y}_i - y_i}{\hat{y}_i(1 - \hat{y}_i)} \]

\[ \frac{\partial \hat{y}_i}{\partial s_i} = \hat{y}_i (1 - \hat{y}_i) \]

\[ \frac{\partial s_i}{\partial w_{ji}} = h_j \]

\[ \Rightarrow \frac{\partial L_i}{\partial w_{ji}} = (\hat{y}_i - y_i)h_j, \quad \frac{\partial L_i}{\partial s_i} = \hat{y}_i - y_i \]
Derivatives of Cross Entropy Loss

Gradients of weights of first layer:

\[
\frac{\partial L}{\partial h_j} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_j} \frac{\partial s_j}{\partial h_j} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial \hat{y}_i} \hat{y}_i (1 - \hat{y}_i) w_{ji} = \sum_{i=1}^{n_{out}} (\hat{y}_i - y_i) w_{ji}
\]

\[
\frac{\partial L}{\partial s_j^1} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_j^1} = \sum_{i=1}^{n_{out}} (\hat{y}_i - y_i) w_{ji} (h_j (1 - h_j))
\]

\[
\frac{\partial L}{\partial w_{kj}^1} = \sum_{i=1}^{n_{out}} \frac{\partial L}{\partial s_j^1} \frac{\partial s_j^1}{\partial w_{kj}^1} = \sum_{i=1}^{n_{out}} (\hat{y}_i - y_i) w_{ji} (h_j (1 - h_j)) x_k
\]
• Inputs $x$ and targets $y$
• Two-layer NN for regression with ReLU activation
• Function we want to optimize:

$$\sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i \|_2^2$$
Gradient Descent for Neural Networks

Initialize $x = 1, \ y = 0,$

$w_1 = \frac{1}{3}, w_2 = 2$

$L(y, \hat{y}; \theta) = \frac{1}{n} \sum_i^n ||\hat{y}_i - y_i||^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial w_2} = \sigma$

Backpropagation

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$
Initialize $x = 1$, $y = 0$, $w_1 = \frac{1}{3}$, $w_2 = 2$

$L(y, \hat{y}; \theta) = \frac{1}{n} \sum_i^n ||\hat{y}_i - y_i||_2^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial y}{\partial w_2} = \sigma$

Backpropagation

$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$

$2 \cdot \frac{2}{3}$
Gradient Descent for Neural Networks

Initialize $x = 1$, $y = 0$, $w_1 = \frac{1}{3}$, $w_2 = 2$

$L(y, \hat{y}; \theta) = \frac{1}{n} \sum_i^n ||\hat{y}_i - y_i||^2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial w_2} = \sigma$

Backpropagation

$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$

$= 2 \cdot \frac{2}{3} \cdot \frac{1}{3}$
Gradient Descent for Neural Networks

Initialize $x = 1$, $y = 0$, $w_1 = \frac{1}{3}, w_2 = 2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2$ \implies $\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma$ \implies $\frac{\partial \hat{y}}{\partial \sigma} = w_2$

$\sigma = \max(0, z)$ \implies $\frac{\partial \sigma}{\partial z} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$

$z = x \cdot w_1$ \implies $\frac{\partial z}{\partial w_1} = x$

Backpropagation:

$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$
Initialize $x = 1$, $y = 0$, $w_1 = \frac{1}{3}, w_2 = 2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$

$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$

$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$

Backpropagation:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$2 \cdot \frac{2}{3}$$
Gradient Descent for Neural Networks

Initialize $x = 1$, $y = 0$,

$w_1 = \frac{1}{3}, w_2 = 2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$

$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$

$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$

Backpropagation

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$2 \cdot \frac{2}{3} \cdot 2$$
Initialize $x = 1$, $y = 0$, $w_1 = \frac{1}{3}, w_2 = 2$

In our case $n, d = 1$:

$L = (\hat{y} - y)^2 \Rightarrow \frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$

$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$

$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$

Gradient Descent for Neural Networks
Gradient Descent for Neural Networks

Initialize $x = 1$, $y = 0$, $w_1 = \frac{1}{3}, w_2 = 2$

In our case $n, d = 1$:

$\hat{y} = w_2 \cdot \sigma \Rightarrow \frac{\partial \hat{y}}{\partial \sigma} = w_2$

$\sigma = \max(0, z) \Rightarrow \frac{\partial \sigma}{\partial z} = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$

$z = x \cdot w_1 \Rightarrow \frac{\partial z}{\partial w_1} = x$

Backpropagation

$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial z} \cdot \frac{\partial z}{\partial w_1}$

$= 2 \cdot \frac{2}{3} \cdot 2 \cdot 1 \cdot 1$
Gradient Descent for Neural Networks

- Function we want to optimize:
  \[ f(x, w) = \sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i \|_2^2 \]
- Computed gradients wrt to weights \( w_1 \) and \( w_2 \)
- Now: update the weights

\[ w' = w - \alpha \cdot \nabla_w f = \left( \begin{array}{c} w_1 \\ w_2 \end{array} \right) - \alpha \cdot \left( \begin{array}{c} \nabla_{w_1} f \\ \nabla_{w_2} f \end{array} \right) \]

\[ = \left( \begin{array}{c} 1/3 \\ 2/3 \end{array} \right) - \alpha \cdot \left( \begin{array}{c} 8/3 \\ 4/9 \end{array} \right) \]

But: how to choose a good learning rate \( \alpha \)?
Gradient Descent

• How to pick good learning rate?

• How to compute gradient for single training pair?

• How to compute gradient for large training set?

• How to speed things up? More to see in next lectures...
Regularization
Recap: Basic Recipe for ML

- Split your data

<table>
<thead>
<tr>
<th></th>
<th>60%</th>
<th>20%</th>
<th>20%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train</td>
<td>validation</td>
<td>test</td>
</tr>
</tbody>
</table>

Find your hyperparameters

Other splits are also possible (e.g., 80%/10%/10%)
Over- and Underfitting

[Graphs showing examples of underfitted, appropriate, and overfitted models]

Source: Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017
Training a Neural Network

- Training/Validation curve

How can we prevent our model from overfitting?

Regularization

Credits: Deep Learning. Goodfellow et al.
Regularization

• Loss function $L(y, \hat{y}, \theta) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$

• Regularization techniques
  – L2 regularization
  – L1 regularization
  – Max norm regularization
  – Dropout
  – Early stopping
  – ...

I2DL: Prof. Niessner, Prof. Leal-Taixé
Regularization: Example

- Input: 3 features $x = [1, 2, 1]$

- Two linear classifiers that give the same result:
  - $\theta_1 = [0, 0.75, 0]$  
    - Ignores 2 features
  - $\theta_2 = [0.25, 0.5, 0.25]$  
    - Takes information from all features
Regularization: Example

• Loss \( L(y, \hat{y}, \theta) = \sum_{i=1}^{n}(x_i \theta_{ji} - y_i)^2 + \lambda R(\theta) \)

• L2 regularization \( R(\theta) = \sum_{i=1}^{n} \theta_i^2 \)

\[ \theta_1 \rightarrow 0 + 0.75^2 + 0 = 0.5625 \]
\[ \theta_2 \rightarrow 0.25^2 + 0.5^2 + 0.25^2 = 0.375 \]  
Minimization

\[ x = [1, 2, 1], \theta_1 = [0, 0.75, 0], \theta_2 = [0.25, 0.5, 0.25] \]
Regularization: Example

- Loss \( L(y, \hat{y}, \theta) = \sum_{i=1}^{n} (x_i \theta_{ji} - y_i)^2 + \lambda R(\theta) \)

- L1 regularization \( R(\theta) = \sum_{i=1}^{n} |\theta_i| \)

\[
\begin{align*}
\theta_1 & \rightarrow 0 + 0.75 + 0 = 0.75 \\
\theta_2 & \rightarrow 0.25 + 0.5 + 0.25 = 1
\end{align*}
\]

Minimization

\( x = [1, 2, 1], \theta_1 = [0, 0.75, 0], \theta_2 = [0.25, 0.5, 0.25] \)
Regularization: Example

• Input: 3 features $x = [1, 2, 1]$

• Two linear classifiers that give the same result:

$\theta_1 = [0, 0.75, 0]$  \quad \text{Ignores 2 features}

$\theta_2 = [0.25, 0.5, 0.25]$  \quad \text{Takes information from all features}
Regularization: Example

• Input: 3 features \( x = [1, 2, 1] \)

• Two linear classifiers that give the same result:

\[
\theta_1 = [0, 0.75, 0] \quad \text{L1 regularization enforces sparsity}
\]

\[
\theta_2 = [0.25, 0.5, 0.25] \quad \text{Takes information from all features}
\]
Regularization: Example

• Input: 3 features \( x = [1, 2, 1] \)

• Two linear classifiers that give the same result:

\[
\theta_1 = [0, 0.75, 0]
\]

L1 regularization enforces \textbf{sparsity}

\[
\theta_2 = [0.25, 0.5, 0.25]
\]

L2 regularization enforces that the weights have \textbf{similar values}
Regularization: Effect

• Dog classifier takes different inputs

Furry

Has two eyes

Has a tail

Has paws

Has two ears

L1 regularization will focus all the attention to a few key features
Regularization: Effect

• Dog classifier takes different inputs

Furry
Has two eyes
Has a tail
Has paws
Has two ears

L2 regularization will take all information into account to make decisions
Regularization for Neural Networks

Combining nodes:
Network output + L2-loss + regularization

\[ \sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i\|_2^2 + \lambda R(w_1, w_2) \]
Combining nodes:
Network output + L2-loss + regularization

\[ \sum_{i=1}^{n} \|w_2 \max(0, w_1 x_i) - y_i \|^2 + \lambda \left\| \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \right\|^2_2 \]
Combining nodes:
Network output + L2-loss + regularization

\[ \sum_{i=1}^{n} \| w_2 \max(0, w_1 x_i) - y_i \|^2 + \lambda (w_1^2 + w_2^2) \]
What is the goal of regularization?

What happens to the training error?
Regularization

- Any strategy that aims to

  Lower validation error

  Increasing training error
Next Lecture

• This week:
  – Check exercises
  – Check office hours 😊

• Next lecture
  – Optimization of Neural Networks
  – In particular, introduction to SGD (our main method!)
See you next week ☺
Further Reading

• Backpropagation
  – Chapter 6.5 (6.5.1 - 6.5.3) in http://www.deeplearningbook.org/contents/mlp.html
  – Chapter 5.3 in Bishop, Pattern Recognition and Machine Learning
  – http://cs231n.github.io/optimization-2/

• Regularization
  – Chapter 7.1 (esp. 7.1.1 & 7.1.2) http://www.deeplearningbook.org/contents/regularization.html
  – Chapter 5.5 in Bishop, Pattern Recognition and Machine Learning