Deep Learning
Multi-Layer Perceptrons,
Backpropagation
Quick Recap- Logistics

- Course website: https://www.cs.cornell.edu/courses/cs4782/2024sp/
  - Tentative schedule, homework policies, grading policies, etc. are on the course page

- We also have a Canvas page
  - Hub for important links (course website, Ed discussion, Gradescope)
  - Let us know if you don’t have access!

- No laptops/mobiles/smart devices in class please!
Agenda

- Perceptron
- Logistic Regression
- Gradient Descent
- Multi-Layer Perceptrons (MLPs)
- Backpropagation
A Classification Problem: Will I Pass This Class?

\[ x^1 = \text{hours spent on project} \]

\[ x^0 = \text{classes attended} \]
A Classification Problem: Will I Pass This Class?

\( x^1 \) = hours spent on project

\( x_i = (x_i^0, x_i^1) \)

\( x^0 \) = classes attended

- Green dots represent Pass
- Red dots represent Fail
Perceptron

- Linear classifier
  - Predecessor to neural network

\[ a_i = \mathbf{w}^\top \mathbf{x}_i + b \]

\[ \hat{y}_i = \begin{cases} 1 & \text{if } a_i \geq 0 \\ 0 & \text{else} \end{cases} \]
A Classification Problem: Will I Pass This Class?

\[ x^1 = \text{hours spent on project} \]

\[ x_i = (x_i^0, x_i^1) \]

\[ x^0 = \text{classes attended} \]
A Classification Problem: Will I Pass This Class?

- Perceptron defines a linear classification boundary

\[ y_i = \begin{cases} 
1 & \text{if } \mathbf{w}^\top \mathbf{x}_i + b \geq 0 \\
0 & \text{else} 
\end{cases} \]

\[ x^1 = \text{hours spent on project} \]

\[ \mathbf{x}_i = (x^0_i, x^1_i) \]

\[ \mathbf{w}: \text{weight vector that defines the hyperplane} \]

Hyperplane perpendicular to \( \mathbf{w} \):
\[ \mathbf{H} = \{ \mathbf{x} : \mathbf{w}^\top \mathbf{x} + b = 0 \} \]
Perceptron

- Linear classifier
  - Predecessor to neural network

\[
a_i = \mathbf{w}^\top \mathbf{x}_i + b
\]

\[
\hat{y}_i = \begin{cases} 
1 & \text{if } a_i \geq 0 \\
0 & \text{else} 
\end{cases}
\]
The “Soft” Perceptron

- Replace step function with continuous approximation

\[ a_i = \mathbf{w}^\top \mathbf{x}_i + b \]

\[ \hat{y}_i = \begin{cases} 1 & \text{if } \sigma(a_i) \geq 0.5 \\ 0 & \text{else} \end{cases} \]

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]
In other words... Logistic Regression

- A single-layer perceptron

\[ a_i = \mathbf{w}^\top \mathbf{x}_i + b \]

\[ \hat{y}_i = \begin{cases} 1 & \text{if } \sigma(a_i) \geq 0.5 \\ 0 & \text{else} \end{cases} \]

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

Absorb bias term into feature vector:

\[ x_i \text{ becomes } \begin{bmatrix} x_i \\ 1 \end{bmatrix} \text{ and } w \text{ becomes } \begin{bmatrix} w \\ b \end{bmatrix} \]

We can see that:

\[ \begin{bmatrix} w \\ b \end{bmatrix}^\top \begin{bmatrix} x_i \\ 1 \end{bmatrix} = w^\top x_i + b \]

Can rewrite logistic regression as

\[ \hat{y}_i = \sigma(w^\top x_i) \]
Maximum Likelihood Estimation

Maximize the likelihood of the observed data \((x_i, y_i)\), where \(y_i \in \{0, 1\}\):

\[
p(y_i | x_i) = \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i}
\]

Note that if \(y_i = 1\), then

\[
p(y_i | x_i) = \hat{y}_i
\]

and if \(y_i = 0\), then

\[
p(y_i | x_i) = 1 - \hat{y}_i
\]
Cross-Entropy Loss (aka Log Loss)

Maximizing the likelihood is equivalent to maximizing the log-likelihood:

$$\log p(y_i|x_i) = \log [\hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i}]$$
$$= y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)$$

Add a negative sign to turn it into a loss, i.e. something to minimize:

$$\ell(\hat{y}_i, y_i) = -\log p(y_i|x_i) = -[y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)]$$

We can plug in our definition of $\hat{y}_i = \sigma(w^\top x_i + b)$:

$$\ell(\hat{y}_i, y_i) = -[y_i \log \sigma(w^\top x_i + b) + (1 - y_i) \log (1 - \sigma(w^\top x_i + b))]$$
Our Goal: Minimize the Loss

Given some training dataset:

$$\mathcal{D}_{TR} = \{x_i, y_i\}_{i=0}^{n}$$

Want to find the parameters $w$ that minimize the empirical risk:

$$\min_w \mathcal{L}(w; \mathcal{D}_{TR}) = \frac{1}{n} \sum_{i}^{n} \ell(\hat{y}_i, y_i)$$

$$= \frac{1}{n} \sum_{i}^{n} \ell(\sigma(w^\top x_i), y_i)$$
Gradient Descent
Visualize Gradient Descent in 1-D
Visualize Gradient Descent in 1-D

Loss

slope of loss at $w^1$ is negative

$w^1$, $w^{min}$, $0$, (goal)

https://web.stanford.edu/~jurafsky/slp3/
Visualize Gradient Descent in 1-D

slope of loss at $w^1$ is negative

one step of gradient descent

$w^1$, $w^{\text{min}}$, $0$, and (goal)
Gradients

**Gradient** of a function of many variables is a vector

- Points in the direction of the greatest **increase** in the function

\[
\nabla_w \mathcal{L}(w; \mathcal{D}_{\text{TR}}) = \begin{bmatrix}
\frac{\partial \mathcal{L}}{\partial w^{(0)}}(w; \mathcal{D}_{\text{TR}}) \\
\frac{\partial \mathcal{L}}{\partial w^{(1)}}(w; \mathcal{D}_{\text{TR}}) \\
\vdots \\
\frac{\partial \mathcal{L}}{\partial w^{(m)}}(w; \mathcal{D}_{\text{TR}})
\end{bmatrix}, \nabla_w \mathcal{L}(w; \mathcal{D}_{\text{TR}}) \in \mathbb{R}^m
\]

**Gradient Descent**: Find the gradient of the loss at the current point

- Move in the **opposite** direction with learning rate \( \alpha \)

\[
w_{t+1} = w_t - \alpha \nabla_{w_t} \mathcal{L}(w_t; \mathcal{D}_{\text{TR}})
\]
Gradient Descent (GD)

\[ w_{t+1} = w_t - \alpha \nabla_w L(w_t; D_{TR}) \]
Demo: Logistic Regression

- Tensorflow Playground
The XOR Problem

- Perceptron can’t learn the XOR function
  - Simple logical operation
- Data is not linearly separable

Demo: The XOR Problem

- Tensorflow Playground
Discuss: What are some ways to handle data that is not linearly separable?

Without deep learning!
Possible Solutions

- Feature engineering
  - Construct a feature space where the data is linearly separable
- Kernel methods
  - Implicitly project the data into a higher-dimensional space where it is linearly separable
- Non-linear classifiers
  - E.g. Nearest neighbor, decision tree algorithms
Demo: Feature Engineering

Tensorflow Playground
Discuss: Feature Engineering

- **Input Image**: Puppy
  - Classification: "dog"

- **Input Image**: Kitten
  - Classification: "cat"
Multi-Layer Perceptron (MLP)

- Compose multiple perceptrons to learn intermediate features

An MLP with 1 hidden layer with 3 hidden units

\[
\begin{align*}
    z_i^0 &= \sigma(w_0^T x_i) \\
    z_i^1 &= \sigma(w_1^T x_i) \\
    z_i^2 &= \sigma(w_2^T x_i) \\
    \hat{y}_i &= \begin{cases} 
        1 & \text{if } \sigma(w_3^T z) \geq 0.5 \\
        0 & \text{else} \end{cases}
\end{align*}
\]
A Simplified MLP Diagram

1 Hidden Layer, 3 Hidden Units
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers
Complex Decision Boundaries

- What does this extra layer give us?
  - Can compose multiple linear classifiers
MLP Demo (1 Hidden Layer)

Tensorflow Playground
Increasing Depth

Discuss: How to construct the decision boundary?
Increasing Depth

- MLP with 1 hidden layer composes linear classifiers
- MLP with 2 hidden layers can compose polygon classifiers
Increasing Depth
Increasing Depth
Increasing Depth
Discuss: What about just one layer?
What about just one layer?
What about just one layer?
Increasing Depth

$\mathbf{x}_i^0$ $\mathbf{x}_i^1$ $\hat{y}_i$
Complex Decision Boundaries

- Can compose *arbitrarily* complex decision boundaries

Activation Functions

- Can replace the sigmoid with other nonlinear functions
  - Still universal approximators!

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]
- Squash between 0 and 1

\[
tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1
\]
- Squash between -1 and 1

\[
ReLU(x) = \max(0, x)
\]
- Threshold at 0

MLP Demo (3 Hidden Layers)

Tensorflow Playground
How to learn MLP weights?

Gradient descent!
Calculus Review: The Chain Rule

Lagrange’s Notation: If \( h(x) = f(g(x)) \), then \( h' = f'(g(x))g'(x) \)

Leibniz’s Notation: If \( z = h(y), y = g(x) \), then \( \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} \)
Calculus Review: The Chain Rule

Lagrange’s Notation: If \( h(x) = f(g(x)) \), then \( h' = f'(g(x))g'(x) \)

Leibniz’s Notation: If \( z = h(y), y = g(x) \), then \( \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} \)

Example: If \( z = \ln(y), y = x^2 \), then

\[
\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = \left( \frac{1}{y} \right)(2x) = \left( \frac{1}{x^2} \right)(2x) = \frac{2}{x}
\]
Try out chain rule!

Differentiate $f(y) = \ln(\ln(-2y^3))$
Solution

\[ f(y) = \ln(\ln(-2y^3)) \]

\[
\frac{d}{dy} f(y) = \frac{1}{\ln(-2y^3)} \cdot \frac{d}{dy} (\ln(-2y^3)) \quad \text{[Using chain rule]}
\]

\[
= -\frac{1}{2y^3} \cdot \frac{d}{dy} (-2y^3)
\]

\[
= -\frac{1}{2y^3} \cdot -6y^2
\]

\[
= \frac{3}{\ln(-2y^3)}
\]

\[
= \frac{3}{y\ln(-2y^3)}
\]

[Using chain rule again]
Multivariate Chain Rule

\[ u \rightarrow v = v(u) \]
\[ w = w(u) \rightarrow z = z(v, w) \]
Multivariate Chain Rule

If $f(u)$ is $z = f(v(u), w(u))$, then

$$\frac{\partial f}{\partial u} = \left( \frac{\partial v}{\partial u} \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \frac{\partial z}{\partial w} \right)$$
Backpropagation - Key Idea

If you know \( \frac{\partial z}{\partial v_1}, \frac{\partial z}{\partial v_2}, \frac{\partial z}{\partial v_3} \)

You can compute \( \frac{\partial z}{\partial u} \)

\[
\frac{\partial z}{\partial u} = \left( \frac{\partial v_1}{\partial u} \cdot \frac{\partial z}{\partial v_1} + \frac{\partial v_2}{\partial u} \cdot \frac{\partial z}{\partial v_2} + \frac{\partial v_3}{\partial u} \cdot \frac{\partial z}{\partial v_3} \right)
\]

https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/
Backpropagation - An Example

\[ u \]
\[ v = u^3 \]
\[ w = u + u \]
\[ z = v \cdot w \]
Backpropagation- An Example

Forward

5

\( u \)

\( v = u^3 \)

\( z = v \cdot w \)

\( w = u + u \)
Backpropagation - An Example

Forward:
- $u = 5$
- $v = u^3 = 125$
- $w = u + u = 10$
- $z = v \cdot w = 1250$

[Source: https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/]
Backpropagation - An Example

Forward

Backward

\[
\frac{\partial z}{\partial u} = \left( \frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} \right)
\]

5

125
\(v = u^3\)

1250
\(z = v \cdot w\)

10
\(w = u + u\)

https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/
Backpropagation - An Example

\[ \frac{\partial z}{\partial u} = \left( \frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} \right) \]

https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/
Backpropagation - An Example

Forward

Backward

\[ v = u^3 \]
\[ \frac{\partial z}{\partial v} = 10 \]

\[ w = u + u \]
\[ \frac{\partial z}{\partial w} = 125 \]

\[ z = v \cdot w \]

\[ \frac{\partial z}{\partial w} = v = 125 \]

\[ \frac{\partial z}{\partial u} = \left( \frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} \right) \]

https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/
Backpropagation - An Example

Forward:
\[ \frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} = 10 \cdot 3 \cdot u^2 = 750 \]

Backward:
- \( v = u^3 \):
  - \( \frac{\partial z}{\partial v} = 10 \)
- \( w = u + u \):
  - \( \frac{\partial z}{\partial w} = 125 \)
- \( z = v \cdot w \):
  - \( \frac{\partial z}{\partial v} = w = 10 \)

5
\[ \frac{\partial z}{\partial u} = 1000 \]

10
\[ \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} = 2 \cdot 125 = 250 \]

1250
\[ \frac{\partial z}{\partial w} = v = 125 \]

\[ \frac{\partial z}{\partial u} = (\frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w}) \]

https://windowsontheory.org/2020/11/03/yet-another-backpropagation-tutorial/
### Backpropagation - MLPs

**Algorithm** Forward Pass through MLP

1. **Input:** input $\mathbf{x}$, weight matrices $\mathbf{W}^{[1]}, \ldots, \mathbf{W}^{[L]}$, bias vectors $\mathbf{b}^{[1]}, \ldots, \mathbf{b}^{[L]}$
2. $\mathbf{z}^{[0]} = \mathbf{x}$  \hspace{1cm} $\triangleright$ Initialize input
3. **for** $l = 1$ to $L$ **do**
4. \hspace{1cm} $\mathbf{a}^{[l]} = \mathbf{W}^{[l]} \mathbf{z}^{[l-1]} + \mathbf{b}^{[l]}$ \hspace{1cm} $\triangleright$ Linear transformation
5. \hspace{1cm} $\mathbf{z}^{[l]} = \sigma^{[l]}(\mathbf{a}^{[l]})$ \hspace{1cm} $\triangleright$ Nonlinear activation
6. **end for**
7. **Output:** $\mathbf{z}^{[L]}$

\[ a_i^{[1]} = \mathbf{W}^{[1]} \mathbf{z}_i^{[0]} + \mathbf{b}_i^{[1]} \quad a_i^{[2]} = \mathbf{W}^{[2]} \mathbf{z}_i^{[1]} + \mathbf{b}_i^{[2]} \]

\[ a_i^{[3]} = \mathbf{W}^{[3]} \mathbf{z}_i^{[2]} + \mathbf{b}_i^{[3]} \]

\[ a_i^{[1]} = \mathbf{W}^{[1]} \mathbf{z}_i^{[0]} + \mathbf{b}_i^{[1]} \quad a_i^{[2]} = \mathbf{W}^{[2]} \mathbf{z}_i^{[1]} + \mathbf{b}_i^{[2]} \]

\[ a_i^{[3]} = \mathbf{W}^{[3]} \mathbf{z}_i^{[2]} + \mathbf{b}_i^{[3]} \]

\[ z_i^{[0]} = x_i \quad z_i^{[1]} = \sigma(a_i^{[1]}) \quad z_i^{[2]} = \sigma(a_i^{[2]}) \quad z_i^{[3]} = \sigma(a_i^{[3]}) \]
**Algorithm** Forward Pass through MLP
1: **Input:** input $x$, weight matrices $W^{[1]}, \ldots, W^{[L]}$, bias vectors $b^{[1]}, \ldots, b^{[L]}$
2: $z^{[0]} = x$  
3: for $l = 1$ to $L$ do
4: $a^{[l]} = W^{[l]}z^{[l-1]} + b^{[l]}$  
5: $z^{[l]} = \sigma^{[l]}(a^{[l]})$  
6: end for
7: **Output:** $z^{[L]}$

**Algorithm** Backward Pass through MLP
1: **Input:** $\{z^{[1]}, \ldots, z^{[L]}\}$, $\{a^{[1]}, \ldots, a^{[L]}\}$, loss gradient $\frac{\partial L}{\partial z^{[L]}}$  
2: $\delta^{[L]} = \frac{\partial L}{\partial z^{[L]}} \odot \sigma^{[L]}'(a^{[L]})$  
3: for $l = L$ to 1 do
4: $\frac{\partial L}{\partial W^{[l]}} = \delta^{[l]}(z^{[l-1]})^T$  
5: $\frac{\partial L}{\partial b^{[l]}} = \delta^{[l]}$  
6: $\delta^{[l-1]} = ((W^{[l]})^T \delta^{[l]}) \odot \sigma^{[l-1]}'(a^{[l-1]})$  
7: end for
8: **Output:** $\frac{\partial L}{\partial W^{[1:L]}}$, $\frac{\partial L}{\partial b^{[1:L]}}$
Backpropagation - MLPs

Algorithm Forward Pass through MLP
1: Input: input $x$, weight matrices $W^{[1]}, \ldots, W^{[L]}$, bias vectors $b^{[1]}, \ldots, b^{[L]}$
2: $z^{[0]} = x$ $\triangleright$ Initialize input
3: for $l = 1$ to $L$ do
4: $a^{[l]} = W^{[l]} z^{[l-1]} + b^{[l]}$ $\triangleright$ Linear transformation
5: $z^{[l]} = \sigma^{[l]}(a^{[l]})$ $\triangleright$ Nonlinear activation
6: end for
7: Output: $z^{[L]}$

Algorithm Backward Pass through MLP (Detailed)
1: Input: $\{z^{[1]}, \ldots, z^{[L]}\}$, $\{a^{[1]}, \ldots, a^{[L]}\}$, loss gradient $\frac{\partial L}{\partial z^{[L]}}$
2: $\delta^{[L]} = \frac{\partial L}{\partial a^{[L]}} = \frac{\partial L}{\partial z^{[L]}} \circ \sigma^{[L]}'(a^{[L]})$ $\triangleright$ Error term
3: for $l = L$ to 1 do
4: $\frac{\partial L}{\partial W^{[l]}} = \frac{\partial L}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial W^{[l]}} = \delta^{[l]}(z^{[l-1]})^T$ $\triangleright$ Gradient of weights
5: $\frac{\partial L}{\partial b^{[l]}} = \frac{\partial L}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial b^{[l]}} = \delta^{[l]}$ $\triangleright$ Gradient of biases
6: $\frac{\partial L}{\partial z^{[l-1]}} = \frac{\partial L}{\partial a^{[l]}} \frac{\partial a^{[l-1]}}{\partial z^{[l-1]}} = (W^{[l]})^T \delta^{[l]}$
7: $\delta^{[l-1]} = \frac{\partial L}{\partial a^{[l-1]}} = \frac{\partial L}{\partial z^{[l-1]}} \frac{\partial z^{[l-1]}}{\partial a^{[l-1]}} = ((W^{[l]})^T \delta^{[l]}) \circ \sigma^{[l-1]}'(a^{[l-1]})$
8: end for
9: Output: $\frac{\partial L}{\partial W^{[1:L]}}, \frac{\partial L}{\partial b^{[1:L]}}$
Discuss: Activation functions

- How do different activation functions behave during backprop?
  - Visualize their derivatives!

**Algorithm** Backward Pass through MLP

1. **Input:** \{\(z^{[1]}, \ldots, z^{[L]}\), \(a^{[1]}, \ldots, a^{[L]}\)}, loss gradient \(\frac{\partial L}{\partial z^{[L]}}\)
2. \(\delta^{[L]} = \frac{\partial L}{\partial z^{[L]}} \odot \sigma^{[L]'}(a^{[L]})\)
3. for \(l = L\) to 1 do
4. \(\frac{\partial L}{\partial W^{[l]}} = \delta^{[l]}(z^{[l-1]})^T\)
5. \(\frac{\partial L}{\partial b^{[l]}} = \delta^{[l]}\)
6. \(\delta^{[l-1]} = ((W^{[l]})^T \delta^{[l]} \odot \sigma^{[l-1]'}(a^{[l-1]}))\)
7. end for
8. **Output:** \(\frac{\partial L}{\partial W^{[1:L]}}, \frac{\partial L}{\partial b^{[1:L]}}\)

*Visualize their derivatives!*
Recap

- MLPs consist of stacks of perceptron units
- MLPs can learn complex decision boundaries by composing simple features into more complex features
- Learn MLP weights with gradient descent
  - Backpropagation efficiently computes gradient
  - Hierarchical feature learning!
Next Week

A deep dive into training neural networks!

https://arxiv.org/abs/1712.09913
Action Items

- Make sure you can access the Canvas and Ed Discussion!

- If you still waiting for a permission code
  - Come talk to us!