Announcement

• Deadline of HW6 (after midterm)
• Midterm in 1 week!
• Midterm sample questions to be released tomorrow
• Midterm evaluation due Saturday
Today’s outline

• Single-layer Perceptron Review
• Multi-layer Perceptron
  • Single output
  • Multiple output
• How to train neural networks
  • Gradient descent
Review: Perceptron

• Given input $x$, weight $w$ and bias $b$, perceptron outputs:

$$o = \sigma(\langle w, x \rangle + b)$$

$$\sigma(x) = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{otherwise}
\end{cases}$$

Activation function

Cats vs. dogs?
Learning AND function using perceptron

The perceptron can learn an AND function

\[ \sigma(x_1w_1 + x_2w_2 + b) \]

\[ \sigma(x) = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{otherwise} 
\end{cases} \]

\[ w_1 = 1, w_2 = 1, b = -1.5 \]
Learning OR function using perceptron

The perceptron can learn an OR function

Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$\sigma(x) = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{otherwise}
\end{cases}$

$w_1 = 1, w_2 = 1, b = -0.5$
The perceptron can learn NOT function (single input)

Output $\sigma(xw_1 + b)$

$\sigma(x) = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{otherwise}
\end{cases}$

$w_1 = -1, b = 0.5$
The limited power of a single neuron

The perceptron cannot learn an **XOR** function (neurons can only generate linear separators)

\[ \text{XOR}(x_1, x_2) = (x_1 \land \neg x_2) \lor (\neg x_1 \land x_2) \]

- \( x_1 = 1, x_2 = 1, y = 0 \)
- \( x_1 = 1, x_2 = 0, y = 1 \)
- \( x_1 = 0, x_2 = 1, y = 1 \)
- \( x_1 = 0, x_2 = 0, y = 0 \)
The limited power of a single neuron

**XOR problem**

If one can represent AND, OR, NOT, one can represent any logic circuit (including XOR), by connecting them.

\[
\text{XOR}(x_1, x_2) = (x_1 \land \neg x_2) \lor (\neg x_1 \land x_2)
\]
Learning XOR
Multilayer Perceptron
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2

\[ h_1 = \sigma \left( \sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1 \right) \]
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2

\[ h_2 = \sigma\left( \sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2 \right) \]
Multi-layer perceptron: Example

• Standard way to connect Perceptrons
• Example: 1 hidden layer, 1 output layer, depth = 2

\[ x \in \mathbb{R}^d \]

\[ h_3 = \sigma(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3) \]
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2

\[
\begin{align*}
  h_1 &= \sigma \left( \sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1 \right) \\
  h_2 &= \sigma \left( \sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2 \right) \\
  h_3 &= \sigma \left( \sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3 \right)
\end{align*}
\]
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2

$$\mathbf{x} \in \mathbb{R}^d$$

$$h_1 = \sigma \left( \sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1 \right)$$

$$h_2 = \sigma \left( \sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2 \right)$$

$$h_3 = \sigma \left( \sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3 \right)$$

Output

Hidden layer
m=3 neurons

Input

$$w_1^{(2)}$$

$$w_2^{(2)}$$

$$w_3^{(2)}$$
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2

\[
x \in \mathbb{R}^d
\]

Input

Hidden layer
m=3 neurons

\[
h_1 = \sigma \left( \sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1 \right)
\]

\[
h_2 = \sigma \left( \sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2 \right)
\]

\[
h_3 = \sigma \left( \sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3 \right)
\]

Output

\[
\hat{y} = \sigma \left( \sum_{i=1}^{m} h_i w_i^{(2)} + b' \right)
\]

Sigmoid activation
Multi-layer perceptron: Matrix Notation

- **Input** \( x \in \mathbb{R}^d \)
- **Hidden** \( W^{(1)} \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^m \)
- **Intermediate output**
  \[ h = \sigma(W^{(1)}x + b) \]
  \[ h \in \mathbb{R}^m \]
Multi-layer perceptron: Matrix Notation

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}$, $\mathbf{b} \in \mathbb{R}^m$
- Intermediate output
  \[ h = \sigma(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}) \]
  \[ h \in \mathbb{R}^m \]

\[ \mathbf{W}^{(0)} = \begin{bmatrix}
    w_{1,1}^0 & \cdots & \cdots & w_{1,d}^0 \\
    \vdots & \ddots & \vdots & \vdots \\
    w_{m,1}^0 & \cdots & \cdots & w_{m,d}^0 
\end{bmatrix} \]
Classify cats vs. dogs
Multi-layer perceptron

Input

Hidden layer
m neurons

Output

Why do we need an a nonlinear activation?
Why do we need an a nonlinear activation?

$h = Wx + b$

$f = w_2^T h + b_2$

hence $f = w_2^T Wx + b'$
Neural network for k-way classification

- K outputs in the final layer

\[ x \in \mathbb{R}^d \]

Input

Hidden layer
m=3 neurons

- \( h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1) \)
- \( h_2 = \sigma(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2) \)
- \( h_3 = \sigma(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3) \)

Output

\[ f_1 = \sum_{i=1}^{m} h_i w_{1i}^{(2)} + b_1' \]

No activation function applied in output layer
Neural network for k-way classification

• K outputs units in the final layer

Multi-class classification (e.g., ImageNet with k=1000)
Softmax

Turns outputs $f$ into probabilities (sum up to 1 across $k$ classes)

$$p(y | x) = \text{softmax}(f) = \frac{\exp f_{y}(x)}{\sum_{i}^{k} \exp f_{i}(x)}$$
Softmax

Turns outputs $f$ into probabilities (sum up to 1 across $k$ classes)

Output layer

\[
\begin{bmatrix}
1.3 \\
5.1 \\
2.2 \\
0.7 \\
1.1
\end{bmatrix}
\]

Softmax activation function

\[
\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}
\]
Softmax

Turns outputs $f$ into probabilities (sum up to 1 across $k$ classes)

\[
\begin{bmatrix}
1.3 \\
5.1 \\
2.2 \\
0.7 \\
1.1 \\
\end{bmatrix}
\xrightarrow{\text{Softmax activation function}}
\begin{bmatrix}
\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \\
\end{bmatrix}
\xrightarrow{\text{Probabilities}}
\begin{bmatrix}
0.02 \\
0.90 \\
0.05 \\
0.01 \\
0.02 \\
\end{bmatrix}
\]

Normalized
Classification Tasks at Kaggle

Classify human protein microscope images into 28 categories

More complicated neural networks

$$y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)$$
More complicated neural networks

• Input $x \in \mathbb{R}^d$
• Hidden $W^{(1)} \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$

\[
\begin{align*}
    h &= \sigma(W^{(1)}x + b) \\
    f &= \sigma(W^{(2)}h + b^{(2)}) \\
    y &= \text{softmax}(f)
\end{align*}
\]

\[y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)\]
More complicated neural networks: multiple hidden layers

\[
h_1 = \sigma(W_1 x + b_1) \\
h_2 = \sigma(W_2 h_1 + b_2) \\
h_3 = \sigma(W_3 h_2 + b_3) \\
f = W_4 h_3 + b_4 \\
y = \text{softmax}(f)
\]
Quiz Break

Which output function is often used for multi-class classification tasks?

A  Sigmoid function
B  Rectified Linear Unit (ReLU)
C  Softmax function
D  Max function
Quiz Break

Which output function is often used for multi-class classification tasks?

A  Sigmoid function
B  Rectified Linear Unit (ReLU)
C  Softmax function
D  Max function
Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?
Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

$$1024 \times 2 + 1024 + 512 \times 1024 + 512 + 512 \times 3 + 3 = 529411$$
Consider a three-layer network with **linear Perceptrons** for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

a) Yes

b) No
Consider a three-layer network with linear Perceptrons for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

a) Yes
b) No

Solution:
A combination of linear Perceptrons is still a linear function.
How to train a neural network?

Classify cats vs. dogs

Input

Hidden layer
100 neurons

Output
How to train a neural network?

\[ x \in \mathbb{R}^d \] One training data point in the training set D

\[ \hat{y} \] Model output for example \( x \)

\[ y \] Ground truth label for example \( x \)

Learning by matching the output to the label

We want \( \hat{y} \rightarrow 1 \) when \( y = 1 \), and \( \hat{y} \rightarrow 0 \) when \( y = 0 \)
How to train a neural network?

Loss function:
\[
\frac{1}{|D|} \sum_{i} \ell(x_i, y_i)
\]

Per-sample loss:
\[
\ell(x_i, y_i) = -y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})
\]

Also known as binary cross-entropy loss
How to train a neural network?

Loss function: \[ \frac{1}{|D|} \sum_{i} \ell(x_i, y_i) \]

Per-sample loss:
\[ \ell(x, y) = \sum_{j=1}^{K} -y_j \log p_j \]

Also known as cross-entropy loss or softmax loss.
How to train a neural network?

Update the weights $W$ to minimize the loss function

$$L = \frac{1}{|D|} \sum_i \ell(x_i, y_i)$$

Use gradient descent!
Gradient Descent

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters $w_0$
- For $t = 1, 2, \ldots$
  
  - Update parameters:
    $$w_t = w_{t-1} - \alpha \frac{\partial L}{\partial w_{t-1}}$$
  
    $$= w_{t-1} - \alpha \frac{1}{|D|} \sum_{x \in D} \frac{\partial \ell(x_i, y_i)}{\partial w_{t-1}}$$
  
- Repeat until converges
Minibatch Stochastic Gradient Descent

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters $w_0$
- For $t = 1, 2, \ldots$
  - Randomly sample a subset (mini-batch) $\hat{D} \in D$
  - Update parameters:
    \[
    w_t = w_{t-1} - \alpha \frac{1}{|\hat{D}|} \sum_{x \in \hat{D}} \frac{\partial \ell(x_i, y_i)}{\partial w_{t-1}}
    \]
- Repeat until converges
Non-convex Optimization

[Gao and Li et al., 2018]
Calculate Gradient (on one data point)

- Want to compute \( \frac{\partial \ell(x, y)}{\partial w_{11}} \)
Calculate Gradient (on one data point)

\[ \ell(x, y) \]
Calculate Gradient (on one data point)

\[ \ell(x, y) \]

\[ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) \]

\[ \frac{\partial \ell(x, y)}{\partial \hat{y}} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}} \]

By chain rule:

\[ \frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}} \]
Calculate Gradient (on one data point)

\[
\ell(x, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})
\]

\[
\frac{\partial \ell(x, y)}{\partial \hat{y}} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}}
\]

By chain rule:

\[
\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_1
\]
Calculate Gradient (on one data point)

\[ \ell(x, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \]

\[ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z)) \]

\[ z = w_{11}x_1 + w_{2,1}x_2 \]

\[ \frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \hat{y}(1 - \hat{y})x_1 \]

- By chain rule:
  
  \[ w_{11}x_1 \]
  
  \[ w_{21}x_2 \]
Calculate Gradient (on one data point)

\[ \ell(x, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \]

By chain rule:

\[ \frac{\partial l}{\partial w_{11}} = \left( \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}} \right)\hat{y}(1 - \hat{y})x_1 \]

\[ z = w_{11}x_1 + w_{2,1}x_2 \]

\[ \hat{y} = \sigma(z) = \sigma(z)(1 - \sigma(z)) \]

\[ \frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z)) \]
Calculate Gradient (on one data point)

\[ \ell(x, y) \]

\[ \ell(x, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \]

\[ z = w_{11}x_1 + w_{21}x_2 \]

\[ \hat{y} = \sigma(z) = \sigma(z)(1 - \sigma(z)) \]

- By chain rule:
  \[ \frac{\partial l}{\partial w_{11}} = (\hat{y} - y)x_1 \]
Calculate Gradient (on one data point)

\[ \ell \left( x, y \right) = \sigma'(z) = \sigma(z)(1 - \sigma(z)) \]

\[ \begin{align*}
\frac{\partial \hat{y}}{\partial z} &= \sigma'(z) \\
\frac{\partial l}{\partial x_1} &= \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y) w_{11}
\end{align*} \]
Calculate Gradient (on one data point)

\[
\ell \left( x, y \right) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})
\]

\[
\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))
\]

- By chain rule:
  \[
  \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}, \quad \frac{\partial l}{\partial a_{12}} = (\hat{y} - y)w_{21}^{(2)}
  \]

Make it deeper
Calculate Gradient (on one data point)

By chain rule:

\[
\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}} = (\hat{y} - y)w_{11} \frac{\partial a_{11}}{\partial w_{11}}
\]
Calculate Gradient (on one data point)

• By chain rule:
  \[
  \frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}} = (\hat{y} - y)w_{11}^{(2)} a_{11}(1 - a_{11})x_1
  \]
Calculate Gradient (on one data point)

- By chain rule:

\[
\frac{\partial l}{\partial a_{11}} = \sigma'(z_{11}) \quad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}
\]

\[
\frac{\partial l}{\partial \partial x_1} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}} \frac{\partial a_{12}}{\partial x_1}
\]
Quiz Break

Gradient Descent in neural network training computes the ______ of a loss function with respect to the model ______ until convergence.

A  gradients, parameters
B  parameters, gradients
C  loss, parameters
D  parameters, loss
Quiz Break

Gradient Descent in neural network training computes the ______ of a loss function with respect to the model ______ until convergence.

A  gradients, parameters
B  parameters, gradients
C  loss, parameters
D  parameters, loss
Quiz Break

Suppose you are given a dataset with 1,000,000 images to train with. Which of the following methods is more desirable if training resources are limit but enough accuracy is needed?

A  Gradient Descent
B  Stochastic Gradient Descent
C  Minibatch Stochastic Gradient Descent
D  Computation Graph
Quiz Break

Suppose you are given a dataset with 1,000,000 images to train with. Which of the following methods is more desirable if training resources are limited but enough accuracy is needed?

A  Gradient Descent
B  Stochastic Gradient Descent
C  Minibatch Stochastic Gradient Descent
D  Computation Graph
HW6
HW6 (working with MNIST dataset)
Demo: Learning XOR using neural net

https://playground.tensorflow.org/
What we’ve learned today...

• Single-layer Perceptron Review
• Multi-layer Perceptron
  • Single output
  • Multiple output
• How to train neural networks
  • Gradient descent
Thanks!