CS 540 Introduction to Artificial Intelligence
Neural Networks (II)

Sharon Yixuan Li
University of Wisconsin-Madison

March 9, 2021
Announcement

• HW3 grade released last Friday (https://piazza.com/class/kk1k70vbawp3ts?cid=551)

• HW6 is going out today, due on **Friday March 19**

• Extended deadline of HW6 (due to midterm)
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:

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

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

Activation function

Cats vs. dogs?
Learning AND function using perceptron

The perceptron can learn an AND function

Output $\sigma(x_1w_1 + x_2w_2 + b)$

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

$w_1 = 1, w_2 = 1, b = -1.5$
Learning OR function using perceptron

The perceptron can learn an OR function

Output $\sigma(x_1w_1 + x_2w_2 + b)$

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

$w_1 = 1, w_2 = 1, b = -0.5$
Learning NOT function using perceptron

The perceptron can learn NOT function (single input)

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

\[ w_1 = -1, b = 0.5 \]
The limited power of a single neuron

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

\[
\begin{align*}
  x_1 &= 1, x_2 = 1, y = 0 \\
  x_1 &= 1, x_2 = 0, y = 1 \\
  x_1 &= 0, x_2 = 1, y = 1 \\
  x_1 &= 0, x_2 = 0, y = 0
\end{align*}
\]

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

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

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

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

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

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

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

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

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

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

\[ \hat{y} = \sigma \left( \sum_{i=1}^{m} h_i w_{i}^{(2)} + b' \right) \]
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
  $h = \sigma(W^{(1)}x + b)$
  $h \in \mathbb{R}^m$
Classify cats vs. dogs

Input

Hidden layer
100 neurons

Output
Neural network for k-way classification

- K outputs in the final layer

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

Input

Hidden layer
m=3 neurons

Output

No activation function applied in output layer

\[ f_1 = \sum_{i=1}^{m} h_i w^{(2)}_{1i} + b'_1 \]

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

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

\[ h_3 = \sigma \left( \sum_{i=1}^{d} x_i w^{(1)}_{3i} + b_3 \right) \]
Neural network for k-way classification

- K outputs units in the final layer

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

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

\[ h_1 = \sigma(\sum_{i=1}^{d} x_iw^{(1)}_{1i} + b_1) \]

\[ h_2 = \sigma(\sum_{i=1}^{d} x_iw^{(1)}_{2i} + b_2) \]

\[ h_3 = \sigma(\sum_{i=1}^{d} x_iw^{(1)}_{3i} + b_3) \]

\[ f_k = \sum_{i=1}^{m} h_iw^{(2)}_{ki} + b'_k \]
**Softmax**

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

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

\[
p(y | x) = \text{softmax}(f) = \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}
\]
Softmax

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

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

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

$$\text{Softmax activation function} = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Output layer

| 1.3 | 5.1 | 2.2 | 0.7 | 1.1 |

Probabilities

| 0.02 | 0.90 | 0.05 | 0.01 | 0.02 |

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

$$h = \sigma(W^{(1)}x + b)$$
$$f = \sigma(W^{(2)}h + b^{(2)})$$
$$y = \text{softmax}(f)$$

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

\[
\begin{align*}
  h_1 &= \sigma(W_1 x + b_1) \\
  h_2 &= \sigma(W_2 h_1 + b_2) \\
  h_3 &= \sigma(W_3 h_2 + b_3) \\
  f &= W_4 h_3 + b_4 \\
  y &= \text{softmax}(f)
\end{align*}
\]
How to train a neural network?

Classify cats vs. dogs

Input

Hidden layer
100 neurons

Output
How to train a neural network?

\( \mathbf{x} \in \mathbb{R}^d \) One training data point in the training set \( D \)

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

\( y \) Ground truth label for example \( \mathbf{x} \)

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:
\[
\ell(x_i, y_i) = -y \log(p_i) + (1 - y)\log(1 - p_i)
\]

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

D can be very large. Expensive
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 (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)

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

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

\[ \hat{y} = \text{sigmoid function} (z) \]

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

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

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

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

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)

\[ \ell(x, y) = \begin{cases} -y \log(\hat{y}) & \text{if } y = 1 \\ -(1 - y) \log(1 - \hat{y}) & \text{if } y = 0 \end{cases} \]

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)

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

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

- By chain rule:

\[ \frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \hat{y}(1 - \hat{y})x_1 \]
Calculate Gradient (on one data point)

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

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

By chain rule:

\[ \frac{\partial l}{\partial w_{11}} = \left( \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}} \right)\hat{y}(1 - \hat{y})x_1 \]
Calculate Gradient (on one data point)

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

By chain rule:

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

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

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

By chain rule:

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

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

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

By chain rule:

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

\[
\begin{align*}
\frac{\partial a_{11}}{\partial z_{11}} &= \sigma'(z_{11}) \\
\frac{\partial l}{\partial a_{11}} &= (\hat{y} - y)w_{11}^{(2)} \\
\frac{\partial l}{\partial w_{11}} &= \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}} = (\hat{y} - y)w_{11}^{(2)} a_{11}(1 - a_{11})x_1
\end{align*}
\]
Calculate Gradient (on one data point)

By chain rule:

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

Based on slides from Xiaojin (Jerry) Zhu, Yingyu Liang and Yin Li (http://pages.cs.wisc.edu/~jerryzhu/cs540.html), and Alex Smola: https://courses.d2l.ai/berkeley-stat-157/units/mlp.html