Today’s outline

• Single-layer Perceptron Continued
• Multi-layer Perceptron
  • Single output
  • Multiple output
• How to train neural networks
  • Gradient descent
Step Function activation

Step function is discontinuous, which cannot be used for gradient descent

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

Map input into $[0, 1]$, a soft version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$
Logistic regression

$x \in \mathbb{R}^d, y = \{-1, +1\}$

\[ p(y = 1 \mid x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)} \]

\[ p(y = -1 \mid x) = 1 - \sigma(w^T x) = \frac{1}{1 + \exp(w^T x)} \]
Logistic regression

Given: \( \{(x_i, y_i)\}_{i=1}^{n} \)

Training: maximize likelihood estimate (on the conditional probability)

\[
\max_{w} \sum_{i} \log \frac{1}{1 + \exp(-y_i w^T x_i)}
\]
Logistic regression

Given: \( \{(x_i, y_i)\}_{i=1}^{n} \)

Training: maximize likelihood estimate (on the conditional probability)

When training data is linearly separable, many solutions
Logistic regression

Given: $\{(x_i, y_i)\}_{i=1}^n$

Training: maximum A posteriori (MAP)

$$\min_w \sum_i - \log \frac{1}{1 + \exp(-y_i w^T x_i)} + \frac{\lambda}{2} ||w||^2_2$$

- Convex optimization
- Solve via (stochastic) gradient descent
Tanh Activation

Map inputs into (-1, 1)

\[ \tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \]
ReLU Activation

ReLU: rectified linear unit (commonly used in modern neural networks)

$$\text{ReLU}(x) = \max(x, 0)$$
Multilayer Perceptron
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
Learning XOR
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

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

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

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

Hidden layer
m=3 neurons

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

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

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

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

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

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

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

Output

\[ \hat{y} = \sigma(\sum_{i=1}^{m} h_i w_{i}^{(2)} + b') \]
Multi-layer perceptron: Example

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

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

\[ \hat{y} = \sigma(\sum_{i=1}^{m} h_i w_i^{(2)} + b') \]
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
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
  \[ h = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}) \]
  \[ h \in \mathbb{R}^m \]
Multi-layer perceptron

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

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

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

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

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

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

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

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

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

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

$$p(y | x) = \text{softmax}(f) = \frac{\exp f_y(x)}{\sum_{i=1}^k \exp f_i(x)}$$
Softmax

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

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

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

Normalized
Classification Tasks at Kaggle

Classify human protein microscope images into 28 categories

More complicated neural networks

Input layer

Hidden layer

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

\[
h = \sigma(W^{(1)}x + b)
\]

\[
f = W^{(2)}h + b^{(2)}
\]

\[
y = \text{softmax}(f)
\]
More complicated neural networks

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

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

\[
y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)
\]
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 \( \hat{y} \to 1 \) when \( y = 1 \), and \( \hat{y} \to 0 \) when \( y = 0 \)
How to train a neural network?

Loss function: \( \frac{1}{|D|} \sum_i \ell(x_i, y_i) \)
How to train a neural network?

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

Per-sample loss:
\[
\ell(x, y) = -y \log(\hat{y}) - (1 - y)\log(1 - \hat{y})
\]
How to train a neural network?

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

Per-sample loss:

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

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

Loss function: \[ \frac{1}{|D|} \sum_{i} \ell(x_i, y_i) \]
How to train a neural network?

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

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

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

$D$ can be very large. Expensive

• 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)

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

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

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

\[
\frac{\partial z}{\partial w_{11}} = x_1
\]

\[
\frac{\partial z}{\partial w_{21}} = x_2
\]

\[
\frac{\partial \hat{y}}{\partial w_{11}} = w_{11} x_1
\]

\[
\frac{\partial \hat{y}}{\partial w_{21}} = w_{21} x_2
\]
Calculate Gradient (on one data point)

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

**By chain rule:**

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

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

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

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

\[ \frac{\partial \hat{y}}{\partial \hat{y}} = 1 \]

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

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

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

\[ \frac{\partial \ell(x, y)}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_1 \]

By chain rule:
Calculate Gradient (on one data point)

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

By chain rule:

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

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

\[ \ell (x, y) \]

\[ y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \]

By chain rule:

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

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

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

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

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)

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

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:

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

By chain rule:

\[
\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}} \frac{\partial a_{12}}{\partial x_1}
\]
Brief history of neural networks

- **1943** S. McCulloch – W. Pitts
- **1957** F. Rosenblatt
- **1960** B. Widrow – M. Hoff
- **1969** M. Minsky – S. Papert
- **1986** D. Rumelhart – G. Hinton – R. Williams
- **1995** V. Vapnik – C. Cortes
- **2006** G. Hinton – S. Rupinar

**Key Events**

- **ADALINE**: 1960
- **XOR Problem**: 1960
- **Multi-layered Perceptron (Backpropagation)**: 1980
- **SVM**: 1995
- **Deep Neural Network (Pretraining)**: 2006

**Timeline**

- **1940**
- **1950**
- **1960**
- **1970**
- **1980**
- **1990**
- **2000**
- **2010**

**Key Features**

- Adjustable Weights
- Weights are not Learned
- Learnable Weights and Threshold
- XOR Problem
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting
- Limitations of learning prior knowledge
- Kernel function: Human intervention
- Hierarchical feature Learning
<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
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!