CS540  Introduction to Artificial Intelligence
Neural Networks: Review
University of Wisconsin-Madison

Spring 2023
Announcements

• Homeworks:
  • HW 7 due in one week
• Midterms are being graded; solutions on Canvas.
• Final exam is May 12, 5:05 - 7:05 pm.
• Class roadmap:
  • Practice Questions on Canvas

<table>
<thead>
<tr>
<th>Tuesday, April 4</th>
<th>Neural Network Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thursday, April 6</td>
<td>Uninformed Search</td>
</tr>
<tr>
<td>Tuesday, April 11</td>
<td>Informed Search</td>
</tr>
<tr>
<td>Thursday, April 13</td>
<td>Advanced Search</td>
</tr>
</tbody>
</table>
How to classify

Cats vs. dogs?

Neural networks can also be used for regression.

- Typically, no activation on outputs, mean squared error loss function.
How to classify Cats vs. dogs?

Neural networks can also be used for regression.

- Typically, no activation on outputs, mean squared error loss function.
Inspiration from neuroscience

- Inspirations from human brains
- Networks of simple and homogenous units (a.k.a neuron)
Perceptron

- Given input \( x \), weight \( w \) and bias \( b \), perceptron outputs:

\[
o = \sigma \left( w^T x + b \right)
\]

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

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

$$o = \sigma \left( w^\top x + b \right)$$

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

Activation function

---

Cats vs. dogs?

![Cat](image1.png) ![Dog](image2.png)
Perceptron

• Goal: learn parameters $\mathbf{w} = \{w_1, w_2, \ldots, w_d\}$ and $b$ to minimize the classification error

Cats vs. dogs?
Example 2: Predict whether a user likes a song or not
Example 2: Predict whether a user likes a song or not

Using Perceptron

\[ y = 1 \]

\[ y = 0 \]

User Sharon

- DisLike
- Like

Intensity

Relaxed  Tempo  Fast
Learning logic functions using perceptron

The perceptron can learn an AND function

- $x_1 = 1, x_2 = 1, y = 1$
- $x_1 = 1, x_2 = 0, y = 0$
- $x_1 = 0, x_2 = 1, y = 0$
- $x_1 = 0, x_2 = 0, y = 0$
Learning logic functions using perceptron

The perceptron can learn an AND function

\[ x_1 = 1, x_2 = 1, y = 1 \]
\[ x_1 = 1, x_2 = 0, y = 0 \]
\[ x_1 = 0, x_2 = 1, y = 0 \]
\[ x_1 = 0, x_2 = 0, y = 0 \]
The perceptron can learn an AND function.

Learning logic functions using perceptron

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

\[ \sigma(x_1 w_1 + x_2 w_2 + b) \]

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

\[ w_1 = 1, w_2 = 1, b = -0.5 \]
XOR Problem (Minsky & Papert, 1969)

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

\[ x_1 = 1, x_2 = 1, y = 0 \]
\[ x_1 = 1, x_2 = 0, y = 1 \]
\[ x_1 = 0, x_2 = 1, y = 1 \]
\[ x_1 = 0, x_2 = 0, y = 0 \]
XOR Problem (Minsky & Papert, 1969)

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

\[ x_1 = 1, x_2 = 1, y = 0 \]
\[ x_1 = 1, x_2 = 0, y = 1 \]
\[ x_1 = 0, x_2 = 1, y = 1 \]
\[ x_1 = 0, x_2 = 0, y = 0 \]

This contributed to the first AI winter
Quiz break

Which one of the following is NOT true about perceptron?

A. Perceptron only works if the data is linearly separable.
B. Perceptron can learn AND function
C. Perceptron can learn XOR function
D. Perceptron is a supervised learning algorithm
Quiz break

Which one of the following is NOT true about perceptron?

A. Perceptron only works if the data is linearly separable.
B. Perceptron can learn AND function
C. Perceptron can learn XOR function
D. Perceptron is a supervised learning algorithm
Multilayer Perceptron
How to classify
Cats vs. dogs?
Single Hidden Layer

- Input $x \in \mathbb{R}^d$
- Hidden $W \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$
- Intermediate output

$$h = \sigma(Wx + b)$$

$\sigma$ is an element-wise activation function
Neural networks with one hidden layer

\[ m \times d \times d \times 1 \]

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

\[ W \]
Neural networks with one hidden layer
Neural networks with one hidden layer

\[ \mathbf{W} \in \mathbb{R}^{m \times d}, \quad \mathbf{x} \in \mathbb{R}^{d}, \quad \mathbf{b} \in \mathbb{R}^{m}, \quad \frac{1}{1 + e^{-x}} \]

\[ \mathbf{W} \mathbf{x} + \mathbf{b} \]

Element-wise activation function
Neural networks with one hidden layer

**Key elements:** linear operations + Nonlinear activations

\[
\begin{align*}
W & \in \mathbb{R}^{d \times m} \\
x & \in \mathbb{R}^{d} \\
b & \in \mathbb{R}^{m} \\
Wx + b & \in \mathbb{R}^{m} \\
\tanh & \text{ (or similar function)} \\
\end{align*}
\]
Single Hidden Layer

- Output \( f = w_2^T h + b_2 \)
- Normalize the output into probability using sigmoid
  \[
p(y = 1 \mid x) = \frac{1}{1 + e^{-f}}
  \]
Multi-class classification

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

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

$$p(y \mid x) = \text{softmax}(f)$$

$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$
Deep neural networks (DNNs)

\[ h_1 = \sigma(W_1x + b_1) \]
\[ h_2 = \sigma(W_2h_1 + b_2) \]
\[ h_3 = \sigma(W_3h_2 + b_3) \]
\[ f = W_4h_3 + b_4 \]
\[ y = \text{softmax}(f) \]
Deep neural networks (DNNs)

NNs are composition of nonlinear functions

\[
\begin{align*}
h_1 &= \sigma(W_1x + b_1) \\
h_2 &= \sigma(W_2h_1 + b_2) \\
h_3 &= \sigma(W_3h_2 + b_3) \\
f &= W_4h_3 + b_4 \\
y &= \text{softmax}(f)
\end{align*}
\]
Classify MNIST handwritten digits
Classify MNIST handwritten digits
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
Cross-Entropy Loss

\[ L_{CE} = \sum_j - y_j \log(p_j) \]

\[ = - \log(0.8) \]

Goal: push \( p \) and \( Y \) to be identical
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:
    \[ w_t = w_{t-1} - \alpha \frac{\partial L}{\partial w_{t-1}} \]
    \[ = w_{t-1} - \alpha \frac{1}{|D|} \sum_{x \in D} \frac{\partial \ell(x_i, y_i)}{\partial w_{t-1}} \]
- 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 (mini-batch) \( B \subset D \)
  Update parameters:

\[
    w_t = w_{t-1} - \alpha \frac{1}{|B|} \sum_{x \in B} \frac{\partial \ell(x_i, y_i)}{\partial w_{t-1}}
\]

• Repeat
Calculate gradient: backpropagation with chain rule

- Define a loss function $L$, must compute $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$ for all weights and biases.
Calculate gradient: backpropagation with chain rule

• Define a loss function $L$, must compute $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$ for all weights and biases.

• Gradient to a variable =
  gradient on the top $\times$ gradient from the current operation

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial W}$$
Calculate gradient: backpropagation with chain rule

- Define a loss function $L$, must compute $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$ for all weights and biases.

- Gradient to a variable = gradient on the top $\times$ gradient from the current operation

\[
\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial W}
\]
Calculate gradient: backpropagation with chain rule

- Define a loss function $L$, must compute $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial b}$ for all weights and biases.

- Gradient to a variable = gradient on the top $\times$ gradient from the current operation
  
  \[
  \frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial W}
  \]

\[a = \text{sigmoid}(Wx + b)\]
Calculate gradient: backpropagation with chain rule

- Define a loss function $L$, must compute $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$ for all weights and biases.

- Gradient to a variable = gradient on the top $\times$ gradient from the current operation

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial W}$$

$a = \text{sigmoid}(Wx + b)$
Calculate gradient: backpropagation with chain rule

- Define a loss function $L$, must compute $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$ for all weights and biases.

- Gradient to a variable = gradient on the top $\times$ gradient from the current operation.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial W}$$

**Diagram:**

- $x$ connects to $z_1$.
- $z_1$ connects to $z_2$.
- $z_2$ connects to $a$.
- $a = \text{sigmoid}(Wx + b)$.
- $a$ connects to $L$. 

*This diagram represents a simple neural network with a loss function at the output.*
Non-convex Optimization

[Gao and Li et al., 2018]
How to classify
Cats vs. dogs?
How to classify Cats vs. dogs?

36M floats in a RGB image!

Dual 12MP wide-angle and telephoto cameras
Cats vs. dogs?
Cats vs. dogs?
Cats vs. dogs?

~ 36M elements x 100 = ~3.6B parameters!
Convolutions come to rescue!
Where is Waldo?
Why Convolution?

- Translation Invariance
- Locality
2-D Convolution
2-D Convolution

Input

\[
\begin{pmatrix}
0 & 1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8 \\
\end{pmatrix}
\]

Kernel

\[
\begin{pmatrix}
0 & 1 \\
2 & 3 \\
\end{pmatrix}
\]

Output

\[
\begin{pmatrix}
19 & 25 \\
37 & 43 \\
\end{pmatrix}
\]
### 2-D Convolution

<table>
<thead>
<tr>
<th>Input</th>
<th>Kernel</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>0 1</td>
<td>19 25</td>
</tr>
<tr>
<td>3 4 5</td>
<td>2 3</td>
<td>37 43</td>
</tr>
<tr>
<td>6 7 8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19, \\
1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25, \\
3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37, \\
4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43. 
\]
2-D Convolution

<table>
<thead>
<tr>
<th>Input</th>
<th>Kernel</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>0 1</td>
<td>19 25</td>
</tr>
<tr>
<td>3 4 5</td>
<td>2 3</td>
<td>37 43</td>
</tr>
<tr>
<td>6 7 8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19, \\
1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25, \\
3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37, \\
4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.
\]
2-D Convolution Layer

- **input matrix**
- **kernel matrix**
- **b: scalar bias**
- **output matrix**

\[
W \text{ and } b \text{ are learnable parameters}
\]

\[
Y = X \star W + b
\]
2-D Convolution Layer with Stride and Padding

- Stride is the \#rows/\#columns per slide
- Padding adds rows/columns around input
- Output shape

\[
\begin{align*}
\left\lfloor \frac{(n_h - k_h + p_h + s_h)}{s_h} \right\rfloor \times \left\lfloor \frac{(n_w - k_w + p_w + s_w)}{s_w} \right\rfloor
\end{align*}
\]
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels

\[(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56\]
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels

\[
\begin{align*}
\text{Input} & \quad \text{Kernel} & \\
\begin{array}{ccc}
0 & 1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8
\end{array} & \begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{array} & \\
\times & = & \\
\begin{array}{ccc}
0 & 1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8
\end{array} & \begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{array} & \\
+ & \\
\end{align*}
\]
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{pmatrix}
\times
\begin{pmatrix}
0 & 1 \\
2 & 3 \\
6 & 7 & 8 \\
\end{pmatrix}
+
\begin{pmatrix}
0 & 1 \\
3 & 4 \\
6 & 7 & 8 \\
\end{pmatrix}
\times
\begin{pmatrix}
1 & 2 \\
3 & 4 \\
0 & 1 \\
\end{pmatrix}
\times
\begin{pmatrix}
2 & 3 \\
5 & 4 \\
1 & 2 \\
\end{pmatrix}
= 56
\]
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels

\[
\begin{array}{c}
\text{Input} \\
0 \ 1 \ 2 \\
3 \ 4 \ 5 \\
6 \ 7 \ 8 \\
\end{array}
\quad *
\quad \begin{array}{c}
\text{Kernel} \\
0 \ 1 \\
2 \ 3 \\
\end{array}
\quad =
\quad \begin{array}{c}
\text{Input} \\
1 \ 2 \ 3 \\
4 \ 5 \ 6 \\
7 \ 8 \ 9 \\
\end{array}
\quad *
\quad \begin{array}{c}
\text{Kernel} \\
1 \ 2 \\
3 \ 4 \\
\end{array}
\quad +
\quad \begin{array}{c}
\text{Output} \\
56 \ 72 \\
104 \ 120 \\
\end{array}
\end{array}
\]

\[
\text{Input} \times \text{Kernel} + \text{Output} = 56
\]
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels

\[(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56\]
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a 2D kernel for each channel, and then sum results over channels
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Also call each 3D kernel a “filter”, which produce only one output channel (due to summation over channels)
Multiple filters (in one layer)

- Apply multiple filters on the input
- Each filter may learn different features about the input
- Each filter (3D kernel) produces one output channel

RGB (3 input channels)
Conv1 Filters in AlexNet

- 96 filters (each of size 11x11x3)
- Gabor filters

Figures from Visualizing and Understanding Convolutional Networks by M. Zeiler and R. Fergus
Multiple Output Channels

- The # of output channels = # of filters
- Input $X: c_i \times n_h \times n_w$
- Kernel $W: c_o \times c_i \times k_h \times k_w$
- Output $Y: c_o \times m_h \times m_w$

\[ Y_{i,:,:,} = X \star W_{i,:,:,} \]

for $i = 1, \ldots, c_o$
Multiple Output Channels

• The # of output channels = # of filters
• Input $X: c_i \times n_h \times n_w$
• Kernel $W: c_o \times c_i \times k_h \times k_w$
• Output $Y: c_o \times m_h \times m_w$

$Y_{i,:,:} = X \star W_{i,:,:,:,:}$
for $i = 1, \ldots, c_o$
Multiple Output Channels

- The # of output channels = # of filters
- Input $X: c_i \times n_h \times n_w$
- Kernel $W: c_o \times c_i \times k_h \times k_w$
- Output $Y: c_o \times m_h \times m_w$

$Y_{i,:,:,} = X \star W_{i,:,:,}$
for $i = 1, \ldots, c_o$
Convolutional Neural Networks
LeNet Architecture

32x32 image

convolution

6@28x28
C1 feature map

6@14x14
S2 feature map

convolution

pooling

16@10x10
C3 feature map

pooling

16@5x5
S4 feature map

full

120 - F5 full

84 - F6 full

full

Gauss

10 - Out
Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998
Gradient-based learning applied to document recognition
Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998
Gradient-based learning applied to document recognition
Quiz break

Which one of the following is NOT true?

A. LeNet has two convolutional layers
B. The first convolutional layer in LeNet has 5x5x6x3 parameters, in case of RGB input
C. Pooling is performed right after convolution
D. Pooling layer does not have learnable parameters
Quiz break

Which one of the following is NOT true?

A. LeNet has two convolutional layers
B. The first convolutional layer in LeNet has 5x5x6x3 parameters, in case of RGB input
C. Pooling is performed right after convolution
D. Pooling layer does not have learnable parameters

Pooling is performed after ReLU: conv->relu->pooling
Evolution of neural net architectures
Evolution of neural net architectures

- **LeNet**
- **AlexNet**
- **Inception Net**
- **ResNet**
- **DenseNet**
AlexNet

[Krizhevsky et al. 2012]
AlexNet vs LeNet Architecture

Larger pool size, change to max pooling

Larger kernel size, stride because of the increased image size, and more output channels.

AlexNet:
- 3x3 MaxPool, stride 2
- 11x11 Conv (96), stride 4
- Image (3x224x224)

LeNet:
- 2x2 AvgPool, stride 2
- 5x5 Conv (6), pad 2
- Image (32x32)
AlexNet Architecture

3 additional convolutional layers

More output channels.

AlexNet

3x3 MaxPool, stride 2

3x3 Conv (384), pad 1

3x3 Conv (384), pad 1

3x3 MaxPooling, stride 2

5x5 Conv (256), pad 2

LeNet

2x2 AvgPool, stride 2

5x5 Conv (16)
**ResNet:** Going deeper in depth

ImageNet Top-5 error%

[He et al. 2015]
Going deeper in deep learning
Going deeper in deep learning

- Convolutional neural networks are one of many special types of layers.
Going deeper in deep learning

• Convolutional neural networks are one of many special types of layers.

• Main use is for processing images.
Going deeper in deep learning

- Convolutional neural networks are one of many special types of layers.
  - Main use is for processing images.
  - Also can be useful for handling time series.
Going deeper in deep learning

• Convolutional neural networks are one of many special types of layers.
  • Main use is for processing images.
  • Also can be useful for handling time series.
• Other common architectures:
Going deeper in deep learning

- Convolutional neural networks are one of many special types of layers.
  - Main use is for processing images.
  - Also can be useful for handling time series.
- Other common architectures:
  - Recurrent neural networks: hidden activations are a function of input and activations from previous inputs. Designed for sequential data such as text.
Going deeper in deep learning

• Convolutional neural networks are one of many special types of layers.
  • Main use is for processing images.
  • Also can be useful for handling time series.

• Other common architectures:
  • Recurrent neural networks: hidden activations are a function of input and activations from previous inputs. Designed for sequential data such as text.
  • Graph neural networks: take graph data as input.
Going deeper in deep learning

- Convolutional neural networks are one of many special types of layers.
  - Main use is for processing images.
  - Also can be useful for handling time series.
- Other common architectures:
  - Recurrent neural networks: hidden activations are a function of input and activations from previous inputs. Designed for sequential data such as text.
  - Graph neural networks: take graph data as input.
  - Transformers: take sequences as input and learn what parts of input to pay attention to.
What we’ve learned today…
What we’ve learned today…

• Modeling a single neuron
What we’ve learned today…

• Modeling a single neuron
  • Linear perceptron
What we’ve learned today…

• Modeling a single neuron
  • Linear perceptron
  • Limited power of a single neuron
What we’ve learned today…

- Modeling a single neuron
  - Linear perceptron
  - Limited power of a single neuron
- Multi-layer perceptron
What we’ve learned today…

• Modeling a single neuron
  • Linear perceptron
  • Limited power of a single neuron
• Multi-layer perceptron
• Training of neural networks
What we’ve learned today...

- Modeling a single neuron
  - Linear perceptron
  - Limited power of a single neuron
- Multi-layer perceptron
- Training of neural networks
  - Loss function (cross entropy)
What we’ve learned today…

- Modeling a single neuron
  - Linear perceptron
  - Limited power of a single neuron
- Multi-layer perceptron
- Training of neural networks
  - Loss function (cross entropy)
  - Backpropagation and SGD
What we’ve learned today...

- Modeling a single neuron
  - Linear perceptron
  - Limited power of a single neuron
- Multi-layer perceptron
- Training of neural networks
  - Loss function (cross entropy)
  - Backpropagation and SGD
- Convolutional neural networks
What we’ve learned today…

• Modeling a single neuron
  • Linear perceptron
  • Limited power of a single neuron
• Multi-layer perceptron
• Training of neural networks
  • Loss function (cross entropy)
  • Backpropagation and SGD
• Convolutional neural networks
  • Convolution, pooling, stride, padding
What we’ve learned today…

• Modeling a single neuron
  • Linear perceptron
  • Limited power of a single neuron
• Multi-layer perceptron
• Training of neural networks
  • Loss function (cross entropy)
  • Backpropagation and SGD
• Convolutional neural networks
  • Convolution, pooling, stride, padding
  • Basic architectures (LeNet etc.)
What we’ve learned today…

- Modeling a single neuron
  - Linear perceptron
  - Limited power of a single neuron
- Multi-layer perceptron
- Training of neural networks
  - Loss function (cross entropy)
  - Backpropagation and SGD
- Convolutional neural networks
  - Convolution, pooling, stride, padding
  - Basic architectures (LeNet etc.)
  - More advanced architectures (AlexNet, ResNet etc)
Thank you!

Some of the slides in these lectures have been adapted from materials developed by Alex Smola and Mu Li: