Lecture 11:

- Multi-layer Perceptron
- Forward Pass

Aykut Erdem
March 2016
Hacettepe University
Last time… Linear classification

\[ f(x, W) = Wx \]

\[ 10 \times 1 \times 10 \times 3072 \]

[32x32x3] array of numbers 0...1

parameters, or “weights”

10 numbers, indicating class scores
Last time... Linear classification
Last time... Linear classification

\[ f(x_i, W, b) = W x_i + b \]

[32x32x3]
array of numbers 0...1
(3072 numbers total)
Interactive web demo time....

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/
Last time... Perceptron

\[ f(x) = \sum_i w_i x_i = \langle w, x \rangle \]
This Week

• Multi-layer perceptron

• Forward Pass

• Backward Pass
Introduction
A brief history of computers

- Data grows at higher exponent
- Moore’s law (silicon) vs. Kryder’s law (disks)
- Early algorithms data bound, now CPU/RAM bound

---

<table>
<thead>
<tr>
<th></th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>$10^2$</td>
<td>$10^3$</td>
<td>$10^5$</td>
<td>$10^8$</td>
<td>$10^{11}$</td>
</tr>
<tr>
<td>RAM</td>
<td>?</td>
<td>1MB</td>
<td>100MB</td>
<td>10GB</td>
<td>1TB</td>
</tr>
<tr>
<td>CPU</td>
<td>?</td>
<td>10MF</td>
<td>1GF</td>
<td>100GF</td>
<td>1PF GPU</td>
</tr>
</tbody>
</table>

---

slide by Alex Smola
Not linearly separable data

- Some datasets are not linearly separable!
  - e.g. XOR problem

- Nonlinear separation is trivial
Addressing non-linearly separable data

- Two options:
  - Option 1: Non-linear functions
  - Option 2: Non-linear classifiers
Option 1 — Non-linear features

• Choose non-linear features, e.g.,
  - Typical linear features: \( w_0 + \sum_i w_i x_i \)
  - Example of non-linear features:
    • Degree 2 polynomials, \( w_0 + \sum_i w_i x_i + \sum_{ij} w_{ij} x_i x_j \)

• Classifier \( h_w(x) \) still linear in parameters \( w \)
  - As easy to learn
  - Data is linearly separable in higher dimensional spaces
  - Express via kernels
Option 2 — Non-linear classifiers

• Choose a classifier $h_w(x)$ that is non-linear in parameters $w$, e.g.,
  - Decision trees, neural networks,…

• More general than linear classifiers
• But, can often be harder to learn (non-convex optimization required)
• Often very useful (outperforms linear classifiers)
• In a way, both ideas are related
Biological Neurons

- **Soma (CPU)**
  Cell body - combines signals

- **Dendrite (input bus)**
  Combines the inputs from several other nerve cells

- **Synapse (interface)**
  Interface and *parameter store* between neurons

- **Axon (cable)**
  May be up to 1m long and will transport the activation signal to neurons at different locations
Recall: The Neuron Metaphor

- Neurons accept information from multiple inputs,
  transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node
Types of Neuron

- **Linear Neuron**
  
  \[ y = \theta_0 + \sum_i x_i \theta_i \]

- **Logistic Neuron**
  
  \[ z = \theta_0 + \sum_i x_i \theta_i \]
  \[ y = \frac{1}{1 + e^{-z}} \]

- **Perceptron**
  
  \[ z = \theta_0 + \sum_i x_i \theta_i \]
  \[ y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases} \]

- Potentially more. Requires a convex loss function for gradient descent training.
Limitation

- A single “neuron” is still a linear decision boundary

- What to do?

- Idea: Stack a bunch of them together!
Nonlinearities via Layers

- Cascade neurons together
- The output from one layer is the input to the next
- Each layer has its own sets of weights

$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$
$$y_{2}(x) = \sigma(\langle w_{2}, y_{1} \rangle)$$

$$y_{1i} = k(x_i, x)$$

Kernels

Deep Nets

optimize all weights
Nonlinearities via Layers

\[ y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle) \]
\[ y_{2i}(x) = \sigma(\langle w_{2i}, y_{1i} \rangle) \]
\[ y_3(x) = \sigma(\langle w_3, y_{2i} \rangle) \]
Representational Power

- Neural network with at least one hidden layer is a universal approximator (can represent any function).
  Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko, paper

- The capacity of the network increases with more hidden units and more hidden layers
A simple example

• Consider a neural network with two layers of neurons.
  - neurons in the top layer represent known shapes.
  - neurons in the bottom layer represent pixel intensities.

• A pixel gets to vote if it has ink on it.
  - Each inked pixel can vote for several different shapes.

• The shape that gets the most votes wins.

\[ f(\sum w_i x_i) \]
How to display the weights

Give each output unit its own “map” of the input image and display the weight coming from each pixel in the location of that pixel in the map.

Use a black or white blob with the area representing the magnitude of the weight and the color representing the sign.
How to learn the weights

Show the network an image and **increment** the weights from active pixels to the correct class.

Then **decrement** the weights from active pixels to whatever class the network guesses.
The image
The image
The image
The learned weights

The details of the learning algorithm will be explained later.
Why insufficient

• A two layer network with a single winner in the top layer is equivalent to having a rigid template for each shape.
  - The winner is the template that has the biggest overlap with the ink.

• The ways in which hand-written digits vary are much too complicated to be captured by simple template matches of whole shapes.
  - To capture all the allowable variations of a digit we need to learn the features that it is composed of.
Multilayer Perceptron

- Layer Representation
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]

- (typically) iterate between linear mapping \( Wx \) and nonlinear function

- Loss function \( l(y, y_i) \) to measure quality of estimate so far
Forward Pass
Forward Pass: What does the Network Compute?

- Output of the network can be written as:
  \[ h_j(x) = f(v_j + \sum_{i=1}^{D} x_i v_{ji}) \]
  \[ o_k(x) = g(w_k + \sum_{j=1}^{J} h_j(x) w_{kj}) \]

(j indexing hidden units, k indexing the output units, D number of inputs)

- Activation functions \( f, g \): sigmoid/logistic, tanh, or rectified linear (ReLU)

\[ \sigma(z) = \frac{1}{1 + \exp(-z)} \]
\[ \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \]
\[ \text{ReLU}(z) = \max(0, z) \]
Forward Pass in Python

- Example code for a forward pass for a 3-layer network in Python:

```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

- Can be implemented efficiently using matrix operations
- Example above: $W_1$ is matrix of size $4 \times 3$, $W_2$ is $4 \times 4$. What about biases and $W_3$?

[http://cs231n.github.io/neural-networks-1/]
Special Case

• What is a single layer (no hiddens) network with a sigmoid act. function?

\[ o_k(x) = \frac{1}{1 + \exp(-z_k)} \]

\[ z_k = w_{k0} + \sum_{j=1}^{J} x_j w_{kj} \]

• Network:

• Logistic regression!
Example

- Classify image of handwritten digit (32x32 pixels): 4 vs non-4

- How would you build your network?
- For example, use one hidden layer and the sigmoid activation function:

\[
o_k(x) = \frac{1}{1 + \exp(-z_k)}
\]

\[
z_k = w_{k0} + \sum_{j=1}^{J} h_j(x)w_{kj}
\]

- How can we train the network, that is, adjust all the parameters \(w\)?
Training Neural Networks

- Find weights:

\[ w^* = \underset{w}{\text{argmin}} \sum_{n=1}^{N} \text{loss}(o^{(n)}, t^{(n)}) \]

where \( o = f(x; w) \) is the output of a neural network

- Define a loss function, e.g.:
  - Squared loss: \( \sum_k \frac{1}{2} (o_k^{(n)} - t_k^{(n)})^2 \)
  - Cross-entropy loss: \( -\sum_k t_k^{(n)} \log o_k^{(n)} \)

- Gradient descent:

\[ w^{t+1} = w^t - \eta \frac{\partial E}{\partial w^t} \]

where \( \eta \) is the learning rate (and \( E \) is error/loss)
## Useful derivatives

<table>
<thead>
<tr>
<th>name</th>
<th>function</th>
<th>derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid</td>
<td>$\sigma(z) = \frac{1}{1+\exp(-z)}$</td>
<td>$\sigma(z) \cdot (1 - \sigma(z))$</td>
</tr>
<tr>
<td>Tanh</td>
<td>$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$</td>
<td>$1/\cosh^2(z)$</td>
</tr>
<tr>
<td>ReLU</td>
<td>$\text{ReLU}(z) = \max(0, z)$</td>
<td>$\begin{cases} 1, &amp; \text{if } z &gt; 0 \ 0, &amp; \text{if } z \leq 0 \end{cases}$</td>
</tr>
</tbody>
</table>
Next lecture: Back-propagation