Topics:
- Regularization
- Neural Networks
- Optimization
- Computing Gradients
Recap from last time
Parametric Approach: Linear Classifier

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

- **Image**
- **Array of 32x32x3 numbers** (3072 numbers total)
- **W** parameters or weights
- **10 numbers giving class scores**

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Error Decomposition

Reality

Modeling Error

Optimization Error

Estimation Error

Multi-class Logistic Regression

model class

Input

Softmax

FC HxWx3
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

\[
\begin{bmatrix}
56 & 231 \\
24 & 2
\end{bmatrix}
\]

Stretch pixels into column

\[
\begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3
\end{bmatrix}
\]

\[
\begin{bmatrix}
56 \\
231 \\
24 \\
2
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.1 \\
3.2 \\
-1.2
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
-96.8 \\
437.9 \\
61.95
\end{bmatrix}
\]

Cat score
Dog score
Ship score

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Linear Classifier: Three Viewpoints

Algebraic Viewpoint

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

Visual Viewpoint

One template per class

Geometric Viewpoint

Hyperplanes cutting up space

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Recall from last time: Linear Classifier

<table>
<thead>
<tr>
<th></th>
<th>airplane</th>
<th>automobile</th>
<th>bird</th>
<th>cat</th>
<th>deer</th>
<th>dog</th>
<th>frog</th>
<th>horse</th>
<th>ship</th>
<th>truck</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-3.45</td>
<td>-0.51</td>
<td>6.04</td>
<td>2.9</td>
<td>4.48</td>
<td>8.02</td>
<td>3.78</td>
<td>1.06</td>
<td>-0.36</td>
<td>-0.72</td>
</tr>
<tr>
<td></td>
<td>-8.87</td>
<td>5.31</td>
<td>6.64</td>
<td>-4.22</td>
<td>-4.19</td>
<td>5.58</td>
<td>4.49</td>
<td>-4.37</td>
<td>-2.09</td>
<td>-2.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

1. Come up with a way of efficiently finding the parameters that minimize the loss function. *(optimization)*

Cat image by Nikita is licensed under CC-BY 2.0; Car image is in the public domain; Frog image is in the public domain
Softmax vs. SVM

\[ L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = W x$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>score</td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td>score</td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Multiclass SVM loss:

Given an example where $x$ is the image and $y$ is the (integer) label, and using the shorthand for the scores vector:

The SVM loss has the form:

"Hinge loss"

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

cat  3.2  1.3  2.2

car  5.1  4.9  2.5

frog -1.7  2.0  -3.1

Multiclass SVM loss:

"Hinge loss"

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Softmax vs. SVM

\[ L_i = -\log\left( \frac{e^{s_{y_i}}}{\sum_{j} e^{s_j}} \right) \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[
s = f(x_i; W)
\]

\[
P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[
s = f(x_i; W)
\]

\[
P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

Probabilities must be \( \geq 0 \)

<table>
<thead>
<tr>
<th>Object</th>
<th>Score</th>
<th>Unnormalized Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>( 24.5 )</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>( 164.0 )</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>( 0.18 )</td>
</tr>
</tbody>
</table>

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[
s = f(x_i; W)
\]

\[
P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

Probabilities must be \( \geq 0 \)

Probabilities must sum to 1

<table>
<thead>
<tr>
<th>Class</th>
<th>Score</th>
<th>Unnormalized Probabilities</th>
<th>Normalized Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>24.5</td>
<td>0.13</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.0</td>
<td>0.87</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
**Softmax Classifier** (Multinomial Logistic Regression)

- **Want to interpret raw classifier scores as probabilities**
  
  $s = f(x_i; W)$
  
  $P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>exp</td>
<td>24.5</td>
<td>164.0</td>
<td>0.18</td>
</tr>
<tr>
<td>norm</td>
<td>0.13</td>
<td>0.87</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Probabilities must be $\geq 0$

Probabilities must sum to 1

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities

---

*Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n*
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as *probabilities*

\[ s = f(x_i; W) \]

\[ P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i|X = x_i) \]

\[
\begin{array}{c|c|c|c}
\text{cat} & 3.2 & 24.5 & 0.13 \\
\text{car} & 5.1 & 164.0 & 0.87 \\
\text{frog} & -1.7 & 0.18 & 0.00 \\
\end{array}
\]

Unnormalized log-probabilities / logits

unnormalized probabilities

probabilities

\[ \text{exp} \]

\[ \text{normalize} \]

\[ L_i = -\log(0.13) = 2.04 \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

Probabilities must be >= 0
Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Maximum Likelihood Estimation
Choose probabilities to maximize the likelihood of the observed data

Unnormalized log-probabilities / logits

exp

normalize

probabilities

Unnormalized
probabilities
Log-Likelihood / KL-Divergence / Cross-Entropy

\[ D = \{(x_i, y_i)\} \quad \text{IID} \sim P_x \]

\[ \hat{W}_{\text{MLE}} = \max \frac{P(CD|w)}{P(x_i|w)} \]

\[ \approx \max \sum_i \log P(y_i|x_i,w) \]
**Softmax Classifier** (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; W) \]

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

Probabilities must be \( \geq 0 \)

Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i | X = x_i) \]

\[ \exp \text{ normalize} \]

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities

Correct probs

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as \textbf{probabilities}

\[ s = f(x_i; W) \]

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i | X = x_i) \]

\text{Kullback–Leibler divergence}

\[ D_{KL}(P || Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)} \]

Correct probs

\begin{tabular}{c|c|c|c|c}
  \textbf{cat} & \textbf{3.2} & \textbf{24.5} & \textbf{0.13} & \textbf{1.00} \\
  \textbf{car} & 5.1 & 164.0 & \textbf{0.87} & 0.00 \\
  \textbf{frog} & -1.7 & \textbf{0.18} & \textbf{0.00} & 0.00 \\
\end{tabular}

Unnormalized log-probabilities / logits

unnormalized probabilities

probabilities

exp

normalize

Compare

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
**Softmax Classifier** (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[
s = f(x_i; W)
\]

\[
P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}
\]

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>-1.7</td>
</tr>
<tr>
<td>exp</td>
<td>24.5</td>
<td>0.18</td>
</tr>
<tr>
<td>\exp</td>
<td>164.0</td>
<td>0.00</td>
</tr>
<tr>
<td>exp</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>prob</td>
<td>0.87</td>
<td></td>
</tr>
<tr>
<td>prob</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>prob</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

- Probabilities must be >= 0
- Probabilities must sum to 1

**Cross Entropy**

\[
H(P, Q) = H(p) + D_{KL}(P \| Q)
\]

-

Correct Probs

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
matrix multiply + bias offset

\[ W = \begin{bmatrix}
0.01 & -0.05 & 0.1 & 0.05 \\
0.7 & 0.2 & 0.05 & 0.16 \\
0.0 & -0.45 & -0.2 & 0.03 \\
\end{bmatrix} \]

\[ b = \begin{bmatrix}
-15 \\
22 \\
-44 \\
56 \\
\end{bmatrix} \]

\[ x_i = \begin{bmatrix}
\end{bmatrix} \]

\[ y_i = 2 \]

hinge loss (SVM)

\[ \max(0, -2.85 - 0.28 + 1) + \max(0, 0.86 - 0.28 + 1) = 1.58 \]

cross-entropy loss (Softmax)

\[ \begin{bmatrix}
-2.85 \\
0.86 \\
0.28 \\
\end{bmatrix} \]

\[ \exp \rightarrow \begin{bmatrix}
0.058 \\
2.36 \\
1.32 \\
\end{bmatrix} \]

\[ \text{normalize to sum to one} \rightarrow \begin{bmatrix}
0.016 \\
0.631 \\
0.353 \\
\end{bmatrix} \]

\[ - \log(0.353) = 0.452 \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Plan for Today

- Regularization
- Neural Networks
- Optimization
- Computing Gradients
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) \]

**Data loss**: Model predictions should match training data

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Regularization

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing too well on training data

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

**Data loss**: Model predictions should match training data

**Regularization**: Prevent the model from doing *too* well on training data

\[ \lambda \] = regularization strength (hyperparameter)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Regularization: Prefer Simpler Models

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Polynomial Regression

\[
\hat{y} = w_0 + w_1 x \\
\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d
\]

\[
[w_0, w_1, \ldots, w_d] = \left(\begin{array}{c}
\phi(x) \\
\phi(x) \\
\vdots
\end{array}\right) = \Phi^T w
\]

\[
D = \{(x_i, y_i)\}_{i=1}^n
\]

\[
w^* = \min_w \frac{1}{2} \sum (y_i - \hat{y}_i)^2
\]
Polynomial Regression
Polynomial Regression

\[ y = \mathbf{w}^T \mathbf{x} \]
Polynomial Regression

• Demo:
  – https://arachnoid.com/polysolve/

• Data:
  – 10 6
  – 15 9
  – 20 11
  – 25 12
  – 29 13
  – 40 11
  – 50 10
  – 60 9
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing too well on training data

\( \lambda \) = regularization strength (hyperparameter)
Regularization

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing too well on training data

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

\( \lambda \) = regularization strength (hyperparameter)

Simple examples

L2 regularization: \( R(W) = \sum_k \sum_l W_{k,l}^2 \)

L1 regularization: \( R(W) = \sum_k \sum_l |W_{k,l}| \)

Elastic net (L1 + L2): \( R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}| \)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing too well on training data

Simple examples

- L2 regularization: \[ R(W) = \sum_k \sum_l W_{k,l}^2 \]
- L1 regularization: \[ R(W) = \sum_k \sum_l |W_{k,l}| \]
- Elastic net (L1 + L2): \[ R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}| \]

More complex:

- Dropout
- Batch normalization
- Stochastic depth, fractional pooling, etc

\( \lambda \) = regularization strength (hyperparameter)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

**Data loss**: Model predictions should match training data

**Regularization**: Prevent the model from doing *too* well on training data

Why regularize?
- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

\[ \hat{y} = w_0 + w_1 x + \ldots \]

*Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n*
- We have some dataset of \((x,y)\)
- We have a **score function**: \(s = f(x; W) = WX\)
- We have a **loss function**:

\[
L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)
\]

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\]

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)
\]

*Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n*
We have some dataset of \((x,y)\)

- We have a **score function**: \(s = f(x; W) = Wx\)

- We have a **loss function**:

\[
L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right)
\]

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\]

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)
\]

**Softmax**

**SVM**

**Full loss**

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Error Decomposition

Error = Model Error + Optimization Error + Estimation Error + Reality

Model class

Multi-class Logistic Regression

Softmax
F HxWx3
Input
Neural networks: without the brain stuff

(Before) Linear score function:

\[ f = Wx \]

\[ f = W_2 W_1 x \]
Neural networks: without the brain stuff

(Before) Linear score function: \[ f = Wx \]

(Now) 2-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]
Neural networks: without the brain stuff

(Before) Linear score function: \( f = Wx \)

(Now) 2-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]

\[
\begin{array}{c}
\text{x} \\
3072
\end{array}
\]

\[
\begin{array}{c}
W_1 \\
100
\end{array}
\]

\[
\begin{array}{c}
h \\
W_2
\end{array}
\]

\[
\begin{array}{c}
s \\
10
\end{array}
\]

\[
K^{10c} \Rightarrow h = \begin{bmatrix}
\end{bmatrix}, \quad s = W_2 \begin{bmatrix}
\end{bmatrix}
\]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Neural networks: without the brain stuff

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$
Neural networks: without the brain stuff

(Before) Linear score function:

\[ f = Wx \]

(Now) 2-layer Neural Network or 3-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]

\[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]
Full implementation of training a 2-layer Neural Network needs ~20 lines:

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
    h = 1.0 / (1 + np.exp(-x.dot(w1)))
    y_pred = h.dot(w2)
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T)
    grad_w1 = x.T.dot(grad_h * h * (1 - h))

    w1 -= 1e-4 * grad_w1
    w2 -= 1e-4 * grad_w2
```
Impulses carried toward cell body

dendrite

Impulses carried away from cell body

axon

cell body

presynaptic terminal

\[ a = \sum_{j} w_j x_j \]

\[ = w^T x \]

\[ y = f(a) \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Impulses carried toward cell body

dendrite

presynaptic terminal

axon

Impulses carried away from cell body

cell body

This image by Felipe Perucho is licensed under CC-BY 3.0

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Impulses carried toward cell body

Impulses carried away from cell body

dendrite

presynaptic terminal

axon

cell body

This image by Felipe Perucho is licensed under CC-BY 3.0

sigmoid activation function

\[
\frac{1}{1 + e^{-x}}
\]

\[f(x) = \frac{1}{1 + e^{-x}}\]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Impulses carried toward cell body

---

Impulses carried away from cell body

dendrite

axon

cell body

dendrite

presynaptic terminal

---

class Neuron:
    # ...
    def neuron_tick(inputs):
    
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
        return firing_rate

---

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Be very careful with your brain analogies!

**Biological Neurons:**
- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Rate code may not be adequate

[Dendritic Computation. London and Hausser]
Activation functions

Sigmoid
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

ReLUs
\[ \text{ReLU}(x) = \max(0, x) \]

Leaky ReLU
\[ \max(0.1x, x) \]

Maxout
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

ELU
\[ \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Activation Functions

- sigmoid vs tanh

\[
\sigma(a) = \frac{1}{1 + e^{-a}}
\]

\[
\text{tanh}(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}
\]

\[
\sigma(\sigma(a)) = 2\sigma(2a) - 1
\]
Fig. 4. (a) Not recommended: the standard logistic function, $f(x) = 1/(1 + e^{-x})$. (b) Hyperbolic tangent, $f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$. 

(C) Dhruv Batra

Image Credit: LeCun et al. ‘98
Rectified Linear Units (ReLU)

\[ f(x) = \max(0, x) \]

[Krizhevsky et al., NIPS12]
Limitation

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

• What to do?

• Idea: Stack a bunch of them together!
Multilayer Networks

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

\[ h_i = \left( w_i^T x \right) \]
Neural networks: Architectures

“2-layer Neural Net”, or “1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or “2-hidden-layer Neural Net”

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Demo Time

- https://playground.tensorflow.org
Optimization
Strategy: Follow the slope

\[ \min_{\theta} L(\theta; D) \]
Strategy: **Follow the slope**

In 1-dimension, the derivative of a function:

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension.

The slope in any direction is the **dot product** of the direction with the gradient. The direction of steepest descent is the **negative gradient**.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Gradient Descent

```
# Vanilla Gradient Descent

while True:
    [weights_grad = evaluate_gradient(loss_fun, data, weights)]
    weights += - step_size * weights_grad # perform parameter update
```

\[ w(0) = \text{init} \]

\[ \text{for } t=1 \ldots \text{tired} \]

\[ w(t+1) = w(t) - \eta \hat{L} \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
negative gradient direction

original W

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n