Lecture 6 Recap
Learning Rate: Implications

- What if too high?
- What if too low?

Training Schedule

Manually specify learning rate for entire training process

• Manually set learning rate every n-epochs
• How?
  – Trial and error (the hard way)
  – Some experience (only generalizes to some degree)

Consider: #epochs, training set size, network size, etc.
Basic Recipe for Training

- Given ground dataset with ground labels
  - \{x_i, y_i\}
    - \(x_i\) is the \(i^{th}\) training image, with label \(y_i\)
    - Often \(\text{dim}(X) \gg \text{dim}(y)\) (e.g., for classification)
    - \(i\) is often in the 100-thousands or millions
  - Take network \(f\) and its parameters \(W, b\)
  - Use SGD (or variation) to find optimal parameters \(W, b\)
    - Gradients from backprop
Basic Recipe for Machine Learning

• Split your data

60% train

20% validation

20% test

Ground truth error ..... 1%
Training set error ..... 5%
Val/test set error ..... 8%

Bias (or underfitting)
Variance (overfitting)

Example scenario

I2DL: Prof. Niessner, Prof. Leal-Taixé
Over and Underfitting

Source: Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017
Over and Underfitting

Source: https://srdas.github.io/DLBook/ImprovingModelGeneralization.html
Hyperparameters

- Network architecture (e.g., num layers, #weights)
- Number of iterations
- Learning rate(s) (i.e., solver parameters, decay, etc.)
- Regularization (more later next lecture)
- Batch size
- ...
- Overall: learning setup + optimization = hyperparameters
Hyperparameter Tuning

• Methods:
  – Manual search: most common 😊
  – Grid search (structured, for ‘real’ applications)
    • Define ranges for all parameters spaces and select points
    • Usually pseudo-uniformly distributed
      → Iterate over all possible configurations
  – Random search:
    Like grid search but one picks points at random in the predefined ranges
Lecture 7
Training NN (part 2)
What we have seen so far

Data points $X$

Model parameters $\theta$

Estimation $\hat{y}$

Optimization

Loss function

Labels (ground truth) $y$

Best practices to obtain good results
Output and Loss Functions
Neural Networks

What is the shape of this function?

Loss

Prediction
Naïve Losses

- **L2 Loss:**
  \[ L^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 \]

- **L1 Loss:**
  \[ L^1 = \sum_{i=1}^{n} |y_i - f(x_i)| \]

Training pairs \([x_i; y_i]\) (input and labels)

\[
\begin{array}{cccc}
12 & 24 & 42 & 23 \\
34 & 32 & 5 & 2 \\
12 & 31 & 12 & 31 \\
31 & 64 & 5 & 13 \\
\end{array}
\]

\[
\begin{array}{cccc}
15 & 20 & 40 & 25 \\
34 & 32 & 5 & 2 \\
12 & 31 & 12 & 31 \\
31 & 64 & 5 & 13 \\
\end{array}
\]

\[
L^2(x, y) = 9 + 16 + 4 + 4 + 0 + \cdots + 0 = 33 \\
L^1(x, y) = 3 + 4 + 2 + 2 + 0 + \cdots + 0 = 11
\]
Naïve Losses: L2 vs L1

- **L2 Loss:**
  \[
  L^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2
  \]
  - Sum of squared differences (SSD)
  - Prone to outliers
  - Compute-efficiency optimization
  - Optimum is the mean

- **L1 Loss:**
  \[
  L^1 = \sum_{i=1}^{n} |y_i - f(x_i)|
  \]
  - Sum of absolute differences
  - Robust (cost of outliers is linear)
  - Costly to optimize
  - Optimum is the median
**Binary Classification: Sigmoid**

training pairs \([x_i; y_i]\), \(x_i \in \mathbb{R}^D, y_i \in \{1, 0\}\) (2 classes)

\[
p(y_i = 1|x_i, \theta) = \sigma(s) = \frac{1}{1 + e^{-\sum_{d=0}^{D} \theta_d x_{id}}}
\]

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

0 < \(\sigma(s)\) < 1

Sigmoid output can be interpreted as a probability

Score of class 1 (\(y_i = 1\))

Probability of class 1 (\(y_i = 1\))
Multiclass Classification: Softmax

training pairs $[\mathbf{x}_i; y_i]$, $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{1, 2, \ldots, C\}$ (C classes)

Score of class $k$ ($y_i = k$)

weights for class $k$ ($y_i = k$)

Probability of class $k$ ($y_i = k$)

$p(y_i = k | \mathbf{x}_i, \theta)$
Multiclass Classification: Softmax

Weights for each class

Scores for each class

Probabilities for each class

\[ p(y_i = 1|\mathbf{x}_i, \Theta) = \frac{e^{x_i\theta_1}}{e^{x_i\theta_1} + e^{x_i\theta_2} + e^{x_i\theta_3}} \]

\[ p(y_i = 2|\mathbf{x}_i, \Theta) = \frac{e^{x_i\theta_2}}{e^{x_i\theta_1} + e^{x_i\theta_2} + e^{x_i\theta_3}} \]

\[ p(y_i = 3|\mathbf{x}_i, \Theta) = \frac{e^{x_i\theta_3}}{e^{x_i\theta_1} + e^{x_i\theta_2} + e^{x_i\theta_3}} \]
Multiclass Classification: Softmax

- **Softmax**

\[
p(y_i | x_i, \Theta) = \frac{e^{s_y_i}}{\sum_{k=1}^{C} e^{s_k}} = \frac{e^{x_i \theta_{y_i}}}{\sum_{k=1}^{C} e^{x_i \theta_k}}
\]

Probability of the true class

- **Parameters:**
  \[
  \Theta = [\theta_1, \theta_2, ..., \theta_C]
  \]

- **C**: number of classes

- **s**: score of the class

1. **Exponential operation**: make sure probability > 0
2. **Normalization**: make sure probabilities sum up to 1.

Training pairs \([x_i; y_i]\), \(x_i \in \mathbb{R}^D, y_i \in \{1, 2, ..., C\}\)

\(y_i\): label (true class)
Multiclass Classification: Softmax

• Numerical Stability

\[ p(y_i | x_i, \Theta) = \frac{e^{s_{yi}}}{\sum_{k=1}^{C} e^{s_k}} = \frac{e^{s_{yi}}}{\sum_{k=1}^{C} e^{s_k - s_{max}}} \]

Try to prove it by yourself 😊

• Cross-Entropy Loss (Maximum Likelihood Estimate)

\[ L_i = -\log(p(y_i | x_i, \Theta)) = -\log\left(\frac{e^{s_{yi}}}{\sum_k e^{s_k}}\right) \]
Example: Cross-Entropy Loss

Cross Entropy

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

Score function

\[ s = f(x_i, \theta) \]
e.g., \( f(x_i, \theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C] \)

Suppose: 3 training examples and 3 classes

<table>
<thead>
<tr>
<th>Scores</th>
<th>cat</th>
<th>chair</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Given a function with weights \( \theta \), training pairs \( [x_i; y_i] \) (input and labels)

\[ \theta_k = \left[ \begin{array}{c} b_k \\ w_k \end{array} \right] \]

parameters for each class with \( C \) classes
Example: Cross-Entropy Loss

Cross Entropy

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

Score function

\[ s = f(x_i, \theta) \]

e.g., \[ f(x_i, \theta) = [x_{i0}, x_{i2}, \ldots , x_{id}] \cdot [\theta_1, \theta_2, \ldots , \theta_C] \]

Suppose: 3 training examples and 3 classes

Given a function with weights \( \theta \), training pairs \([x_i; y_i]\) (input and labels)

\[ \theta_k = [w_k^b] \]

parameters for each class with \( C \) classes

Suppose: 3 training examples and 3 classes

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

Loss \( 2.04 \)
Example: Cross-Entropy Loss

Cross Entropy

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

Score function

\[ s = f(x_i, \Theta) \]
e.g., \[ f(x_i, \Theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C] \]

Suppose: 3 training examples and 3 classes

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

Given a function with weights \( \Theta \), training pairs \([x_i; y_i]\) (input and labels)

\[ \theta_k = [b_k, \mathbf{w}_k] \] parameters for each class with \( C \) classes

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i = \frac{L_1 + L_2 + L_3}{3} \]

\[ = \frac{2.04 + 0.079 + 6.156}{3} = 2.76 \]
Hinge Loss (SVM Loss)

- Score Function \( s = f(x_i, \theta) \)
  - e.g., \( f(x_i, \theta) = [x_{i0}, x_{i2}, ..., x_{id}] \cdot [\theta_1, \theta_2, ..., \theta_C] \)

- Hinge Loss (Multiclass SVM Loss)
  \[
  L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1)
  \]
Example: Hinge Loss (SVM Loss)

Multiclass SVM loss

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

Score function

\[ s = f(x_i, \theta) \]

E.g., \[ f(x_i, \theta) = [x_{i0}, x_{i2}, ..., x_{id}] \cdot [\theta_1, \theta_2, ..., \theta_C] \]

Suppose: 3 training examples and 3 classes

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

Given a function with weights \( \theta \), training pairs \([x_i; y_i]\) (input and labels)

\[ \theta_k = [b_k, w_k] \]

parameters for each class with \( C \) classes
Example: Hinge Loss (SVM Loss)

Multiclass SVM loss \( L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \)

Score function \( s = f(x_i, \theta) \)
e.g., \( f(x_i, \theta) = [x_{i0}, x_{i2}, ..., x_{id}] \cdot [\theta_1, \theta_2, ..., \theta_C] \)

Suppose: 3 training examples and 3 classes

<table>
<thead>
<tr>
<th>Scores</th>
<th>Cat</th>
<th>Chair</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Given a function with weights \( \theta \), training pairs \([x_i; y_i]\) (input and labels)
\( \theta_k = [b_k; w_k] \) parameters for each class with \( C \) classes

\[
L_1 = \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1) \\
= \max(0, 2.9) + \max(0, -3.9) \\
= 2.9 + 0 \\
= 2.9
\]
Example: Hinge Loss (SVM Loss)

Multiclass SVM loss \[ L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \]

Score function \[ s = f(x_i, \theta) \]

e.g., \[ f(x_i, \theta) = [x_{i0}, x_{i2}, \ldots, x_{id}] \cdot [\theta_1, \theta_2, \ldots, \theta_C] \]

Suppose: 3 training examples and 3 classes

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

Given a function with weights \( \theta \), training pairs \([x_i; y_i]\) (input and labels) \( \theta_k = [b_k] \) parameters for each class with \( C \) classes

\[
L_2 = \max(0, 1.3 - 4.9 + 1) + \\
\max(0, 2.0 - 4.9 + 1) \\
= \max(0, -2.6) + \max(0, -1.9) \\
= 0 + 0 = 0
\]
Example: Hinge Loss (SVM Loss)

Multiclass SVM loss: \[ L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \]

Score function: \( s = f(x_i, \theta) \)
e.g., \( f(x_i, \theta) = [x_{i0}, x_{i2}, ..., x_{id}] \cdot [\theta_1, \theta_2, ..., \theta_C] \)

Suppose: 3 training examples and 3 classes

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

Given a function with weights \( \theta \), training pairs \([x_i; y_i]\) (input and labels)
\( \theta_k = [b_k; w_k] \) parameters for each class with \( C \) classes

\[
L_3 = \max(0, 2.2 - (-3.1) + 1) + \\
\max(0, 2.5 - (-3.1) + 1) \\
= \max(0, 6.3) + \max(0, 6.6) \\
= 6.3 + 6.6 \\
= 12.9
\]
Example: Hinge Loss (SVM Loss)

Multiclass SVM loss \( L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \)

Score function \( s = f(x_i, \theta) \)
e.g., \( f(x_i, \theta) = [x_{i0}, x_{i2}, ..., x_{id}] \cdot [\theta_1, \theta_2, ..., \theta_C] \)

Suppose: 3 training examples and 3 classes

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

Given a function with weights \( \theta \), training pairs \([x_i; y_i] \) (input and labels)
\( \theta_k = [b_k \, w_k] \) parameters for each class with \( C \) classes

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i = \frac{L_1 + L_2 + L_3}{3}
\]

\[
= \frac{2.9 + 0 + 12.9}{3} = \frac{5.3}{3}
\]
Multiclass Classification: Hinge vs Cross-Entropy

- Hinge Loss: \( L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \)

- Cross Entropy Loss: \( L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right) \)
### Example: Hinge vs Cross-Entropy Loss

**Hinge Loss:**

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

**Cross Entropy Loss:**

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

For image \( x_i \) (assume \( y_i = 0 \)):

<table>
<thead>
<tr>
<th>Scores</th>
<th>Hinge loss:</th>
<th>Cross Entropy loss:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model 1</strong></td>
<td>( s = [5, -3, 2] )</td>
<td></td>
</tr>
<tr>
<td><strong>Model 2</strong></td>
<td>( s = [5, 10, 10] )</td>
<td></td>
</tr>
<tr>
<td><strong>Model 3</strong></td>
<td>( s = [5, -20, -20] )</td>
<td></td>
</tr>
</tbody>
</table>
**Example: Hinge vs Cross-Entropy**

Hinge Loss: \( L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \)

Cross Entropy: \( L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_k e^{s_k}} \right) \)

For image \( x_i \) (assume \( y_i = 0 \)):

<table>
<thead>
<tr>
<th>Scores</th>
<th>Hinge loss: ( )</th>
<th>Cross Entropy loss:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1 ( s = [5, -3, 2] )</td>
<td>( \max(0, -3 - 5 + 1) ) + ( \max(0, 2 - 5 + 1) = 0 )</td>
<td></td>
</tr>
<tr>
<td>Model 2 ( s = [5, 10, 10] )</td>
<td>( \max(0, 10 - 5 + 1) ) + ( \max(0, 10 - 5 + 1) = 12 )</td>
<td></td>
</tr>
<tr>
<td>Model 3 ( s = [5, -20, -20] )</td>
<td>( \max(0, -20 - 5 + 1) ) + ( \max(0, -20 - 5 + 1) = 0 )</td>
<td></td>
</tr>
</tbody>
</table>

Apparently Model 3 is better, but losses show no difference between Model 1&3, since they all have same loss=0.
**Example: Hinge vs Cross-Entropy**

**Hinge Loss:**  
\[ L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \]

**Cross Entropy:**  
\[ L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right) \]

For image \( x_i \) (assume \( y_i = 0 \)):

<table>
<thead>
<tr>
<th>Scores</th>
<th>Hinge loss: ( \max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0 )</th>
<th>Cross Entropy loss: ( -\ln\left(\frac{e^5}{e^5 + e^3 + e^2}\right) = 0.05 )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model 1</strong></td>
<td>( s = [5, -3, 2] )</td>
<td></td>
</tr>
<tr>
<td><strong>Model 2</strong></td>
<td>( s = [5, 10, 10] )</td>
<td>( \max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12 )</td>
</tr>
<tr>
<td><strong>Model 3</strong></td>
<td>( s = [5, -20, -20] )</td>
<td>( \max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0 )</td>
</tr>
</tbody>
</table>

Model 3 has a clearly smaller loss now.
Example: Hinge vs Cross-Entropy

Hinge Loss: \( L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \)

Cross Entropy: \( L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right) \)

For image \( x_i \) (assume \( y_i = 0 \)):

<table>
<thead>
<tr>
<th>Scores</th>
<th>Hinge loss: ( \max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0 )</th>
<th>Cross Entropy loss: ( -\ln\left(\frac{e^5}{e^5 + e^3 + e^2}\right) = 0.05 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1 ( s = [5, -3, 2] )</td>
<td>( \max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12 )</td>
<td>( -\ln\left(\frac{e^5}{e^5 + e^{10} + e^{10}}\right) = 5.70 )</td>
</tr>
<tr>
<td>Model 2 ( s = [5, 10, 10] )</td>
<td>( \max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0 )</td>
<td>( -\ln\left(\frac{e^5}{e^5 + e^{-20} + e^{-20}}\right) = 2 \times 10^{-11} )</td>
</tr>
<tr>
<td>Model 3 ( s = [5, -20, -20] )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Cross Entropy *always* wants to improve! (loss never 0)
- Hinge Loss saturates.
Loss in Compute Graph

• How do we combine loss functions with weight regularization?

• How to optimize parameters of our networks according to multiple losses?
Loss in Compute Graph

Want to find optimal $\theta$. (weights are unknowns of optimization problem)

- Compute gradient w.r.t. $\theta$.
- Gradient $\nabla_\theta L$ is computed via backpropagation.

$\lambda R^2(W)$

$\frac{1}{N} \sum L_i$

$\lambda R^2(W)$

Data loss

Score function

Loss function

Full Loss

Regularization loss

Input data

$X$

Labels

$y$

$\theta$

$L$
Loss in Compute Graph

• Score function $s = f(x_i, \theta)$

• Data Loss
  - Cross Entropy
    \[ L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right) \]
  - SVM
    \[ L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \]

• Regularization Loss: e.g., L2-Reg: $R^2(W) = \sum w_i^2$

• Full Loss
  \[ L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R^2(W) \]

• Full Loss = Data Loss + Reg Loss

Given a function with weights $\theta$, Training pairs $[x_i; y_i]$ (input and labels)
Example: Regularization & SVM Loss

Multiclass SVM loss $L_i = \sum_{k \neq y_i} \max(0, f(x_i; \theta)_k - f(x_i; \theta)_{y_i} + 1)$

Full loss $L = \frac{1}{N} \sum_{i=1}^{N} \sum_{k \neq y_i} \max(0, f(x_i; \theta)_k - f(x_i; \theta)_{y_i} + 1) + \lambda R(W)$

$L1$-Reg: $R^1(W) = \sum_{i=1}^{D} |w_i|$

$L2$-Reg: $R^2(W) = \sum_{i=1}^{D} w_i^2$

Example:

$x = [1, 1, 1, 1]^T$

$w_1 = [1, 0, 0, 0]^T$

$w_2 = [0.25, 0.25, 0.25, 0.25]^T$

$x^T w_1 = x^T w_2 = 1$

$R^2(w_1) = 1$

$R^2(w_2) = 0.25^2 + 0.25^2 + 0.25^2 + 0.25^2 = 0.25$

$R^2(W) = 1 + 0.25 = 1.25$
Activation Functions
Neural Networks

\[
\begin{align*}
&x_1 \\
&x_2 \\
&\vdots \\
&x_n \\
&+1
\end{align*}
\]

\[
\begin{align*}
&s_1^{(2)} \\
&s_2^{(2)} \\
&\vdots \\
&s_n^{(2)}
\end{align*}
\]

\[
\begin{align*}
&f \\
&f \\
&\vdots \\
&f
\end{align*}
\]

\[
\begin{align*}
&a_1^{(2)} \\
&a_2^{(2)} \\
&\vdots \\
&a_n^{(2)}
\end{align*}
\]

\[
\begin{align*}
&z_1^{(3)} \\
&z_2^{(3)}
\end{align*}
\]
Activation Functions or Hidden Units

\[
\sum \theta
\]

Activation Function
Sigmoid Activation

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

\[ \sigma(s_i) \in (0,1) \]
Sigmoid Activation

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

Forward:

\[ \frac{\partial L}{\partial w} = \frac{\partial s}{\partial w} \frac{\partial L}{\partial s} \]

\[ \chi^T \]

\[ \frac{\partial L}{\partial s} = \frac{\partial \sigma}{\partial s} \frac{\partial L}{\partial \sigma} \]

\[ \frac{\partial \sigma}{\partial s} \]

\[ \frac{\partial L}{\partial \sigma} \]
Sigmoid Activation

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

Forward

\[ \frac{\partial L}{\partial w} = \frac{\partial s}{\partial w} \frac{\partial L}{\partial s} \]

\[ \frac{\partial L}{\partial s} = \frac{\partial \sigma}{\partial s} \frac{\partial L}{\partial \sigma} \]

\[ \frac{\partial \sigma}{\partial s} \]

\[ \frac{\partial L}{\partial \sigma} \]

Saturated neurons kill the gradient flow
Sigmoid Activation

Forward

\[
\frac{\partial L}{\partial w} = \frac{\partial s}{\partial w} \frac{\partial L}{\partial s}
\]

Active region for gradient descent

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

\[
\frac{\partial L}{\partial s} = \frac{\partial \sigma}{\partial s} \frac{\partial L}{\partial \sigma}
\]

\[
\frac{\partial L}{\partial \sigma} = \frac{\partial L}{\partial s} \frac{\partial s}{\partial \sigma}
\]
Sigmoid Activation

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

Output is always positive!

- Sigmoid output provides positive input for the next layer

What is the disadvantage of this?
Sigmoid Output not Zero-centered

- We want to compute the gradient w.r.t. the weights

Assume we have all positive data:
\[ x = (x_1, x_2)^T > 0 \]

It is going to be either positive or negative for all weights' update.

\[
\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_1} \\
\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_2}
\]
Sigmoid Output not Zero-centered

\( w_1, w_2 \) can only be increased or decreased at the same time, which is not good for update.

That is also why you need zero-centered data.

TanH Activation

Still saturates
Zero-centered

[LeCun et al. 1991] Improving Generalization Performance in Character Recognition
Rectified Linear Units (ReLU)

$$\sigma(x) = \max(0, x)$$

Fast convergence

Does not saturate

[Krizhevsky et al. NeurIPS 2012] ImageNet Classification with Deep Convolutional Neural Networks
Rectified Linear Units (ReLU)

- **Dead ReLU**
  - What happens if a ReLU outputs zero?

- **Fast convergence**

- **Does not saturate**

Krizhevsky et al. NeurIPS 2012: ImageNet Classification with Deep Convolutional Neural Networks
Rectified Linear Units (ReLU)

• Initializing ReLU neurons with slightly positive biases (0.01) makes it likely that they stay active for most inputs.

\[ f \left( \sum_i w_i x_i + b \right) \]
Leaky ReLU

\[ \sigma(x) = \max(0.01x, x) \]

Does not die

[Mass et al., ICML 2013] Rectifier Nonlinearities Improve Neural Network Acoustic Models
Parametric ReLU

\[ \sigma(x) = \max(\alpha x, x) \]

One more parameter to backprop into

Does not die

[He et al. ICCV 2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

I2DL: Prof. Niessner, Prof. Leal-Taixé
Maxout Units

\[
\text{Maxout} = \max(w_1^T x + b_1, w_2^T x + b_2)
\]
Maxout Units

Piecewise linear approximation of a convex function with $N$ pieces

[Goodfellow et al. ICML 2013] Maxout Networks
Maxout Units

- Generalization of ReLUs
- Linear regimes
- Does not die
- Increases of the number of parameters
- Does not saturate
In a Nutshell

<table>
<thead>
<tr>
<th>ACTIVATION FUNCTION</th>
<th>EQUATION</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Function</td>
<td>$f(x) = x$</td>
<td>$(-\infty, \infty)$</td>
</tr>
<tr>
<td>Step Function</td>
<td>$f(x) = \begin{cases} 0 &amp; \text{for } x &lt; 0 \ 1 &amp; \text{for } x \geq 0 \end{cases}$</td>
<td>$[0, 1]$</td>
</tr>
<tr>
<td>Sigmoid Function</td>
<td>$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$</td>
<td>$(0, 1)$</td>
</tr>
<tr>
<td>Hyperbolic Tanhant Function</td>
<td>$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$</td>
<td>$(-1, 1)$</td>
</tr>
<tr>
<td>ReLU</td>
<td>$f(x) = \begin{cases} 0 &amp; \text{for } x &lt; 0 \ x &amp; \text{for } x \geq 0 \end{cases}$</td>
<td>$[0, \infty)$</td>
</tr>
<tr>
<td>Leaky ReLU</td>
<td>$f(x) = \begin{cases} 0.01 &amp; \text{for } x &lt; 0 \ x &amp; \text{for } x \geq 0 \end{cases}$</td>
<td>$(-\infty, \infty)$</td>
</tr>
<tr>
<td>Swish Function</td>
<td>$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 &amp; \text{for } f(x) = x \ \beta \to \infty &amp; \text{for } f(x) = 2\max(0, x) \end{cases}$</td>
<td>$(-\infty, \infty)$</td>
</tr>
</tbody>
</table>

Source: https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a
Quick Guide

• Sigmoid is not really used.

• ReLU is the standard choice.

• Second choice are the variants of ReLU or Maxout.

• Recurrent nets will require TanH or similar.
Weight Initialization
How do I start?

Input layer

Hidden layer

Output layer

Forward
Initialization is Extremely Important

\[ x^* = \arg \min f(x) \]

Initialization

Optimum

Not guaranteed to reach the optimum
How do I start?

Forward

\[ f\left(\sum_i w_i x_i + b\right) \]

Input layer

Hidden layer

Output layer

What happens to the gradients?

\( w = 0 \)
All Weights Zero

• What happens to the gradients?

• The hidden units are all going to compute the same function, gradients are going to be the same
  – No symmetry breaking
Small Random Numbers

• Gaussian with zero mean and standard deviation 0.01

• Let’s see what happens:
  – Network with 10 layers with 500 neurons each
  – Tanh as activation functions
  – Input unit Gaussian data
Small Random Numbers

$tanh$ as activation functions

Output become to zero
Small Random Numbers

Small $w_i^l$ cause small output for layer $l$:

$$f_l \left( \sum_i w_i^l x_i^l + b^l \right) \approx 0$$
Even activation function's gradient is ok, we still have vanishing gradient problem.

Small outputs of layer \( l \) (input of layer \( l + 1 \)) cause small gradient w.r.t to the weights of layer \( l + 1 \):

\[
f_{l+1} \left( \sum_i w_i^{l+1} x_i^{l+1} + b^{l+1} \right)
\]

\[
\frac{\partial L}{\partial w_i^{l+1}} = \frac{\partial L}{\partial f_{l+1}} \cdot \frac{\partial f_{l+1}}{\partial w_i^{l+1}} = \frac{\partial L}{\partial f_{l+1}} \cdot x_i^{l+1} \approx 0
\]

Vanishing gradient, caused by small output.
Big Random Numbers

• Gaussian with zero mean and standard deviation 1

• Let us see what happens:
  – Network with 10 layers with 500 neurons each
  – Tanh as activation functions
  – Input unit Gaussian data
Big Random Numbers

tanh as activation functions

Output saturated to -1 and 1
Output saturated to -1 and 1. Gradient of the activation function becomes close to 0.

\[
f(s) = f \left( \sum_i w_i x_i + b \right)
\]

\[
\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial s} \cdot \frac{\partial s}{\partial w_i} \approx 0
\]

Vanishing gradient, caused by saturated activation function.
How to solve this?

- Working on the initialization
- Working on the output generated by each layer
Xavier Initialization

• Gaussian with zero mean, but what standard deviation?

\[ \text{Var}(s) = \text{Var} \left( \sum_{i}^{n} w_i x_i \right) = \sum_{i}^{n} \text{Var}(w_i x_i) \]

Notice: \( n \) is the number of input neurons for the layer of weights you want to initialized. This \( n \) is not the number \( N \) of input data \( X \in \mathbb{R}^{N \times D} \). For the first layer \( n = D \).

Tipps:

\[ E[X^2] = \text{Var}[X] + E[X]^2 \]

If \( X, Y \) are independent:

\[ \text{Var}[XY] = E[X^2Y^2] - E[XY]^2 \]

\[ E[XY] = E[X]E[Y] \]
Xavier Initialization

• Gaussian with zero mean, but what standard deviation?

\[ Var(s) = Var\left(\sum_{i}^{n} w_i x_i\right) = \sum_{i}^{n} Var(w_i x_i) \]

\[ = \sum_{i}^{n} \left[ E(w_i)^2 Var(x_i) + Var(x_i)^2 E(w_i)^2 + \right. \]

\[ \left. Var(x_i)Var(w_i) \right] \]

Independent

Zero mean

Zero mean

[I2DL: Prof. Niessner, Prof. Leal-Taixé]

[Glorot and Bengio, AISTATS'10] Xavier Initialization
Xavier Initialization

• Gaussian with zero mean, but what standard deviation?

\[
\text{Var}(s) = \text{Var} \left( \sum_{i}^{n} w_i x_i \right) = \sum_{i}^{n} \text{Var}(w_i x_i)
\]

\[
= \sum_{i}^{n} \left[ E(w_i) \right]^2 \text{Var}(x_i) + E[(x_i)]^2 \text{Var}(w_i) + \text{Var}(x_i) \text{Var}(w_i)
\]

\[
= \sum_{i}^{n} \text{Var}(x_i) \text{Var}(w_i) = n(\text{Var}(w) \text{Var}(x))
\]

Identically distributed (each random variable has the same distribution)

[Glorot and Bengio, AISTATS'10] Xavier Initialization
**Xavier Initialization**

- How to ensure the variance of the output is the same as the input?

**Goal:**

\[
Var(s) = Var(x) \quad \implies \quad n \cdot Var(w)Var(x) = Var(x) = 1
\]

\[
Var(w) = \frac{1}{n}
\]

\( n \): number of input neurons
Xavier Initialization

\[ \text{Var}(w) = \frac{1}{n} \]

tanh as activation functions
Xavier Initialization with ReLU

\[ \text{Var}(w) = \frac{1}{n} \]

ReLU kills Half of the Data

What's the solution?

When using ReLU, output close to zero again 😞

I2DL: Prof. Niessner, Prof. Leal-Taixé
Xavier/2 Initialization with ReLU

\[ \text{Var}(w) = \frac{1}{n/2} = \frac{2}{n} \]

tanh as activation functions
Xavier/2 Initialization with ReLU

\[ \text{Var}(w) = \frac{2}{n} \]

It makes a huge difference!

- Use ReLU and Xavier/2 initialization

\[ \frac{1}{2} \hat{n}_i \text{Var}[w_i] = 1 \quad \text{others} \]

\[ \hat{n}_i \text{Var}[w_i] = 1 \quad \text{Xavier} \]
### Summary

<table>
<thead>
<tr>
<th>Image Classification</th>
<th>Output Layer</th>
<th>Loss function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Classification</td>
<td>Sigmoid</td>
<td>Binary Cross entropy</td>
</tr>
<tr>
<td>Multiclass Classification</td>
<td>Softmax</td>
<td>Cross entropy</td>
</tr>
</tbody>
</table>

Other Losses:
SVM Loss (Hinge Loss), L1/L2-Loss

Initialization of optimization
- How to set weights at beginning
Next Lecture

- Next lecture
  - More about training neural networks: regularization, dropout, data augmentation, batch normalization, etc.
  - Followed by CNNs
See you next week!
References

  – Chapter 6: Deep Feedforward Networks

• Bishop “Pattern Recognition and Machine Learning” (2006),
  – Chapter 5.5: Regularization in Network Nets

• http://cs231n.github.io/neural-networks-1/

• http://cs231n.github.io/neural-networks-2/