Lab1, Homework 1 and Quiz 1 out today!
Today’s goal – Continue discussion on perceptron and learn about the loss functions

(1) Perceptron learning algorithm

(2) Extending perceptron for Multi-class classification

(3) Loss functions for
   - Regression
   - Classification
Recap: A Binary Perceptron for MNIST

- **Inputs** \([x_1, x_2, \ldots, x_n]\) are all positive
  - \(n = 784\) (28×28 pixel values)
- **Output** is either 0 or 1
  - 0 → input is not the digit type we’re looking for
  - 1 → input is the digit type we’re looking for

![Diagram of a binary perceptron for MNIST](image)
Training a perceptron

0. set the parameters $\Phi = \{wUb\}$ to 0

1. Iterate over training set several times, feeding in each training example into the model, producing an output, and adjusting the parameters according to whether that output was right or wrong

2. Stop once we either (a) get every training example right or (b) after $N$ iterations, a number set by the programmer.

$N$ is known as the number of **epochs**, where each epoch is an iteration of going through all data points in the training set.

As a general rule of thumb, $N$ grows with the number of parameters.

**Training Data**

Target: $Y$

Training Labels

$f$ ("model")

**Output**

**Loss function**

**Optimizer**

**Error**
The Perceptron Learning Algorithm

1. set $w$’s to 0.

2. for $N$ iterations, or until the weights do not change:
   a) for each training example $x^k$ with label $y^k$
      i. if $y^k - f(x^k) = 0$ continue
      ii. else for all weights $w_i$, $\Delta w_i = (y^k - f(x^k)) x_i^k$

• $b =$ bias
• $w =$ weights
• $N =$ maximum number of training iterations
• $x^k =$ $k^{th}$ training example
• $y^k =$ label for the $k^{th}$ example
• $w_i =$ weight for the $i^{th}$ input where $i \leq n$
• $n =$ number of pixels per image
• $x_i^k =$ $i^{th}$ input of the example where $i \leq n$
The Perceptron Learning Algorithm

1. set w’s to 0.
2. for $N$ iterations, or until the weights do not change:
   a) for each training example $x^k$ with label $y^k$
      i. if $y^k - f(x^k) = 0$ continue
      ii. else for all weights $w_i, \Delta w_i = \left(y^k - f(x^k)\right)x_i^k$

- If the output of our model matches the label, we continue
- If the correct label is 1, and our output is 1, $1 - 1 = 0$
- If the correct label is 0, and our output is 0, $0 - 0 = 0$
The Perceptron Learning Algorithm

1. set \( w \)'s to 0.

2. for \( N \) iterations, or until the weights do not change:
   a) for each training example \( x^k \) with label \( y^k \)
      i. if \( y^k - f(x^k) = 0 \) continue
      ii. else for all weights \( w_i, \Delta w_i = (y^k - f(x^k)) x_i^k \)

• If our label \( y^k \) is a 1, and our model’s output is a 0, we update the \( i^{th} \) weight by:
  • \( (1 - 0) \cdot x_i^k = x_i^k \)
  • Output was 0 and should have been 1, so make the output more positive

• If our label \( y^k \) is a 0, and our model’s output is a 1, we update the \( i^{th} \) weight by:
  • \( (0 - 1) \cdot x_i^k = -x_i^k \)
  • Output was 1 and should have been 0, so make the output more negative
Example: Predict whether a digit is a “2”
Predict whether a digit is a “2”

Just look at the effect of these two pixels

\[ x_1 = 0.8 \]
\[ x_2 = 0 \]
Predict whether a digit is a “2”

• Start off training with all parameters as 0, so $w_1 = 0$, $w_2 = 0$, and $b = 0$

• $f(x) = (w_1 \cdot x_1 + w_2 \cdot x_2 + b) \cdot 1$

• $f(x) = (0 \cdot 0.8 + 0 \cdot 0 + 0 \cdot 1) = 0$
  • Return 0 because value is not greater than 0

• Predict that it is not a 2!
• Correct answer: it is a 2...

• Parameter update:
  • $\Delta w_1 = (1 - 0) \cdot 0.8 = 0.8$
  • $\Delta w_2 = (1 - 0) \cdot 0 = 0$
  • $\Delta b = (1 - 0) \cdot 1 = 1$

• Now
  • $w_1 = 0.8$
  • $w_2 = 0$
  • $b = 1$

True label = 1

$x_1 = 0.8$
$x_2 = 0$
Predict whether a digit is a “2”

• Next example:

Remember the starting weights are now:

\[ w_1 = 0.8 \]
\[ w_2 = 0 \]
\[ b = 1 \]

\[ x_1 = 0.9 \]
\[ x_2 = 0.9 \]
Predict whether a digit is a “2”

• At end of last iteration:
  • \( w_1 = 0.8, w_2 = 0, \) and \( b = 1 \)
  • \( f(x) = (w_1 \cdot x_1 + w_2 \cdot x_2 + b) \cdot 1 \)
  • \( f(x) = (0.8 \cdot 0.9 + 0 \cdot 0.9 + 1 \cdot 1) > 0 \)
    • Return 1 because value is greater than 0
  • Predict that it is a 2!
  • Correct answer: it is not a 2...
  • Parameter update:
    • \( \Delta w_1 = (0 - 1) \cdot 0.9 = -0.9 \)
    • \( \Delta w_2 = (0 - 1) \cdot 0.9 = -0.9 \)
    • \( \Delta b = (0 - 1) \cdot 1 = -1 \)
  • Now
    • \( w_1 = 0.8 - 0.9 = -0.1 \)
    • \( w_2 = 0 - 0.9 = -0.9 \)
    • \( b = 1 - 1 = 0 \)

Any questions?

True label = 0

\[ x_1 = 0.9 \]
\[ x_2 = 0.9 \]
Multi-class problem
Bringing back the complexity

Classifying MNIST digits requires predicting 1 of 10 possible values

Pixel Grid

Input: $\mathbf{X}$

Target: $\mathbf{Y}$

Which digit is it?

$\mathbf{x}^{(1)} = \begin{array}{c}
2
\end{array}$

28x28 pixels

Function: $f$

How do we do that?

$\mathbf{y}^{(1)} = “2”$

$f(\mathbf{X}) \rightarrow \mathbf{Y}$

$\mathbf{x}^{(2)} = \begin{array}{c}
0
\end{array}$

Rather than predicting whether a handwritten digit is of a certain class, we predict the class it is most likely in

$\mathbf{y}^{(2)} = “0”$
Using multiple perceptrons

• We can extend perceptrons to multi-class problems by creating $m$ perceptrons, where $m$ = the number of classes

• For MNIST, we would have 10 perceptrons

• Each individual perceptron returns a value, so our model will return the class whose perceptron value is the highest.
  • Here, “perceptron value” refers to the value of the weighted sum before being thresholded.
Using multiple perceptrons

Perceptron for predicting whether handwritten digit is a 0

Perceptron for predicting whether handwritten digit is a 9
Multi-class Perceptron

Three separate perceptrons

Is there anything the Perceptron can’t learn?
AND Function

Perceptrons work well with linearly separable data

\[ w_1 \cdot x_1 + w_2 \cdot x_2 + b > 0 \]

Linear decision boundary
OR Function

Output = 1
Output = 0

\[ w_1 \cdot x_1 + w_2 \cdot x_2 + b > 0 \]
XOR Function

Need **two** linear decision boundaries to represent this function...

Complicated data would need a complicated function!
May see the term Multi-Layer Perceptron (MLP), HOWEVER "perceptrons" are not perceptrons in the strictest possible sense.

We really don’t use the threshold function of a perceptron but still use the linear function.

Any questions?
How do we train multi-layer networks?

• Unfortunately, the perceptron algorithm doesn’t generalize beyond the one-layer case...

• We need a new algorithm...

1. set \( w \)'s to 0.

2. for \( N \) iterations, or until the weights do not change:
   a) for each training example \( x^k \) with label \( a^k \)
      i. if \( a^k - f(x^k) = 0 \) continue
      ii. else for all weights \( w_i \), \( \Delta w_i = (a^k - f(x^k))x_i \)
A critical ingredient for our new approach: Loss functions
A critical ingredient for our new approach: Loss functions
What is a Loss Function?

• A function $L$ which measures how “wrong” a network is.

• $L$ is computed by comparing two values (predicted and true) that shows which is better.

• Evaluate $L$ and update parameters to decrease $L$, making the network “less wrong”.
Recap – regression task

Input: $\mathbf{X}$

- "Temperature"

$\mathbf{X} \in \mathbb{R}$

- $x^{(1)} = 100.1$
- $x^{(2)} = 60.0$
- $x^{(3)} = 30.3$

Target: $\mathbf{Y}$

- "Profit made on selling lemonade"

$\mathbf{Y} \in \mathbb{R}$

- $y^{(1)} = 200.0$
- $y^{(2)} = 160.5$
- $y^{(3)} = 145.1$

Function: $f$

$f(\mathbf{X}) \rightarrow \mathbf{Y}$

(Image only for explaining concept, not drawn accurately)
Recap: Learning function \( f \)

**Input:** \( \mathbf{X} \)

- "Temperature"

\[
\begin{align*}
\mathbf{X} &\in \mathbb{R} \\
\mathbf{X}^{(1)} &= 100.1 \\
\mathbf{X}^{(2)} &= 60.0 \\
\mathbf{X}^{(3)} &= 30.3 \\
\end{align*}
\]

**Target:** \( \mathbf{Y} \)

- "Profit made on selling lemonade"

\[
\begin{align*}
\mathbf{Y} &\in \mathbb{R} \\
\mathbf{y}^{(1)} &= 200.0 \\
\mathbf{y}^{(2)} &= 160.5 \\
\mathbf{y}^{(3)} &= 145.1 \\
\end{align*}
\] (Numerical output)

**What could be our loss function?**

**Linear function**

\[
y = wx + b
\]

**Image only for explaining concept, not drawn accurately**
Mean Squared Error (MSE)

Average squared residual (residual: difference between predicted and true value)

Decreasing the MSE = the model has less error = data points fall closer to the regression line

$$MSE = \frac{\sum_{k=1}^{n} (y^k - \hat{y}^k)^2}{n}$$

$y^k$: true output value

$\hat{y}^k$: predicted output value

$n$: number of samples

MSE is the average squared distance between the observed and predicted values

What could be the purpose of squaring the distance?

Courtesy: https://statisticsbyjim.com/regression/mean-squared-error-mse/
Mean Squared Error (MSE)

Average squared residual (residual: difference between predicted and true value)

Decreasing the MSE = the model has less error = data points fall closer to the regression line

\[ MSE = \frac{\sum_{k=1}^{n} (y^k - \hat{y}^k)^2}{n} \]

- \( y^k \): true output value
- \( \hat{y}^k \): predicted output value
- \( n \): number of samples

What could be the purpose of squaring the distance?

Courtesy: https://statisticsbyjim.com/regression/mean-squared-error-mse/
Recap: Binary classification

Input: \( \mathbf{X} \)

Pixel Grid

\[ x^{(1)} = 2 \]

28x28 pixels

Target: \( \mathbf{Y} \)

Function: \( f \)

Is it digit 2?

\[ y^{(1)} = "1" \]

\[ y^{(2)} = "0" \]

1. Make the network output a probability for class 1 (a value between 0 and 1)

2. Use this probability to compute a loss

What is a good loss for our binary classification?
Cross Entropy Loss (for Binary classification)

\[ y = \text{true label of class } (0 \text{ or } 1) \]
\[ p = \text{predicted probability of class } 1 \]

\[ \log(p) \]

Some examples:
\[ \log(0.9) = -0.04 \]
\[ \log(0.5) = -0.3 \]
\[ \log(0.001) = -3 \]
Cross Entropy Loss (for Binary classification)

\[ y = \text{true label of class } (0 \text{ or } 1) \]
\[ p = \text{predicted probability of class } 1 \]

\[ -\log (p) \]

Some examples:
\[ \log (0.9) = -0.04 \]
\[ \log (0.5) = -0.3 \]
\[ \log (0.001) = -3 \]
Cross Entropy Loss (for Binary classification)

\[ y = \text{true label of class (0 or 1)} \]
\[ p = \text{predicted probability of class 1} \]

\[-(y \log (p))\]

Some examples:
\[ \log (0.9) = -0.04 \]
\[ \log (0.5) = -0.3 \]
\[ \log (0.001) = -3 \]
Cross Entropy Loss (for Binary classification)

\[ y = \text{true label of class (0 or 1)} \]
\[ p = \text{predicted probability of class 1} \]

\[ -(y \log(p) + (1-y) \log(1-p)) \]

- \( y = 1, p = 0.9 \)
- \( y = 0, p = 0.9 \)
- \( y = 1, p = 0.001 \)
- \( y = 0, p = 0.001 \)

Some examples:
log (0.9) = −0.04
log (0.5) = −0.3
log (0.001) = −3

We get this probability by using a Sigmoid function

Any questions?
Recap: Multi-class classification

Input: $\mathbf{X}$

Pixel Grid

28x28 pixels

$\mathbf{x}^{(1)} = \begin{array}{c}
2 \\
end{array}$

$\mathbf{x}^{(2)} = \begin{array}{c}
0 \\
end{array}$

Function: $f$

We want our network to produce the right answer with high probability

Target: $\mathbf{Y}$

Which digit is it?

$\mathbf{y}^{(1)} = “2”$  

1. Make the network output probabilities for all classes (values between 0 and 1)

$\mathbf{y}^{(2)} = “0”$  

2. Use these probabilities to compute a loss
Cross Entropy Loss (for Multi-class classification)

\[ - \sum_{j=1}^{m} y_j \log(p_j) \]

<table>
<thead>
<tr>
<th>p</th>
<th>Classes (m)</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>“0”</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>“1”</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>“2”</td>
<td>1</td>
</tr>
</tbody>
</table>

Some examples:
\[
\log (0.9) = -0.04 \\
\log (0.5) = -0.3 \\
\log (0.001) = -3
\]

We want model to assign high probability to the true class and low to others.
Recap

Perceptron training w/ working example

Multi-class classification

When perceptron fails

MSE loss for regression

Cross entropy loss for binary classification

Cross entropy loss for multi-class classification

Loss functions
Some Trivia: The Fall of Perceptrons

• In 1969, Marvin Minsky and Seymour Papert released a book, *Perceptrons*, demonstrating that perceptrons are not able to learn the XOR function

• Many earlier researchers heavily focused on logical reasoning, a feature of high-level human cognition, so a machine’s ability for logical reasoning was thought to indicate “artificial intelligence”

• Part of a funding battle: Minksy and Papert wanted federal AI funding to go to their kind of ‘symbolic’ AI, not the early neural net folks...