Artificial Neural Networks: Perceptron

Mehrdad Mahdavi
CMPSC 448
Spring 2023
Outline

- The binary classification problem
- Linear classifiers and their decision boundaries
- The Perceptron algorithm (an online mistake driven learning algorithm)
- Convergence of Perceptron
- Performance evaluation
Recommended reading

Chapter 4: The Perceptron
Abstract binary classification

In binary classification the label set is binary, i.e., \( \mathcal{Y} = \{-1, +1\} \)

Examples:
- Cancer diagnostics
- Credit approval or denial
- ...
The key questions

- **How do we choose a hypothesis space?**
  - Often we use prior knowledge to guide this choice
  - The ability to answer to the next two questions also affects choice

- **How can we gauge the accuracy of a hypothesis on unseen testing data?**
  - How to evaluate a model? What loss function we should use in training?
    - We know that choosing the hypothesis which simply minimizes training set error (i.e., empirical risk minimization) is not optimal
    - This question is the main topic of learning theory
    - We have to use regularization to avoid overfitting and use cross-validation to tune the hyper-parameters

- **How do we find the best hypothesis?**
  - This is an algorithmic question, at the intersection of computer science and optimization research
Linear classifiers

- Consider the credit approval or denial problem (binary classification)

- **Input features** vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^\top$

- Give **importance weights** to the different features and compute a ‘**credit score**’

$$\text{credit score} = \sum_{i=1}^{d} w_i x_i$$

- How to choose importance weights?

  - Feature $x_i$ is important $\quad \rightarrow \quad |w_i|$
  - Feature $x_i$ is beneficial for credit $\quad \rightarrow \quad w_i > 0$
  - Feature $x_i$ is detrimental for credit $\quad \rightarrow \quad w_i < 0$
  - Feature $x_i$ is not important for credit $\quad \rightarrow \quad w_i = 0$
From scores to binary labels

Approve credit if
\[ \sum_{i=1}^{d} w_i x_i > \text{threshold} \rightarrow y = +1 \]

Deny credit if
\[ \sum_{i=1}^{d} w_i x_i \leq \text{threshold} \rightarrow y = -1 \]
From scores to binary labels

Approve credit if
\[
\sum_{i=1}^{d} w_i x_i > \text{threshold} \implies y = +1
\]

Deny credit if
\[
\sum_{i=1}^{d} w_i x_i \leq \text{threshold} \implies y = -1
\]

Can be written formally as the following prediction function:
\[
h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + b \right)
\]

The “bias (intercept) weight” \( b \) corresponds to the threshold (how?)
The geometry of linear classifiers

Consider an arbitrary hyperplane with normal vector $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ as our classifier $h$

In other words, our classifier is parametrized by vector which corresponds to a hyperplane
What would be the binary prediction of \( w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \) for following three vectors?
The geometry of linear classifiers

\[ \mathbf{w}^\top \mathbf{x}_1 = 0 \]

\[ \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \]

\[ \mathbf{w}^\top \mathbf{x}_3 > 0 \]

\[ \mathbf{w}^\top \mathbf{x}_2 < 0 \]

\[ \mathbf{a}^\top \mathbf{b} = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos(\theta) \]
The geometry of linear classifiers

For samples on decision boundary, linear classifier can not decide, just flip a coin!

Every hyperplane (linear classifier) divides the space in half, half + and half -

$$\mathbf{w}^\top \mathbf{x} + b = 0$$

$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

$$\mathbf{w}^\top \mathbf{x} + b < 0$$

$$\mathbf{w}^\top \mathbf{x} + b > 0$$

Compare to the **non-linear complex** decision boundary of nearest neighbors!
Let’s assume you find the best parameter and come up with the following binary linear classifier:

\[ h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + b \right) \]

How can we evaluate the classifier on \( S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \)?
Evaluation

Let’s assume you find the best parameter and come up with following binary linear classifier:

\[ h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + b \right) \]

How can we evaluate the classifier on \( S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \)?

\[
\frac{\text{# of } h(x_i) \neq y_i}{n} = \frac{\text{# of examples that } \text{sign}(w^\top x_i + b) \neq y_i}{n}
\]
Let’s assume you find the best parameter and come up with following binary linear classifier:

\[ h(\boldsymbol{x}) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + b \right) \]

How can we evaluate the classifier on \( S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \)?

\[
\frac{\text{\# of } h(x_i) \neq y_i}{n} = \frac{\# \text{ of examples that } \text{sign} (\mathbf{w}^\top \mathbf{x}_i + b) \neq y_i}{n}
\]

The correct prediction of classifier on a training sample can also be written as:

\[
\text{sign} \left( \mathbf{w}^\top \mathbf{x}_i + b \right) = y_i \iff y_i \left( \mathbf{w}^\top \mathbf{x}_i + b \right) \geq 0
\]
The zero-one loss function

- The loss of classifier $h$ with parameters $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$ on training example $(\mathbf{x}, y)$ is:

$$
\ell(\mathbf{w}, b; (\mathbf{x}, y)) = \begin{cases} 
0 & h(\mathbf{x}) = \text{sign} \left( \mathbf{w}^\top \mathbf{x} + b \right) = y \\
1 & h(\mathbf{x}) = \text{sign} \left( \mathbf{w}^\top \mathbf{x} + b \right) \neq y
\end{cases}
$$
The zero-one loss function

- The loss of classifier $h$ with parameters $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$ on training example $(\mathbf{x}, y)$ is:

$$\ell(\mathbf{w}, b; (\mathbf{x}, y)) = \begin{cases} 
0 & h(\mathbf{x}) = \text{sign} (\mathbf{w}^\top \mathbf{x} + b) = y \\
1 & h(\mathbf{x}) = \text{sign} (\mathbf{w}^\top \mathbf{x} + b) \neq y 
\end{cases}$$

The above loss function is called the 0/1 loss:
Binary classification task

Given a separable binary labeled training data:

\[ S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \]

Hypothesis (model space)

[we will drop the intercept by adding a constant 1 feature to all training examples]

\[ \mathbf{w} \in \mathbb{R}^d, \quad b \in \mathbb{R} \]

Prediction

\[ h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + b \right) \]

Goal: find a hyperplane that minimizes # of mis-classifications over \( S \) (loss function):

\[ \ell(\mathbf{w}, b; (x_i, y_i)) = \begin{cases} 
0 & \text{sign} (\mathbf{w}^\top x_i + b) = y_i \\
1 & \text{sign} (\mathbf{w}^\top x_i + b) \neq y_i 
\end{cases} \]
The Perceptron algorithm

- 1957: Perceptron algorithm invented by Rosenblatt
- 1960: Perceptron Mark I Computer – hardware implementation (using hardware circuits)
- 1969: Minksky & Papert book shows perceptrons limited to linearly separable data, and Rosenblatt dies in boating accident
- 2006+: Deep neural networks by many layers of Perceptron with some tweaks
The neurons are connected to one another with the use of axons and dendrites, and the connecting regions between axons and dendrites are referred to as synapses.

Each input to a neuron is scaled with a weight, which affects the function computed at that unit. The neuron is fired (activated) when the input is beyond a threshold.

An artificial neural network computes a function of the inputs by propagating the computed values from the input neurons to the output neuron(s) and using the weights as intermediate parameters.

The training data provides feedback to the correctness of the weights in the neural network depending on how well the predicted output (e.g., probability of credit approval) for a particular input matches the annotated output label in the training data.

Learning occurs by changing the weights connecting the neurons.
Linear classifier as an artificial neuron

Perceptron is a simple neural network (a single input layer and an output layer) with **sign activation** function!

\[
\sum_{i=1}^{d} w_i x_i + b
\]
The Perceptron algorithm

- **Input** \( S = \{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \} \)
- **Initialize** \( w_0 = 0 \)
- **While** not converged (iteration \( t = 1, 2, \ldots, T \)):
  
  **Find** an (BAD) example \((x_t, y_t) \in S\) **misclassified** by current solution,
The Perceptron Algorithm

- **Input** \( S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)
- **Initialize** \( w_0 = 0 \)
- **While** not converged (iteration \( t = 1, 2, \ldots, T \)):
  
  Find an (**BAD**) example \( (x_t, y_t) \in S \) **misclassified** by current solution \( w_t \)

\[
\text{sign} \left( w_t^\top x_t \right) \neq y_t
\]

Our current prediction vs True label
The Perceptron algorithm

- **Input** $S = \{ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \}$
- **Initialize** $w_0 = 0$
- **While** not converged (iteration $t = 1, 2, \ldots, T$):

  Find an **BAD** example $(x_t, y_t) \in S$ **misclassified** by current solution $w_t$

  \[
  \text{sign} \left( w_t^T x_t \right) \neq y_t
  \]

  Our current prediction \hspace{1cm} \text{True label}

  **Update:**

  \[
  w_{t+1} = w_t + \eta y_t x_t
  \]

  The learning rate

  - For **separable** data it is **guaranteed** to converge to a **solution**.
    (separable means there exists a hyperplane that can correctly classify training data, our assumption)
Geometric interpretation
Geometric interpretation
Geometric interpretation
Geometric interpretation
Geometric interpretation
Geometric interpretation
Note that we always make some progress but it is not guaranteed that new model correctly classifies the BAD example!
Before and after a single update

$w_{t+1}$

$w_t$
Before and after a single update
How much progress?

Prediction by updated model

\[ y_t \left( w_{t+1}^\top x_t \right) = y_t \left( w_t + \eta y_t x_t \right)^\top x_t \]

\[ = y_t w_t^\top x_t + \eta y_t y_t x_t^\top x_t \]

\[ = y_t w_t^\top x_t + \eta y_t y_t \|x_t\|^2_2 \]

\[ > y_t \left( w_t^\top x_t \right) \]

A positive term is added

Prediction by previous model
Does Perceptron work?

**Theorem (convergence of Perceptron).** If the data can be fit by a linear separator (i.e., the training data is linearly separable), the Perceptron algorithm will find one, after the finite number of iterations bounded by:

\[ \left( \frac{R}{\gamma} \right)^2 \]

**Note.** This basically shows the maximum number of mistakes the Perceptron algorithm makes to find a classifier that perfectly separates the training data (the training data is linearly separable).

Perceptron can be considered as an online mistake driven learning algorithm!
Does Perceptron work?

**Theorem (convergence of Perceptron).**

If the data can be fit by a linear separator (i.e., the training data is linearly separable), the Perceptron algorithm will find one, after the finite number of iterations bounded by:

\[
\left( \frac{R}{\gamma} \right)^2
\]

\(R\) is the maximum length of training examples

\(\gamma\) is the distance of closest training sample to the optimal classifier (known as the margin of solution)

\[
R = \max_{i=1,2,\ldots,n} \|x_i\|_2
\]

is the maximum length of training examples
Useful facts about Perceptron

- The scale of the weight vector is irrelevant from the perspective of classification!
  (Why? Only side of the hyperplane matters)

  \[ h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + b \right) \]

- The learning rate does not affect the convergence rate (\# of mistake made) of Perceptron!
  Why? Check the proof of convergence in Appendix.

- The \( w^T x \) is just the distance of \( x \) from the origins when projected onto the vector \( w \)

- The role of the bias (intercept) \( b \) is to shift the decision boundary away from the origin, in the direction of \( w \) (the normal vector of hyperplane)

- Sort all the weights from largest (positive) to largest (negative), and take the top and bottom features, e.g., say ten. The top ten are the features that the Perceptron is most sensitive to for making positive predictions. The bottom ten are the features that the Perceptron is most sensitive to for making negative predictions
The Perceptron updating rule is similar to **stochastic gradient descent** we saw before:

\[ w_{t+1} = w_t + \eta y_t x_t = \begin{cases} w_t + \eta x_t & y_t = +1 \\ w_t - \eta x_t & y_t = -1 \end{cases} \]

What **loss function** the Perceptron is trying to optimize?
What loss function the Perceptron is optimizing?

The Perceptron updating rule is similar to stochastic gradient descent we saw before:

\[ w_{t+1} = w_t + \eta y_t x_t = \begin{cases} 
  w_t + \eta x_t & y_t = +1 \\
  w_t - \eta x_t & y_t = -1 
\end{cases} \]

What loss function the Perceptron is trying to optimize?

The loss of classifier on \((x_i, y_i)\) is measured by the following function:

\[ \max\{0, -y_i \left( w^\top x_i \right) \} \]

This objective function is defined by dropping the sign function in the 0/1 loss and setting negative values to 0 in order to treat all correct predictions in a uniform and lossless way.

Does SGD on above loss function gives the Perceptron updating rule?
In Perceptron, the loss function we use in training:

$$\max\{0, -y (w^\top x)\}$$

The evaluation metric we use in evaluating the accuracy on test data [omitting the intercept $b$]:

$$\begin{cases} 
0 & \text{sign} \left( w^\top x \right) = y \\
1 & \text{sign} \left( w^\top x \right) \neq y
\end{cases}$$

OR

$$\begin{cases} 
0 & y \left( w^\top x \right) \geq 0 \\
1 & y \left( w^\top x \right) < 0
\end{cases}$$
Perceptron loss versus 0-1 loss

Why don’t we directly minimize training error (empirical risk) with respect to the zero-one loss?

The reason is that the training error with the zero-one loss is computationally difficult to optimize directly. In fact, this optimization problem is NP-hard even for linear prediction rules. For example SGD fails entirely on the zero-one loss objective.

This why we usually optimize a convex surrogate loss of 0-1 loss:

We will see more example of convex surrogate losses in future lectures (Hinge, Logistic, etc).
Perceptron as a linear programming

Recall when the training data is separable data it is guaranteed to converge to a solution.

Any hyperplane that can perfectly classify the training data is a potential solution of Perceptron!

\[
\begin{align*}
\min_{w,b} & \quad 0 \\
\text{such that} \quad y_i (w^\top x_i + b) & \geq 0, \quad i = 1, 2, \ldots, n
\end{align*}
\]
The Perceptron algorithm initially gave a huge wave of excitement ("digital brains")

Then, contributed to the AI Winter. Famous counter-example XOR problem (Minsky 1969):

The key issues: if data is not linearly separable, it loops forever.

We can map data to a higher dimensional space and hopefully the data will be separable!

Make the model space richer (reduce the approximation error (bias)) by adding layers to Perceptron!
# Overfitting and underfitting in classification

<table>
<thead>
<tr>
<th>Symptoms</th>
<th>Underfitting</th>
<th>Just right</th>
<th>Overfitting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• High training error</td>
<td>• Training error slightly lower than test error</td>
<td>• Very low training error</td>
</tr>
<tr>
<td></td>
<td>• Training error close to test error</td>
<td></td>
<td>• Training error much lower than test error</td>
</tr>
<tr>
<td></td>
<td>• High bias</td>
<td></td>
<td>• High variance</td>
</tr>
</tbody>
</table>

### Regression illustration

- **Underfitting**: Linear model that doesn't capture the trend in the data.
- **Just right**: Non-linear model that captures the trend in the data.
- **Overfitting**: Model that captures noise in the data.

### Classification illustration

- **Underfitting**: Linear separator that separates the classes with a large margin.
- **Just right**: Non-linear separator that precisely separates the classes.
- **Overfitting**: Model that overfits the training data, capturing noise and generalizing poorly to new data.

Training loss versus evaluation metric

In general, the loss function used in training might **NOT** be the same as the evaluation metric!

E.g., consider the following imbalanced data! (in **cancer** diagnostics applications)

The predictor which always predicts □ has small training error \( \frac{3}{n} \) but obviously terrible test error if we use 0/1 for both training loss and evaluation metric

If evaluation metric is different from training loss, we need to tweak the learning process to minimize the discrepancy and entail good generalization, e.g.,:

- Directly incorporate evaluation metric is training loss
- Undersampling the majority class
- Use weighted training loss where we assign different weights to different training samples
Binary predictions can be right or wrong in four different ways summarized by the confusion matrix:

<table>
<thead>
<tr>
<th>Predicted Label</th>
<th>True Label</th>
<th>True Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} = 0$</td>
<td>$y = 0$</td>
<td>$y = 1$</td>
</tr>
<tr>
<td>True negative (TN)</td>
<td>False negative (FN) (Type 2 Error)</td>
<td></td>
</tr>
<tr>
<td>$\hat{y} = 1$</td>
<td>$y = 0$</td>
<td>$y = 1$</td>
</tr>
<tr>
<td>False positive (FP) (Type 1 Error)</td>
<td>True positive (TP)</td>
<td></td>
</tr>
</tbody>
</table>

**TN**: predicted correctly as class 0 (negative label - )

**TP**: predicted correctly as class 1 (positive label + )

**FN**: predicted incorrectly as class 0

**FP**: predicted incorrectly as class 1
There are other quantities that are also of interest in statistics and machine learning:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>( \frac{TP + TN}{TP + TN + FP + FN} )</td>
<td>Overall performance of model</td>
</tr>
<tr>
<td>Precision</td>
<td>( \frac{TP}{TP + FP} )</td>
<td>How accurate the positive predictions are</td>
</tr>
<tr>
<td>Recall / Sensitivity</td>
<td>( \frac{TP}{TP + FN} )</td>
<td>Coverage of actual positive sample</td>
</tr>
<tr>
<td>Specificity</td>
<td>( \frac{TN}{TN + FP} )</td>
<td>Coverage of actual negative sample</td>
</tr>
<tr>
<td>F1 score</td>
<td>( \frac{2TP}{2TP + FP + FN} )</td>
<td>Hybrid metric useful for unbalanced classes</td>
</tr>
</tbody>
</table>
Example in information retrieval

- Consider a database of 100,000 documents.
- Query for machine learning returns 200 documents.
- 100 of them are actually about machine learning.
- 50 documents about machine learning were not returned.

<table>
<thead>
<tr>
<th>True Label</th>
<th>Predicted Label</th>
<th>$y = 0$</th>
<th>$y = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} = 0$</td>
<td>True negative (TN)</td>
<td>99,750</td>
<td>False negative (FN)</td>
</tr>
<tr>
<td>$\hat{y} = 1$</td>
<td>False positive (FP)</td>
<td>100</td>
<td>True positive (TP)</td>
</tr>
</tbody>
</table>
Example in information retrieval

The precision is the accuracy of the + predictions: $TP / (TP + FP) = 100/200 = 50\%$

The recall is the accuracy of the positive class: $TP/(TP+FN) = 100/(100+50) = 67\%$

<table>
<thead>
<tr>
<th>True Label</th>
<th>$y = 0$</th>
<th>$y = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} = 0$</td>
<td>True negative (TN)</td>
<td>False negative (FN)</td>
</tr>
<tr>
<td></td>
<td>99,750</td>
<td>50</td>
</tr>
<tr>
<td>$\hat{y} = 1$</td>
<td>False positive (FP)</td>
<td>True positive (TP)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
True and false positive rate lead to another fundamental notion, called the receiver operating characteristic (ROC) curve.

The area under the receiving operating curve is called AUC!
Connectionism versus symbolism

Two school of thoughts in artificial intelligence (AI)

**Symbolism**: AI can be achieved by representing concepts as symbols

- Example: rule-based expert system, formal grammar

**Connectionism**: explain intellectual abilities using connections between neurons (i.e., artificial neural networks)

- Example: perceptron, larger scale neural networks

If Other-Delinquent-Accounts > 2, and
Number-Delinquent-Billing-Cycles > 1
Then Profitable-Customer? = No
[Deny Credit Card application]

If Other-Delinquent-Accounts = 0, and
(Income > $30k) OR (Years-of-Credit > 3)
Then Profitable-Customer? = Yes
[Accept Credit Card application]

Example from Machine learning lecture notes by Tom Mitchell
Perceptron can be seen as SGD with the loss function \( \max\{0, -y (\mathbf{w}^\top \mathbf{x})\} \)

\[
\mathbf{w}_0 = 0 \\
M = 0 \quad \text{[number of mistakes]} \\
t = 1 \quad \text{[number of iterations]}
\]

while \( \exists \) a sample misclassified do:

Randomly pick a training sample \( (\mathbf{x}_t, y_t) \)

If the sample is misclassified, \( y_t (\mathbf{w}_t^\top \mathbf{x}_t) < 0 \) update:

\[
\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t \\
M \leftarrow M + 1 \\
t \leftarrow t + 1
\]

end for

return \( \mathbf{w}_t \)
Proof of convergence

**Lemma 1**
After running the Perceptron algorithm for $T$ iterations, for final solution $\mathbf{w}_T$ it holds that:

$$\mathbf{w}_T^\top \mathbf{w}^* \geq \eta \gamma T$$

**Lemma 2**
For the final solution we have:

$$\|\mathbf{w}_T\|_2 \leq \eta^2 T R^2$$
Proof of Theorem 1

From Lemma 1 and assumption that \( \|w^*\|_2 = 1 \)
we have
\[
\eta Y^T < w^T w^* < \|w_T\|_2 \|w^*\|_2 = \|w\|
\]

Combining with Lemma 2, we have
\[
\eta Y^T < \|w_T\|_2 \leq \eta R \sqrt{T}
\]

\[
\Rightarrow \sqrt{T} \leq \frac{R}{\eta} \Rightarrow T \leq \left(\frac{R}{\eta}\right)^2
\]
Proof of Lemma 1

We have:

\[ w_{t+1} = w_t + \eta y_t x_t^T w_* \]

\[ w_{t+1} = w_t + \eta \begin{pmatrix} y_t^T & x_t^T \end{pmatrix} w_* \]

\[ w_* \text{ separates data with margin} \]

\[ w_t w_* + \eta y \]

\[ \Rightarrow w_t w_* \Rightarrow w_* \eta y \]

\[ \Rightarrow w_t w_* + \eta \gamma \]

\[ \Rightarrow w_t w_* + \eta y T \]

\[ \Rightarrow \begin{pmatrix} w_t^T & w_* \end{pmatrix} \eta y T \]
Proof of Lemma 2

We have:

\[\|w_{t+1}\|^2 = \|w_t + \eta y_t x_t\|^2\]

\[= \|w_t\|^2 + \eta^2 \|x_t\|^2 + 2\eta y_t \langle w_t, x_t \rangle\]

\[\leq \|w_t\|^2 + \eta R^2\]

Summing up for iterations,

\[\|w_T\|^2 \leq \|w_0\|^2 + \eta \sum_{t=1}^{T} R^2\]

\[\leq \|w_0\|^2 + \eta R^2 + \eta^2 R^2 + \cdots + \eta^T R^2\]

\[\Rightarrow \|w_T\|^2 \leq \eta R^2 \cdot T\]