CMPSC 448: Machine Learning

Lecture 7. Perceptrons and Deep Neural Networks

Rui Zhang

Fall 2021

PennState

Outline

- Linear classifiers and their decision boundaries
- Perceptrons
- Neural Networks and Back-propagation
- A shallow look at deep neural networks (DNNs)

Binary classification

In binary classification the label set is binary.

In this lecture, let us use

$$\mathcal{Y} = \{-1, +1\}$$

Linear classifiers

Consider the credit approval or denial problem (binary classification)

Input features vector

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^\top$$

Give importance weights to the different features and compute a "credit score"

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

Linear classifiers

Consider the credit approval or denial problem (binary classification)

Input features vector

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^\top$$

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 $\longrightarrow$ $|w_i|$

Feature x_i is beneficial for credit $\longrightarrow$ $w_i > 0$

Feature x_i is detrimental for credit $\longrightarrow$ $w_i < 0$

Feature x_i is not important for credit $\longrightarrow$ $w_i = 0$

From scores to binary labels

Approve credit if $\sum_{i=1}^{d} w_i x_i > \text{threshold} \longrightarrow y = +1$

Deny credit if $\sum_{i=1}^{d} w_i x_i \leq \text{threshold} \longrightarrow y = -1$

From scores to binary labels

Approve credit if $\sum_{i=1}^{d} w_i x_i > \text{threshold} \longrightarrow y = +1$

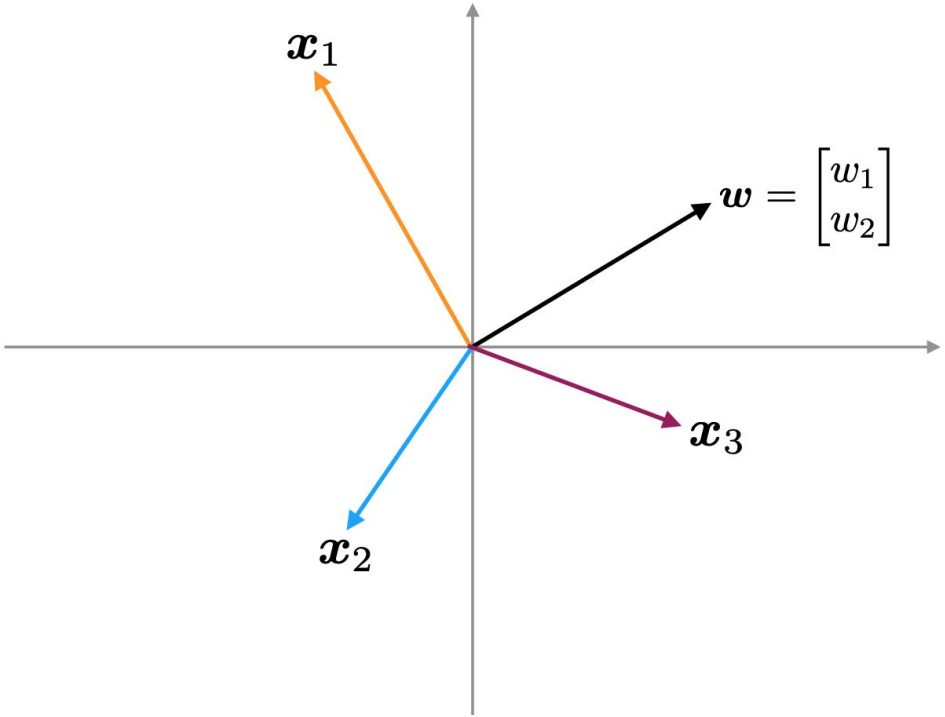
Deny credit if $\sum_{i=1}^{d} w_i x_i \leq \text{threshold} \longrightarrow y = -1$

Can be written formally as $f(\boldsymbol{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right)$

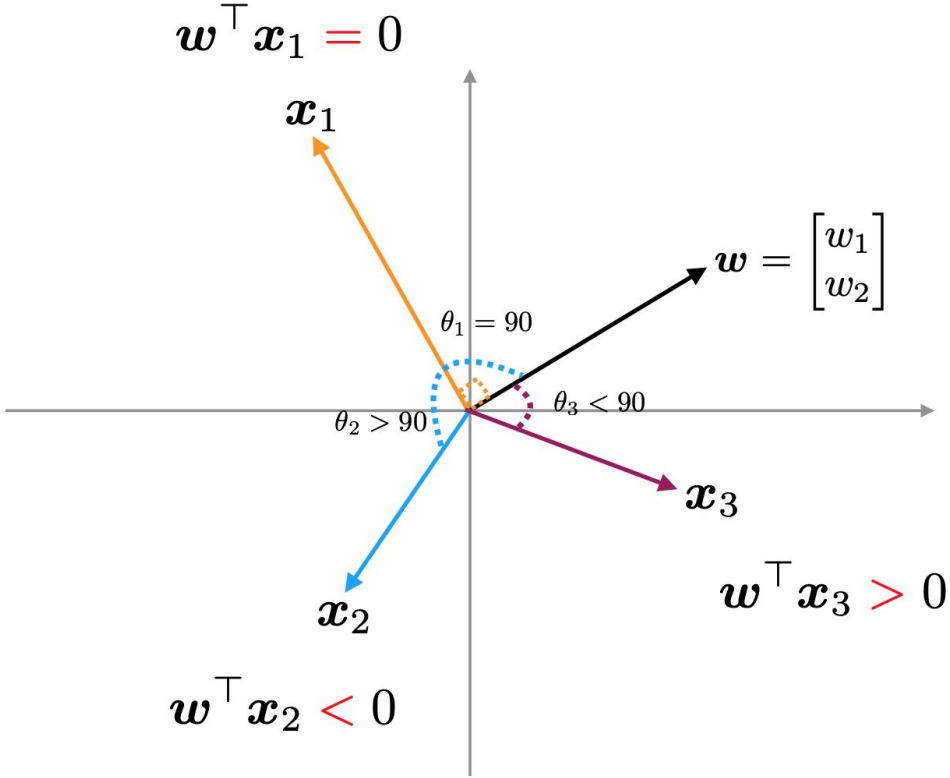
The “bias (intercept) weight” b corresponds to the threshold (how?)

The geometry of linear classifiers

What would be the binary prediction of $\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ for following three vectors (no bias term)?



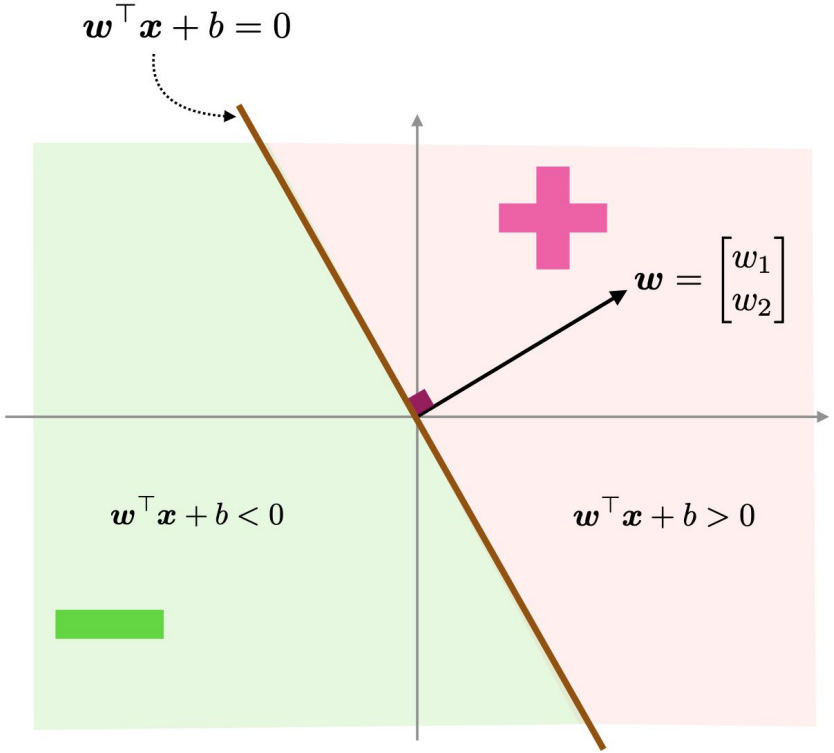
The geometry of linear classifiers



$$a^\top b = \|a\|_2 \|b\|_2 \cos(\theta)$$

The geometry of linear classifiers

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



Compare to the **non-linear** complex decision boundary of nearest neighbors!

Evaluation

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

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

How can we evaluate the classifier on $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$?

$$\frac{\#\text{ of } \; f(\boldsymbol{x_i}) \neq y_i}{n} = \frac{\#\text{ of examples that } \; \text{sign}\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) \neq y_i}{n}$$

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

$$y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) \geq 0$$

The 0/1 loss function

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

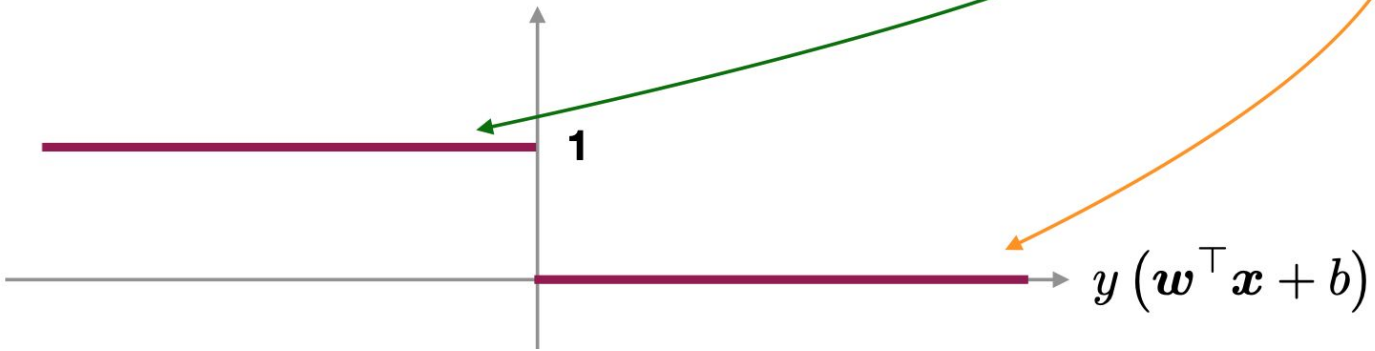
$$\ell(\boldsymbol{w}, b; (\boldsymbol{x}, y)) = \begin{cases} 0 & f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right) = y \\ 1 & f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right) \neq y \end{cases}$$

The 0/1 loss function

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

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

The above loss function is called the **0/1 loss**:



Task

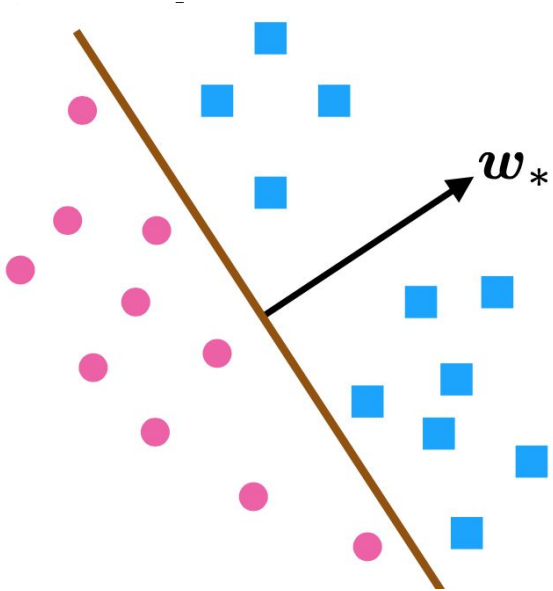
Given training data

$$\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$$

Hypothesis (model space)

$$\boldsymbol{w} \in \mathbb{R}^d, \quad b \in \mathbb{R}$$

Prediction $$f(\boldsymbol{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right)$$



Goal: minimize # of mis-classifications over $\mathcal{S}$

$$\ell(\boldsymbol{w}, b; (\boldsymbol{x}_i, y_i)) = \begin{cases} 0 & \text{sign}\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) = y \\ 1 & \text{sign}\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) \neq y \end{cases}$$

The Perceptron algorithm

Input $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$

Initialize $\boldsymbol{w}_0 = \mathbf{0}$

While not converged (iteration $t = 1, 2, \ldots, T$):

The Perceptron algorithm

Input $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$

Initialize $\boldsymbol{w}_0 = \mathbf{0}$

While not converged (iteration $t = 1, 2, \ldots, T$):

Find an (**BAD**) example $(\boldsymbol{x}_t, y_t) \in \mathcal{S}$ **misclassified** by current solution,

$$\text{sign}\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t\right) \neq y_t$$

Our current prediction

True label

The Perceptron algorithm

Input $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$

Initialize $\boldsymbol{w}_0 = \mathbf{0}$

While not converged (iteration $t = 1, 2, \ldots, T$):

Find an (**BAD**) example $(\boldsymbol{x}_t, y_t) \in \mathcal{S}$ **misclassified** by current solution,

$$\text{sign}\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t\right) \neq y_t$$

Our current prediction

True label

Update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta y_t \boldsymbol{x}_t$$

The learning rate

The Perceptron algorithm

Input $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$

Initialize $\boldsymbol{w}_0 = \mathbf{0}$

While not converged (iteration $t = 1, 2, \ldots, T$):

Find an (**BAD**) example $(\boldsymbol{x}_t, y_t) \in \mathcal{S}$ **misclassified** by current solution,

$$\text{sign}\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t\right) \neq y_t$$

Our current prediction

True label

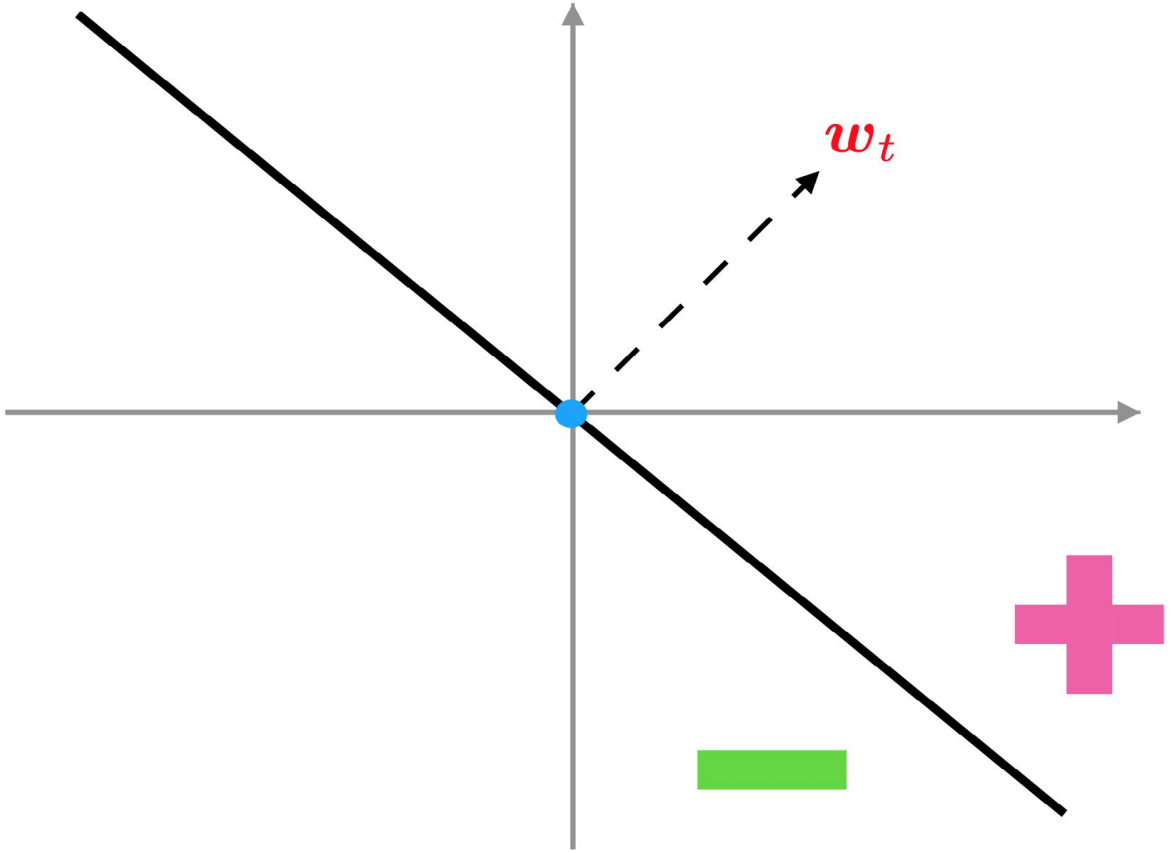
Update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta y_t \boldsymbol{x}_t$$

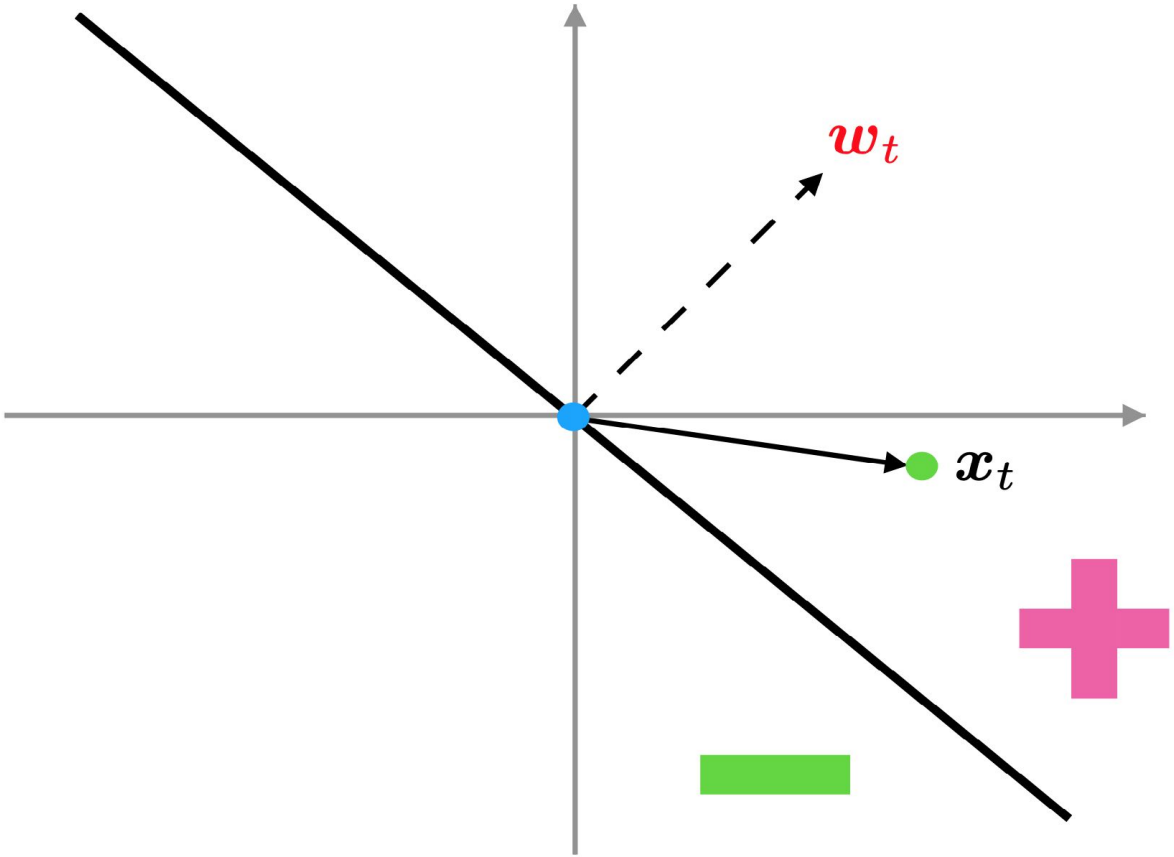
The learning rate

For linearly separable data, it is guaranteed to converge to A solution.

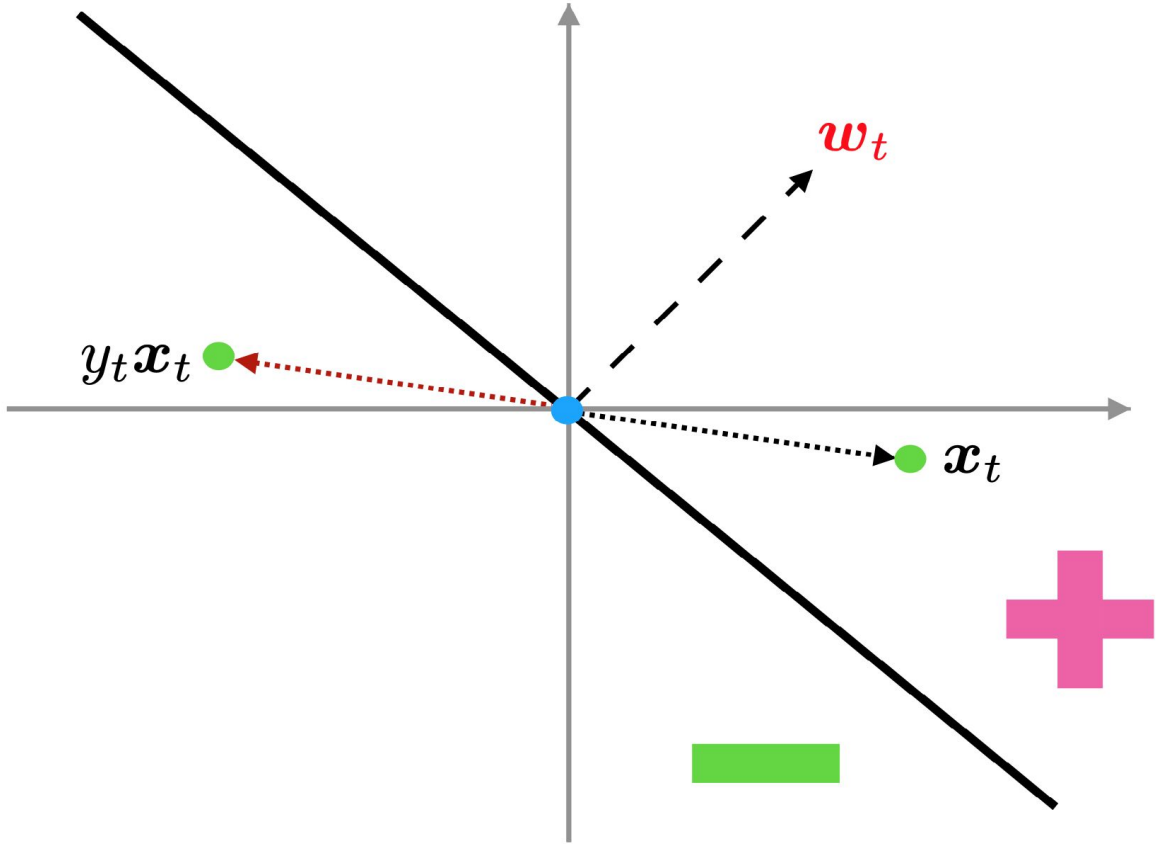
Geometric interpretation



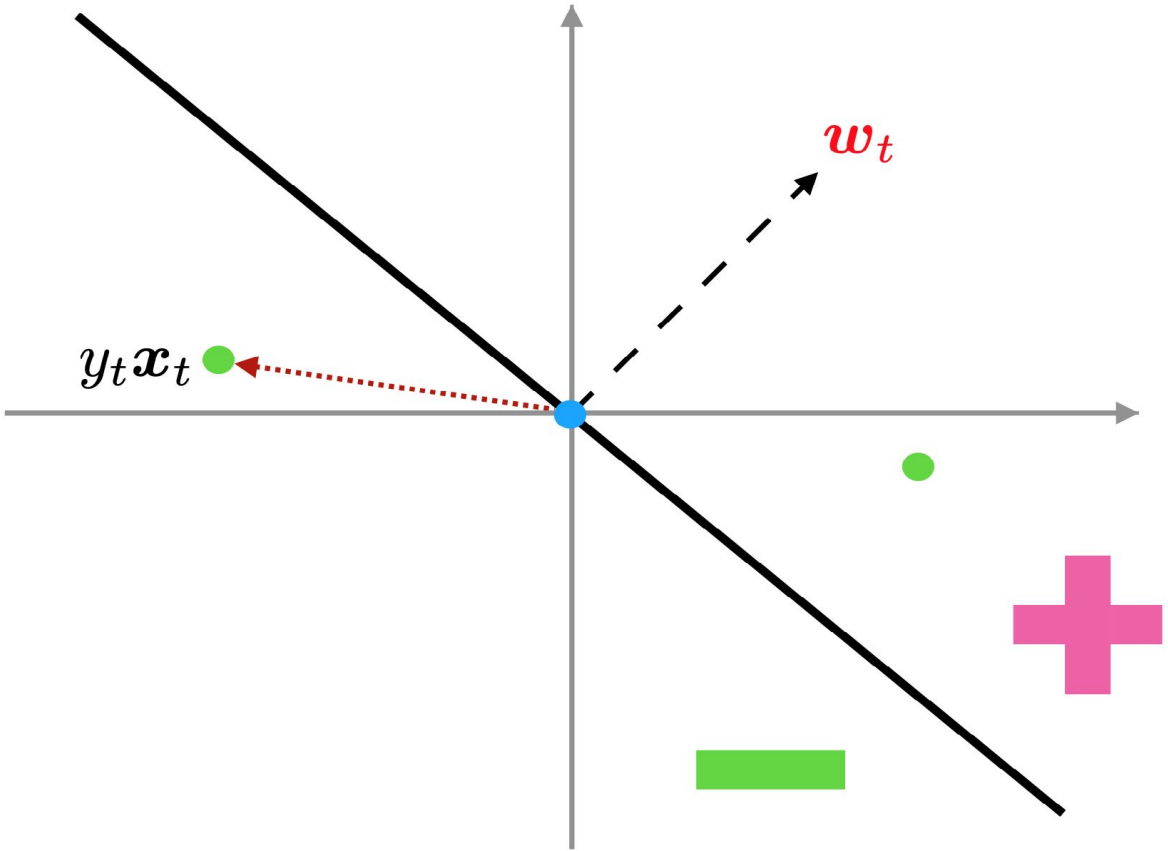
Geometric interpretation



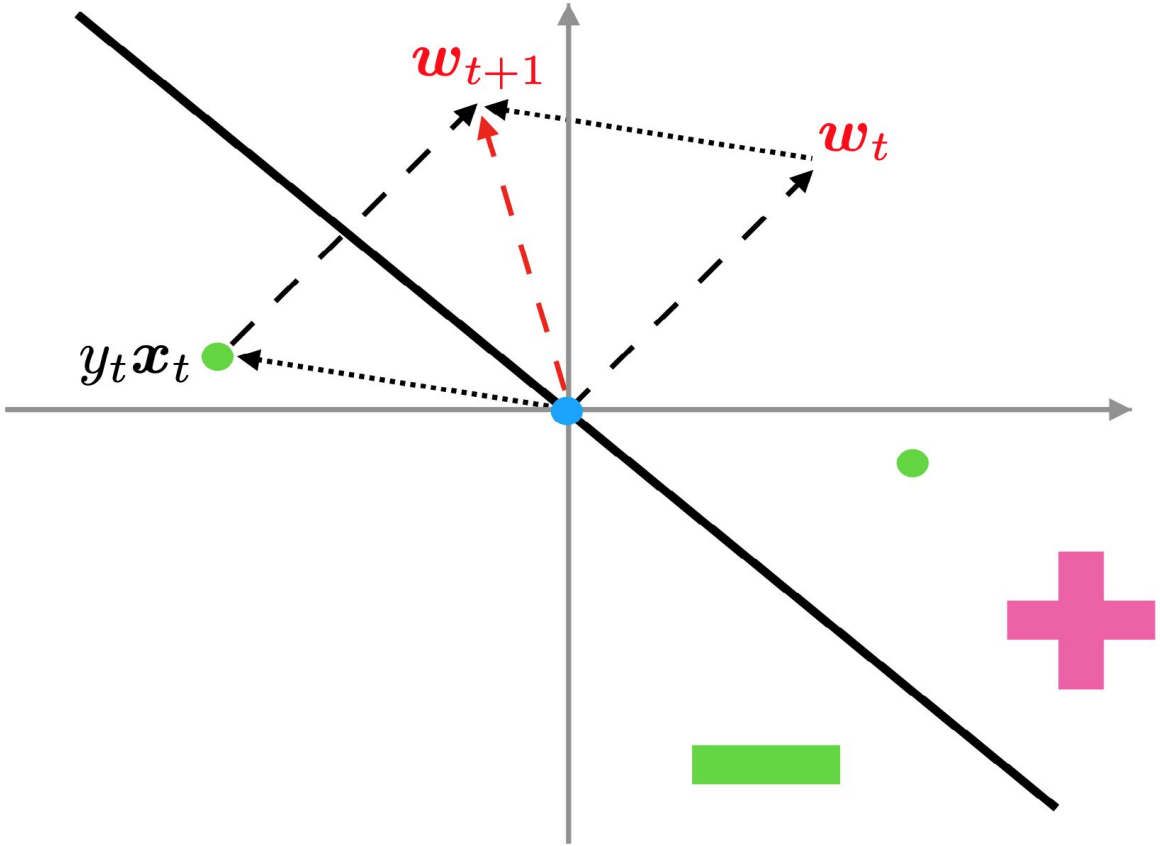
Geometric interpretation



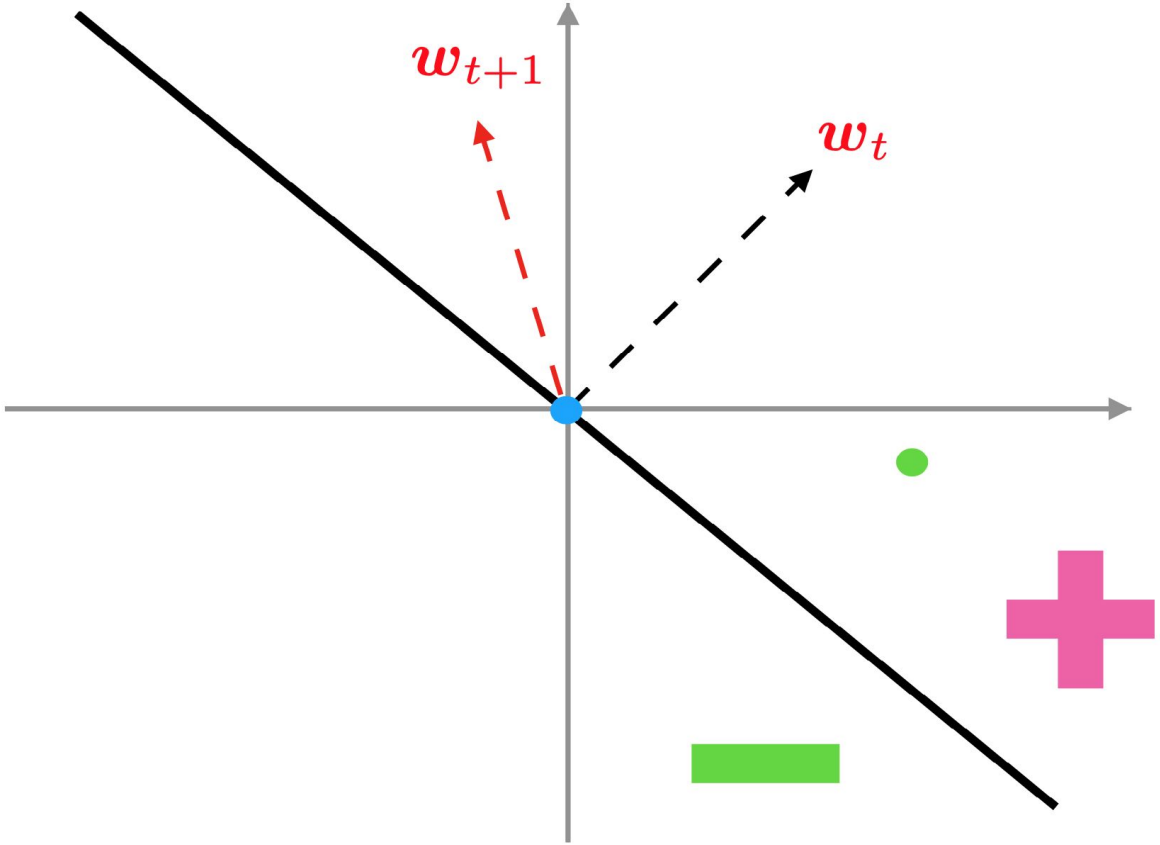
Geometric interpretation



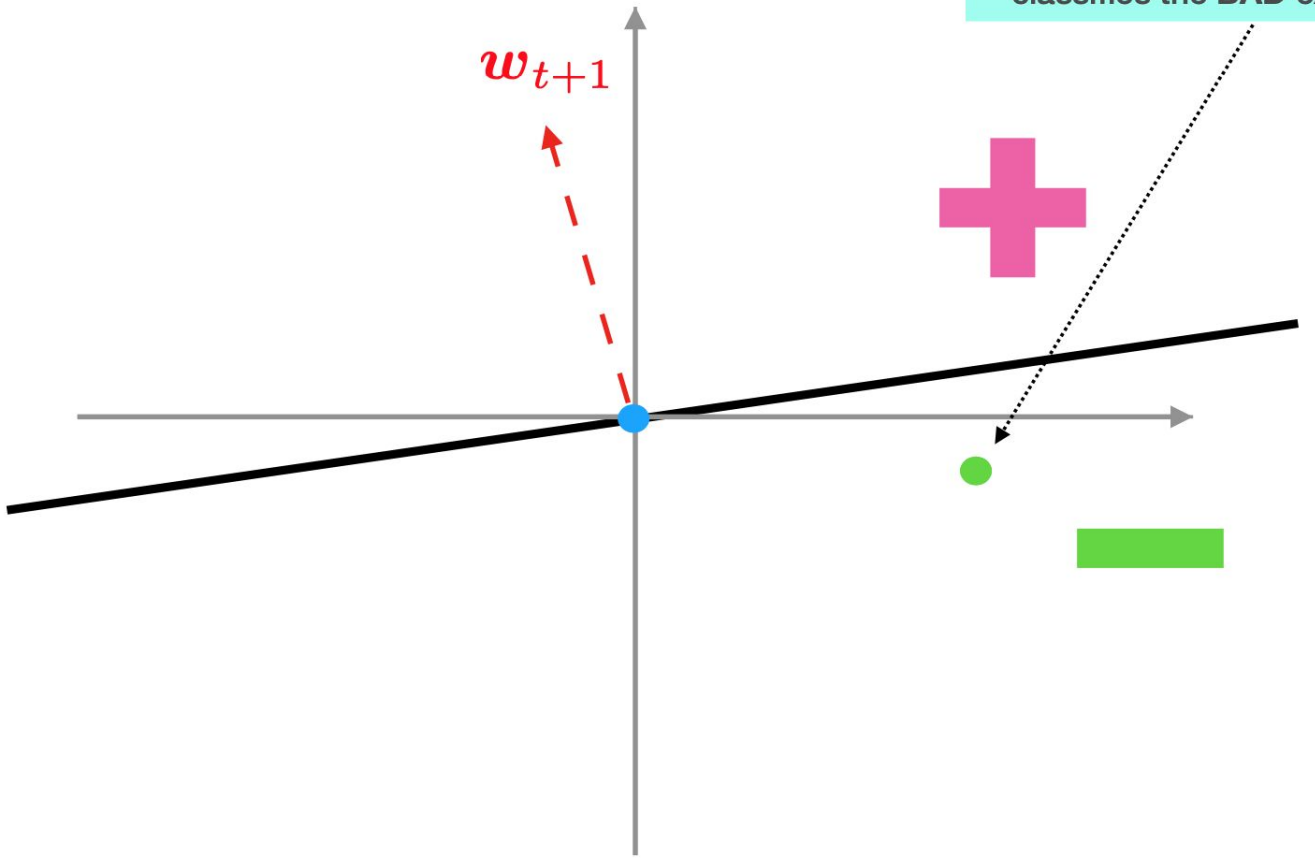
Geometric interpretation



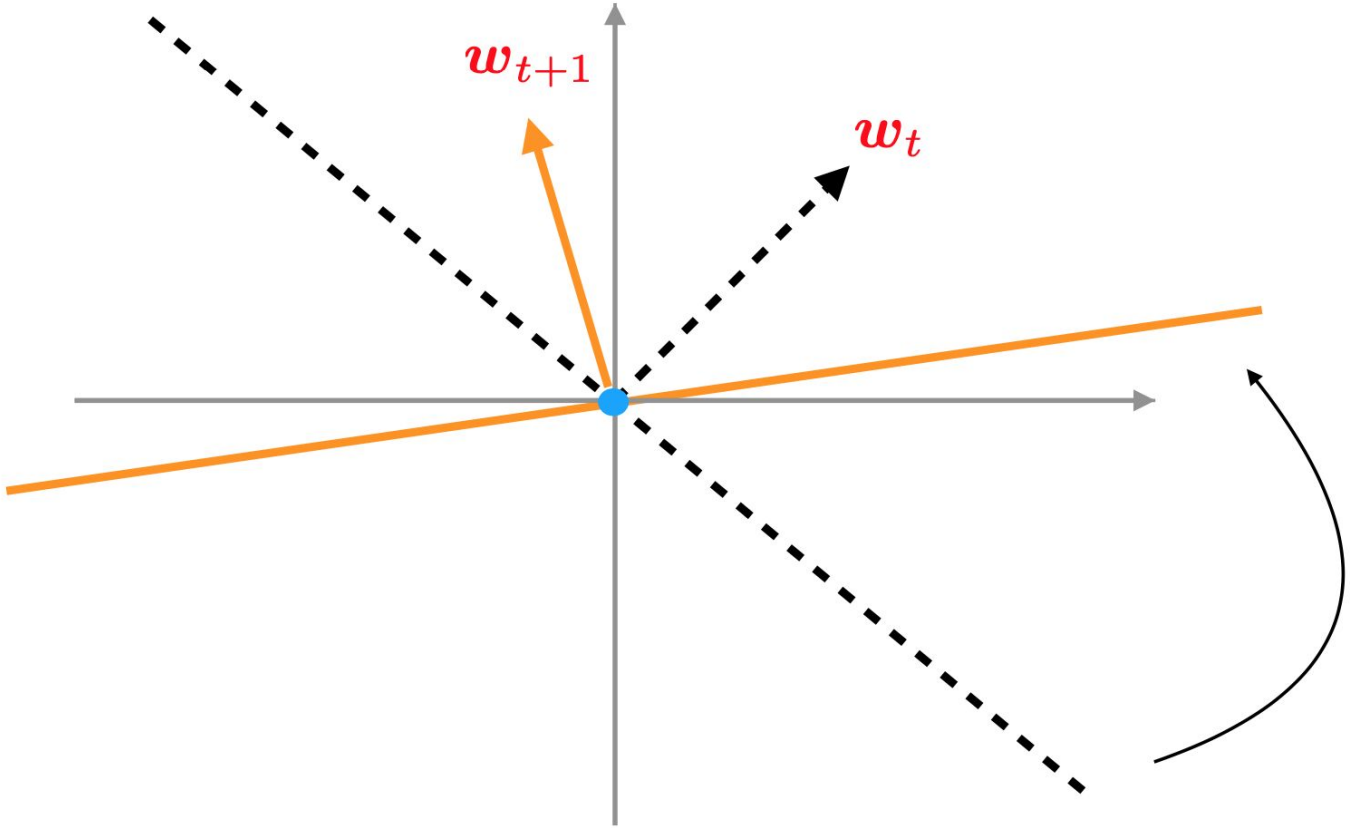
Geometric interpretation



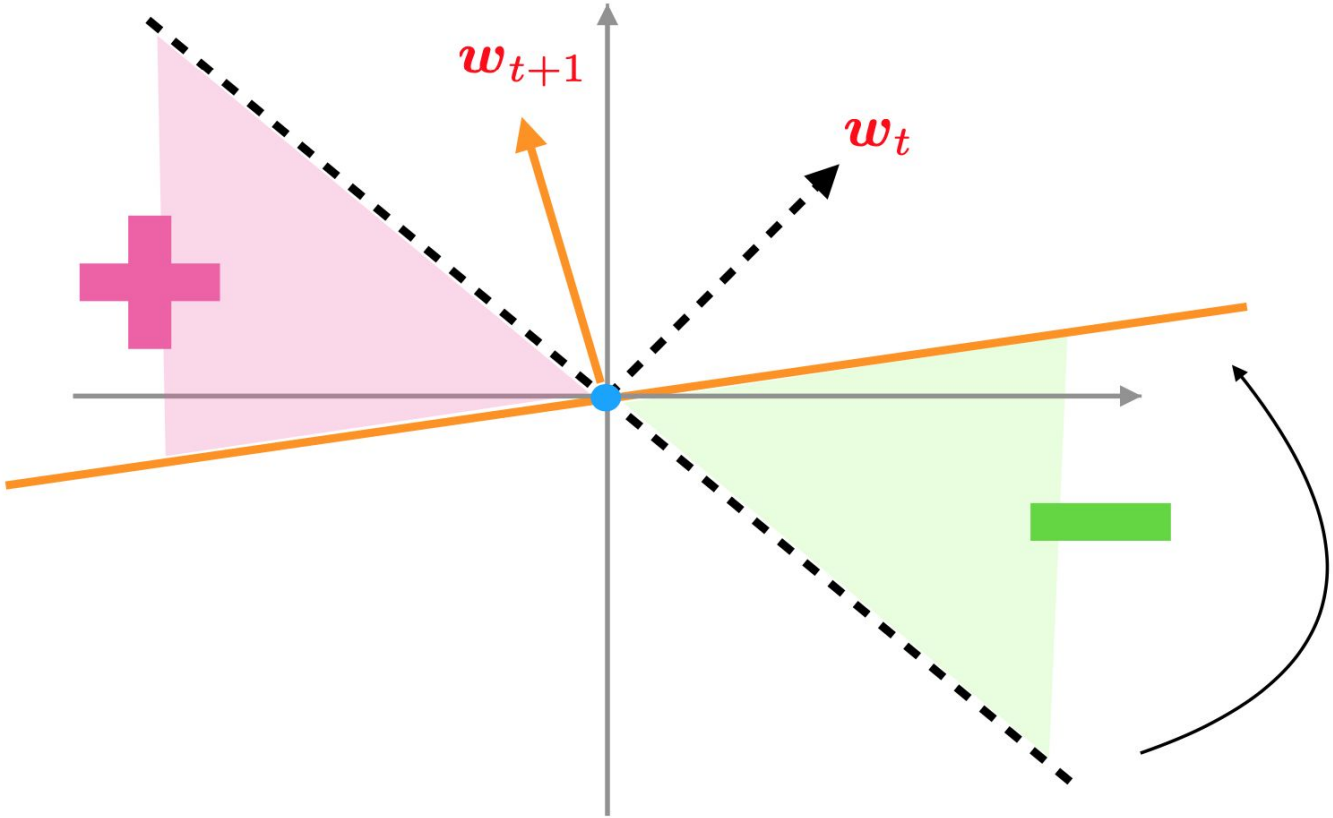
Geometric interpretation



Before and after a single update



Before and after a single update



How much progress?

Prediction by updated model

$$y_t(\boldsymbol{w}_{t+1}{}^\top \boldsymbol{x}_t) = y_t\left(\boldsymbol{w}_t + \eta y_t \boldsymbol{x}_t\right)^\top \boldsymbol{x}_t$$

How much progress?

Prediction by updated model

$$
\begin{aligned}
y_t(\boldsymbol{w}_{t+1}{}^\top \boldsymbol{x}_t) &= y_t\left(\boldsymbol{w}_t + \eta y_t \boldsymbol{x}_t\right)^\top \boldsymbol{x}_t \\
&= y_t \boldsymbol{w}_t^\top \boldsymbol{x}_t + \eta y_t y_t \boldsymbol{x}_t^\top \boldsymbol{x}_t \\
&= y_t \boldsymbol{w}_t^\top \boldsymbol{x}_t + \eta y_t y_t \|\boldsymbol{x}_t\|_2^2 \\
&= y_t \boldsymbol{w}_t^\top \boldsymbol{x}_t + \eta \|\boldsymbol{x}_t\|_2^2 \\
&> y_t(\boldsymbol{w}_t^\top \boldsymbol{x}_t)
\end{aligned}
$$

A positive term is added

Prediction by previous model

Does Perceptron work?

The Perceptron was arguably the first algorithm with a strong formal guarantee.

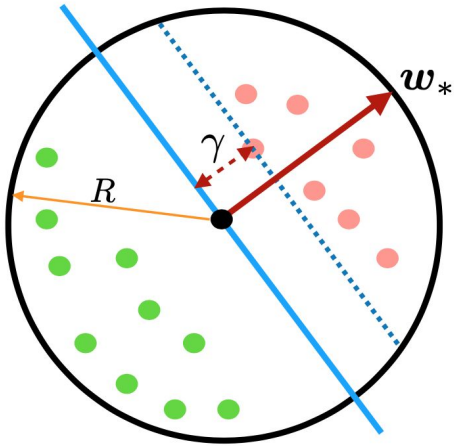
- If a data set is linearly separable, the Perceptron will find a separating hyperplane in a finite number of updates.
- If the data is not linearly separable, it will loop forever.

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:

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



γ is the distance of closest training sample to the classifier (known as the margin of solution)

$$R = \max_{i=1,2,\ldots,n} \|\boldsymbol{x}_i\|_2$$

is the maximum length of training examples

Does Perceptron work?

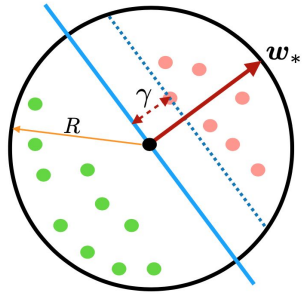
Theorem (convergence of Perceptron).

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)

Proof of Convergence of Perceptron



Proof:
Keeping what we defined above, consider the effect of an update ($\mathbf{w}$ becomes $\mathbf{w} + y\mathbf{x}$) on the two terms $\mathbf{w}^\top \mathbf{w}^*$ and $\mathbf{w}^\top \mathbf{w}$. We will use two facts:

- $y(\mathbf{x}^\top \mathbf{w}) \leq 0$: This holds because $\mathbf{x}$ is misclassified by $\mathbf{w}$ - otherwise we wouldn't make the update.
- $y(\mathbf{x}^\top \mathbf{w}^*) > 0$: This holds because $\mathbf{w}^*$ is a separating hyper-plane and classifies all points correctly.

1. Consider the effect of an update on $\mathbf{w}^\top \mathbf{w}^*$:

$$(\mathbf{w} + y\mathbf{x})^\top \mathbf{w}^* = \mathbf{w}^\top \mathbf{w}^* + y(\mathbf{x}^\top \mathbf{w}^*) \geq \mathbf{w}^\top \mathbf{w}^* + \gamma$$

The inequality follows from the fact that, for $\mathbf{w}^*$, the distance from the hyperplane defined by $\mathbf{w}^*$ to $\mathbf{x}$ must be at least γ (i.e. $y(\mathbf{x}^\top \mathbf{w}^*) = |\mathbf{x}^\top \mathbf{w}^*| \geq \gamma$).

This means that for each update, $\mathbf{w}^\top \mathbf{w}^*$ grows by **at least** γ.

2. Consider the effect of an update on $\mathbf{w}^\top \mathbf{w}$:

$$(\mathbf{w} + y\mathbf{x})^\top (\mathbf{w} + y\mathbf{x}) = \mathbf{w}^\top \mathbf{w} + \underbrace{2y(\mathbf{w}^\top \mathbf{x})}_{<0} + \underbrace{y^2(\mathbf{x}^\top \mathbf{x})}_{0 \leq \leq 1} \leq \mathbf{w}^\top \mathbf{w} + 1$$

The inequality follows from the fact that

- $2y(\mathbf{w}^\top \mathbf{x}) < 0$ as we had to make an update, meaning $\mathbf{x}$ was misclassified
- $0 \leq y^2(\mathbf{x}^\top \mathbf{x}) \leq 1$ as $y^2 = 1$ and all $\mathbf{x}^\top \mathbf{x} \leq 1$ (because $||\mathbf{x}|| \leq 1$).

This means that for each update, $\mathbf{w}^\top \mathbf{w}$ grows by **at most** 1.

3. Now we know that after M updates the following two inequalities must hold:

(1) $\mathbf{w}^\top \mathbf{w}^* \geq M\gamma$

(2) $\mathbf{w}^\top \mathbf{w} \leq M$.

We can then complete the proof:

$$\begin{aligned}
M\gamma &\leq \mathbf{w}^\top \mathbf{w}^* && \text{By (1)} \\
&= ||\mathbf{w}|| \cos(\theta) && \text{by definition of inner-product, where } \theta \text{ is the angle between } \mathbf{w} \text{ and } \mathbf{w}^*. \\
&\leq ||\mathbf{w}|| && \text{by definition of cos, we must have } \cos(\theta) \leq 1. \\
&= \sqrt{\mathbf{w}^\top \mathbf{w}} && \text{by definition of } \|\mathbf{w}\| \\
&\leq \sqrt{M} && \text{By (2)} \\
&\Rightarrow M\gamma \leq \sqrt{M} \\
&\Rightarrow M^2\gamma^2 \leq M \\
&\Rightarrow M \leq \frac{1}{\gamma^2} && \text{And hence, the number of updates } M \text{ is bounded from above by a constant.}
\end{aligned}$$

https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html

What loss function the Perceptron is optimizing

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

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta y_t \boldsymbol{x}_t = \begin{cases} \boldsymbol{w}_t + \eta \boldsymbol{x}_t & y_t = +1 \\ \boldsymbol{w}_t - \eta \boldsymbol{x}_t & y_t = -1 \end{cases}$$

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:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta y_t \boldsymbol{x}_t = \begin{cases} \boldsymbol{w}_t + \eta \boldsymbol{x}_t & y_t = +1 \\ \boldsymbol{w}_t - \eta \boldsymbol{x}_t & y_t = -1 \end{cases}$$

What loss function the Perceptron is trying to optimize?

The loss of classifier on $(\boldsymbol{x}_i, y_i)$ is measured by the following function:

$$\max\{0, -y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i\right)\}$$

Perceptron as Stochastic Gradient Descent

Perceptron can be seen as SGD with the loss function $\max\{0, -y\left(\boldsymbol{w}^\top \boldsymbol{x}\right)\}$

$\boldsymbol{w}_0 = \mathbf{0}$

$M = 0$ [number of mistakes]

$t = 1$ [number of iterations]

while $\exists$ **a sample misclassified do:**

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

If the sample is misclassified, $y_t\left(\boldsymbol{w}_t^\top \boldsymbol{x}_t\right) < 0$ **update:**

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t \boldsymbol{x}_t$$

$$M \leftarrow M + 1$$

$t \leftarrow t + 1$

end for

return $\boldsymbol{w}_t$

Training loss versus evaluation metric

The loss function we use in training with gradient descent:

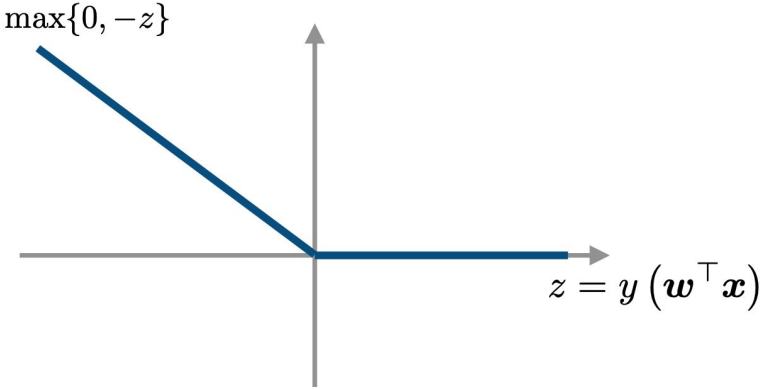
$$\max\{0, -y\left(\boldsymbol{w}^\top \boldsymbol{x}\right)\}$$

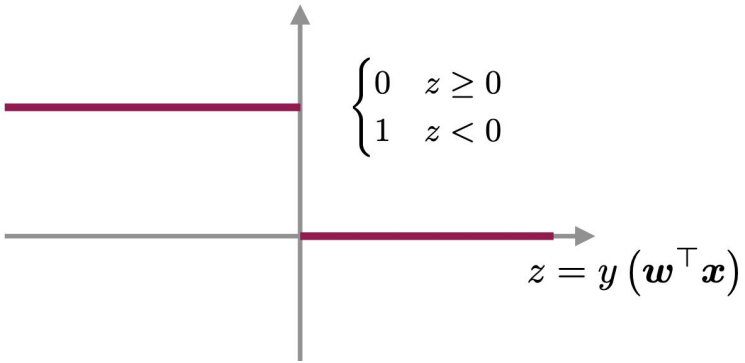
The evaluation metric we use in evaluating the accuracy on test data [omitting the intercept b]:

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

OR

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





Training loss versus evaluation metric

The loss function we use in training with gradient descent:

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

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

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

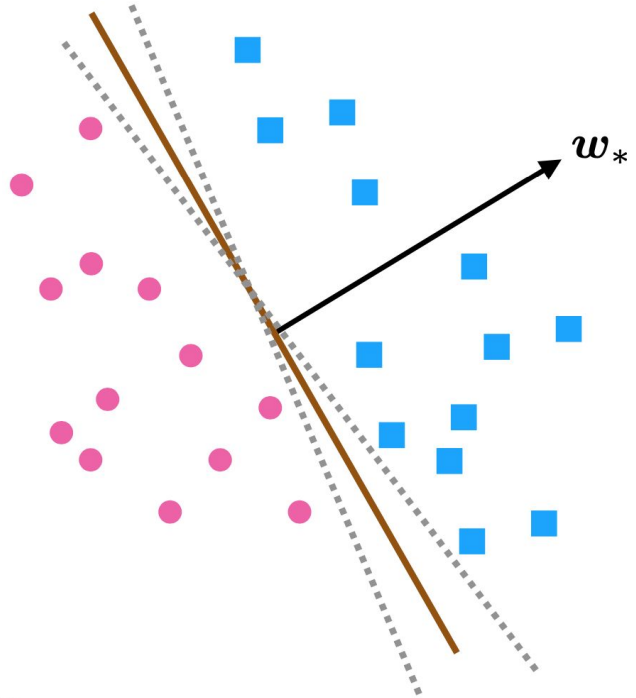
OR

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

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

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!

$$\min_{\boldsymbol{w},b} \quad 0$$

$$\text{such that} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0, \quad i = 1, 2, \ldots, n$$

Useful facts about Perceptron

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

The $\boldsymbol{w}^\top \boldsymbol{x}$ is just the distance of $\boldsymbol{x}$ from the origins when projected onto the vector $\boldsymbol{w}$ (assume the vector is unit length)

The role of the bias (intercept) b is to shift the decision boundary away from the origin, in the direction of $\boldsymbol{w}$

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

And …

The Perceptron algorithm initially gave a huge wave of excitement ("digital brains")
Then, contributed to the A.I. Winter. Famous counter-example **XOR** problem (Minsky 1969):

ARCHIVES | 1958

Electronic 'Brain' Teaches Itself

JULY 13, 1958

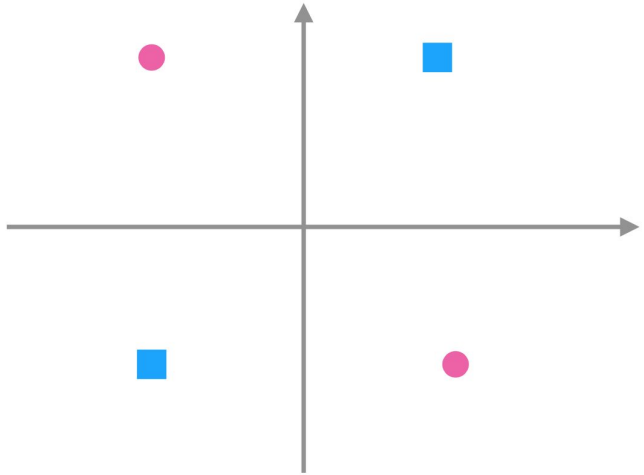
About the Archive

This is a digitized version of an article from The Times's print archive, before the start of online publication in 1996. To preserve these articles as they originally appeared, The Times does not alter, edit or update them.

Occasionally the digitization process introduces transcription errors or other problems. Please send reports of such problems to archive_feedback@nytimes.com.

The Navy last week demonstrated the embryo of an electronic computer named the Perceptron which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control."

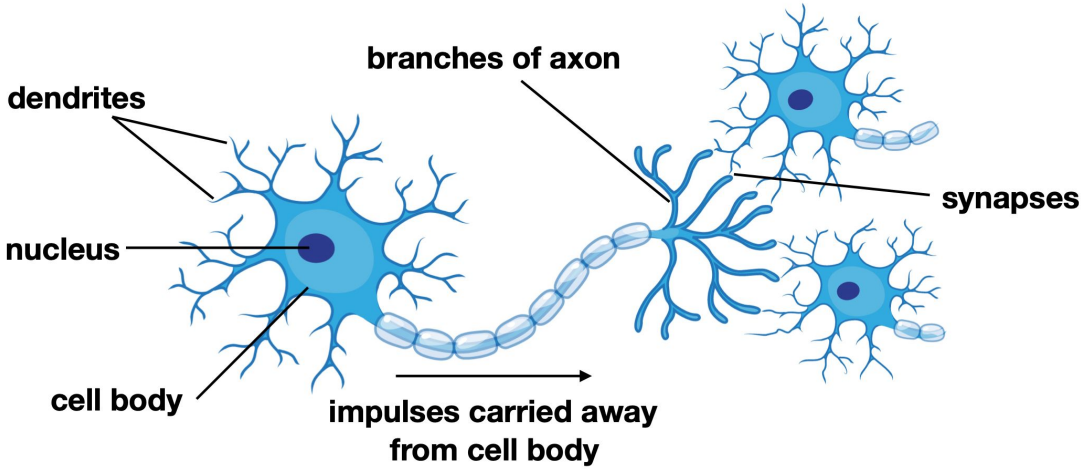
The New York Times



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!

Biological neurons



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.

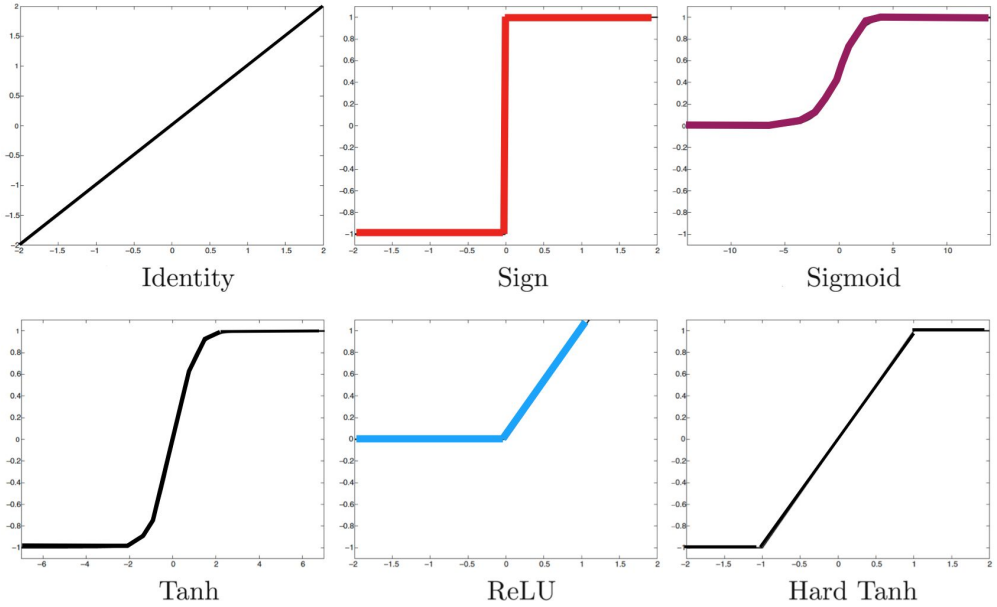
Each input to a neuron is scaled with a weight, which affects the function computed at that unit. The neuron is fired (activated) when the input is beyond a threshold.

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.

Choices of activation functions



$$\sigma(z) = \text{sign}(z) \qquad \text{(Sign)}$$

$$\sigma(z) = \max\{0, z\} \qquad \text{(Rectified Linear Unit [ReLU])}$$

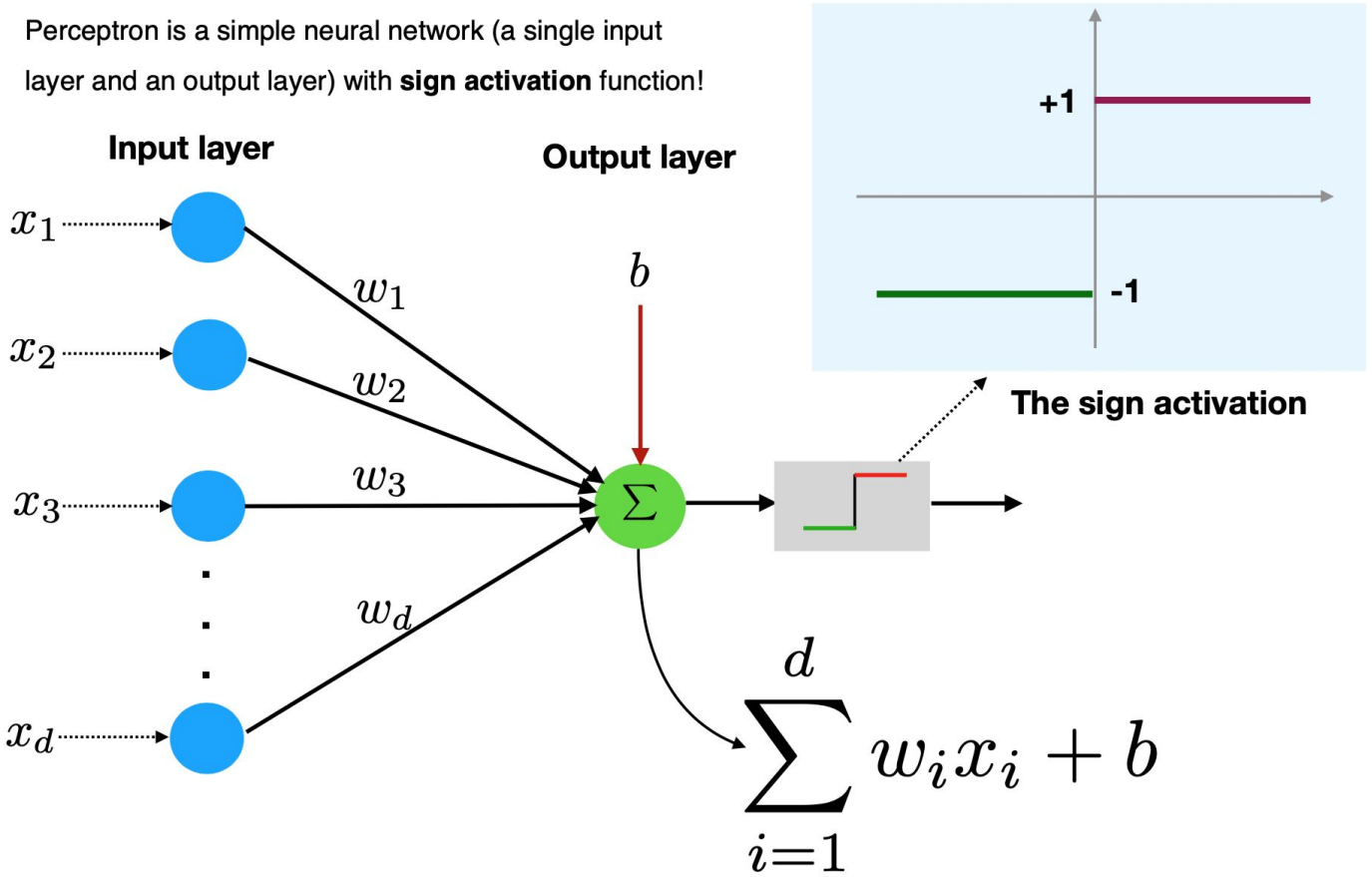
$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad \text{(Sigmoid)}$$

Perceptron as an artificial neuron

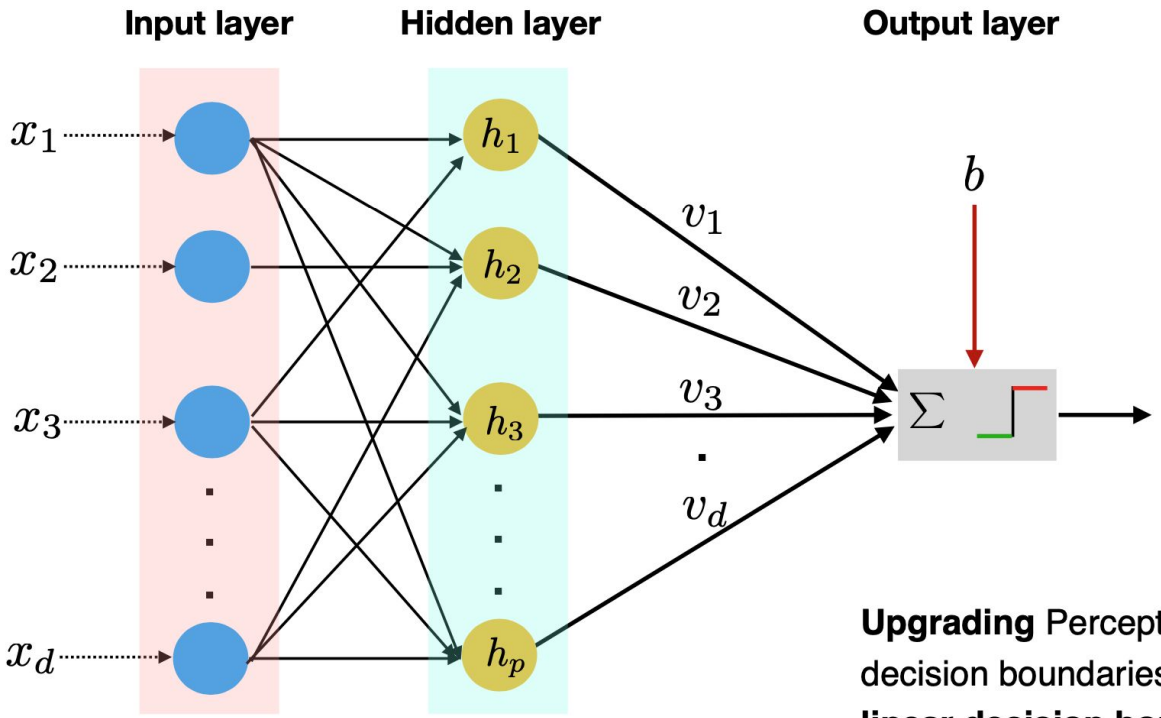
Perceptron is a simple neural network (a single input layer and an output layer) with **sign activation** function!

The sign activation

Perceptron as an artificial neuron

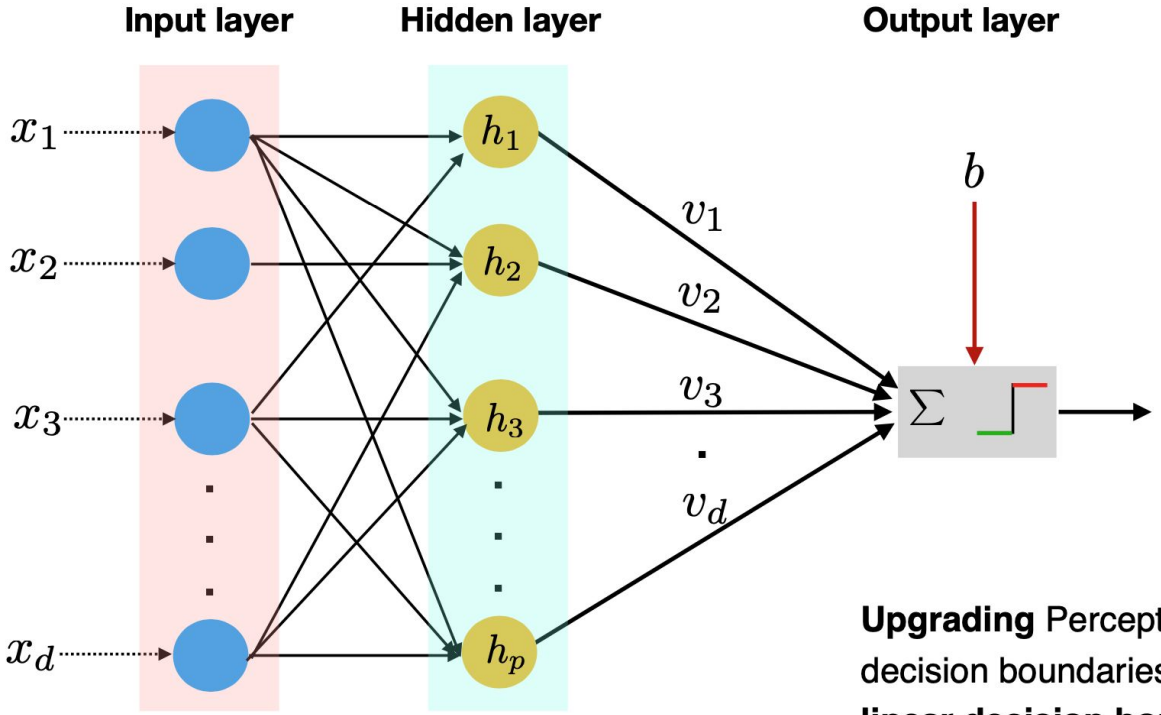


Two layer neural networks



Upgrading Perceptron (linear decision boundaries) to **non-linear decision boundaries**!

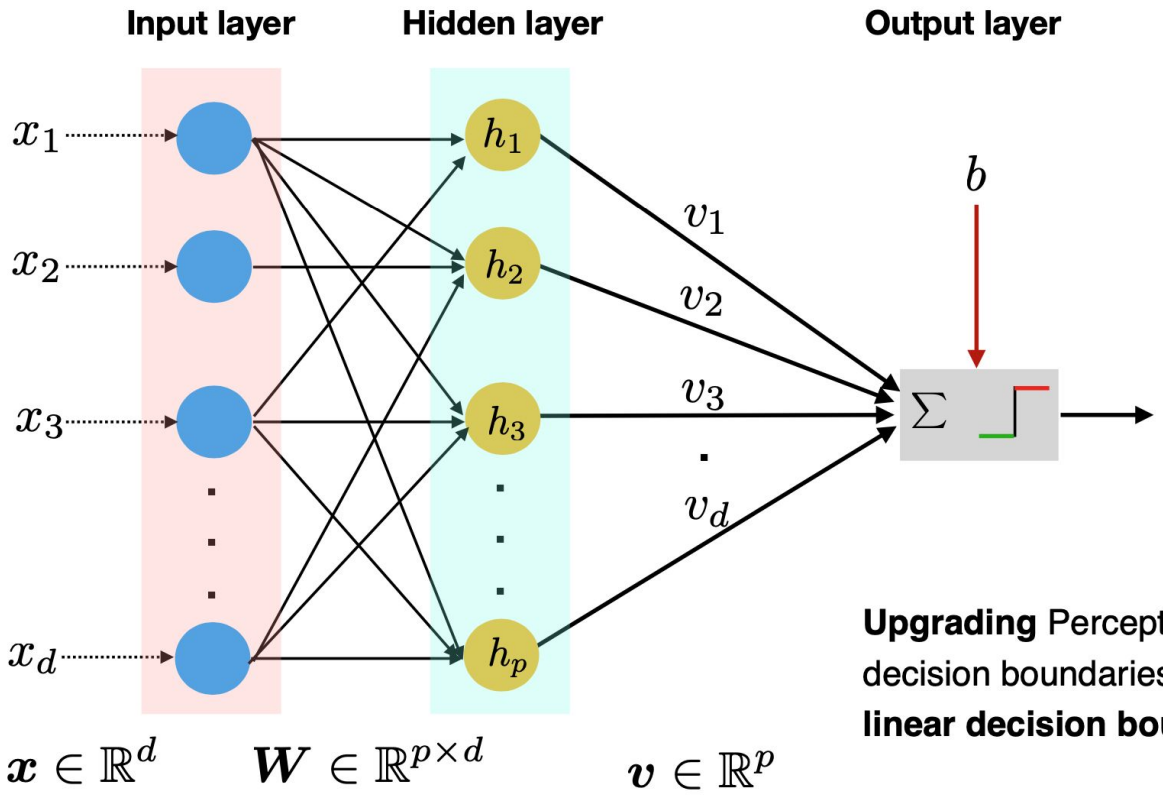
Two layer neural networks



$\boldsymbol{x} \in \mathbb{R}^d$ $\boldsymbol{W} \in \mathbb{R}^{p \times d}$ $\boldsymbol{v} \in \mathbb{R}^p$

Upgrading Perceptron (linear decision boundaries) to **non-linear decision boundaries**!

Two layer neural networks



Expressiveness: A two-layer neural network can be used as a **universal function approximator** if a large number of hidden units are used within the hidden layer

Upgrading Perceptron (linear decision boundaries) to **non-linear decision boundaries**!

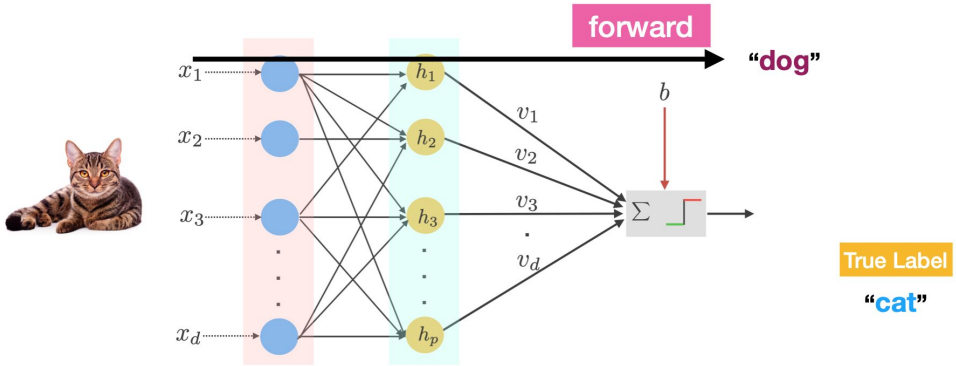
How to train two-layer Perceptron?

Forward propagation:

Backward propagation:

How to train two-layer Perceptron?

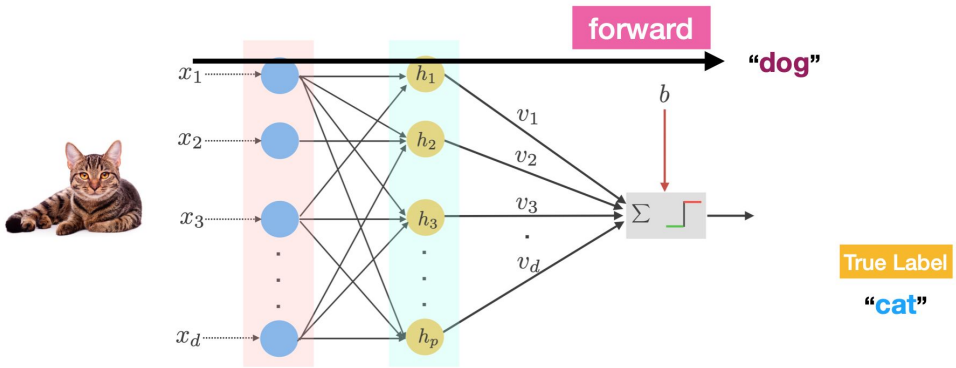
Forward propagation: In this phase, the inputs for a training instance are fed into the neural network. This results in a forward cascade of computations across the layers, using the current set of weights. The output is the prediction of current weights.



Backward propagation:

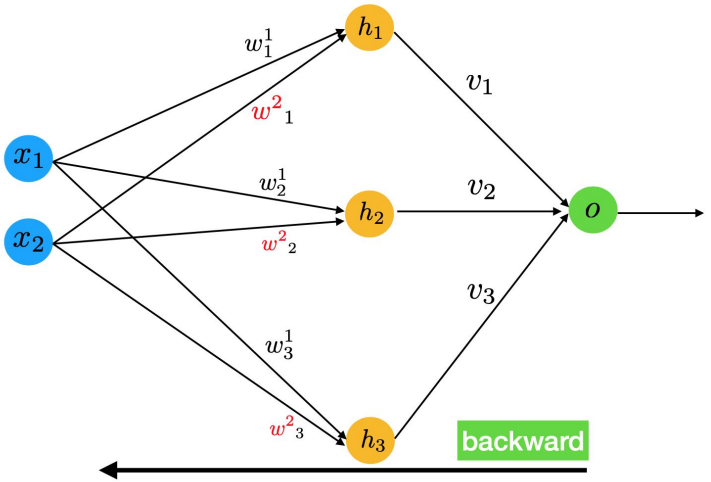
How to train two-layer Perceptron?

Forward propagation: In this phase, the inputs for a training instance are fed into the neural network. This results in a forward cascade of computations across the layers, using the current set of weights. The output is the prediction of current weights.



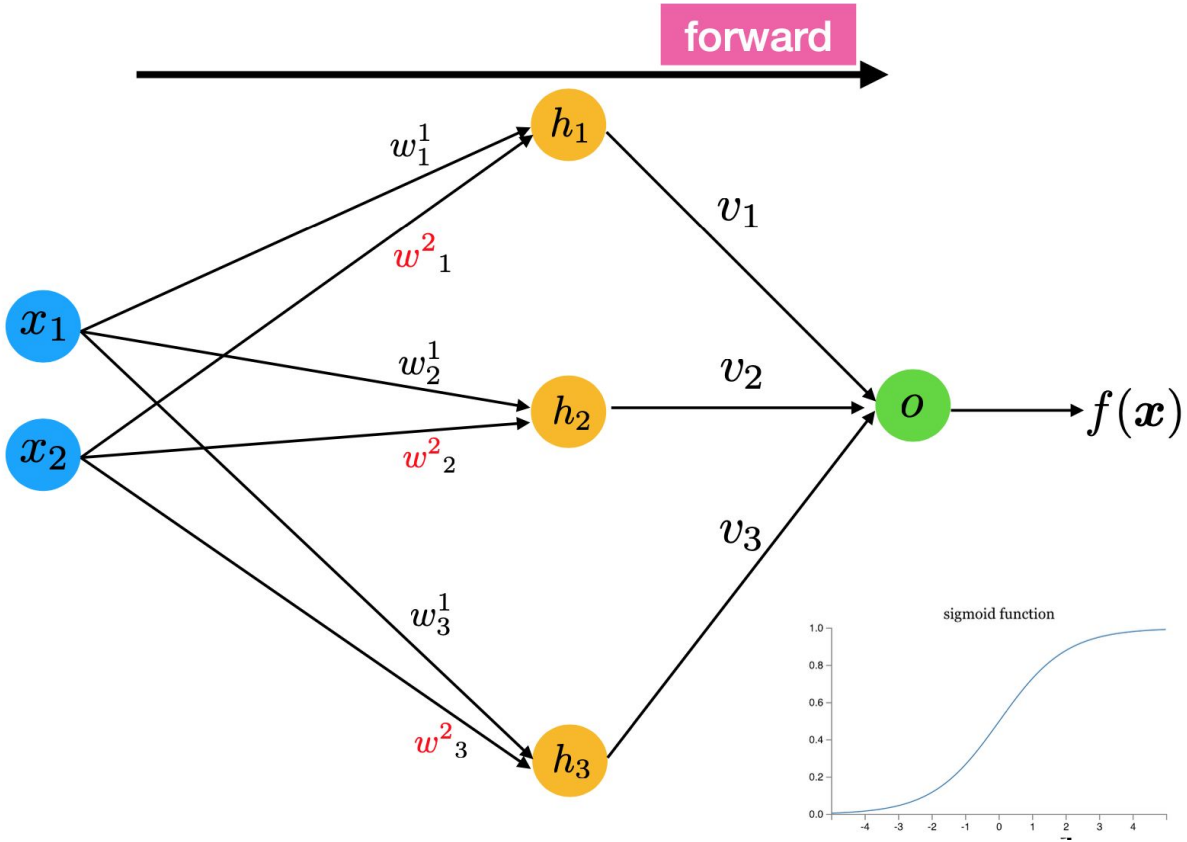
True Label

"cat"



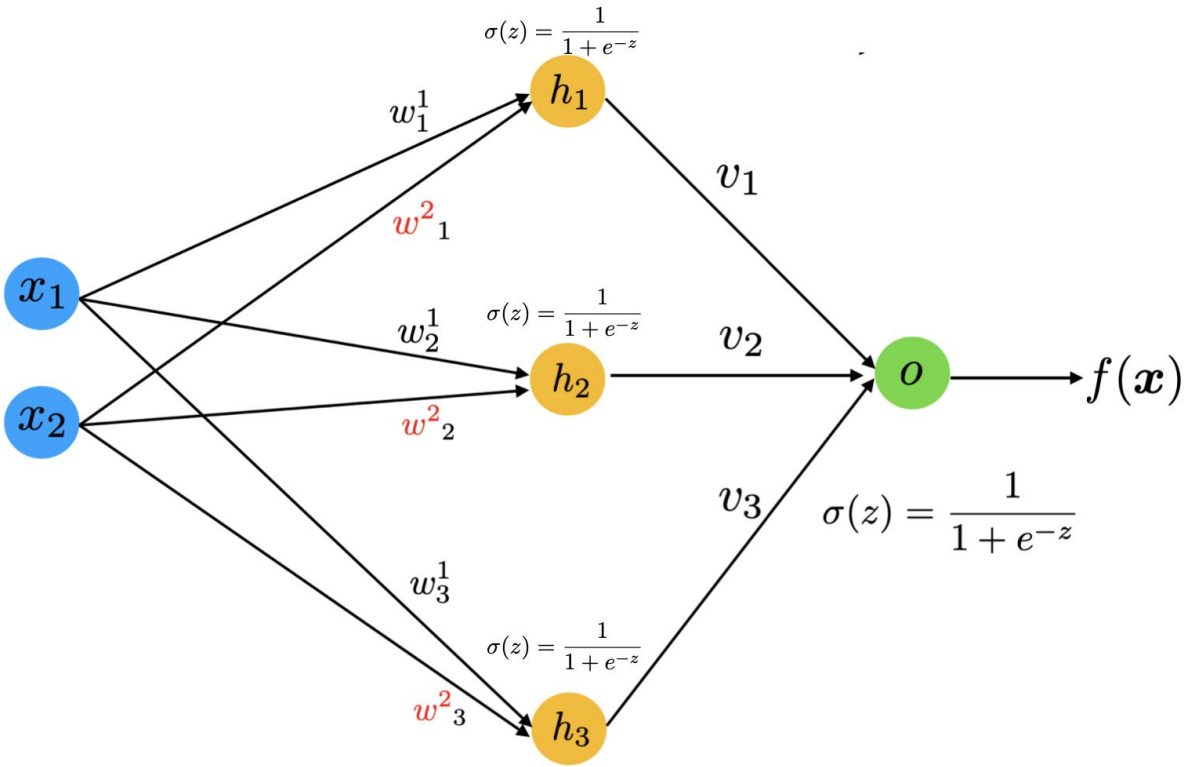
Backward propagation: The main goal of the backward phase is to **learn the gradient of the loss function** with respect to the different weights by **using the chain rule** of differential calculus. The weights are just adjusted based on error.

Forward propagation

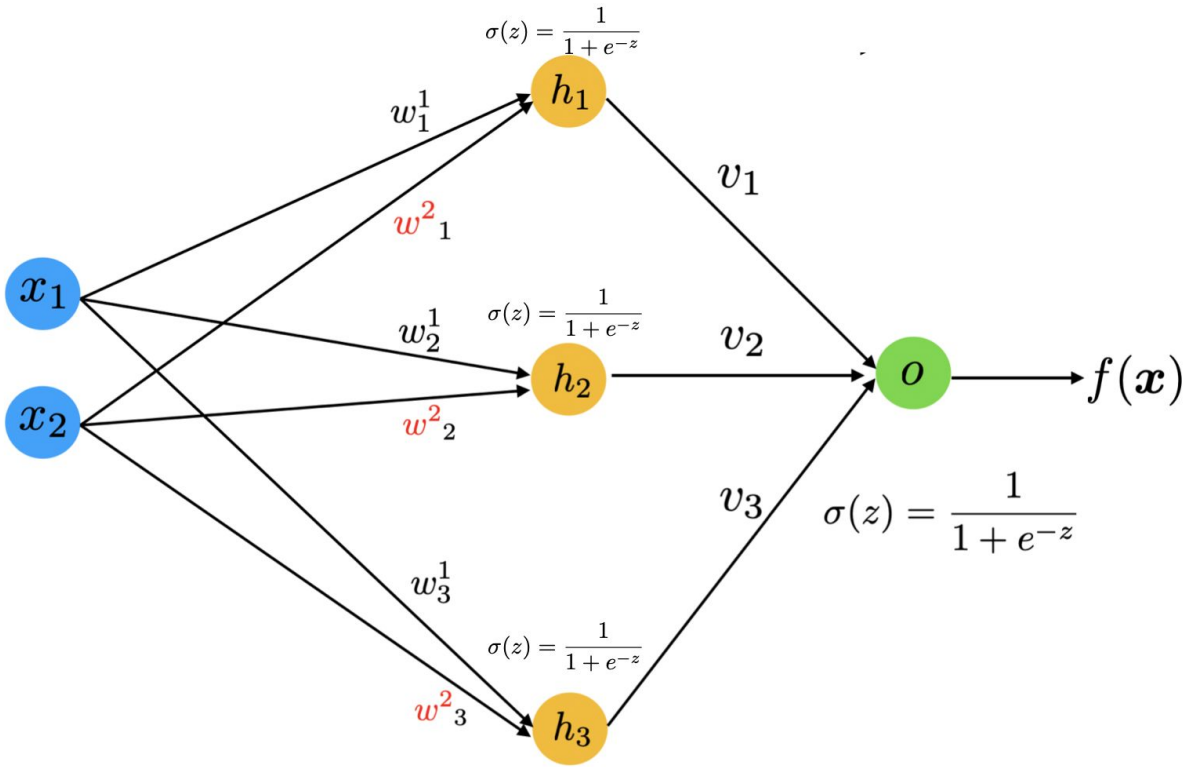


We use the **sigmoid** activation function which introduces better non-linearity:

Forward propagation



Forward propagation



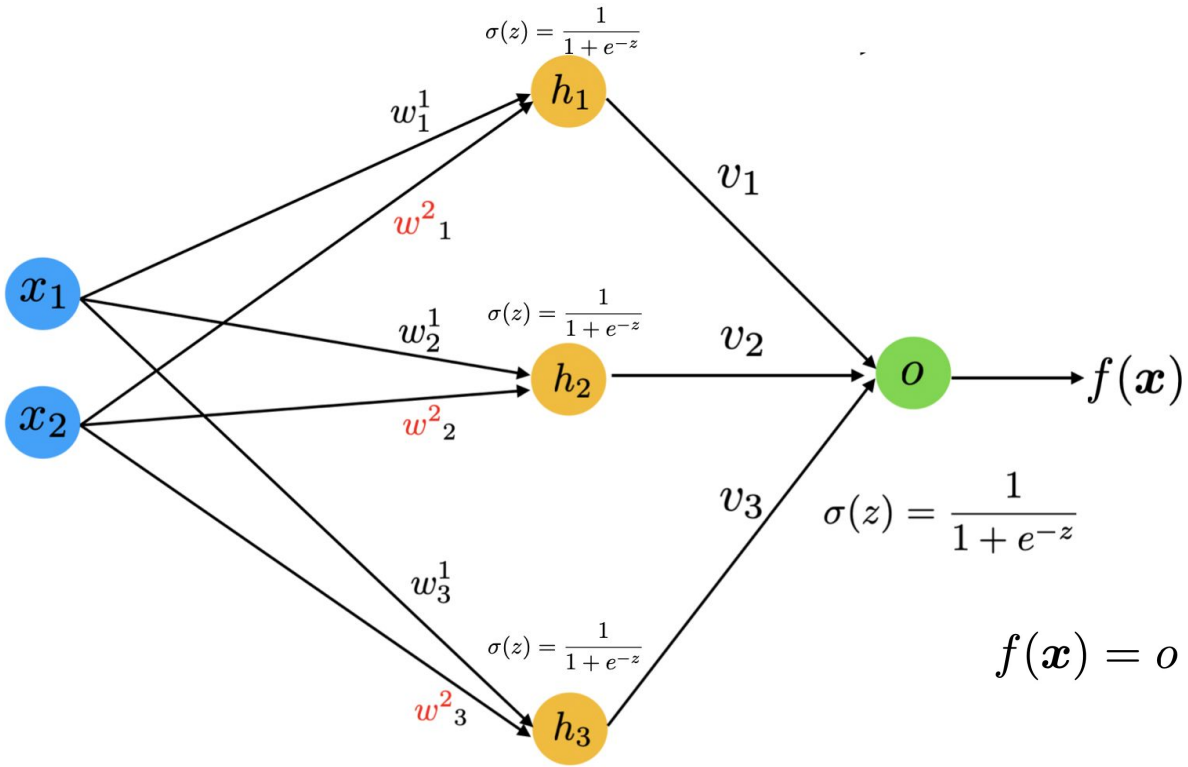
$$h_1 = \sigma(w_1^1 x_1 + w_1^2 x_2)$$

$$h_2 = \sigma(w_2^1 x_1 + w_2^2 x_2)$$

$$h_3 = \sigma(w_3^1 x_1 + w_3^2 x_2)$$

$$o = \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3)$$

Forward propagation



$$h_1 = \sigma(w_1^1 x_1 + w_1^2 x_2)$$

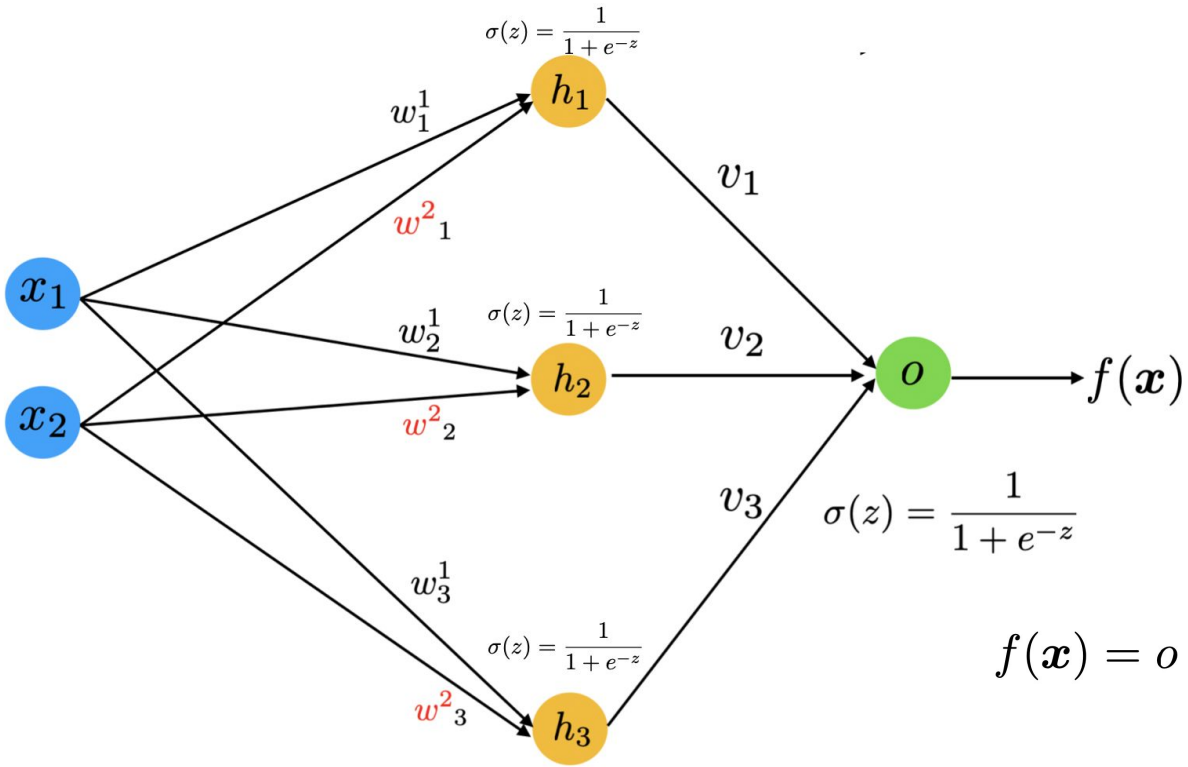
$$h_2 = \sigma(w_2^1 x_1 + w_2^2 x_2)$$

$$h_3 = \sigma(w_3^1 x_1 + w_3^2 x_2)$$

$$o = \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3)$$

$$f(\boldsymbol{x}) = o = \quad \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3)$$

Forward propagation - Compact Form



$$h_1 = \sigma(w_1^1 x_1 + w_1^2 x_2)$$

$$h_2 = \sigma(w_2^1 x_1 + w_2^2 x_2)$$

$$h_3 = \sigma(w_3^1 x_1 + w_3^2 x_2)$$

$$o = \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3)$$

$$f(\boldsymbol{x}) = o = \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3)$$

$$= \sigma\left(\boldsymbol{v}^\top \boldsymbol{\sigma}\left(\boldsymbol{W}^\top \boldsymbol{x}\right)\right)$$

$\boldsymbol{W} \in \mathbb{R}^{3\times 2}$ $\quad \boldsymbol{v} \in \mathbb{R}^3$

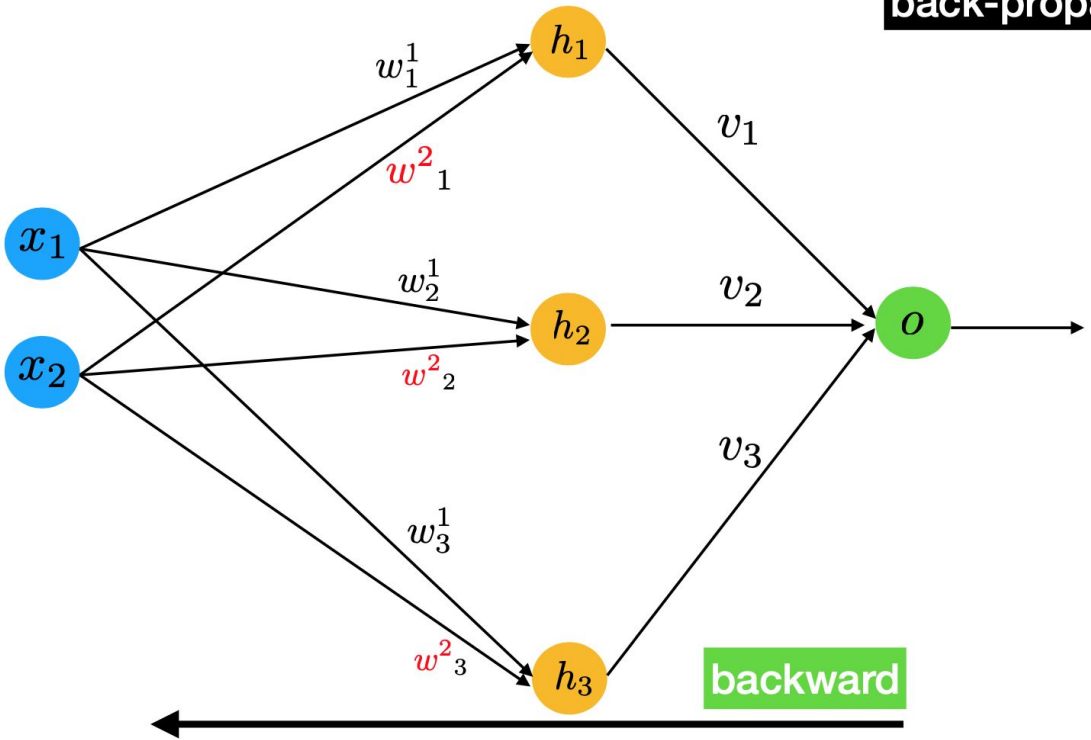
Backpropagation

After we feed a training data and compute the error, we need to adjust the weights!

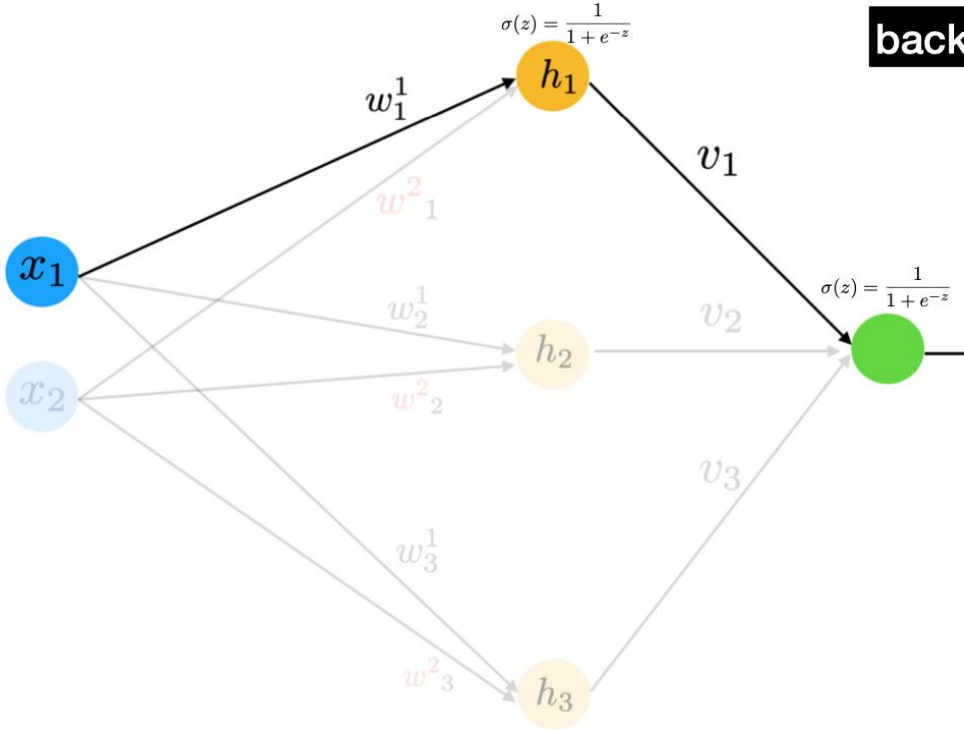
Backpropagation

After we feed a training data and compute the error, we need to adjust the weights!

back-propagation = gradient descent + chain rule



Backpropagation



back-propagation = gradient descent + chain rule

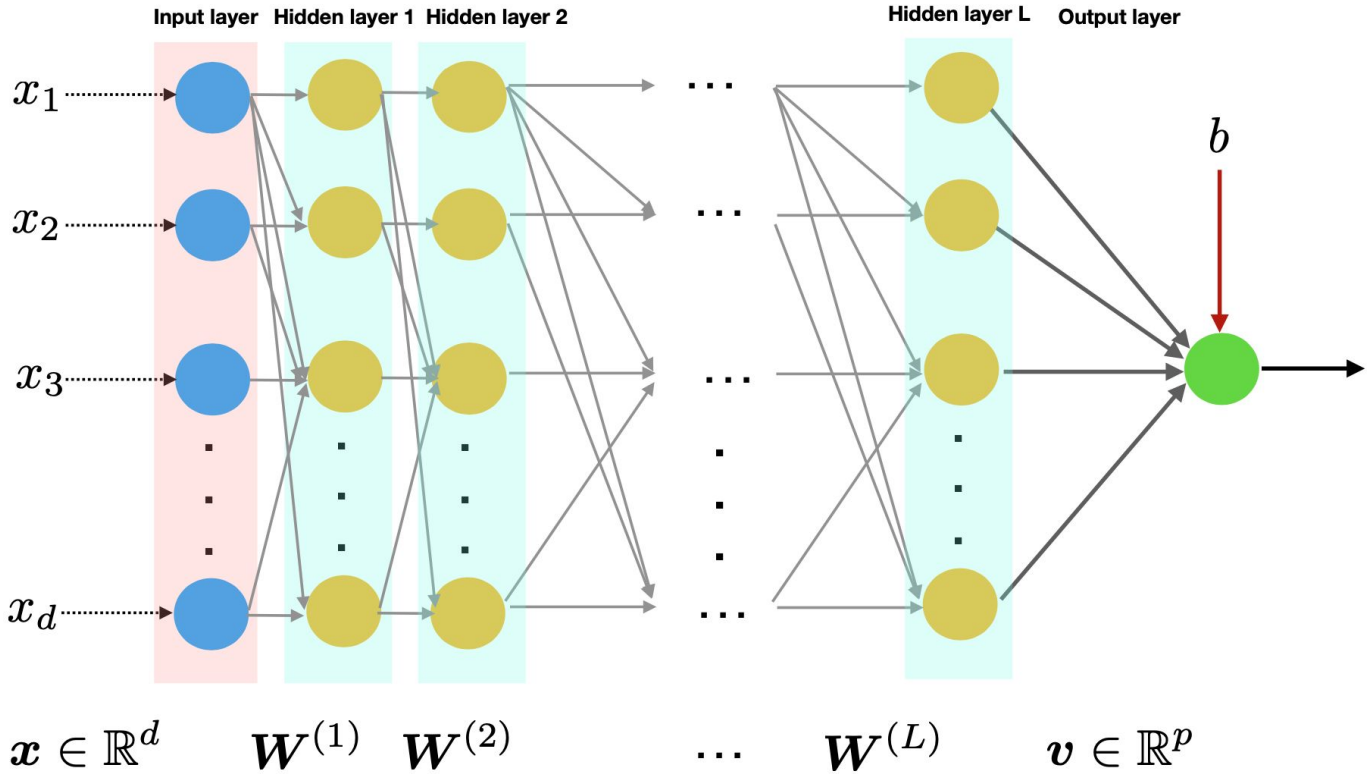
$$f(\boldsymbol{x}) = \sigma\left(\boldsymbol{v}^\top \boldsymbol{\sigma}\left(\boldsymbol{W}^\top \boldsymbol{x}\right)\right)$$

$$\frac{\partial f}{\partial v_1}$$

$$\frac{\partial f}{\partial w_1^1} = \frac{\partial f}{\partial h_1}\frac{\partial h_1}{\partial w_1^1}$$

Beyond two layers: multilayer neural networks

Why not add more and more layers to learn more complex decision boundaries?

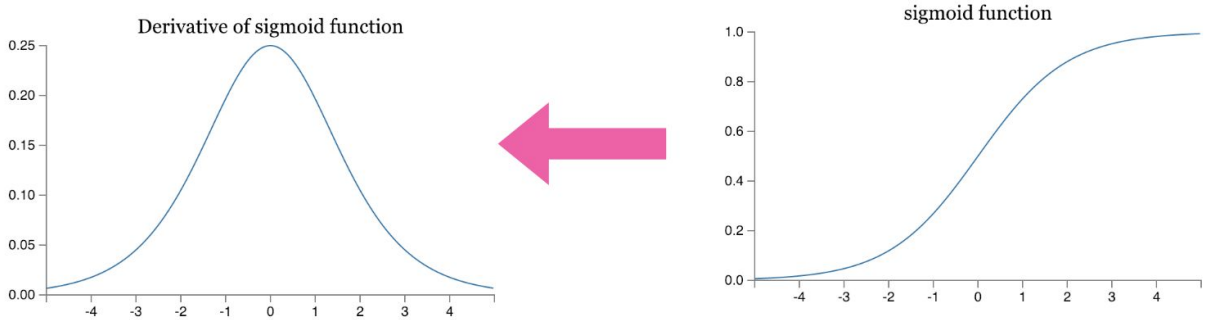


The vanishing and exploding gradient issues

Propagating backwards using the chain rule has its drawbacks in networks with a large number of layers in terms of the stability of the updates.

In particular, the updates in earlier layers can either be negligibly small (**vanishing gradient**), or can be increasingly large (**exploding gradient**) in certain types of neural network architectures.

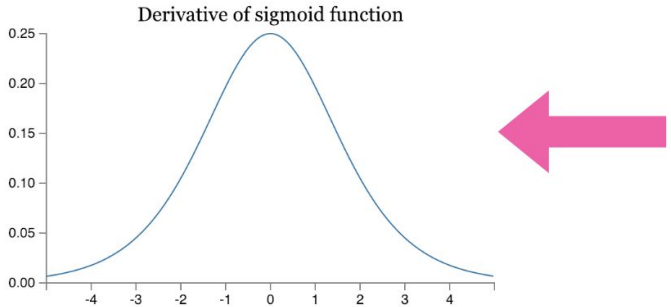
Vanishing gradient

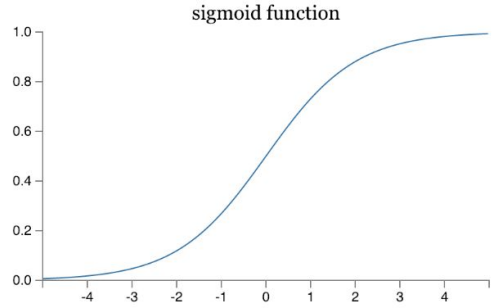


Vanishing gradient

$$f(x) = \sigma\left(w_4\sigma\left(w_3\sigma\left(w_2\sigma\left(w_1x\right)\right)\right)\right)$$

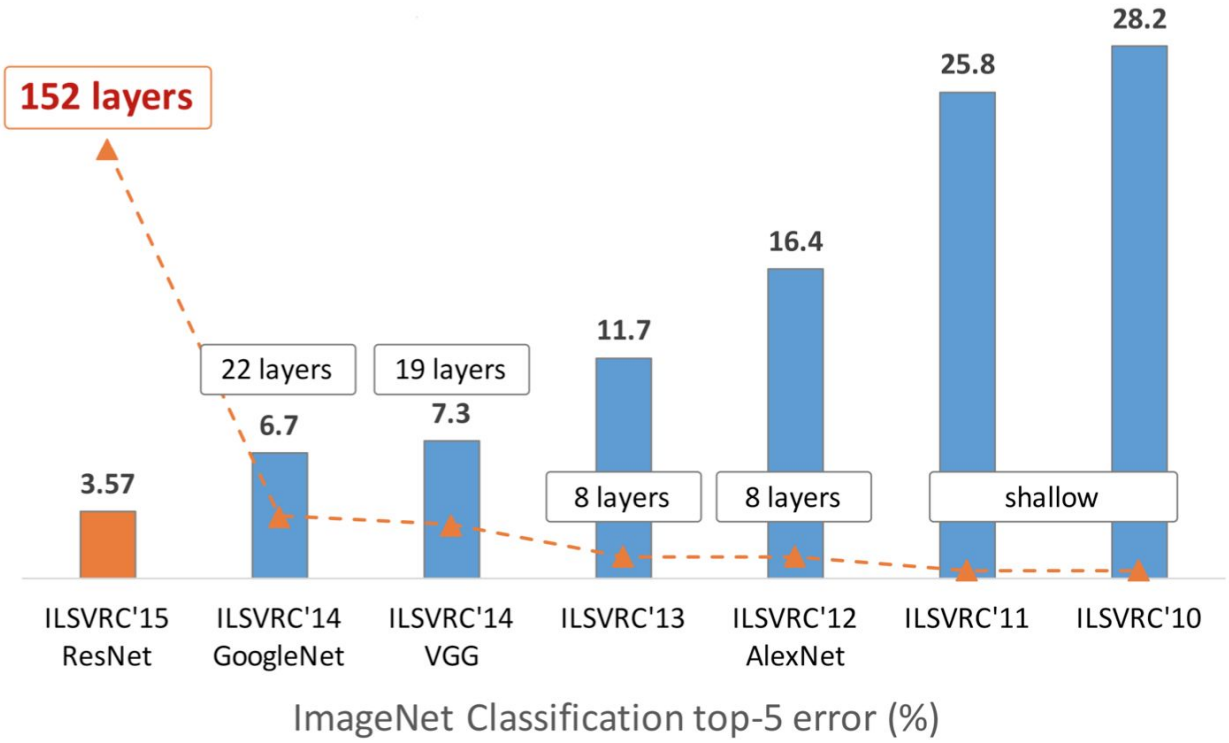
$$\frac{\partial f}{\partial w_1} = \sigma'(h_1) \times w_2 \times \sigma'(h_2) \times w_3 \times \sigma'(h_3) \times w_4 \times \sigma'(h_4) \times x$$





When we take a product of many such terms, the product will tend to exponentially decrease: the more terms, the smaller the product will be (ReLU can helps us to overcome vanishing gradient issue (why?)).

Revolution of depth: deep neural networks (DNNs)



Source: Deep Residual Networks. Kaiming He. 2016

Neural network architecture

Even for a basic Neural Network, there are many design decisions to make

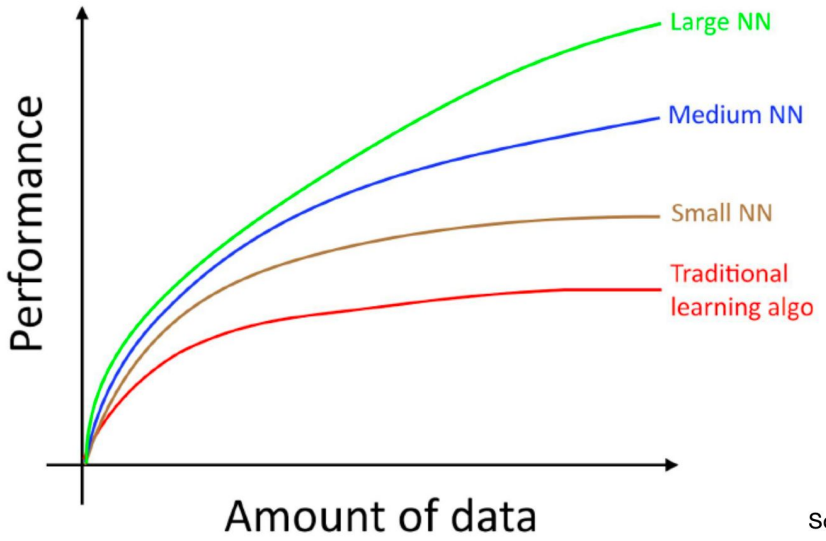
- # of hidden layers (depth)
- # of units per hidden layer (width)
- Type of activation function (non-linearity)
- Form of objective (loss) function
- Type of regularization
- The optimization algorithm and learning rate
- Initialization of solution (can not simply set to zero due to non-convexity)

Challenges of training DNNs

Basically training of a DNN is applying the back-propagation algorithm to thousands of hidden layers with millions of neurons in each layer, but

- Lots and lots of data is needed (high-variance (estimation) error)
- Computational issues (requires high-performance hardware)
- Regularization of DNNS (can overfit anything :-))
- Vanishing and exploding gradients issues
- Hyperparameter optimization Interpretability issues (Neural networks are essentially a Black Box)
- The science of deep learning and lack of theoretical understanding

Bias and variance of DNNs



Source: Machine Learning Yearning, Andrew Ng

The bias of DNNs (approximation error) is zero: Recall universal function approximation

Lots and lots of data is needed (high-variance, i.e. estimation error) to train DNNs

The computational issues

A significant challenge in neural network design is the running time required to train the network.

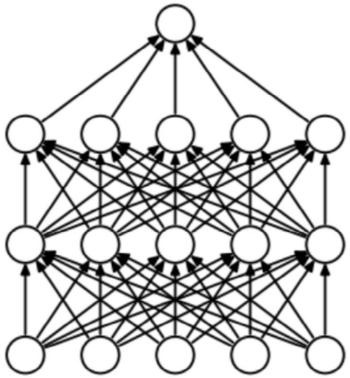
In recent years, advances in hardware technology such as Graphics Processor Units (GPUs) have helped to a significant extent.

GPUs are specialized hardware processors that can significantly speed up the kinds of operations commonly used in neural networks.

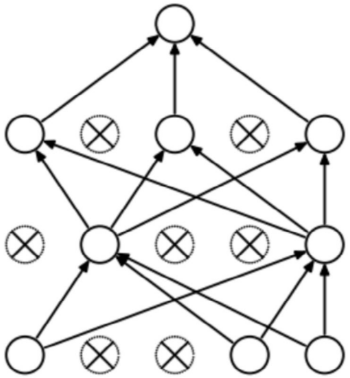
In this sense, some algorithmic frameworks are particularly convenient because they have GPU support tightly integrated into the platform.

Regularization of DNNs

A common regularization is **Dropout**, for each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction of nodes (and corresponding activations).



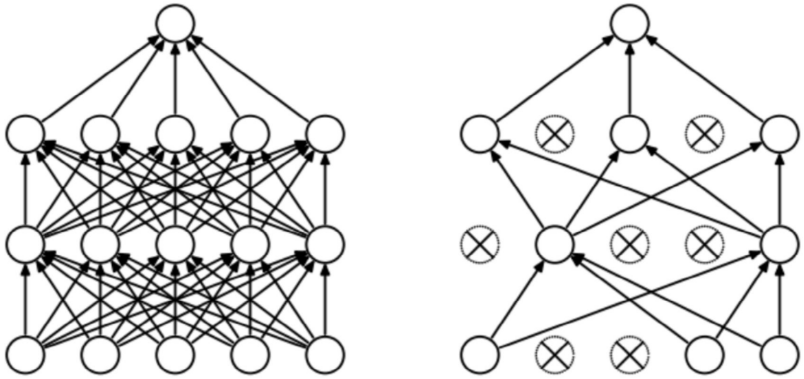
(a) Standard Neural Net



(b) After applying dropout.

Regularization of DNNs

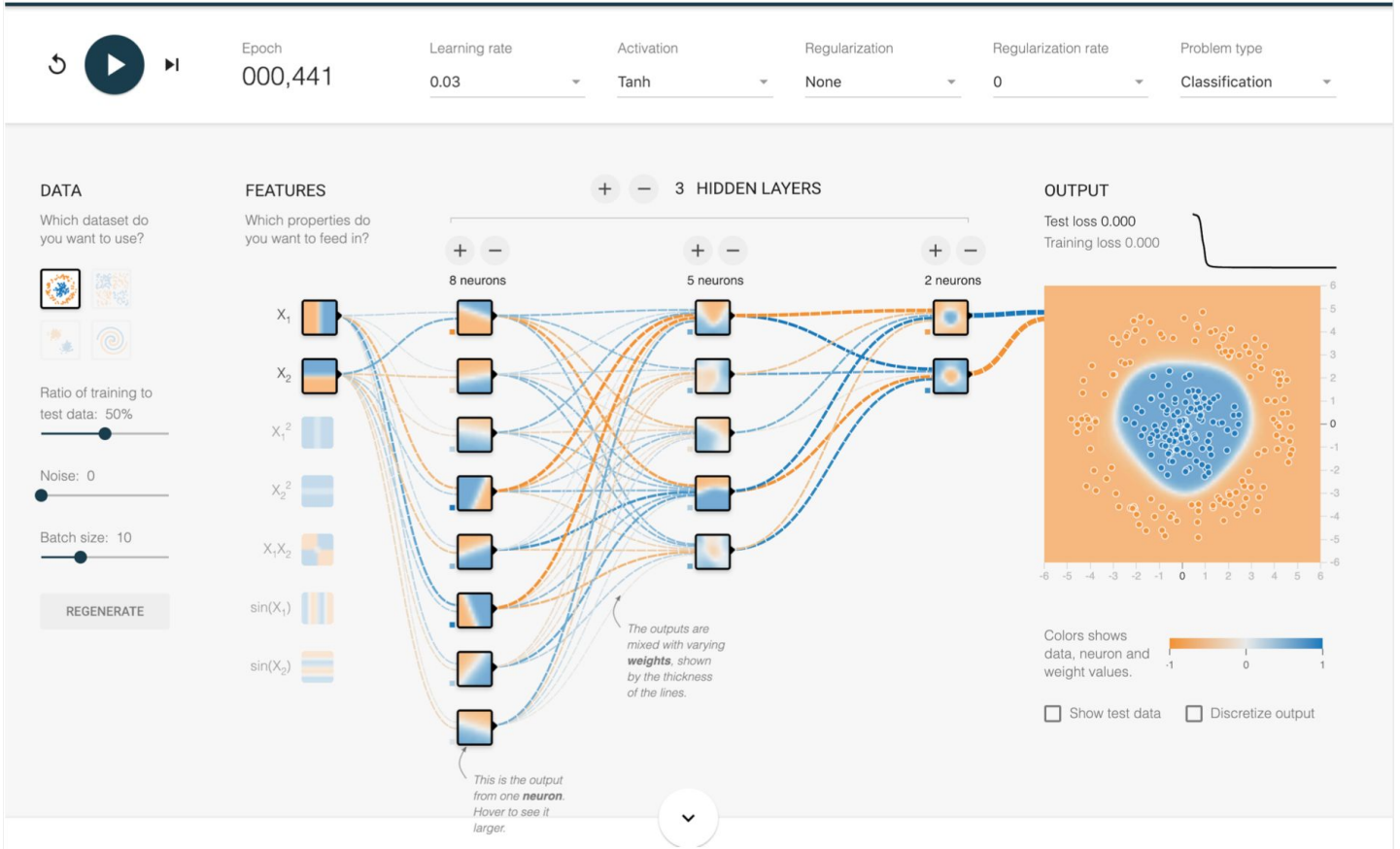
A common regularization is **Dropout**, for each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction of nodes (and corresponding activations).



(a) Standard Neural Net (b) After applying dropout.

Another common form of regularization is **early stopping**, in which the gradient descent is ended after only a few iterations. One way to decide the stopping point is by holding out a part of the training data, and then testing the error of the model on the held-out set. The gradient-descent approach is terminated when the error on the held-out set begins to rise.

Playground (https://playground.tensorflow.org/)



Symbolism vs Connectionism

Two school of thoughts in artificial intelligence (AI)

Symbolism: AI can be achieved by representing concepts as symbols

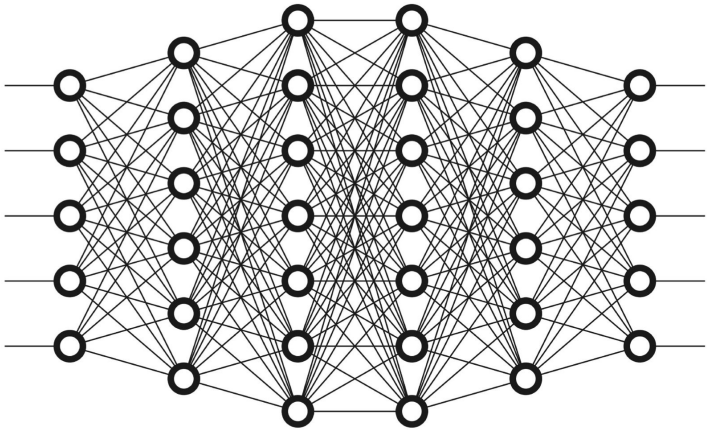
- Example: rule-based expert system, formal grammar

```
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]
```

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

- Example: perceptron, larger scale neural networks



[Tom Mitchell]