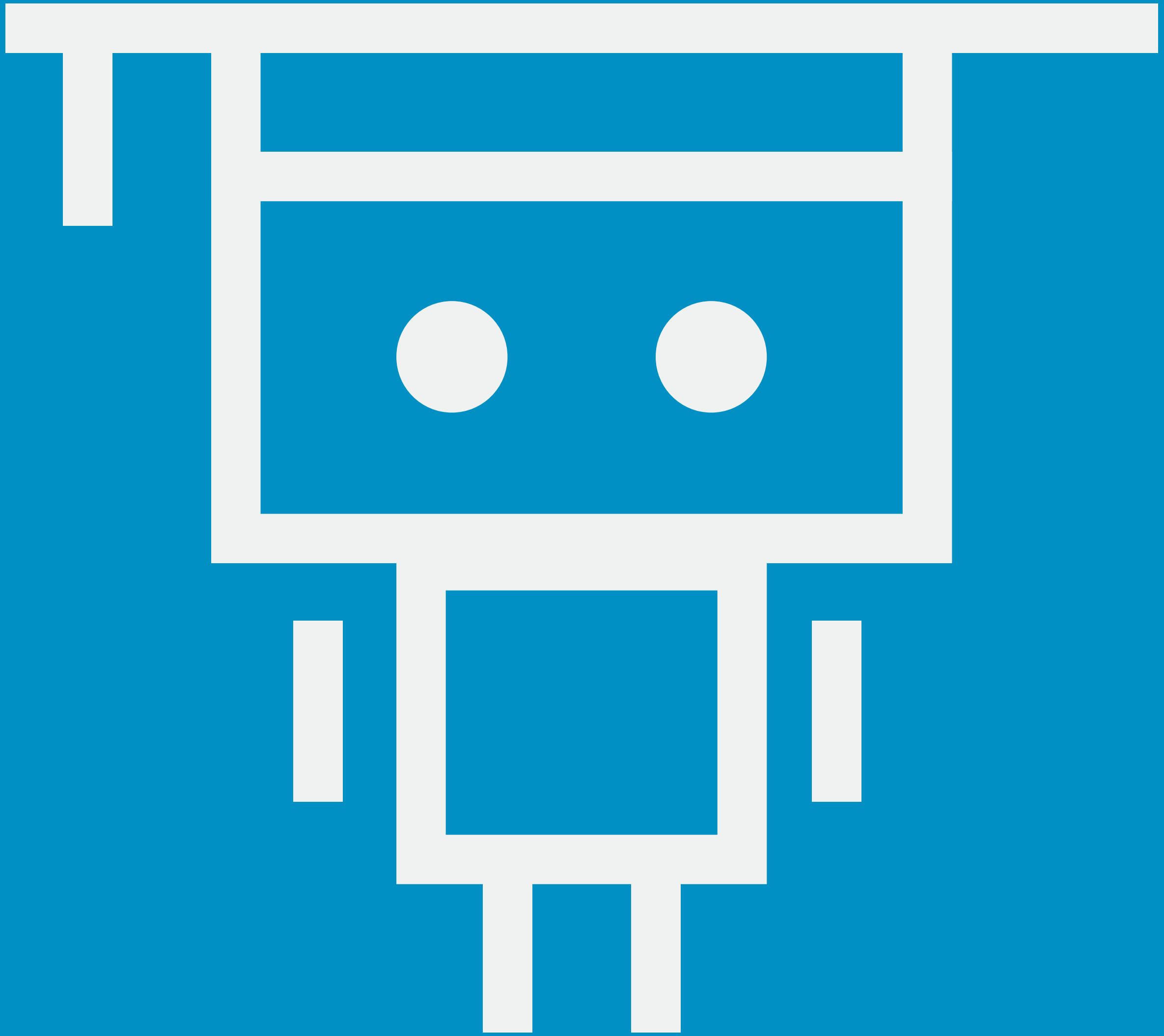


Course Description

- Covers techniques for managing data throughout an end-to-end ML process
- Learn statistical analysis and visualization techniques, including methods for detecting and remedying overfitting
- Gain familiarity with tools in standard ML and data processing libraries

Course Description

- **Lecture 1:** ML Life-cycle, Data Analysis and Basic ML model: K Nearest Neighbors
- **Lecture 2:** Feature Engineering, Decision Trees, Hyper-parameters and AWS Sagemaker
- **Lecture 3:** Optimization, Regression Models, Boosting and Neural Networks

Final Project-Completion

- Apply and experiment with a real-world Amazon dataset.
- Completion of this course is based on your Leaderboard submission.
- Submission page: https://leaderboard.corp.amazon.com/tasks/478
- Submission is open until Saturday 5:00 PM (PST)

Top 3 submissions - this week

Completions

Final Project-Completion

You will get a score after each submission. You can improve your score by making multiple submissions (no upper limit on number of submissions).

- **Requirements:** At least **ONE** submission is **REQUIRED**
- **Non-completion:** Submit 0 models to the Class Leaderboard

Day 1

Day 2

Day 3

Course Complete

After completion, student and manager receive confirmation email.

Topics for today

- Optimization
- Regression
- Boosting
- Neural Networks
- MxNet
- Final Project

Topics for today

- Optimization
- Regression
- Boosting
- Neural Networks
- MxNet
- Final Project

Optimization

- Optimization is an important part of our life.
- For example we:
  - Optimize the order of our work tasks based on priorities.
  - Optimize our route home to minimize our travel time.

- In machine learning, similarly, we use optimization to usually minimize some error rate of our ML model.

- We will learn the Gradient Descent Optimization.

Gradient Descent

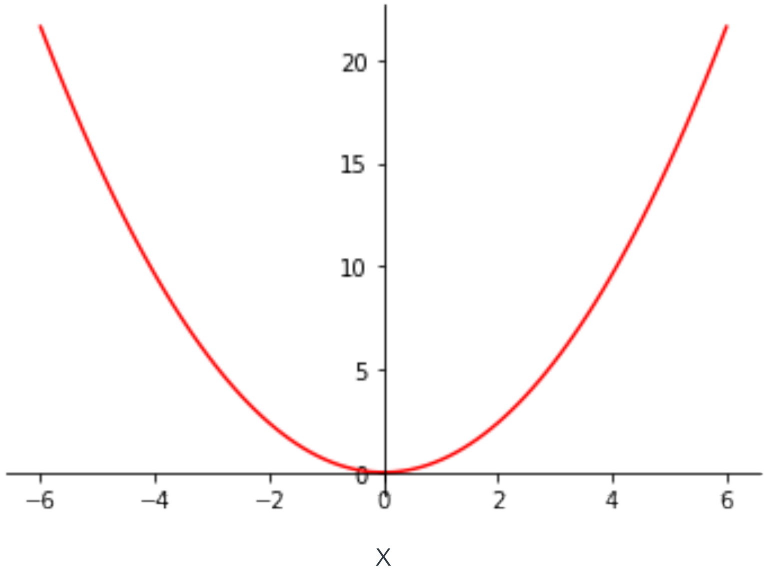
- In machine learning, we usually optimize functions.

  y = f(x) where y=output, x=input, f=function

- **Minimizing f** means **finding** the input **x** that results in **the lowest** value **y**. (Maximize: Find x that gives the largest y)
- **Gradient:**
  - **Definition:** It is the direction and rate of the fastest increase in a function.
  - It can be calculated with **partial derivatives** of the function f with respect to each input variable in x.
  - Because it has a direction and rate, we also call it a “vector”.

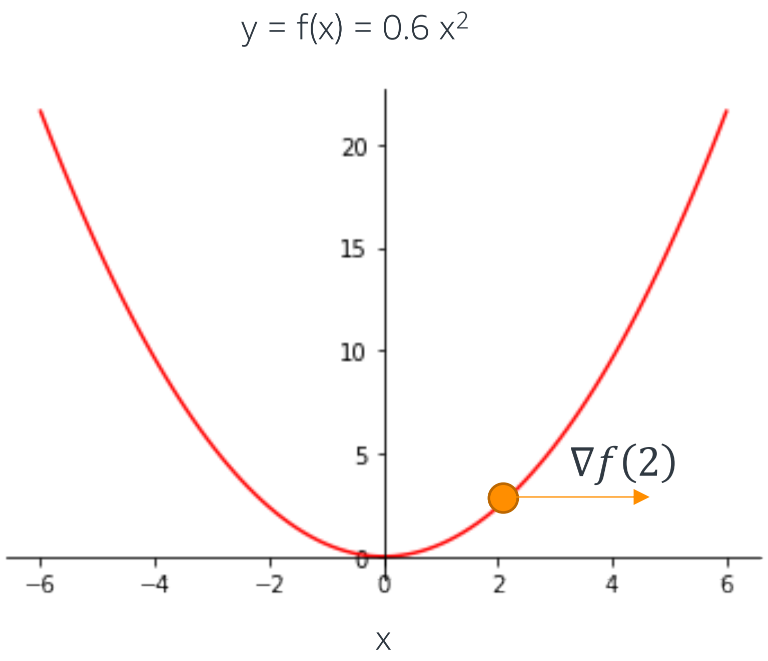
Gradient Example

$y = f(x) = 0.6\ x^2$



$y = f(x) = 0.6\ x^2$ and gradient is $\nabla f = < 1.2x >$

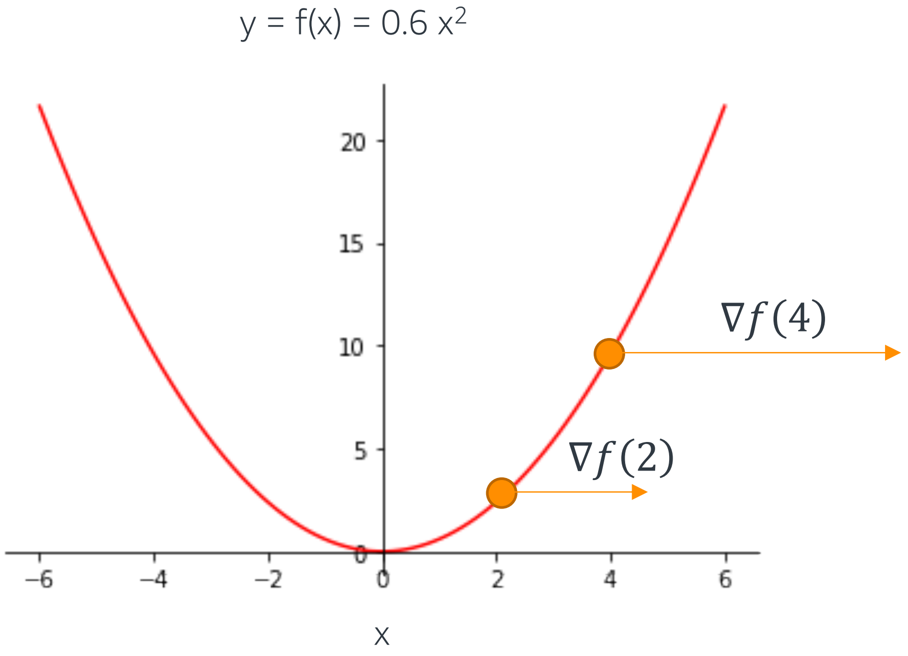
Gradient Example



$y = f(x) = 0.6\ x^2$ and gradient is $\nabla f =< 1.2x >$

$\nabla f(2) =< 2.4 >$

Gradient Example

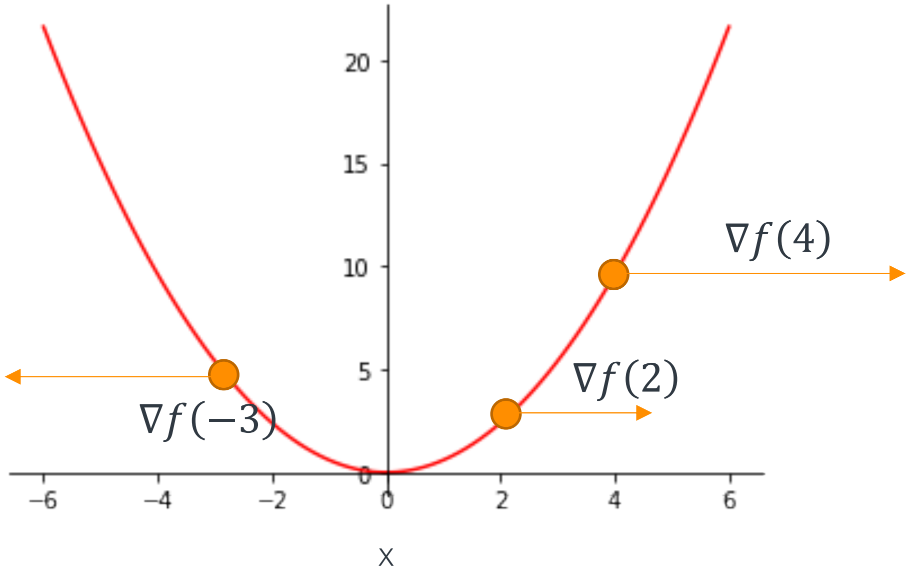


$y = f(x) = 0.6\ x^2$ and gradient is $\nabla f = < 1.2x >$

$\nabla f(2) =< 2.4 >$
$\nabla f(4) =< 4.8 >$

Gradient Example

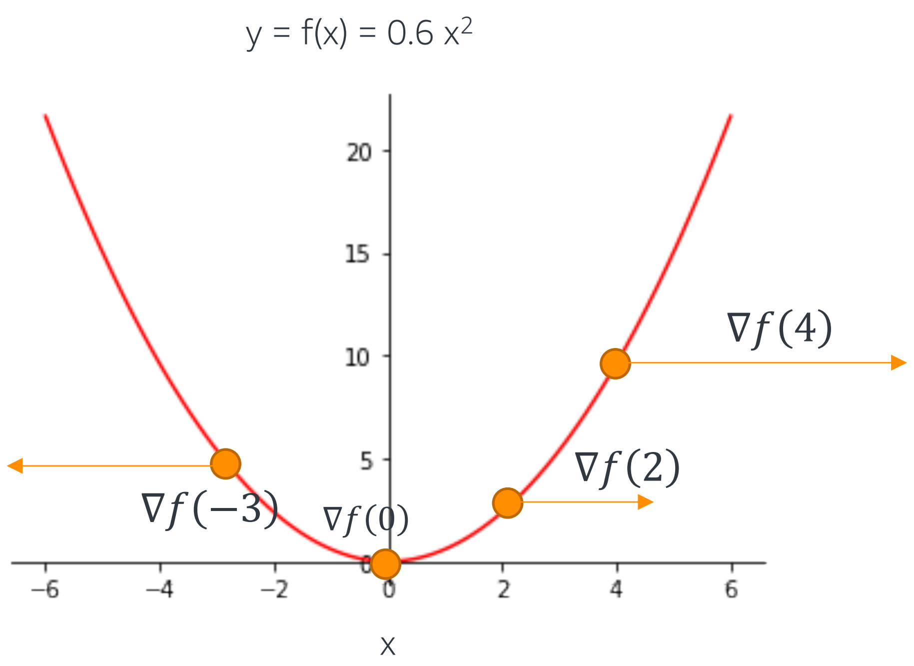
$y = f(x) = 0.6\ x^2$



$y = f(x) = 0.6\ x^2$ and gradient is $\nabla f = < 1.2x >$

$$\nabla f(2) = < 2.4 >$$
$$\nabla f(4) = < 4.8 >$$
$$\nabla f(-3) = < -3.6 >$$

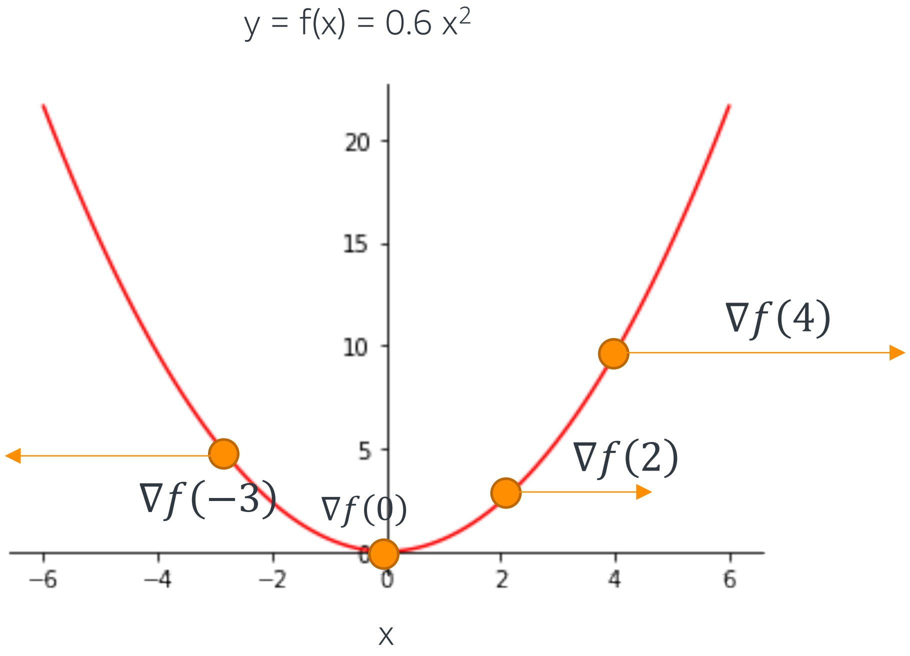
Gradient Example



y = f(x) = 0.6 x^2 and gradient is $\nabla f =< 1.2x >$

$$\nabla f(2) =< 2.4 >$$
$$\nabla f(4) =< 4.8 >$$
$$\nabla f(-3) =< -3.6 >$$
$$\nabla f(0) =< 0 >$$

Gradient Example



y = f(x) = 0.6 x^2 and gradient is $\nabla f =< 1.2x >$

$$\nabla f(2) =< 2.4 > \Rightarrow |\nabla f(2)| = 2.4$$
$$\nabla f(4) =< 4.8 > \Rightarrow |\nabla f(4)| = 4.8$$
$$\nabla f(-3) =< -3.6 > \Rightarrow |\nabla f(-3)| = 3.6$$
$$\nabla f(0) =< 0 > \Rightarrow |\nabla f(0)| = 0$$

- Sign of the gradient shows direction: + right and – left (the direction the func. increases)
- As we go towards to the bottom part of the function, gradient length gets smaller and becomes zero

Gradient - Math

Writing the formula:

$$\nabla f = grad\ f = < \frac{\partial f}{\partial x}(x) >$$

$< >$ shows that gradient is a "vector" here and $\frac{\partial y}{\partial x}$ is used for partial derivative

Example:

$y = f(x) = 0.6\ x^2$

Gradient is $\nabla f = < 1.2x >$

Gradient – Math (Higher Dim.)

Writing the formula:

$$\nabla f = grad\ f = < \frac{\partial f}{\partial x_1}(x_1, x_2, .., x_n), \frac{\partial f}{\partial x_2}(x_1, x_2, .., x_n), .., \frac{\partial f}{\partial x_n}(x_1, x_2, .., x_n) >$$

$< >$ shows that gradient is a "vector" here and $\frac{\partial y}{\partial x}$ is used for partial derivative

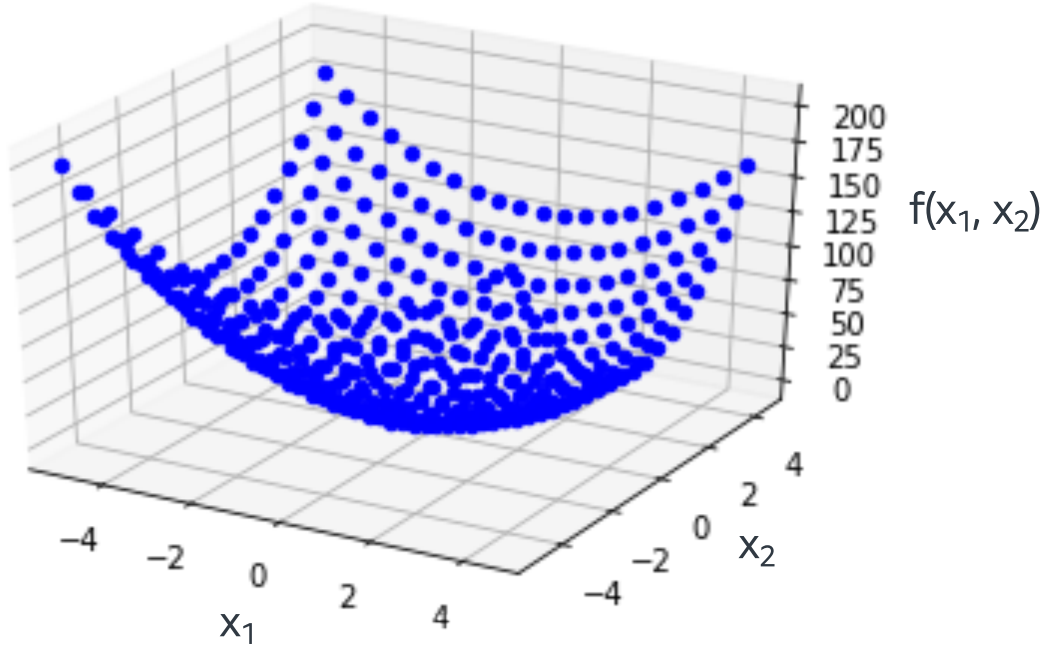
Example:

$y = f(x_1, x_2) = 3\, x_1^2 + 5\, x_2^2$

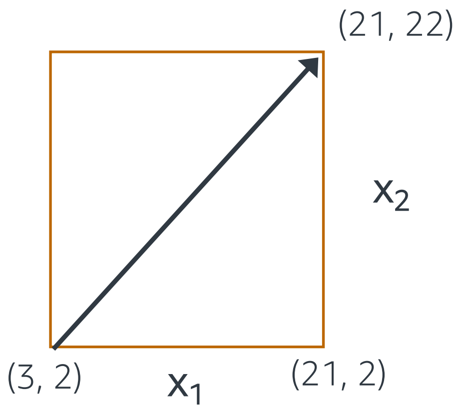
Gradient is $\nabla f =< 6x_1, 10x_2 >$

Gradient – Math (Higher Dim.)

$y = f(x_1, x_2) = 3\,x_1^2 + 5\,x_2^2$ and gradient is $\nabla f = < 6x_1, 10x_2 >$



$\nabla f(3, 2) = <18, 20>$



Gradient Descent Method

- Gradient descent method uses gradients to find the **minimum** of a function **iteratively**.
- We take **small steps** towards the minimum (opposite direction of gradient).

Starting at an initial point x

- Update x_{new}: $x_{current}$ - step_size * gradient

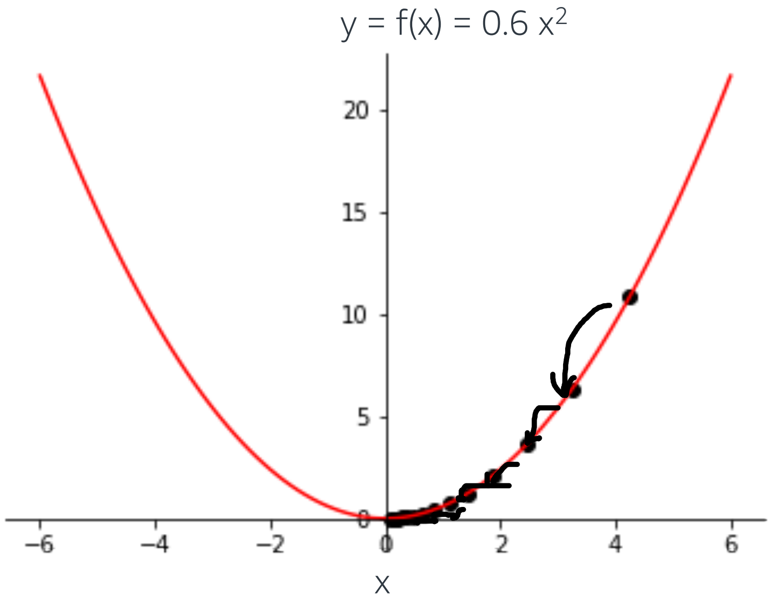
- Let's find the minimum of $y=0.6x^2$, we will start from an initial point assume x=4.25 and step_size=0.2

Gradient Descent Method

- Let's find the minimum of $y=0.6x^2$, we will start from an initial point assume $x=4.25$ and step_size=0.2

Gradient is $\nabla f =< 1.2x >$

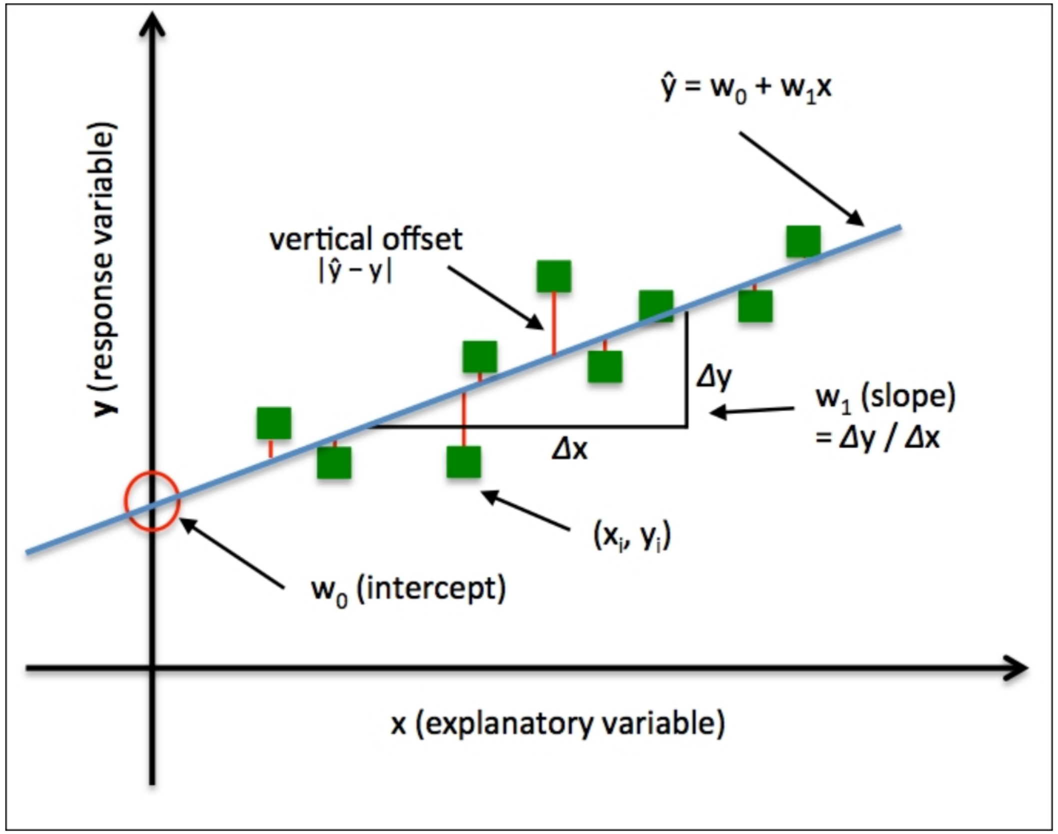
x_{new}: $x_{curr.}$ - step_size * gradient



Topics for today

- Optimization
- Regression
- Boosting
- Neural Networks
- MxNet
- Final Project

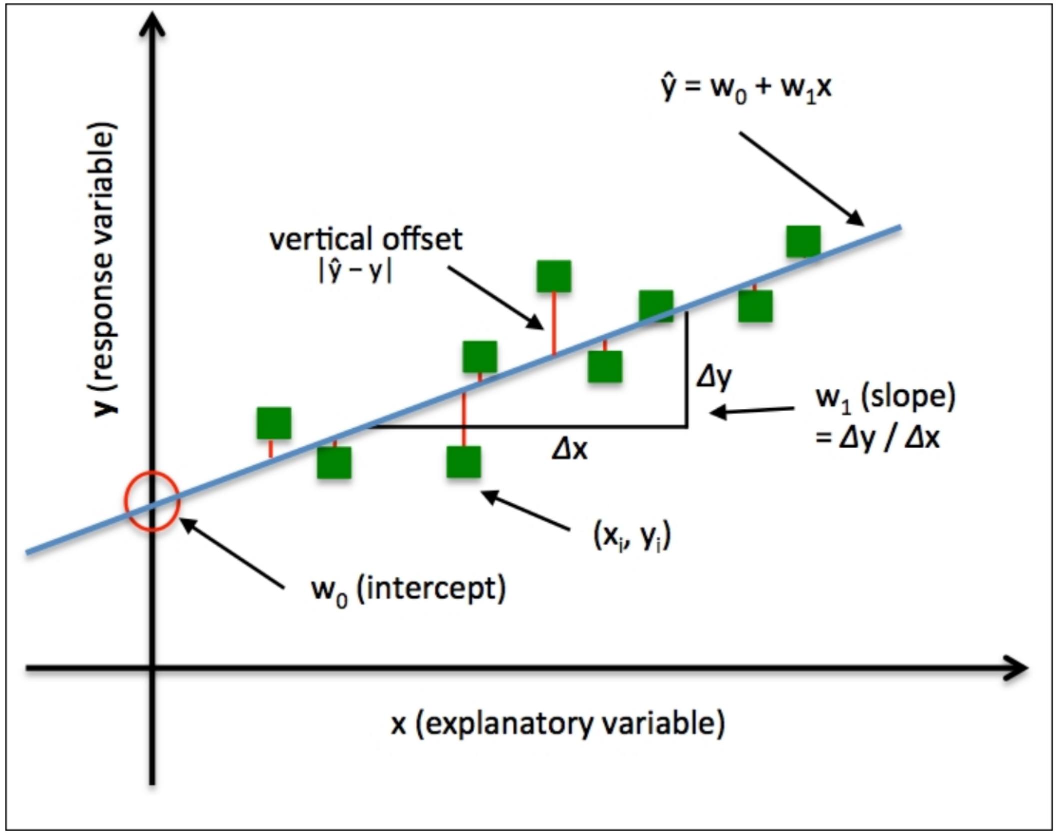
Linear Regression



- We use it for numerical value prediction.
- For example: Predicting house prices by looking at square footage, number of rooms etc.
- When there is a single feature (explanatory variable x) and a real-valued response (target variable y):

$$y = w_0 + w_1 x$$

Linear Regression



- Given data (x, y), a line: $\boldsymbol{y = w_0 + w_1 x}$ is defined by $\boldsymbol{w_0}$ (i.e. intercept) and $\boldsymbol{w_1}$ (slope)
- The vertical offset for each data point from the line is the error between the true label y and the prediction based on x.
- The best line minimizes the sum of squared errors (SSE): $\sum(y_i - \hat{y}_i)^2$

Linear Regression

- Multiple linear regression includes m features with m >= 2:

$$y = w_0 + w_1 x_1 + \ \dots + w_m x_m$$

- Sensitive to correlation between features resulting in high variance of coefficients.
- Features are allowed to have interactions and higher order terms.

Gradient Descent - Fitting a model

- Let's have a linear regression model:

$$y = w_0 + w_1x_1 + \ w_2x_2 + \cdots + w_qx_q$$

where w_0, w_1, .., w_m: coefficients and x_0, x_1, .., x_m: input variables

- Minimize the cost function **Mean Squared Error**:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$ where $\boldsymbol{y_i}$: Data value, $\boldsymbol{\hat{y}_i}$: Predicted value and n: # of records

- Iteratively update coefficients with Gradient Descent:

$w_{new} = w_{current}$ – step_size*gradient

Logistic Regression

Linear regression was useful when predicting continuous values.

$$y = \boldsymbol{w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_q x_q}$$

Can we use a similar approach to solve classification problems?

The most simple classification problem is binary classification y=0 or 1:

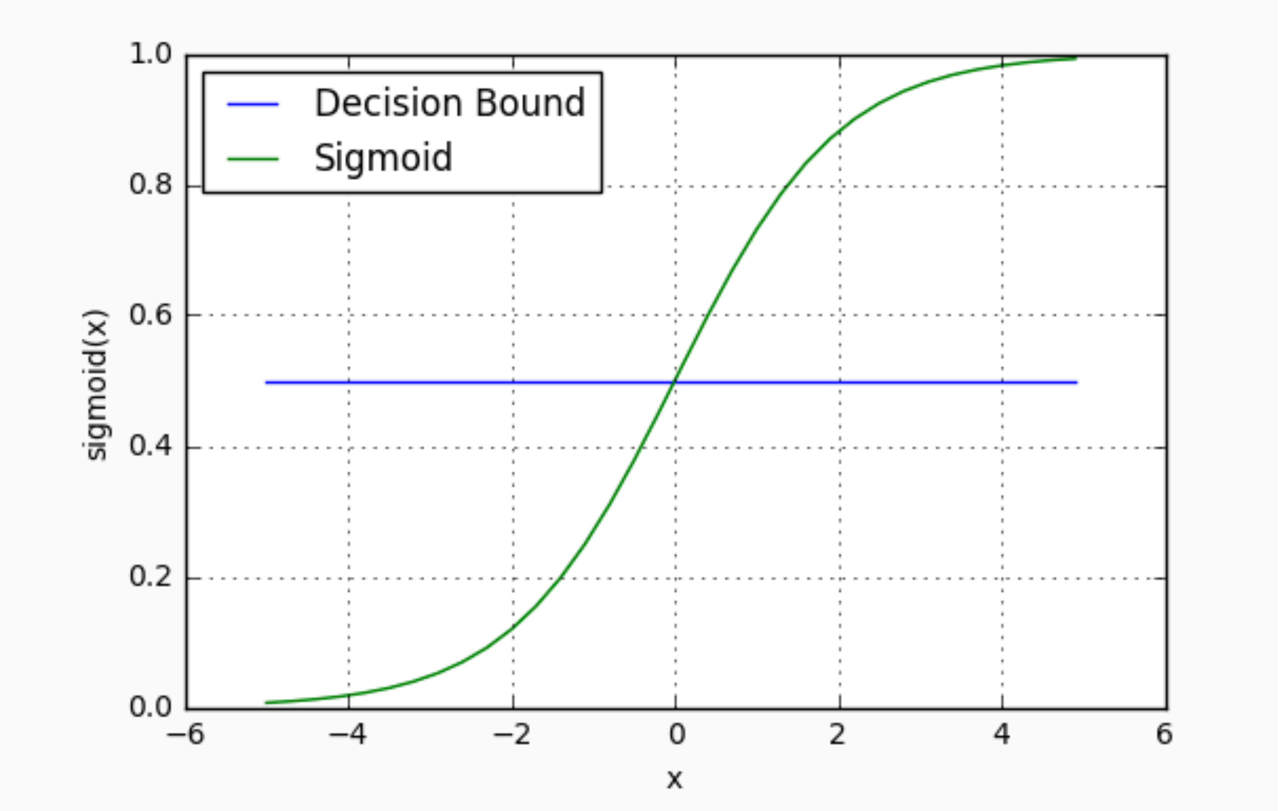
Examples:

- Email: Spam or Not spam
- Text: Positive or negative meaning
- Image: Hotdog or Not hotdog

How can we modify this linear regression equation to work for this?

$$y = \boldsymbol{w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_q x_q}$$

Applying Sigmoid Function



- $sigmoid(x) = \frac{1}{1+e^{-x}}$
- We can classify with this:
  - if $sigmoid(x) < 0.5$, round down:(0)
  - if $sigmoid(x) >= 0.5$, round up:(1)
- Our regression function becomes:

$$h_w(x) = p = \frac{1}{1 + e^{-(w_0+w_1x_1+w_2x_2+\cdots+w_qx_q)}}$$

We can say p is the probability that y=1 for input x.
Classify as 1 if p>=0.5 , as 0 if p<0.5

Log-loss (Binary Cross-entropy)

Log-loss: A numeric value that measures the performance of a binary classifier when the output of the model is probability between 0 and 1.

- We prefer small values for the loss (we try to minimize it). A loss of 0 means the perfect classifier.
- Log loss gets larger when the predicted probability diverges from the actual label (0 or 1).
- In mathematical terms:

$$Logloss = -(y * log(p) + (1 - y) * \log(1 - p))$$

$$where$$

$$y: True\ class, p: Prob.\ of\ the\ model, log: Logarithm\ function$$

Log-loss (Binary Cross-entropy)

Example: Let's calculate the log-loss for these situations below

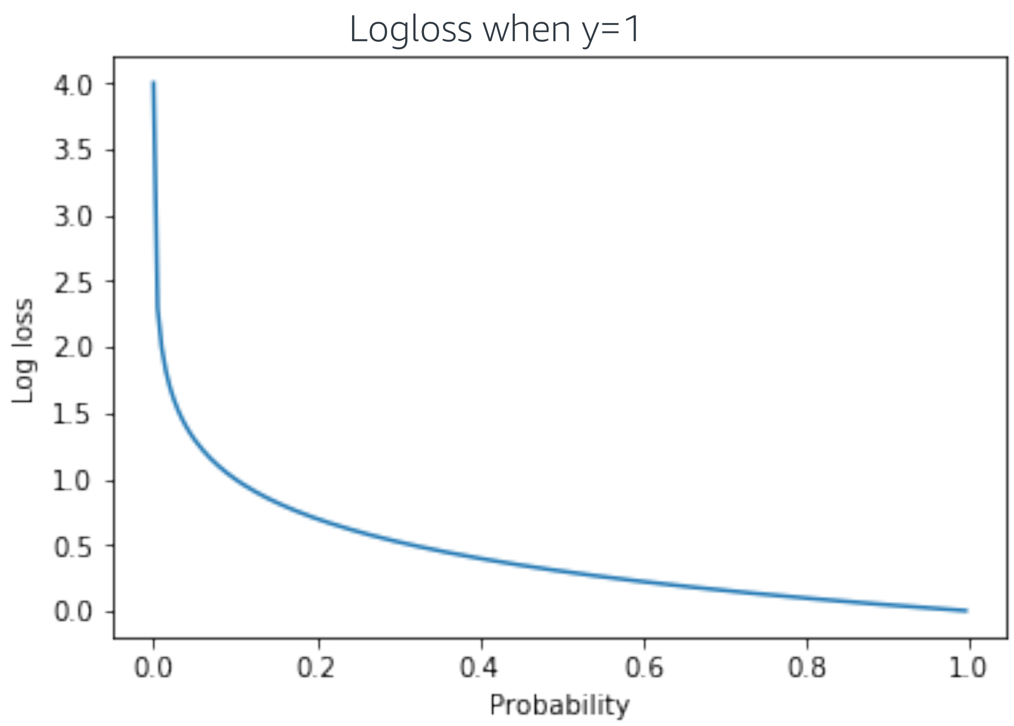
$$Logloss = -(y * log(p) + (1 - y) * log(1 - p))$$

- y=True class:1 and p: 0.3

$$-(1 * log(0.3) + (1 - 1) * log(0.7)) = 0.52$$

- y=True class:1 and p: 0.8

$$-(1 * log(0.75) + (1 - 1) * log(0.25)) = 0.1$$



Log-loss (Binary Cross-entropy)

Example: Let's calculate the log-loss for these situations below

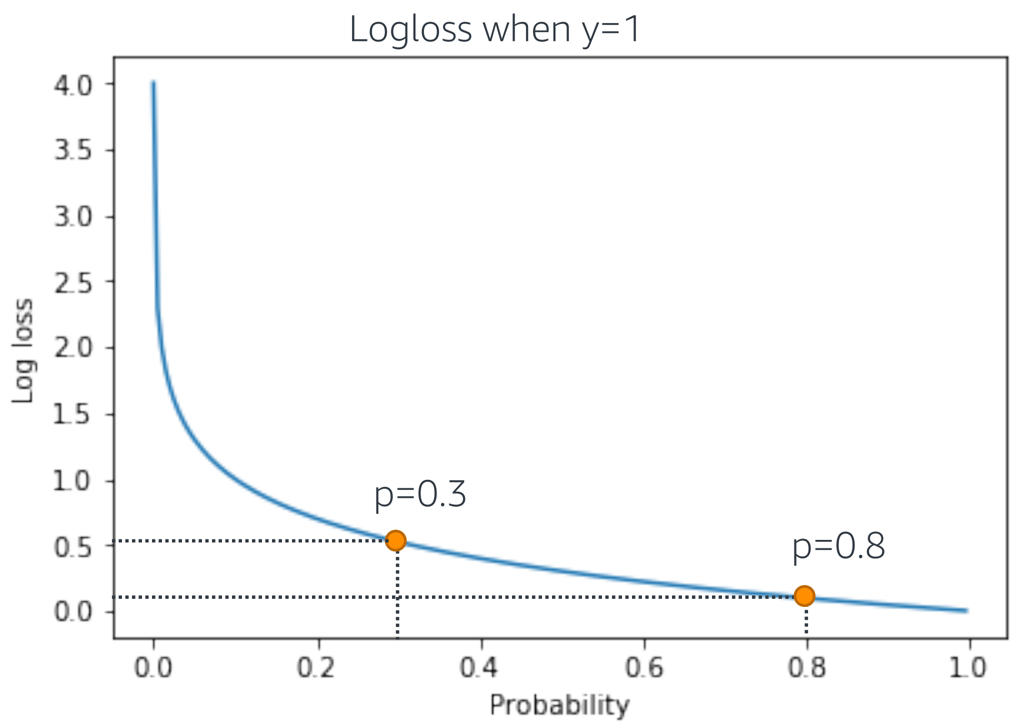
$$Logloss = -(y * log(p) + (1 - y) * log(1 - p))$$

- y=True class:1 and p: 0.3

$$-(1 * log(0.3) + (1 - 1) * log(0.7)) = 0.52$$

- y=True class:1 and p: 0.8

$$-(1 * log(0.75) + (1 - 1) * log(0.25)) = 0.1$$



Log-loss (Binary Cross-entropy)

Example: Let's calculate the log-loss for these situations below

$$Logloss = -(y * log(p) + (1 - y) * log(1 - p))$$

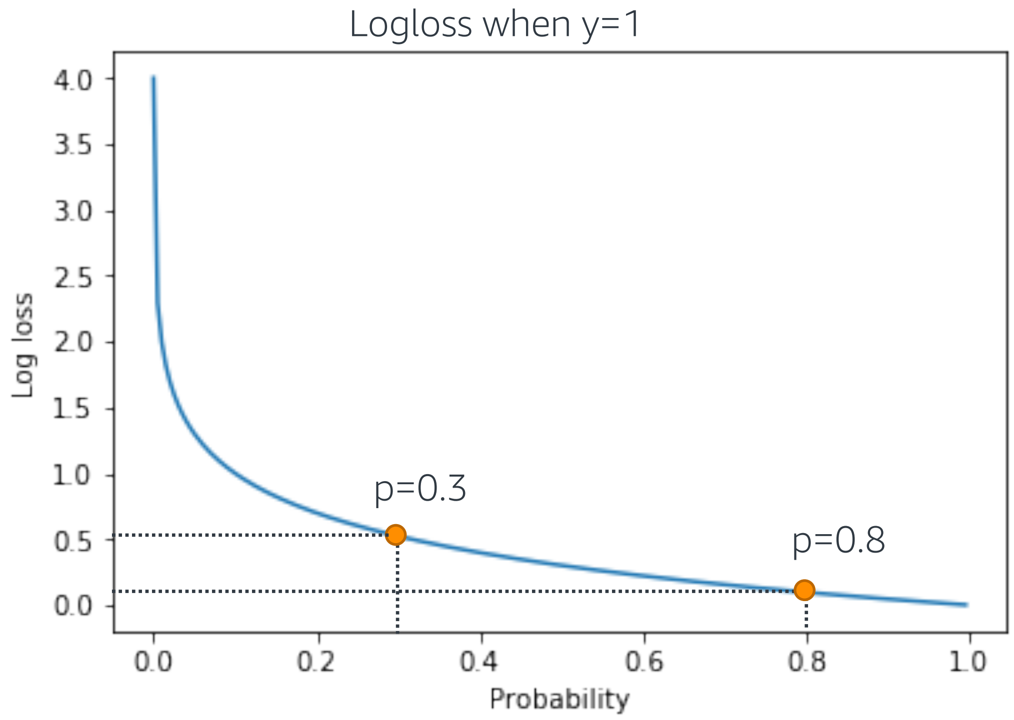
- y=True class:1 and p: 0.3

$$-(1 * log(0.3) + (1 - 1) * log(0.7)) = \boxed{0.52}$$

- y=True class:1 and p: 0.8

$$-(1 * log(0.75) + (1 - 1) * log(0.25)) = \boxed{0.1}$$

Better prediction gives smaller loss



MACHINE LEARNING UNIVERSITY

Gradient Descent - Fitting a model

- Let's have our logistic regression model:

$$y = \text{sigmoid}(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_q x_q)$$

where $w_0, w_1, .., w_m$: coefficients and $x_0, x_1, .., x_m$: input variables

- Minimize the cost function: **Log-loss**:

$$\text{Logloss} = \sum_{i=1}^{n} -(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i))$$

$$where$$

$$i: index, n: \#\ number\ of\ records, y_i: True\ label, y_i: \text{Probability of prediction}$$

- Iteratively update coefficients with Gradient Descent:

$w_{new} = w_{current} - \text{step\_size}*\text{gradient}$

Topics for today

- Optimization
- Regression
- Boosting
- Neural Networks
- MxNet
- Final Project

Adaptive Boosting (AdaBoost)

Main idea of boosting: Instead of learning a single model, learn multiple weak models that are good at different parts of the data.

It works as below:

- Output of the system (classification) is a weighted sum of individual models.
- We will first use a model using our training data. Then, create a second model that tries to correct initially misclassified samples.
- Add more models until all data correctly classified or we reach the max. number of iterations.

Adaptive Boosting (AdaBoost)

On each iteration t:

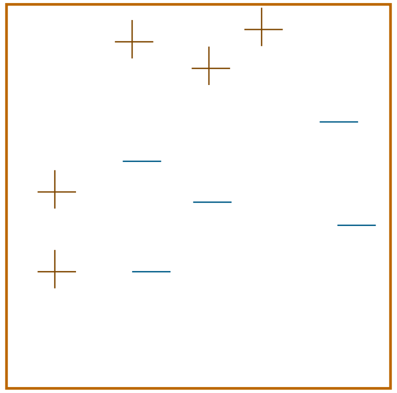
- Learn a weak model h_t and calculate its error rate $\boldsymbol{\epsilon_t}$
- Assign the strength of this weak model: $\boldsymbol{\alpha_t}$
- Assign a weight **D(i)** to each data point i that is correlated to how incorrectly classified (We will use $\boldsymbol{\epsilon_t}$ and $\boldsymbol{\alpha_t}$).

Final classifier: $H(X) = sign(\sum_{t=1}^{T} \alpha_t \, h_t(X))$

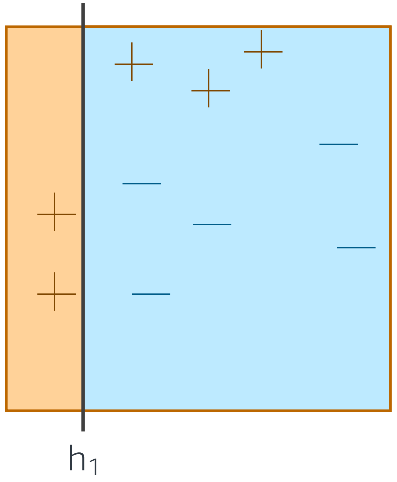
MACHINE LEARNING UNIVERSITY

AdaBoost – Example

D_1: Dataset

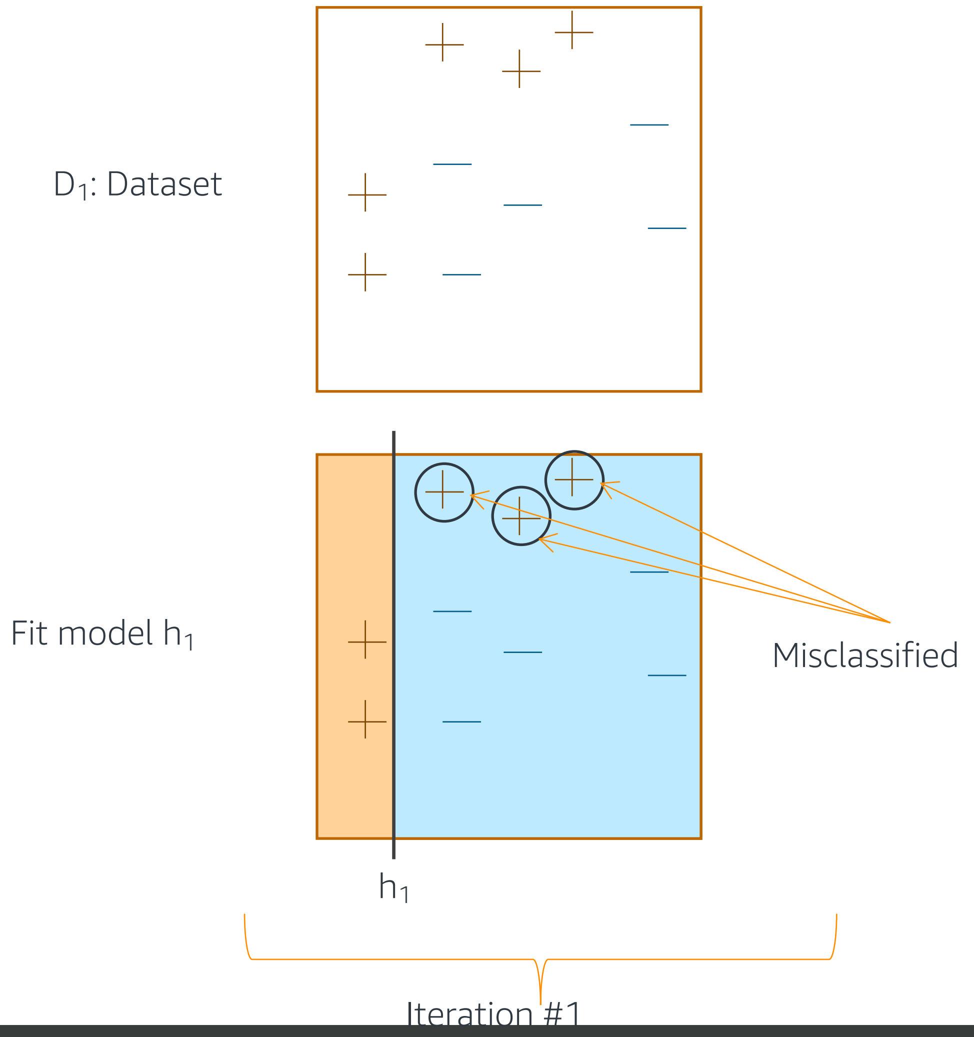


Fit model h_1

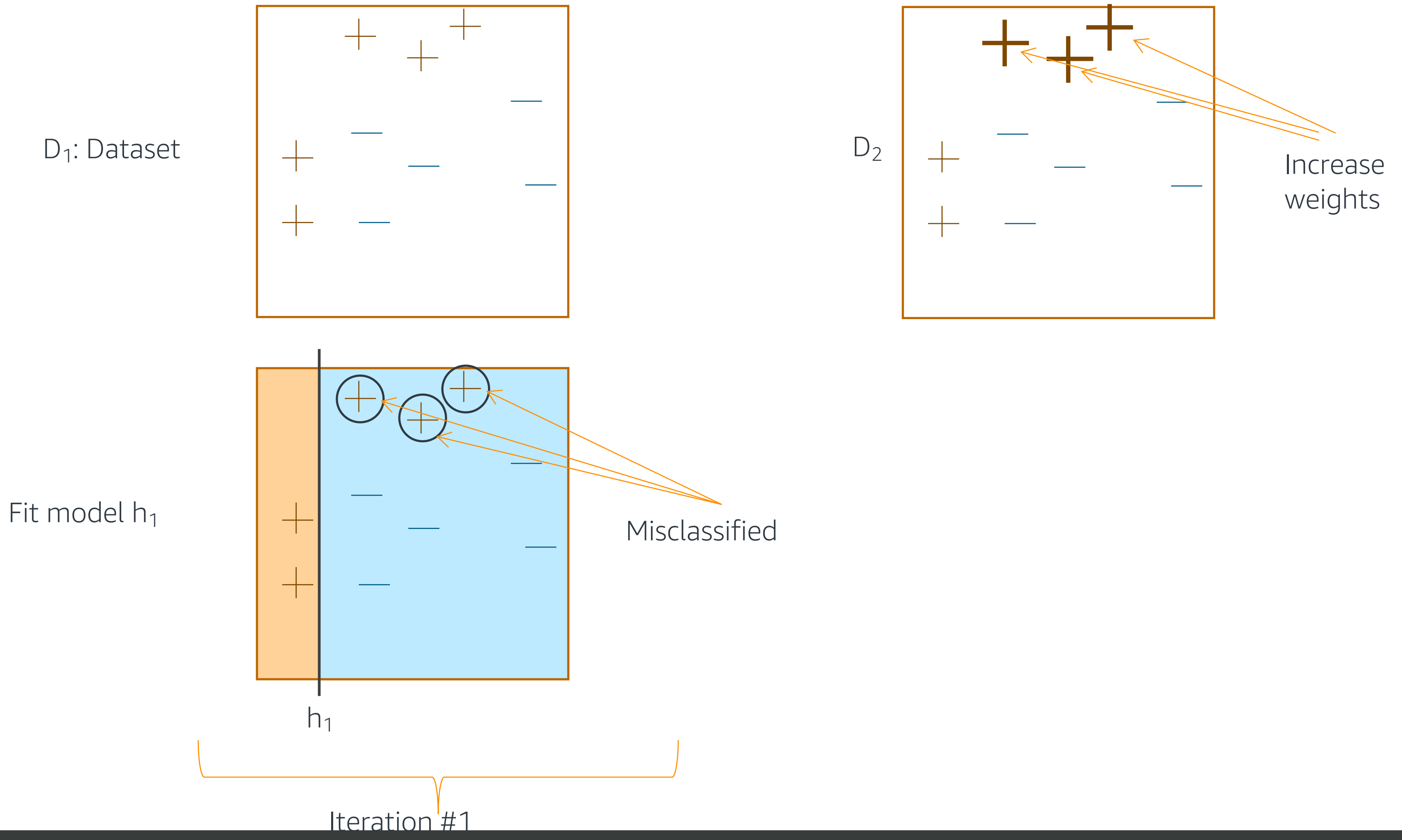


Iteration #1

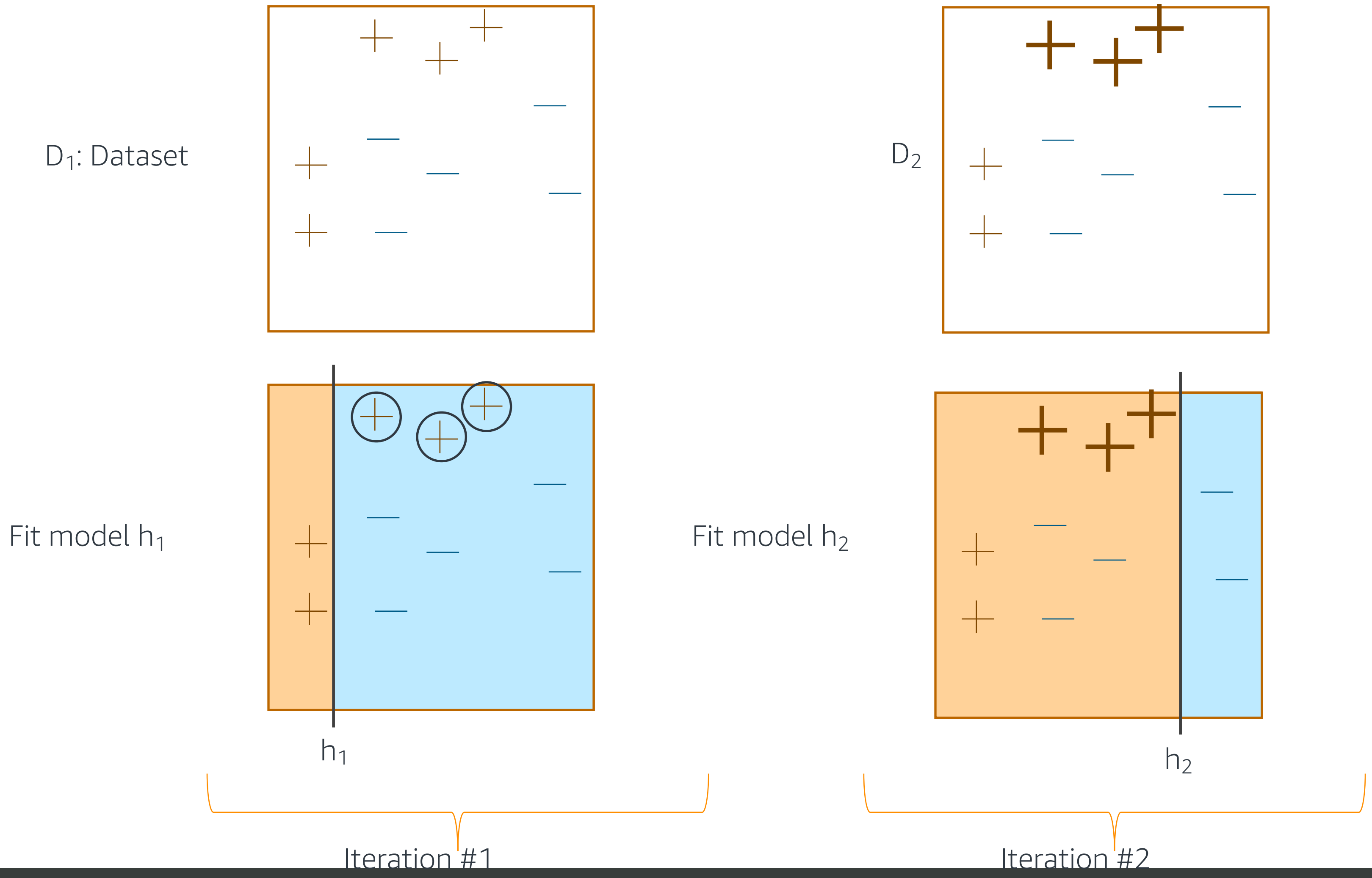
AdaBoost – Example



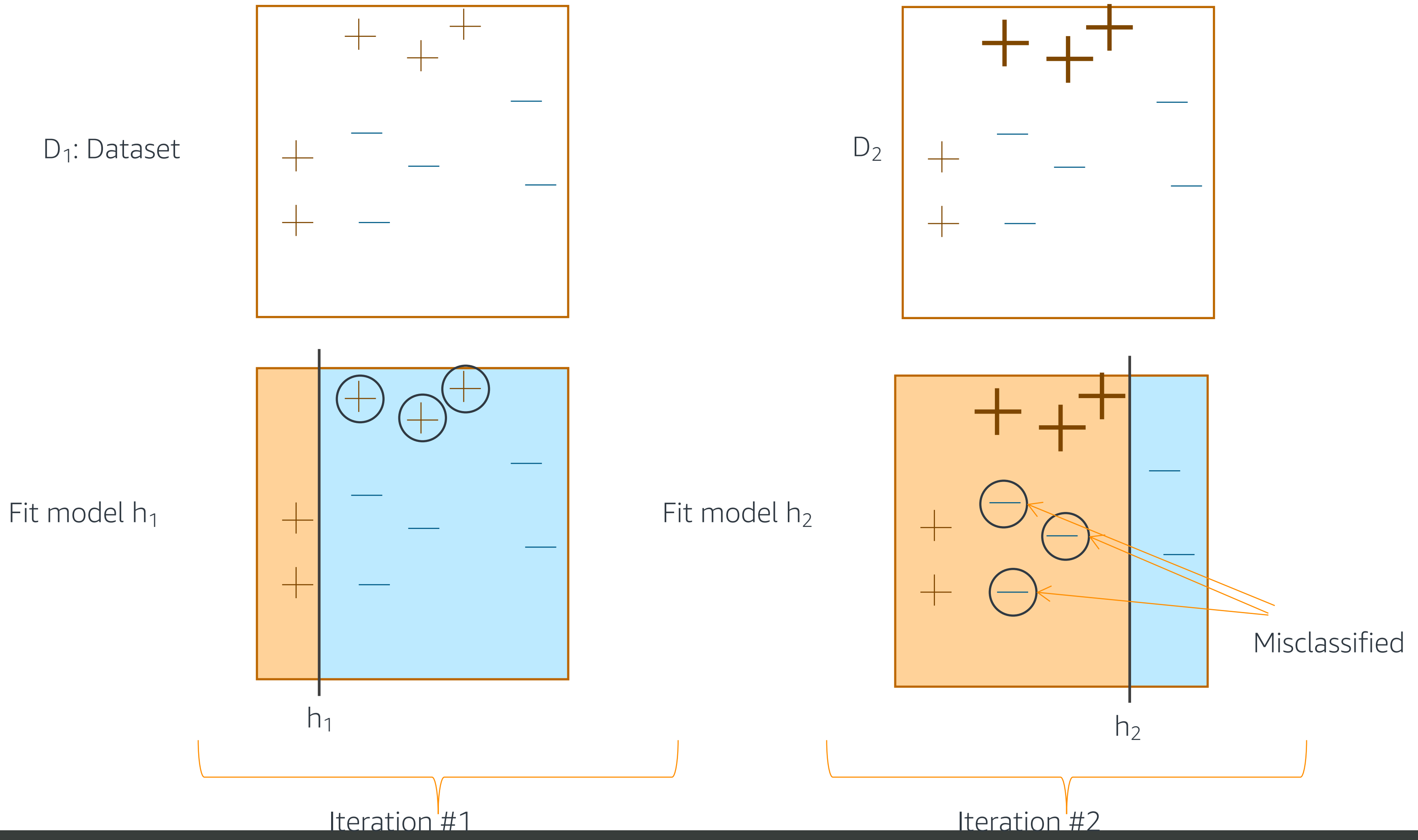
AdaBoost – Example



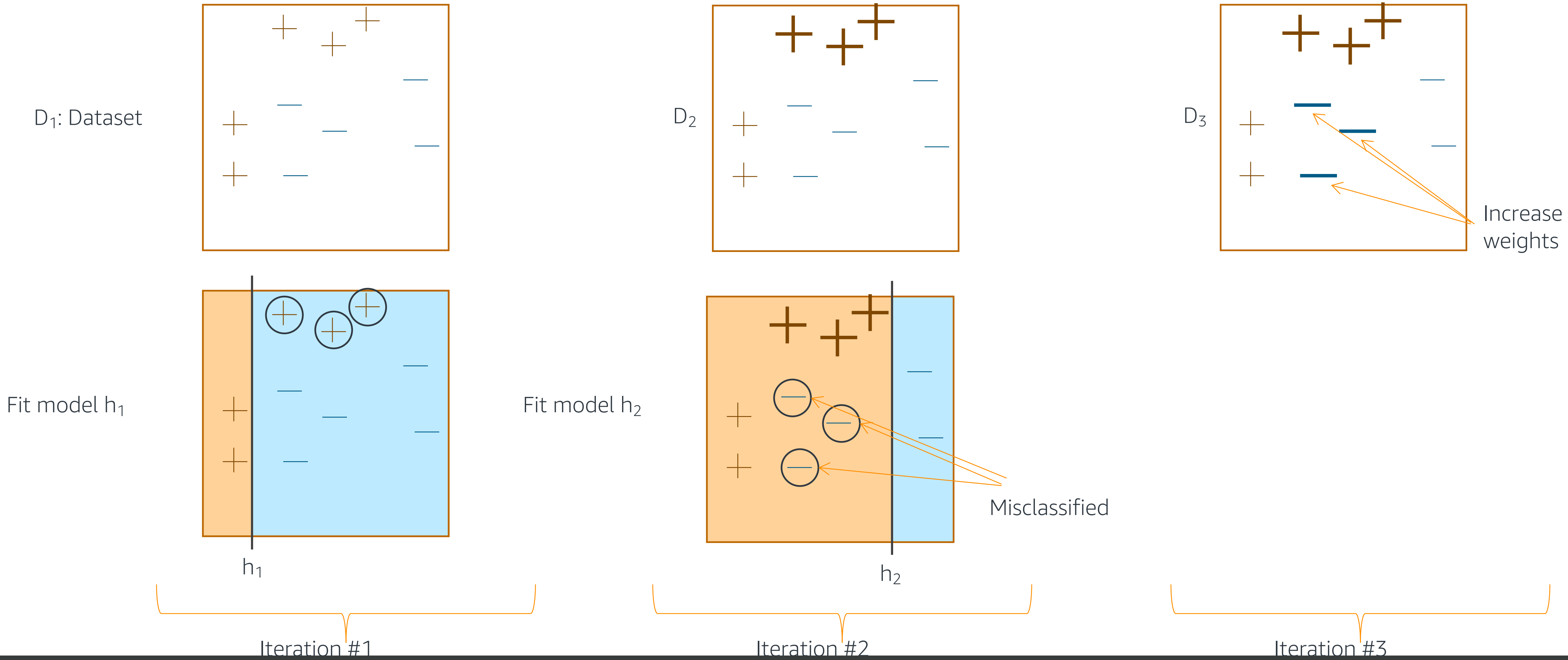
AdaBoost – Example



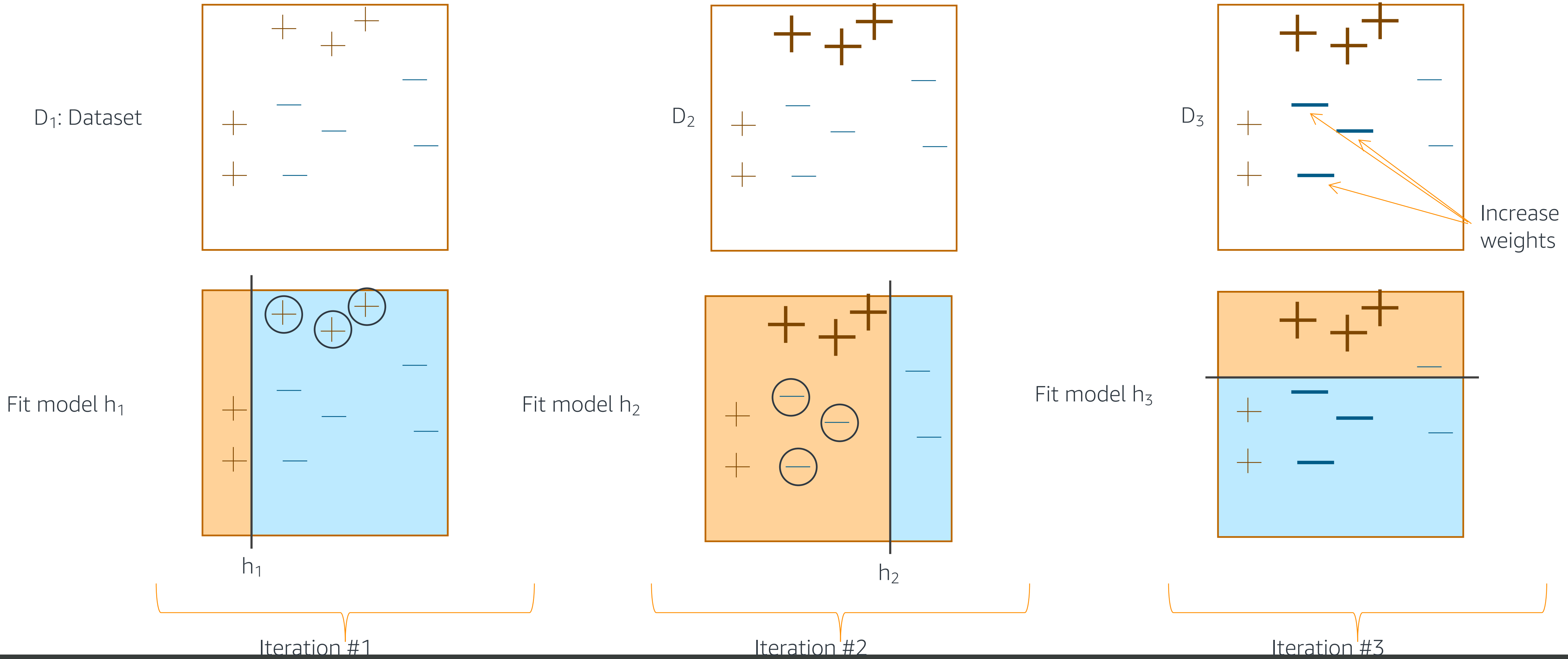
AdaBoost – Example



AdaBoost – Example

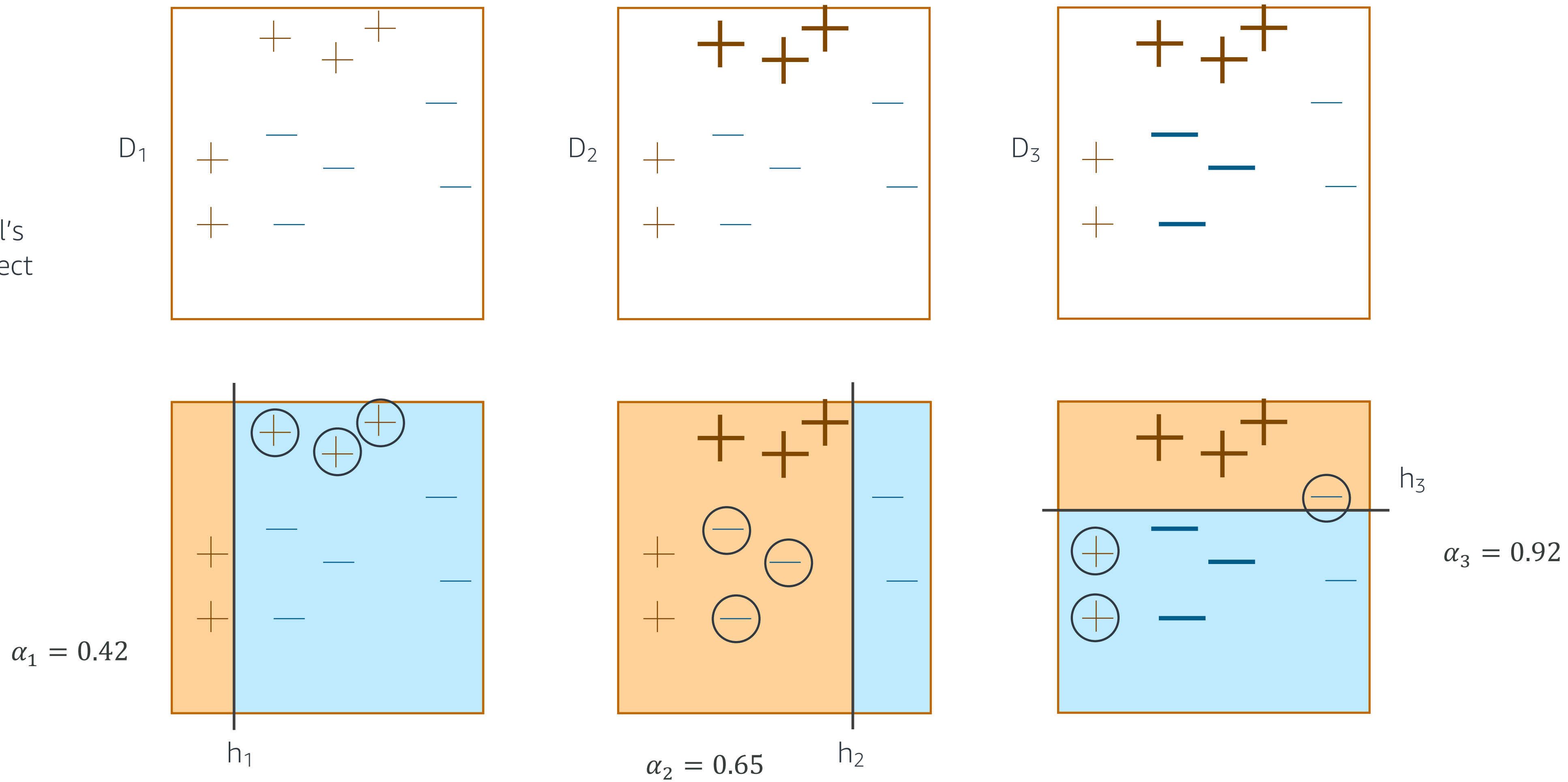


AdaBoost – Example



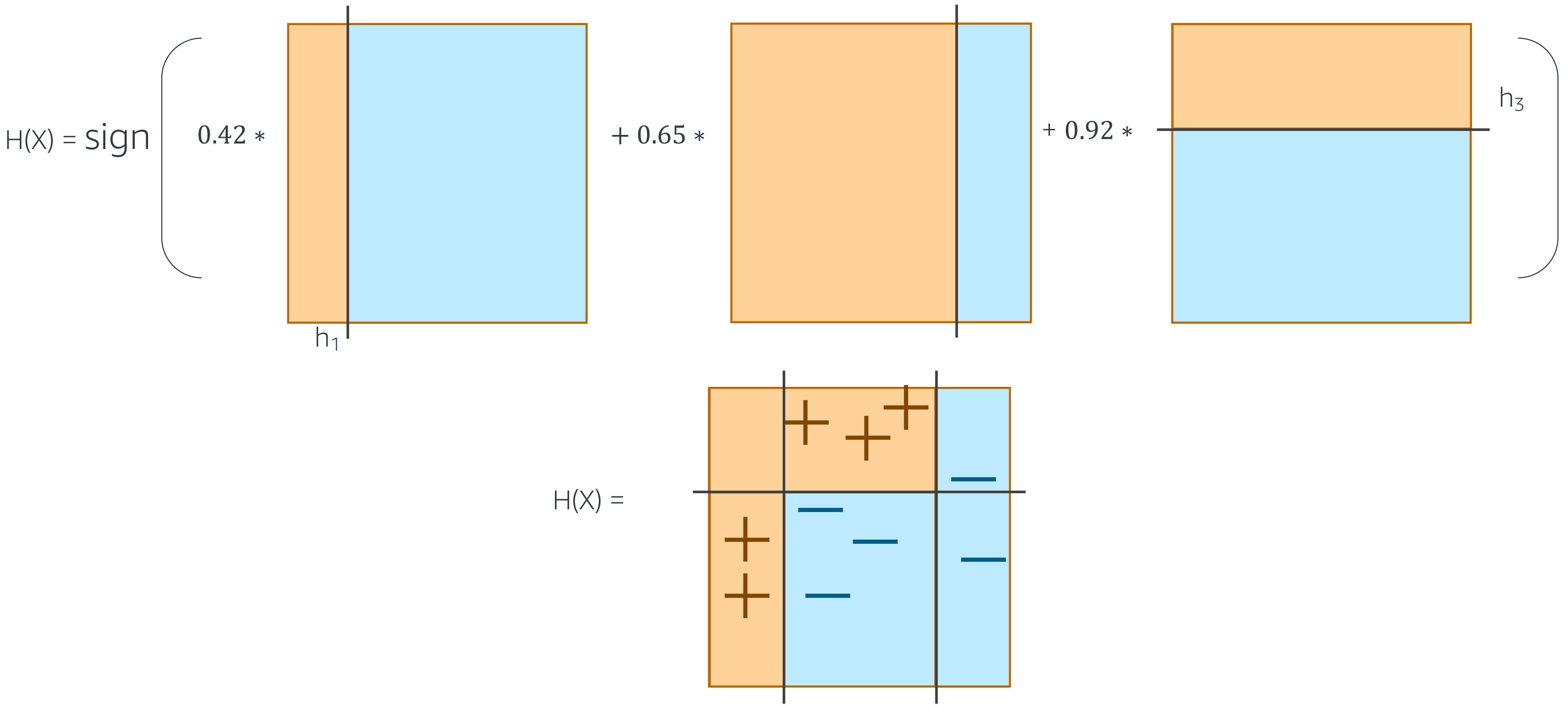
AdaBoost – Example

We also calculate each model's strength: $\boldsymbol{\alpha}$ based on its correct predictions



AdaBoost – Example

Final classifier: $H(X) = sign(\sum_{t=1}^{3} \alpha_t \, h_t(X))$



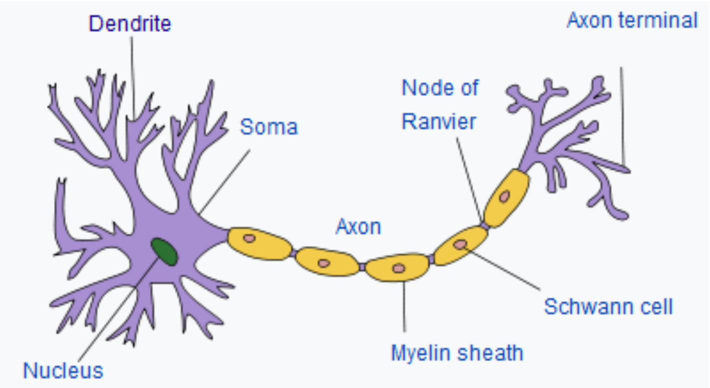
Topics for today

- Optimization
- Regression
- Boosting
- Neural Networks
- MxNet
- Final Project

Why Neural Networks?

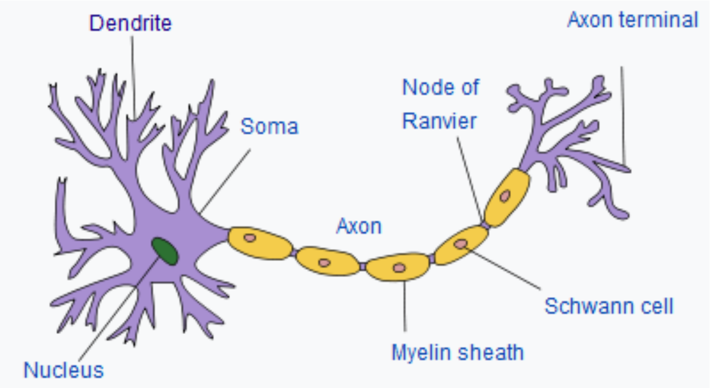
- In the last 5-10 years, neural networks achieved **state-of-the art results** in many different machine learning areas.
- Neural networks can automatically **extract useful features** from input data. Traditionally feature engineering has been the most laborious and unprincipled part of building ML models
- When using neural networks, we should be careful about **model complexity** and make sure we have enough amount of data for training.

Real Neural Networks



- **Neuron** = cell that processes and transmits information through electrical and chemical signals
- **Synapses** = connections between neurons transmitting signals
- **Dendrites** = thin structures forming a complex tree receiving signals
- **Axon** = single output sending signals
- Neurons are **electrically excitable**, maintaining voltage gradients
- If the voltage changes by a large enough amount, an **electrochemical pulse** is generated and travels rapidly along the axon and activates synaptic connections with other cells when it arrives.
- Neurons form complex network structures, the human brains has 10^{11} neurons.

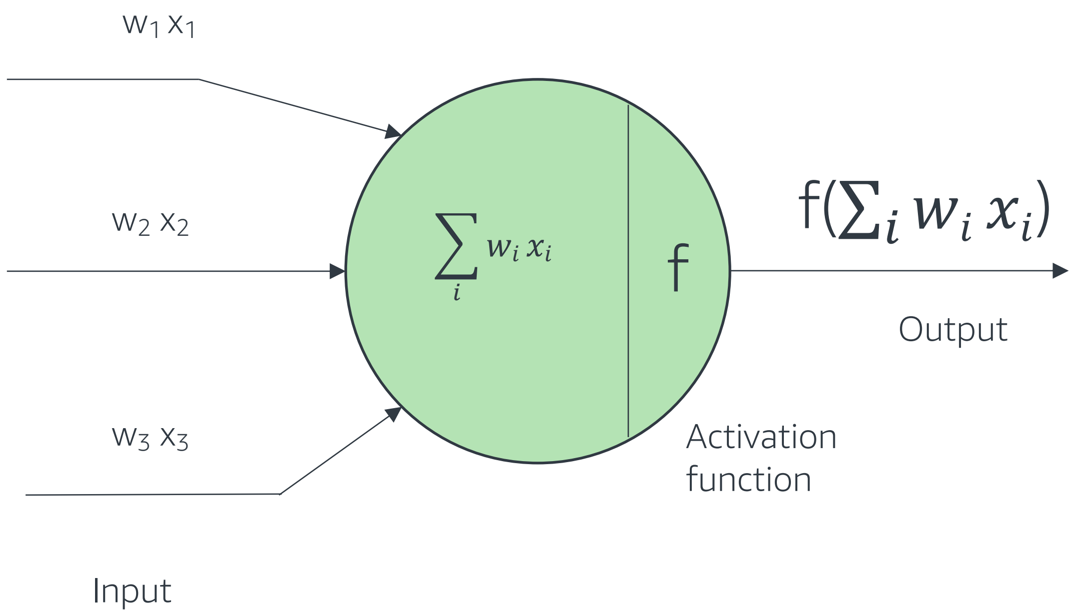
Artificial Neural Networks



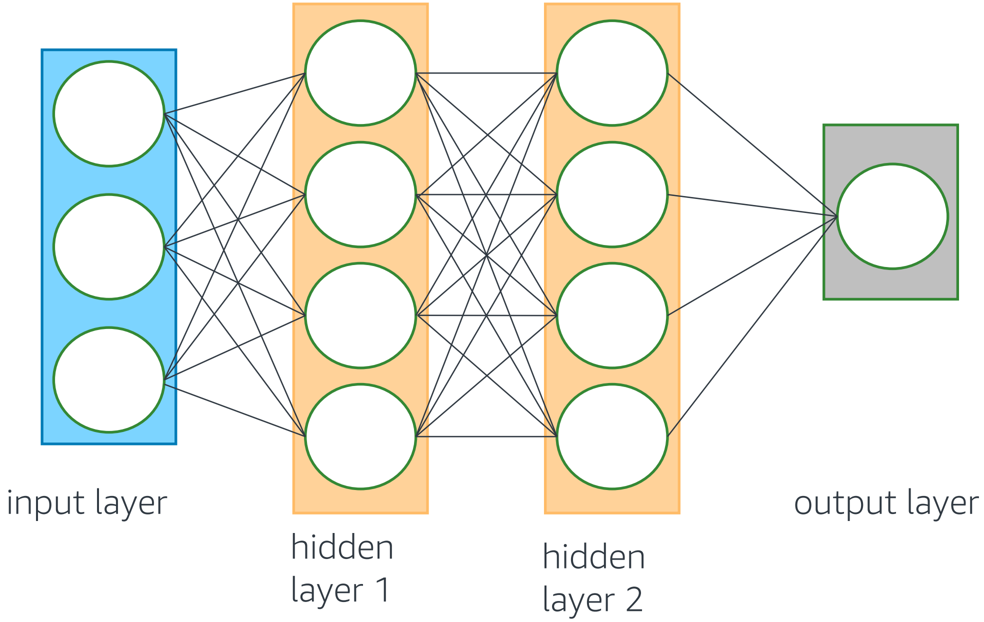
- Has many processing units (neurons)
- Received numbers from many other neurons (synapses)
- Combines numbers into weighted sum (dendrites)
- Has single outgoing number (axon)
- Applies function to decide how strong of a signal to send (pulse)

Artificial Neural Networks

Perceptron: A single neuron within a connected set of neurons.

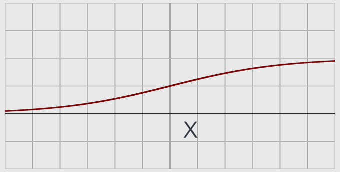
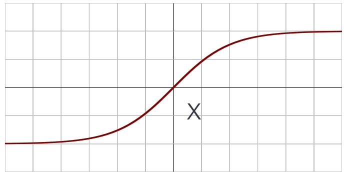
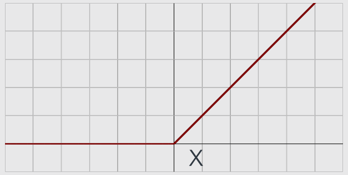


Multi Layer Perceptron



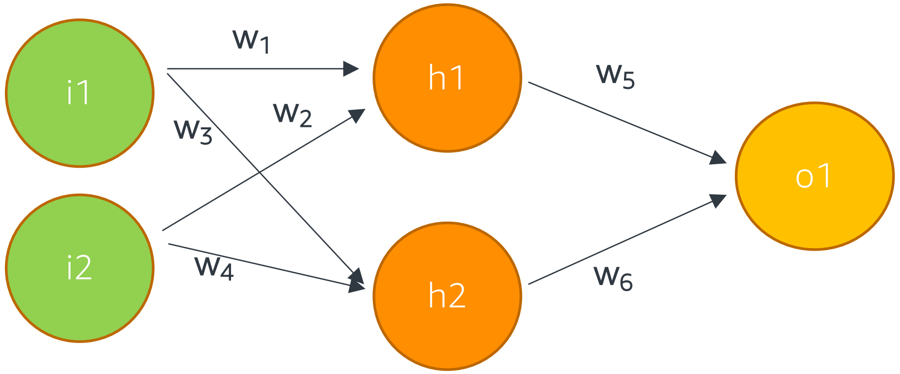
- A neural network consisting of input, hidden and output layers.
- Each layer is connected to the next layer.
- Except for the input units, each neuron has an activation function.

Activation Functions

Name	Plot	Function	Description
Logistic (sigmoid)		$f(x) = \frac{1}{1+e^{-x}}$	The most common activation function. Squashes input to [0,1]
Hyperbolic tangent (tanh)		$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	Squashes input to [-1, 1]
Rectified Linear Unit (ReLu)		$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \; \geq 0 \end{cases}$	Popular act. Function. Anything less than 0, results in zero activation

Derivatives of these functions are also important when we use gradient descent.

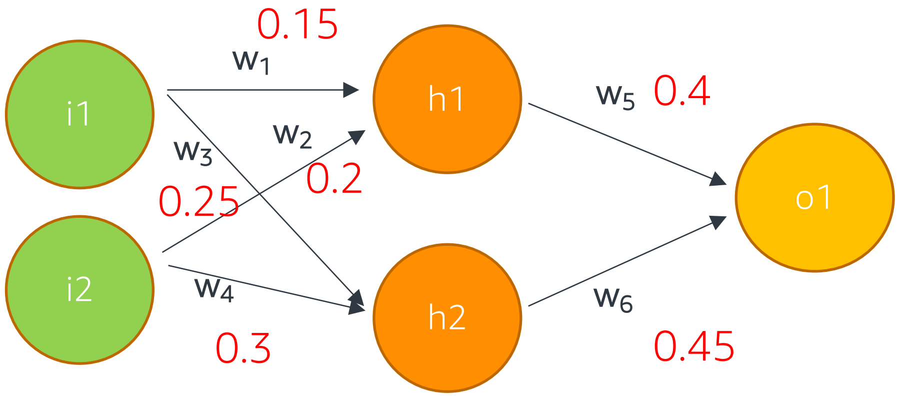
Forward Pass



Assume we have a network with:

- 2 inputs: i_1=0.05 and i_2=0.1
- 1 output (Binary classification: 0 or 1)
- 1 Hidden layer
- All neurons have sigmoid activation function

Forward Pass



$$net_{h1} = w_1 * \ i_1 + w_2 * \ i_2$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 = 0.0275$$

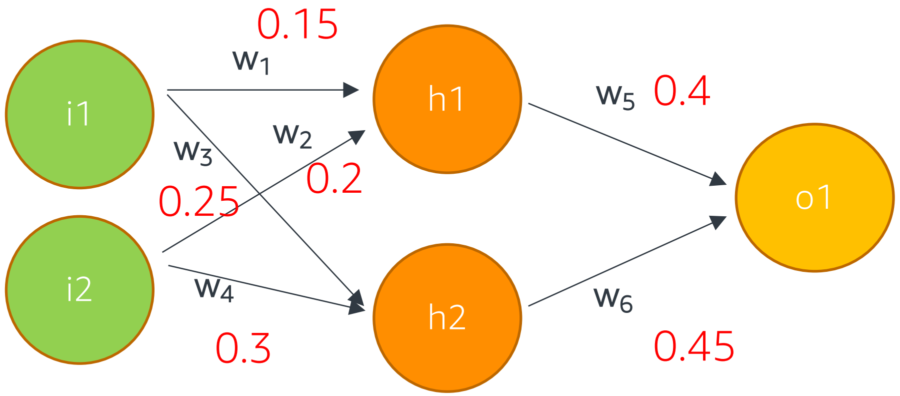
$$out_{h1} = \boxed{\frac{1}{1+e^{-net_{h1}}}} = \frac{1}{1+e^{-0.0275}} = 0.507$$

$$out_{h2} = 0.511$$

Sigmoid function

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

Forward Pass



$$net_{o1} = w_5 * \ out_{h1} + w_6 * \ out_{h2}$$

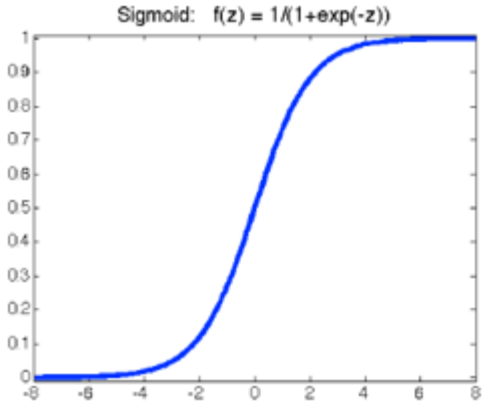
$$net_{o1} = 0.4 * 0.507 + 0.45 * 0.511$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-0.432}} = 0.61$$

For binary classification, we would classify this as class 1 (0.61>0.5)

Output Function

- Binary classification: Sigmoid
  - Outputs P(target class|x) in [0,1] $\sigma(z) = \frac{1}{1+\exp(-z)}$
  - Last layer is Logistic Regression of output of previous layers



- Multi-class classification: Softmax
  - Still want P for each class output in [0,1]
  - Want sum of output to be 1 (probability distribution)
  - Training drives value for target class up, others down.

$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

- Regression: Output activation can be linear or relu

Cost Functions

Binary classification: Cross entropy for logistic

$$C = -\frac{1}{n}\sum_{examples} y \ln p + (1-y)\ln(1-p)$$

Multi-class classification: Cross entropy for Softmax

$$C = -\frac{1}{n}\sum_{examples}\sum_{classes} y_j \ln p_j$$

Regression: Mean Squared Error: $C = \frac{1}{n}\sum_{examples}(y-p)^2$

Notation for Classification

- n training examples
- j classes
- p = prediction (probability)
- y = true class (1 for yes, 0 for no)

Notation for Regression

- n training examples
- p = prediction (numeric)
- y = true value

Training Neural Networks

- During the training of neural networks, we can update the weights in the network by applying the gradient descent method.
- Cost function is selected accr. to problem: Binary, multi-class classification or regression.
- Weight update formula:

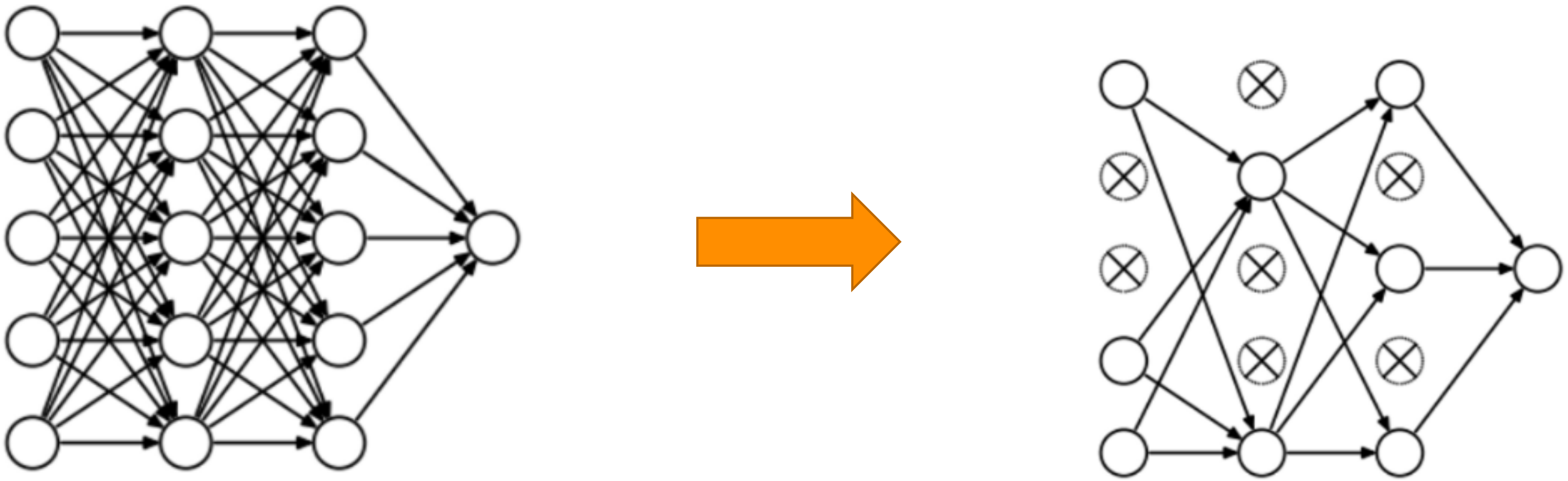
$$w_n : w_n - learning\_rate * \frac{\partial E}{\partial w_n}$$

E: Error

Gradient of w_n

Dropout

- Regularization technique to prevent overfitting.
- Randomly removes some nodes and their connections during the training.



Topics for today

- Optimization
- Regression
- Boosting
- Neural Networks
- MXNet
- Final Project

MXNet

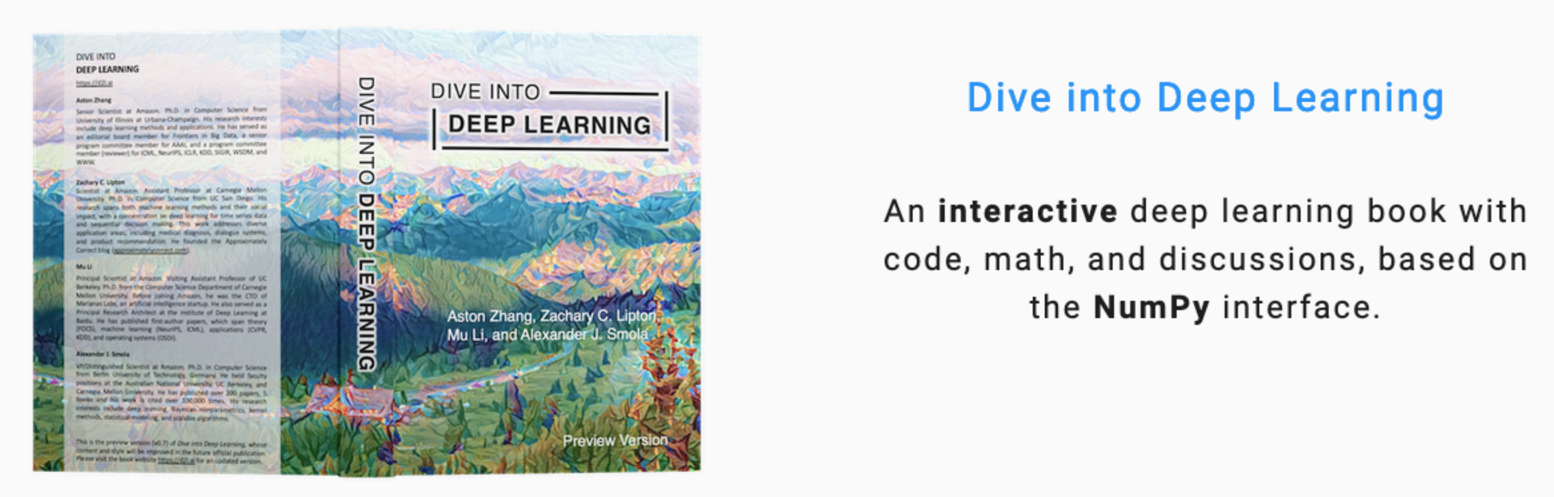
- Open source Deep Learning Library to train and deploy neural networks.
- With the **Gluon** interface, we can define and train neural networks easily.
- We will go over the tutorials below:
  - Mxnet-Ndarrays-Autograd intro: https://eider.corp.amazon.com/sazaracs/notebook/NBM9GVVWI98P
  - Building Neural Network with MXNet: https://eider.corp.amazon.com/sazaracs/notebook/NBLQY2I99MEG

Hands-on exercise

- In this exercise we will work with an internal Amazon Dataset.
- We will do the following tasks:
  - Exploratory Data Analysis
  - Splitting dataset into training and test sets.
  - Fit a Neural Network
  - Check the performance metrics on test set.
- Open the notebook below:

https://eider.corp.amazon.com/sazaracs/notebook/NB8A43NS520E

Dive into Deep Learning



E-book on deep learning by Amazon Scientists, available here: https://d2l.ai

Related chapters:

Chapters 3: Linear Neural Networks: https://d2l.ai/chapter_linear-networks/index.html

Chapters 4: Multilayer Perceptrons: https://d2l.ai/chapter_multilayer-perceptrons/index.html

AutoML

AutoML helps automating some of the tasks related to ML model development and training such as:

- Preprocessing and cleaning data
- Feature selection
- ML model selection
- Hyper-parameter optimization

Auto GLUON AutoML

- Open source AutoML Toolkit (AMLT) created by Amazon AI
- With AutoGluon, state of the art ML results can be achieved in a few lines of Python code.

```
>>> from autogluon import TabularPrediction as task
>>> predictor = task.fit(train_data=task.Dataset(file_path=TRAIN_DATA.csv), label=COLUMN_NAME)
>>> predictions = predictor.predict(task.Dataset(file_path=TEST_DATA.csv))
```

AutoGluon AutoML

- Easy to Use – Built-in Applications

Tabular Prediction

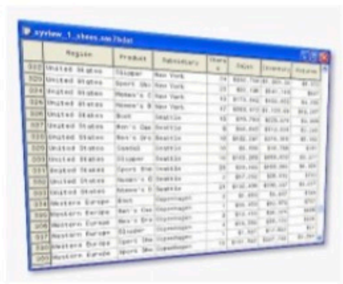
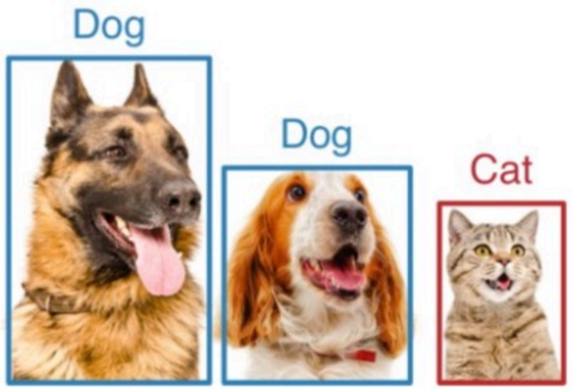


Image Classification

Object Detection



Text Classification

AutoGLUON AutoML

- AutoGluon is able to produce top solutions to many real-world business problems. AutoGluon was utilized to create:
  - The first place solution to the MLA NLP I Hazmat Challenge
    - First place against over 1,000 competitors and 10,000 manual submissions over the course of 12 months.
  - The first place solution to the MLE Ops Tech IT Challenge
  - The first place solution to the SCOT Grand Challenge 2018
    - Solution was able to determine if two products are identical better than a human evaluator.
    - Used as the production solution to the product matching problem within Amazon.
  - The first place solution to the SCOT Grand Challenge 2019
    - Solution approximately halved the predictive error of Amazon.com shipment costs worldwide.
  - The existing production solution to Amazon Grocery (F3) sales forecasting
- pip install to get started: https://github.com/awslabs/autogluon

Topics for today

- Optimization
- Regression
- Boosting
- Neural Networks
- MXNet
- Final Project

Final Project

Product substitute: Given products (A, B), predict whether B is a substitute for A

- We say that B is a "substitute" for A if a customer would buy B in place of A -- say, if A were out of stock.
- **The goal** of this project is to predict a substitute relationship between pairs of products.
- **Link:** https://leaderboard.corp.amazon.com/tasks/478

Final Project Walkthrough – Day 3

- See the provided project walkthrough below to get started with this project.
- You can use Boosted Trees and Neural Networks this time.
- Complete the following notebook and submit your results
  https://eider.corp.amazon.com/sazaracs/notebook/NBLR85NVZ4V4

Notebooks for day 3

- MXNet – NDarrays intro: https://eider.corp.amazon.com/sazaracs/notebook/NBM9GVVWI98P
- Neural Networks intro: https://eider.corp.amazon.com/sazaracs/notebook/NBLQY2I99MEG
- Neural Networks full example: https://eider.corp.amazon.com/sazaracs/notebook/NB8A43NS520E
- Final project walkthrough for day 3: https://eider.corp.amazon.com/sazaracs/notebook/NBLR85NVZ4V4

MACHINE LEARNING UNIVERSITY