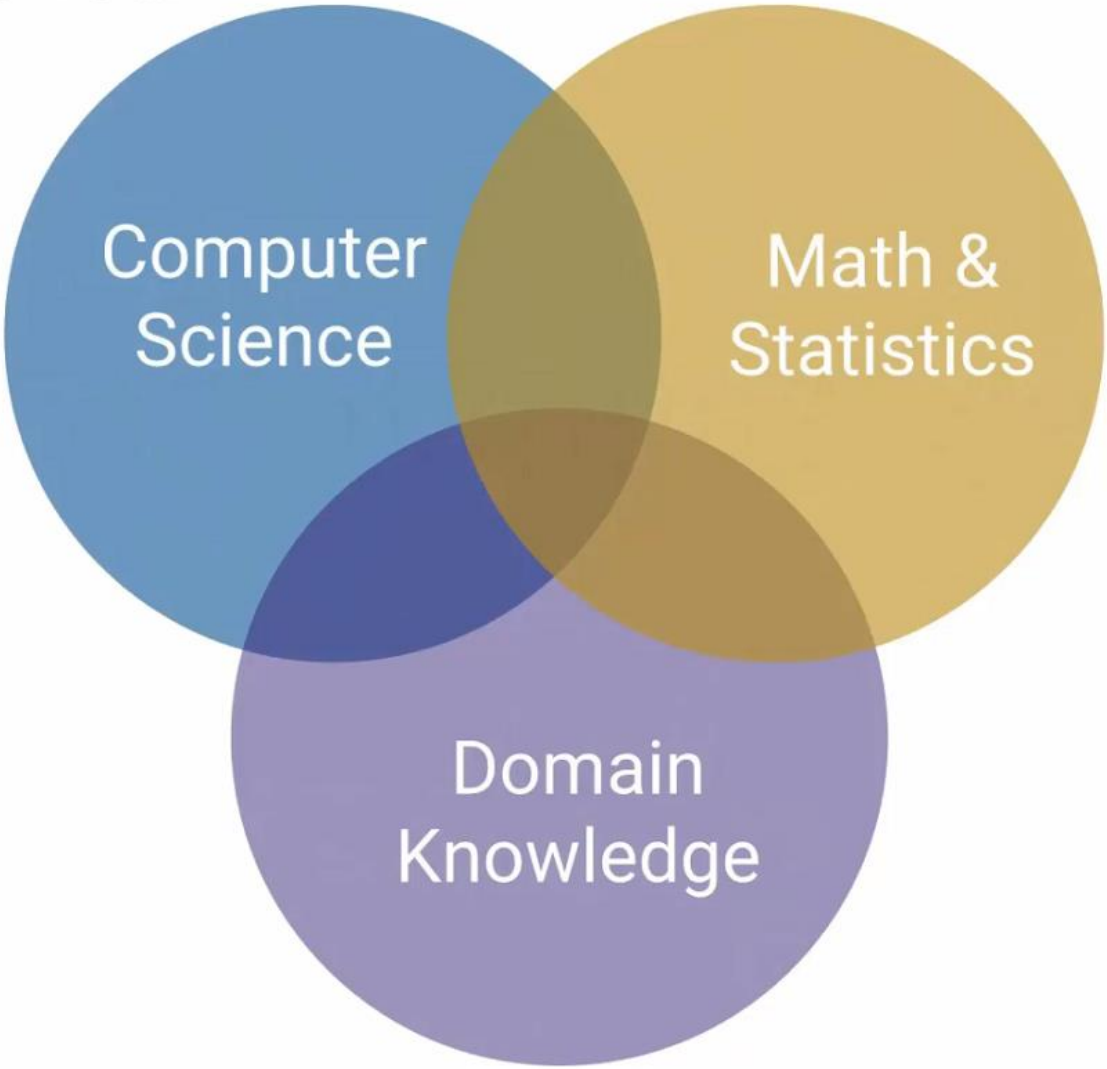
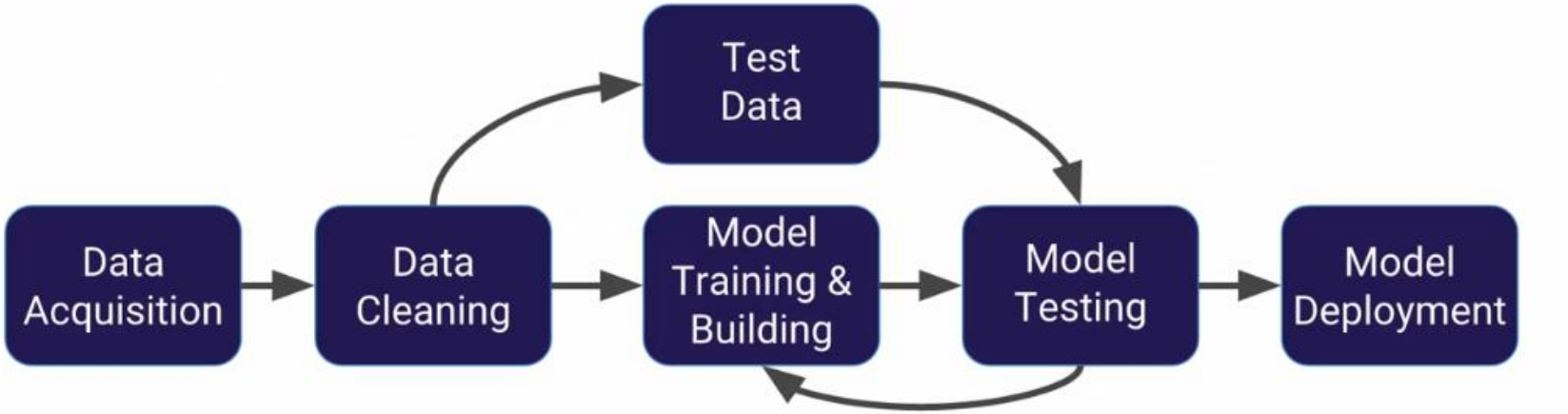
Machine Learning Algorithms

Data Science



- Machine learning is a method of data analysis that automates analytical model building.
- Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look.

- Fraud detection.
- Web search results.
- Real-time ads on web pages
- Credit scoring and next-best offers.
- Prediction of equipment failures.
- New pricing models.
- Network intrusion detection.
- Recommendation Engines
- Customer Segmentation
- Text Sentiment Analysis
- Predicting Customer Churn
- Pattern and image recognition.
- Email spam filtering.
- Financial Modeling


Test Data
Data Acquisition
Data Cleaning
Model Training & Building
Model Testing
Model Deployment

- There a 3 main types of Machine Learning algorithms
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning
- Let's briefly describe each type before diving in with more detail.

- Supervised Learning
  - You have labeled data and are trying to predict a label based off of known features
- Unsupervised Learning
  - You have unlabeled data and are trying to group together similar data points based off of features
- Reinforcement Learning
  - Algorithm learns to perform an action from experience

- **Supervised learning** algorithms are trained using **labeled** examples, such as an input where the desired output is known.
- For example, a piece of equipment could have data points labeled either “F” (failed) or “R” (runs).

- The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors.
- It then modifies the model accordingly.

- Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data.
- Supervised learning is commonly used in applications where historical data predicts likely future events.

- For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.
- Or it can attempt to predict the price of a house based on different features for houses for which we have historical price data.

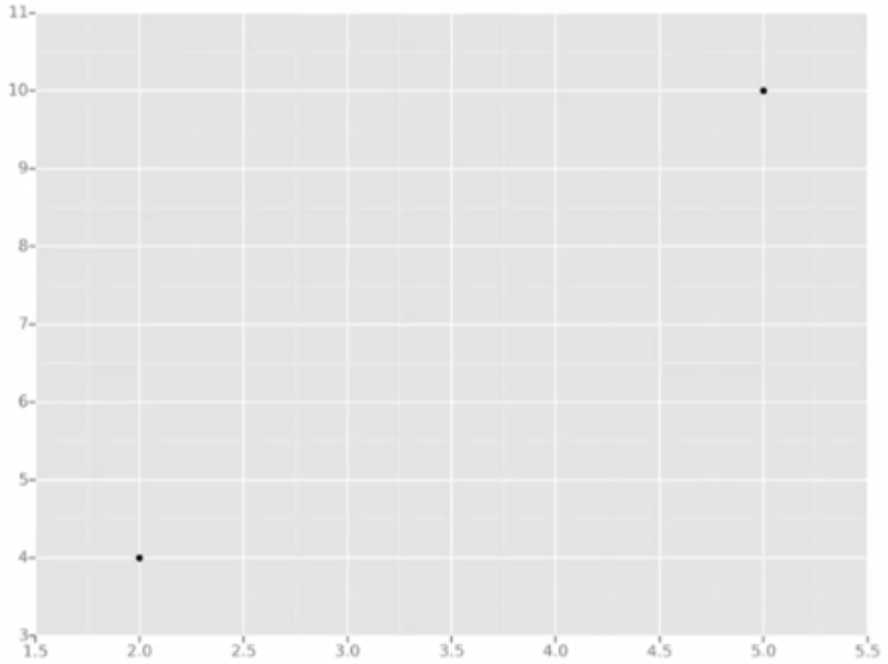
- **Unsupervised learning** is used against data that has no historical labels.
- The system is not told the "right answer." The algorithm must figure out what is being shown.
- The goal is to explore the data and find some structure within.

- Or it can find the main attributes that separate customer segments from each other.
- Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition.

- **Reinforcement learning** is often used for robotics, gaming and navigation.
- With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards.

- This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do).

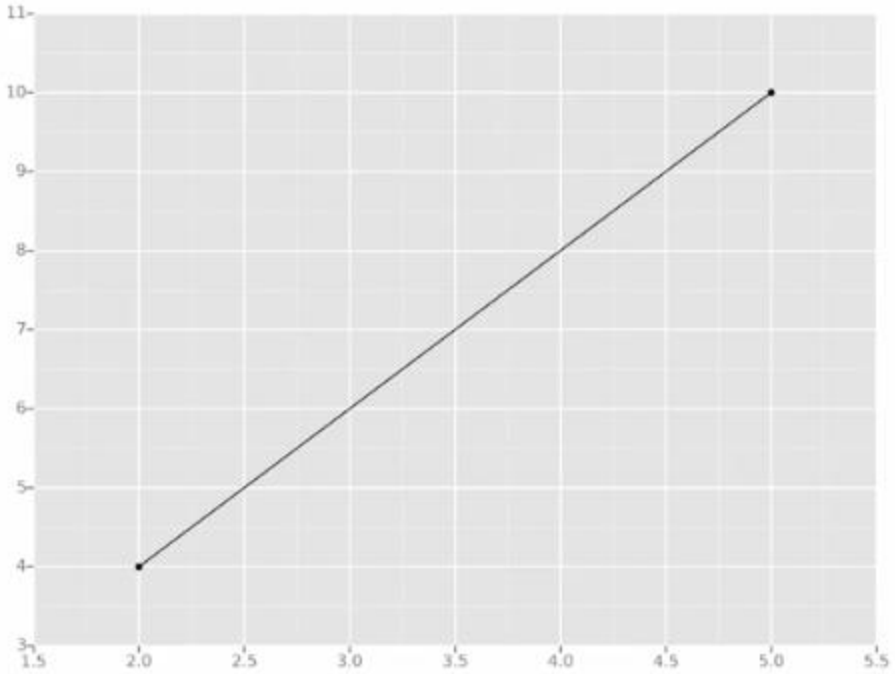
Let's take the simplest possible example: calculating a regression with only 2 data points.



Least squares is a statistical **method** used to determine a line of best fit by minimizing the sum of **squares** created by a mathematical function. A "**square**" is determined by squaring the distance between a data point and the regression line.

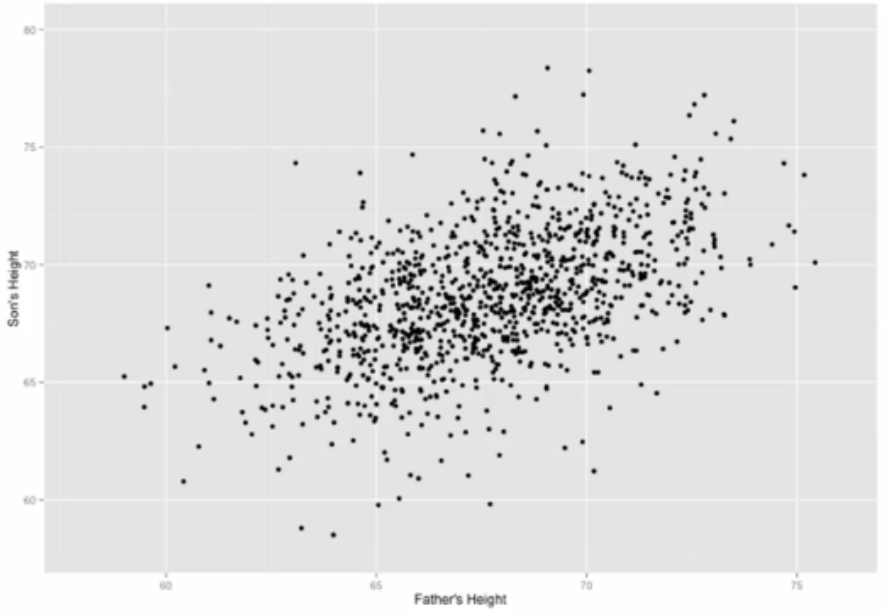
All we're trying to do when we calculate our regression line is draw a line that's as close to every dot as possible.

For classic linear regression, or "Least Squares Method", you only measure the closeness in the "up and down" direction



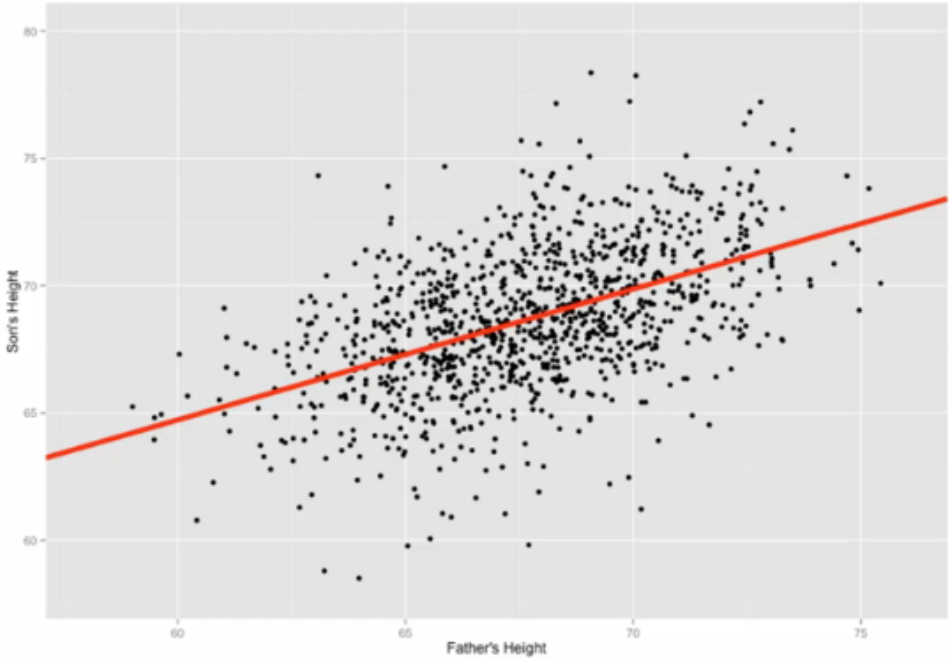
Now wouldn't it be great if we could apply this same concept to a graph with more than just two data points?

By doing this, we could take multiple men and their son's heights and do things like tell a man how tall we expect his son to be...before he even has a son!



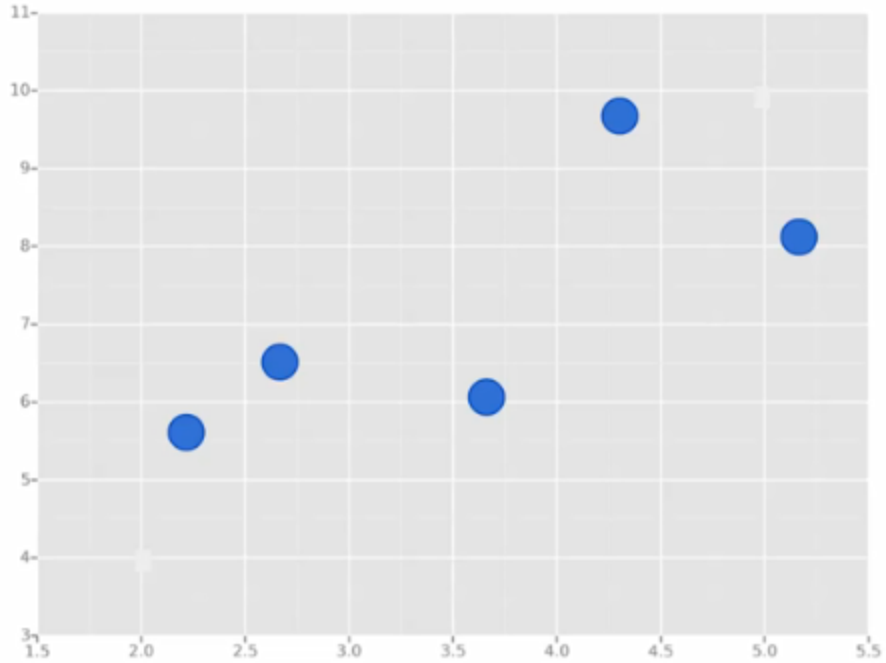
Our goal with linear regression is to **minimize the vertical distance** between all the data points and our line.

So in determining the **best line**, we are attempting to minimize the distance between **all** the points and their distance to our line.



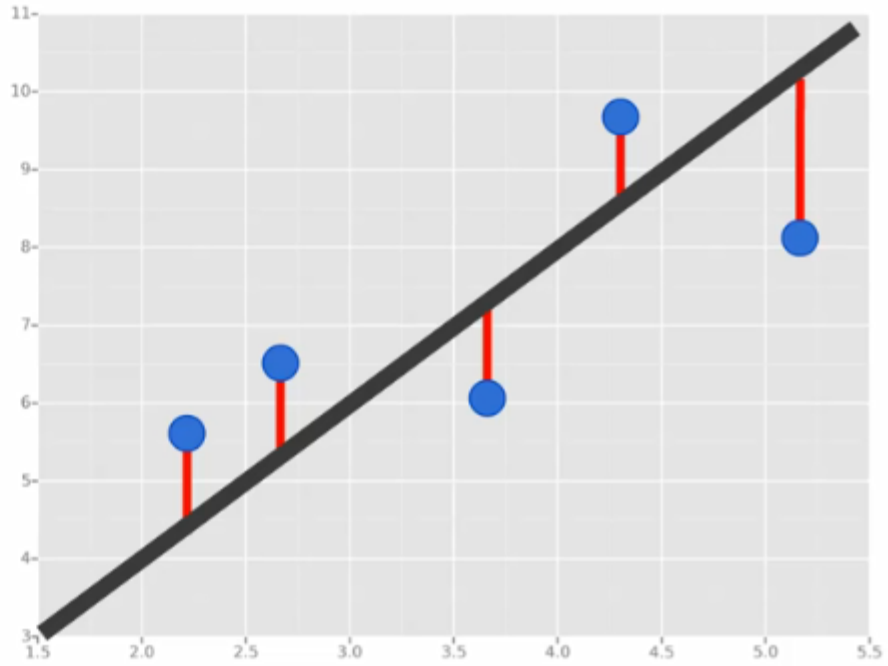
For example, one of the most popular methods is the least squares method.

Here we have blue data points along an x and y axis.



We'll use the Least Squares Method, which is fitted by minimizing the ***sum of squares of the residuals.***

The residuals for an observation is the difference between the observation (the y-value) and the fitted line.



Training: Data used to train the model

Notations:
x = "input" variable feature
y = "output" variable or "target" variable
m = number of training example
n = number of features
(x^i, y^i) = i^{th} training example

Model:
$f_{w,b}(x) = wx + b$

w, b: parameters, coefficients, weights

cost function:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

	x	y
	size in feet	price in $1000's
1	2104	400
2	1416	232
3	1534	315
4	852	178
…		
47	3210	870

$(x^{(1)}, y^{(1)}) = (2104, 400)$
$x^{(2)}$ (2) is an index and!= x^2 not exponent
$x^{(2)} = 1416$
$y^{(3)} = 315$

Linear Regression Model

$f_{w,b}(x) = wx + b$

Cost function

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

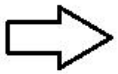
Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha\frac{\partial}{\partial w}J(w,b)$$

$$b = b - \alpha\frac{\partial}{\partial w}J(w,b)$$

}



repeat until convergence {

$$w = w - \alpha\frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

$$b = b - \alpha\frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})$$

}

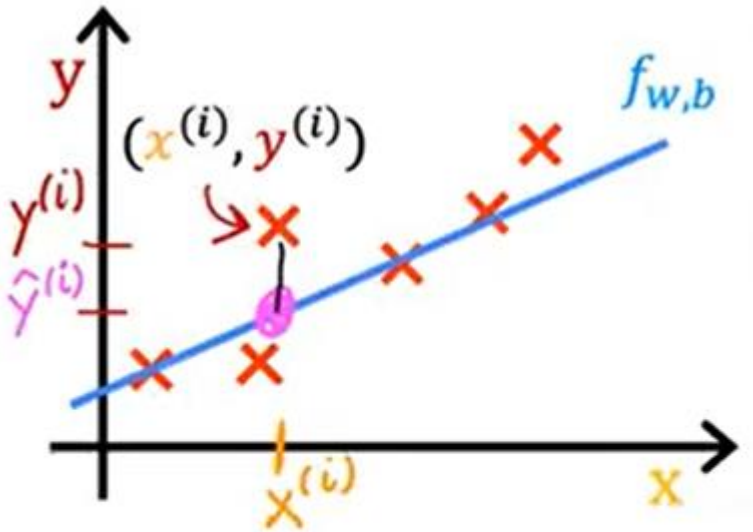
Cost function: Squared error cost function

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

error

m = number of training examples

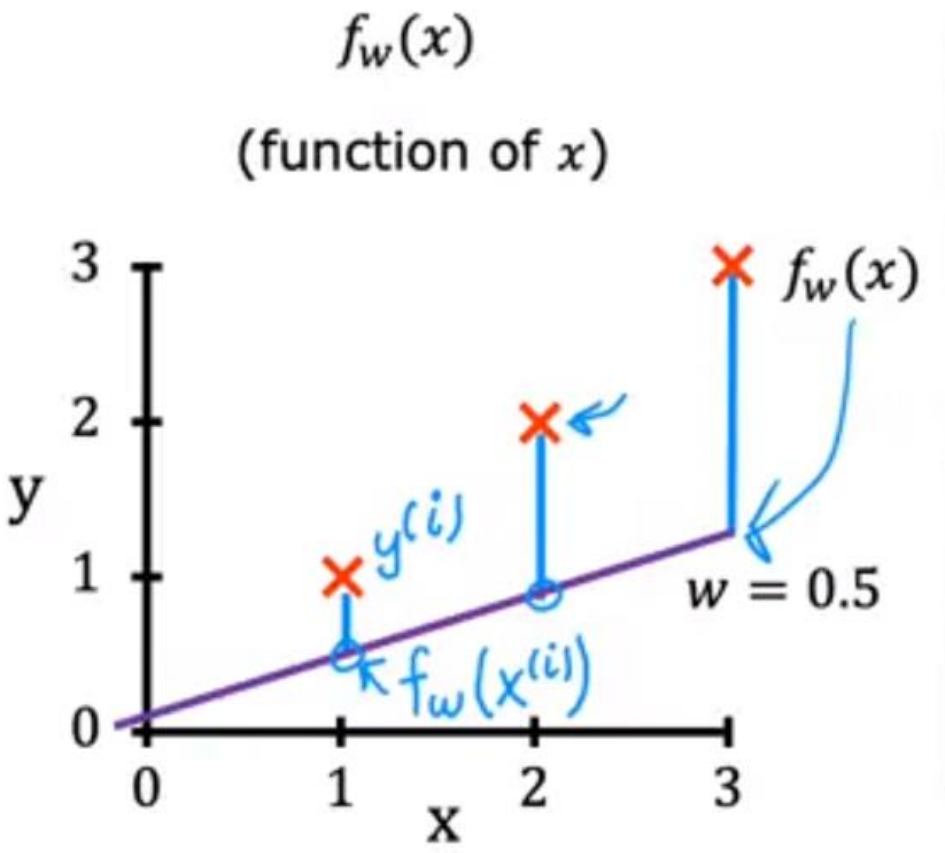
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$$

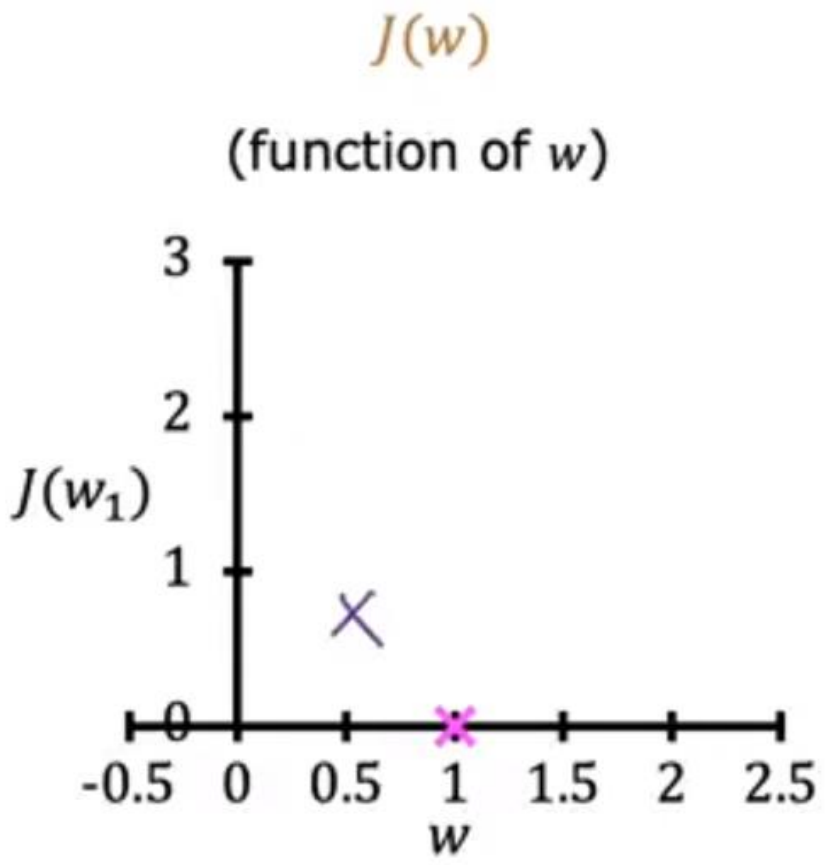


Find w, b:
$\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$.

$$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$$

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b$$





$J(0.5) =$ **1/2m[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2]= 1/2*3 [3.5] = 3.5/6 ~ 0.58**

Multiple features (variables)

	Size in feet²	Number of bedrooms	Number of floors	Age of home in years	Price ($) in $1000's
	x_1	x_2	x_3	x_4	
	2104	5	1	45	460
$i=2$	1416	3	2	40	232
	1534	3	2	30	315
	852	2	1	36	178
	...	...	...	...	...

$j = 1 \ldots 4$

$n = 4$

x_j = j^{th} feature

n = number of features

$\vec{x}^{(i)}$ = feature of i^{th} training example

$x_j^{(i)}$ = value of feature j in the i^{th} training example

$\vec{x}^{(2)} = [1416\ 3\ 2\ 40]$

$\vec{x}_3^{(2)} = 2$

Without vectorization

```
for j in range(0,16):
    f = f + w[j] * x[j]
```

t_0

```
f + w[0] * x[0]
```

t_1

```
f + w[1] * x[1]
```

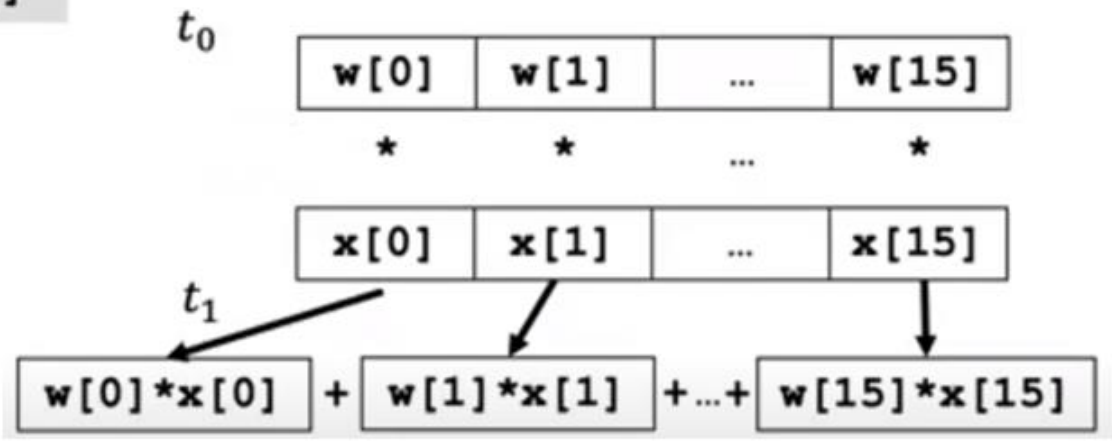
...

t_{15}

```
f + w[15] * x[15]
```

Vectorization

```
np.dot(w,x)
```



Feature and parameter values

$\widehat{price} = w_1 x_1 + w_2 x_2 + b$

x_1: size (feet^2)
range: 300 – 2,000

x_2: # bedrooms
range: 0 – 5

House: $x_1 = 2000$, $x_2 = 5$, $price = \$500k$

size of the parameters w_1, w_2?

$w_1 = 50$, $w_2 = 0.1$, $b = 50$

$\widehat{price} = 50 * 2000 + 0.1 * 5 + 50$

$\widehat{price} = \$100{,}050.5k$

$w_1 = 0.1$, $w_2 = 50$, $b = 50$

$\widehat{price} = 0.1 * 2000k + 50 * 5 + 50$

$\widehat{price} = \$500k$

Gradient Descent With Multiple Variables

Gradient descent for multiple variables:

The equation for the cost function with multiple variables $J(\mathbf{w}, b)$ is:

$$J(\mathbf{w}, b) = \frac{1}{2m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

where:

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{x}^{(i)} + b$$

In contrast to previous labs, $\mathbf{w}$ and $\mathbf{x}^{(i)}$ are vectors rather than scalars supporting multiple features.

repeat until convergence: {

$$w_j = w_j - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial w_j} \qquad \text{for j = 0..n-1}$$

$$b = b - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial b}$$

}

where, n is the number of features, parameters w_j, b, are updated simultaneously and where

$$\frac{\partial J(\mathbf{w}, b)}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

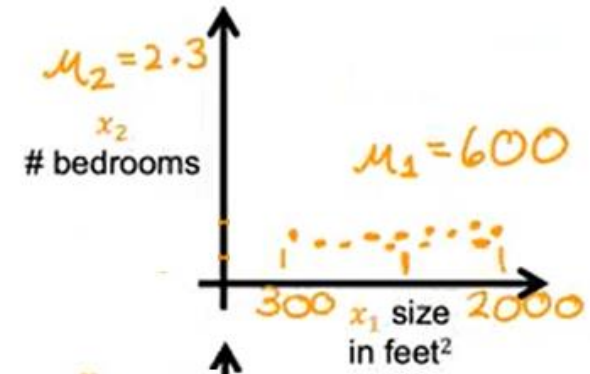
$$\frac{\partial J(\mathbf{w}, b)}{\partial b} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})$$

where:

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{x}^{(i)} + b$$

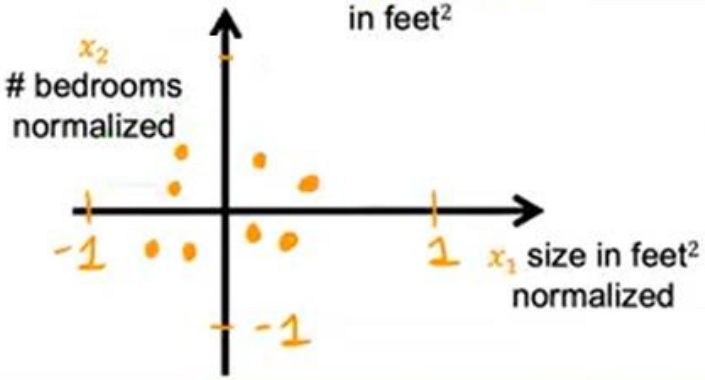
- m is the number of training examples in the data set
- $f_{\mathbf{w},b}(\mathbf{x}^{(i)})$ is the model's prediction, while $y^{(i)}$ is the target value

Feature scaling
Mean normalization



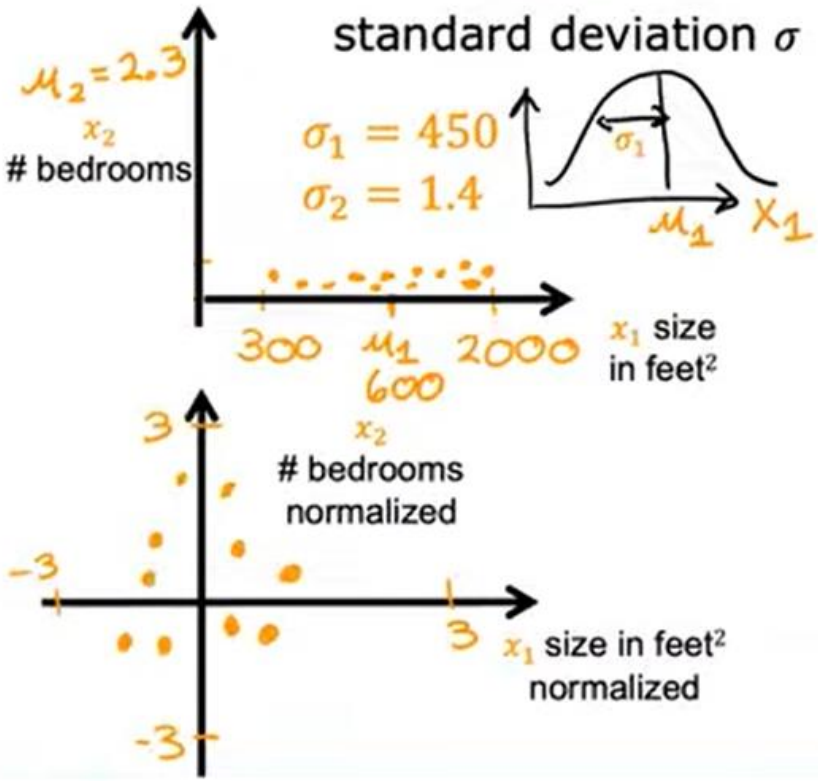
$300 \le x_1 \le 2000$ $\quad 0 \le x_2 \le 5$

$$x_1 = \frac{x_1 - \mu_1}{2000-300} \qquad x_2 = \frac{x_2 - \mu_2}{5-0}$$



$-0.18 \le x_1 \le 0.82$ $\quad -0.46 \le x_2 \le 0.54$

Z-score normalization



$300 \le x_1 \le 2000$ $\quad 0 \le x_2 \le 5$

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1} \qquad x_2 = \frac{x_2 - \mu_2}{\sigma_2}$$

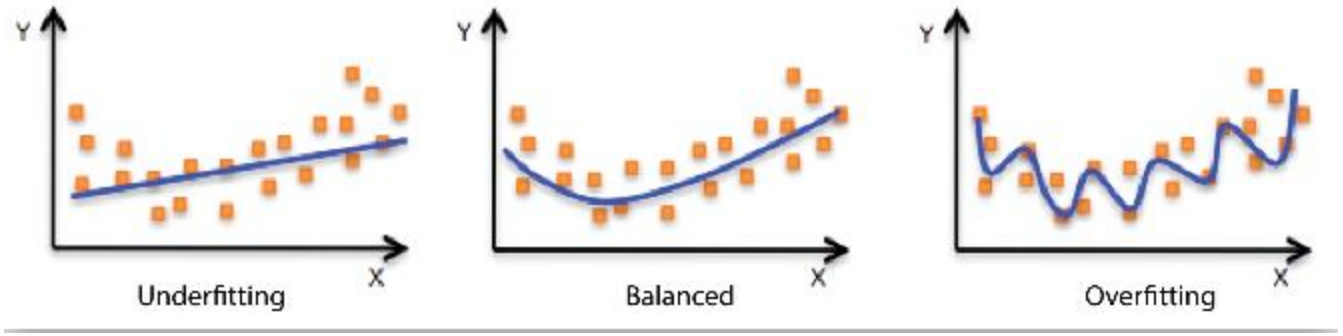
$-0.67 \le x_1 \le 3.1$ $\quad -1.6 \le x_2 \le 1.9$

We can determine whether a predictive model is under fitting or overfitting the training data by looking at the prediction error on the training data and the evaluation data.

Your model is *under fitting* the training data when the model performs poorly on the training data. This is because the model is unable to capture the relationship between the input examples (often called X) and the target values (often called Y).

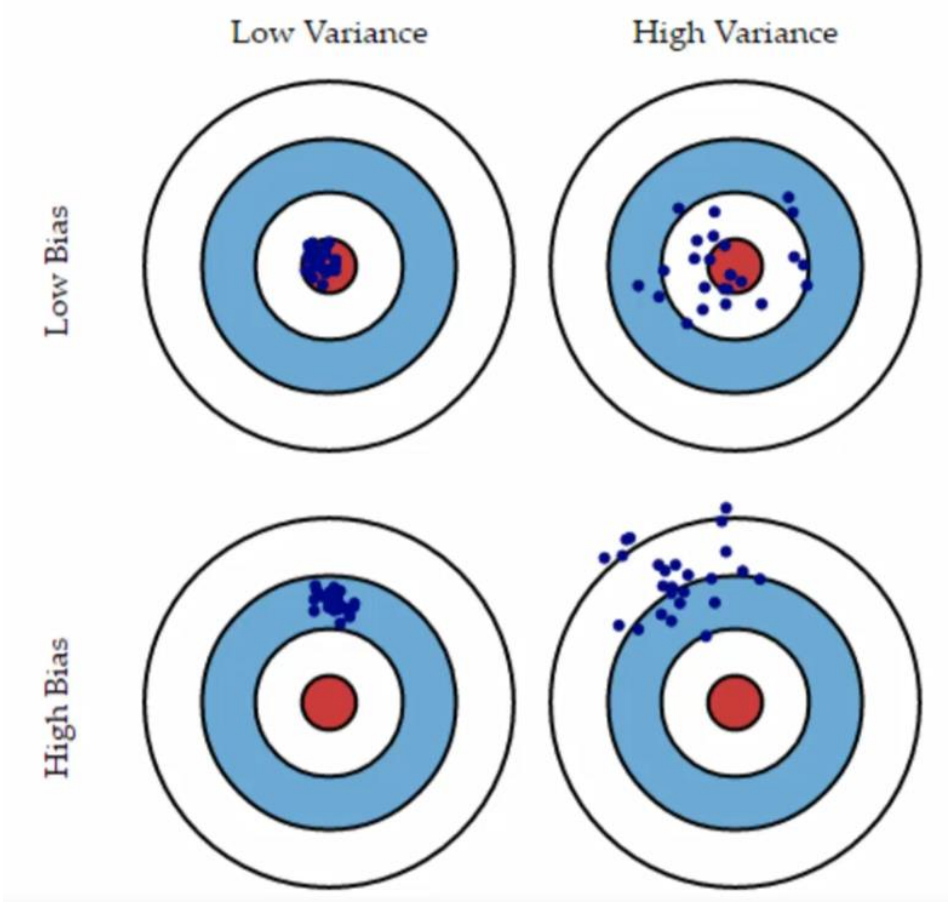
Your model is *overfitting* your training data when you see that the model performs well on the training data but does not perform well on the evaluation data. This is because the model is memorizing the data it has seen and is unable to generalize to unseen examples.

Ref: Amazon Machine Learning

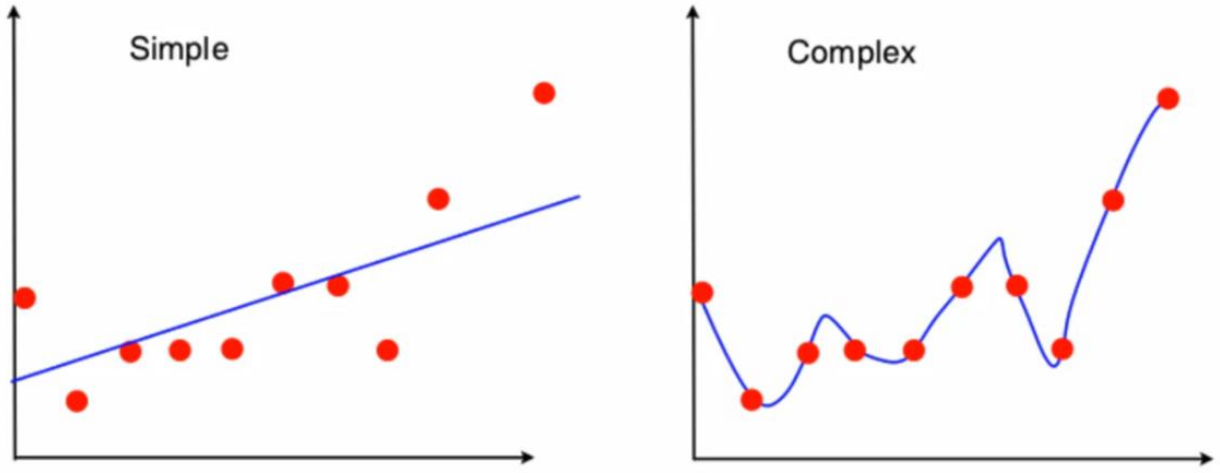


Ref: Amazon Machine Learning

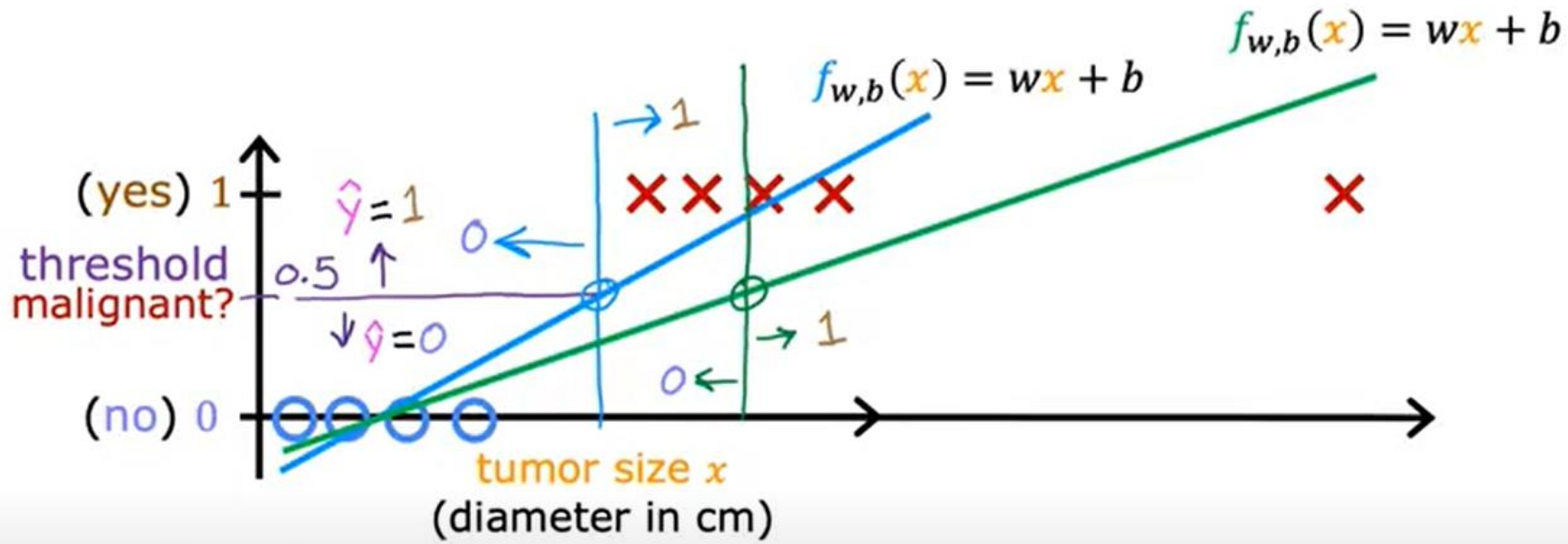
- The bias-variance trade-off is the point where we are adding just noise by adding model complexity (flexibility).
- The training error goes down as it has to, but the test error is starting to go up.
- The model after the bias trade-off begins to overfit.


Low Variance
High Variance
Low Bias
High Bias

- A common temptation for beginners is to continually add complexity to a model until it fits the training set very well.

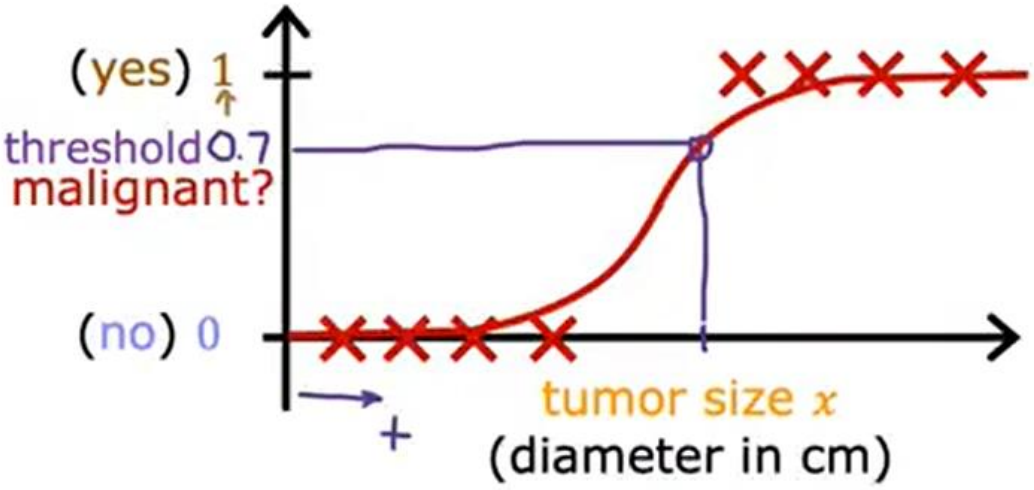


- We want to learn about Logistic Regression as a method for **Classification.**
- Some examples of classification problems:
  - Spam versus "Ham" emails
  - Loan Default (yes/no)
  - Disease Diagnosis
- Above were all examples of Binary Classification

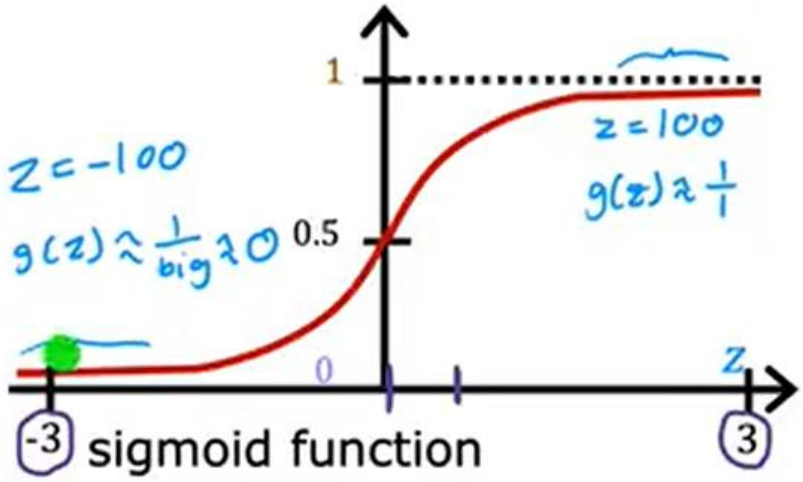


if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$



Want outputs between 0 and 1



sigmoid function

logistic function

outputs between 0 and 1

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

Interpretation of logistic regression output

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

"probability" that class is 1

$P(1)$

$$f_{\vec{w},b}(\vec{x}) = P(y = 1|\vec{x}; \vec{w}, b)$$

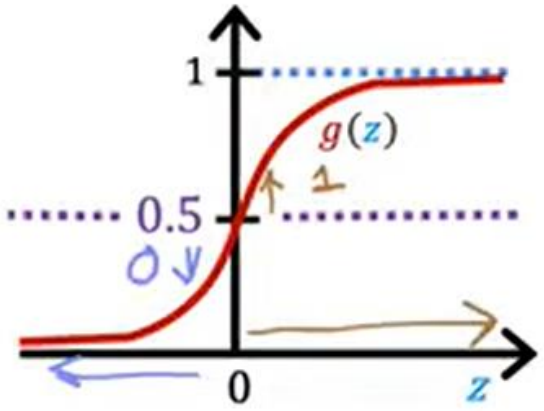
Probability that y is 1,
given input $\vec{x}$, parameters $\vec{w}, b$

Example:
x is "tumor size"
y is 0 (not malignant)
or 1 (malignant)

$$f_{\vec{w},b}(\vec{x}) = 0.7$$

70% chance that y is 1

$$P(y = 0) + P(y = 1) = 1$$



$f_{\vec{w},b}(\vec{x})$

$z = \vec{w} \cdot \vec{x} + b$

z

$g(z) = \frac{1}{1+e^{-z}}$

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w} \cdot \vec{x} + b}_{z}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

$$= P(y = 1 | x; \vec{w}, b) \quad 0.7 \quad 0.3$$

0 or 1?

Is $f_{\vec{w},b}(\vec{x}) \geq \overbrace{0.5}^{\text{threshold}}$?

Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

When is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

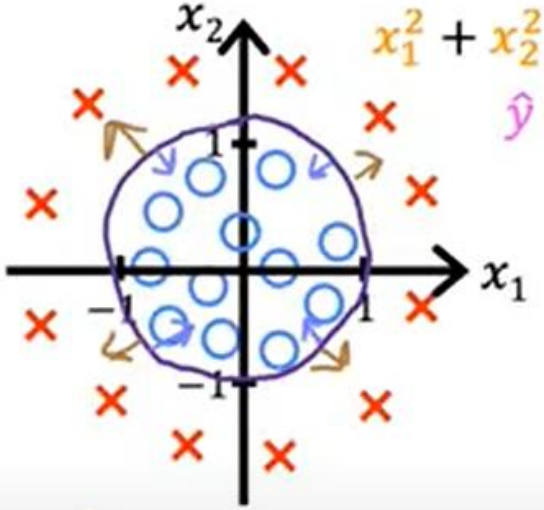
$g(z) \geq 0.5$

$z \geq 0$

$\vec{w} \cdot \vec{x} + b \geq 0$ $\vec{w} \cdot \vec{x} + b < 0$

$\hat{y} = 1$ $\hat{y} = 0$

Non-linear decision boundaries



$x_1^2 + x_2^2 \geq 1$

$\hat{y} = 1$

$x_1^2 + x_2^2 < 1$

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\overbrace{w_1 x_1^2 + w_2 x_2^2 + b}^{z})$$

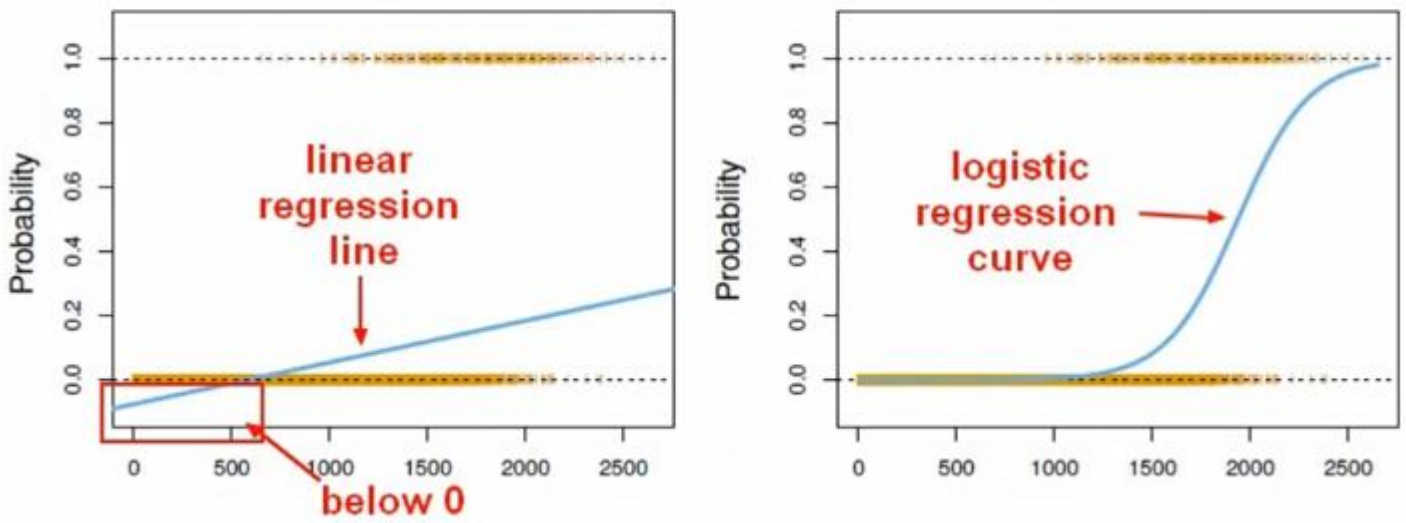
$w_1 = 1$, $w_2 = 1$, $b = -1$

decision boundary $z = x_1^2 + x_2^2 - 1 = 0$

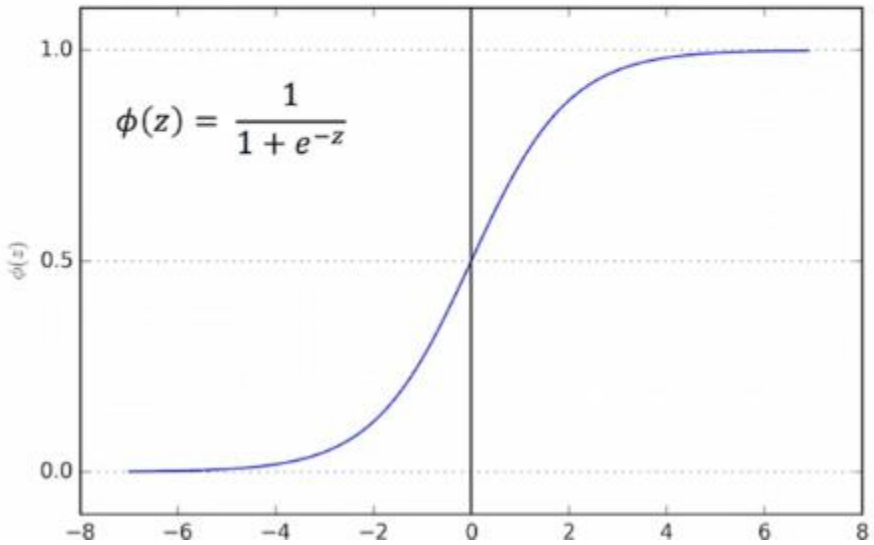
$x_1^2 + x_2^2 = 1$

- So far we've only seen regression problems where we try to predict a continuous value.
- Although the name may be confusing at first, logistic regression allows us to solve classification problems, where we are trying to predict discrete categories.
- The convention for binary classification is to have two classes 0 and 1.

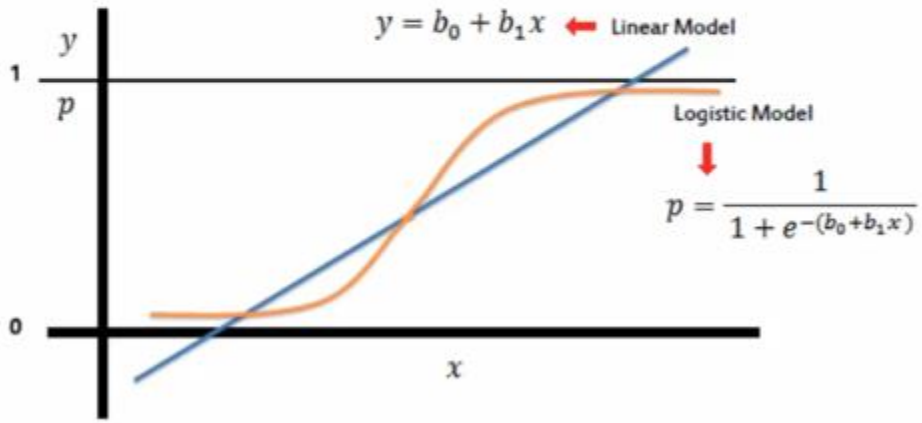
- Instead we can transform our linear regression to a logistic regression curve.



- The Sigmoid (aka Logistic) Function takes in any value and outputs it to be between 0 and 1.



- This means we can take our Linear Regression Solution and place it into the Sigmoid Function.



- After you train a logistic regression model on some training data, you will evaluate your model's performance on some test data.
- You can use a confusion matrix to evaluate classification models.

- We can use a confusion matrix to evaluate our model.
- For example, imagine testing for disease.

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Example: Test for presence of disease
NO = negative test = False = 0
YES = positive test = True = 1

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Basic Terminology:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Accuracy:

- Overall, how often is it **correct**?
- (TP + TN) / total = 150/165 = 0.91

	FALSE	TRUE
0	140	25
1	29	74

TP + TN = 74 + 140 = 214
Total = 169 + 99 = 268
Correctness: 214/268 = 0.79

K-Nearest Neighbor Learning

The k-nearest neighbor algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

Algorithms that do not make particular assumptions about the kind of mapping function are known as non-parametric algorithms. These algorithms do not accept a specific form of the mapping function between input and output data as true. They have the freedom to choose any functional form from the training data.

K-Nearest Neighbor Learning

- Instance-based Learning
  - Learning=storing all training instances
  - Classification=assigning target function to a new instance
- Features
  - All instances correspond to points in an n-dimensional Euclidean space
  - Classification is delayed till a new instance arrives
  - Classification done by comparing feature vectors of the different points
  - Target function may be discrete or real-valued
- An arbitrary instance is represented by $(a_1(x), a_2(x), a_3(x),.., a_n(x))$
  - $a_i(x)$ denotes features
- Euclidean distance between two instances
  - $d(x_i, x_j)$=sqrt (sum for r=1 to n $(a_r(x_i) - a_r(x_j))^2$)

K-Nearest Neighbor Classification

Compute distance between two points:

- Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Manhatten distance

$$d(p,q) = \sum_i |p_i - q_i|$$

- q norm distance

$$d(p,q) = (\sum_i |p_i - q_i|^q)^{1/q}$$

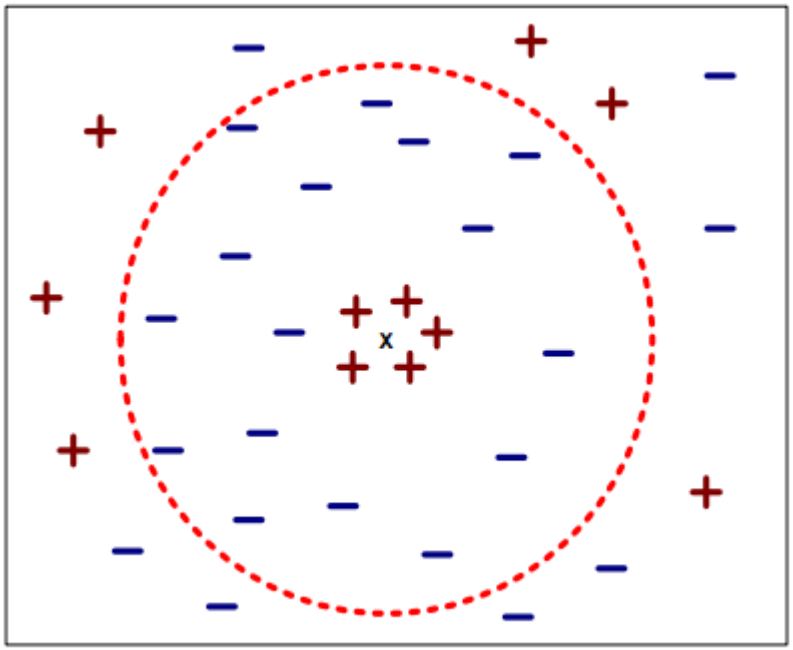
K-Nearest Neighbor Algorithm

Let k be the number of nearest neighbors and D be the set of training examples.

1. **for** each test example z = (**x**',y') **do**
2. Compute d(**x**',**x**), the distance between z and every example, (**x**,y) ϵ D
3. Select $D_z \subseteq D$, the set of k closest training examples to z.
4. $y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i,y_i) \in D_z} I(v = y_i)$
5. **end for**

K-Nearest Neighbor Classification

Choosing the value of k: – If k is too small, sensitive to noise points – If k is too large, neighborhood may include points from other classes



K-Means Clustering

K-Means clustering is an unsupervised learning algorithm. There is no labeled data for this clustering, unlike in supervised learning. K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

K-means is the most widely-used centroid-based clustering algorithm. Centroid-based algorithms are efficient but sensitive to initial conditions and outliers. This course focuses on k-means because it is an efficient, effective, and simple clustering algorithm

K-Means Clustering

- Clustering is the classification of objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often according to some defined distance measure.

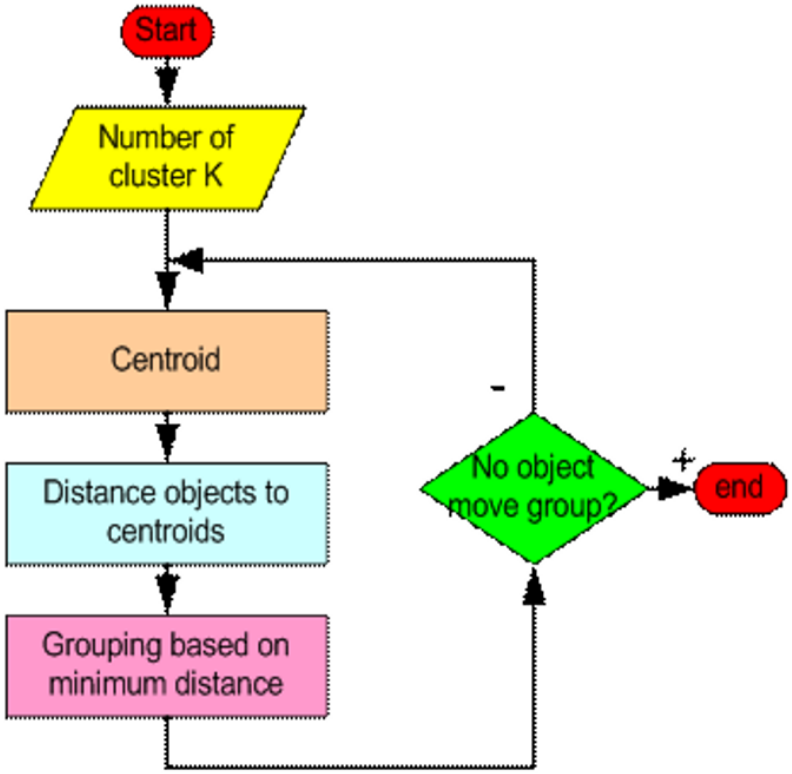
- Distance measure will determine how the similarity of two elements is calculated and it will influence the shape of the clusters. They include:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

  - Manhatten distance

$$d(p,q) = \sum_i |p_i - q_i|$$

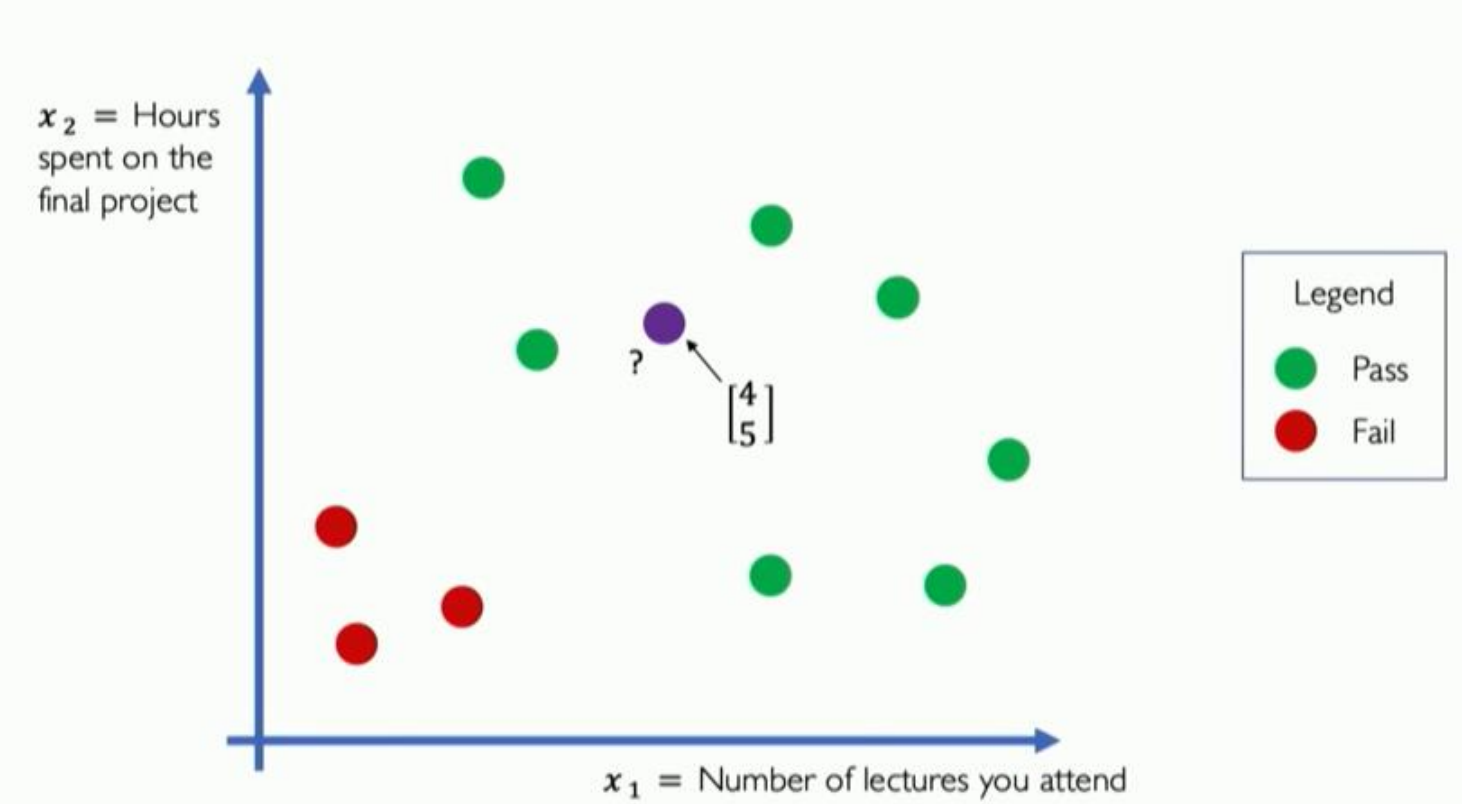
- K-means clustering is an algorithm to classify or to group the objects based on attributes/features into K number of group.
- K is positive integer number.
- The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.



Introduction to Deep Learning

Example Problem: Let's start with a simple two feature model

x1 = Number of lectures attended
x2 = Hours spent on the final exam/project



Decision Tree Algorithms

Information Gain

In order to pick which, feature to split on, we need a way of measuring how good the split is. This is where *information gain* and *entropy* come in.

ID3 (Iterative Dichotomize)

- **Uses Information Theory (Entropy) to split on an attribute that gives the highest information gain**

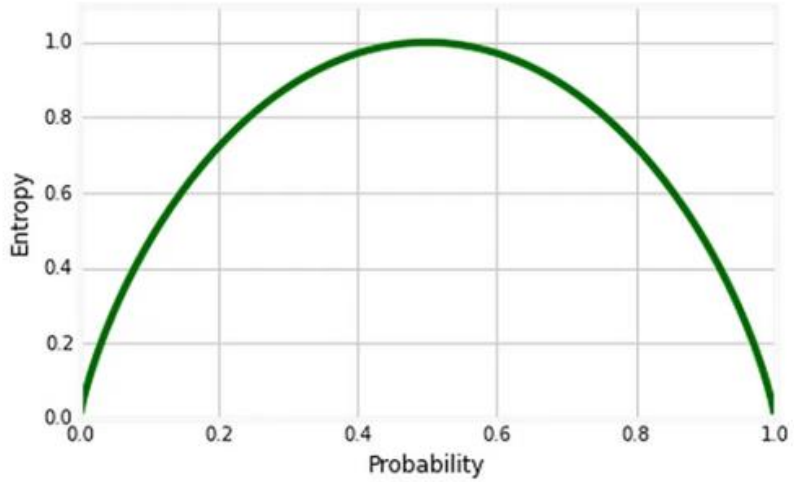
Entropy (Information Theory)

- **A measure of the impurity of a distribution**
- **Calculation: for a discrete random variable Y taking m distinct values {y1, …, ym}**

$$H = \sum_{i=1}^{n} -P_i \log_2 P_i$$

The entropy equation. Introduced by Claude Shannon in 1948.

P_i = probability of occurrence of value i
- High entropy → All the classes are (nearly) equally likely
- Low entropy → A few classes are likely; most of the classes are rarely observed



Entropy versus Probability of belonging to a class.

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Entropy

- Entropy is a measure of the impurity of a distribution, defined as:

$$H = \sum_{i=1}^{n} -P_i \log_2 P_i$$

- P_i = probability of occurrence of value i
  - High entropy → All the classes are (nearly) equally likely
  - Low entropy → A few classes are likely; most of the classes are rarely observed
  - assume $0 \log_2 0 = 0$

Decision Trees

Usually **ID3** algorithm is used to build the decision tree:
~ it is a top-down greedy search of possible branches

→ it uses **entropy** and **information gain** to build the tree

The **H(X)** Shannon-entropy of a dicrete random variable **X** with possible values $x_1 x_2 \dots x_n$ and probability mass function **P(X)** is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

Example: https://en.wikipedia.org/wiki/Entropy_(information_theory)

For completely homogoeneous dataset (all TRUE or all FALSE values): entropy is **0**
If the dataset is equally divided (same amount of TRUEs and FALSEs): entropy is **1**

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

PLAYING GOLF
→ **9** times **YES**
→ **5** times **NO**

We just have to use the Shannon-entropy formula to calculate the **H(x)** values

H(PlayingGolf) = H(9,5) =

$$= -(0.64 \log_2 0.64) - (0.36 \log_2 0.36) = 0.94$$

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Attribute: Outlook

$Values\,(Outlook) = Sunny, Overcast, Rain$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$

$S_{Sunny} \leftarrow [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971$

$S_{Overcast} \leftarrow [4+, 0-]$ $\quad Entropy(S_{Overcast}) = -\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4} = 0$

$S_{Rain} \leftarrow [3+, 2-]$ $\quad Entropy(S_{Rain}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.971$

$$Gain\,(S, Outlook) = Entropy(S) - \sum_{v \in \{Sunny, Overcast, Rain\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Outlook) = Entropy(S) - \frac{5}{14}Entropy(S_{Sunny}) - \frac{4}{14}Entropy(S_{Overcast}) - \frac{5}{14}Entropy(S_{Rain})$$

$$Gain(S, Outlook) = 0.94 - \frac{5}{14}0.971 - \frac{4}{14}0 - \frac{5}{14}0.971 = 0.2464$$

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$

$S_{Hot} \leftarrow [2+, 2-]$ $\quad Entropy(S_{Hot}) = -\frac{2}{4} log_2 \frac{2}{4} - \frac{2}{4} log_2 \frac{2}{4} = 1.0$

$S_{Mild} \leftarrow [4+, 2-]$ $\quad Entropy(S_{Mild}) = -\frac{4}{6} log_2 \frac{4}{6} - \frac{2}{6} log_2 \frac{2}{6} = 0.9183$

$S_{Cool} \leftarrow [3+, 1-]$ $\quad Entropy(S_{Cool}) = -\frac{3}{4} log_2 \frac{3}{4} - \frac{1}{4} log_2 \frac{1}{4} = 0.8113$

$$Gain\ (S, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Temp) = Entropy(S) - \frac{4}{14} Entropy(S_{Hot}) - \frac{6}{14} Entropy(S_{Mild}) - \frac{4}{14} Entropy(S_{Cool})$$

$$Gain(S, Temp) = 0.94 - \frac{4}{14} 1.0 - \frac{6}{14} 0.9183 - \frac{4}{14} 0.8113 = 0.0289$$

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Attribute: Humidity

$$Values\,(Humidity) = High, Normal$$

$$S = [9+, 5-] \qquad Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$$

$$S_{High} \leftarrow [3+, 4-] \qquad Entropy(S_{High}) = -\frac{3}{7} log_2 \frac{3}{7} - \frac{4}{7} log_2 \frac{4}{7} = 0.9852$$

$$S_{Normal} \leftarrow [6+, 1-] \qquad Entropy(S_{Normal}) = -\frac{6}{7} log_2 \frac{6}{7} - \frac{1}{7} log_2 \frac{1}{7} = 0.5916$$

$$Gain\,(S, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Humidity)$$

$$= Entropy(S) - \frac{7}{14} Entropy(S_{High}) - \frac{7}{14} Entropy(S_{Normal})$$

$$Gain(S, Humidity) = 0.94 - \frac{7}{14} 0.9852 - \frac{7}{14} 0.5916 = 0.1516$$

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Attribute: Wind

$Values\,(Wind) = Strong, Weak$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$

$S_{Strong} \leftarrow [3+, 3-]$ $\quad Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [6+, 2-]$ $\quad Entropy(S_{Weak}) = -\frac{6}{8} log_2 \frac{6}{8} - \frac{2}{8} log_2 \frac{2}{8} = 0.8113$

$$Gain\,(S, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Wind) = Entropy(S) - \frac{6}{14} Entropy(S_{Strong}) - \frac{8}{14} Entropy(S_{Weak})$$

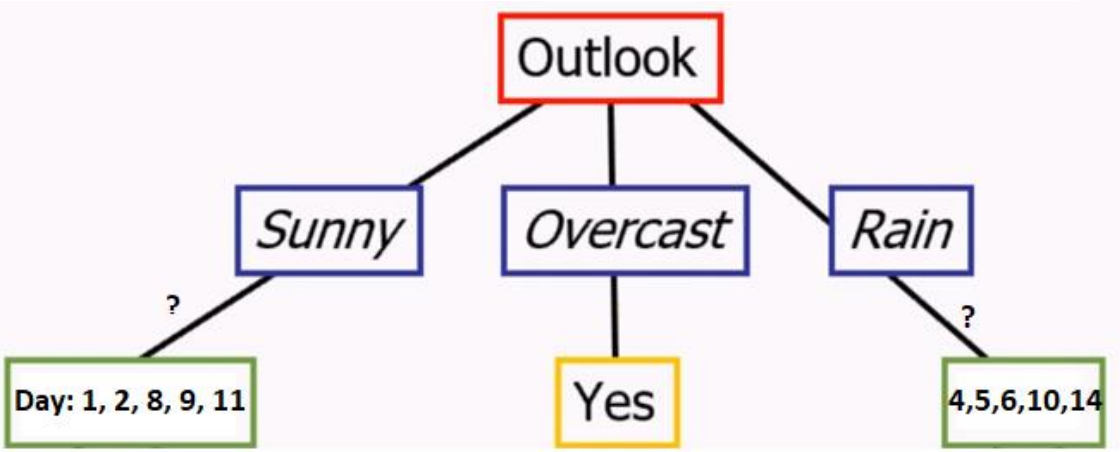
$$Gain(S, Wind) = 0.94 - \frac{6}{14}\, 1.0 - \frac{8}{14}\, 0.8113 = 0.0478$$

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

$$Gain(S, Outlook) = 0.2464 \qquad Gain(S, Humidity) = 0.1516$$

$$Gain(S, Temp) = 0.0289 \qquad Gain(S, Wind) = 0.0478$$



Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
sunny	mild	high	false	no
sunny	cold	normal	false	yes
sunny	mild	normal	true	yes

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 2-]$ $\quad Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Mild}) = 1.0$

$S_{Cool} \leftarrow [1+, 0-]$ $\quad Entropy(S_{Cool}) = 0.0$

$$Gain\ (S_{Sunny}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Temp)$$

$$= Entropy(S) - \frac{2}{5}Entropy(S_{Hot}) - \frac{2}{5}Entropy(S_{Mild})$$

$$- \frac{1}{5}Entropy(S_{Cool})$$

$$Gain(S_{sunny}, Temp) = 0.97 - \frac{2}{5}0.0 - \frac{2}{5}1 - \frac{1}{5}0.0 = 0.570$$

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
sunny	mild	high	false	no
sunny	cold	normal	false	yes
sunny	mild	normal	true	yes

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S) = -\frac{2}{5} log_2 \frac{2}{5} - \frac{3}{5} log_2 \frac{3}{5} = 0.97$

$S_{high} \leftarrow [0+, 3-]$ $\quad Entropy(S_{High}) = 0.0$

$S_{Normal} \leftarrow [2+, 0-]$ $\quad Entropy(S_{Normal}) = 0.0$

$$Gain\ (S_{Sunny}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Humidity) = Entropy(S) - \frac{3}{5} Entropy(S_{High}) - \frac{2}{5} Entropy(S_{Normal})$$

$$Gain(S_{sunny}, Humidity) = 0.97 - \frac{3}{5}\, 0.0 - \frac{2}{5} 0.0 = 0.97$$

Decision Trees

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
sunny	mild	high	false	no
sunny	cold	normal	false	yes
sunny	mild	normal	true	yes

Attribute: Wind

$Values\ (Wind) = Strong, Weak$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Strong} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [1+, 2-]$ $\quad Entropy(S_{Weak}) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$

$$Gain(S_{sunny}, Temp) = 0.570$$

$$Gain(S_{sunny}, Humidity) = 0.97$$

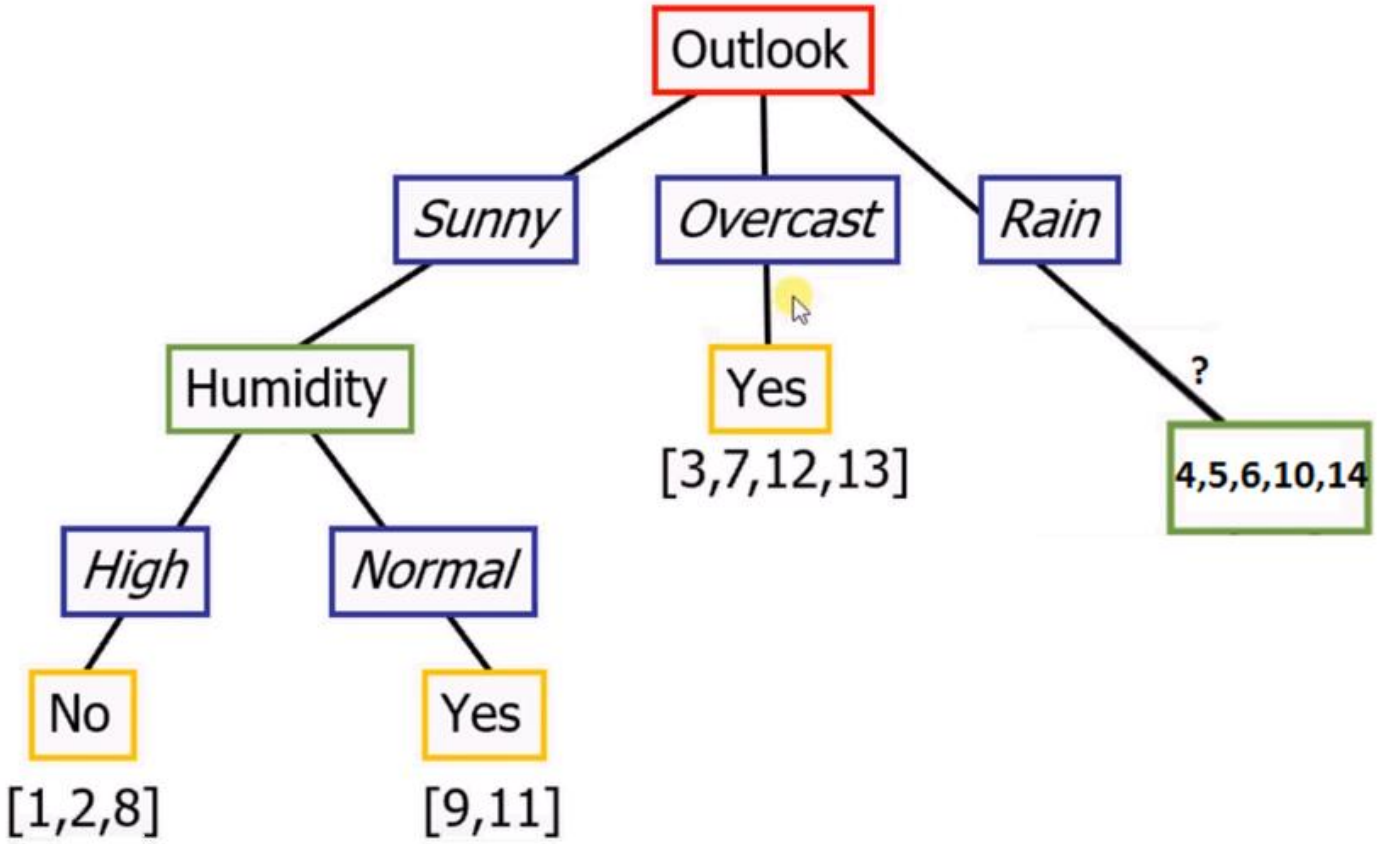
$$Gain(S_{sunny}, Wind) = 0.0192$$

$$Gain\ (S_{Sunny}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{sunny}, Wind) = 0.97 - \frac{2}{5}1.0 - \frac{3}{5}0.918 = 0.0192$$

Decision Trees



Decision Trees

outlook	temperature	humidity	wind	play
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
rainy	mild	normal	false	yes
rainy	mild	high	true	no

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S_{Rain} = [3+, 2-]$ $\quad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 0-]$ $\quad Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [2+, 1-]$ $\quad Entropy(S_{Mild}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$S_{Cool} \leftarrow [1+.1-]$ $\quad Entropy(S_{Cool}) = 1.0$

$$Gain\ (S_{Rain}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Temp)$$

$$= Entropy(S) - \frac{0}{5}Entropy(S_{Hot}) - \frac{3}{5}Entropy(S_{Mild})$$

$$- \frac{2}{5}Entropy(S_{Cool})$$

$$Gain(S_{Rain}, Temp) = 0.97 - \frac{0}{5}0.0 - \frac{3}{5}0.918 - \frac{2}{5}1.0 = 0.0192$$

Decision Trees

outlook	temperature	humidity	wind	play
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
rainy	mild	normal	false	yes
rainy	mild	high	true	no

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S_{Rain} = [3+, 2-]$

$S_{High} \leftarrow [1+, 1-]$

$S_{Normal} \leftarrow [2+, 1-]$

$Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$Entropy(S_{High}) = 1.0$

$Entropy(S_{Normal}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$$Gain\ (S_{Rain}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Humidity) = Entropy(S) - \frac{2}{5}Entropy(S_{High}) - \frac{3}{5}Entropy(S_{Normal})$$

$$Gain(S_{Rain}, Humidity) = 0.97 - \frac{2}{5}\ 1.0 - \frac{3}{5}0.918 = 0.0192$$

Decision Trees

outlook	temperature	humidity	wind	play
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
rainy	mild	normal	false	yes
rainy	mild	high	true	no

Attribute: Wind

$Values\ (wind) = Strong, Weak$

$S_{Rain} = [3+, 2-]$

$S_{Strong} \leftarrow [0+, 2-]$

$S_{Weak} \leftarrow [3+, 0-]$

$Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$Entropy(S_{Strong}) = 0.0$

$Entropy(S_{weak}) = 0.0$

$$Gain(S_{Rain}, Temp) = 0.0192$$

$$Gain(S_{Rain}, Humidity) = 0.0192$$

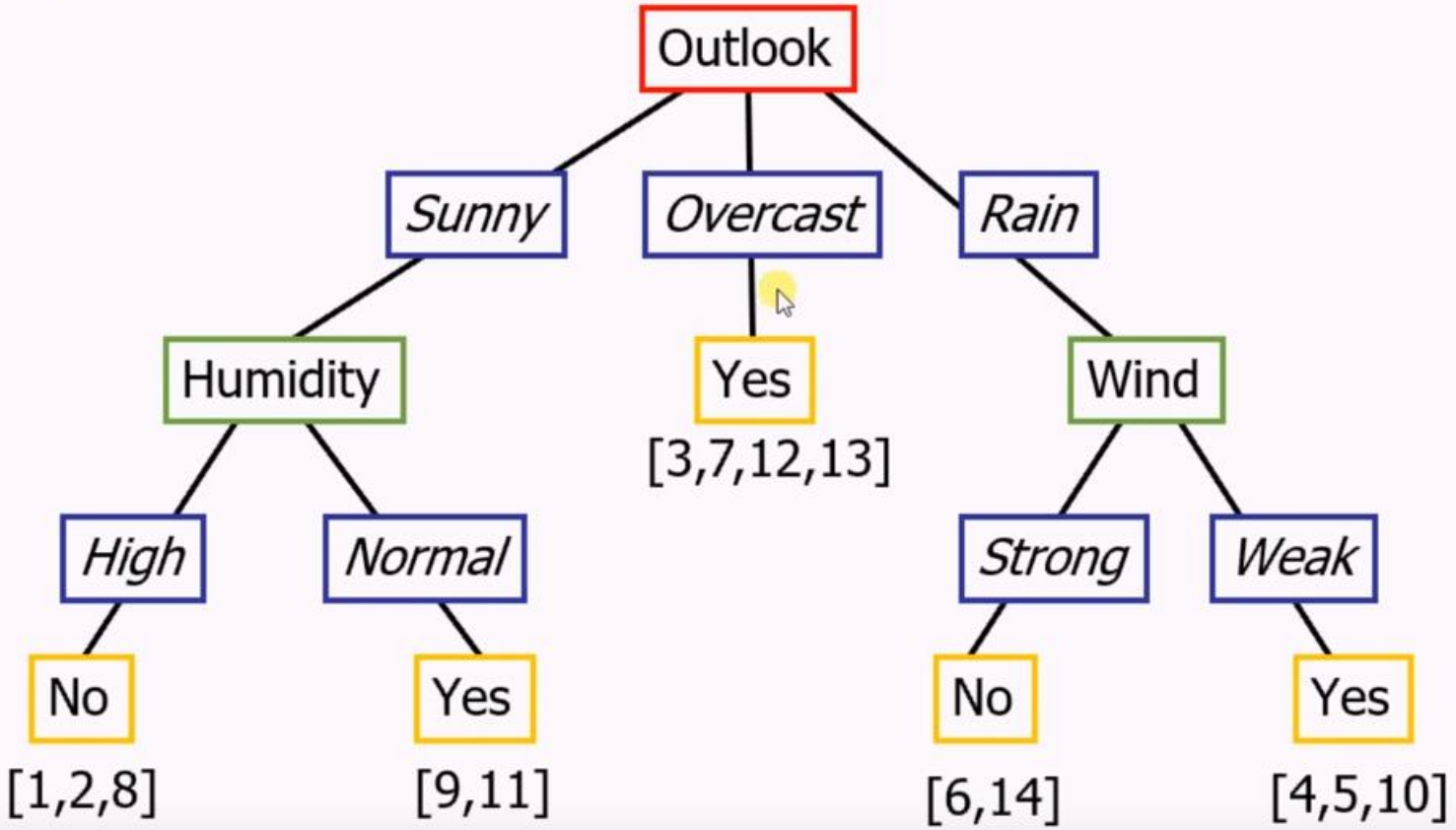
$$Gain(S_{Rain}, Wind) = 0.97$$

$$Gain\ (S_{Rain}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{Rain}, Wind) = 0.97 - \frac{2}{5}\ 0.0 - \frac{3}{5}0.0 = 0.97$$

Complete Decision Tree



Decision Tree Learning Algorithm

- There are many specific decision-tree algorithms. Notable ones include:
- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)
- CHAID (Chi-squared Automatic Interaction Detector). Performs multi-level splits when computing classification trees.
- MARS: extends decision trees to handle numerical data better.

ID3 Algorithm

- ID3 is one of the most common decision tree algorithm
- Dichotomisation means dividing into two completely opposite things.
- Algorithm iteratively divides attributes into two groups which are the most dominant attribute and others to construct a tree.
- Then, it calculates the Entropy and Information Gains of each attribute. In this way, the most dominant attribute can be founded.
- After then, the most dominant one is put on the tree as decision node.
- Entropy and Gain scores would be calculated again among the other attributes.
- Procedure continues until reaching a decision for that branch.

Using XGBoost

Classification

```
from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Using XGBoost

Classification

```
from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

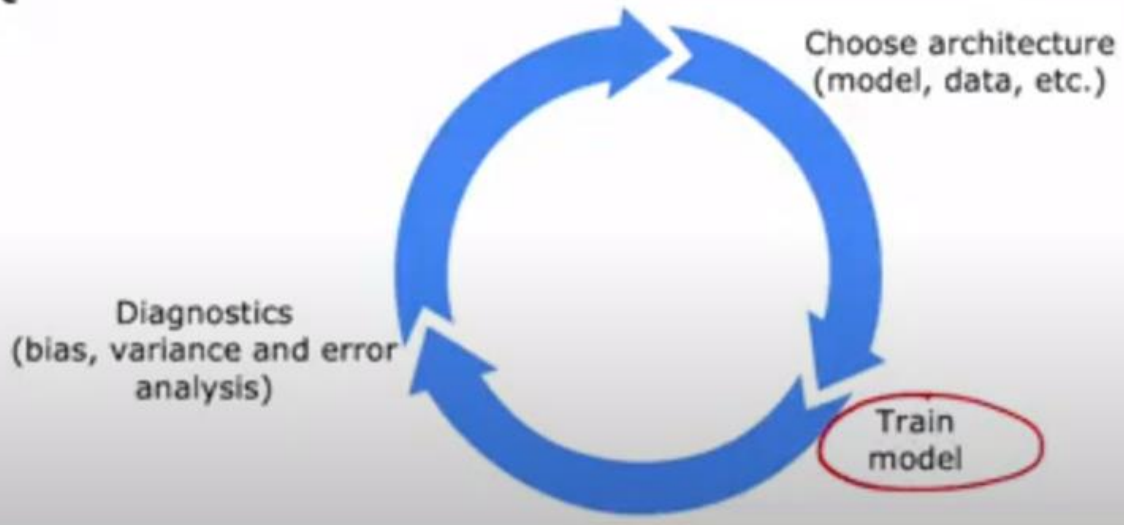
Regression

```
from xgboost import XGBRegressor

model = XGBRegressor()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Decision Trees vs Neural Networks

Decision Trees and Tree ensembles

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast



Random Forest is a popular machine learning algorithm that belongs to the family of ensemble methods. It is based on decision trees and combines the predictions of multiple decision trees to improve the accuracy and robustness of the model. The basic idea behind Random Forest is to build many decision trees, each on a different subset of the training data, and then combine their predictions to obtain a final prediction.

The main advantages of Random Forest are its ability to handle large datasets with many features, its robustness to outliers and noisy data, and its ability to estimate the importance of each feature. The main disadvantages of Random Forest are its computational complexity and the difficulty in interpreting the individual trees.

The idea behind a Random Forest is simple:

We repeatedly select data from the data set (with replacement) and build a Decision Tree with each new sample. It is important to note that since we are sampling with replacement, many data points will be repeated, and many won't be included as well. This is important to keep in mind when we talk about measuring error of a Random Forest. Another important feature of the Random Forest is that each node of the Decision Tree is limited to only considering splits on random subsets of the features.

In the case of classification with Random Forests, we use each tree in our forest to get a prediction, then the label with the most votes becomes the predicted class for that data point. The usual parameters when building a forest (standard defaults used in the SciKit-Learn library) are 10 trees and only considering the square root of ‘n’ features, where n is the total number of features.

The key features of the Random Forest algorithm are:

- Decision Tree: The individual decision trees in a Random Forest are built using the CART (Classification and Regression Trees) algorithm. Each tree is grown independently of the others, using a random subset of the features and a random subset of the training data.

- Bagging: Random Forest uses a technique called bagging (bootstrap aggregating) to create the subsets of the training data. Bagging involves sampling the training data with replacement, which means that some observations may appear in multiple subsets, and some may not appear at all.

- Randomness: Random Forest introduces randomness into the algorithm in two ways. First, the subsets of the training data and the features used to build each tree are selected randomly. Second, at each node of the tree, a random subset of the features is considered for splitting.

- Ensemble: The final prediction of the Random Forest is based on the ensemble of the predictions of the individual trees. In classification problems, the majority vote of the trees is used to make the final prediction. In regression problems, the average of the predictions of the trees is used.

XGBClassifier is a popular machine learning algorithm that belongs to the family of gradient boosting methods. It is based on decision trees and combines the predictions of multiple decision trees to improve the accuracy and robustness of the model.

The basic idea behind XGBClassifier is to build many decision trees sequentially, where each tree corrects the errors of the previous tree.

The main advantages of XGBClassifier are its ability to handle large datasets with many features, its high accuracy, and its efficiency in training and prediction. The main disadvantages of XGBClassifier are its sensitivity to the choice of hyperparameters and its difficulty in interpreting the individual trees.

The key features of the XGBClassifier algorithm are:

- Decision Tree: XGBClassifier uses a modified version of the CART (Classification and Regression Trees) algorithm to build decision trees. The modifications include the use of a regularization term to prevent overfitting and the use of a weighted quantile sketch algorithm to efficiently find the optimal split points.

- Boosting: XGBClassifier uses a technique called boosting to create the sequence of decision trees. Boosting involves training each tree on a modified version of the training data, where the observations that were misclassified by the previous trees are given more weight.

- Gradient Descent: XGBClassifier uses a variant of gradient descent called gradient boosting to minimize the loss function. The loss function measures the difference between the predicted values and the actual values, and the goal is to minimize this difference.

- Regularization: XGBClassifier uses two types of regularization to prevent overfitting. The first type is called shrinkage, which involves multiplying the predictions of each tree by a small constant. The second type is called feature subsampling, which involves randomly selecting a subset of the features to consider for each split.

- Ensemble: The final prediction of XGBClassifier is based on the ensemble of the predictions of the individual trees. In classification problems, the majority vote of the trees is used to make the final prediction. In regression problems, the average of the predictions of the trees is used.

	precision	recall	f1-score	support
absent	0.89	0.80	0.84	20
present	0.43	0.60	0.50	5
accuracy			0.76	25
macro avg	0.66	0.70	0.67	25
weighted avg	0.80	0.76	0.77	25

Precision is the proportion of correctly predicted positive observations (i.e., true positives) out of all observations that the model predicted as positive. It can be calculated as: true positives / (true positives + false positives)

Precision measures the accuracy of the positive predictions made by the model. A high precision indicates that the model makes very few false positive predictions.

Recall is the proportion of correctly predicted positive observations (i.e., true positives) out of all actual positive observations. It can be calculated as: true positives / (true positives + false negatives)

Recall measures the ability of the model to identify all positive observations in the test data. A high recall indicates that the model makes very few false negative predictions.

F1 score is the harmonic mean of precision and recall, and it provides a balanced measure of the two metrics. It can be calculated as: 2 * (precision * recall) / (precision + recall)

F1 score ranges from 0 to 1, where 1 indicates perfect precision and recall, and 0 indicates that the model's predictions are completely wrong.

Decision Trees

We will use the dataset below to learn a decision tree which predicts if people pass machine learning (Yes or No), based on their previous GPA (High, Medium, or Low) and whether or not they studied.

GPA	Studied	Passed
L	F	F
L	T	T
M	F	F
M	T	T
H	F	T
H	T	T

For this problem, you can write your answers using $\log_2$, but it may be helpful to note that $\log_2 3 \approx 1.6$.

1. What is the entropy H(Passed)?

$$H(Passed) = -(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6})$$
$$H(Passed) = -(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3})$$
$$H(Passed) = \boxed{\log_2 3 - \frac{2}{3} \approx 0.92}$$

Decision Trees

We will use the dataset below to learn a decision tree which predicts if people pass machine learning (Yes or No), based on their previous GPA (High, Medium, or Low) and whether or not they studied.

GPA	Studied	Passed
L	F	F
L	T	T
M	F	F
M	T	T
H	F	T
H	T	T

For this problem, you can write your answers using $\log_2$, but it may be helpful to note that $\log_2 3 \approx 1.6$.

1. What is the entropy H(Passed)?

$$H(Passed) = -(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6})$$
$$H(Passed) = -(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3})$$
$$H(Passed) = \boxed{\log_2 3 - \frac{2}{3} \approx 0.92}$$

2. What is the entropy H(Passed | GPA)?

$$H(Passed|GPA) = -\frac{1}{3}(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) - \frac{1}{3}(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) - \frac{1}{3}(1\log_2 1)$$
$$H(Passed|GPA) = \frac{1}{3}(1) + \frac{1}{3}(1) + \frac{1}{3}(0)$$
$$H(Passed|GPA) = \boxed{\frac{2}{3} \approx 0.66}$$

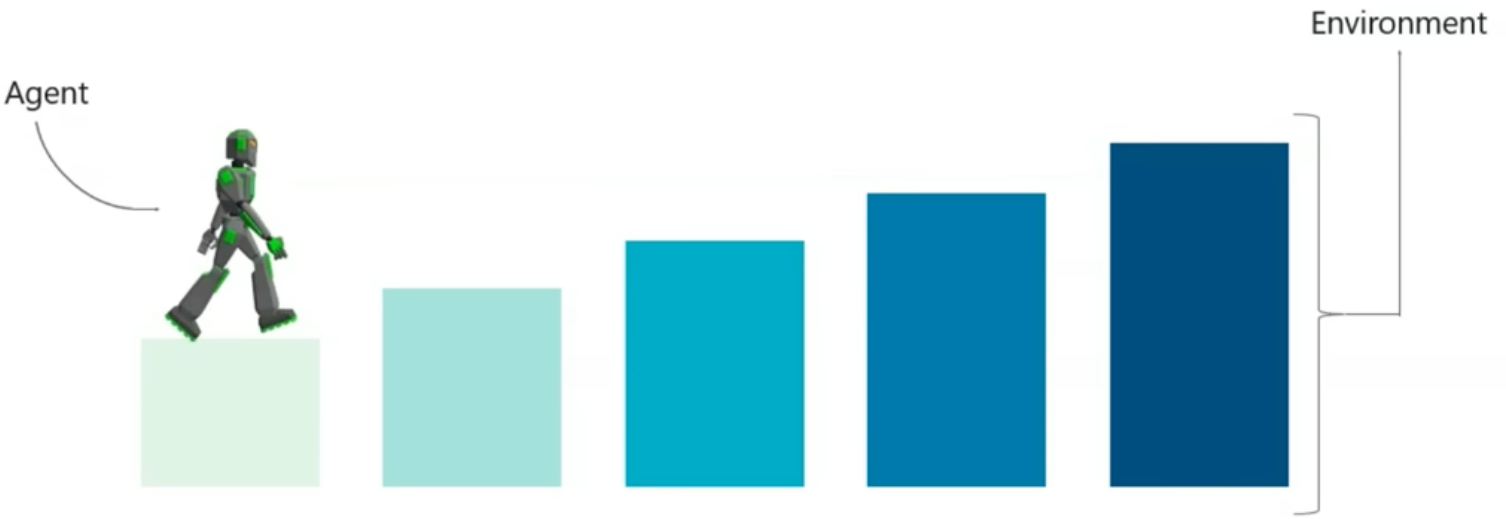
3. What is the entropy H(Passed | Studied)?

$$H(Passed|Studied) = -\frac{1}{2}(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}) - \frac{1}{2}(1\log_2 1)$$
$$H(Passed|Studied) = \frac{1}{2}(\log_2 3 - \frac{2}{3}$$
$$H(Passed|Studied) = \boxed{\frac{1}{2}\log_2 3 - \frac{1}{3} \approx 0.46}$$

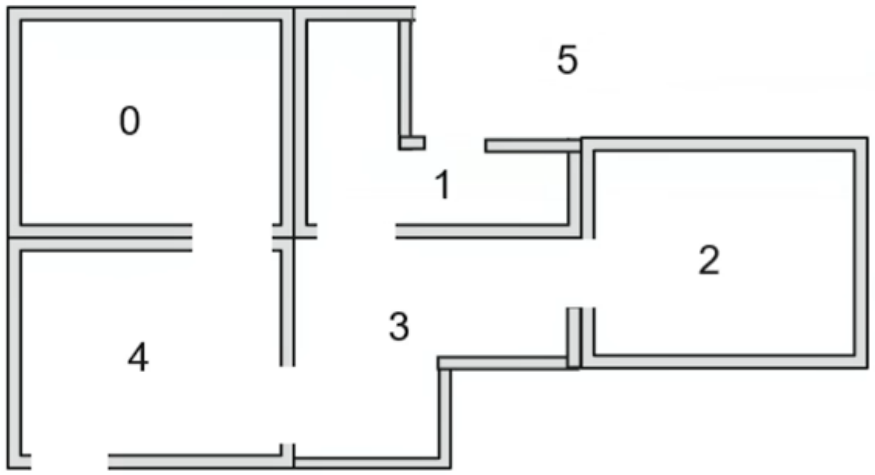
Reinforcement Learning

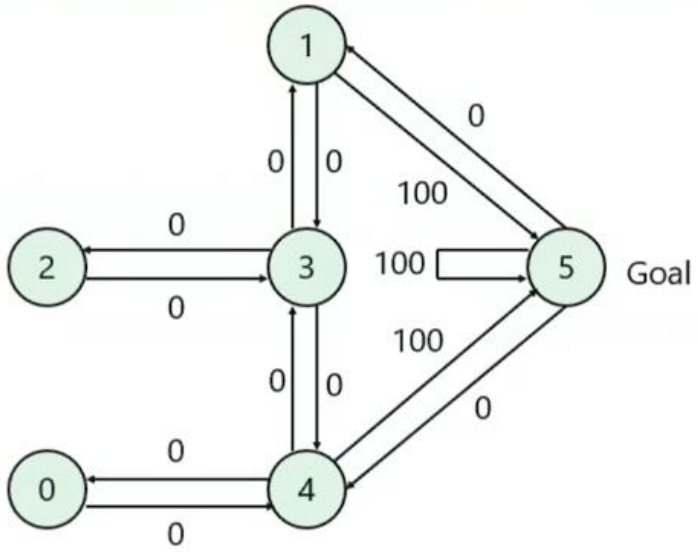
Reinforcement Learning system is comprised of two main components:

- Agent
- Environment



Reinforcement Learning





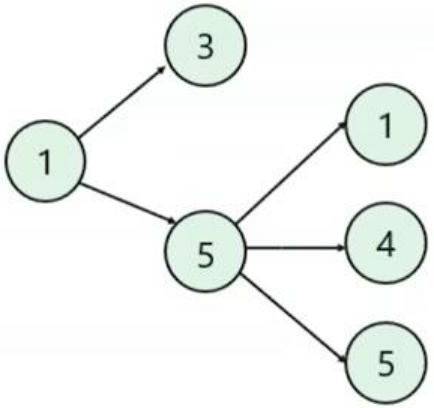
$Q(1,5) = R(1,5) + 0.8 * \text{Max}[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100$

$Q =$

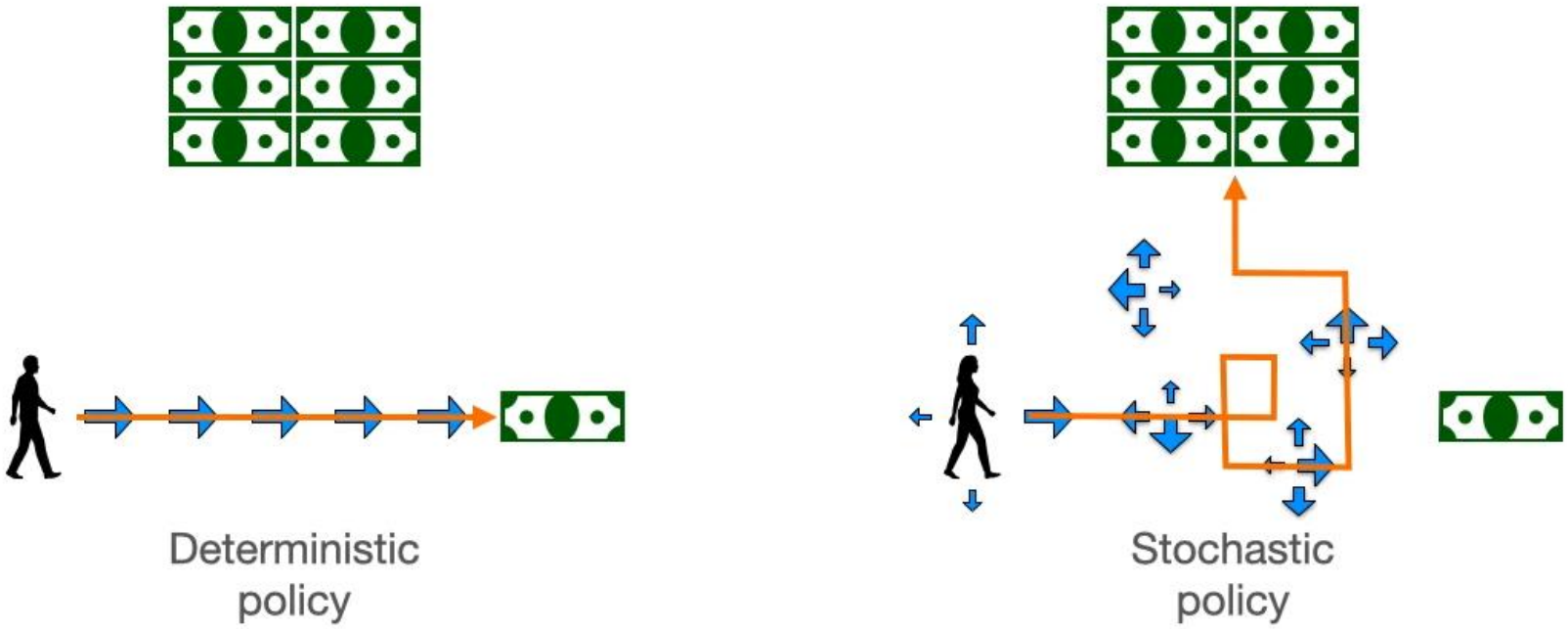
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

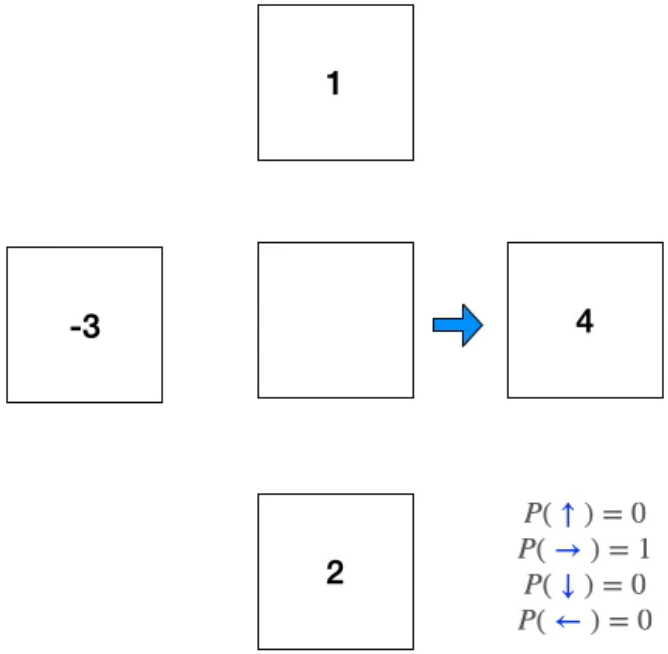
State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

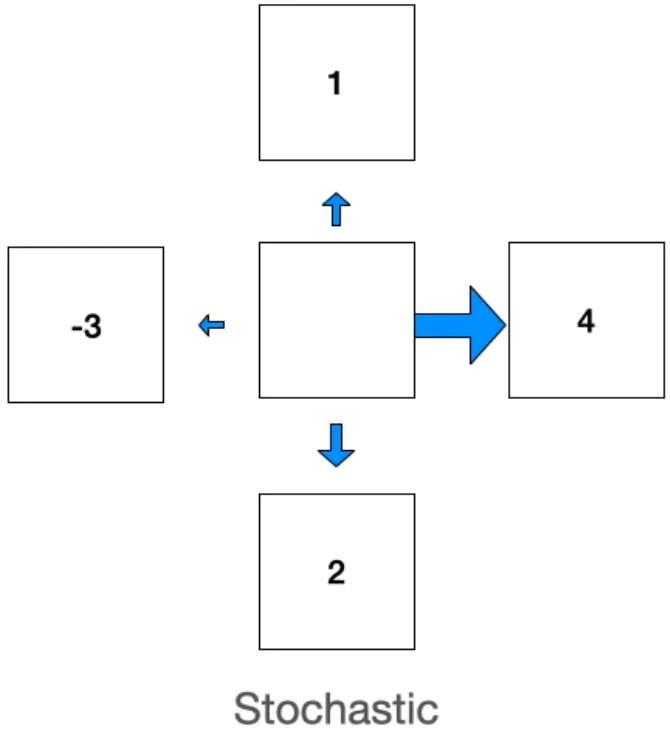


Deterministic vs stochastic



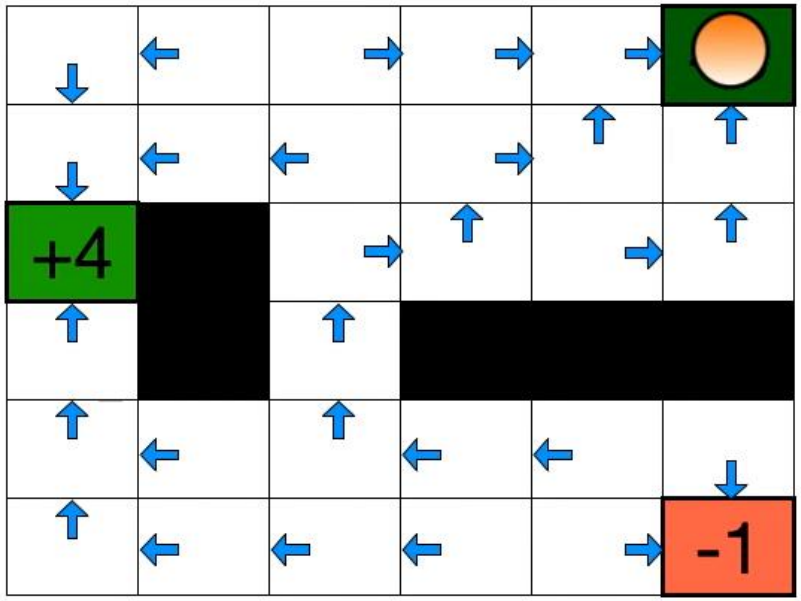
Deterministic vs stochastic



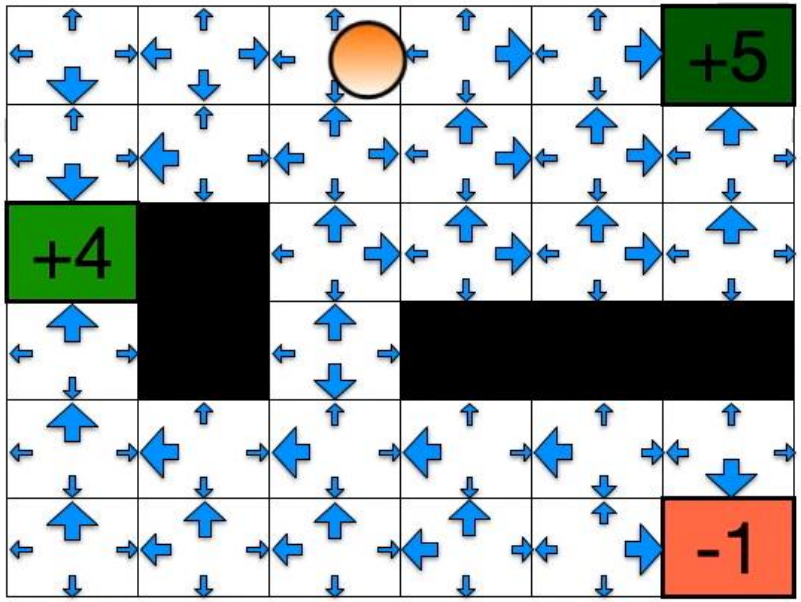


Deterministic vs stochastic

Deterministic

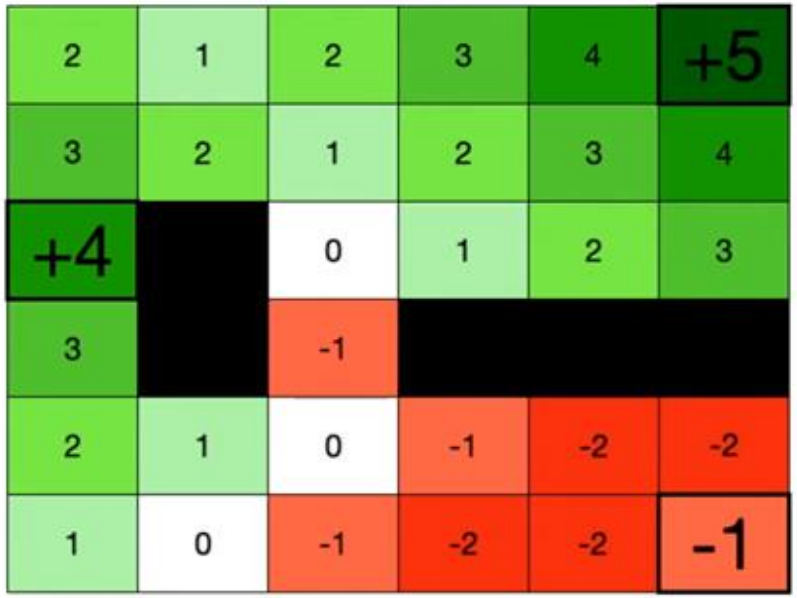


Stochastic

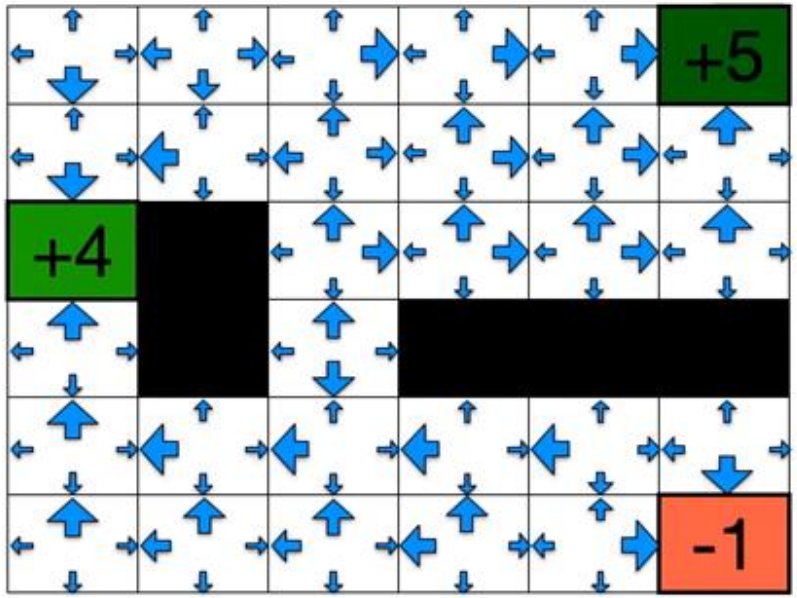


Two neural networks

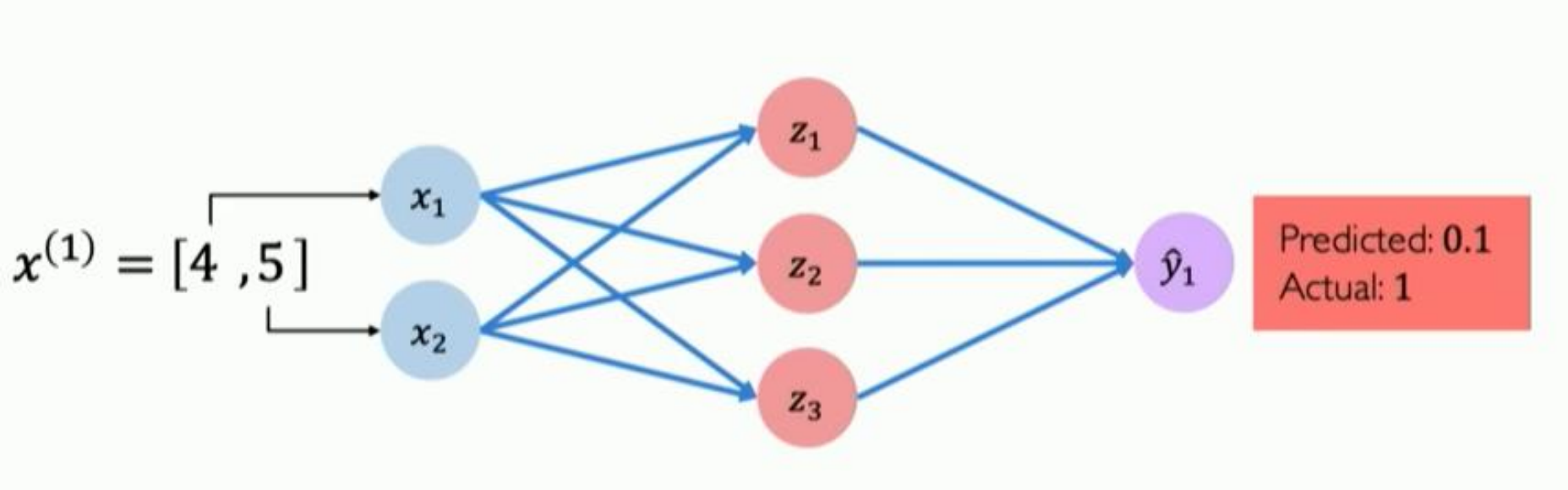
Value neural network



Policy neural network

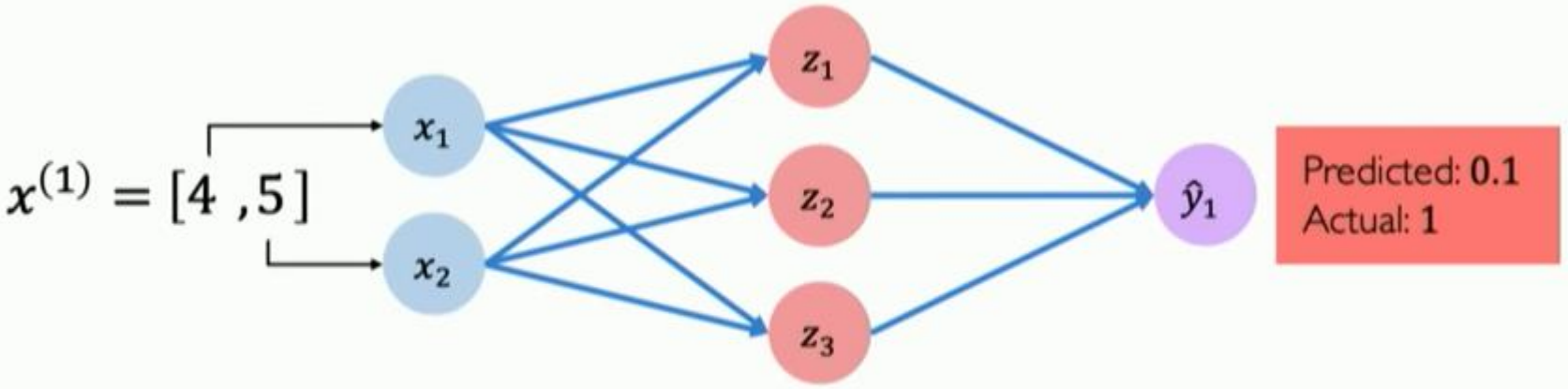


Example Problem: Why is NN predicting 0.1?



Quantifying Loss

*The **loss** of our network measures the cost incurred from incorrect predictions*

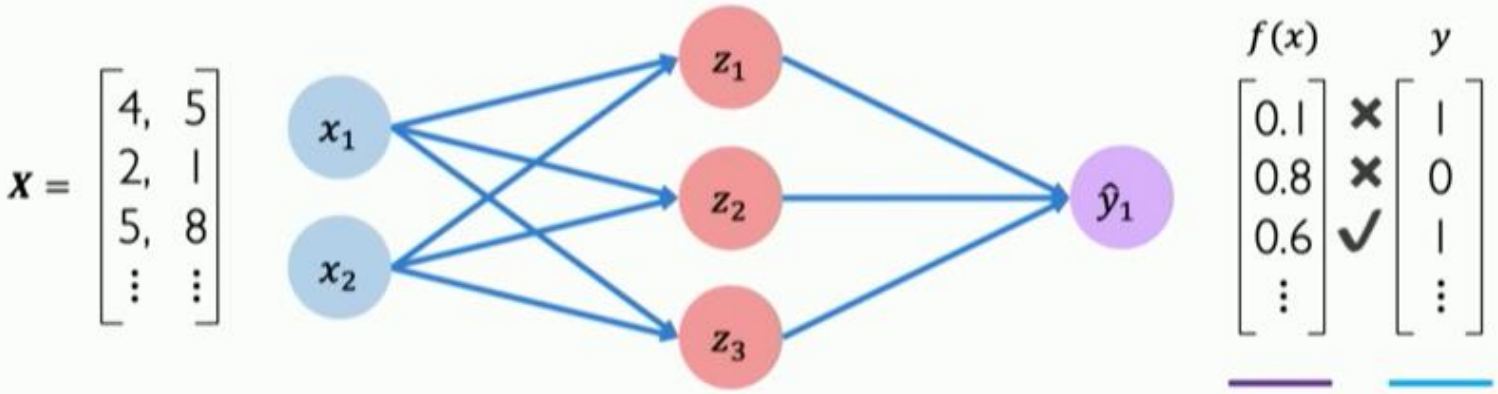


$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Empirical Loss

*The **empirical loss** measures the total loss over our entire dataset*



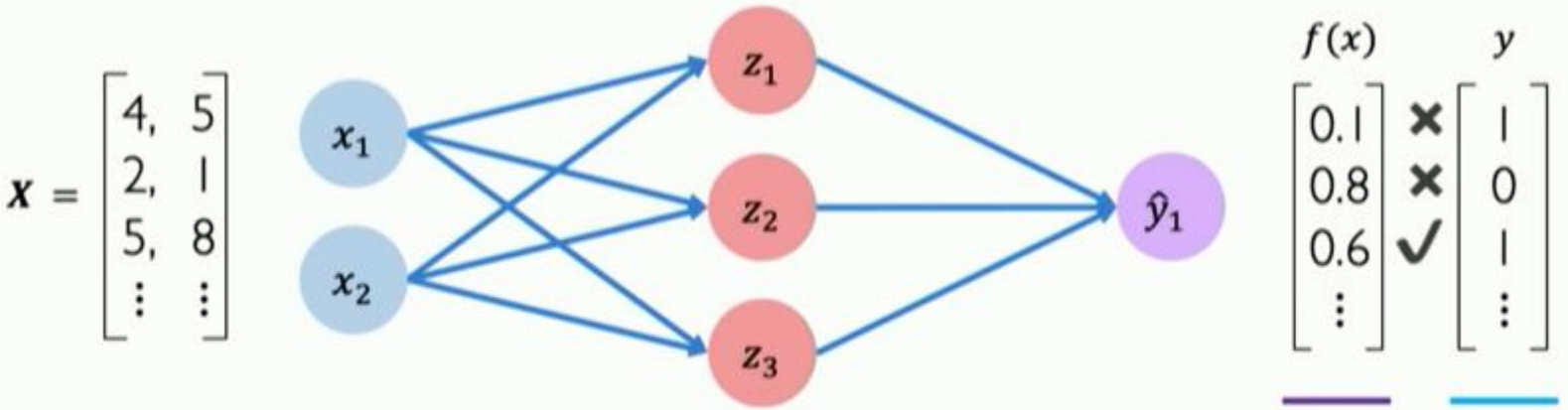
Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

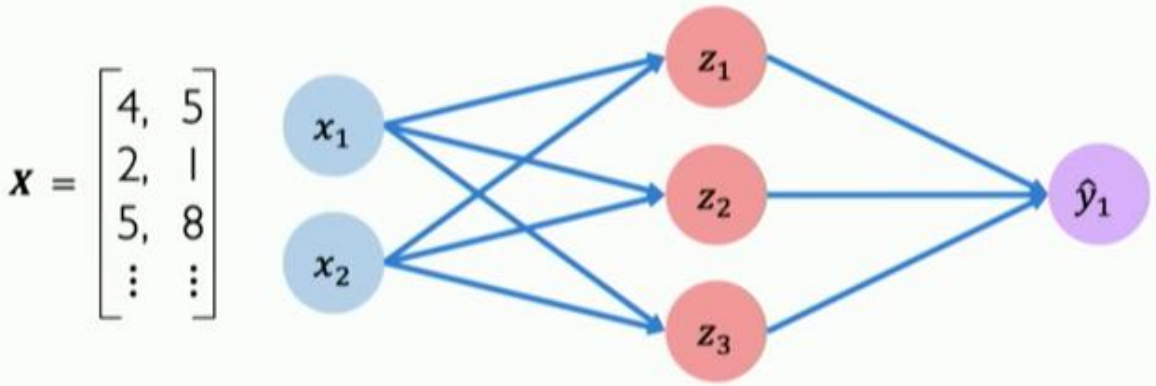


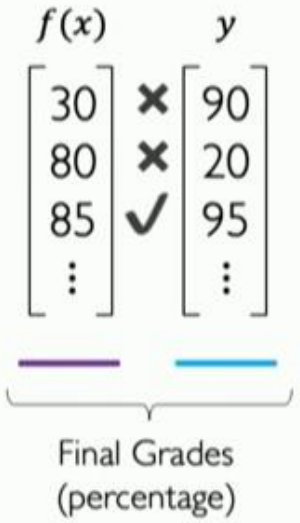
$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log \underbrace{\left(f\left(x^{(i)}; W\right)\right)}_{\text{Predicted}} + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f\left(x^{(i)}; W\right)}_{\text{Predicted}}\right)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*





$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
loss = tf.keras.losses.MSE( y, predicted )
```

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

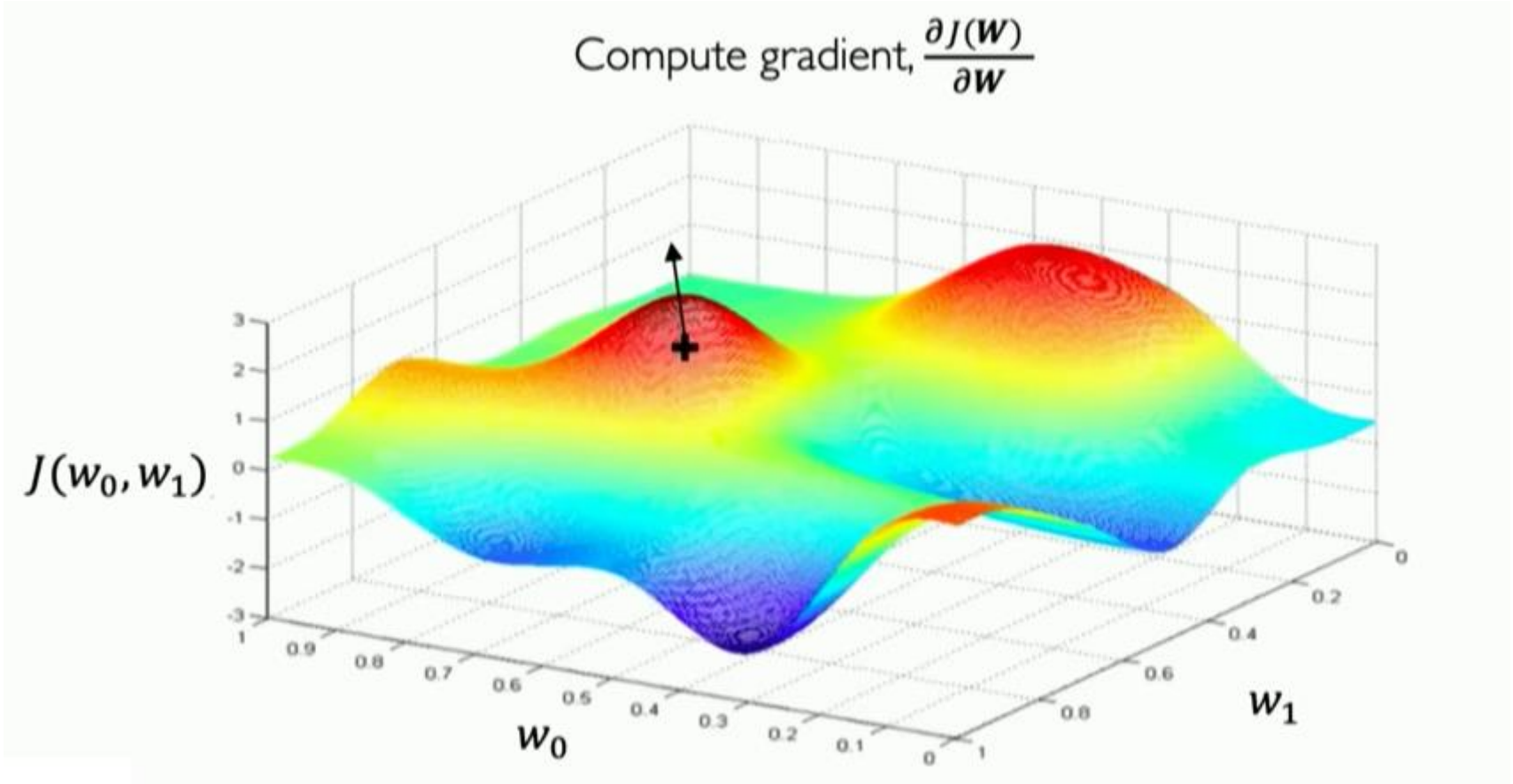
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\text{argmin}}\, J(\boldsymbol{W})$$

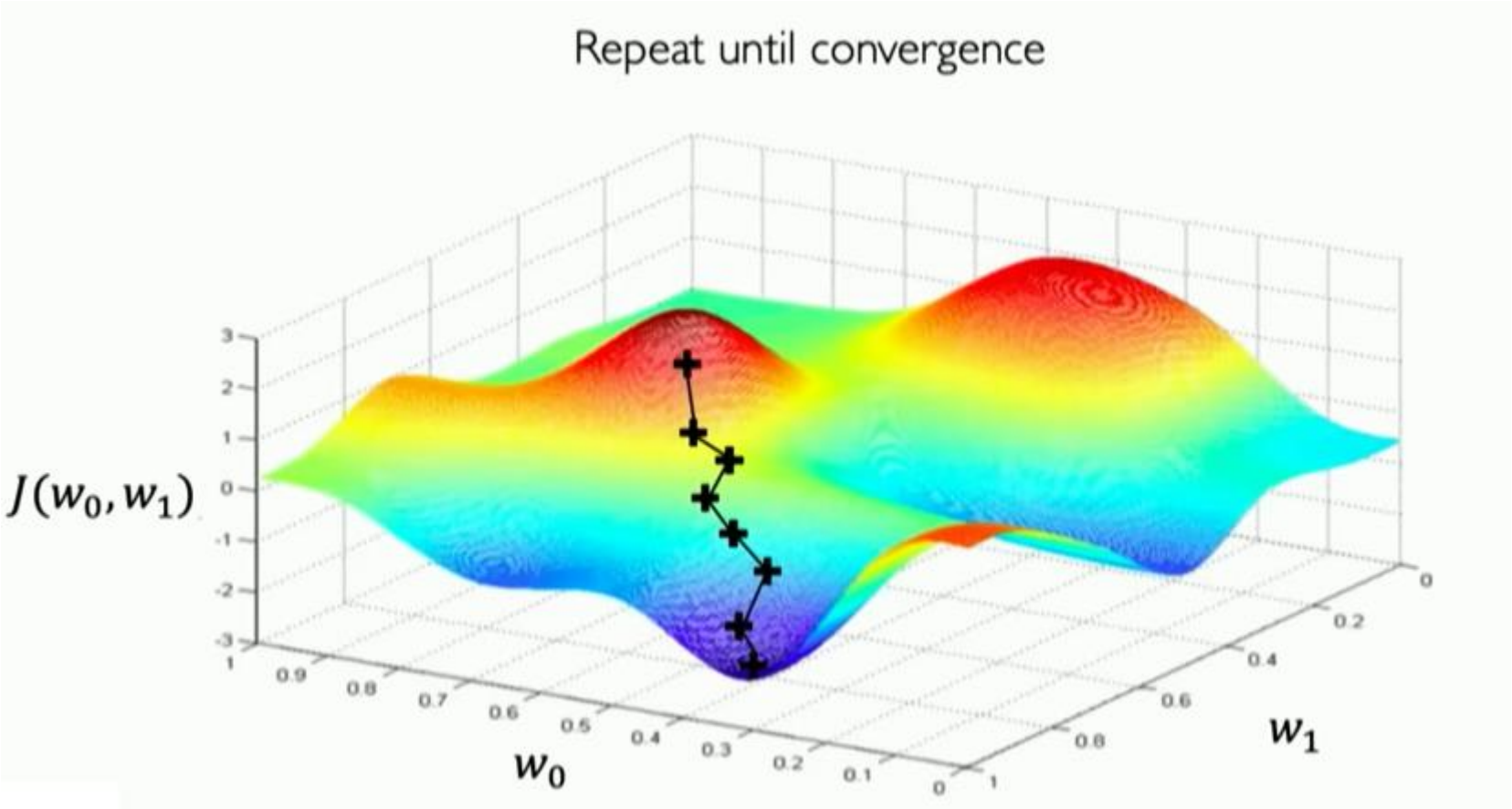
Remember:

$$\boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \dots\}$$

Loss Optimization



Loss Optimization



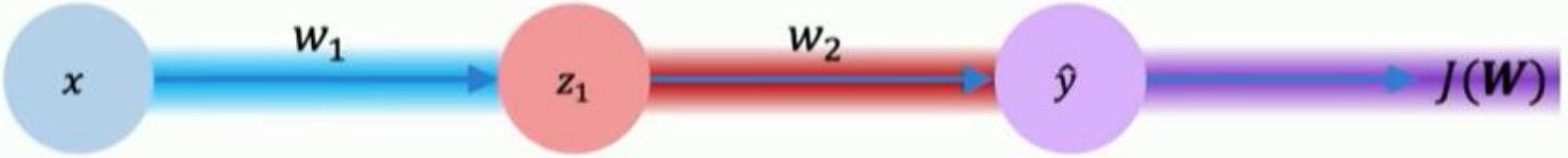
Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

```
import tensorflow as tf

weights = tf.Variable([tf.random.normal()])

while True:    # loop forever
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        gradient = g.gradient(loss, weights)

    weights = weights - lr * gradient
```

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Loss Functions Can Be Difficult to Optimize

Remember:

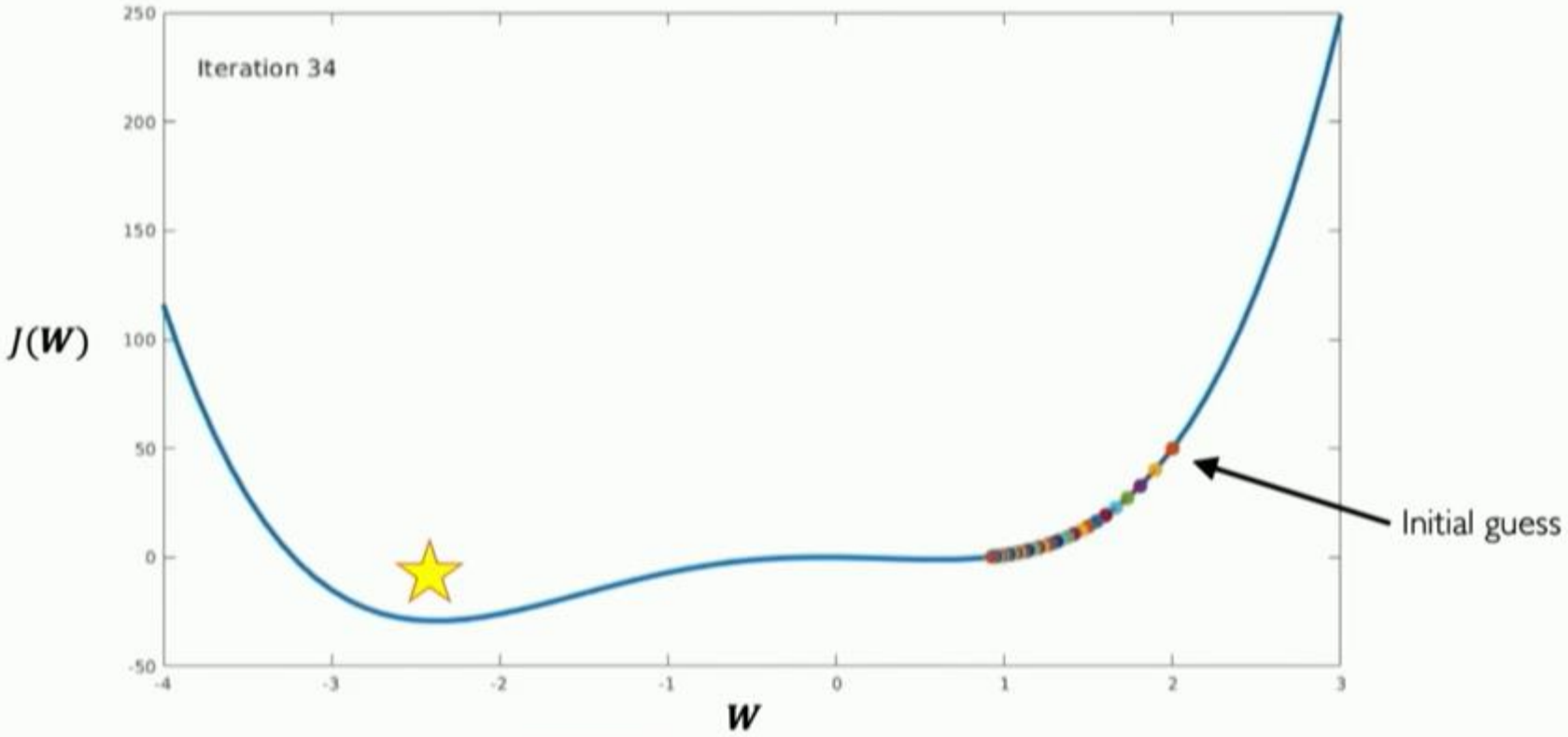
Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

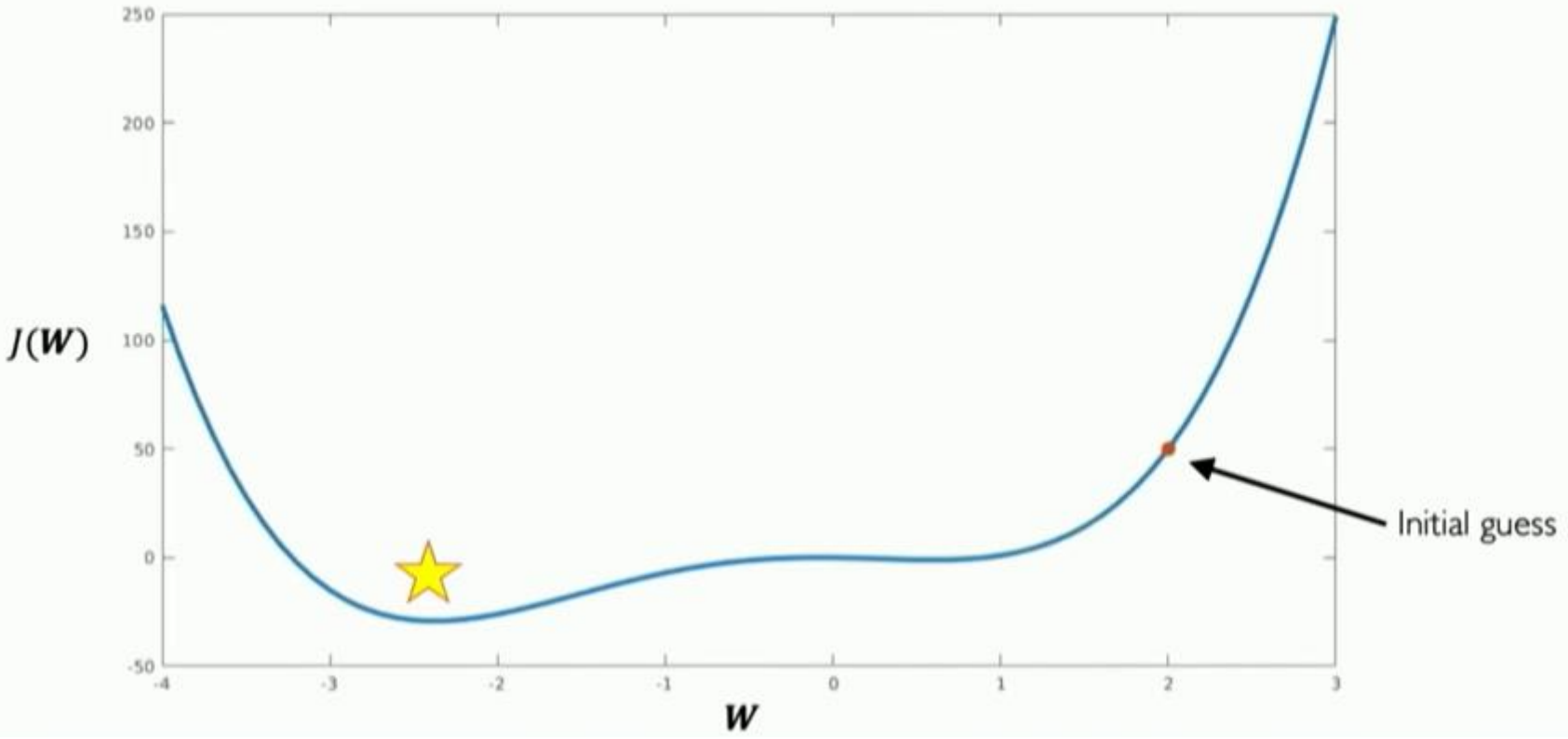
Setting the Learning Rate

Small learning rate *converges slowly and gets stuck in false local minima*



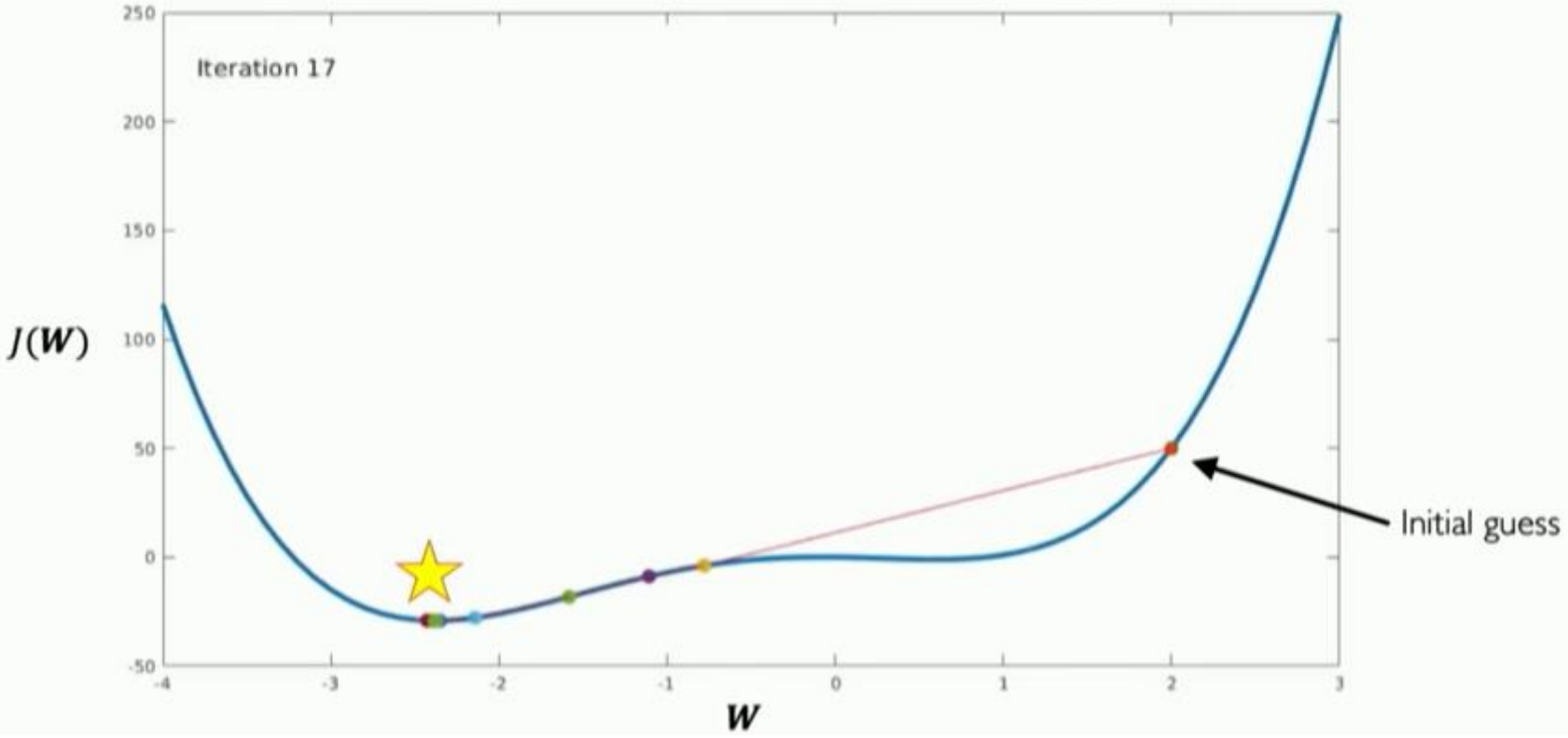
Setting the Learning Rate

Large learning rates *overshoot, become unstable and diverge*



Setting the Learning Rate

Stable learning rates *converge smoothly and avoid local minima*



How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

Idea 2:

Do something smarter!
Design an adaptive learning rate that "adapts" to the landscape

Gradient Descent Algorithms

Algorithm	TF Implementation	Reference
• SGD	`tf.keras.optimizers.SGD`	Kiefer & Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function." 1952.
• Adam	`tf.keras.optimizers.Adam`	Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
• Adadelta	`tf.keras.optimizers.Adadelta`	Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
• Adagrad	`tf.keras.optimizers.Adagrad`	Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
• RMSProp	`tf.keras.optimizers.RMSProp`	

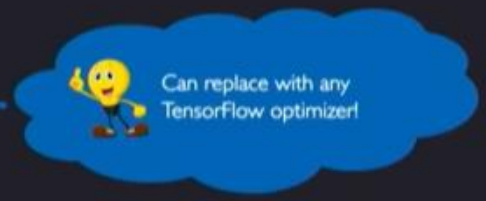
Additional details: http://ruder.io/optimizing-gradient-descent/

Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

Putting it all together

```
import tensorflow as tf

model = tf.keras.Sequential([...])

# pick your favorite optimizer
optimizer = tf.keras.optimizer.SGD()

while True: # loop forever

    # forward pass through the network
    prediction = model(x)

    with tf.GradientTape() as tape:
        # compute the loss
        loss = compute_loss(y, prediction)

    # update the weights using the gradient
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables)))
```

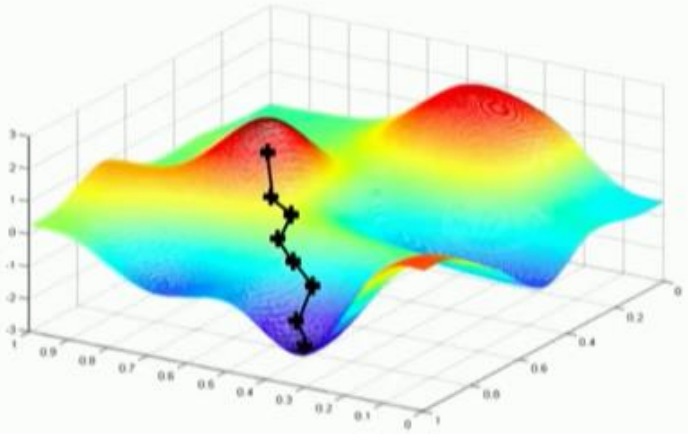


Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

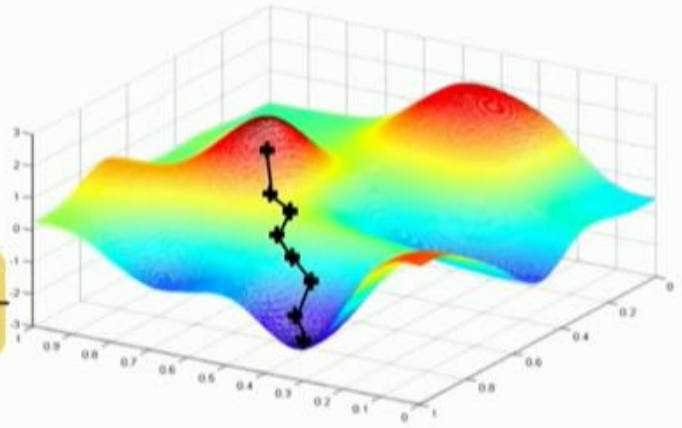
Easy to compute but **very noisy** (stochastic)!



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

Mini-batches while training

More accurate estimation of gradient

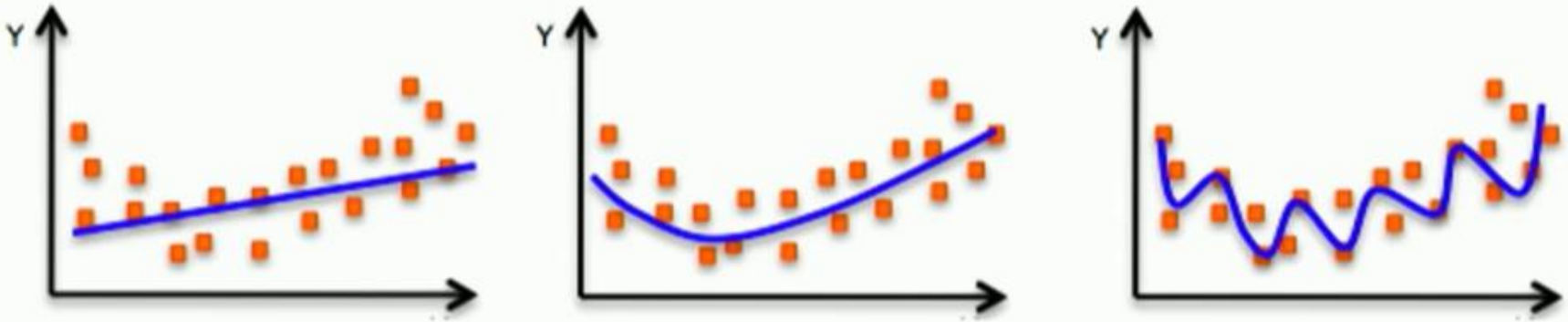
Smoother convergence

Allows for larger learning rates

Mini-batches lead to fast training!

Can parallelize computation + achieve significant speed increases on GPU's

The Problem of Overfitting



Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

Regularization

What is it?

Technique that constrains our optimization problem to discourage complex models

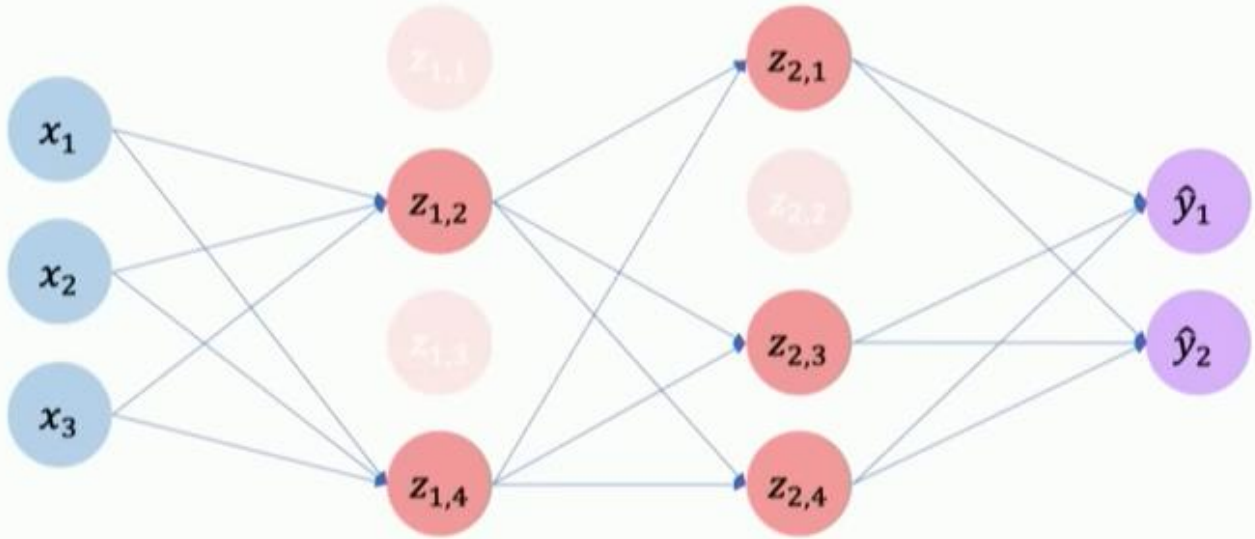
Why do we need it?

Improve generalization of our model on unseen data

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

Convolutional Neural Network (Conv)

When performing a convolution operation in a convolutional neural network (CNN), the kernel is typically flipped or reversed before being applied to the input data. This is also known as a cross-correlation operation.

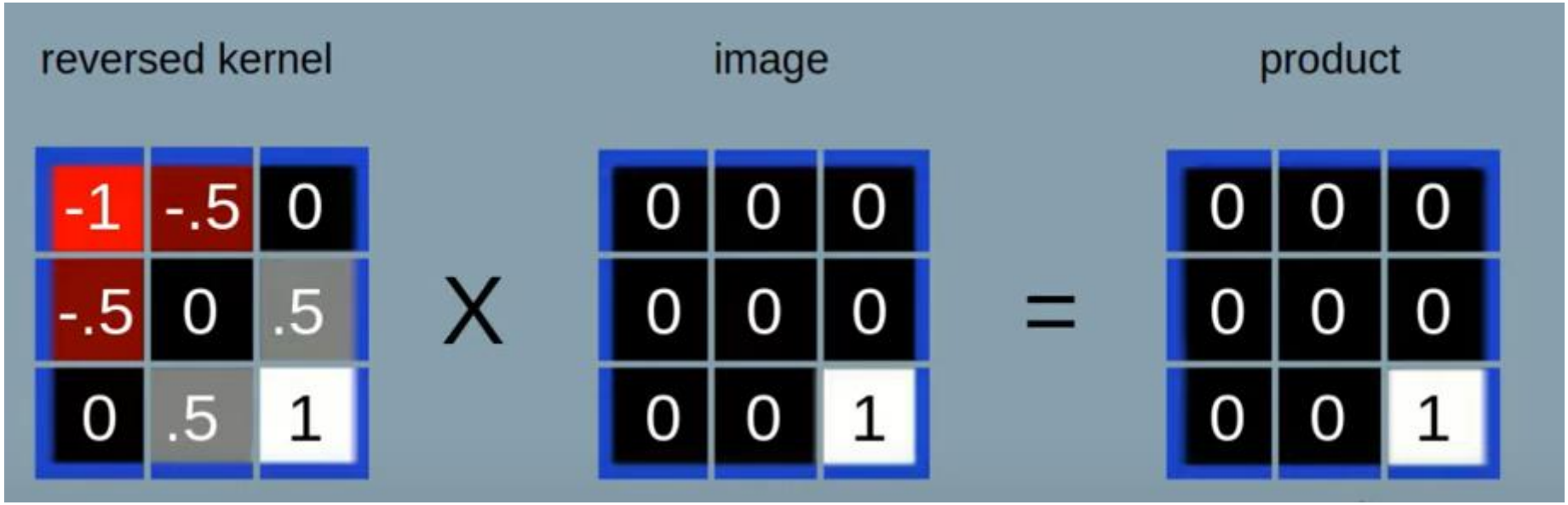
The reason for this reversal is rooted in the mathematical definition of convolution. Convolution involves sliding a kernel over the input data and performing a dot product between the kernel and the corresponding portion of the input. In the standard mathematical definition of convolution, the kernel is flipped before performing the dot product. This flipping ensures that the operation is both associative and commutative.

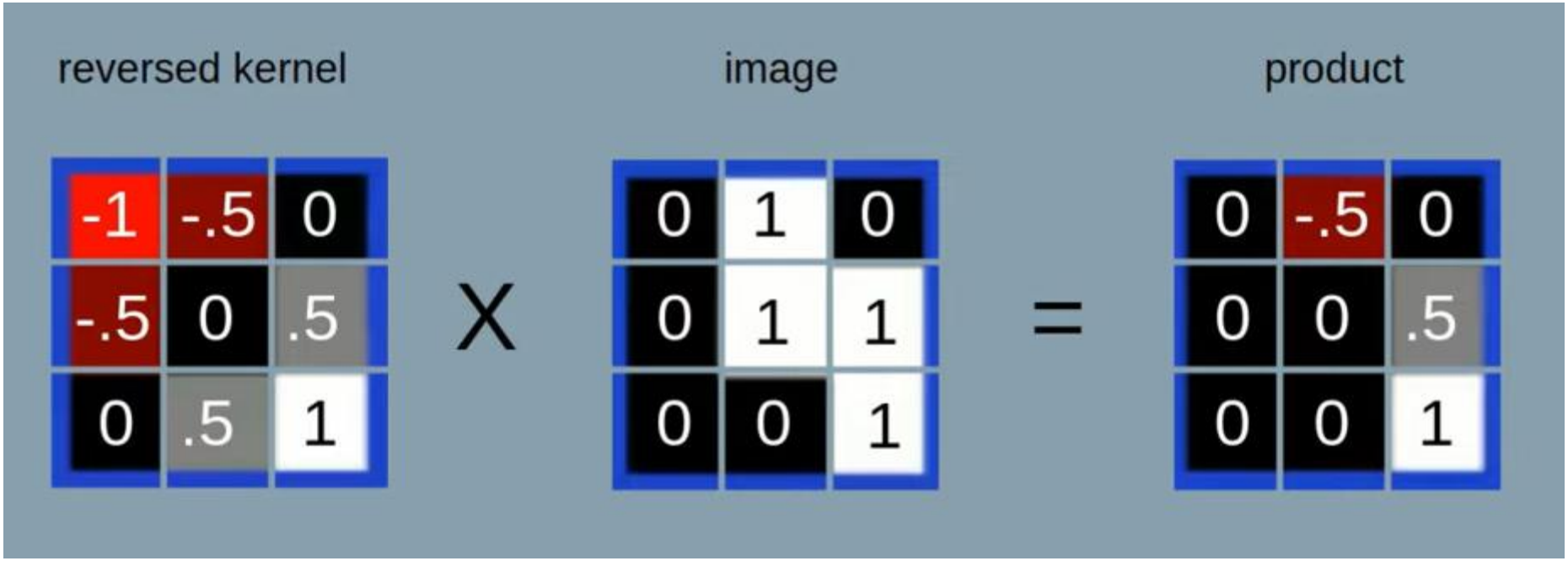
In a practical sense, flipping the kernel ensures that the convolution operation captures the appropriate relationships between features in the input data and the filters in the kernel.

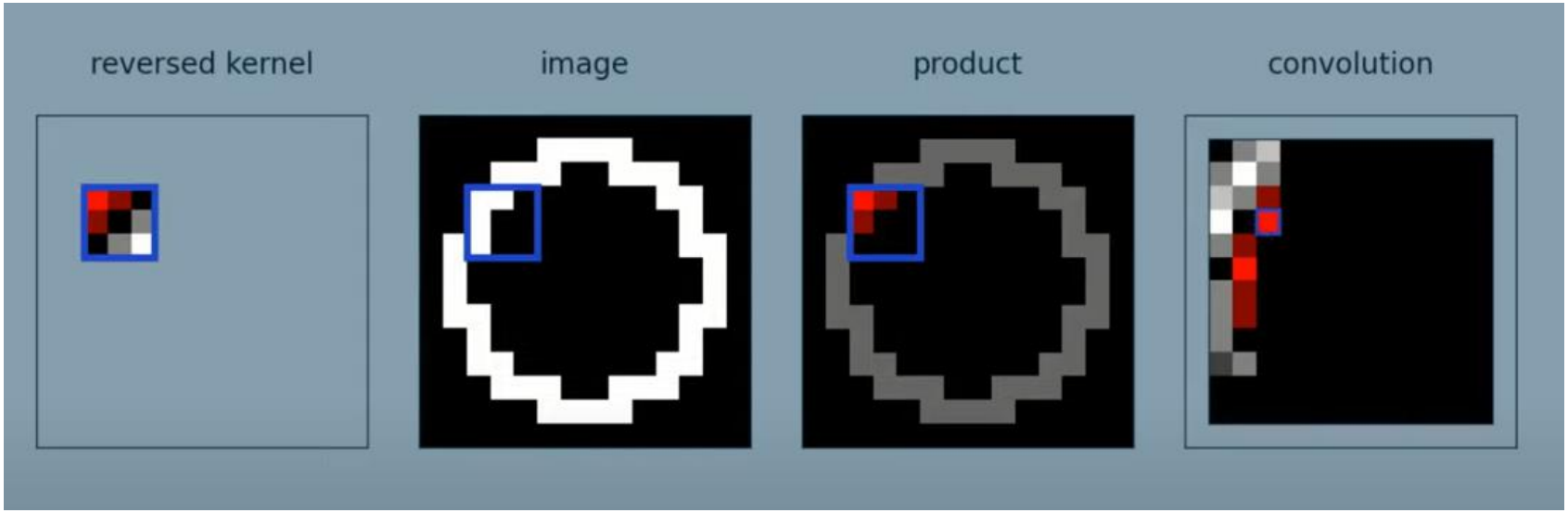
If the kernel were not flipped, the resulting feature maps would be mirror images of the desired output.

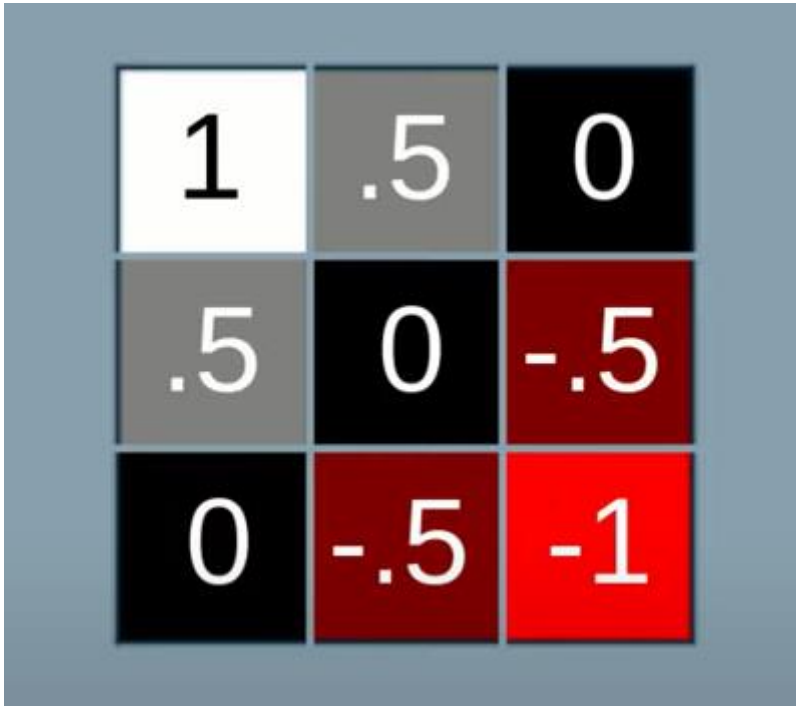
Therefore, the flipping of the kernel is a necessary step in the convolution operation to ensure that the CNN can effectively learn and capture the relevant features in the input data.

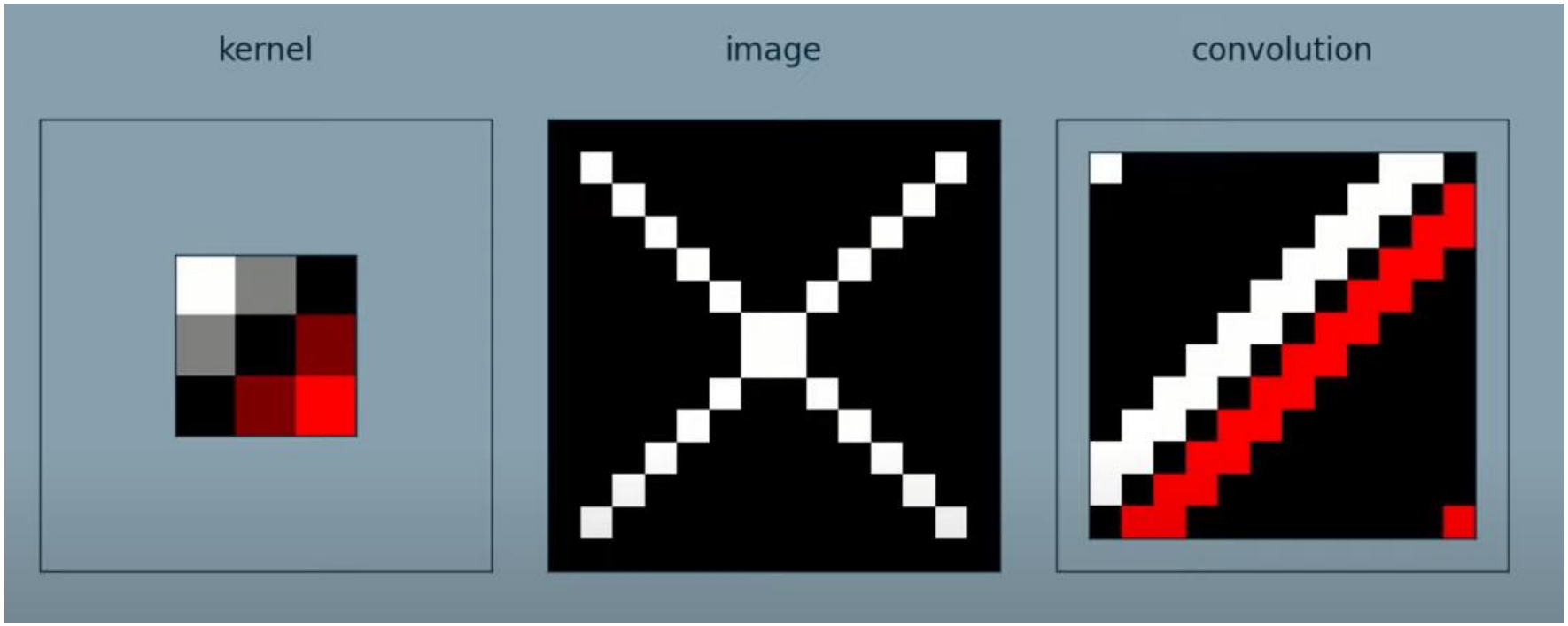
Convolutional Neural Network (Conv)

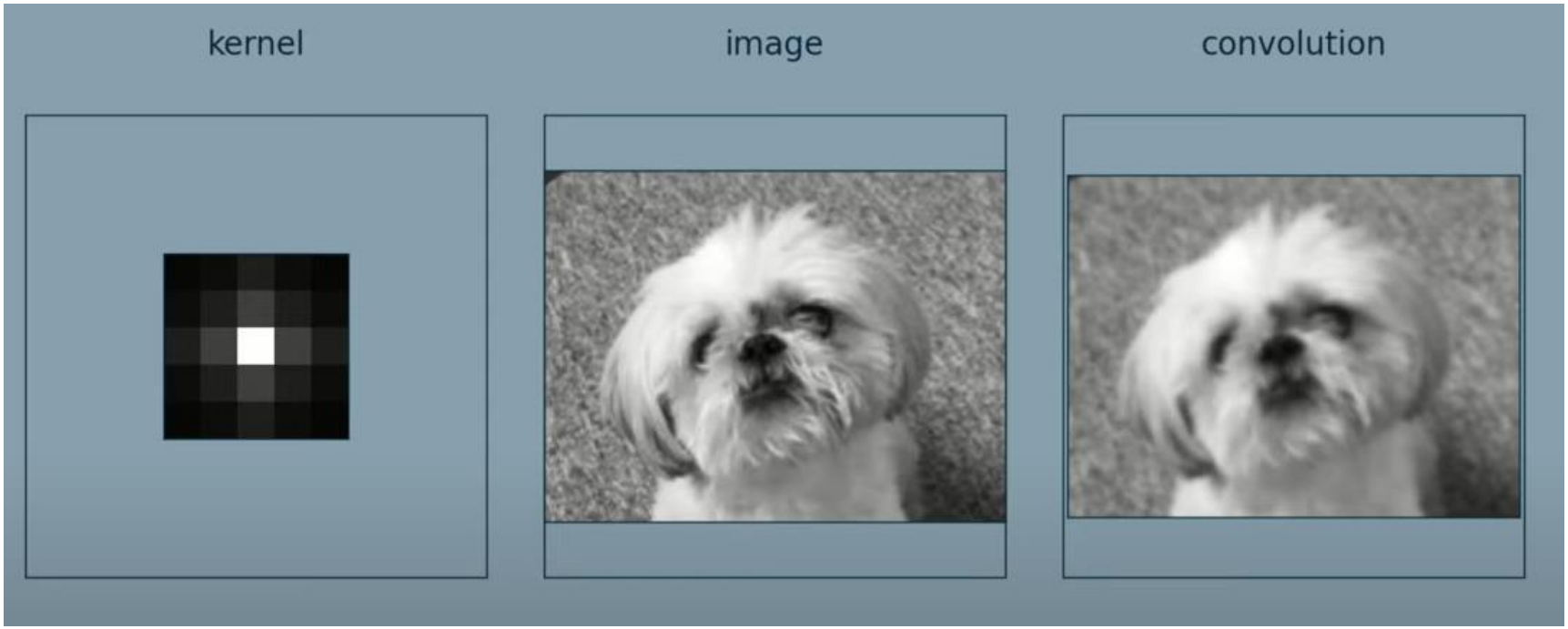



reversed kernel
image
product
-1 -.5 0
-.5 0 .5
0 .5 1
X
0 1 0
0 1 1
0 0 1
=
0 -.5 0
0 0 .5
0 0 1

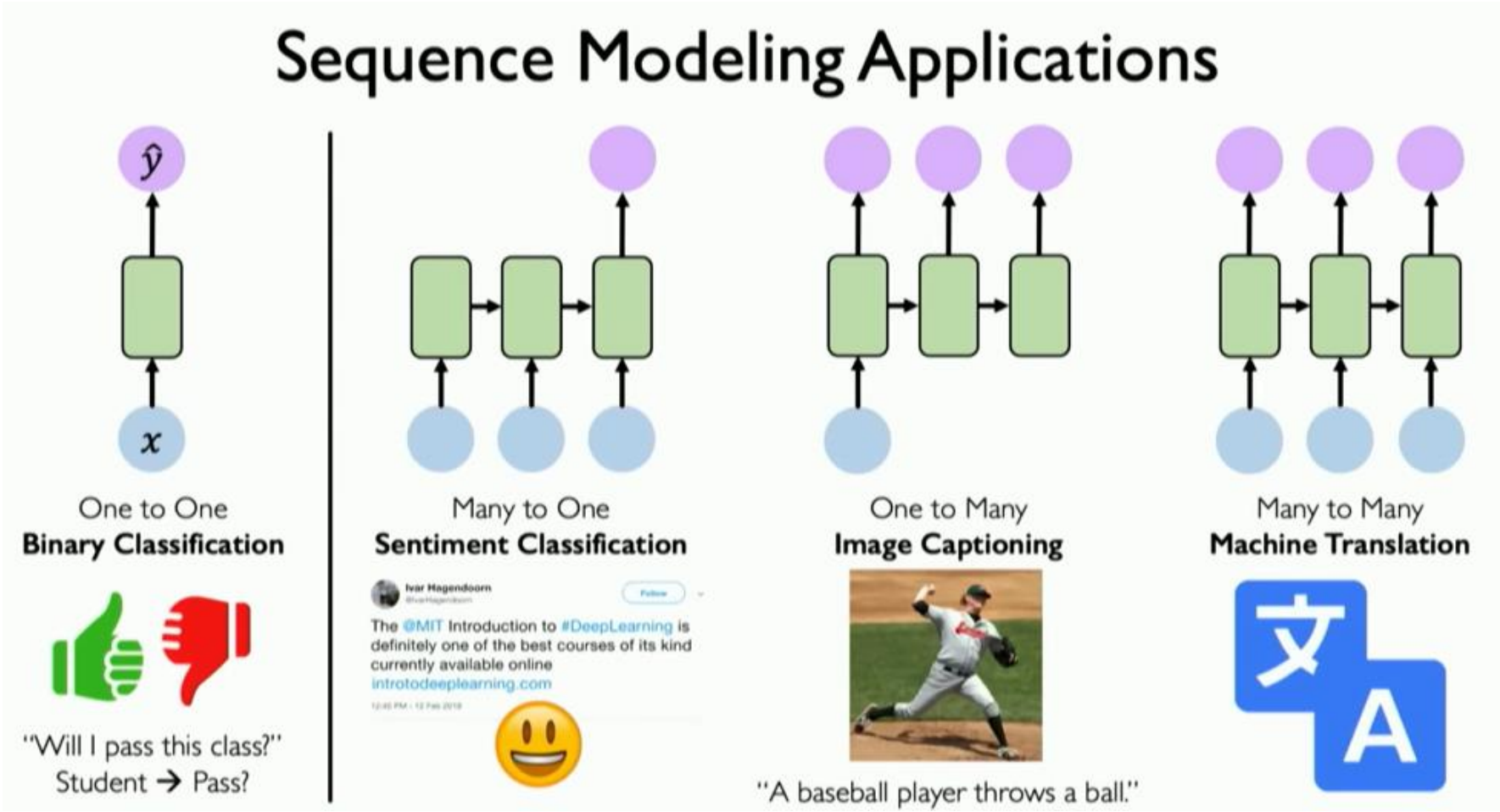

reversed kernel
image
product
convolution


1
.5
0
.5
0
-.5
0
-.5
-1

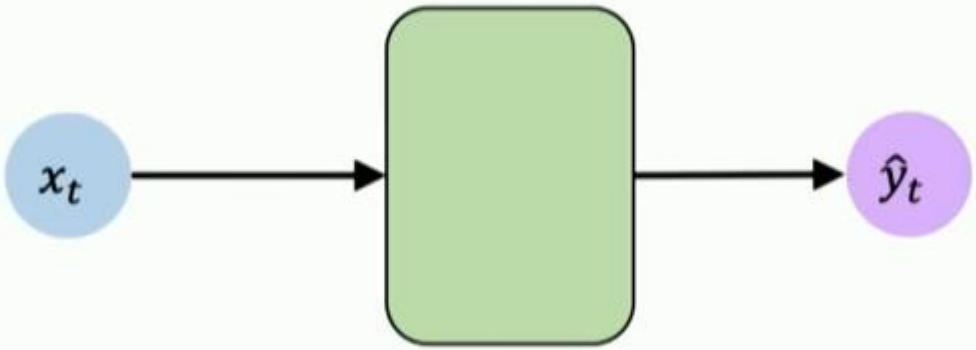

kernel
image
convolution


kernel
image
convolution

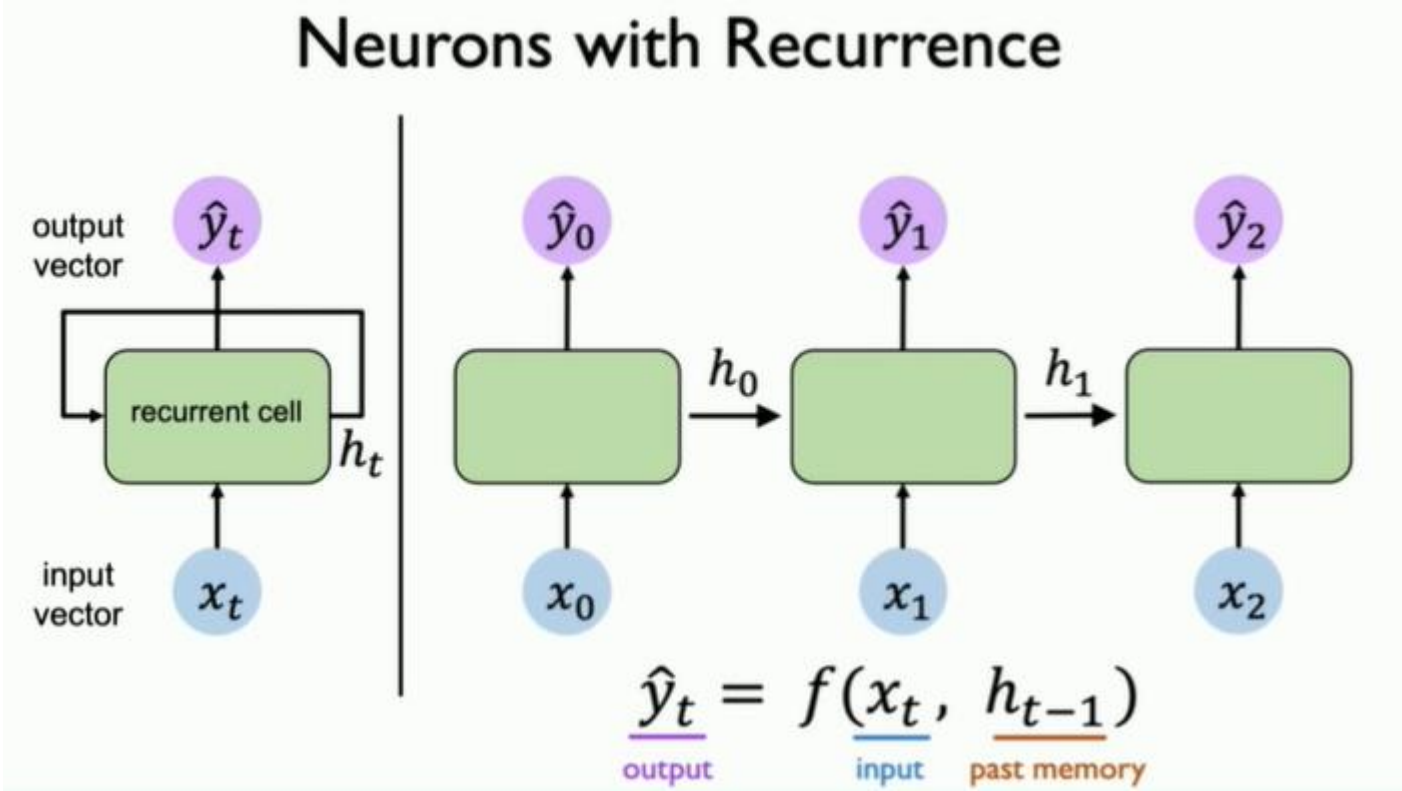
Recurrence Neural Network (RNN)



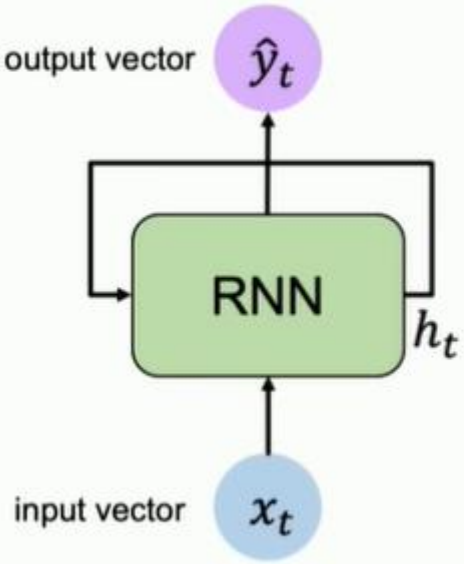
Feed-Forward Networks Revisited



$$\boldsymbol{x_t} \in \mathbb{R}^m \qquad \hat{\boldsymbol{y}}_t \in \mathbb{R}^n$$


Neurons with Recurrence
output vector
$\hat{y}_t$
recurrent cell
h_t
input vector
x_t
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
h_0
h_1
x_0
x_1
x_2
$\hat{y}_t = f(x_t, h_{t-1})$
output
input
past memory

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(x_t, h_{t-1})$$

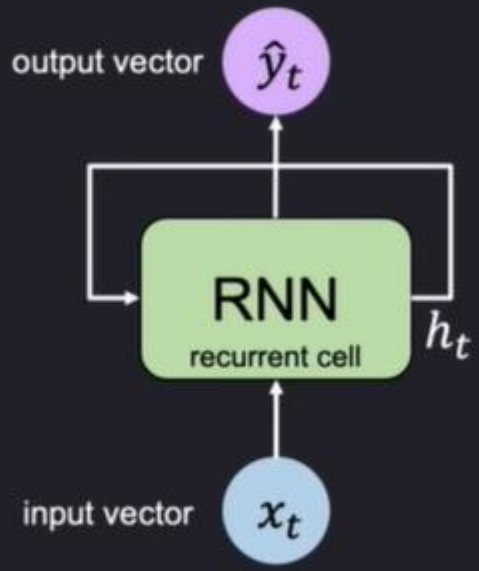
cell state | function with weights W | input | old state

Note: the same function and set of parameters are used at every time step

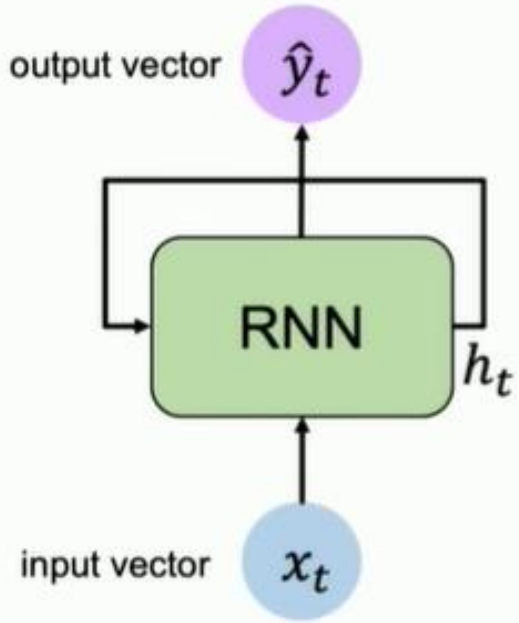
RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{\boldsymbol{hy}}^{\boldsymbol{T}} h_t$$

Update Hidden State

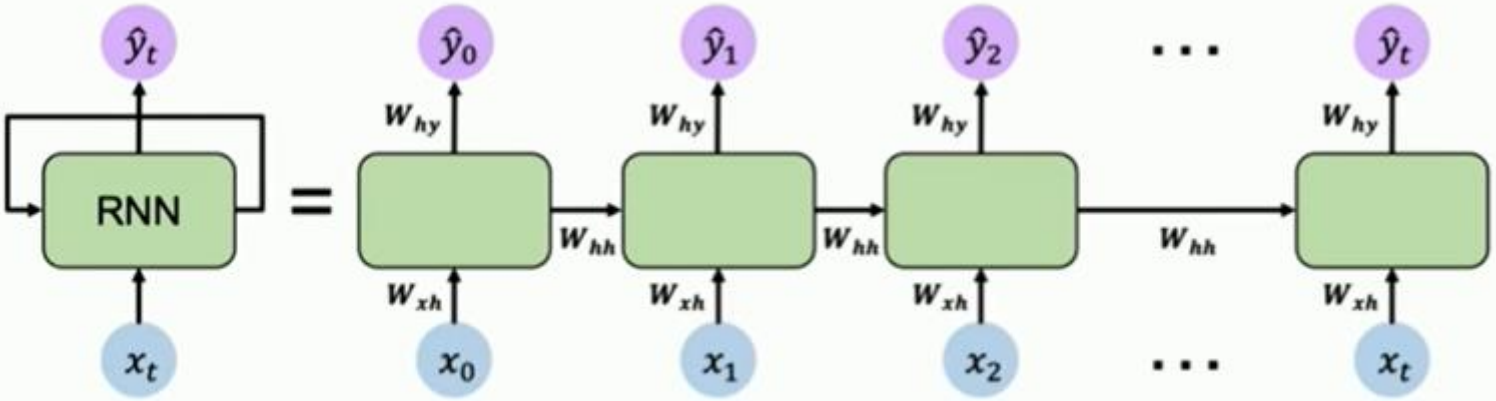
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

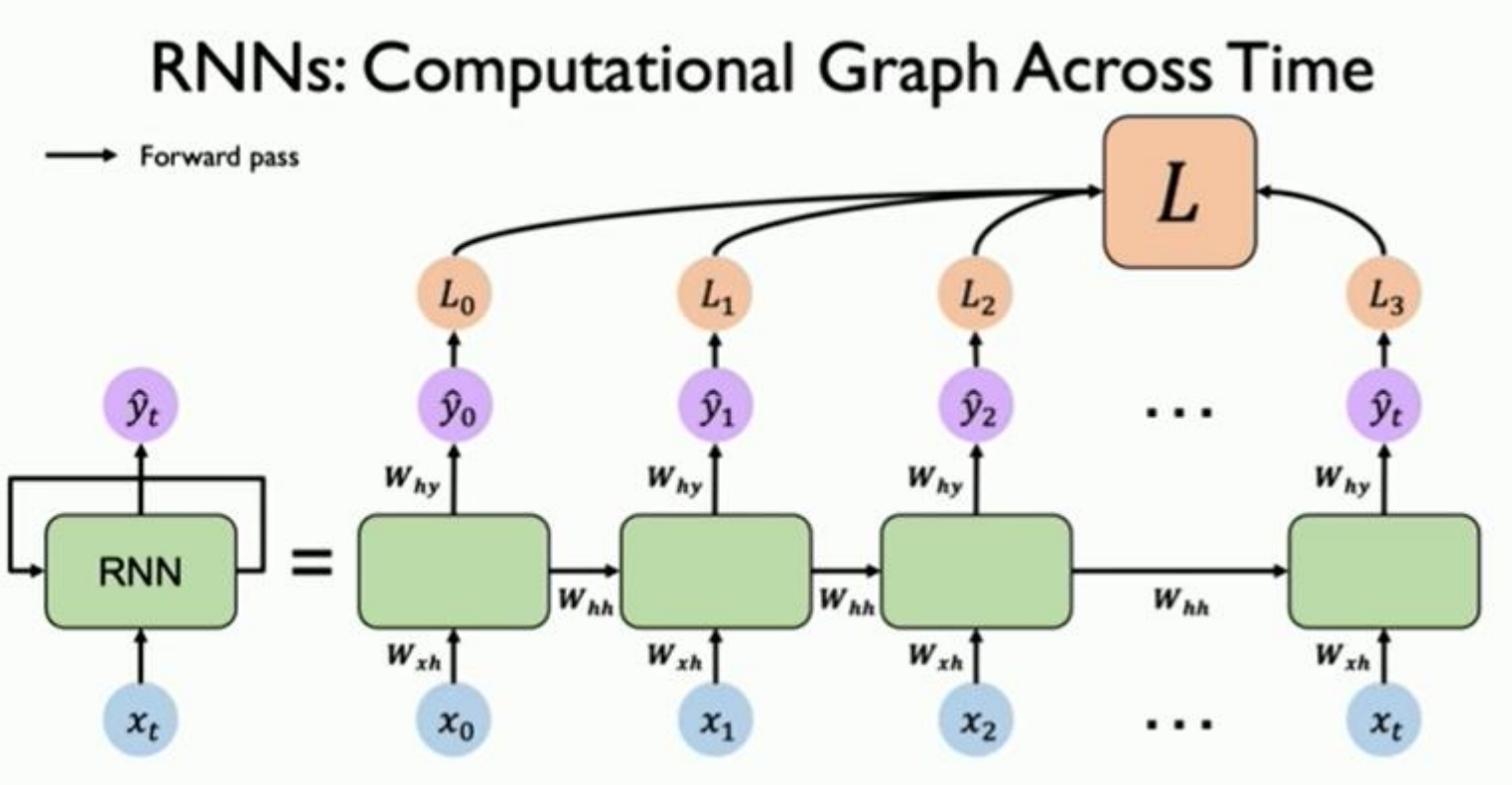
Input Vector

$$x_t$$

RNNs: Computational Graph Across Time

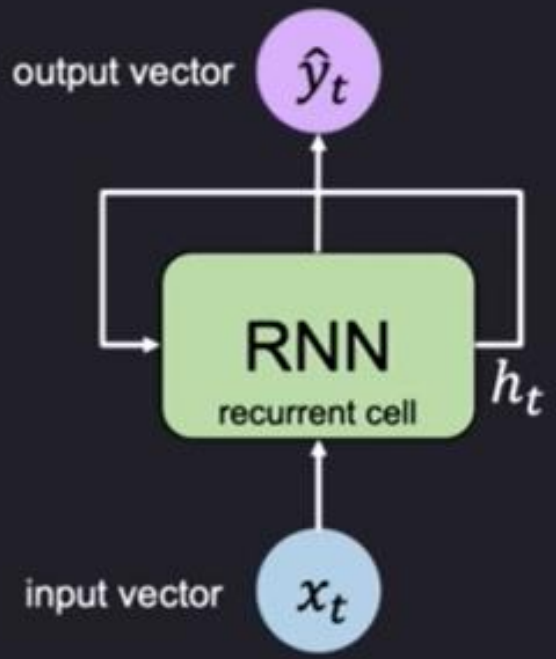
Re-use the **same weight matrices** at every time step




RNNs: Computational Graph Across Time
Forward pass
L
L_0
L_1
L_2
L_3
$\hat{y}_t$
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_t$
RNN
=
W_{hy}
W_{hh}
W_{xh}
x_t
x_0
x_1
x_2

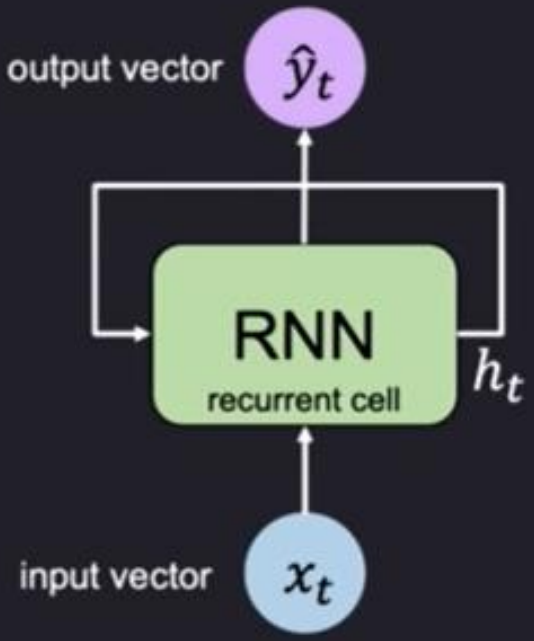
RNNs from Scratch

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```

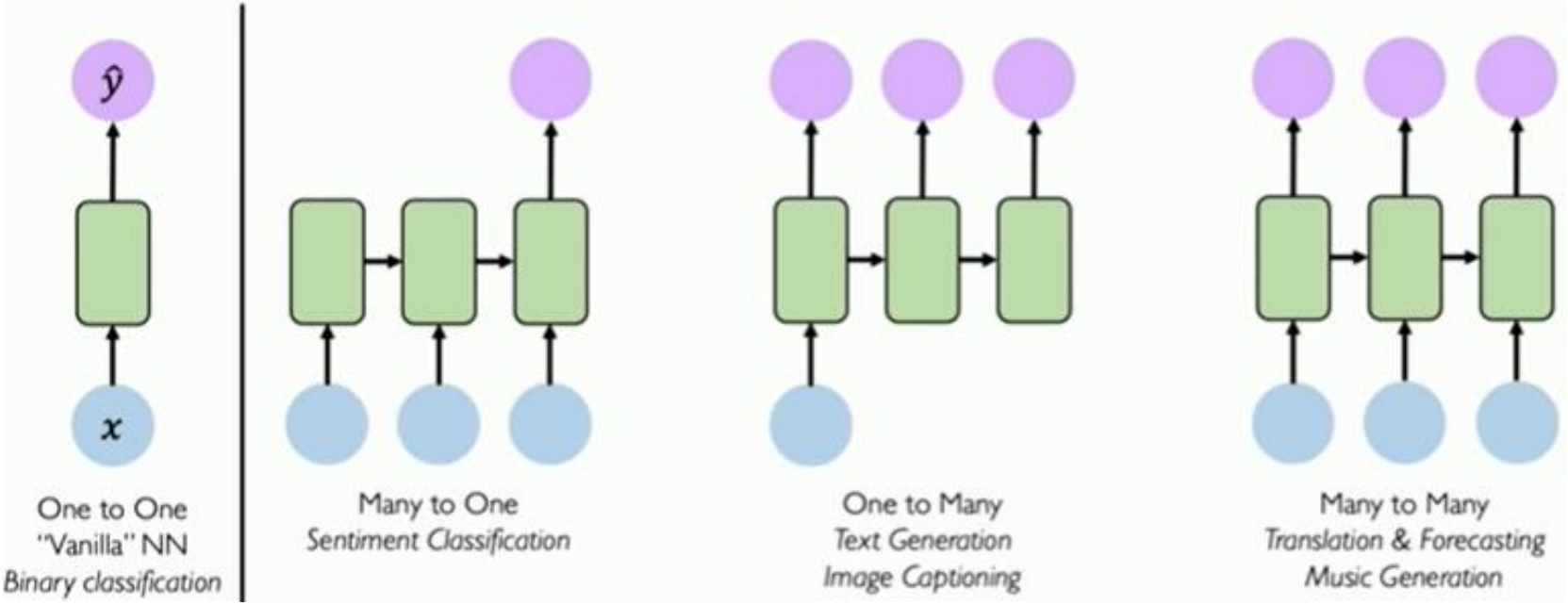


RNN Implementation in TensorFlow

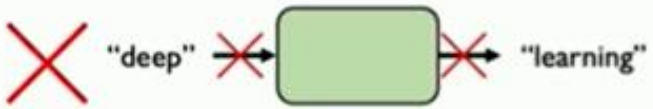
```
tf.keras.layers.SimpleRNN(rnn_units)
```



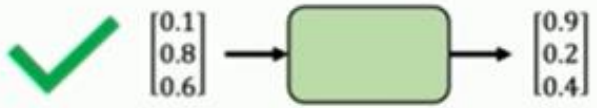
RNNs for Sequence Modeling



Encoding Language for a Neural Network

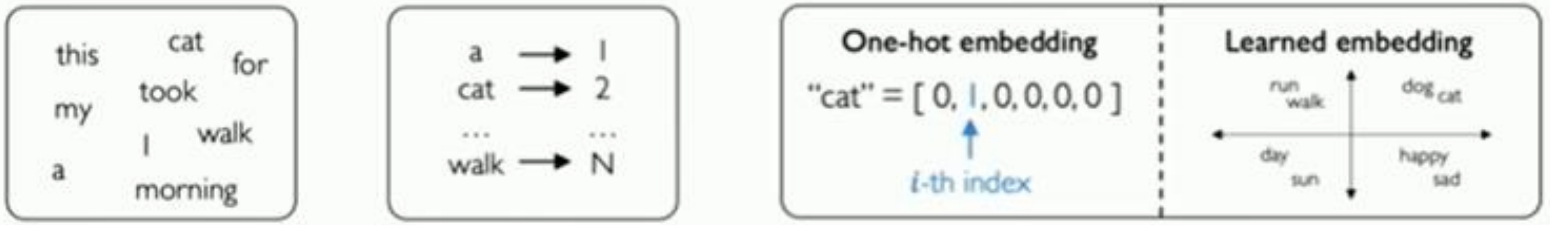


Neural networks cannot interpret words



Neural networks require numerical inputs

Embedding: transform indexes into a vector of fixed size.



1. Vocabulary: Corpus of words

2. Indexing: Word to index

3. Embedding: Index to fixed-sized vector

Handle Variable Sequence Lengths

The food was great

vs.

We visited a restaurant for lunch

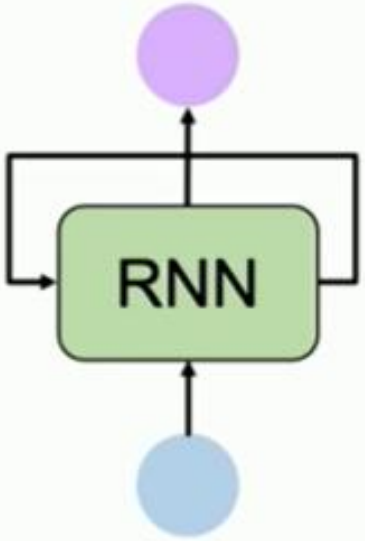
vs.

We were hungry but cleaned the house before eating

Sequence Modeling: Design Criteria

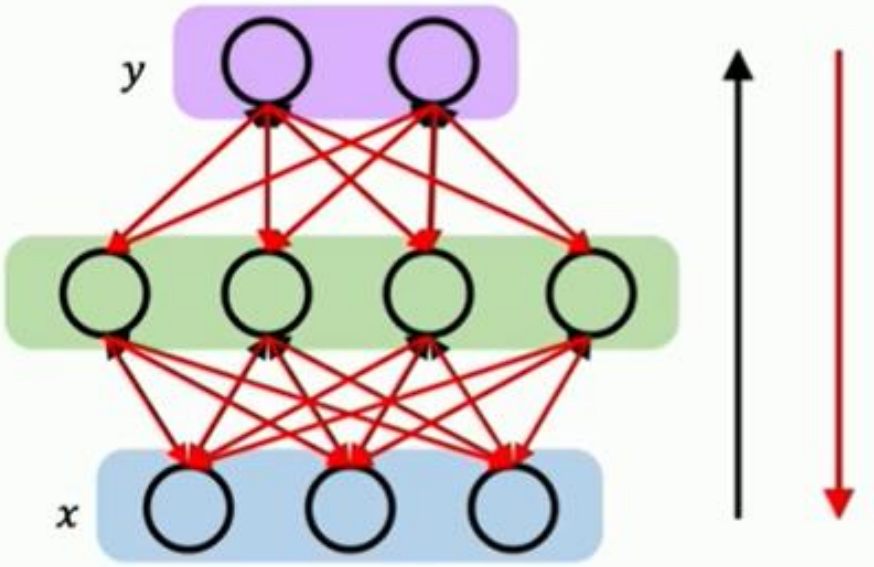
To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



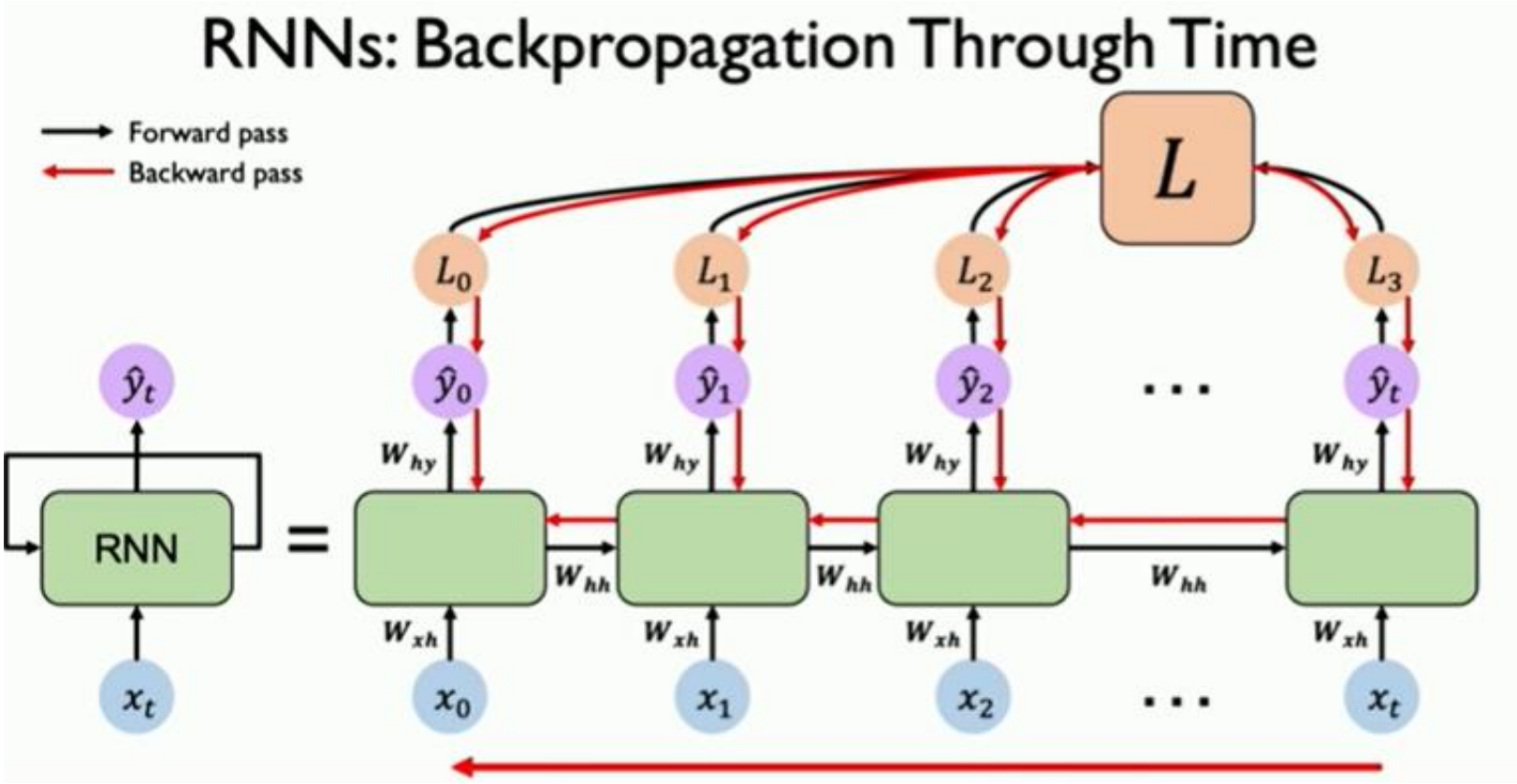
Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

Recall: Backpropagation in Feed Forward Models



Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss


RNNs: Backpropagation Through Time
Forward pass
Backward pass
L
L_0
L_1
L_2
L_3
$\hat{y}_t$
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_t$
RNN
=
W_{hy}
W_{hh}
W_{xh}
x_t
x_0
x_1
x_2
x_t

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

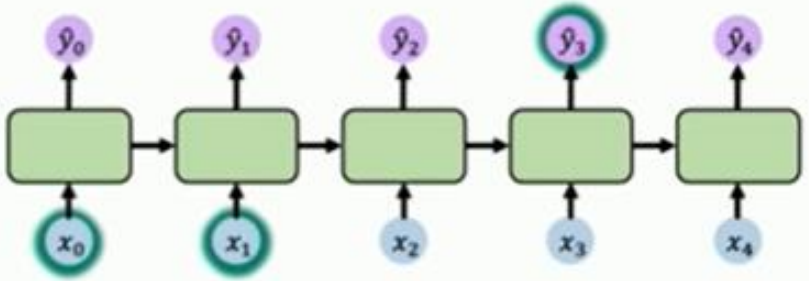
↓

Errors due to further back time steps have smaller and smaller gradients

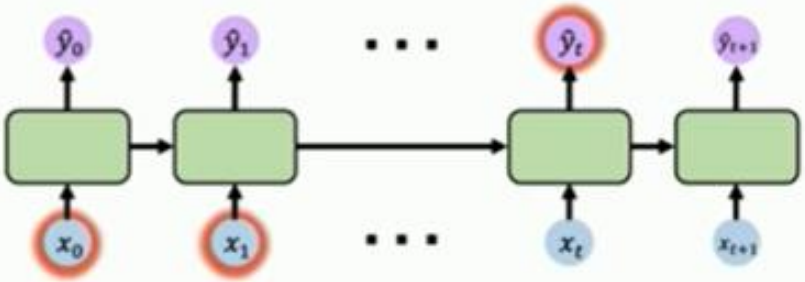
↓

Bias parameters to capture short-term dependencies

"The clouds are in the ___"



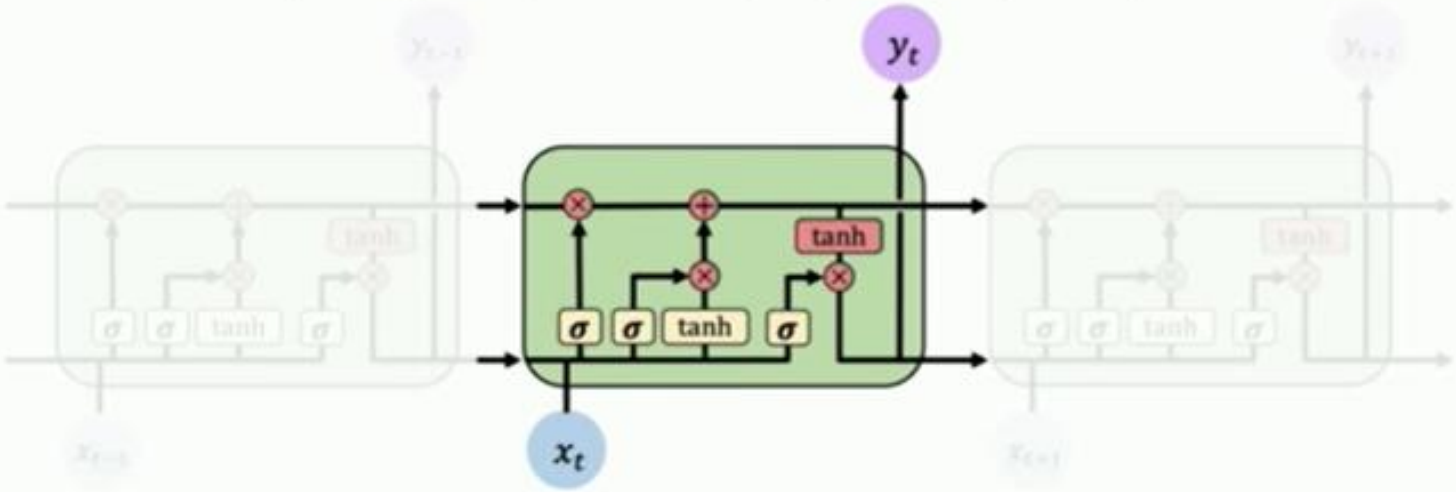
"I grew up in France, … and I speak fluent___ "



Long Short Term Memory (LSTMs)

Gated LSTM cells **control information flow:**

1) Forget 2) Store 3) Update 4) Output



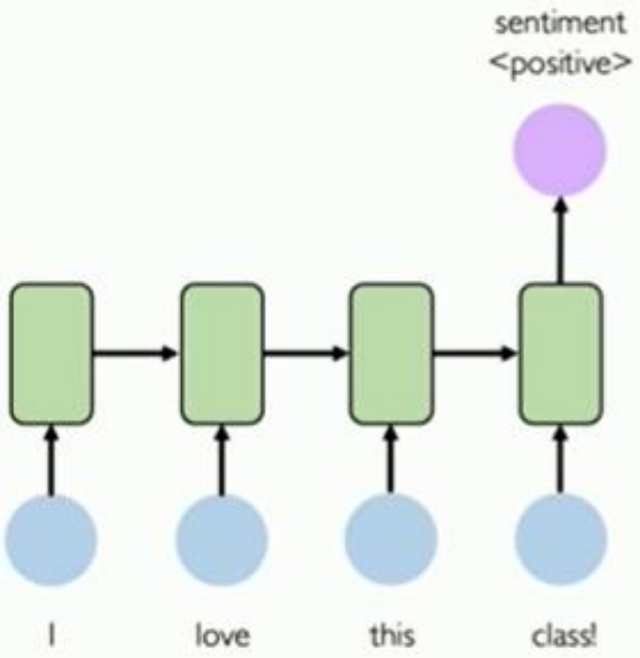
LSTM cells are able to track information throughout many timesteps

```
tf.keras.layers.LSTM(num_units)
```

LSTMs: Key Concepts

1. Maintain a **cell state**
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with partially **uninterrupted gradient flow**

Example Task: Sentiment Classification



Tweet sentiment classification

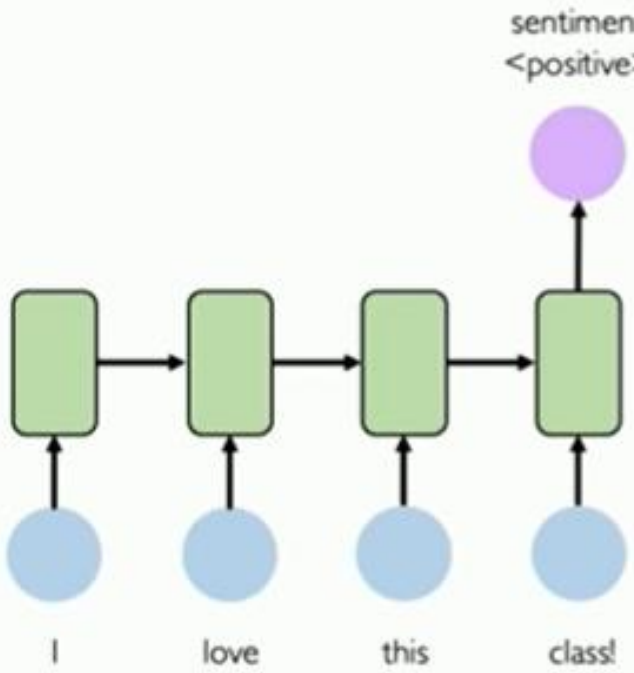
Ivar Hagendoorn

The @MIT Introduction to #DeepLearning is definitely one of the best courses of its kind currently available online introtodeeplearning.com

Angels-Cave

I wouldn't mind a bit of snow right now. We haven't had any in my bit of the Midlands this winter! :(

Limitations of Recurrent Models



Limitations of RNNs

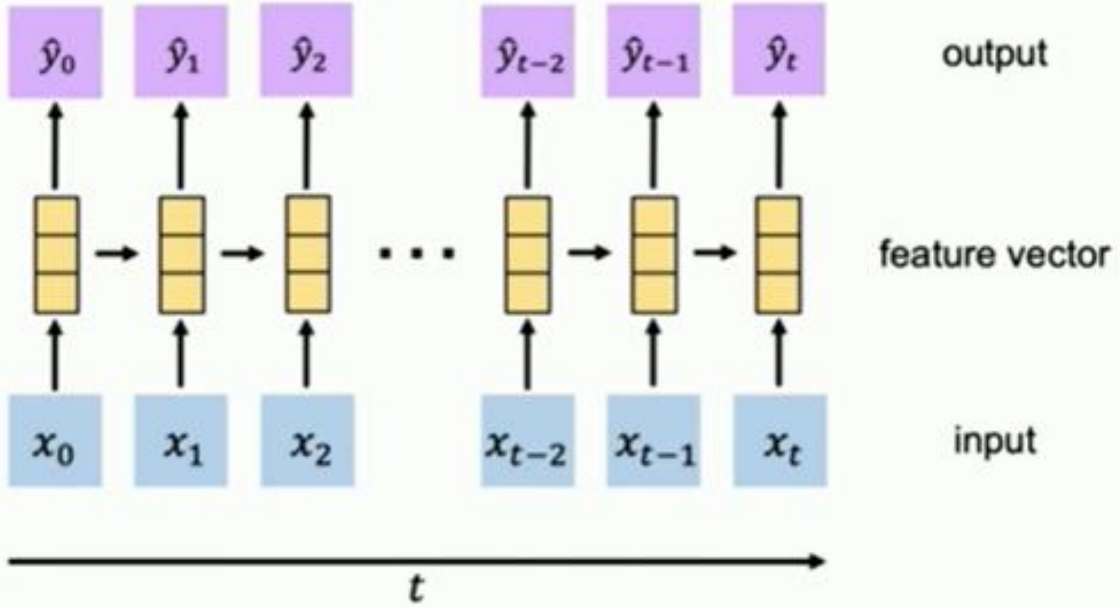
- Encoding bottleneck
- Slow, no parallelization
- Not long memory

Goal of Sequence Modeling

RNNs: recurrence to model sequence dependencies

Limitations of RNNs

- Encoding bottleneck
- Slow, no parallelization
- Not long memory



Goal of Sequence Modeling

Can we eliminate the need for recurrence entirely?

Desired Capabilities

- Continuous stream
- Parallelization
- Long memory