ECEN 685-885 - Machine Learning in Cyber-security

Dr. Mahmoud Nabil

Dr. Mahmoud Nabil
mnmahmoud@ncat.edu

North Carolina A & T State University

March 11, 2020

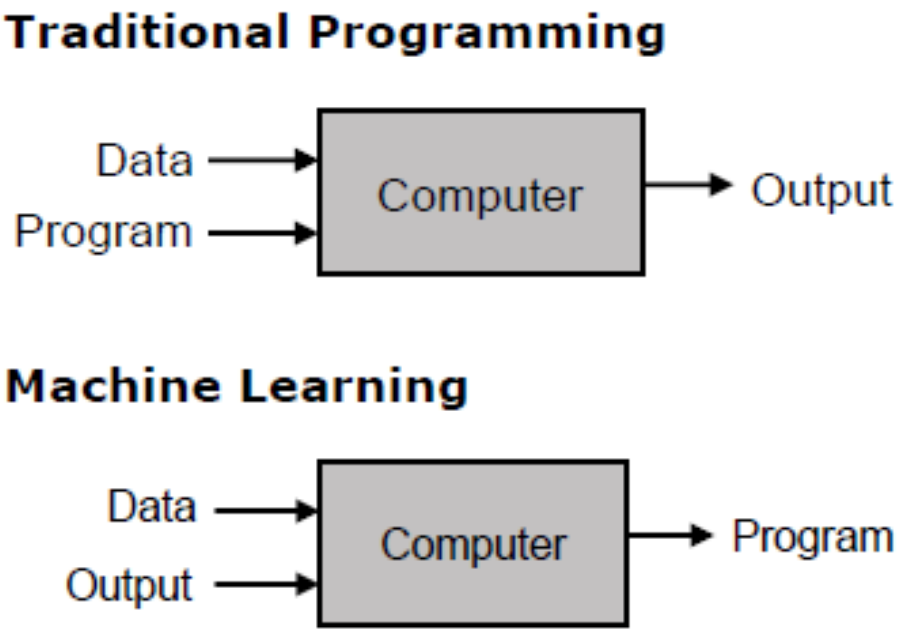
Talk Overview

1. ML inroduction
2. ML Terminologies
3. ML Applications Examples
4. Linear Regression
5. Logistic Regression
6. More Machine Learning Concepts
7. Preprocessing Data
8. Evaluation Metrics
9. Neural Networks

Outline

So What is Machine Learning?

- Automating automation
- Getting computers to program themselves
- Writing software is the bottleneck
- Let the data do the work instead!



Machine Learning Problem Types

- **Based on Type of Data**
  - Supervised, Unsupervised, Semi supervised, Reinforcement Learning
- **Based on Type of Output**
  - Regression, Classification
- **Based on Type of Model**
  - Generative, Discriminative

We will focus on: Supervised → {Regression, Classification} → Discriminative models

Types of Learning based on Type of Data

- Supervised learning
  - Training data includes desired outputs.
  - Trying to learn a relation between input data and the output
- Unsupervised learning
  - Training data does not include desired outputs.
  - Trying to understand the data.
- Semi supervised learning
  - Training data includes a few desired outputs.
- Reinforcement learning
  - Rewards from sequence of actions.

Types of Learning based on Type of Output

Regression

A regression model predicts continuous values.

For example:

- What is the value of a house in California?
- What is the probability that a user will click on this ad?

Classification

A classification model predicts discrete values.

For example:

- Is a given email message spam or not spam?
- Is this an image of a dog, a cat, or a hamster?

Types of Learning based on Type of Model

Generative Model

A Generative nodel explicitly learns the actual distribution of each class.

Discriminative Model

A Discriminative model learns the decision boundary between the classes.

Generative Models

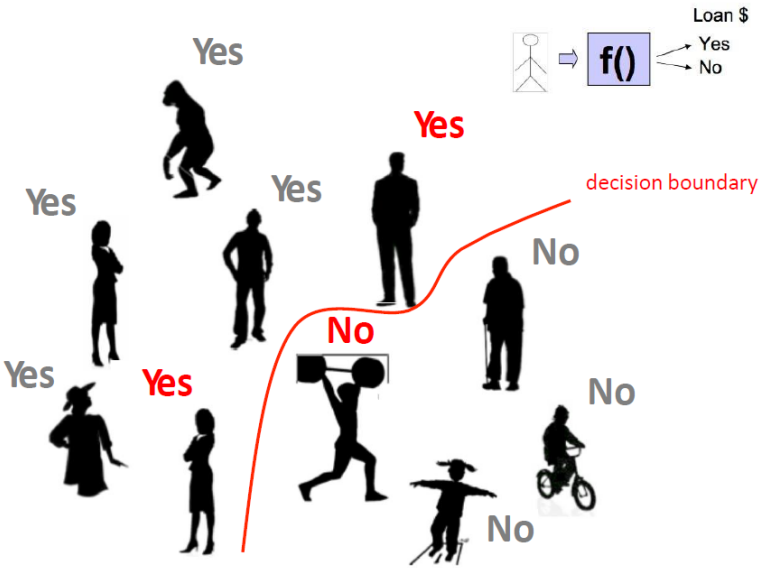
- Nave Bayes
- Hidden Markov Models
- Bayesian networks
- Markov random fields

Discriminative Models

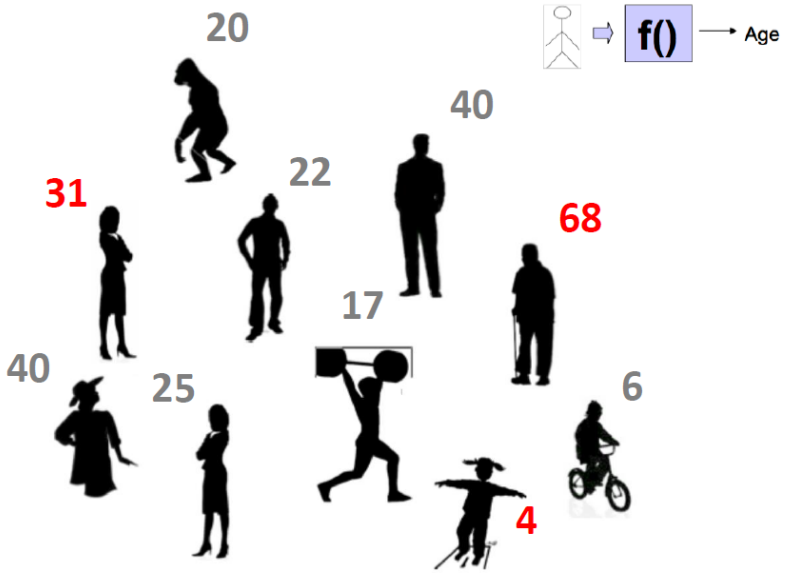
- Logistic regression
- SVMs
- Traditional neural networks
- Nearest neighbor
- Conditional Random Fields (CRF)

Regression vs Classification

Classification (supervised learning)



Regression (supervised learning)



Outline

ML Basic Terminologies

Many terminologies associated with ML. Will cover them in the following slides.

- Labels
- Features
- Examples
- Models

Label

- A label is the thing that we are predicting in classification or regression task. **Example:** Male, Female
- The label could also be the future price of wheat, the kind of animal shown in a picture, the meaning of an audio clip, or just about anything.
- Usually denoted with the variable y.

Features (1/2)

- A feature is an input variable (a.k.a attribute).
- A simple machine learning project might use a single feature, while a more sophisticated machine learning project could use millions of features.
- Usually denoted as:

$$x_1, x_2, \ldots, x_N$$

In the spam detector example, the features could include the following:

- words in the email text
- sender's address
- time of day the email was sent
- ...

Features (2/2)

Generally three types of attributes:

- **Categorical:** red, blue, brown, yellow
- **Ordinal:** poor, satisfactory, good, excellent
- **Numeric:** 3.14, 6E23, 0,

Categorical

- No natural ordering to categories
- Categories usually encoded as numbers

Ordinal

- There is a natural ordering to categories
- Encoded as numbers to preserve ordering

Numeric

- Integers or real numbers
- meaningful to add, mul tiply, compute

Notethat

The process of generating this features for our machine learning problem is called feature engineering.

Data samples (Examples)

Data sample / Example is a particular instance of data, **x**. (Note That. **x** is a vector of features)

We break examples into two categories:

- Labeled examples: (Used for prediction)
- Unlabeled examples: (Used for inference/testing)

Example

housingMedianAge (feature)	totalRooms (feature)	totalBedrooms (feature)	medianHouseValue (label)
15	5612	1283	66900
19	7650	1901	80100
17	720	174	85700
14	1501	337	73400
20	1454	326	65500

Model

A **model** defines a relationship between features and label.

Two phases of a model's life:

- **Training** means creating or learning the model. You show the model labeled examples and enable the model to gradually learn the relationships between features and label.
- **Testing/Inference** means applying the trained model to unlabeled examples. You use the trained model to make useful predictions (y').

Outline

ML Application 1: Credit Approval

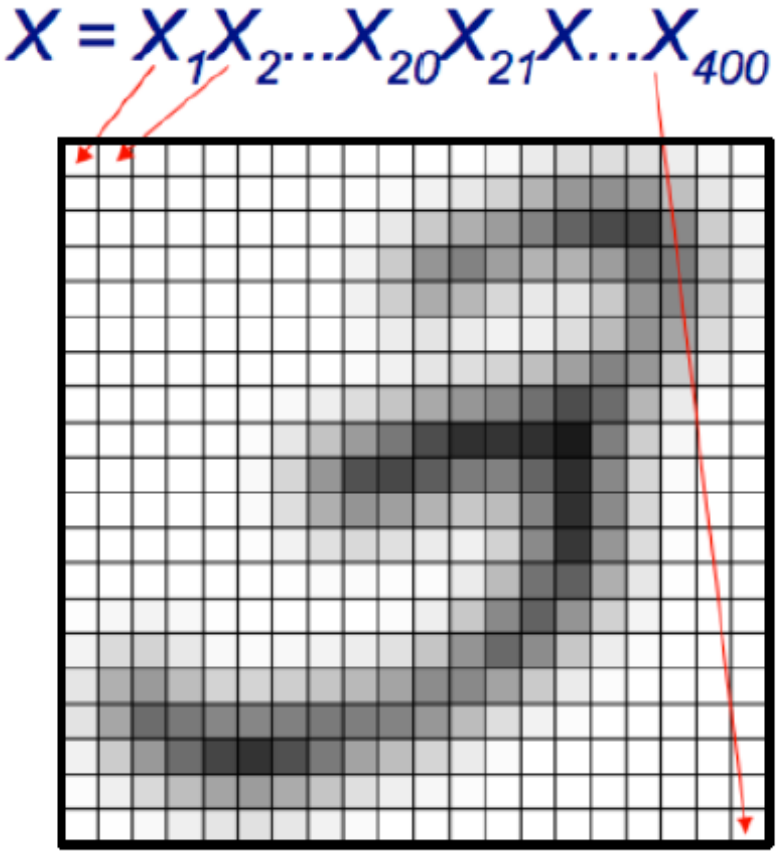
- **Numeric Features:**
  - loan amount (e. g. $1000)
  - Income (e. g. $65000)
- **Ordinal Features:**
  - savings: {none, <100, 100..500, 500..1000, >1000}
  - employed: {unemployed, <1yr, 1..4yrs, 4..7yrs, >7yrs}
- **Categorical Features:**
  - purpose: {car, appliance, repairs, education, business}
  - personal: {single, married, divorced, separated}
- **Labels (Categorical):**
  - Approve credit application
  - Disapprove credit application

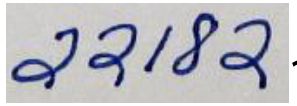
Easy feature engineering process.

ML Application 2: Handwritten Digits Recognation

Represent each pixel as a separate attribute either Categorical **OR** Ordinal:

- **Categorical Features:**
  - (white) or (black) based on a threshold
- **Ordinal Features:**
  - Degree of "blackness" of a pixel
- **Labels (Categorical):** {0,1,2,3,4,5,6,7,8,9}



What if we are dealing with paper like this  ?

Isolate each digit, rescale, de-slant, ..

Hard feature engineering process.

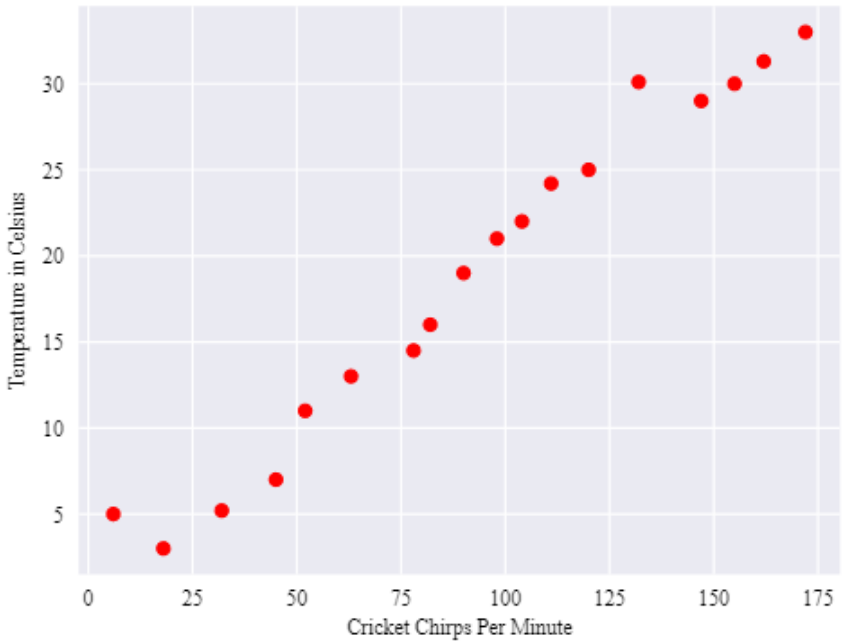
Outline

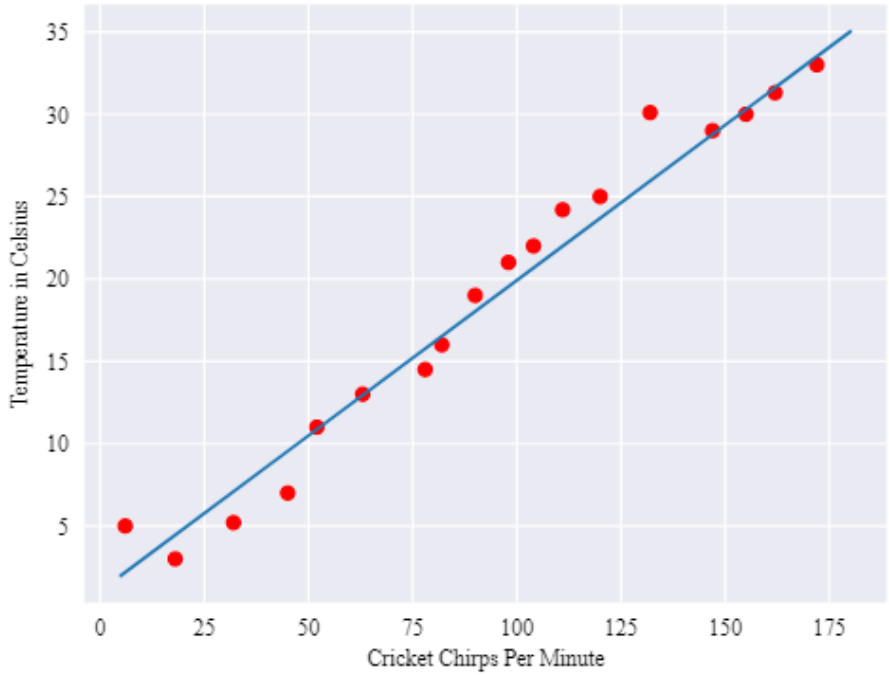
Linear Regression

Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data

Example: Scientists found that crickets (an insect species) chirp more frequently on hotter days than on cooler days.





Linear Regression

A linear relationship

- True, the line doesn't pass through every dot.
- However, the line does clearly show the relationship between chirps and temperature.

$$y = mx + b$$

where:

- y: is the temperature in Celsiusthe value we're trying to predict.
- m: is the slope of the line.
- x: is the number of chirps per minutethe value of our input feature.
- b: is the y-intercept.

Linear Regression

In machine learning, we'll write the equation for a model slightly differently:

$$y' = w_1 x_1 + w_0$$

where:

- y': is the predicted label (a desired output).
- w_1: is the weight of feature 1. Weight is the same concept as the "slope".
- x_1: is feature 1.
- w_0 or b: is the bias (the y-intercept).

Notethat

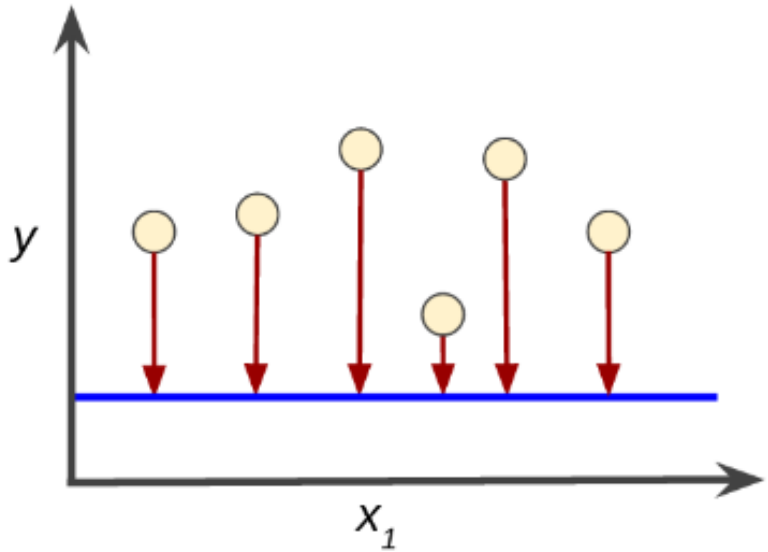
A model that relies on three features might look as follows:

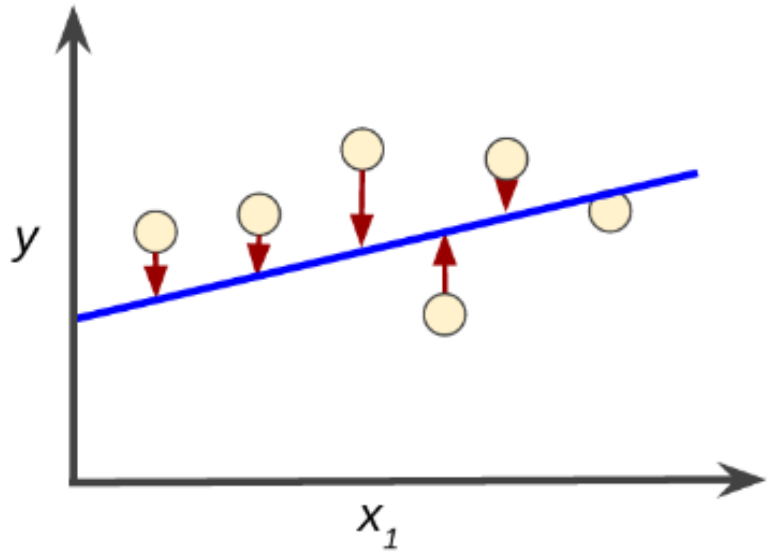
$$y' = w_3 x_3 + w_2 x_2 + w_1 x_1 + w_0$$

Training and Loss

- **Training** a model simply means learning (determining) good values for all the weights and the bias from labeled examples.
- **Loss** is the penalty for a bad prediction.
  - Perfect prediction means the loss is zero
  - Bad model have large loss.
- Suppose we selected the following weights and biases.

Which of them have lower loss?





Squared loss

- The linear regression models use a popular loss function called squared loss.
- Also known as L_2.
- Is represented as follows:

$$[obsevation(x) - prediction(x)]^2$$
$$= (y - y')^2$$

Why squared loss?
Can we do absolute loss?

Mean square error (MSE)

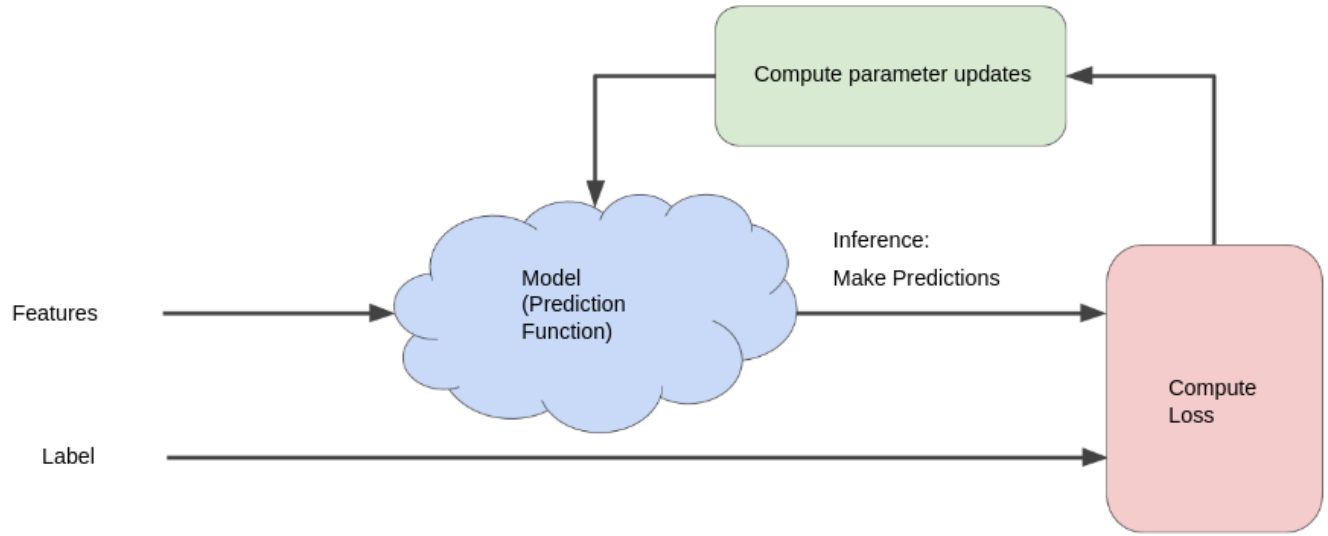
- Is the average squared loss per example over the whole dataset.

$$\text{MSE} = \frac{1}{N} \sum_{(x,y)\in D} (y - \textit{prediction}(x))^2$$

- (x,y) is an example in which
  - y is the label
  - x is a feature
- prediction(x) is equal $y' = w_1 x + w_0$
- D is the dataset that contains all (x,y) pairs
- N is the number of samples in D

Reducing Loss

- **Training** is a feedback process that use the loss function to improve the model parameters.
- The **training** is an iterative process.



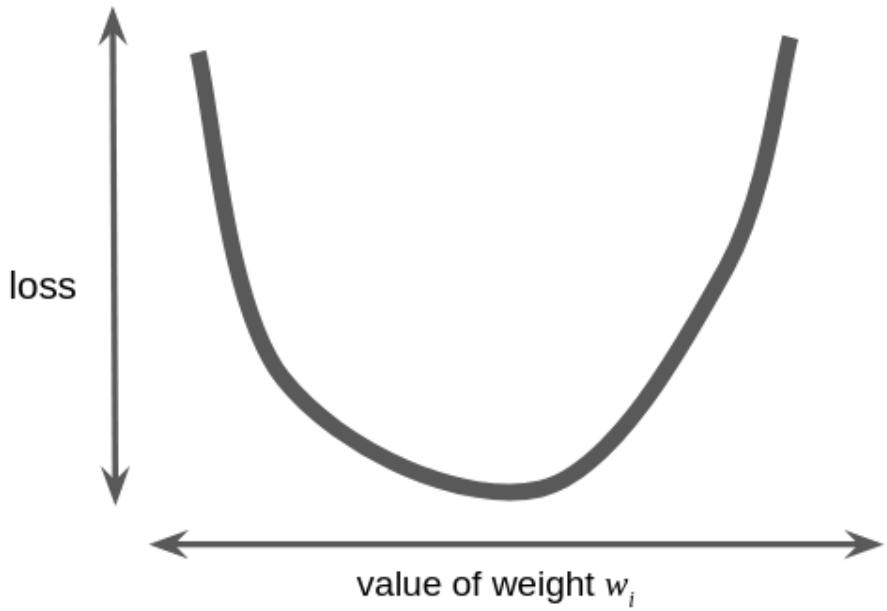
Two Questions

- What initial values should we set for w_1 and w_0?
- How to update w_1 and w_0?

Gradient Descent (1/3)

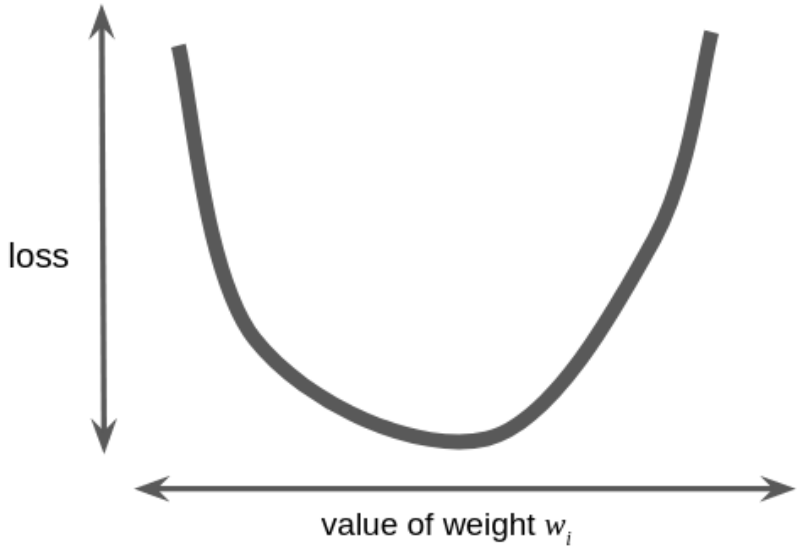
- Assume (for symplicity) we are only concerned with finding w_1.
- Assume we had the time and the computing resources to calculate the loss for all possible values of w_1.

Regression problems yield convex loss vs. weight plots.



Gradient Descent (2/3)

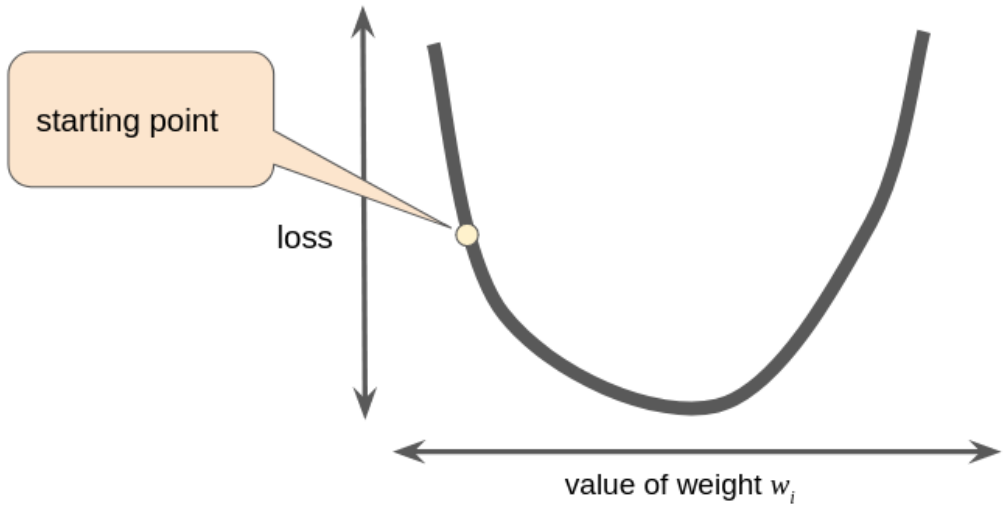
- Gradient descent enables you to find the optimal w without computing for all possible values.
- Gradient descent has the following steps
  1. Pick a random starting point for w
  2. Calculates the gradient of the loss curve at w.
  3. Update w
  4. go to 2, till convergence



Gradient Descent (3/3)

Note that a gradient is a vector, so it has both of the following characteristics:

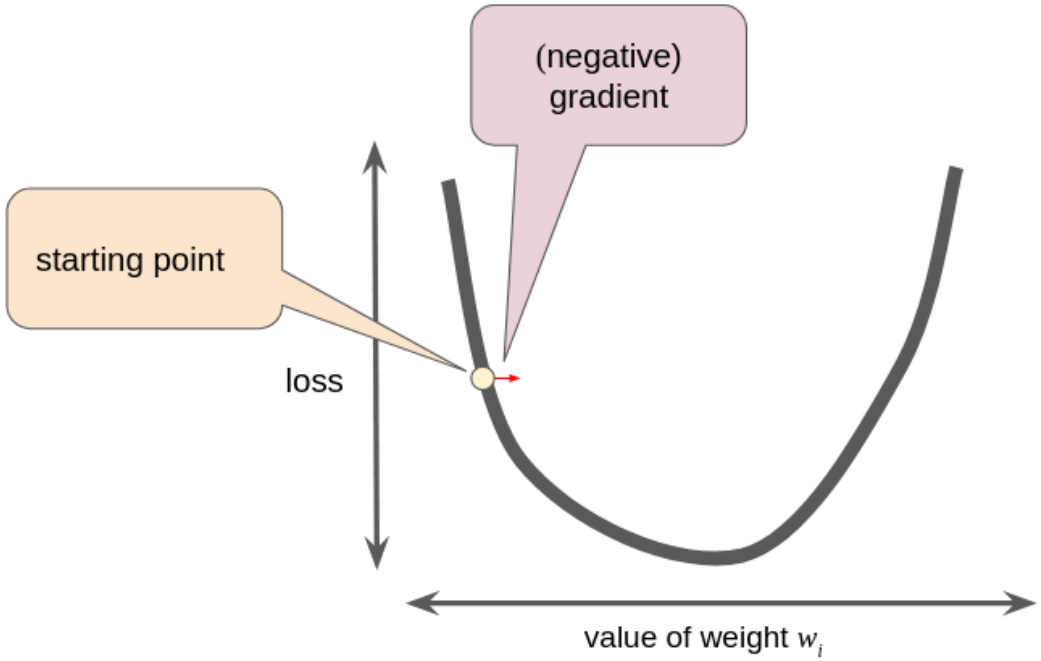
- Magnitude
- Direction



$$w_{new} = w_{old} - \eta * \frac{d\ loss}{dw}$$

Gradient Descent (3/3)

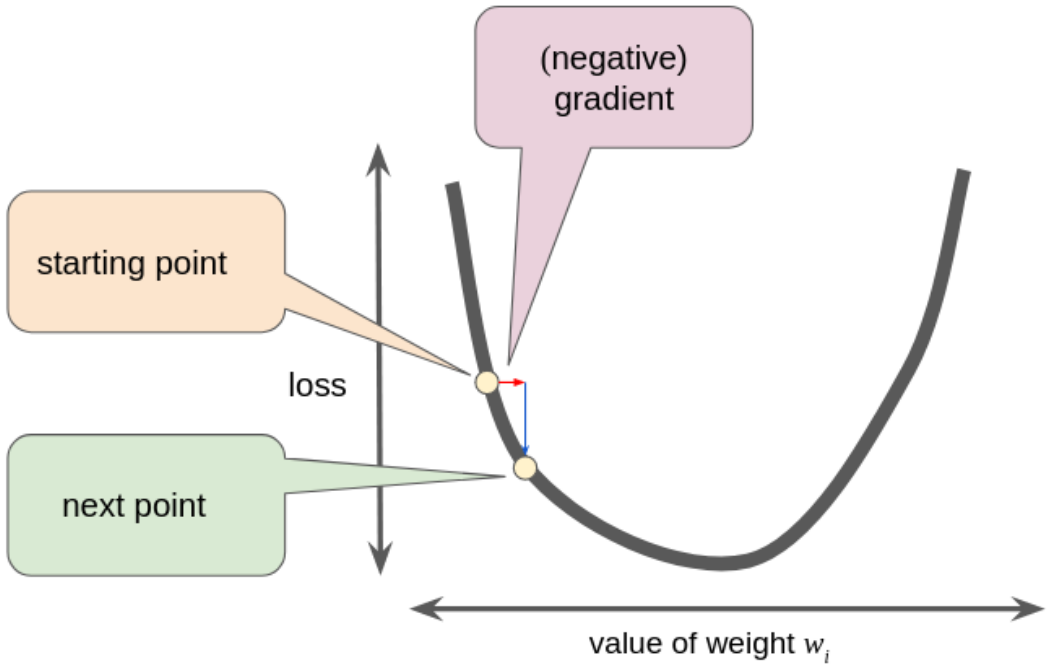
The gradient descent algorithm takes a step in the direction of the negative gradient



$$w_{new} = w_{old} - \eta * \frac{d\ loss}{dw}$$

Gradient Descent (3/3)

the gradient descent algorithm adds **some fraction** of the gradient's magnitude **(Learning Rate η)** to the previous point



$$w_{new} = w_{old} - \eta * \frac{d\ loss}{dw}$$

Convergence Criteria

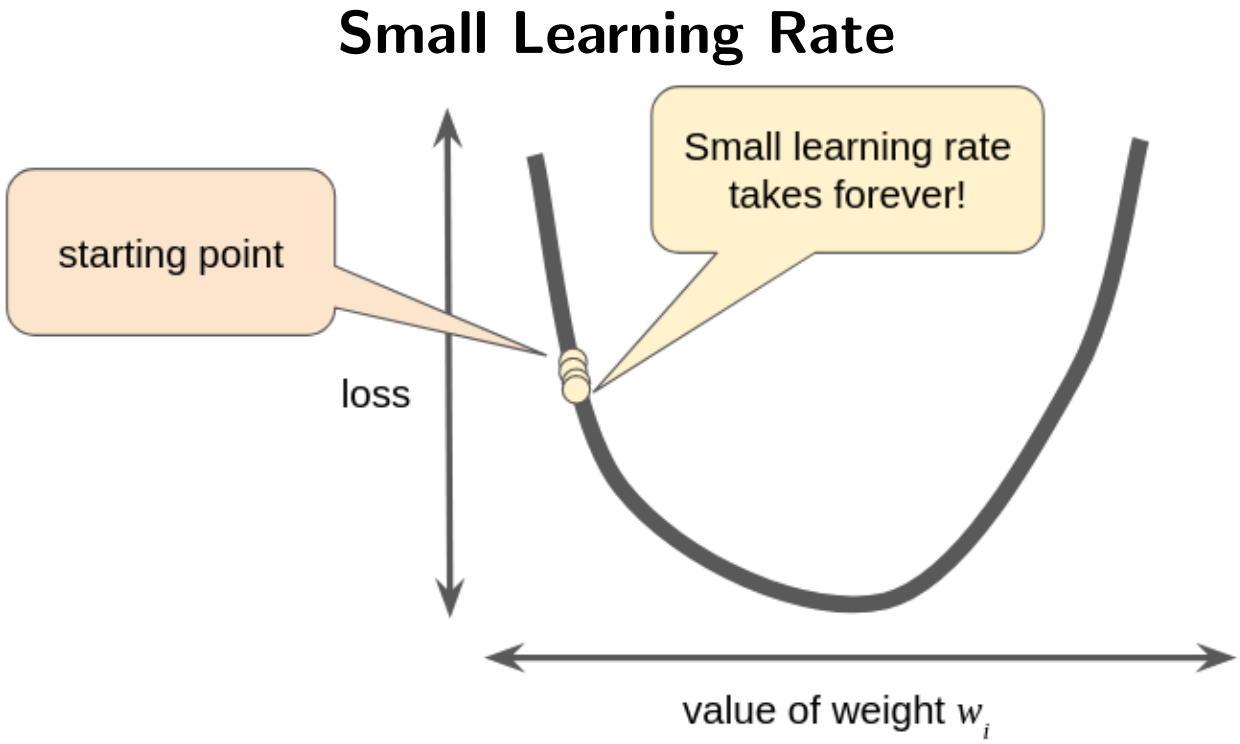
- For convex functions, optimum occurs when
  - $|\frac{d\ loss}{dw}| = 0$
- In practice, stop when
  - $|\frac{d\ loss}{dw}| \leq \epsilon$

Learning rate

- Gradient descent algorithms multiply the gradient by a scalar known as the learning rate (also sometimes called step size) .
- How can we choose the learning rate?

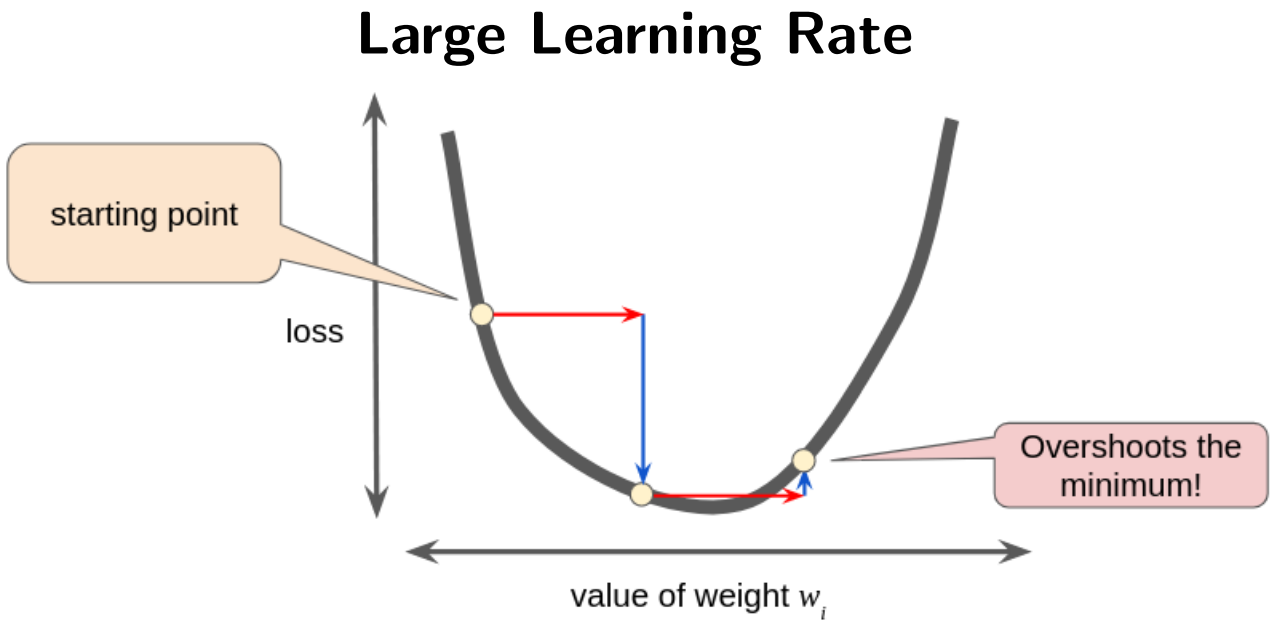
Learning rate

- Gradient descent algorithms multiply the gradient by a scalar known as the learning rate (also sometimes called step size) .
- How can we choose the learning rate?



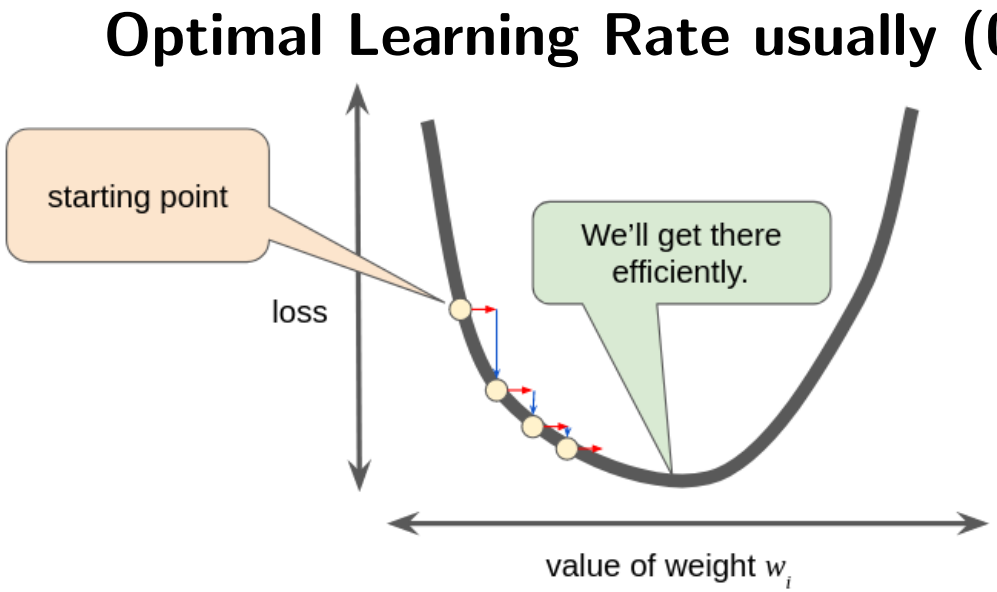
Learning rate

- Gradient descent algorithms multiply the gradient by a scalar known as the learning rate (also sometimes called step size) .
- How can we choose the learning rate?



Learning rate

- Gradient descent algorithms multiply the gradient by a scalar known as the learning rate (also sometimes called step size) .
- How can we choose the learning rate?



Generalization and Gradient

- For n features: $y' = \sum_{i=0}^{i=n} w_i x_i$
- Note w_0 is the bias (intercept), and $x_0 = 1$.
- vector representation y' $= \mathbf{w}^T \mathbf{x}$
- Loss $= \ell = (y - y')^2$
- Gradient derivation

$$\frac{d\ell}{dw_i} = \frac{d\ell}{dy'} \frac{dy'}{dw_i}$$
$$= [2(y - y') * x_i * (-1)]$$

Types of Gradient Descents

- **Batch Gradient Descent:**
  - MSE loss assumes taking gradient for the total number of samples in the data set
  - Data sets often contain billions or even hundreds of billions of examples
  - Can take a very long time to compute.
- **Stochastic Gradient Descent (SGD):**
  - Uses only a single example (a batch size of 1) per iteration.
  - Very noisy.
- **Mini-Batch Gradient Descent:**
  - Compromise between full-batch iteration and SGD
  - Typically a batch of size between 10 and 1,000 examples, chosen at random.

Outline

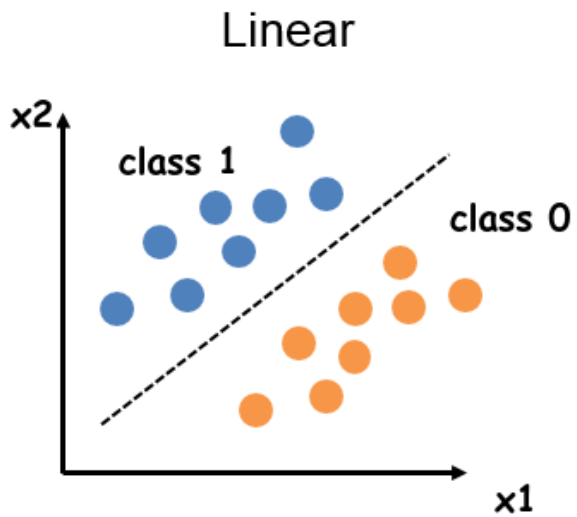
Logistic Regression Introduction

Logistic Regression

It is a statistical model used for binary classification. The inputs are the features values and the output (y) is a probability from 0 to 1.

Note that

- Logistic regression is a linear classifier.
- The equation of the decesion boundry : $0 = w_2x_2 + w_1x_1 + w_0$
- Class 0 condition:
  $0 < w_2x_2 + w_1x_1 + w_0$
- Class 1 condition:
  $0 > w_2x_2 + w_1x_1 + w_0$

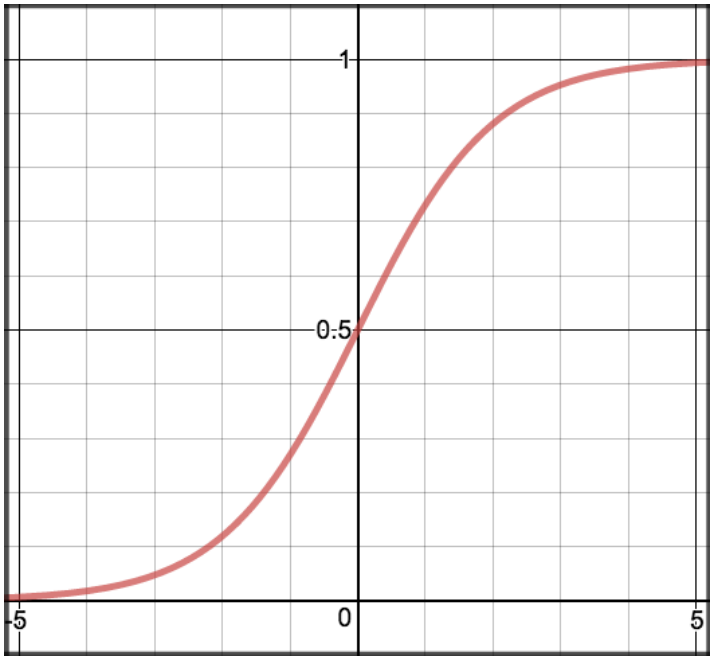


How get y' as probability given these conditions?

Sigmoid Function

- In order to map predicted values to probabilities, we use the sigmoid function.

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

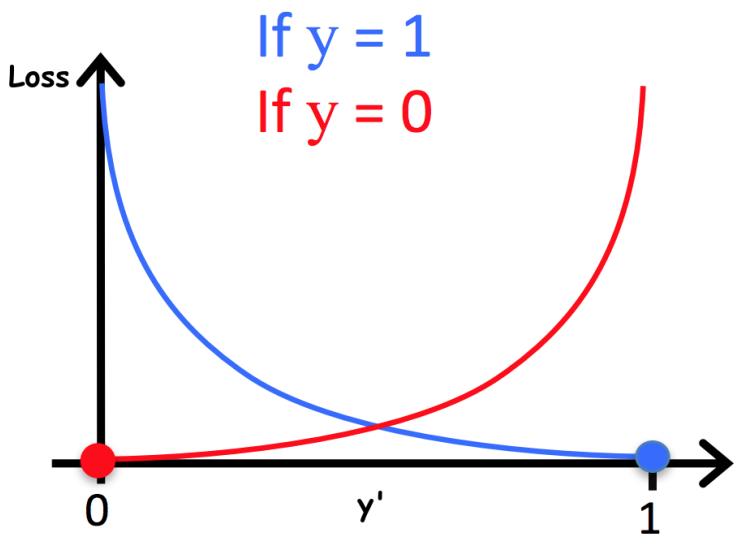


Logistic Regression

- We can set decesion boundary $z = w_2x_2 + w_1x_1 + w_0$
- Then $y' = \sigma(z) = \frac{1}{1+e^{-z}}$
- What if point (x_1, x_2) is below the decesion boundary?
- What if point (x_1, x_2) is above the decesion boundary?
- $\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$

Logistic Loss Function

- Since y' in logistic regression is a probability between 0 and 1.
- Our loss can be defined with the following loss function.
  - if y = 1 : Loss = -log(y')
  - if y = 0 : Loss = -log(1-y')



$$\text{Loss} = \ell = -y\log(y') - (1-y)\log(1-y')$$

Now we can train with gradient descent to find the weights

Generalization and Gradient

- For n features: $z = \sum_{i=0}^{i=n} w_i x_i$, (w_0 is the bias)
- vector representation $z = \mathbf{w}^T \mathbf{x}$
- $y = sigmoid(z) = \sigma(z)$
- $\ell = -ylog(y') - (1-y)log(1-y')$

Gradient Derivation

$$\frac{d\ell}{dw_i} = \frac{d\ell}{dy'}\frac{dy'}{dw_i} = \frac{d\ell}{dy'}\frac{dy'}{dz}\frac{dz}{dw_i}$$

$$= \underbrace{\left[\frac{-y}{\sigma(z)} + \frac{1-y}{1-\sigma(z)}\right]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}}$$

$$= \underbrace{\left[\frac{-y(1-\sigma(z)) + (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}}$$

$$= \underbrace{\left[\frac{-y(1-\sigma(z)) + (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}}$$

$$= (\sigma(z) - y) * x_i$$

Summary Linear vs Logistic Regression

Linear Regression

- $y' = \mathbf{w}^T \mathbf{x}$
- $\ell = (y - y')^2$
- $\frac{d\ell}{dw_i} = [2(y - y') * x_i * (-1)]$

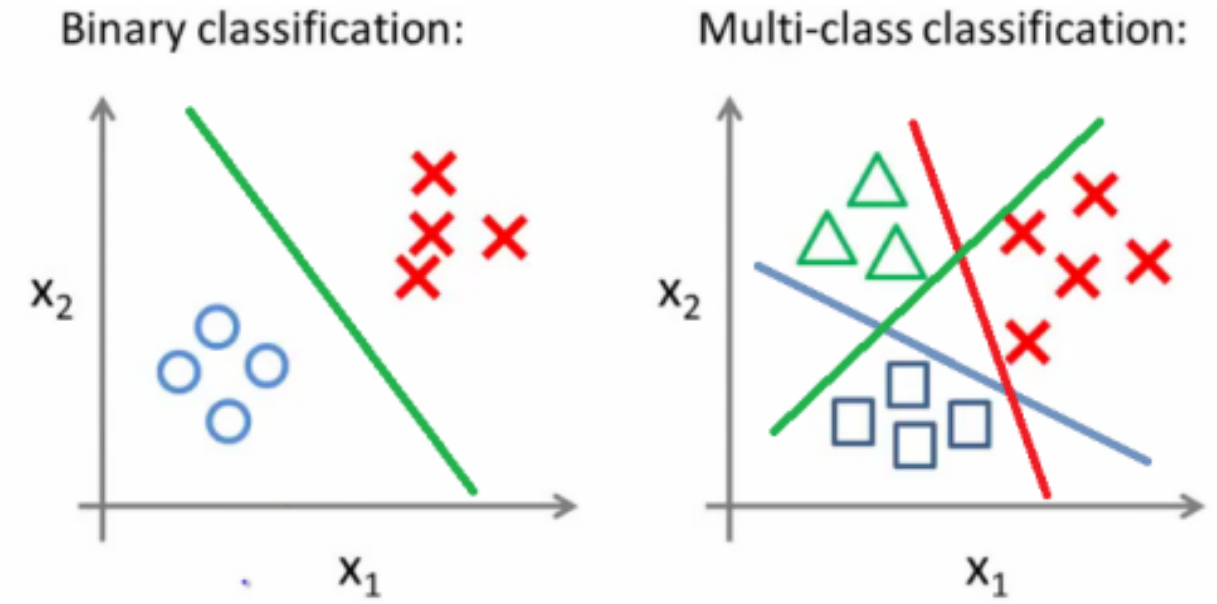
Logistic Regression

- $z = \mathbf{w}^T \mathbf{x}$
- $y' = sigmoid(z) = \frac{1}{1+e^{-z}}$
- $\ell =$ -ylog(y')-(1-y)log(1-y')
- $\frac{d\ell}{dw_i} = (\sigma(z) - y) * x_i$

Loss is convex function interms of the weights (y' is function of the weights)

Outline

Multiclass Classification (One vs All)



For Three classes

$$\text{Result Class} = \underset{k \in \{1,2,3\}}{\text{argmax}}\ f_k(x)$$

From Linear to non-Linear (Feature Crosses)

- Our basic equation for one feature linear regression is

$$y' = w_1 x_1 + w_0$$

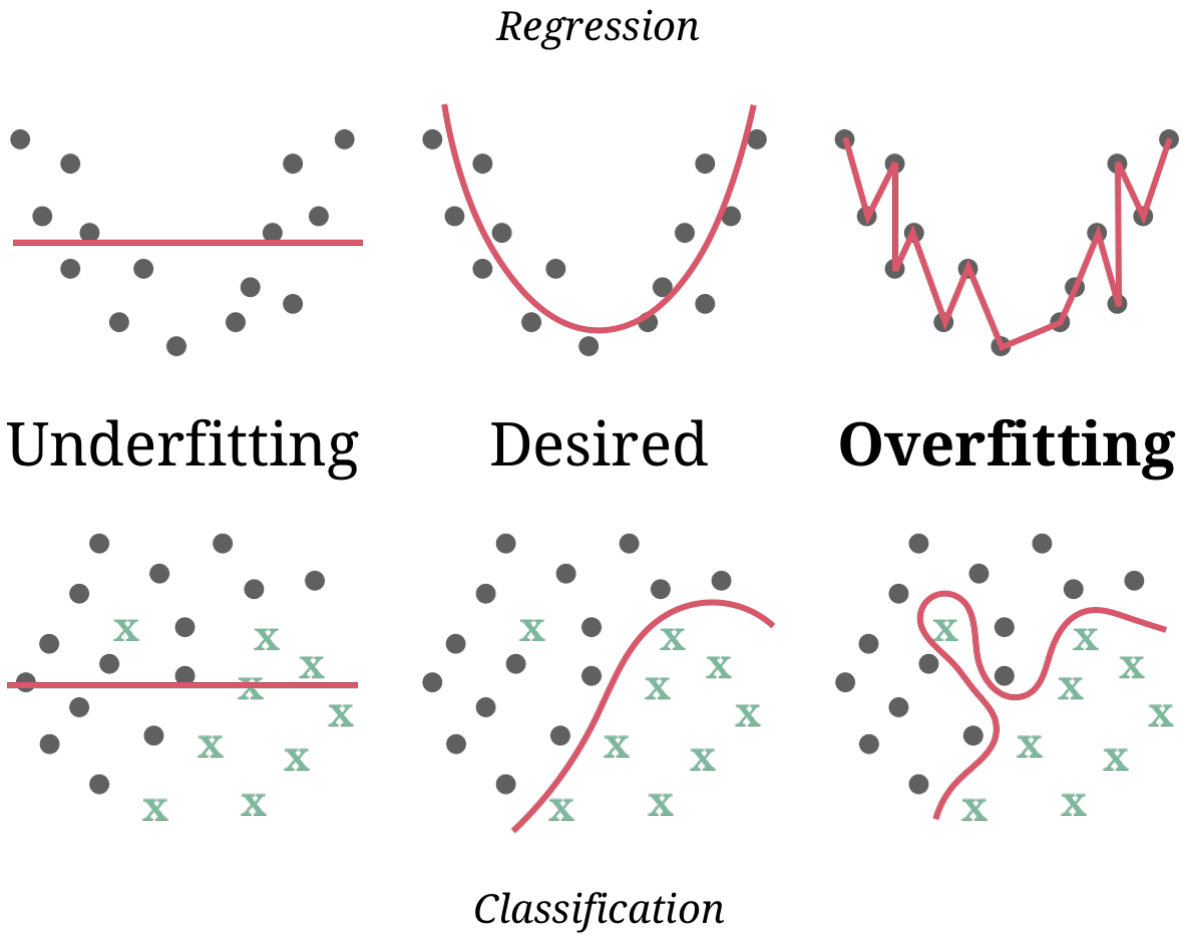
- Having the following equation allow us to fit quadratic input x_1

$$y' = w_2 (x_1)^2 + w_1 x_1 + w_0$$

- The same is applicable for the Logistic Regression we can learn non linear decesion boundries
- By this trick we can increase the model complexity.

But how this effect the learning?

Overfitting and Underfitting (1/2)



Note

A good model (best fit) should be able to generalize to new (unseen) data. How?

Overfitting and Underfitting (2/2)

- **Over-fitting:**
  - Model too complex (flexible)
  - Fits noise in the training data
  - High error is expected on the test data.
- **Under-fitting:**
  - Model too simplistic (too rigid)
  - Not powerful enough to capture salient patterns in training data and test data.

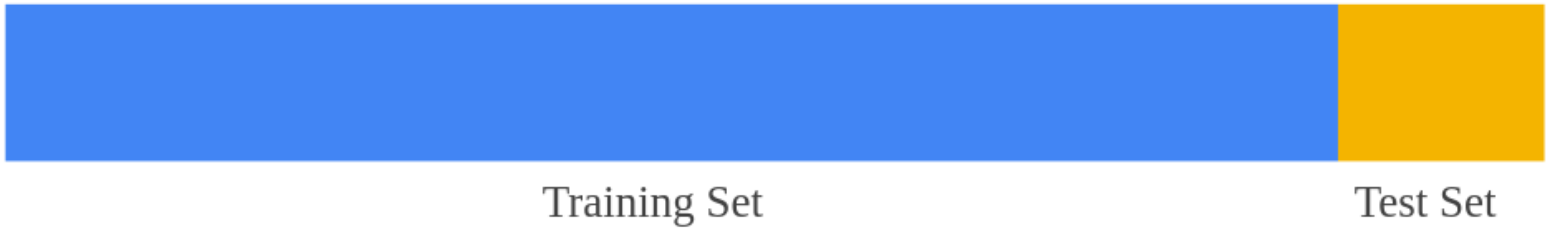
Note

A good model (best fit) should be able to generalize to new (unseen) data. How?

Training and Test Sets

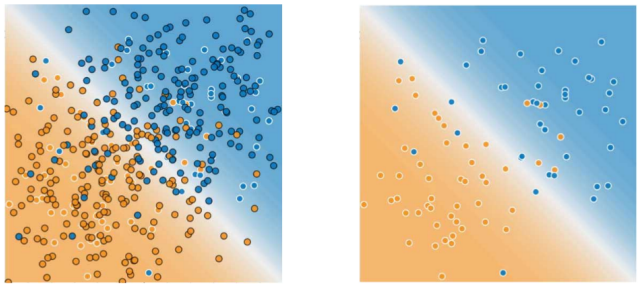
To measure how our model generalize, we split our data to

- **Training set** a subset to train a model.
- **Test set** a subset to evaluate the trained model Estimeate Generalization.



The test should:

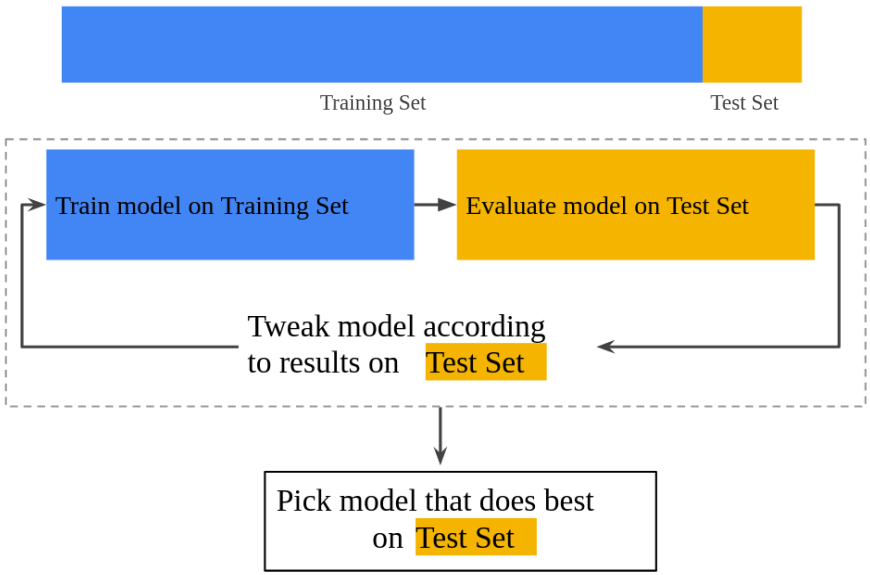
- be large enough to yield statistically meaningful results.
- be representative of the data set as a whole.



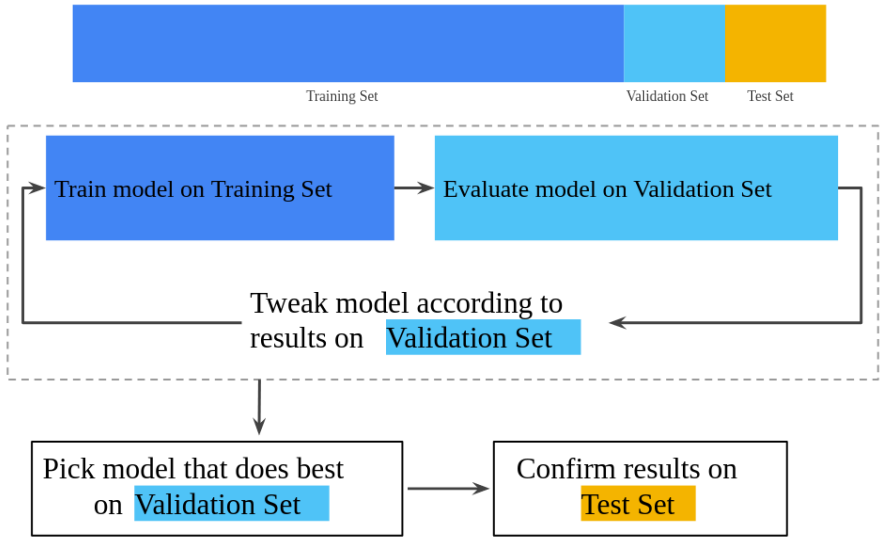
Validation Set

What if we have several model to compare and pick only one?

- Adding or removing features
- Trying different model complexities (linear, quadratic, etc)
- ...



More chances to Overfit.



Less chances to Overfit.

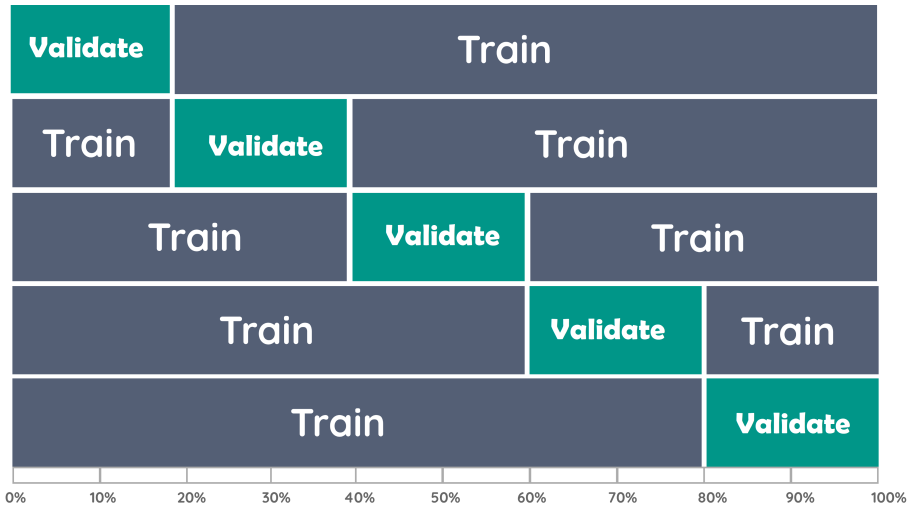
K-Cross Validation

Why?

- We can be exposed to the test set only once.
- We need to estimate future error as accurately as possible.

Ex.

- Randomly split the training into k sets.
- Validate on one in each turn (train on 4 others)
- Average the results over 5 folds



5-fold cross validation

Training vs. Generalization Error (1/3)

Training Error: It measures how we are performing on the training set (same as loss).

$$E_{train} = \frac{1}{|D_{train}|} \sum_{(\mathbf{x},y)\in D_{train}} error(f(\mathbf{x}), y)$$

Generalization Error:

- How well we will do on any kind future data from the same distribution.

$$E_{gen} = \int_{(\mathbf{x},y)\in D} error(f(\mathbf{x}), y) \underbrace{p(\mathbf{x}, y)}_{\text{How often we see (x,y) pair}} d\mathbf{x}$$

Can never compute generalization error practically

Training vs. Generalization Error (2/3)

Test Error:

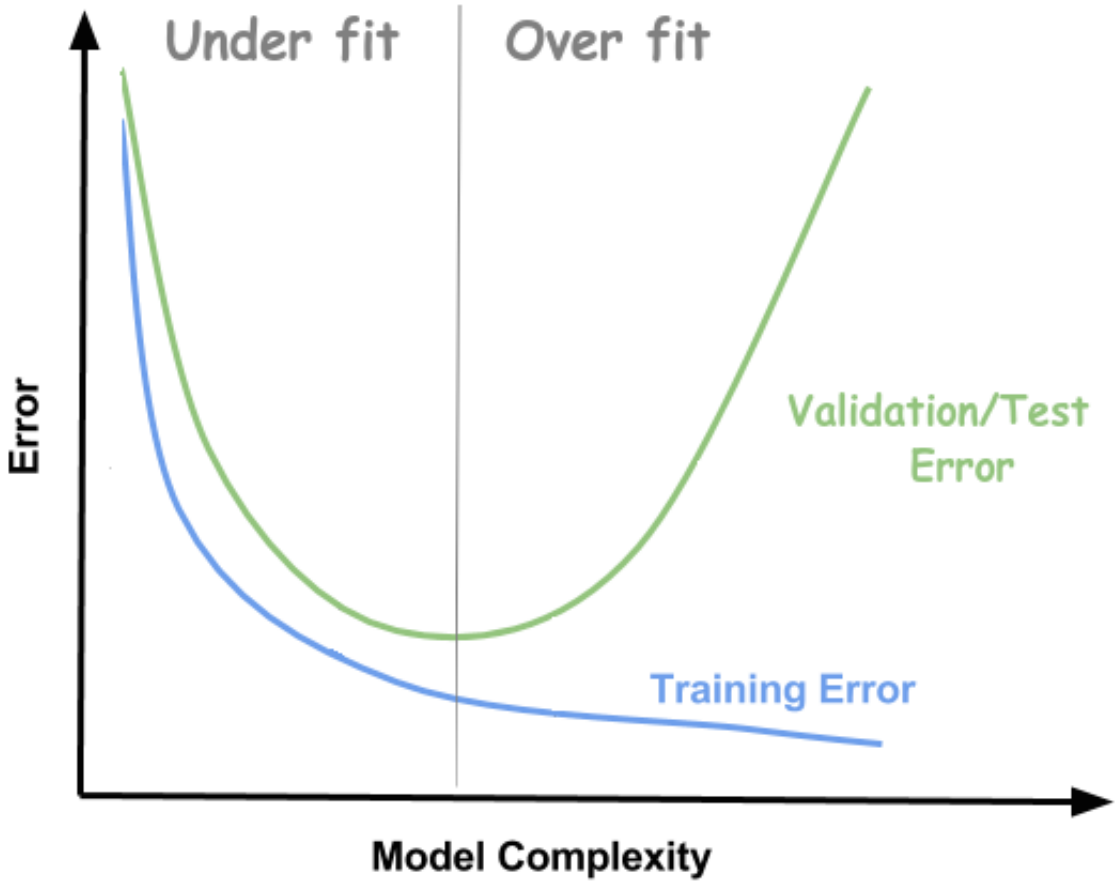
- Introduced to estimate the generalization error.
- That is why we should be exposed to test set only once.

$$E_{test} = \frac{1}{|D_{test}|} \sum_{(\mathbf{x},y)\in D_{test}} error(f(\mathbf{x}), y)$$

- How close E_{gen} to E_{test}? depends on $|D_{test}|$.

$$\lim_{|D_{test}|\to\infty} E_{test} \approx E_{gen}$$

Training vs. Generalization Error (3/3)

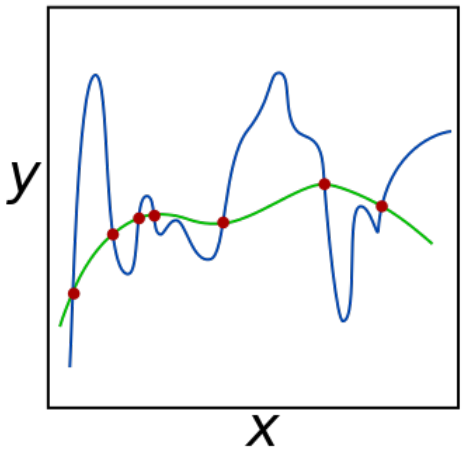


Regularization

- Sparse feature vectors often contain many dimensions. Feature cross results in even more dimensions (more model complexity).
- Can we force the weights for the meaningless features to drop to 0.

L2 and L1 penalize weights differently:

- L_2 penalizes $weight^2$. (L2 has not a discontinuity at 0)
  - $\ell_{new} = \ell_{old} + \mathrm{p} \sum_{i=0}^{i=n} w_i^2$
  - $\frac{\ell_{new}}{dw_i} = \frac{\ell_{old}}{dw_i} + 2\mathrm{p}\ w_i$
- L_1 penalizes $|weight|$. (L1 has a discontinuity at 0)
  - $\ell_{new} = \ell_{old} + \mathrm{p} \sum_{i=0}^{i=n} |w_i|$
  - $\frac{\ell_{new}}{dw_i} = \frac{\ell_{old}}{dw_i} + \mathrm{p}\ sign(w_i)$



p is the tuning parameter which decides how much we want to penalize w_i.

Outline

Scaling feature values

Scaling

Scaling means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range (for example, 0 to 1 or -1 to +1).

Why?

1. Helps gradient descent converge more quickly.
2. Helps to decrease the possibility of weights over/under-flow.
3. Helps the model learn appropriate weights for each feature. Without feature scaling, the model will pay too much attention to the features having a wider range.

Min Max Scaler

Transform features by scaling each feature(f) to a given range [Min, Max].

$$u = \frac{f - f.min}{f.max - f.min}$$

$$f_{scaled} = u * (Max - Min) + Min$$

Z-score Scaler

Another popular scaling tactic is to calculate the Z score of each feature (f). The Z score relates the number of standard deviations away from the mean. In other words:

$$f_{scaled} = (f - f.mean)/f.std\_dev$$

Issues in the Datasets

In real-life, many examples in data sets are unreliable due to one or more of the following:

- **Omitted values.** For instance, a person forgot to enter a value for a house's age.
  - Categorical (N/A) , fill-in mean, remove ,other methods rule-learners, decision trees
- **Duplicate examples.** For example, a server mistakenly uploaded the same logs twice.
  - Remove duplicates
- **Bad labels.** For instance, a person mislabeled a picture of cat as a dog.
  - Clustering can help
- **Bad feature values/Outliers.** For example, someone typed in an extra digit, or a thermometer was left out in the sun.
  - Histogram can help.

How to Handle?

Always try to visualize the features. Histograms are also great mechanism

for visualizing your data in the aggregate. In addition, getting statistics like the following can help:

- Maximum and minimum
- Mean and median
- Standard deviation

Outline

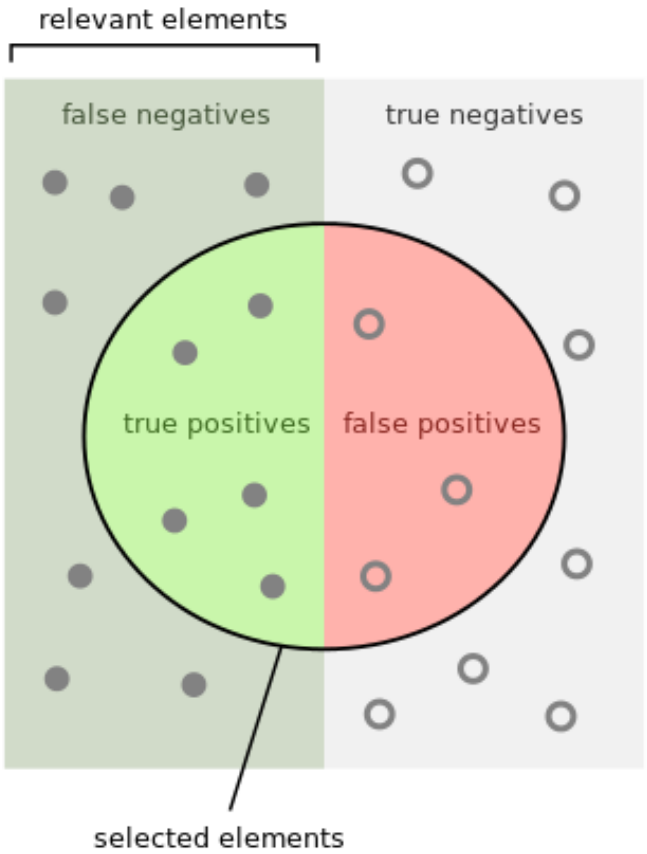
Thresholding

- In order to map a logistic regression value to a binary category, we must define a classification threshold.
- Classification threshold is problem-dependent.
- It does not have to be 0.5.
- Classification metrics are used to define the classification threshold.

Confusion Matrix

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

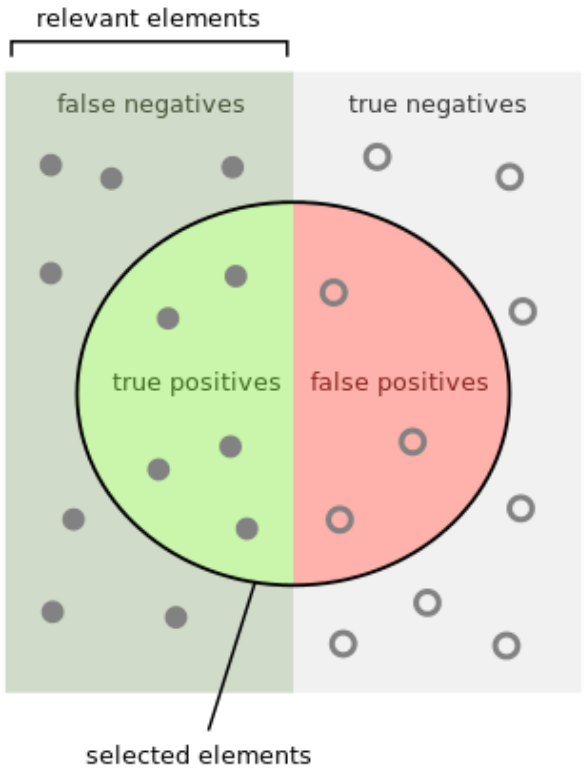


We want large diagonal, small FP, FN

Accuracy and Error

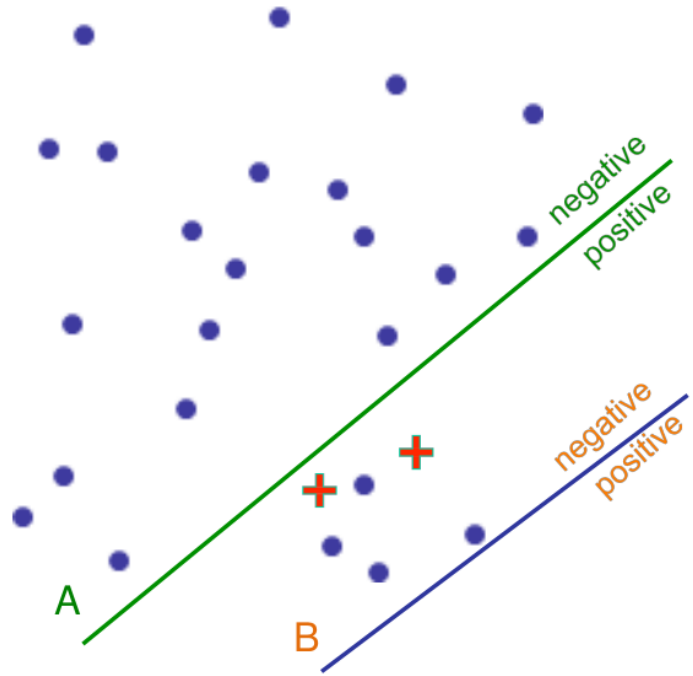
- **Accuracy** is the total correct prediction
  - $\frac{TP+TN}{TP+TN+FP+FN}$
- **Error** is the total false prediction
  - $\frac{FP+FN}{TP+TN+FP+FN} = 1$ - Accuracy
- **Problem:** cannot handle unbalanced classes
  - Predict whether an earthquake is about to happen
  - Happen very rarely, very good accuracy if always predict No.

	Actually Positive (1)	Actually Negative (0)	
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)	P'
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)	N'
	P	N	



Problem with Accuracy

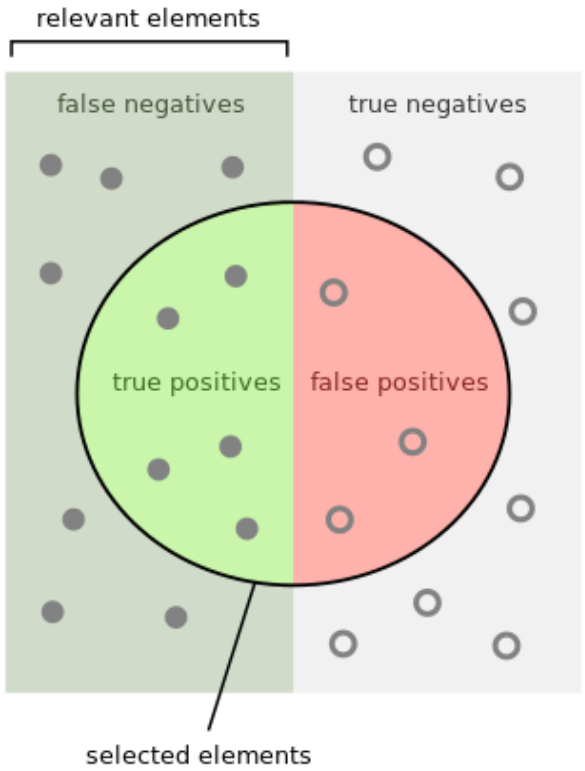
- Youre predicting cancer possiblity (+) vs. not (•)
- Accuracy will prefer classifier B (fewer errors)
- Classifier A is better though.



Metrics (1/3)

	Actually Positive (1)	Actually Negative (0)	
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)	P'
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)	N'
	P	N	

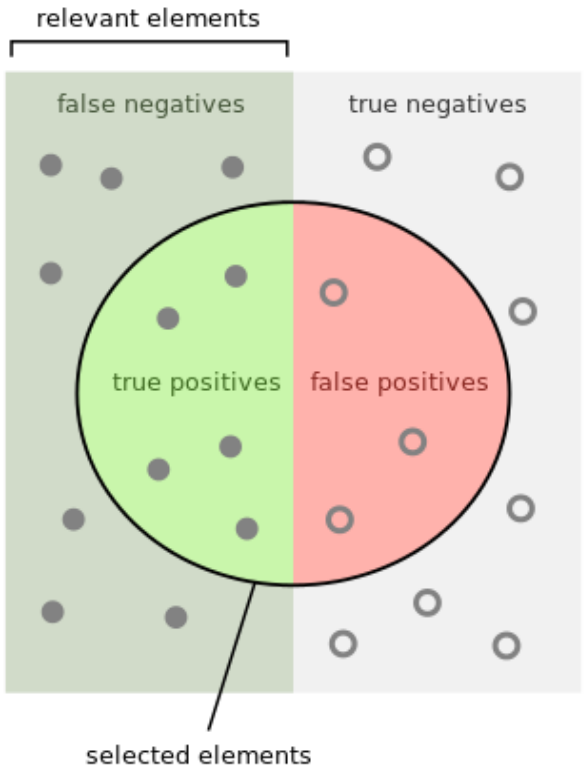
- **Recall** How many (+) we hit? (Recall = Sensitivity = True pos rate = hit rate)
  - $\frac{TP}{P} = \frac{TP}{TP+FN}$
- **Miss Rate** How many (+) we miss? (Miss rate = False neg rate = false rejection = type II error rate)
  - $1 - hitrate = \frac{FN}{P} = \frac{FN}{TP+FN}$



Metrics (2/3)

	Actually Positive (1)	Actually Negative (0)	
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)	P'
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)	N'
	P	N	

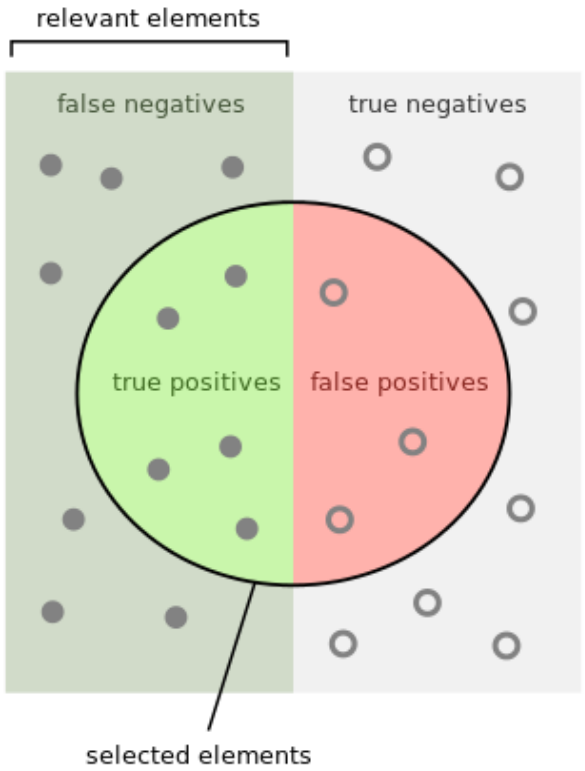
- **Specificity** How many (-) we hit? (Specificity = True neg rate)
  - $\frac{TN}{N} = \frac{TN}{FP+TN}$
- **False Alarm** How many (-) we miss OR How many (+) we falsely accepted? (False alarm = False pos rate = false acceptance = = type I error rate) How many irrelevant items are selected?
  - $1 - Specificity = \frac{FP}{FP+TN}$



Metrics (3/3)

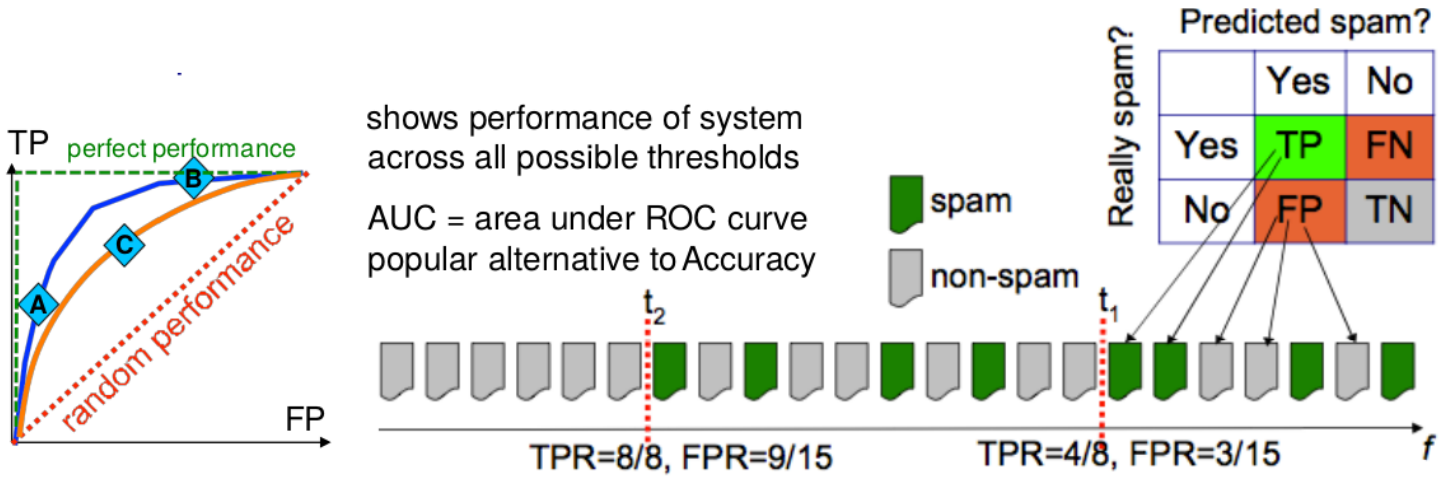
	Actually Positive (1)	Actually Negative (0)	
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)	P'
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)	N'
	P	N	

- **Prcesion** How many of our (+) decesions is correct?
  - $\frac{TP}{P'} = \frac{TP}{TP+FP}$
- **F1 measure** Harmonic mean of precesion and the recall
  - $2\frac{PER*REC}{PER+REC}$



ROC curves (1/2)

- Plot TPR vs. FPR as classification threshold (t) varies from 0 to 1
- RoC summarizes all the confusion matrices for all possible thresholds.
- Each point on the RoC is for a different classification threshold.
- (1,1) point is all (+) threshold.
- (0,0) point is all (-) threshold.
- Benefits:
  - Make us make a decesion for classification threshold.
  - Make us compare different classifier using AUC.

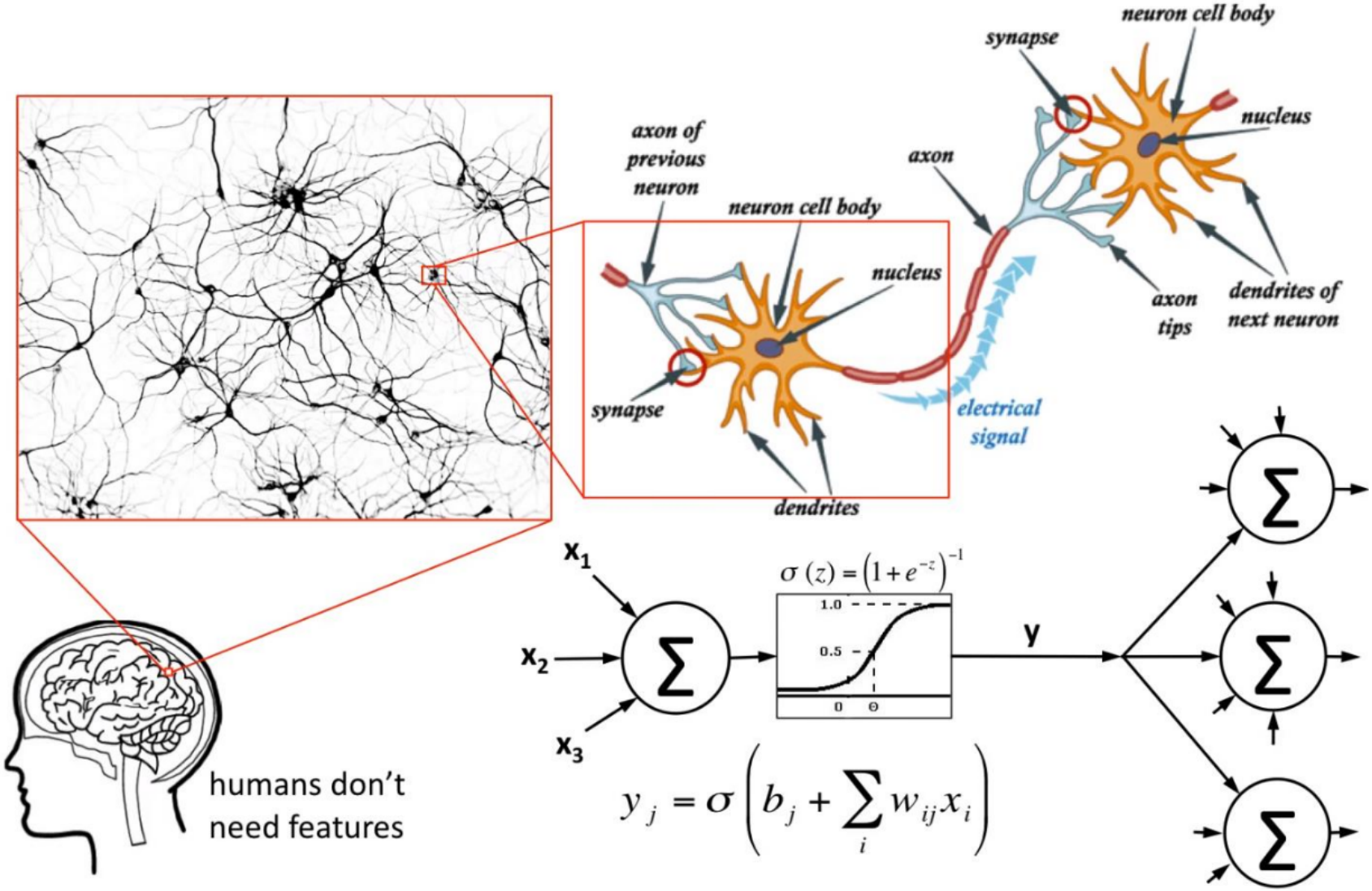


Outline

Introduction

- One of the oldest and one of the newest machine learning models.
- Goes back to 1940, when people started to build modles the imitate the human brain.
- Logistic regression (perceptron) is the core of neural networks started in 1950.
- However, scientists in that time showed that a single perceptron can not solve xor problem (died).
- Reborn in 1980, discovery of merging perceptrons together. But died due to the resources requirements
- Reborn in the last decade with the advancement of the computation resouces.

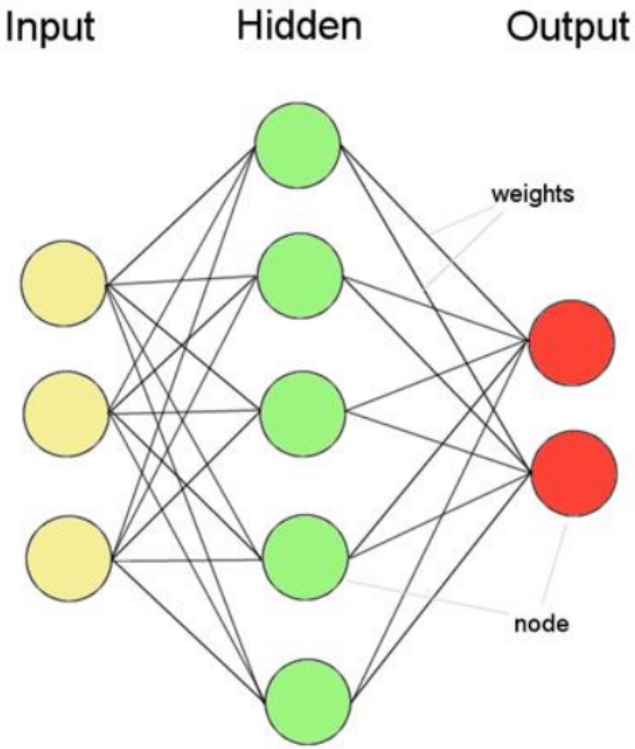
Neurons and the brain



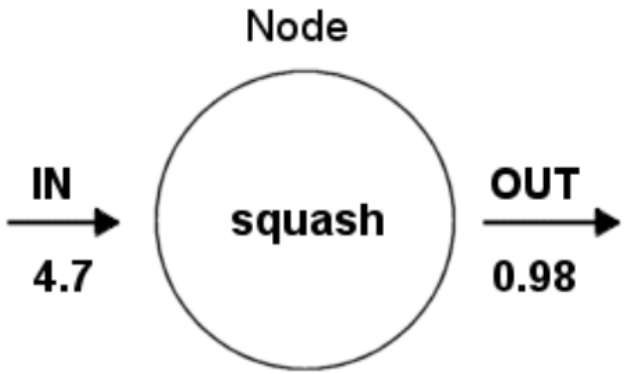
Artificial NNs

- ANNs incorporate the two fundamental components of biological neural nets.

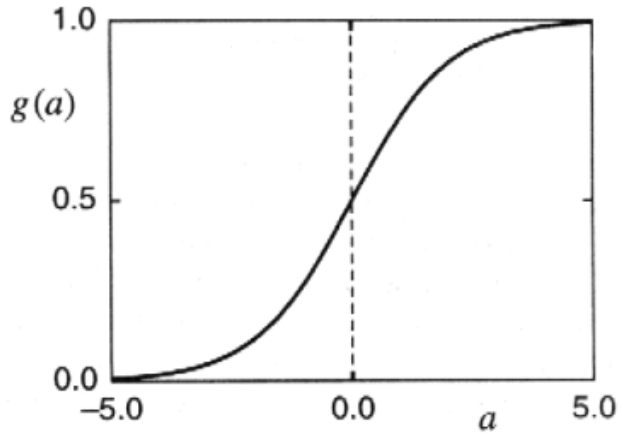
1. Neurones (nodes)
2. Synapses (weights)



Structure of a node

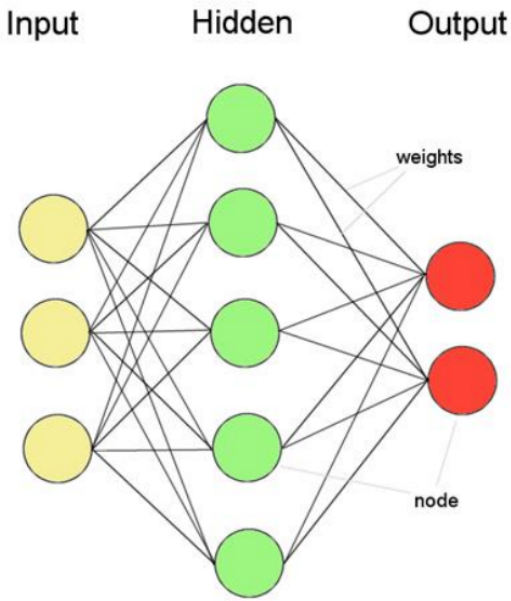


Squashing/Activation function limits node output:



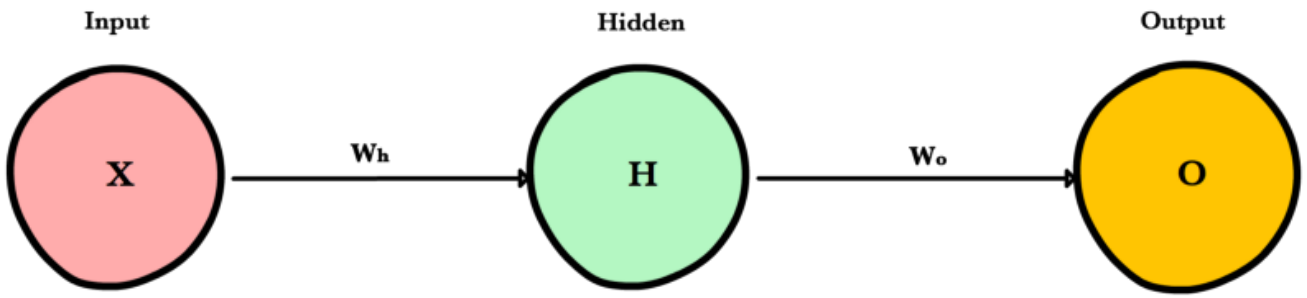
Types of Layers

1. The input layer
   - Introduces input values into the network.
   - No activation function or other processing.
2. The hidden layer(s)
   - Perform classification of features
   - Two hidden layers are sufficient to solve any problem
3. The output layer
   - Functionally just like the hidden layers
   - Outputs are passed on to the world outside the neural network.



Feedforward Simple Network

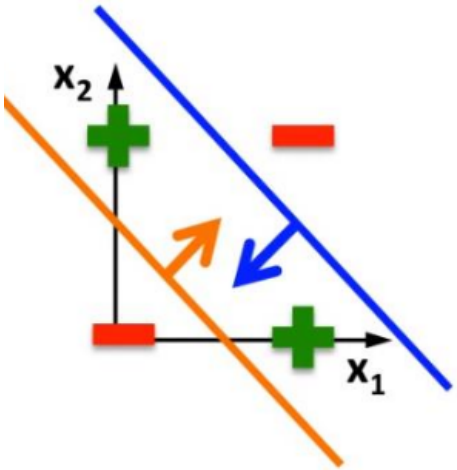
Also known as Multi-Layer Perceptron (MLP).

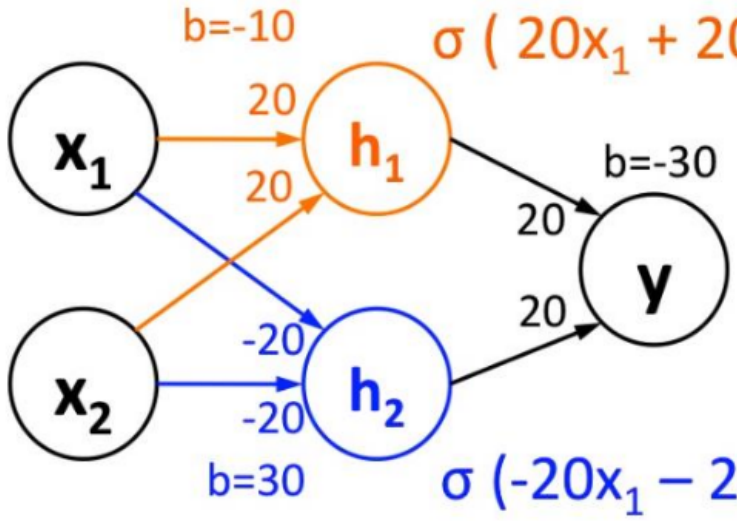


$$O = F(F(X * W_h)W_o)$$

Solving XOR with a Neural Network

Linear classifiers cannot solve this





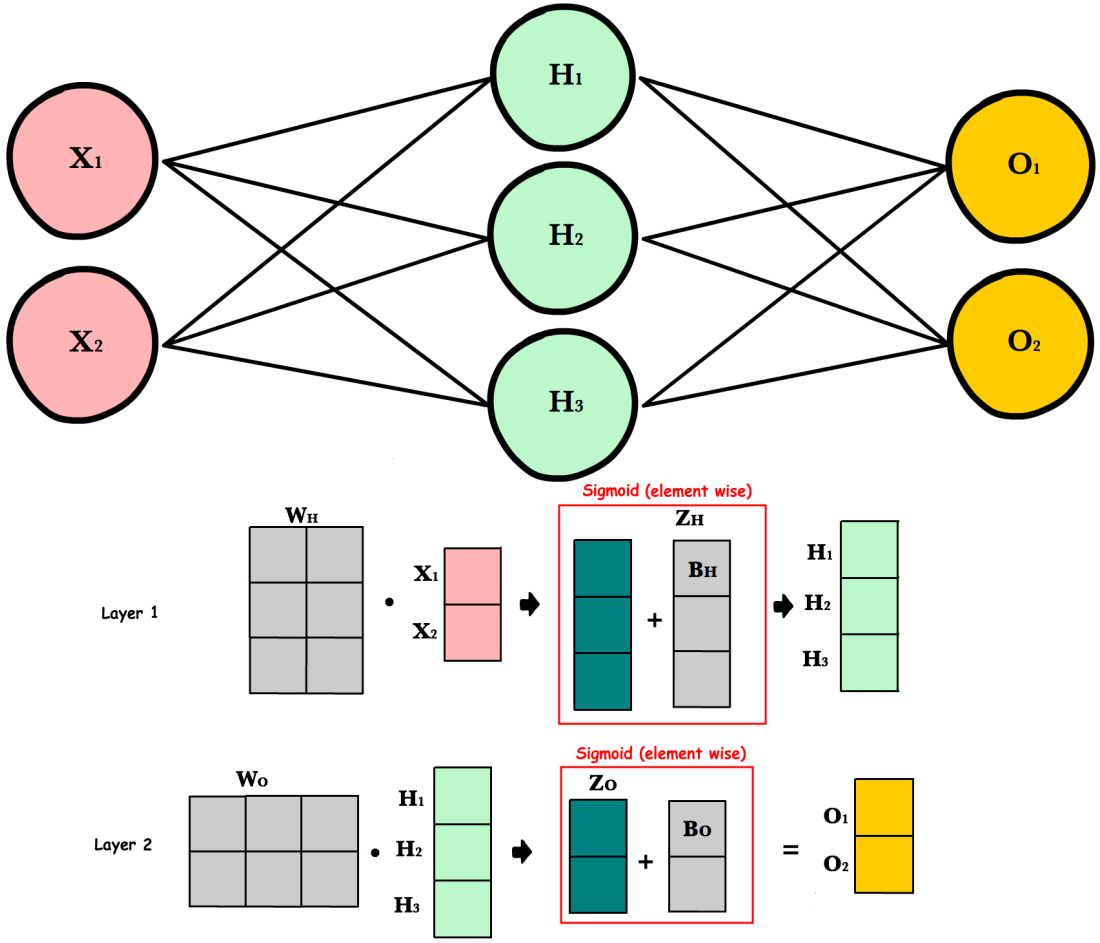
$\sigma(20*0 + 20*0 - 10) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$
$\sigma(20*0 + 20*1 - 10) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$

$\sigma(-20*0 - 20*0 + 30) \approx 1$
$\sigma(-20*1 - 20*1 + 30) \approx 0$
$\sigma(-20*0 - 20*1 + 30) \approx 1$
$\sigma(-20*1 - 20*0 + 30) \approx 1$

$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*1 - 30) \approx 1$

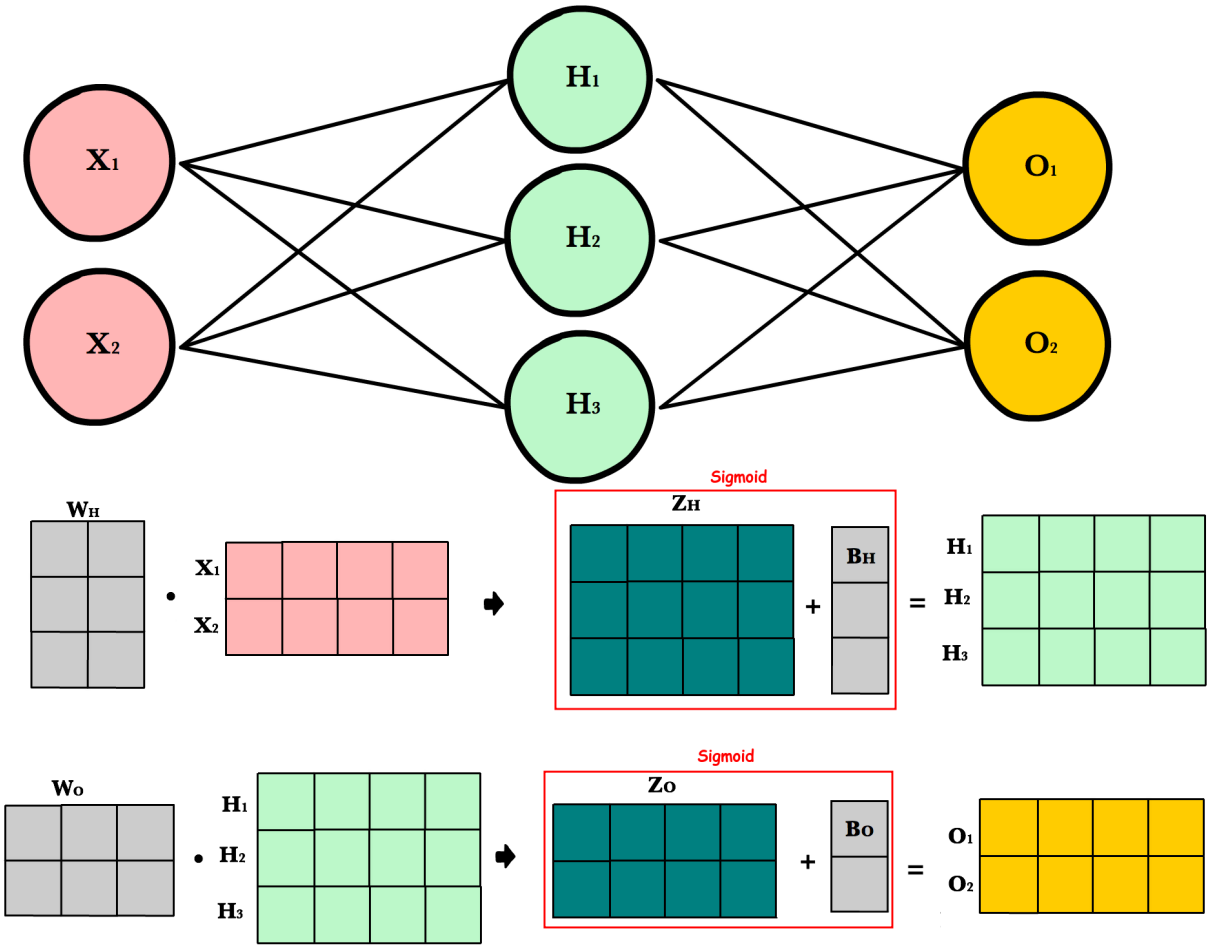
Working with Matrices

- Three features.
- One input at the time.



Working with Matrices

- Three features.
- Four inputs at the time.



Feed Forward Properties

- **Input dimension:** No of features $\times$ No of Samples in the Batch
- **Weight dimension at (just before) layer L:** No of neurons in layer L $\times$ No of neurons in layer (L-1)
- **Data dimension at layer L:** No of neurons in layer L $\times$ No of Samples in the Batch.

FeedForward Equations

1. $\mathbf{Z}^L = \mathbf{W}^L \cdot \mathbf{X} + \mathbf{b}^L \Rightarrow$ Matrices Operations
2. $\mathbf{A}^L = \sigma(\mathbf{Z}^L) \Rightarrow$ Element wise Operations

Chain Rule Refresher

- Foward propagation can be viewed as a long series of nested equations.
- **Backpropagation** is merely an application the Chain Rule to find the Derivatives of cost with respect to any variable in the nested equation.
  Ex.

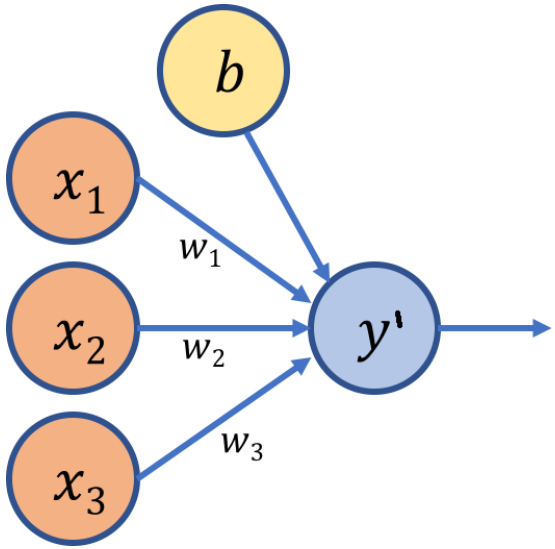
  What is $\frac{df}{dx}$?

$$f(x) = A(w_2 * B(w_1 * C(w_0 * x)))$$

$$\frac{df}{dx} = w_2 * \frac{df}{dA} * w_1 * \frac{dA}{dB} * w_0 * \frac{dB}{dC} * \frac{dC}{dx}$$

Write $\frac{df}{dx}$? if A(x)= Cos(x), B(x)= Sin(x), C(x)=$e^{(x)}$

Logistic Regression as Neural Network (1/2)

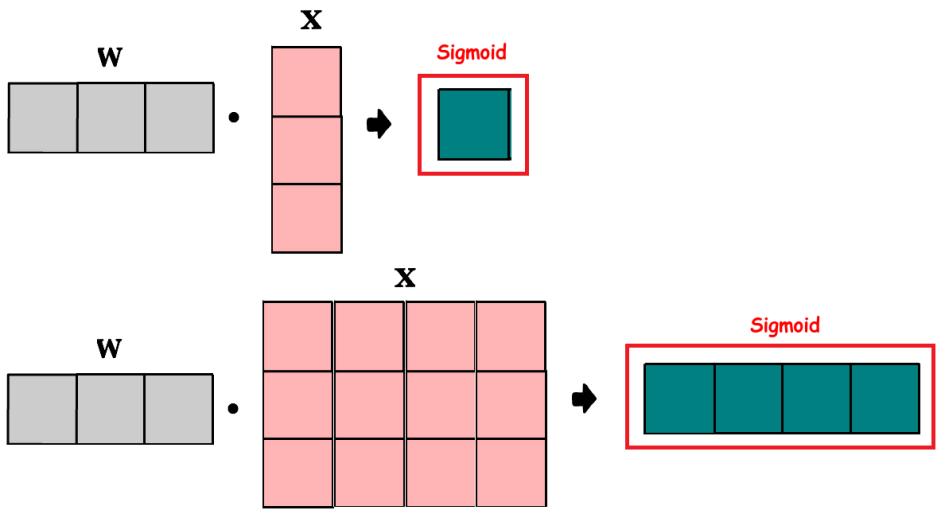


Feedforwad one sample

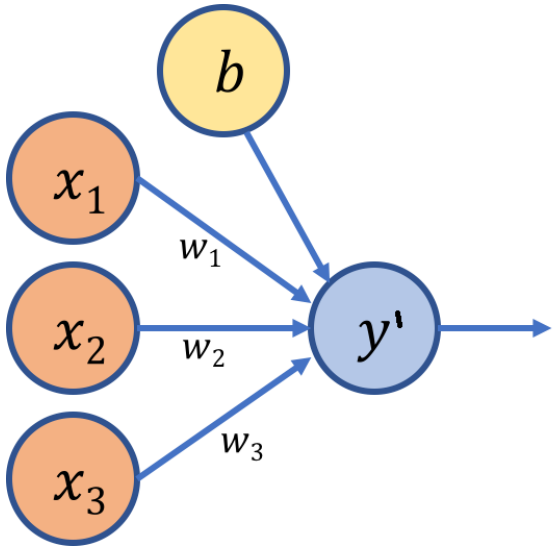
1. $z = \mathbf{w}^T \cdot \mathbf{x} + b$
2. $y = \sigma(z)$

Feedforwad batch

1. $\mathbf{z} = \mathbf{w} \cdot \mathbf{X} + b$
2. $\mathbf{y} = \sigma(\mathbf{z})$



Logistic Regression as Neural Network (2/2)

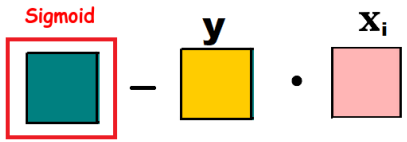


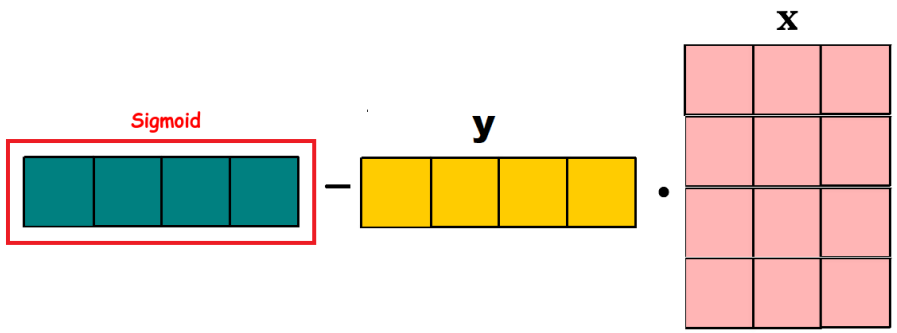
Loss one sample (using log loss)

1. $\frac{d\ell}{dw_i} = (y' - y) * x_i$

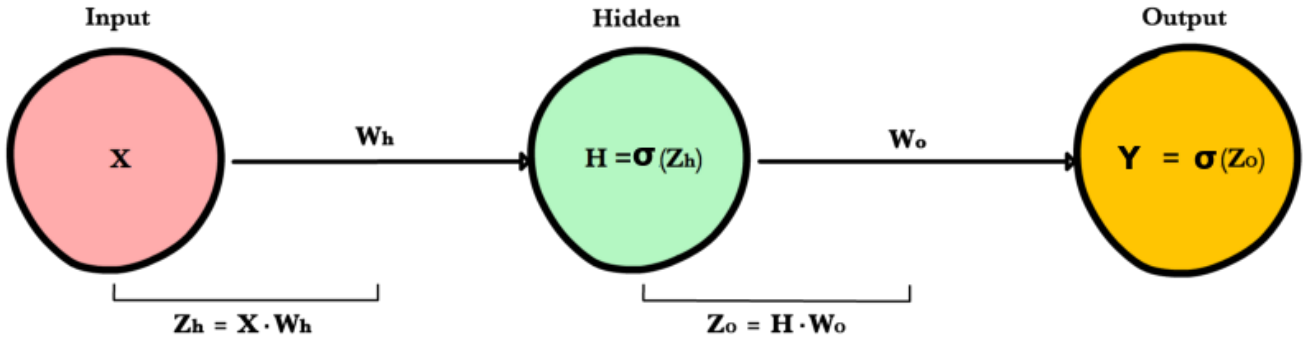
Loss four samples (using log loss)

1. $\frac{d\ell}{d\mathbf{w}} = (\mathbf{y}' - \mathbf{y}) \cdot \mathbf{X}^T$



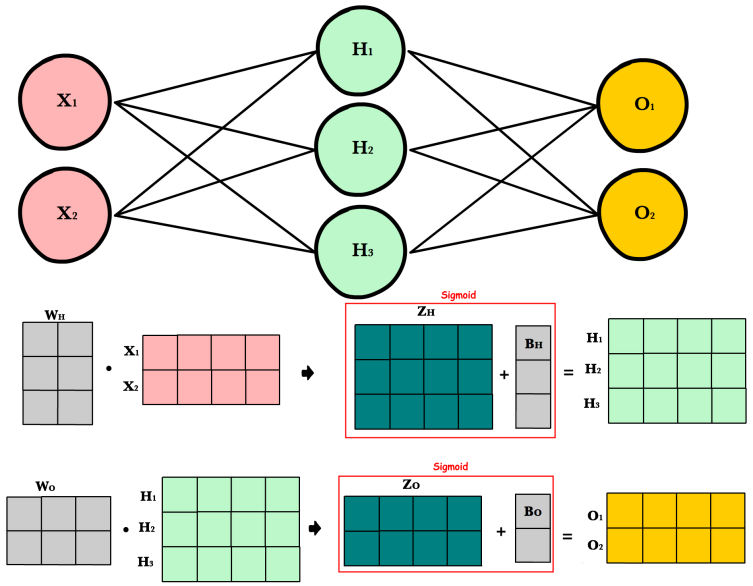


Backpropagation



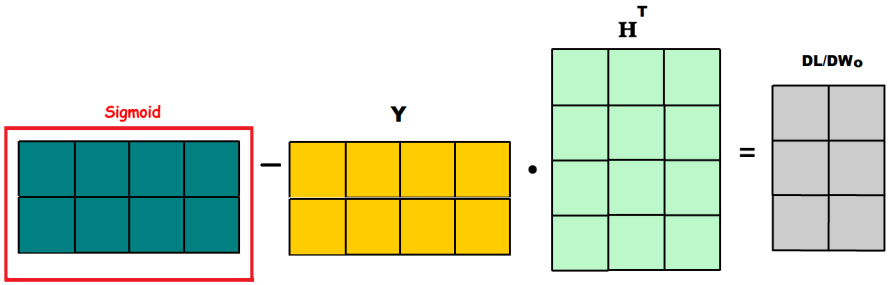
Function	Formula	Derivative
Weighted Input	$\mathbf{Z} = \mathbf{XW}$	$\frac{dZ}{dX} = \mathbf{W}^T$, $\frac{dZ}{dW} = \mathbf{X}^T$
Sigmoid	$\sigma(\mathbf{Z})$	$\sigma(\mathbf{Z}) \otimes \sigma(1 - \mathbf{Z})$
Loss	*LogLoss*	$(\mathbf{Y'}\text{-}\mathbf{Y})$

Backpropagation (1/2)

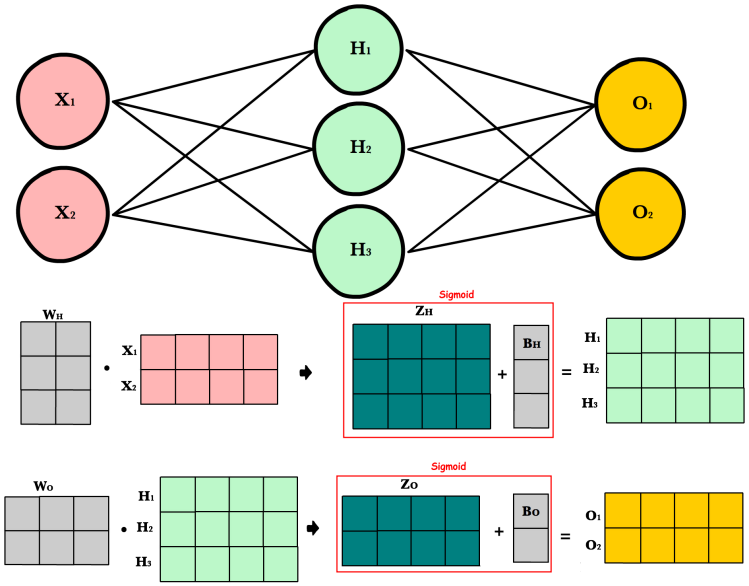


What is the derivative of cost with respect to W_o?

$$\frac{d\ell}{d\mathbf{W}_o} = \frac{d\ell}{d\mathbf{Y}} \frac{d\mathbf{Y}}{d\mathbf{Z}_o} \frac{d\mathbf{Z}_o}{d\mathbf{W}_o}$$
$$= (\mathbf{Y'} - \mathbf{Y}) \cdot \mathbf{H}^T$$
$$= E_o \cdot \mathbf{H}^T$$



Backpropagation (1/2)

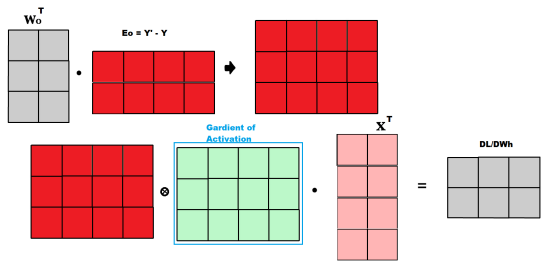


What is the derivative of cost with respect to W_h?

$$\frac{d\ell}{d\mathbf{W}_h} = \frac{d\ell}{d\mathbf{Y}}\frac{d\mathbf{Y}}{d\mathbf{Z}_o}\frac{d\mathbf{Z}_o}{d\mathbf{H}}\frac{d\mathbf{H}}{d\mathbf{Z}_h}\frac{d\mathbf{Z}_h}{d\mathbf{W}_h}$$

$$= ((\mathbf{W}_o{}^T \cdot E_o) \otimes (\sigma(\mathbf{Z}_h) \otimes (1 - \sigma(\mathbf{Z}_h))) \cdot \mathbf{X}^T$$

$$= E_h \cdot \mathbf{X}^T$$



Backpropagation Notes

- For any layer (L) = :

$$\frac{d\ell}{d\mathbf{W}_L} = \mathbf{E}_L \cdot \mathbf{input}^T$$

$$\mathbf{E}_L = (\mathbf{W}_{L+1}^T \cdot \mathbf{E}_{L+1}) \otimes \mathbf{A}'_L$$

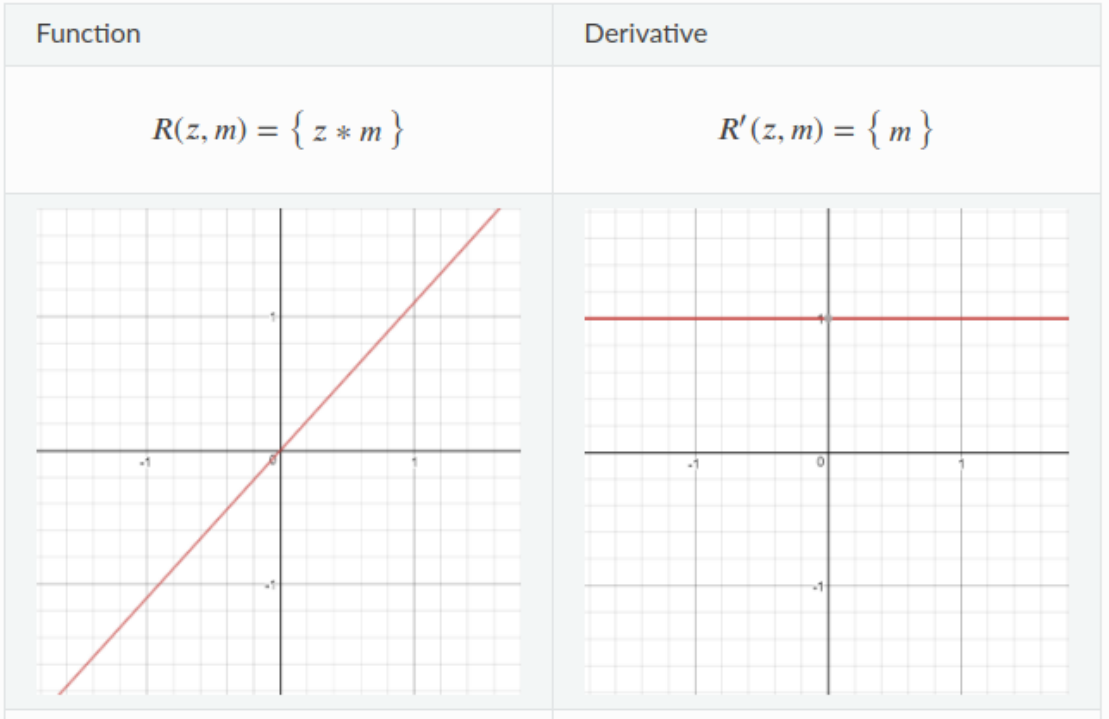
Activation functions

Activation functions typically have the following properties:

- **Non-linear:** To model complex relationships
- **Continuously Differentiable:** To improve our model with gradient descent.
- **Fixed Range Activation** functions typically squash the input data into a narrow range that makes training the model more stable and efficient.

Linear Activation Function

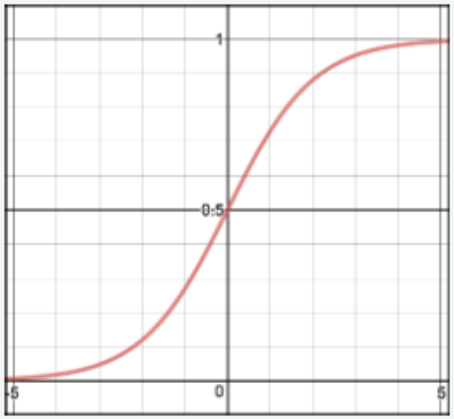
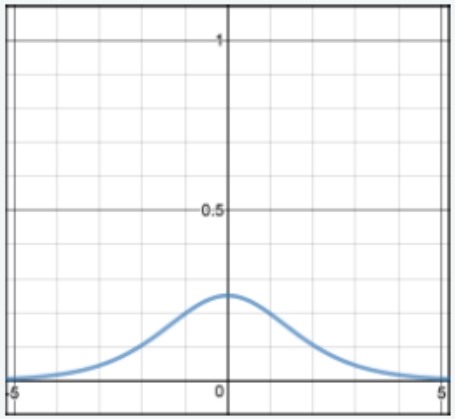
- The stacking of linear functions introduce nothing new. All layers of the neural network collapse into one.
- No one use it.
- It can blow up the activation.



Linear Activation Function

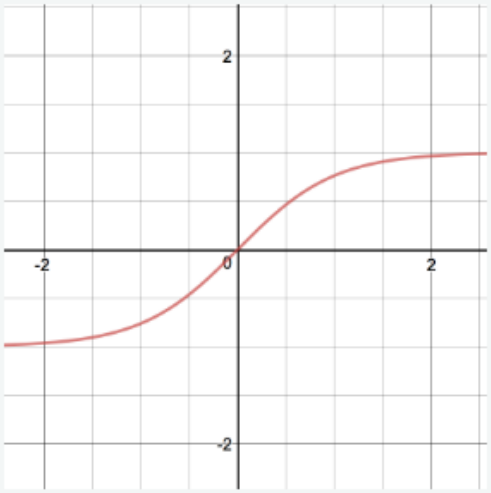
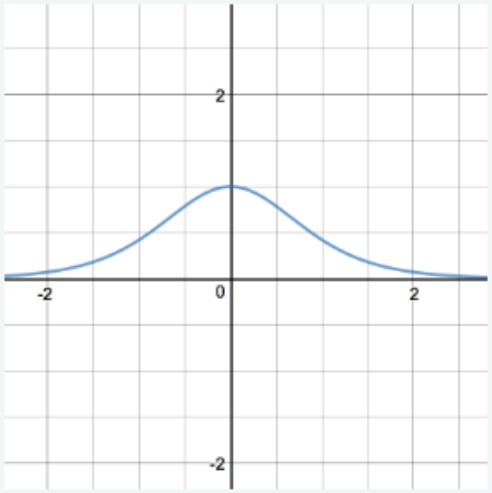
Sigmoid / Logistic

- The stacking is possible.
- Smooth gradient.
- Outputs not zero centered.
- Computationally expensive.

Function	Derivative
$S(z) = \frac{1}{1+e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$
	

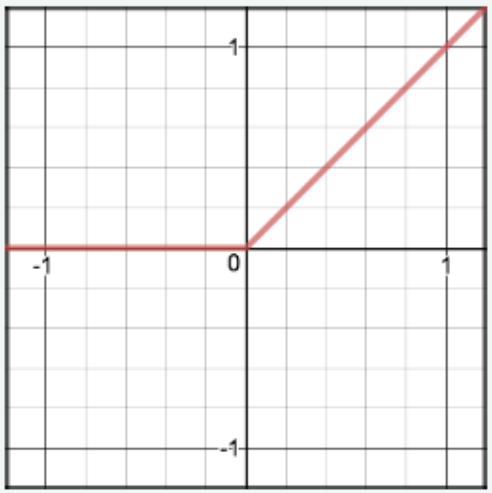
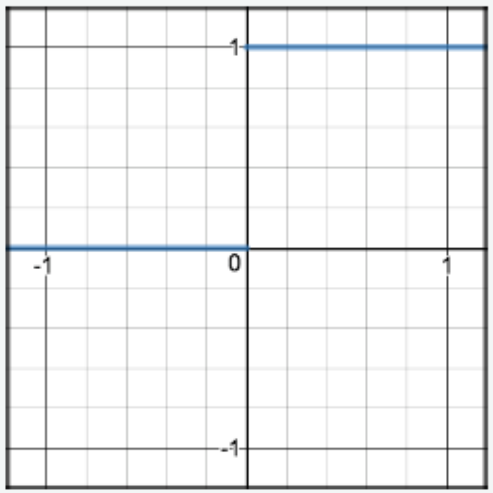
TanH / Hyperbolic Tangent

- The stacking is possible.
- Smooth gradient.
- Zero centered output
- Gradinet is steeper than the sigmoid.
- Computationally expensive.

Function	Derivative
$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$tanh'(z) = 1 - tanh(z)^2$
	

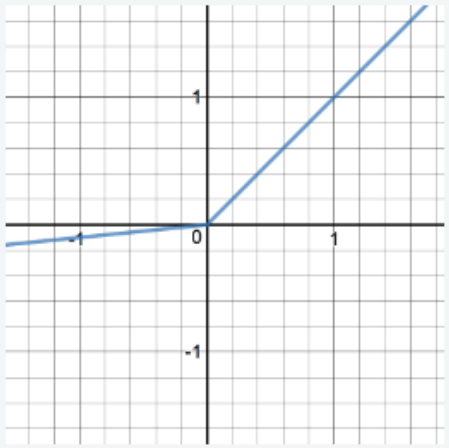
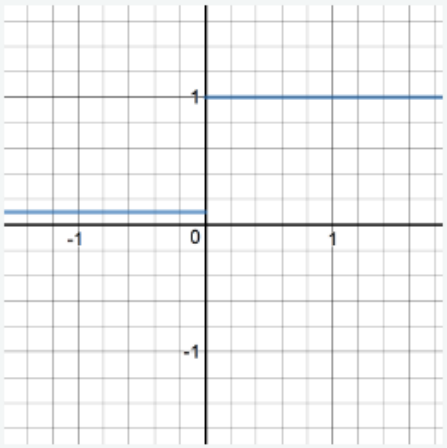
ReLU (Rectified Linear Unit)

- Computationally cheap.
- Suffer from the Dying ReLU problemwhen inputs approach zero, or are negative.
- Gradient not smooth.
- It can blow up the activation.

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z <= 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$
	

Leaky ReLU

- Computationally cheap.
- No Dying ReLU problem.
- Sometimes results are not consistent.
- Gradient not smooth.
- It can blow up the activation.

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ \alpha z & z <= 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ \alpha & z < 0 \end{cases}$
	

Leaky ReLU

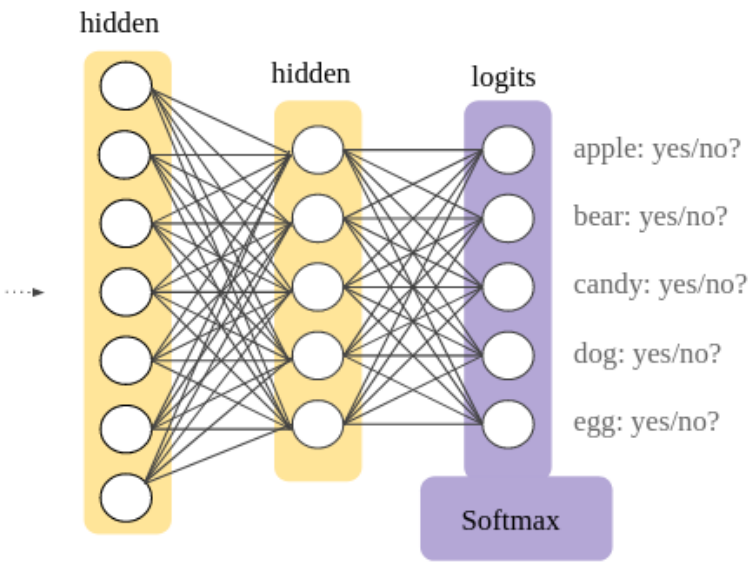
Multi-Class Neural Networks: Softmax Activation

Softmax extends binary logistic regression idea (probability adds upto 1) into a multi-class world.

- It helps training converge more quickly than it otherwise would.
- Usually have better performance against one vs. all classification.

$$p(y = j|\mathbf{z}) = \frac{e^{\mathbf{w}_j^T \mathbf{z}+b_j}}{\sum_{k \in K} e^{\mathbf{w}_k^T \mathbf{z}+b_k}}$$

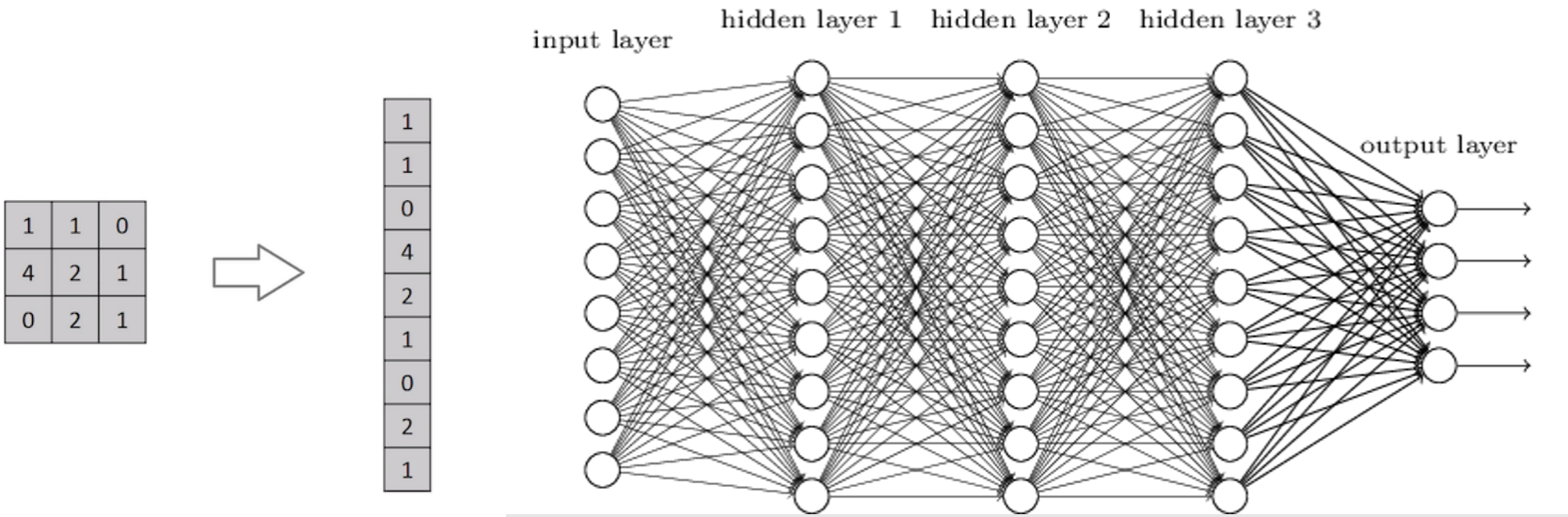
- K is the number of classes. (j and k $\in K$)
- $\mathbf{z}$ is the input vector.
- $\mathbf{w}_{(.)}$ is the weight vector associated with each output neuron.



Why this ugly forumla?

Convolutional Neural Networks (ConvNets)

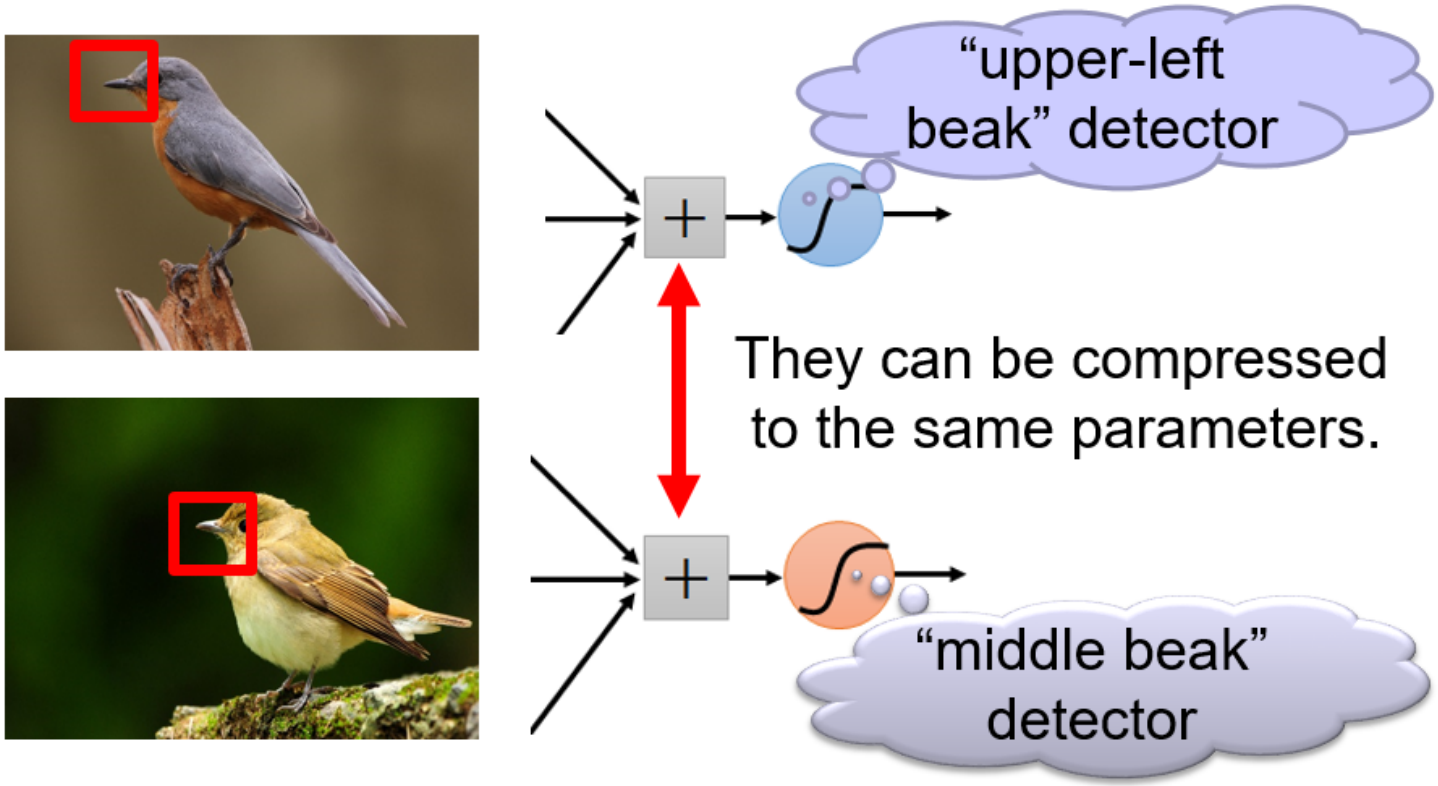
- It is a special structure to deal with image inputs.
- Why not just flatten the image and feed it to a Multi-Level Perceptron for classification purposes?



Do we really need all the edges?

Convolutional Neural Networks (ConvNets)

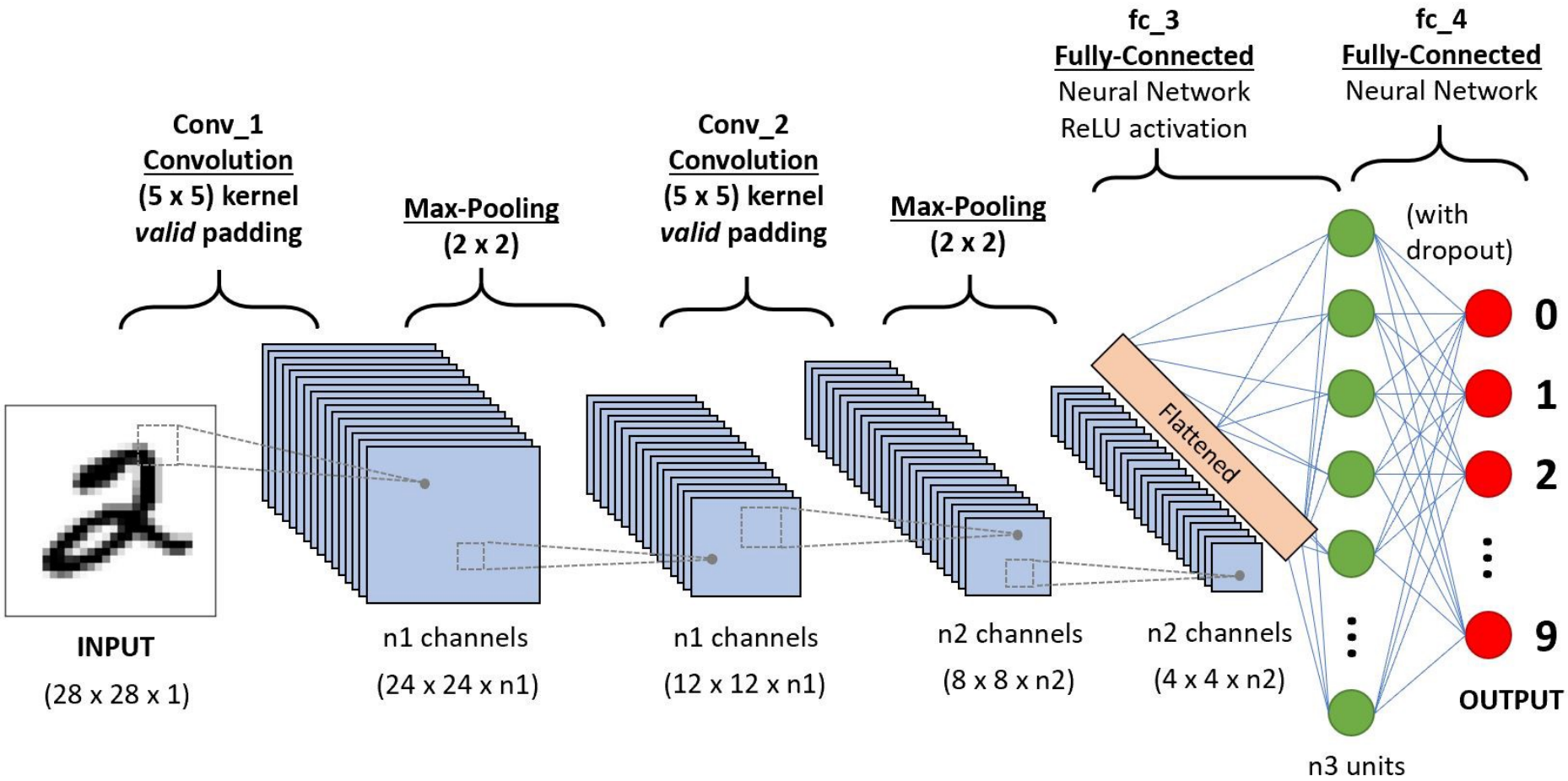
What about Spatial and Temporal dependencies between pixels. (Edges, patterns, curves ...)



In MLP, Large network is needed for all possible positioning of the peak

Convolutional Neural Networks (ConvNets)

In primitive image processing methods filters are hand-engineered, with enough training, ConvNets have the ability to learn filters/characteristics.



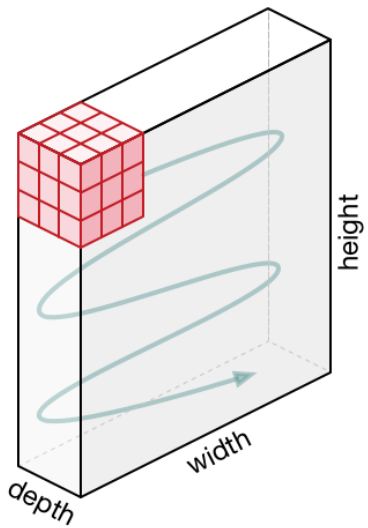
Convolution Layer The Kernel (1/2)

It is a small matrix of learnable weights which have a purpose to detect certain feature in the image (e.g. Horizontal edes).

- **Stride:** The number of shifts the filter is moving for each convolution step.
- **Padding:** Pads the input volume with zeros around the border.
  - Can preserve (as much as possible) the size of the input to prevent losing information.

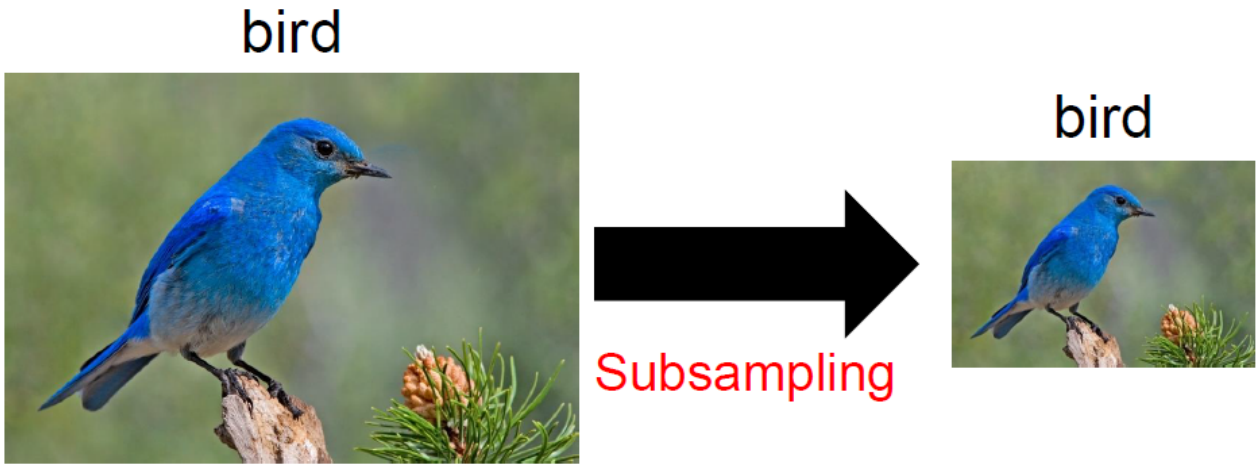
Convolution Layer The Kernel (2/2)

For RGB images kerenls can be 3d matrices.

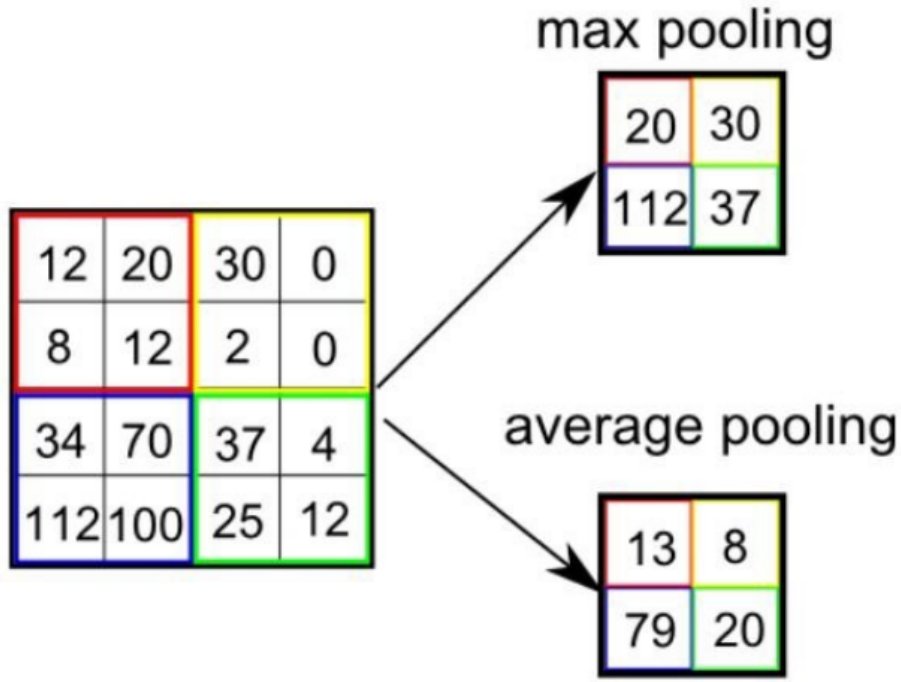


Pooling Layer

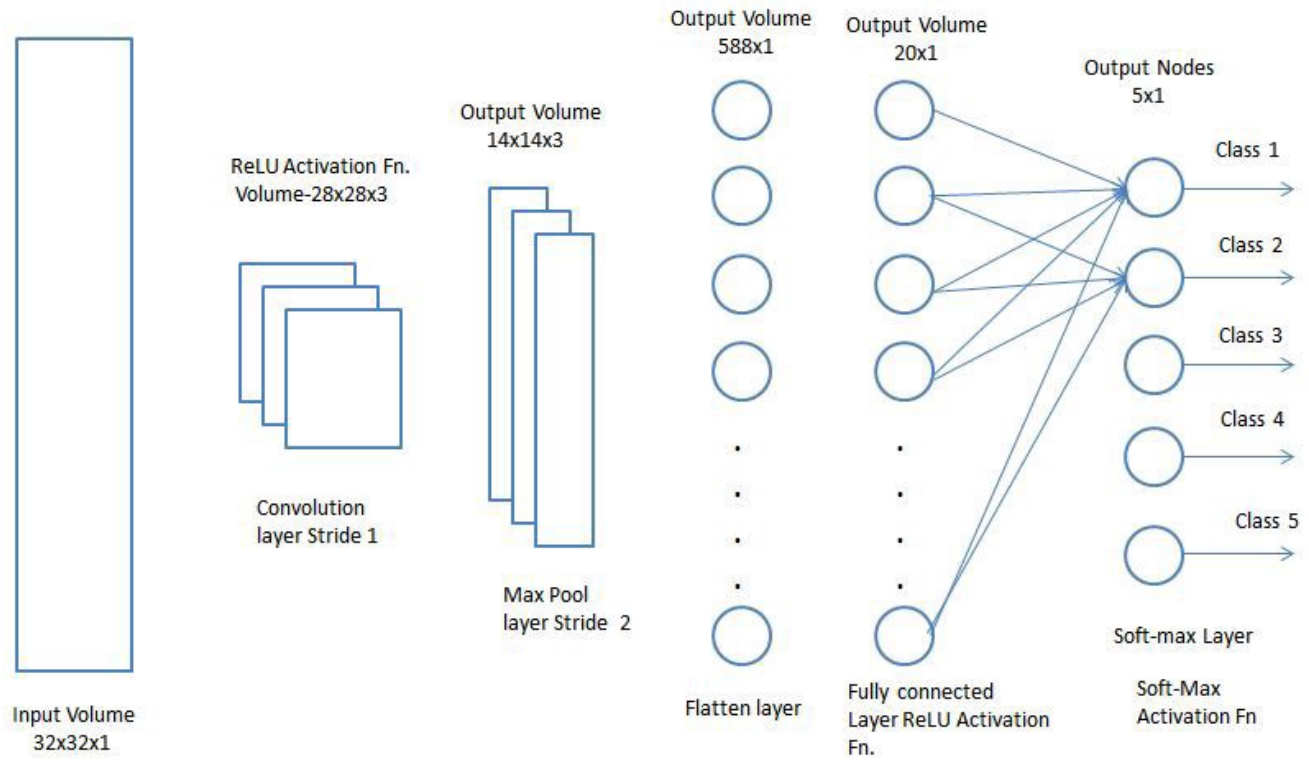
- Pooling layer is responsible for reducing the spatial size of the Convolved Feature.
- This is to decrease the computational power required to process the data.
- Subsampling pixels will not change the object.



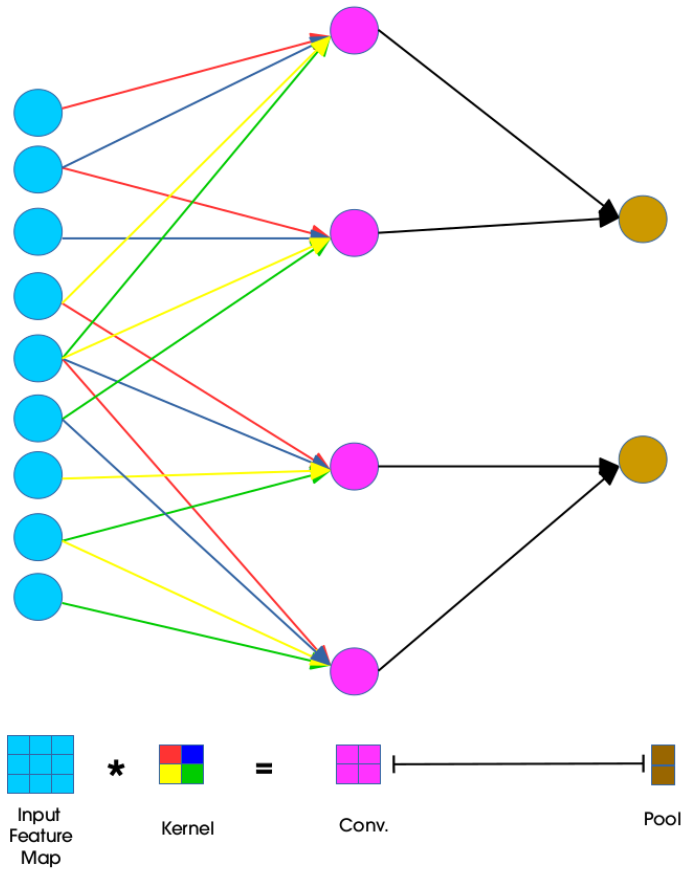
Pooling (2/2)


max pooling
20 30
112 37
12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
average pooling
13 8
79 20

Flatten - Fully Connected Layer



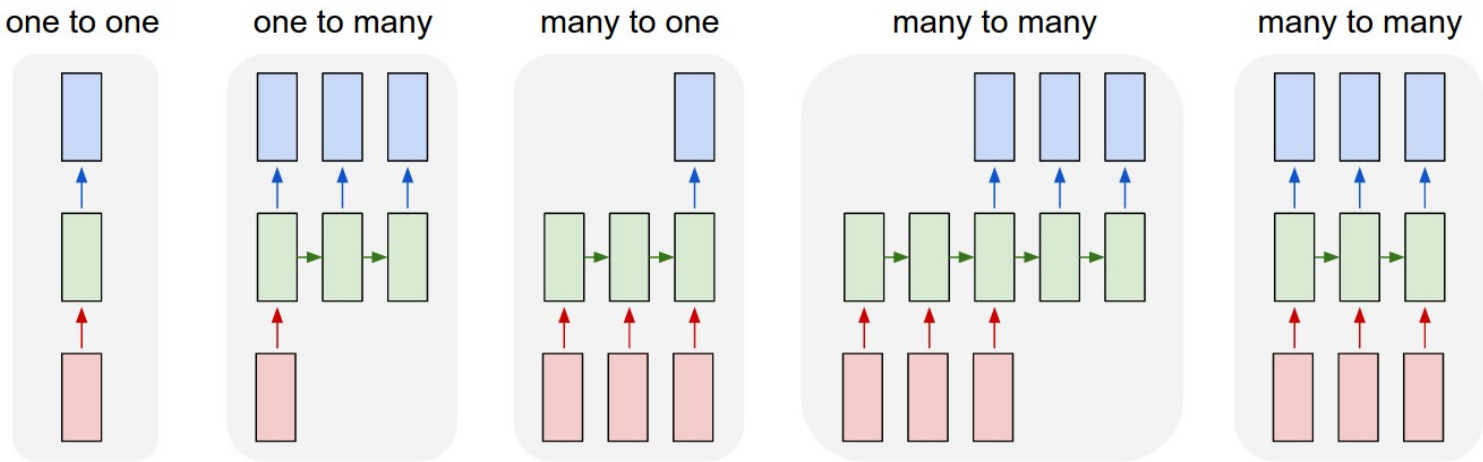
CNN vs MLP



In CNN, very few weights to learn compared to MLP without loss of generality

Recurrent Neural Networks (RNN)

- CNN and MLP accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output
- RNN allow us to operate oversequences of vectors



1. Fixed-sized input to fixed-sized output (e.g. image classification).
2. Sequence output (e.g. image captioning).
3. Sequence input (e.g. sentiment analysis).
4. Sequence input and sequence output (e.g. Machine Translation).
5. Synced sequence input and output (e.g. video classification where we

Recurrent Neural Networks (RNN)

Pros

- Possibility of processing input of any length
- Computation takes into account historical information
- Weights are shared across time

Cons

- Computation being slow
- Difficulty of accessing information from a long time ago (longterm dependencies are hard to capture)

RNN and Gradients

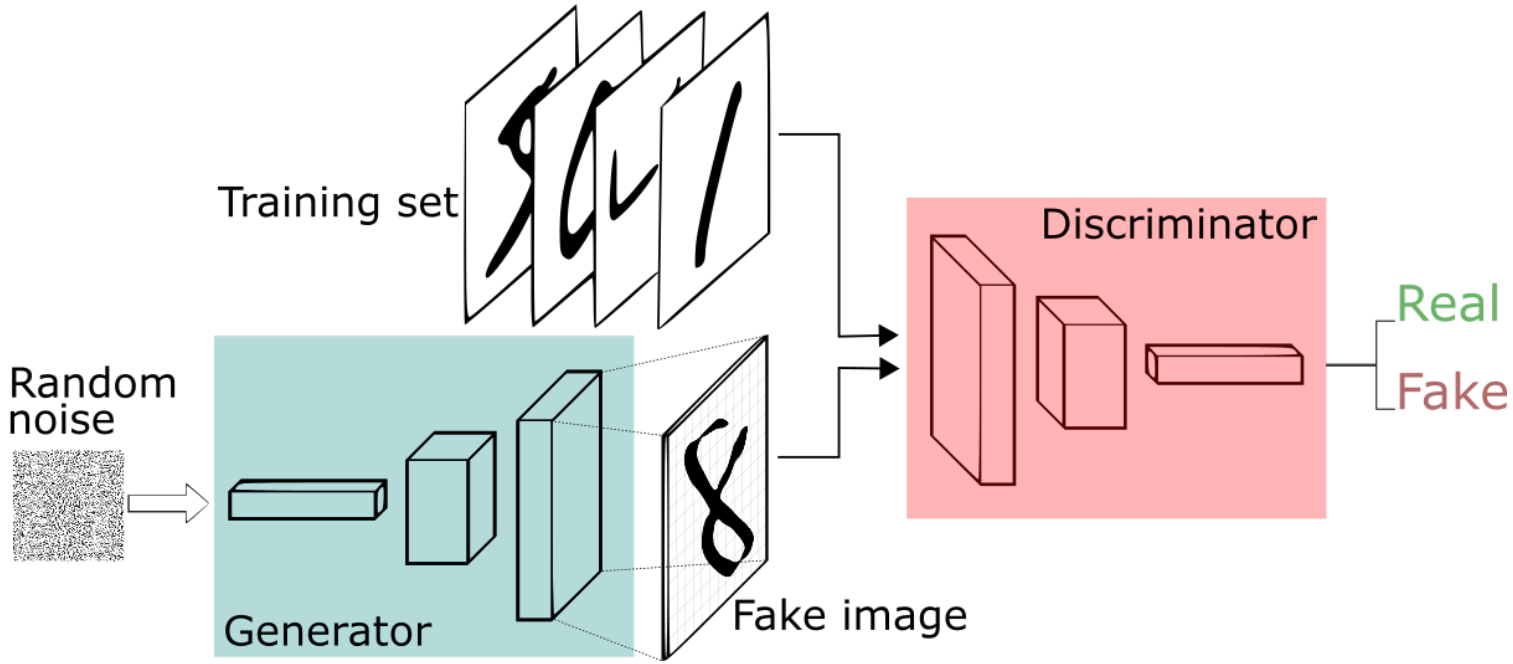
- RNNs usually encounter vanishing and exploding gradient problem.
- The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

Two wellknown architectures can solve this problem.

- Long Short Term Memory (LSTM) (1997).
- Gated Recurrent Unit (GRU) (2014).

Generative Adversial Networks

Generative adversarial networks (GANs) are architectures that use two neural networks, pitting one against the other (thus the adversarial) in order to generate new, synthetic instances of data that can pass for real data.



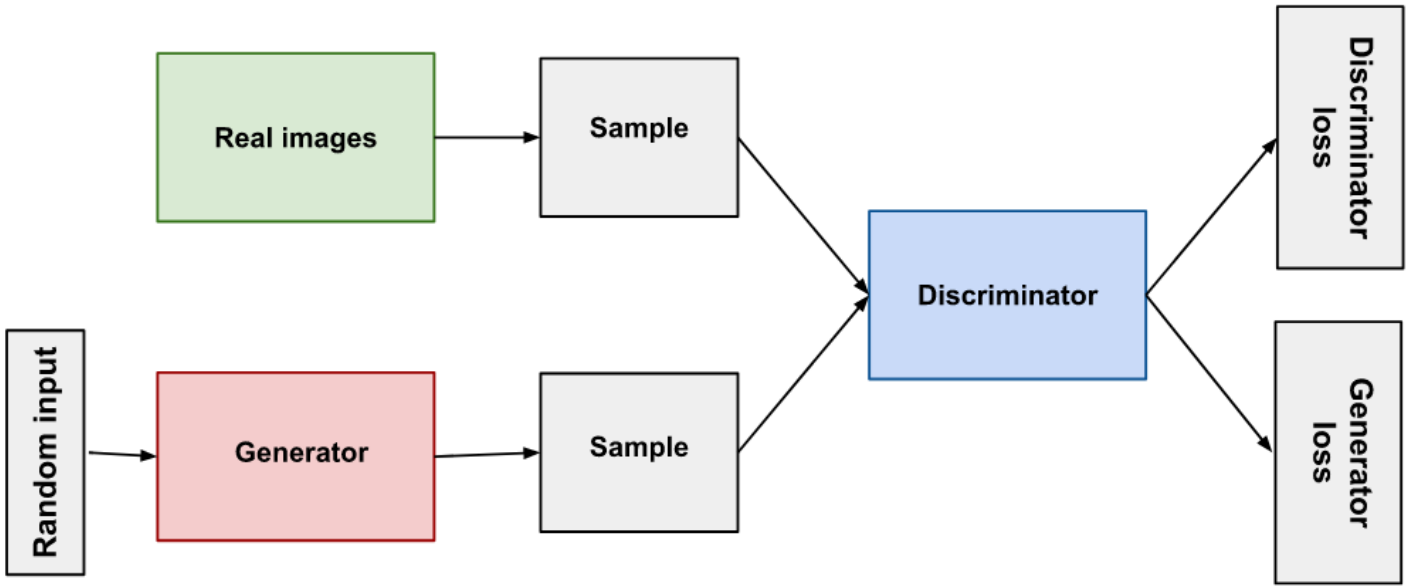
GAN Example[1]

[1] https://www.thispersondoesnotexist.com/

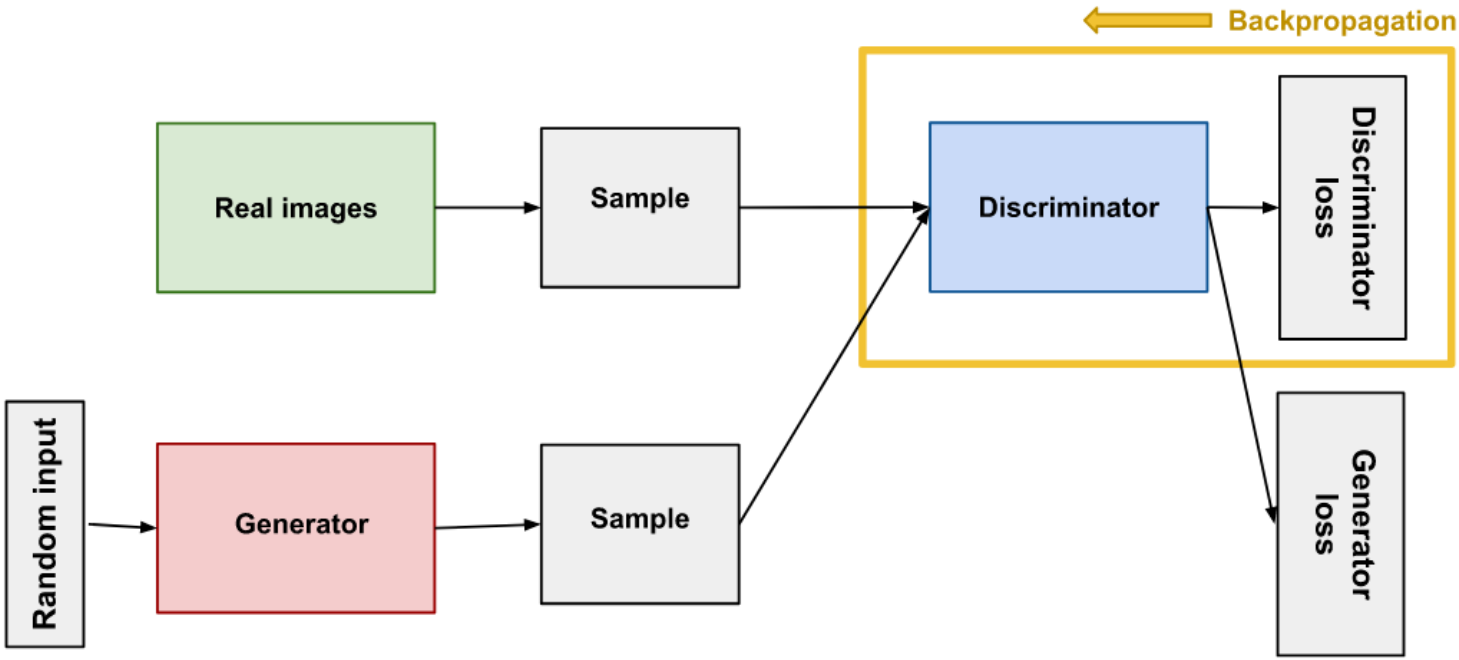
GANs steps

- When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake.
- As training progresses, the generator gets closer to producing output that can fool the discriminator.
- Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

GANs Loss


Real images
Sample
Discriminator loss
Discriminator
Random input
Generator
Sample
Generator loss

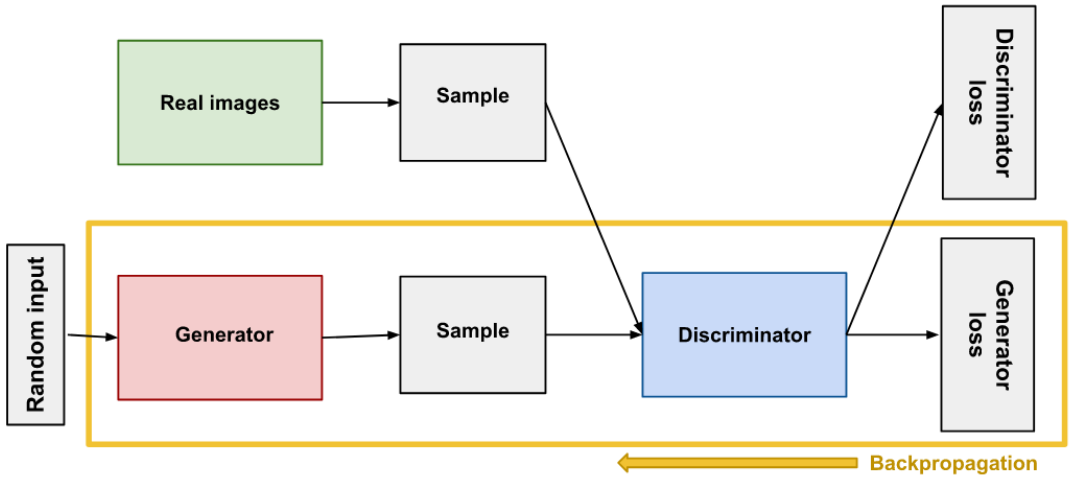
The Discriminator Training



The discriminator's training data comes from two sources:

- Real data instances, such as real pictures of people.
- Fake data instances created by the generator.

The Generator Training



Generator Training Steps:

1. Sample random noise.
2. Produce generator output from sampled random noise.
3. Get discriminator "Real" or "Fake" classification for generator output.
4. Calculate loss from discriminator classification.
5. Backpropagate through both the discriminator and generator to obtain gradients.
6. Use gradients to change only the generator weights.

Training GAN

Alternating Training

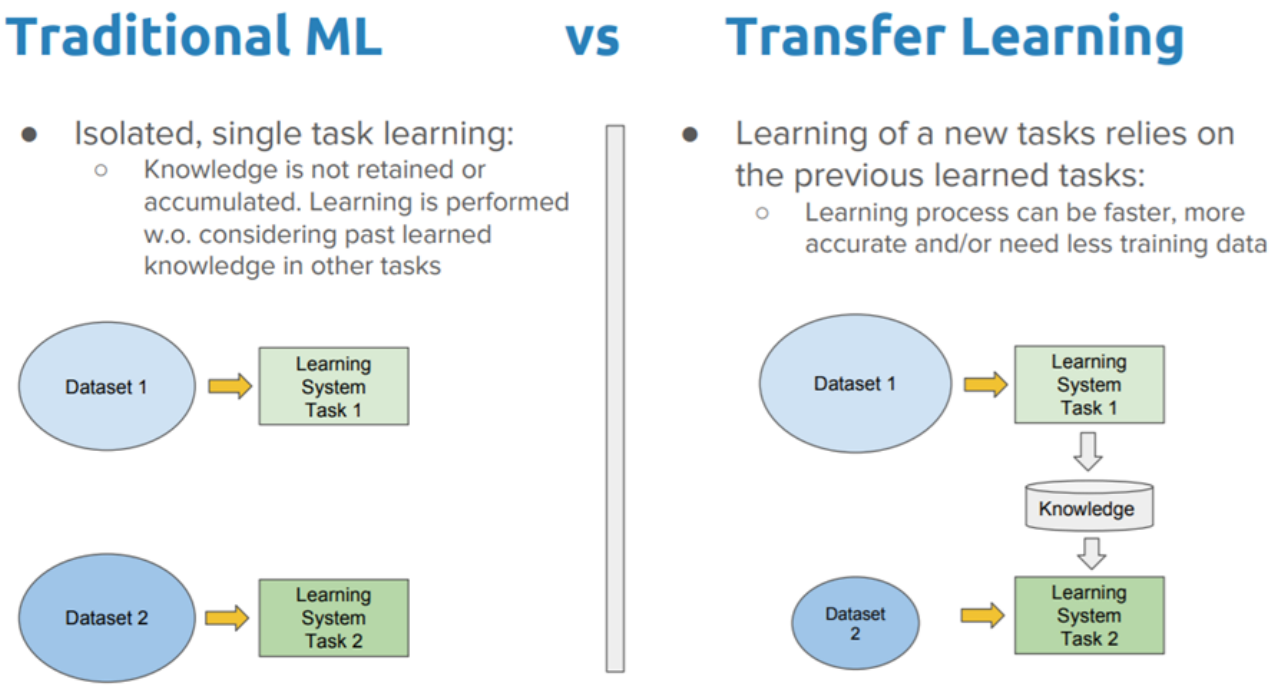
1. GAN training proceeds in alternating periods:
2. The discriminator trains for one or more epochs. The generator trains for one or more epochs. Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

What is Transfer Learning?

Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is re-used on a second related task.

- Only works in deep learning if the model features learned from the first task are general.

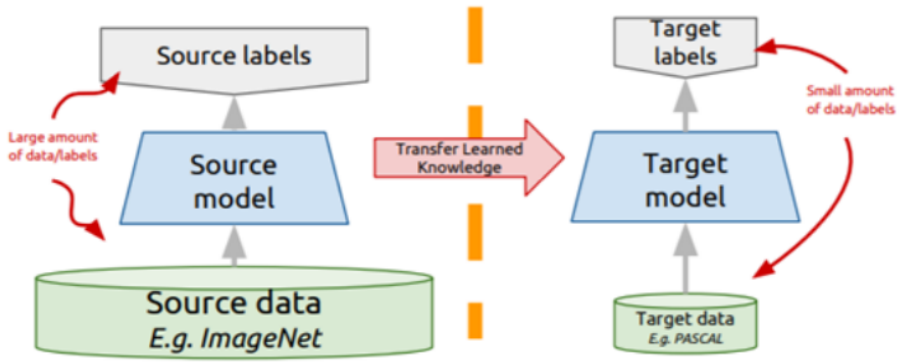


Transfer Learning Idea

- Take a network trained on different domain for a different Source Task.
- Adapt it to for your domain and Target Task

Variations

- Same domain, differnet task.
- Different domain, same task.

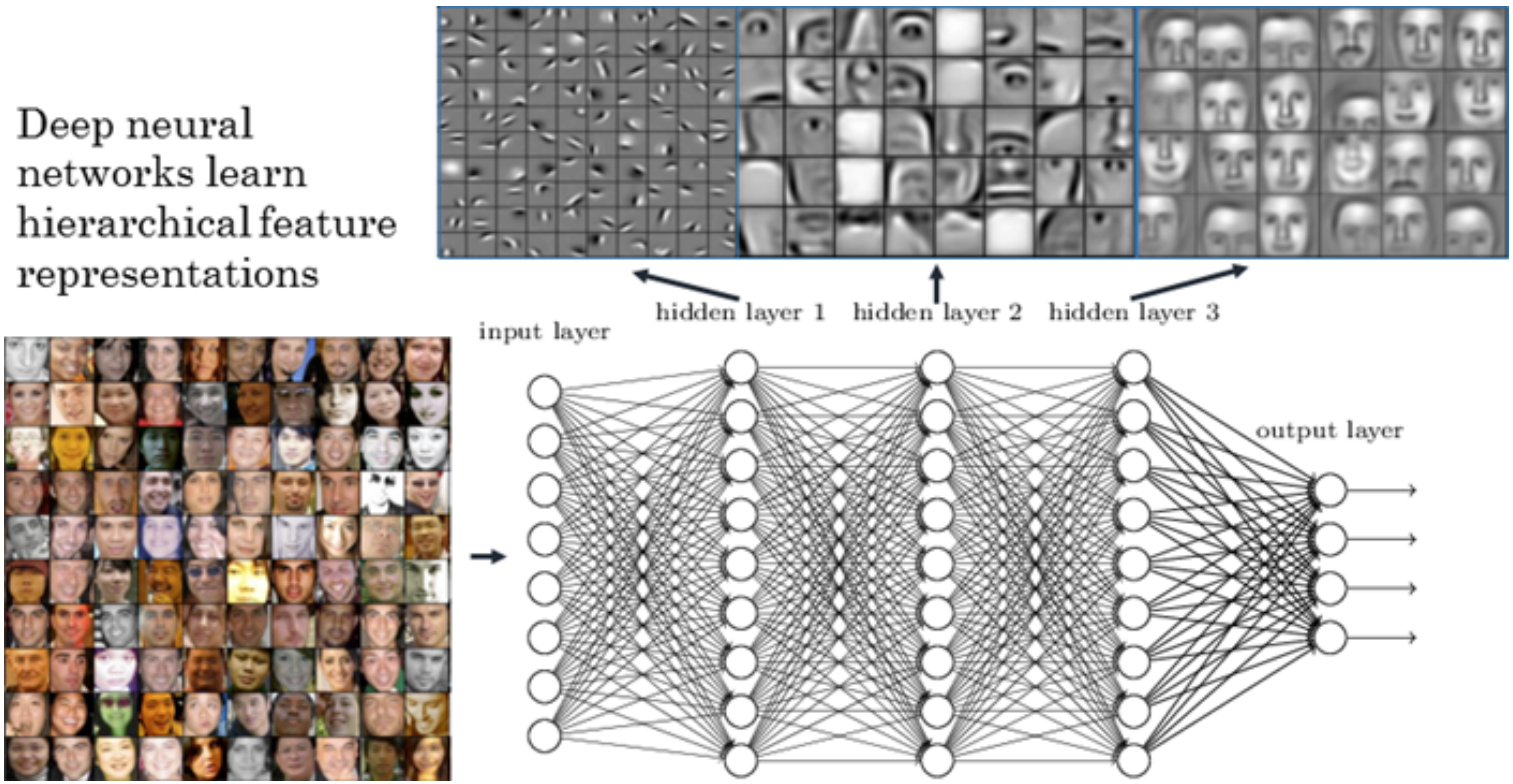


Why Transfer Learning?

- Alternative is to build model from scratch
  - Time cosuming
  - Hard feature engineering process.
- Can Lessen the data demands.
- Transfer Learning can be used for privacy.
- Can be used to improve a model performance.

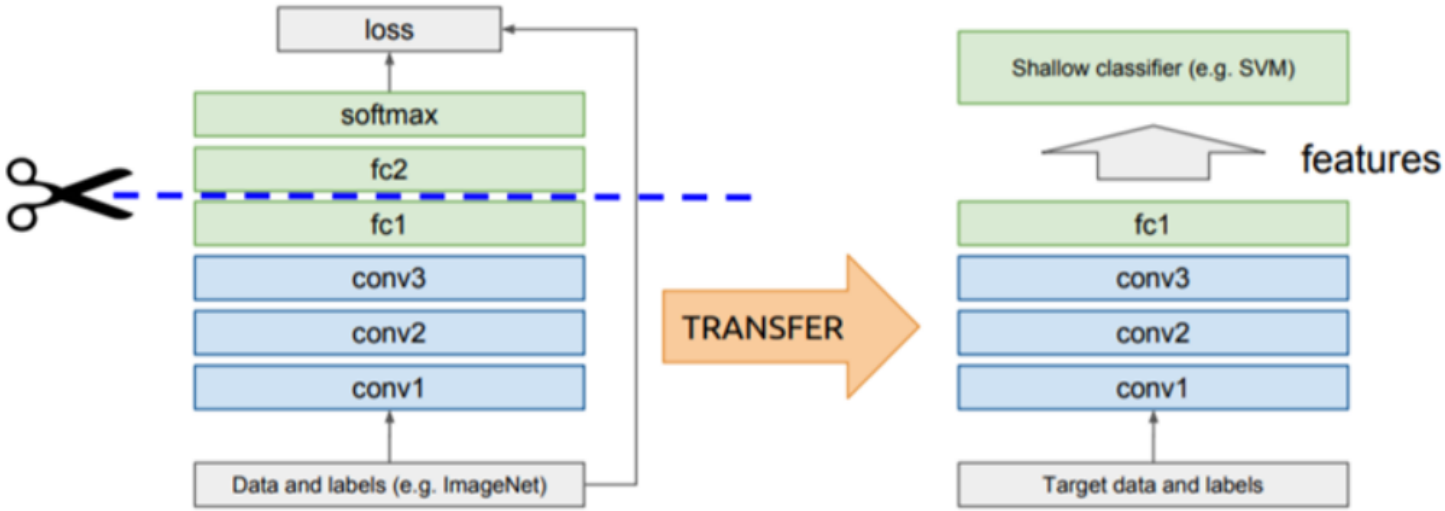
How does it work?

- Deep learning systems and models are layered architectures that learn different features at different layers.
- The initial layers have been seen to capture generic features, while the later ones focus more on the specific task at hand.



How does it work?

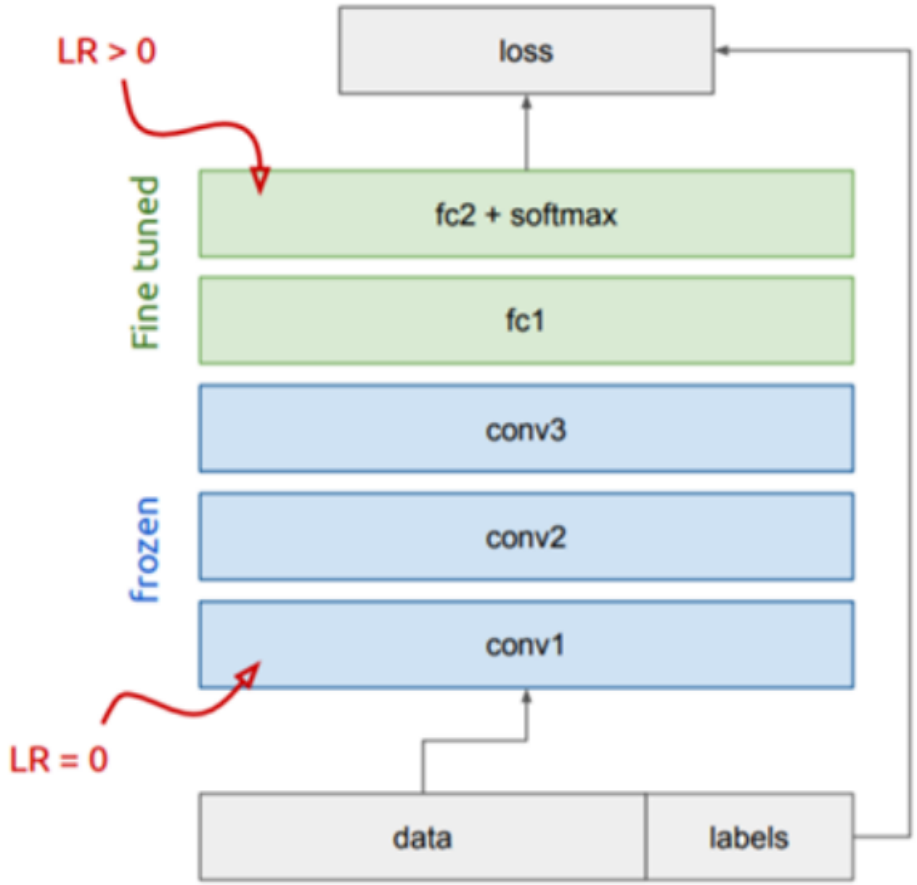
- The key idea is to leverage the pre-trained models weighted layers to extract features but not to update the weights of the models layers during training with new data for the new task.



Freeze or Fine-tune

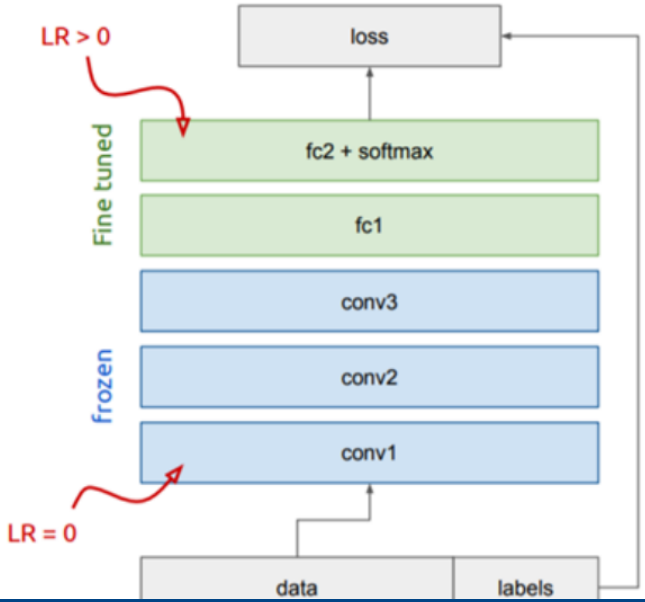
- **Freezed layers:** not updated during backprop.
- **Fine-tuned layer:** updated during backprop.

In general, we can set learning rates to be different for each layer. Our goal is retraining the head of a network to recognize classes it was not originally intended for.



Freeze and Fine-tune

1. If you have very small dataset you may freeze all layers except softmax in the source model and replace it and train only softmax of target model.
2. If you have small dataset you may freeze large number of layers in the source model while train low number of layers.
3. If you have large dataset you may freeze low number of layers while train more number of layers.



Deep Learning Frameworks (1/2)

- TensorFlow 
  - Googles Tensorflow the most popular Deep Learning.
  - TF needs a lot of coding.
- PyTorch 
  - Was developed for Facebook services
  - In PyTorch, you can use standard debuggers, for example, pdb or PyCharm.
- Sonnet 
  - Built on top of TensorFlow.
  - High-level object-oriented libraries.
- Keras 
  - The best Deep Learning framework for those who are just starting out.
  - High level like Sonnet.
- MXNet 
  - Effectively parallel on multiple GPUs and many machines..

Deep Learning Frameworks (2/2)

- Gluon 
  - Gluon is based on MXNet and offers a simple API that simplifies the creation of deep learning models.
- Swift 
  - Swift for Tensorflow.
- Chainer 
  - The code is written in pure Python on top of the Numpy and CuPy libraries. (fast)
- DL4J 
  - Deep Learning for Java.
- ONNX 
  - Enables models to be trained in one framework and transferred to another for inference.

Questions