CS M148 –

Data Science Fundamentals

Lecture #13: Neural Networks

Baharan Mirzasoleiman

UCLA Computer Science

Announcements

HW 1

- Grades are posted

HW 2

- Extended, due Friday Feb 18 at 11pm

HW 3

- Is posted, due Friday Feb 25 at 11pm

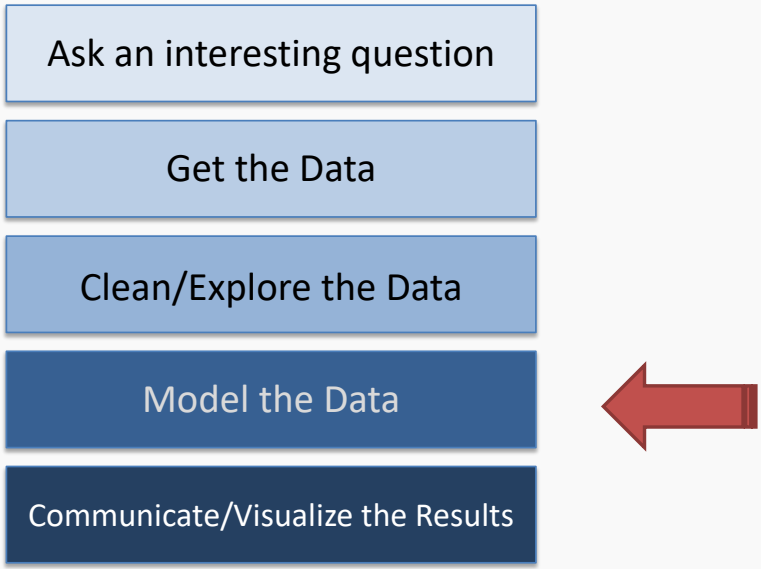
Survey

shorturl.at/fxyzS

Let’s quickly review what we saw last time

Still here!

The Data Science Process



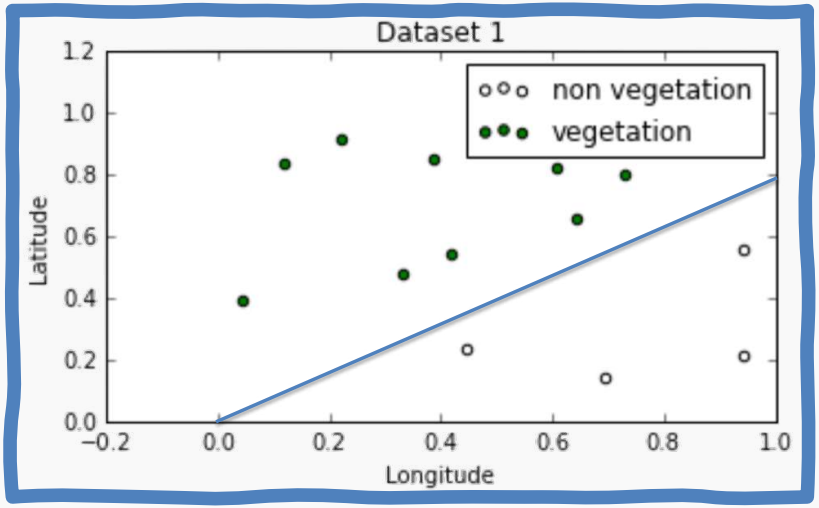
Classification & Regression

Outline

- **Review of Decision Trees**
- Neural Networks

Geometry of Data

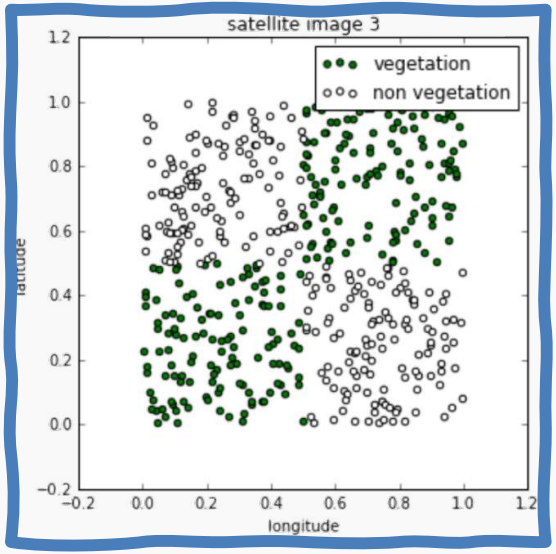
Question: Can you guess the equation that defines the decision boundary below?



$-0.8x_1 + x_2 = 0 \implies x_2 = 0.8x_1 \Rightarrow \ Latitude = 0.8\ Lon$

Geometry of Data

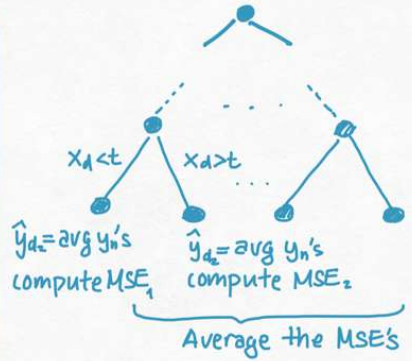
Complicate decision boundaries can not be explained with LogRegression.



Decision Trees

To learn a decision tree model, we take a greedy approach:

1. Start with a node containing all the data.
2. If stopping condition is not met:
   A. Choose the 'optimal' predictor and threshold and divide the data in the node into two sets.
3. For each new node, repeat step 2.

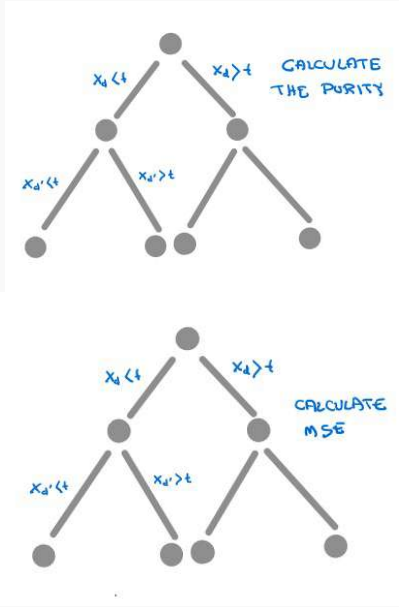


Decision Trees Summary: Splitting Criteria

Splitting Criteria:

For classification, purity of the regions is a good indicator the performance of the model. Entropy as a splitting criterial minimizes the cross-entropy (greedy). Gini is also a splitting criteria.

For regression, we want to select a splitting criterion that promotes splits that improves the predictive accuracy of the model as measured by the MSE

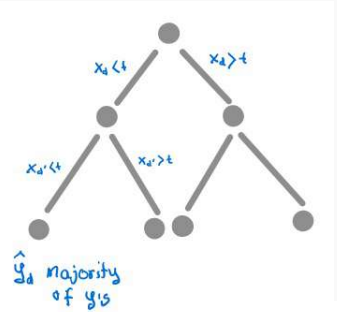


Decision Trees Summary : Prediction

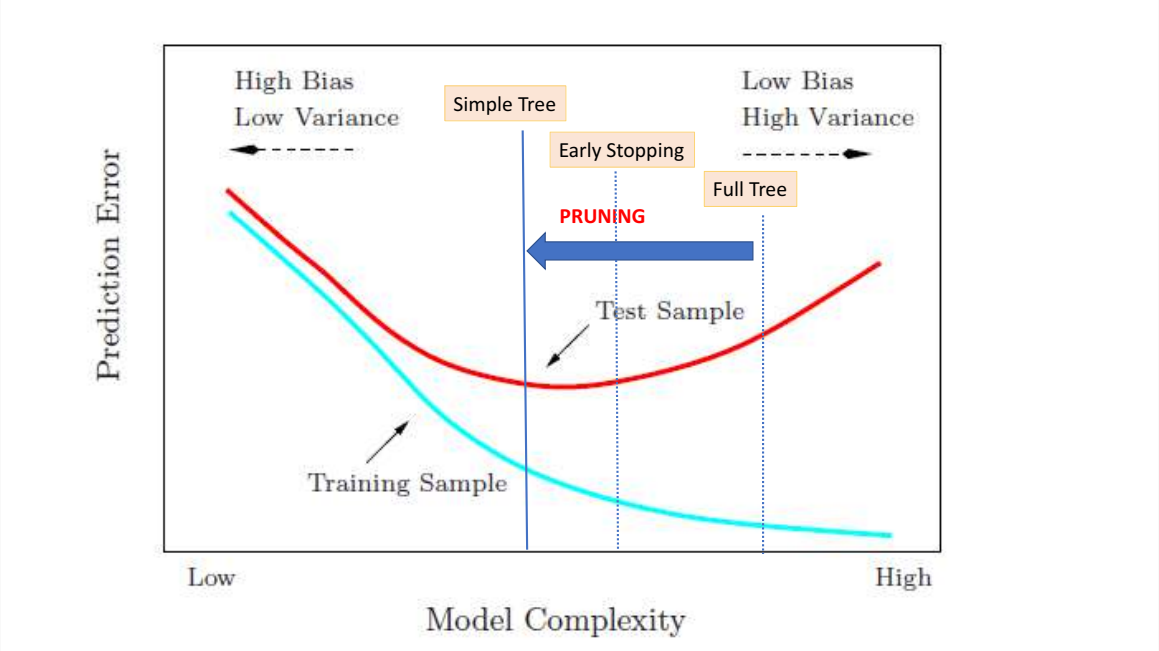
Prediction:

For classification, we label each region in the model with the label of the class to which the **plurality** of the points within the region belong.

For regression, we predict with the **average** of the output values of the training points contained in the region.



Motivation for Pruning



Limitations of Decision Tree Models

Decision trees models are highly interpretable and fast to train, using our greedy learning algorithm.

However, to **capture a complex decision boundary** (or approximate a complex function), we need to use a large tree (since each time we can only do axis-aligned splits).

We've seen that large trees have high variance and are prone to overfitting.

For these reasons, in practice, decision tree models often underperform when compared with other classification or regression methods.

Bagging

One way to adjust for the high variance of the output of an experiment is to perform the experiment multiple times and then average the results.

The same idea can be applied to high variance models:

1. **Bootstrap:** we generate multiple samples of training data, via bootstrapping. We train a deeper decision tree on each sample of data.
2. **Aggregate:** for a given input, we output the averaged outputs of all the models for that input.

This method is called ***Bagging*** (Breiman, 1996), short for, of course, **B**ootstrap **Agg**regating.

For classification, we return the class that is outputted by the plurality of the models. For regression we return the average of the outputs for each tree.

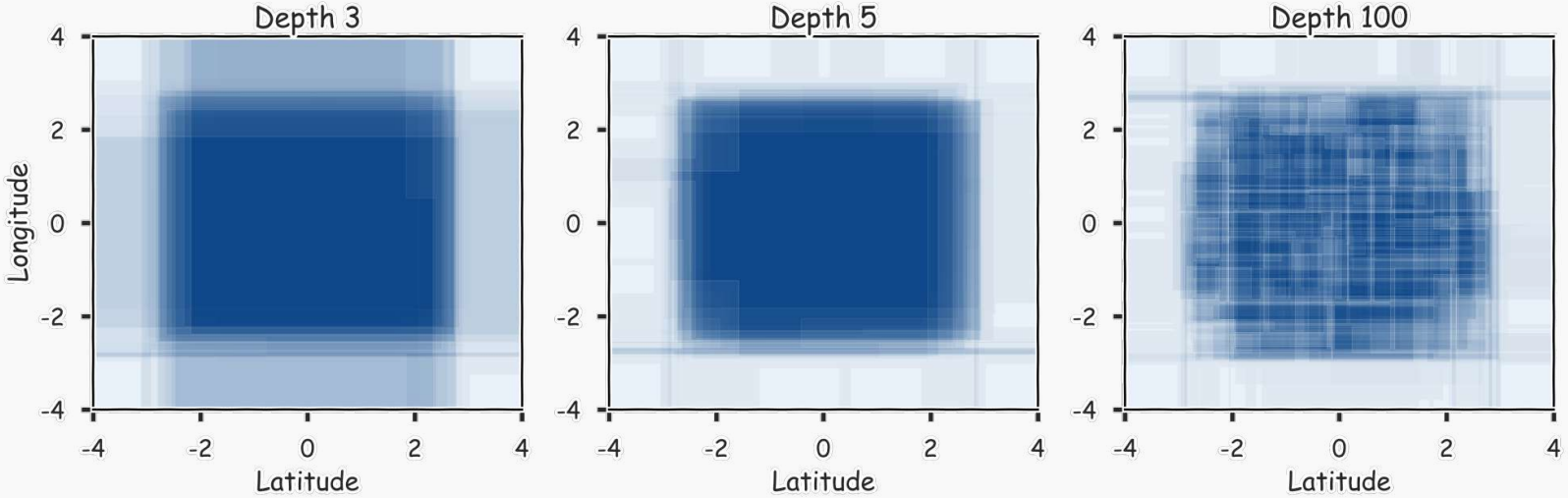
Bagging

Bagging enjoys the benefits of:

1. High expressiveness - by using deeper trees each model is able to approximate complex functions and decision boundaries.

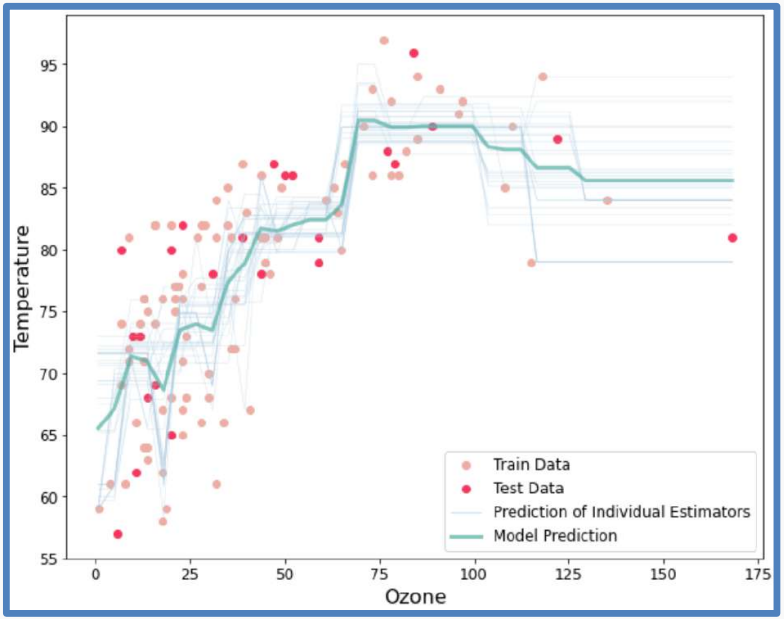
2. Low variance - averaging the prediction of all the models reduces the variance in the final prediction, assuming that we choose a sufficiently large number of trees.

300 magic realisms



Bagging (regression)

The resulting tree is the average of all tree (estimators).



Bagging

Question: Do you see any problems?

- If trees are too shallow it can still underfit.
- Still some overfitting if the trees are too large.
- Interpretability:

  The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

Question: Do you see any problems?

- If trees are too shallow it can still underfit.
- Still some overfitting if the trees are too large.

Cross Validation

Out-of-Bag Error

Bagging is an example of an ***ensemble method***, a method of building a single model by training and aggregating multiple models.

With ensemble methods, we get a new metric for assessing the predictive performance of the model, the ***out-of-bag error***.

Given a training set and an ensemble of models, each trained on a bootstrap sample, we compute the ***out-of-bag error*** of the averaged model by

1. For each point in the training set, we average the predicted output for this point over the models whose bootstrap training set excludes this point. We compute the error or squared error of this averaged prediction. Call this the point-wise out-of-bag error.

2. We average the point-wise out-of-bag error over the full training set.

Bagging

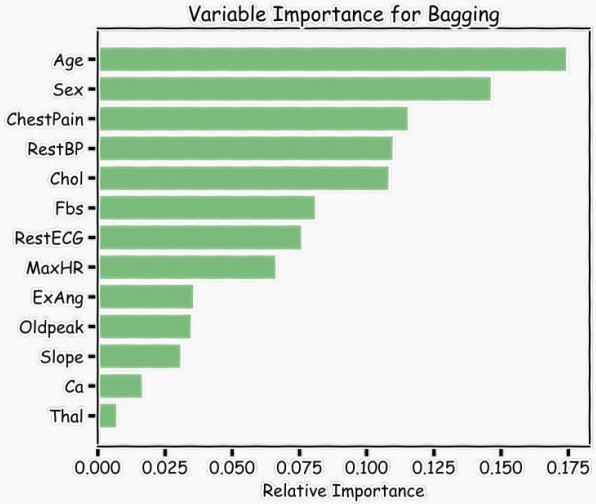
Question: Do you see any problems?

- If trees are too shallow it can still underfit.
- Still some overfitting if the trees are too large.
- Interpretability:

  The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

Variable Importance for Bagging

Calculate the total amount that the MSE (for regression) or Gini index (for classification) is decreased due to splits over a given predictor, averaged over all B trees.



100 trees, max_depth=10

Improving on Bagging

In practice, the ensembles of trees in Bagging tend to be highly correlated.

Suppose we have an extremely strong predictor, x_j , in the training set amongst moderate predictors. Then the greedy learning algorithm ensures that most of the models in the ensemble will choose to split on x_j in early iterations.

However, we assumed that each tree in the ensemble is independently and identically distributed, with the expected output of the averaged model the same as the expected output of any one of the trees.

Random Forests

A Random Forest is a modified form of bagging that creates ensembles of independent decision trees.

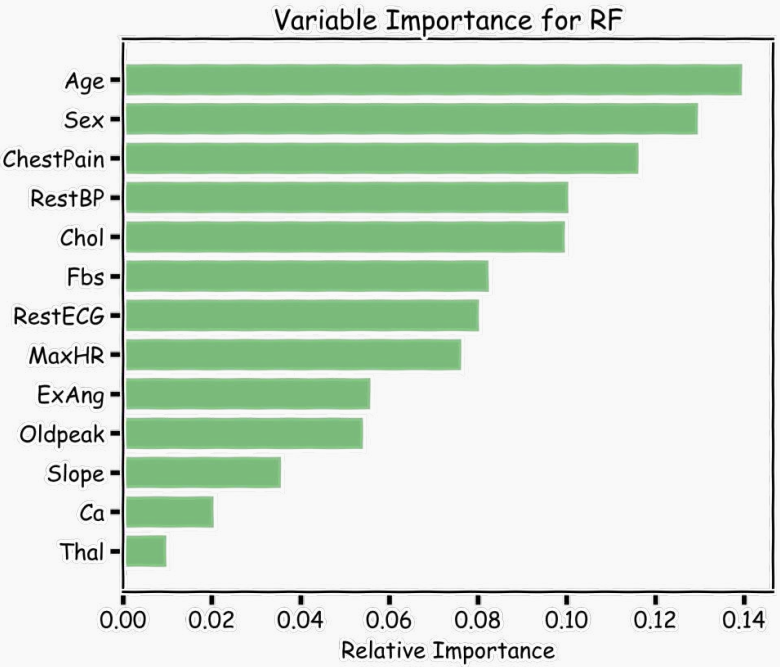
To decorrelate the trees, we:

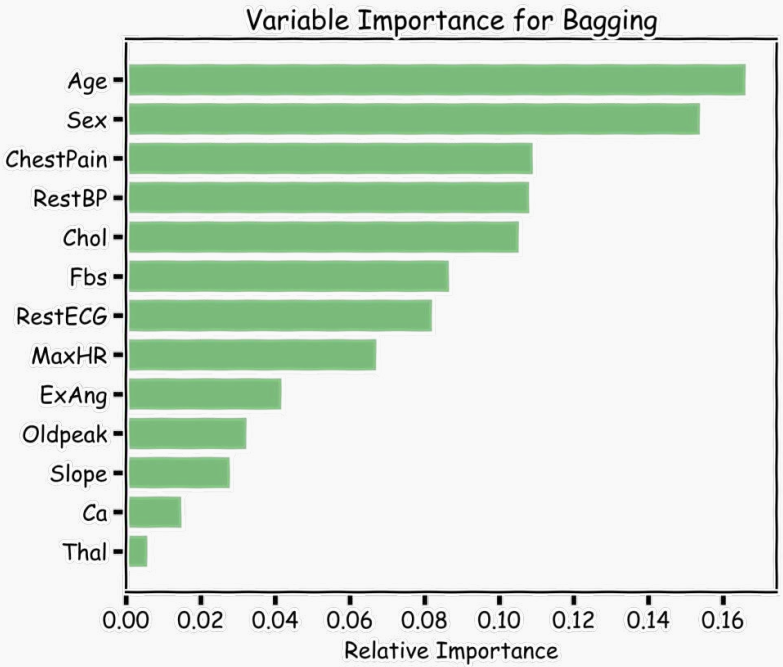
1. train each tree on a separate bootstrap sample of the full training set (same as in bagging).

2. for each tree, at each split, we ***randomly*** select a set of J predictors from the full set of predictors.

3. From amongst the J predictors, we select the optimal predictor and the optimal corresponding threshold for the split.

Variable Importance for RF





100 trees, max_depth=10

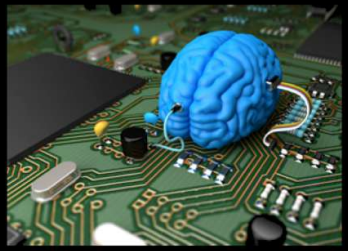
Outline

1. Introduction to Artificial Neural Networks
2. Review of basic concepts
3. Single Neuron Network (‘Perceptron’)
4. Multi-Layer Perceptron (MLP)

Neural Networks

Deep Learning

What society thinks I do



What my friends think I do

What other computer scientists think I do

What mathematicians think I do

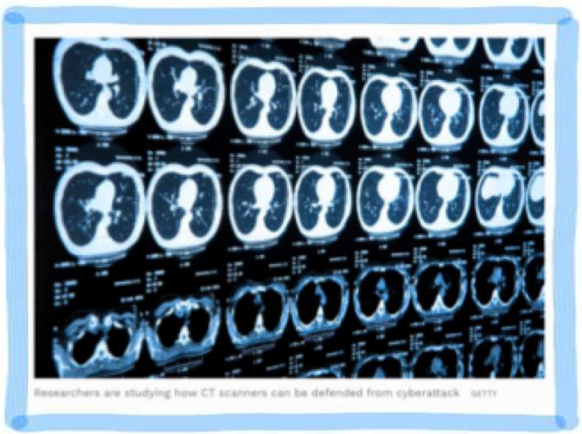
What I think I do

```
In [1]:
import keras
Using TensorFlow backend.
```

What I actually do

Today’s news

Stopping Cyberattacks



Researchers are studying how CT scanners can be defended from cyberattack GETTY

Detecting tampering with the diagnostic images, or quietly upped the radiation levels.

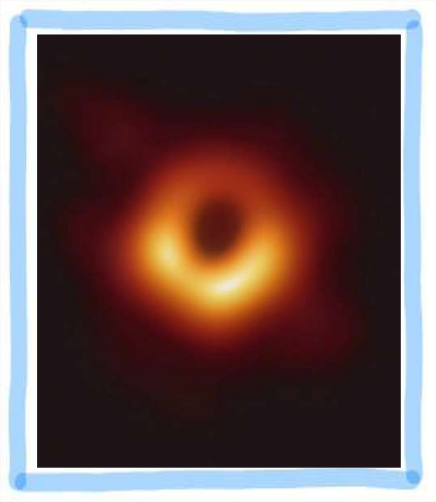
Skin Conditions

iStock-safa uzel

Using Deep Learning in diagnosing skin conditions

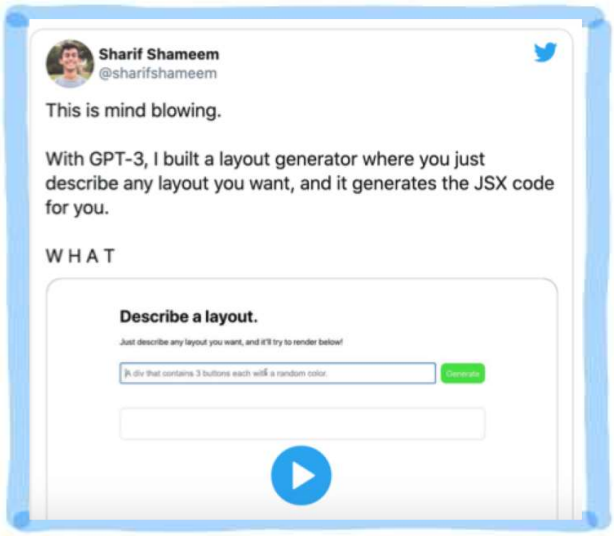
Today’s news

Image generation



Katie Bouman’s CHIRP produces the first-ever image of a black hole.

Computer Code Generation



The Potential of Data Science

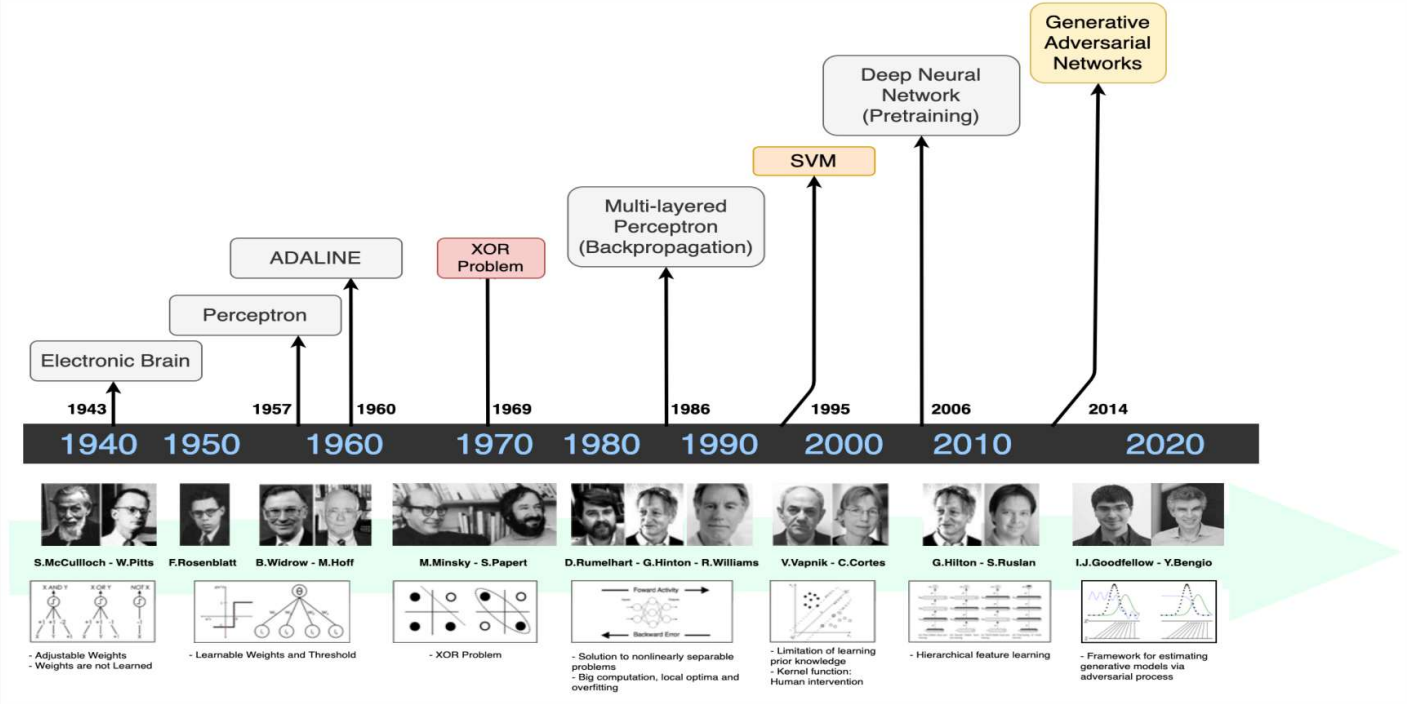
Gender Bias

Some DS models for evaluate job applications show bias in favor of male candidate

Racial Bias

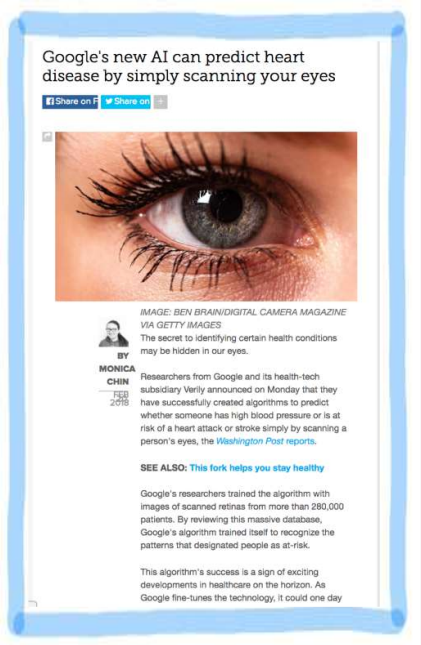
Risk models used in US courts have shown to be biased against non-white defendants

Historical Trends



Historical Trends

Disease prediction



Google's new AI can predict heart disease by simply scanning your eyes

Share on F Share on

IMAGE: BEN BRAIN/DIGITAL CAMERA MAGAZINE VIA GETTY IMAGES

BY MONICA CHIN FEB 2018

The secret to identifying certain health conditions may be hidden in our eyes.

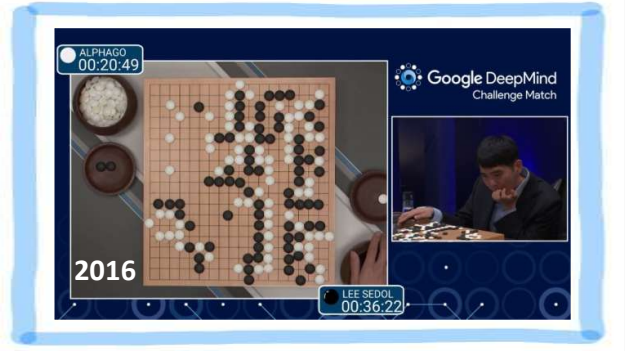
Researchers from Google and its health-tech subsidiary Verily announced on Monday that they have successfully created algorithms to predict whether someone has high blood pressure or is at risk of a heart attack or stroke simply by scanning a person's eyes, the *Washington Post* reports.

SEE ALSO: This fork helps you stay healthy

Google's researchers trained the algorithm with images of scanned retinas from more than 280,000 patients. By reviewing this massive database, Google's algorithm trained itself to recognize the patterns that designated people as at-risk.

This algorithm's success is a sign of exciting developments in healthcare on the horizon. As Google fine-tunes the technology, it could one day

Game strategy



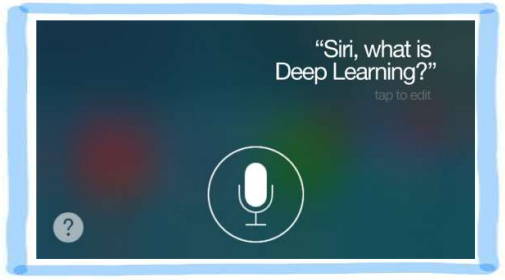
DeepMind

AlphaZero AI beats champion chess program after teaching itself in four hours

Google's artificial intelligence sibling DeepMind repurposes Go-playing AI to conquer chess and shogi without aid of human knowledge

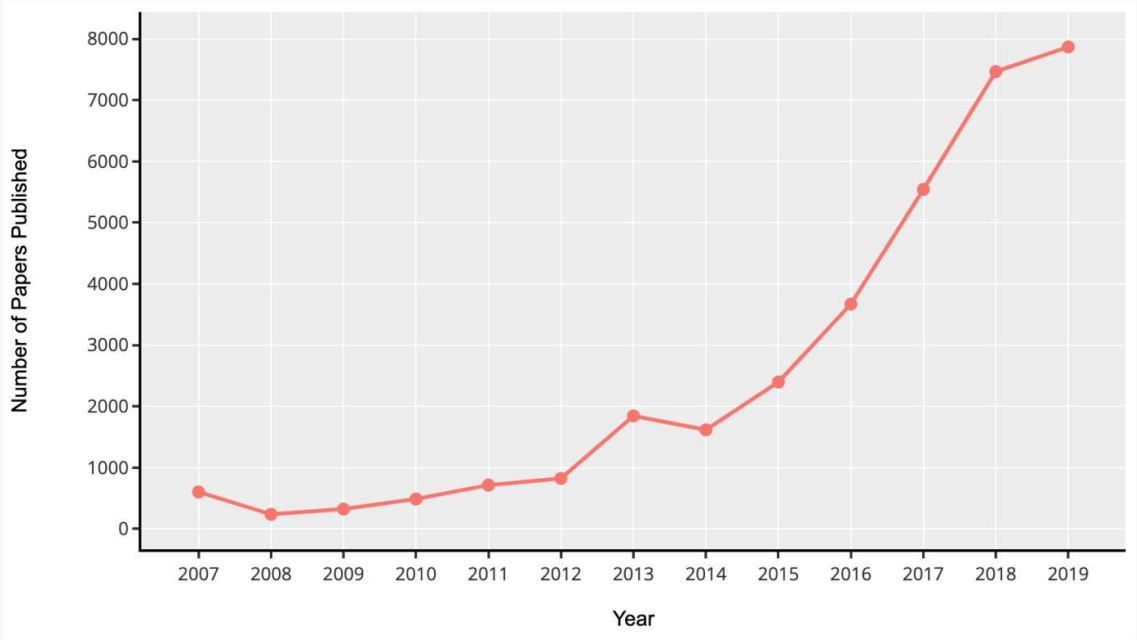
2018

Natural Language Processing



Historical Trends

ArXiv papers on Machine Learning and Artificial Intelligence: 2007-2019



Outline

1. Introduction to Artificial Neural Networks
2. **Review of basic concepts**
3. Single Neuron Network (‘Perceptron’)
4. Multi-Layer Perceptron (MLP)

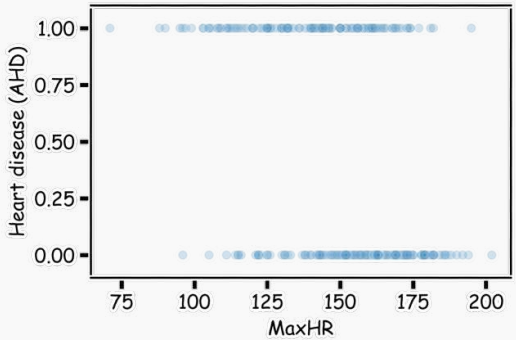
Classification example: Heart Data

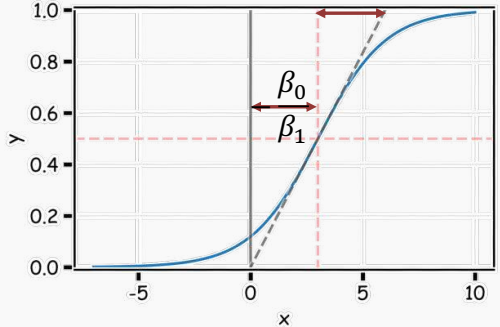
response variable ***Y*** is Yes/No

Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

Heart Data: logistic estimation

We'd like to predict whether or not a person has a heart disease. And we'd like to make this prediction, for now, just based on the MaxHR.





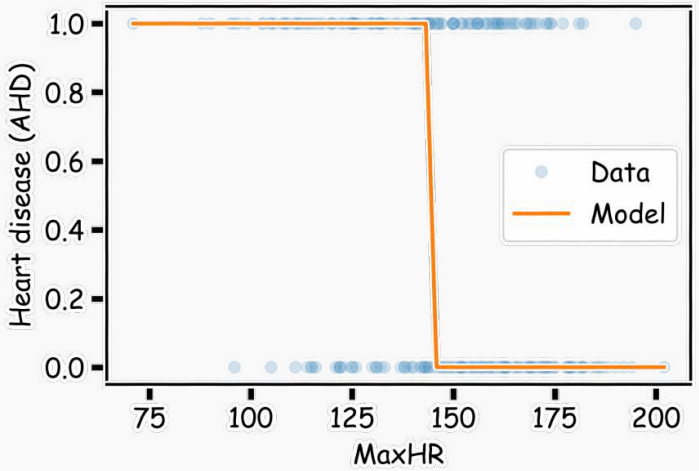
The logistic regression model uses a function, called the ***logistic*** function, to model $P(y = 1)$:

$$P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

Logistic Regression

Find the coefficients that minimize the loss function

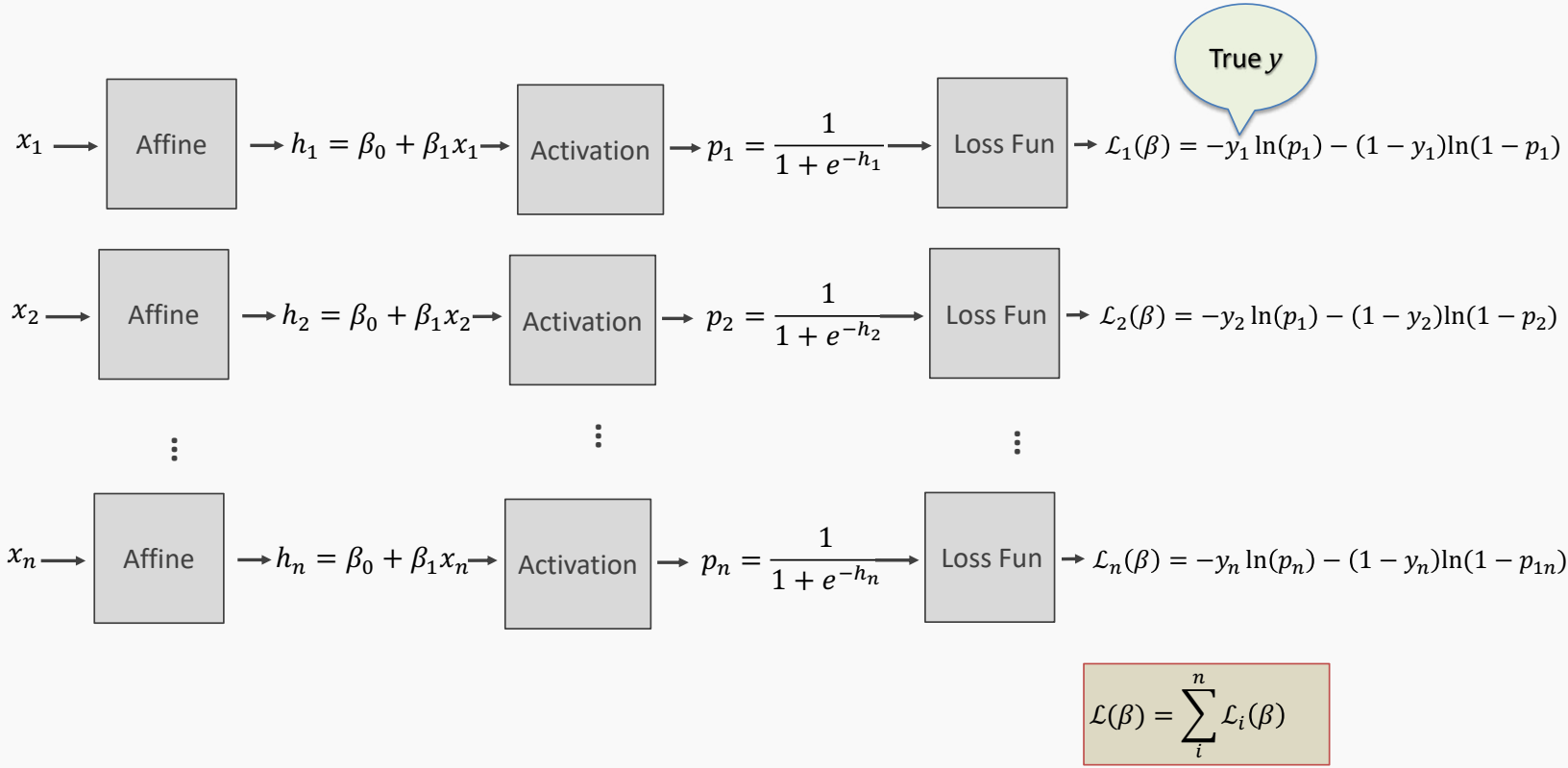
$$\mathcal{L}(\beta_0, \beta_1) = -\sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$



Outline

1. Introduction to Artificial Neural Networks
2. Review of basic concepts
3. **Single Neuron Network (‘Perceptron’)**
4. Multi-Layer Perceptron (MLP)

Logistic Regression Revisited



Build our first ANN

n_p: number of predictors
n_o: number of observations

matrix: $n_o \times n_p$
vector: $n_o \times 1$
vector: $n_p \times 1$
scalar

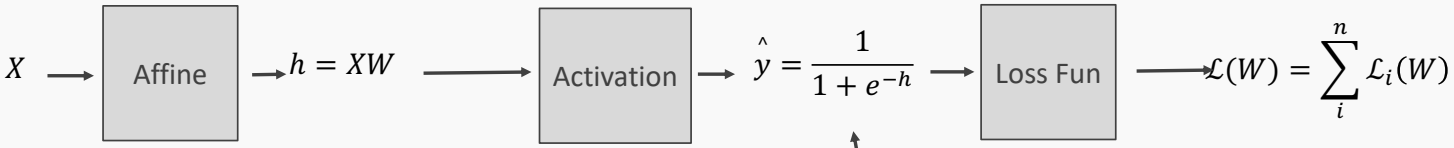
$X \rightarrow$ Affine $\rightarrow h = \beta_0 + X\beta_1 \rightarrow$ Activation $\rightarrow p = \frac{1}{1+e^{-h}} \rightarrow$ Loss Fun $\rightarrow \mathcal{L}(\beta) = \sum_i^n \mathcal{L}_i(\beta)$

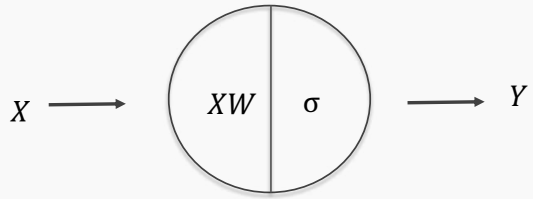
$X \rightarrow$ Affine $\rightarrow h = XW + b \rightarrow$ Activation $\rightarrow \hat{y} = \frac{1}{1+e^{-h}} \rightarrow$ Loss Fun $\rightarrow \mathcal{L}(\beta) = \sum_i^n \mathcal{L}_i(\beta)$

$X \rightarrow$ Affine $\rightarrow h = XW \rightarrow$ Activation $\rightarrow \hat{y} = \frac{1}{1+e^{-h}} \rightarrow$ Loss Fun $\rightarrow \mathcal{L}(\beta) = \sum_i^n \mathcal{L}_i(\beta)$

$$X = \begin{bmatrix} 1 & X_{11} & \dots & X_{1p} \\ 1 & \vdots & \dots & \vdots \\ 1 & X_{o1} & \dots & X_{op} \end{bmatrix} \quad \mathrm{W} = \begin{bmatrix} b \\ W_1 \\ \vdots \\ W_p \end{bmatrix}$$

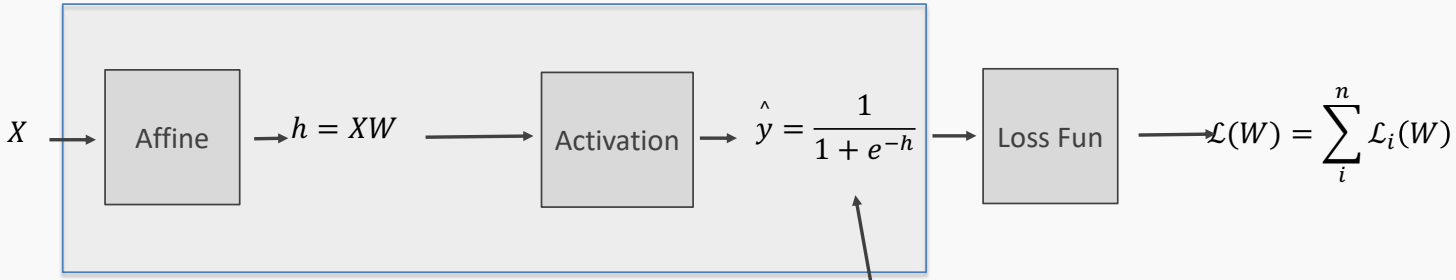
Build our first ANN

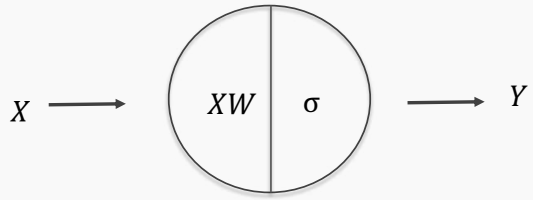




Single Neuron Network aka Perceptron

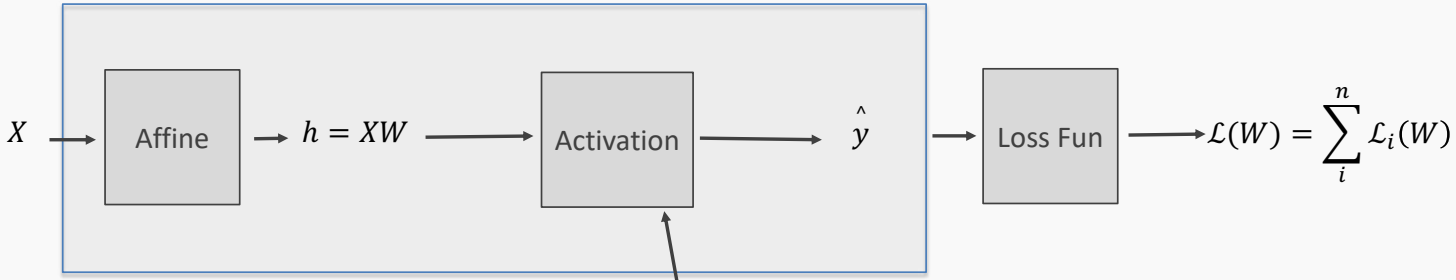
Build our first ANN





Single Neuron Network aka Perceptron

Build our first ANN



Algorithm: Perceptron Learning Algorithm

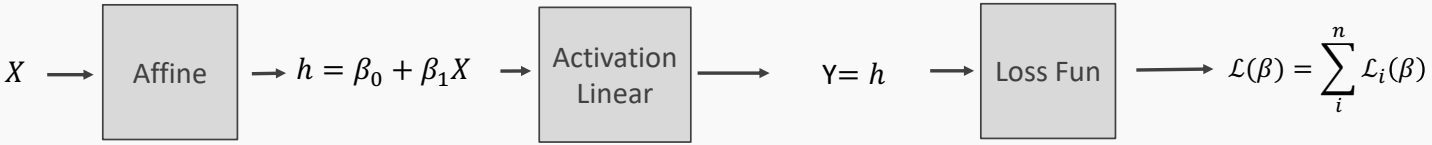
$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

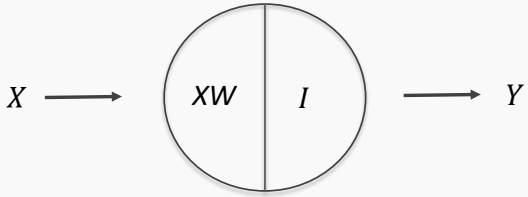
- Activation function of a perceptron doesn't need to be sigmoid
- Traditionally, perceptron has a step function as activation
- For a step function, perceptron update rule is shown here
  - Stops as soon as all the points are classified correctly
  - Doesn't try to find the best linear separator

Single Neuron Network aka Perceptron

Up to this point we just re-branded logistic regression to look like a neuron.

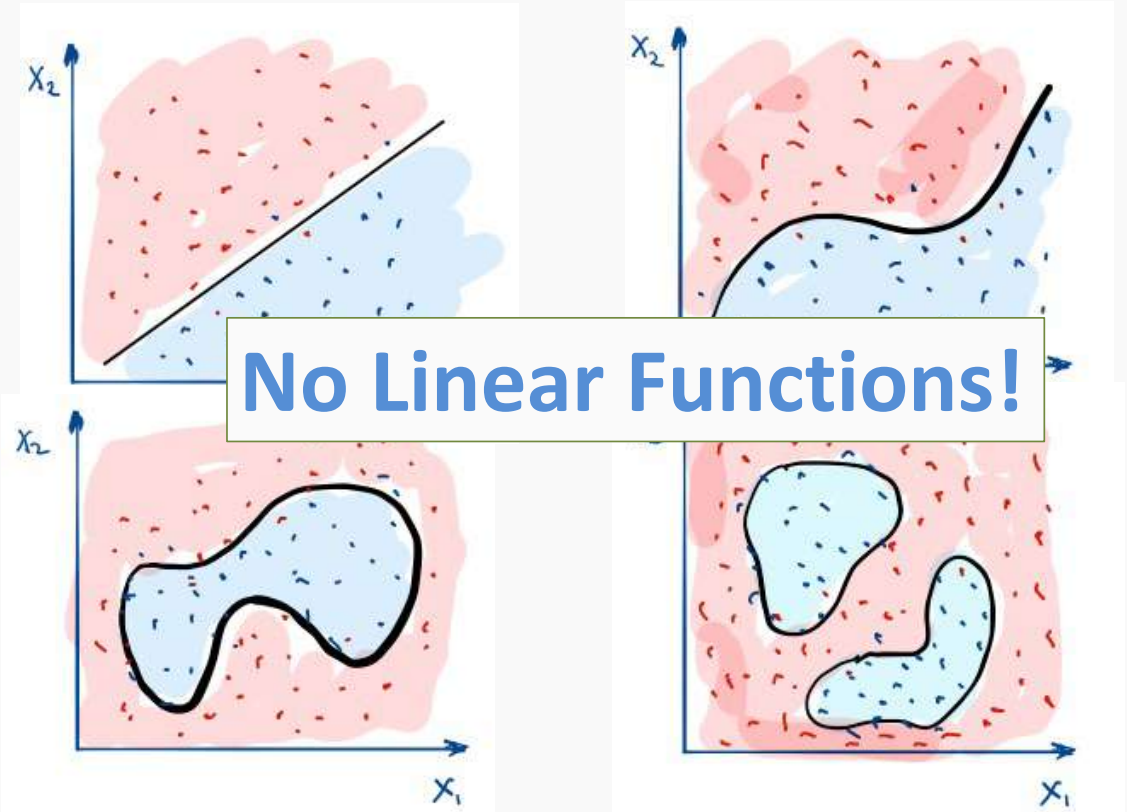
How about regression?



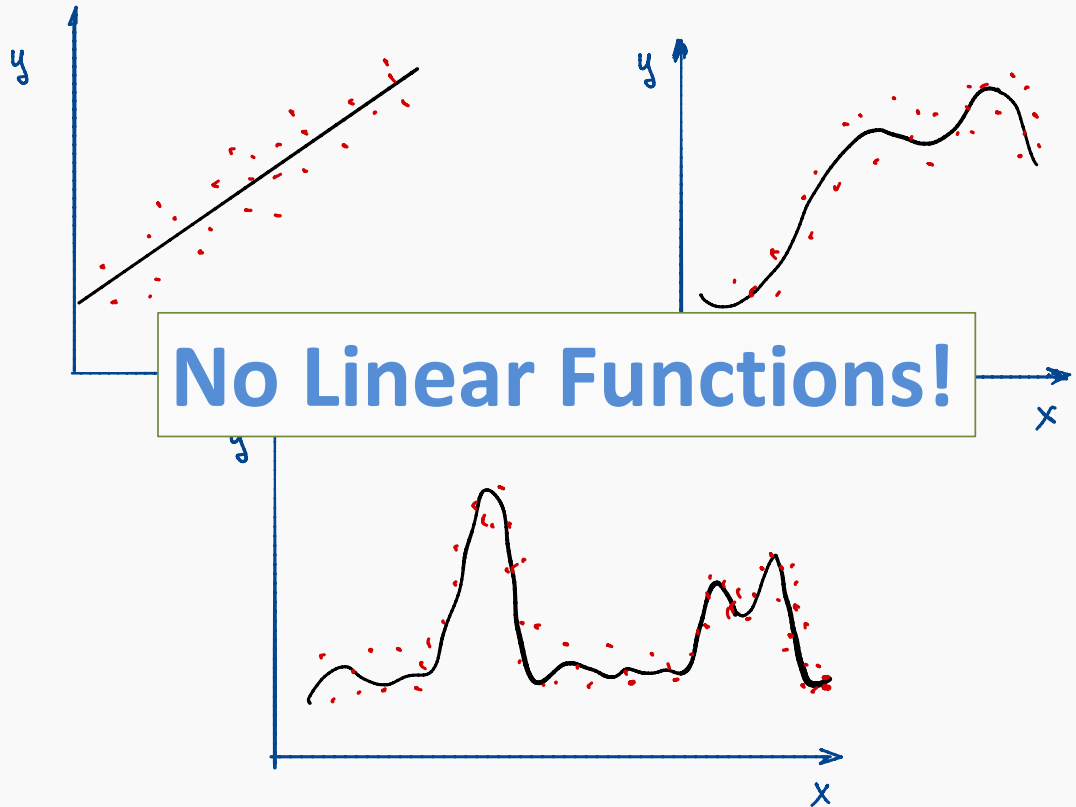


Where I is the identity function

So what's the big deal about Neural Networks?



For regression?



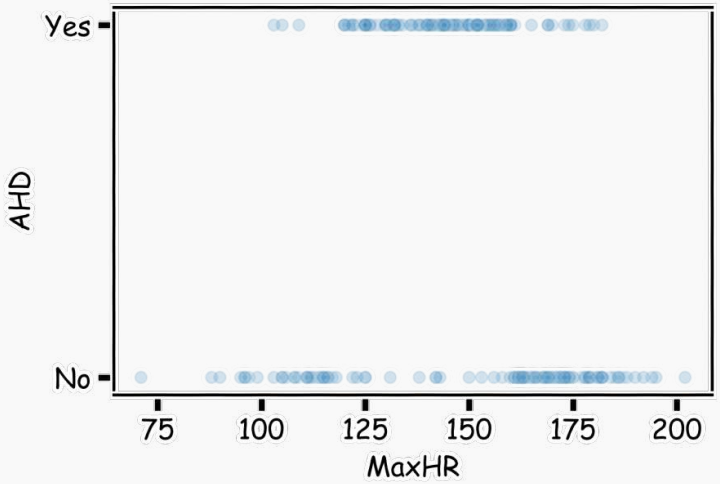
No Linear Functions!

Outline

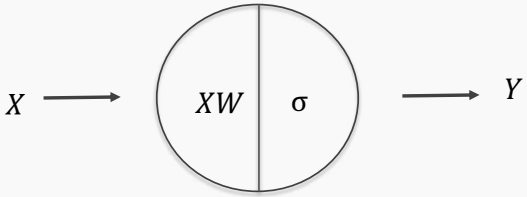
1. Introduction to Artificial Neural Networks
2. Review of Classification and Logistic Regression
3. Single Neuron Network (‘Perceptron’)
4. **Multi-Layer Perceptron (MLP)**

Example Using Heart Data

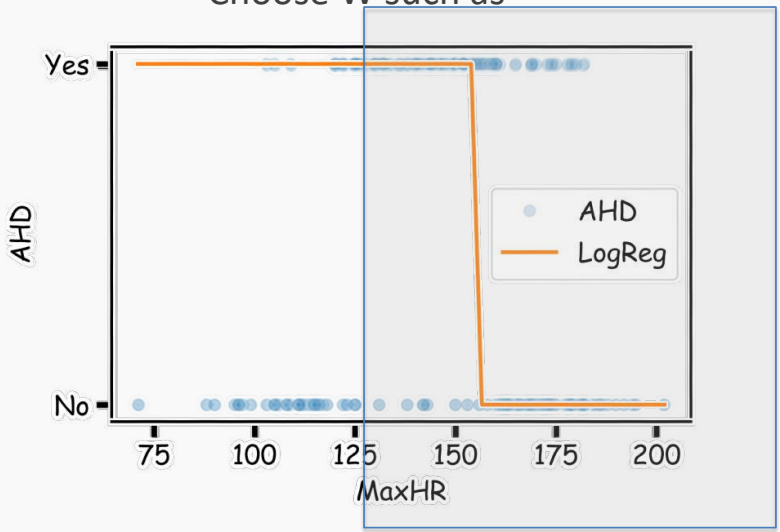
Slightly modified data to illustrate concepts.



Example Using Heart Data

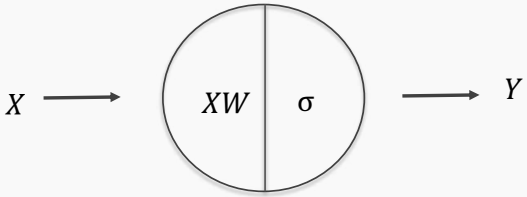


Choose W such as



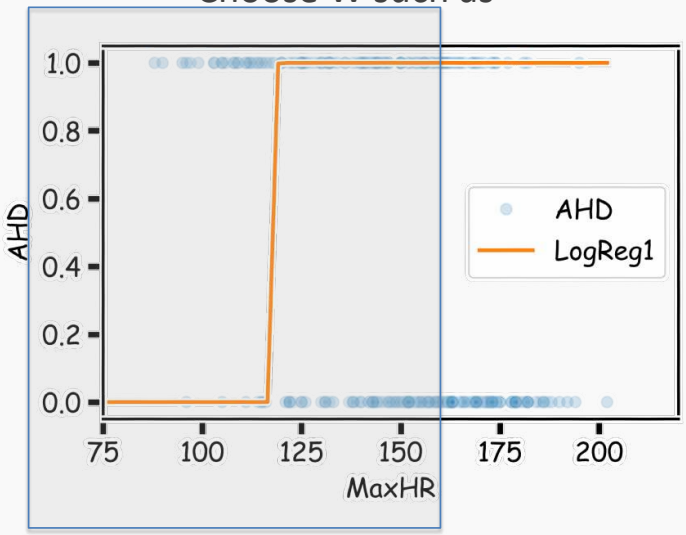
Right part of data are fitted well

Example Using Heart Data



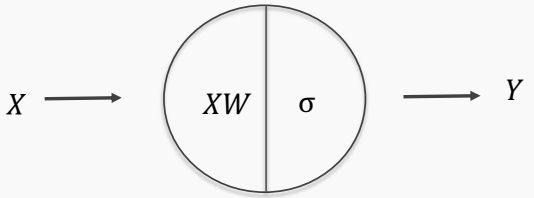
Left part of data are fitted well

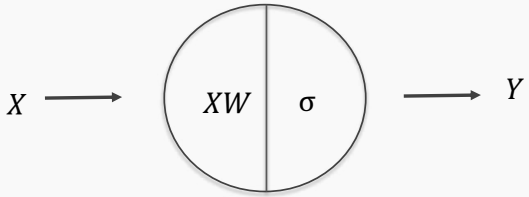
Choose W such as

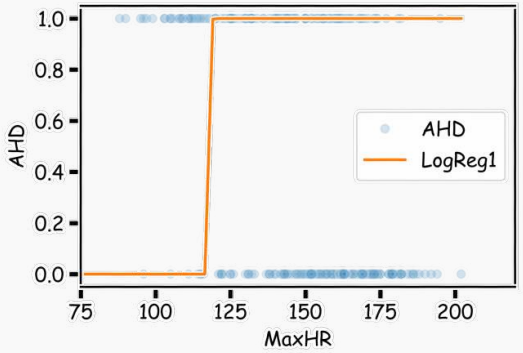


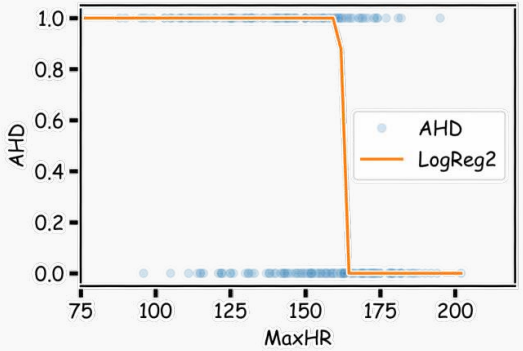
Example Using Heart Data

Two regions, two nodes

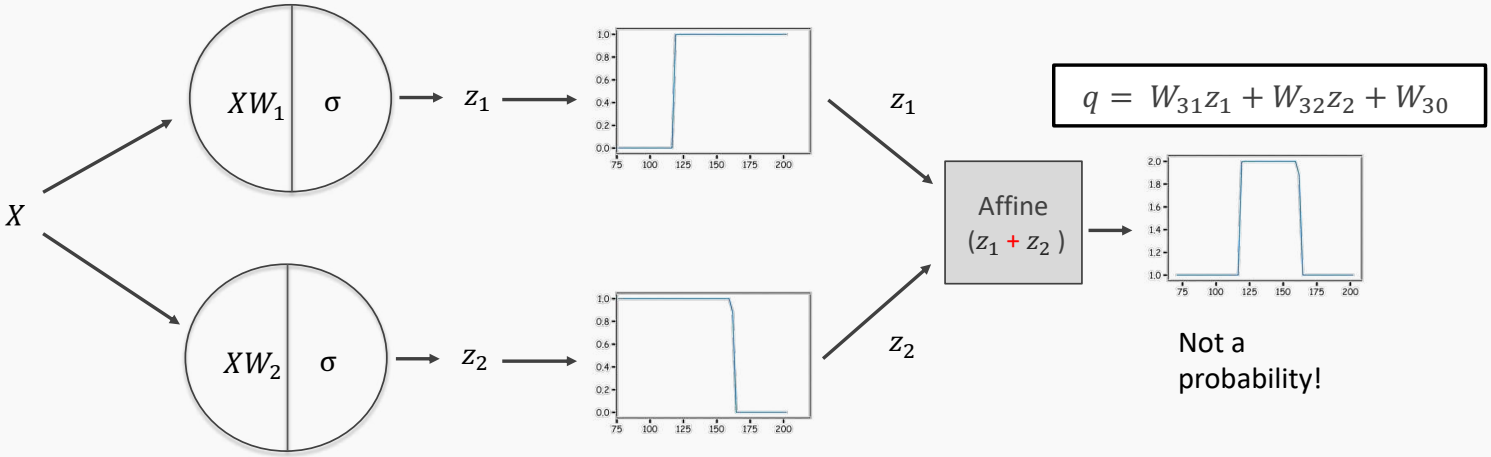




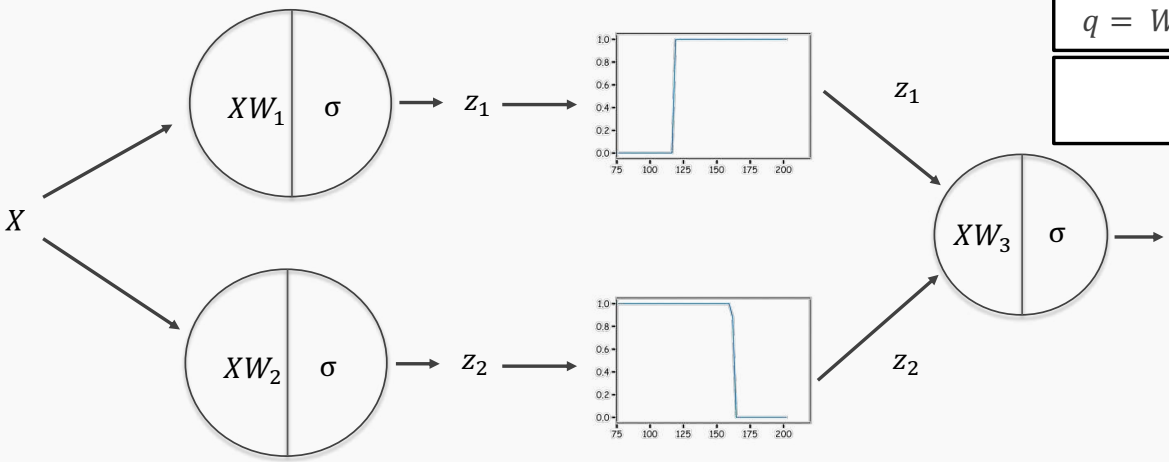




Combining Neurons



Combining Neurons …



$$q = W_{31}z_1 + W_{32}z_2 + W_{30}$$

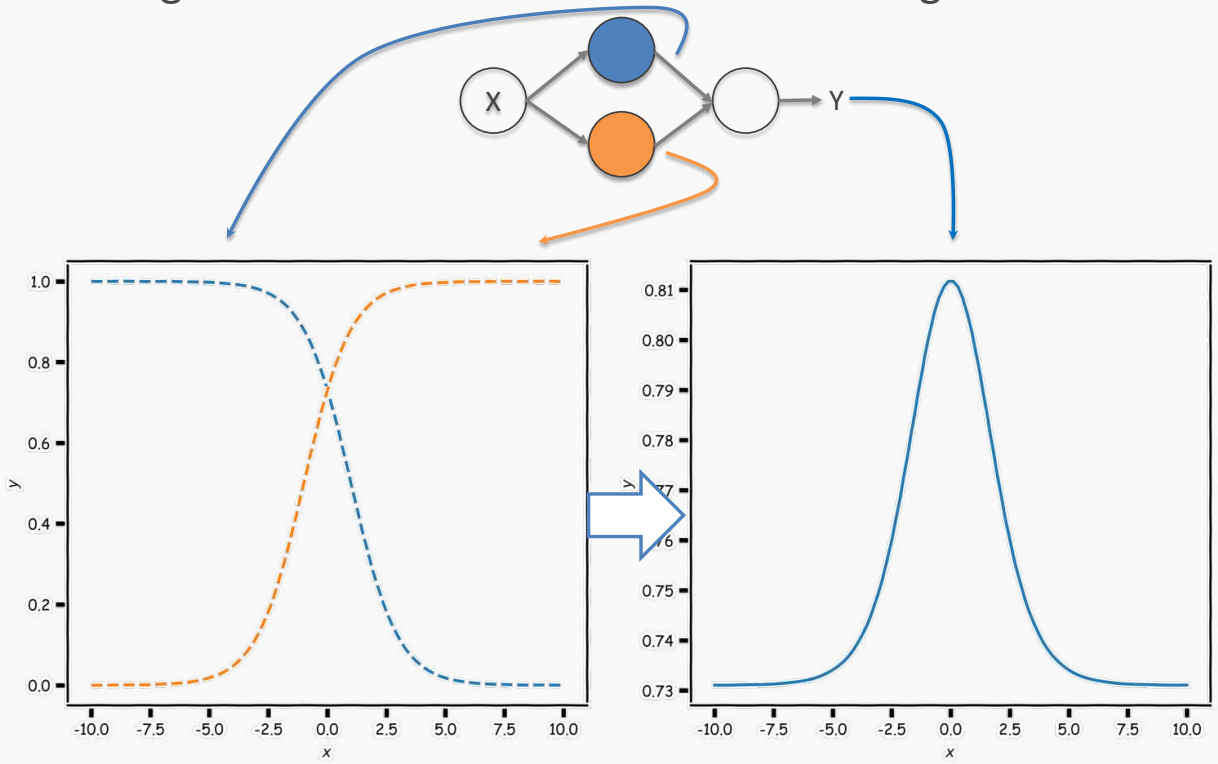
$$p = \frac{1}{1 + e^{-q}}$$

Passing through sigmoid yields probability

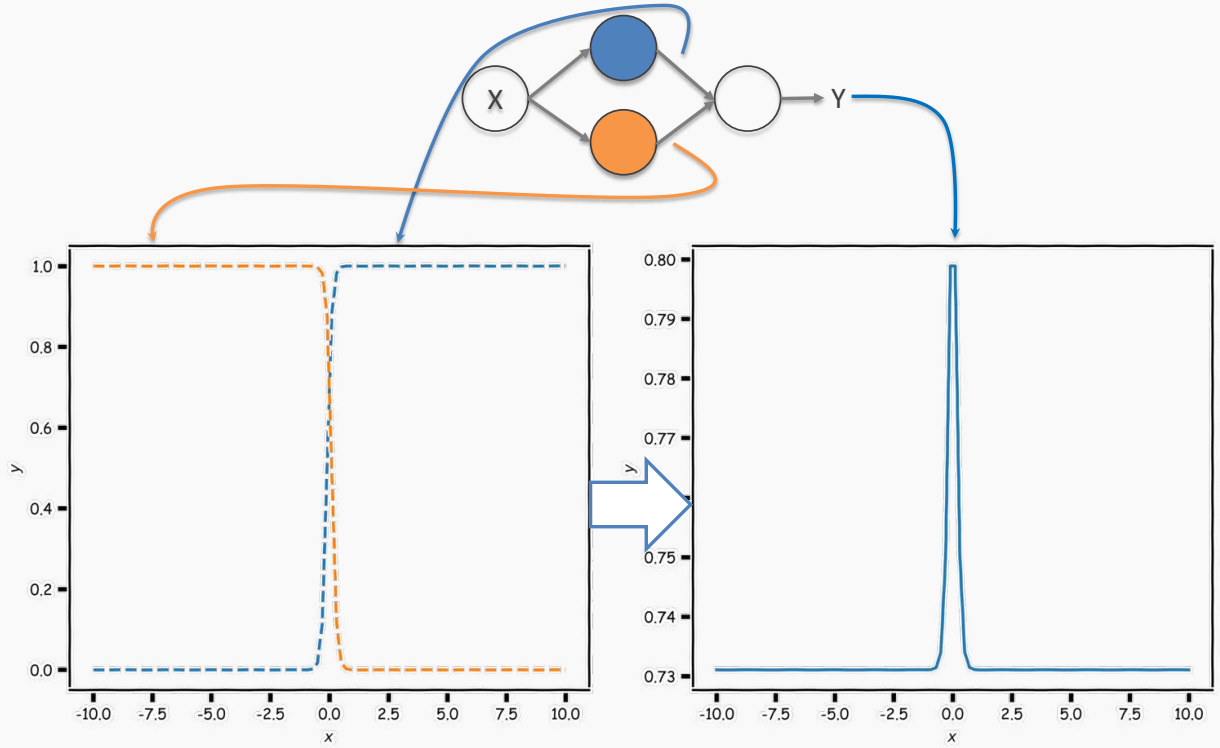
$$L = -y\ln(\mathrm{p}) - (1 - \mathrm{y})\ln(1 - \mathrm{p})$$

Need to learn W_1, W_2 and W_3

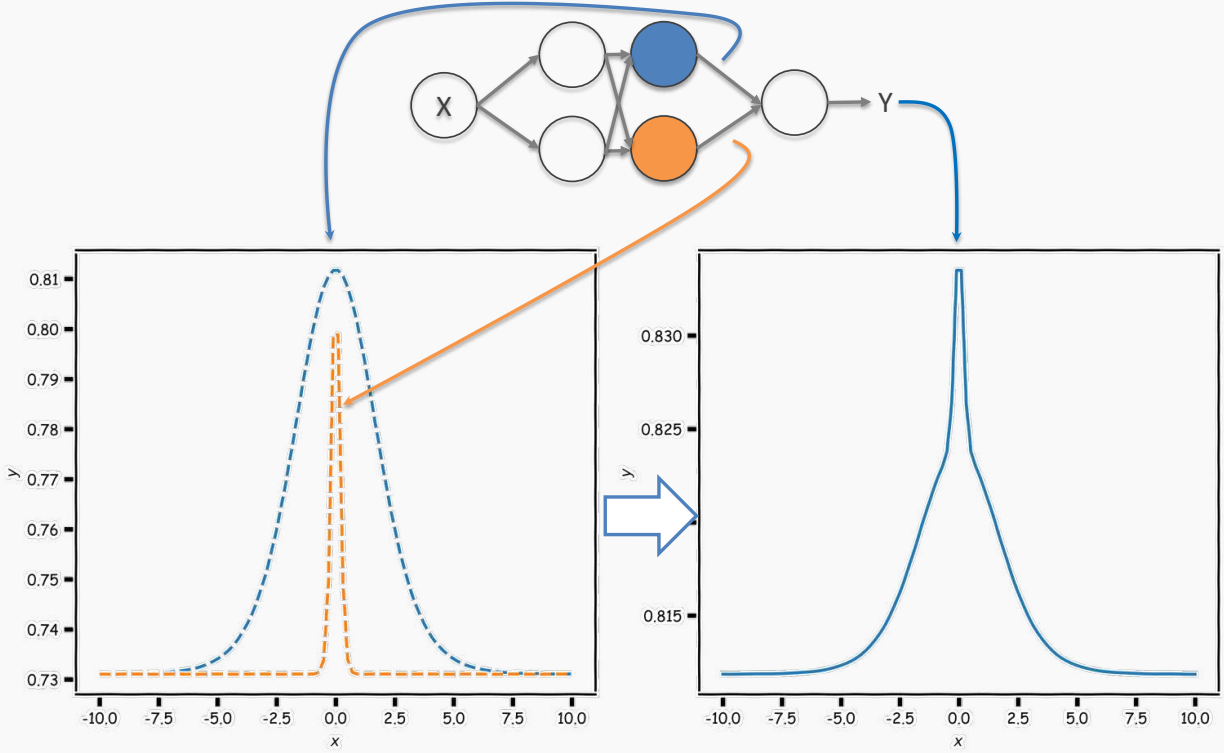
Combining neurons allows us to model interesting functions



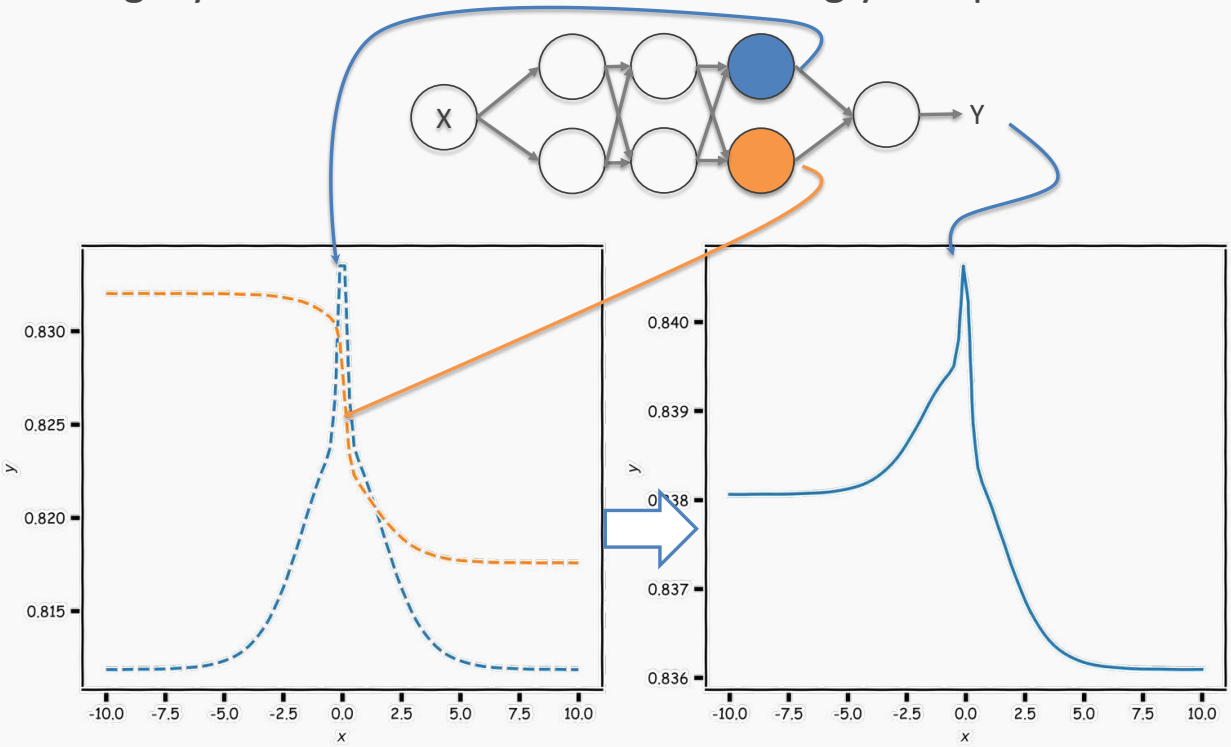
Different weights change the shape and position



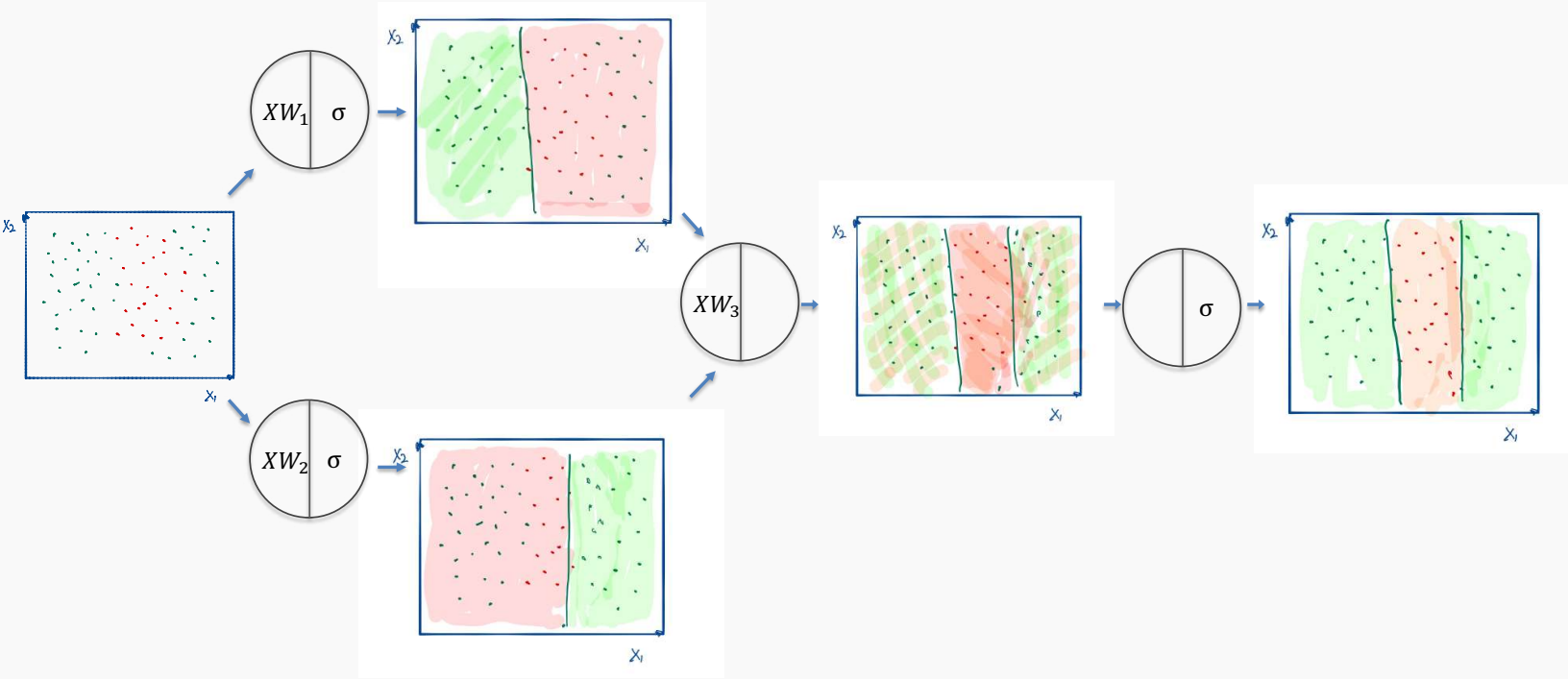
Neural networks can model *any* reasonable function



Adding layers allows us to model increasingly complex functions



For 2-D input the same idea applies.



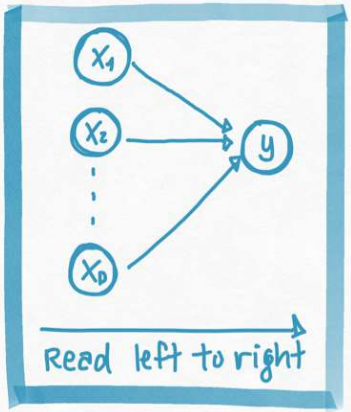
Summary

We build these complex functions by composing simple functions of the form:

$$h_w(x) = f(XW + b)$$

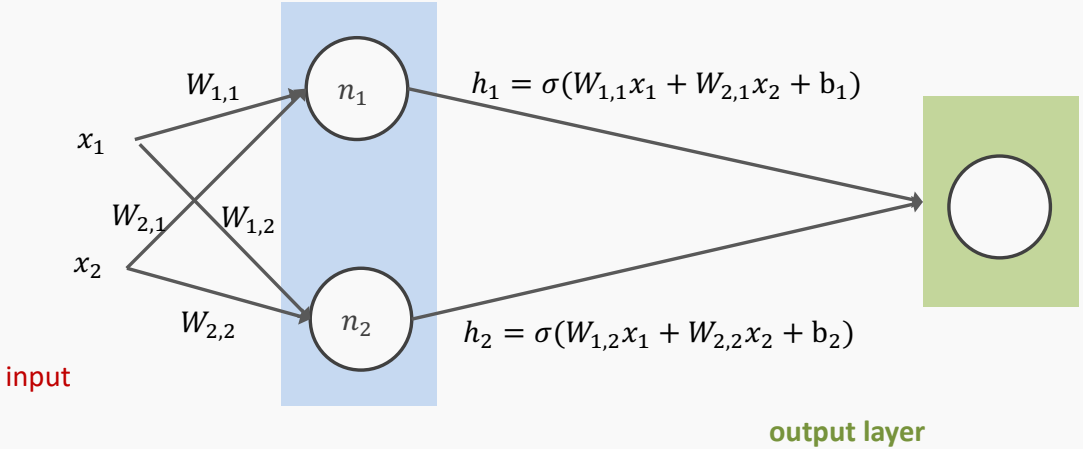
where f is the activation function.

We represent our simple function as a **graph**



Each edge in this graph represents multiplication by a different weight, w_i.

Flow in NN


$W_{1,1}$
n_1
$h_1 = \sigma(W_{1,1}x_1 + W_{2,1}x_2 + \mathrm{b}_1)$
x_1
$W_{2,1}$
$W_{1,2}$
x_2
$W_{2,2}$
n_2
$h_2 = \sigma(W_{1,2}x_1 + W_{2,2}x_2 + \mathrm{b}_2)$
input
output layer
hidden layer 1

Summary

So far:

- A single neuron can be a logistic regression or linear unit. We will soon see other choices of activation function.
- A neural network is a combination of logistic regression (or other types) units.
- A neural network can approximate non-linear functions either for regression or classification.

Next

Next:

- What kind of activations, how many neurons, how many layers, how to construct the output unit and what loss functions are appropriate?

Following lectures on NN:

- How do we estimate the weights and biases?
- How to regularize Neural Networks?

Anatomy of a NN

Outline

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture

Outline

Anatomy of a NN

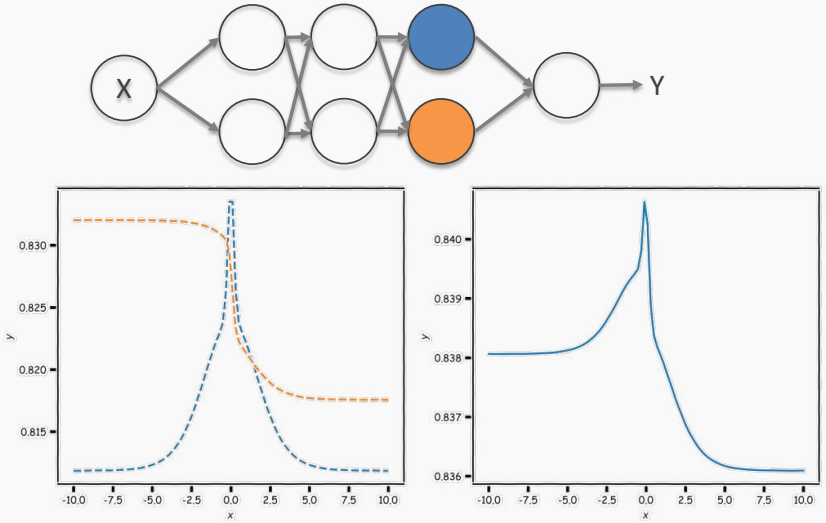
Design choices

- Activation function
- Loss function
- Output units
- Architecture

The central idea behind Neural Networks

Building a model that is both complex and simple to manipulate by **composing** multiple functions together.

Example:



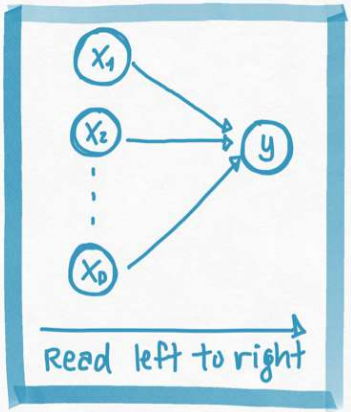
Graphical representation of simple functions

We build these complex functions by composing simple functions of the form:

$$h_w(x) = f(XW + b)$$

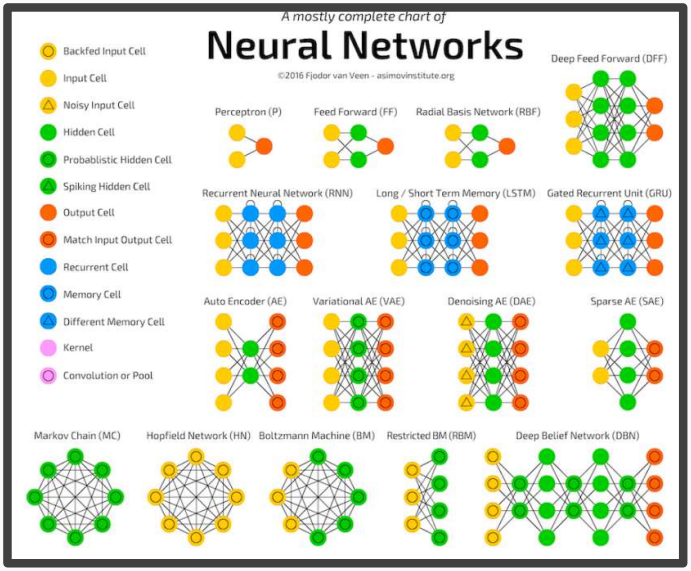
where f is the activation function.

We represent our simple function as a **graph**

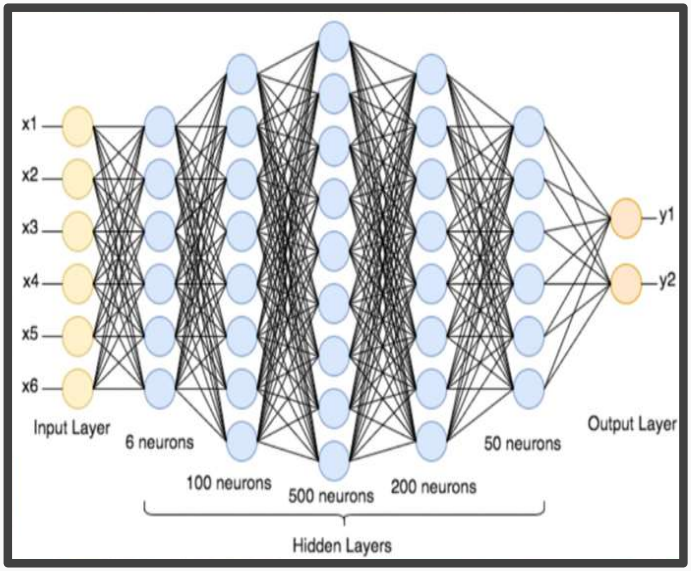


Each edge in this graph represents multiplication by a different weight, w_i.

The zoo of neural network architectures

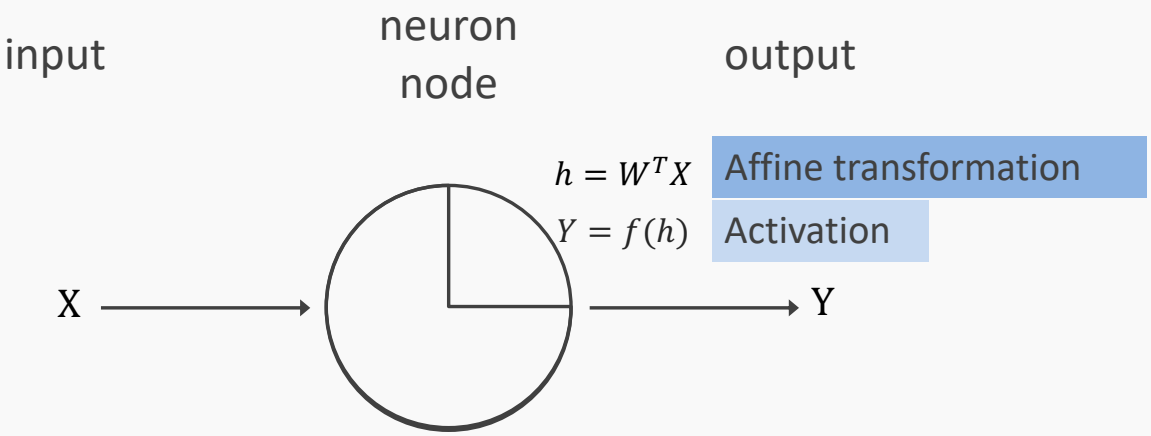


Different architectures result into functions with very different properties.



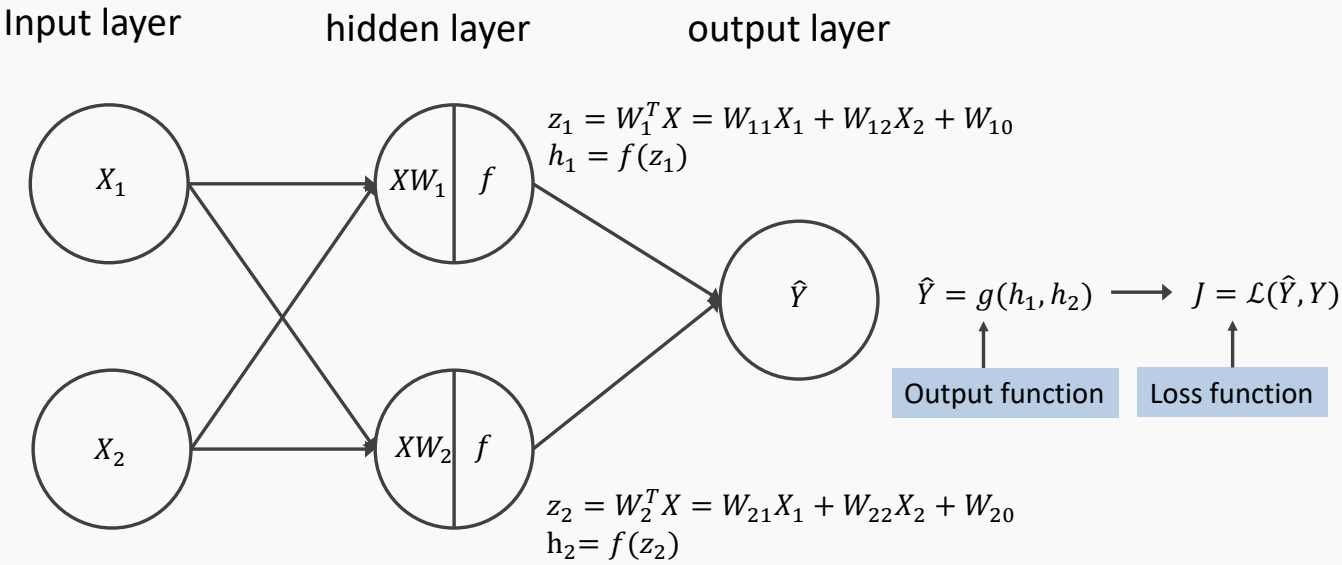
Larger networks can express more complex functions

Anatomy of artificial neural network (ANN)



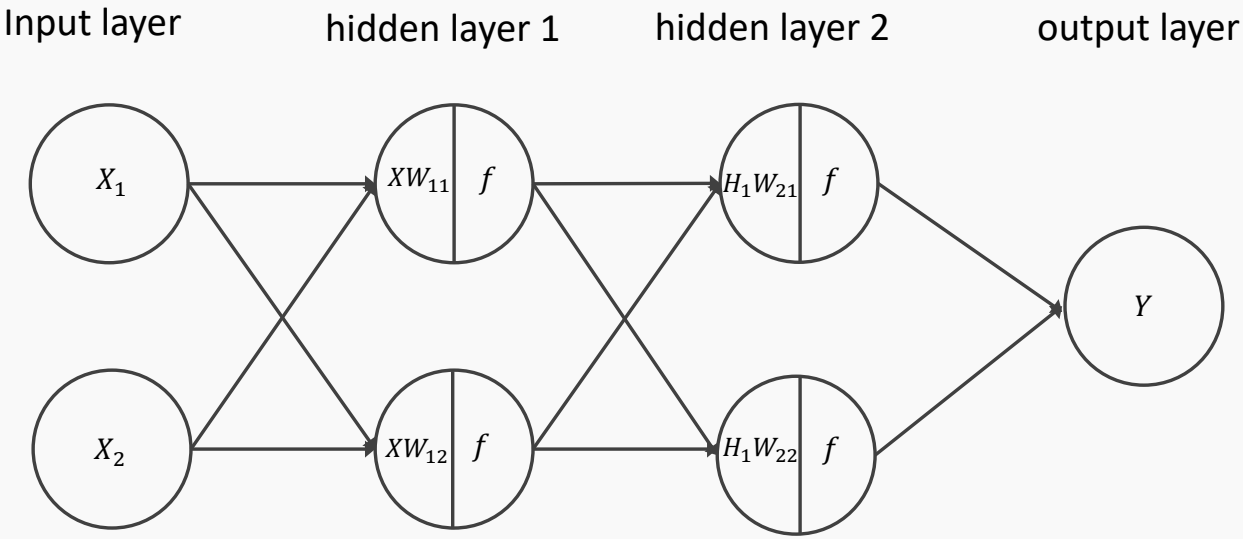
We will talk later about the choice of activation function. So far we have only talked about sigmoid as an activation function but there are other choices.

Anatomy of artificial neural network (ANN)

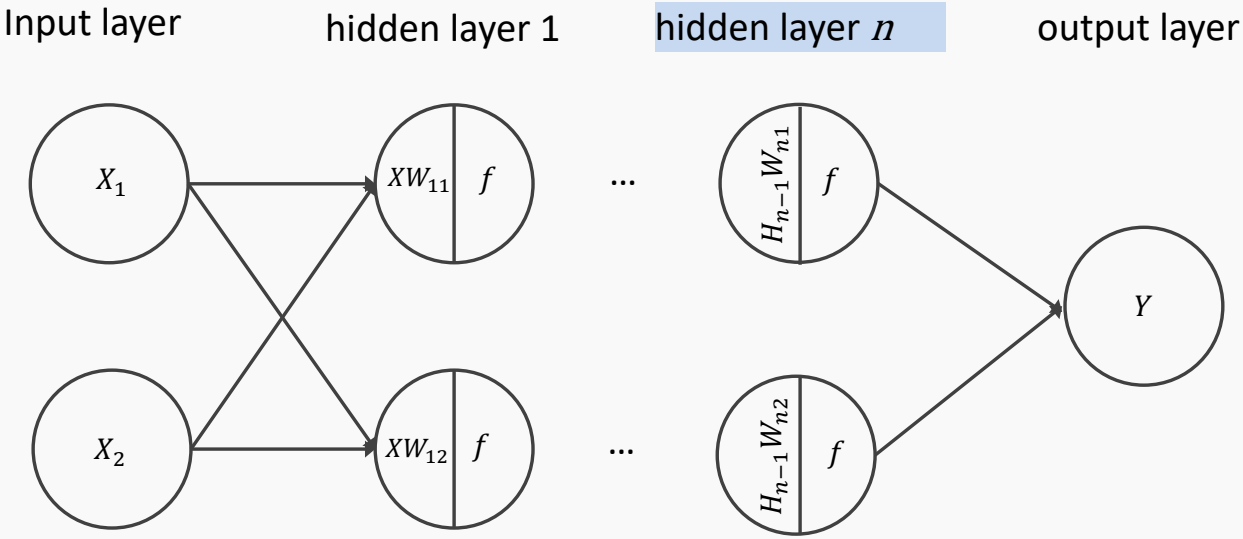


We will talk later about the choice of the output layer and the loss function. So far we consider sigmoid as the output and log-bernoulli.

Anatomy of artificial neural network (ANN)

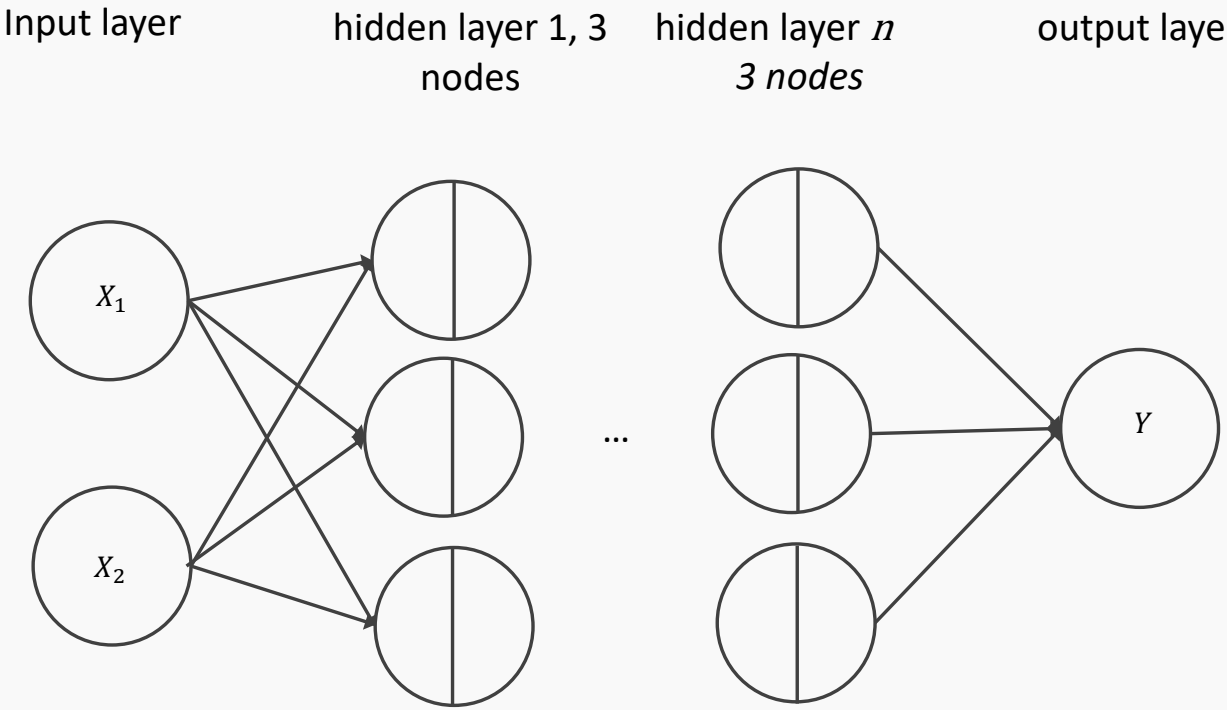


Anatomy of artificial neural network (ANN)



We will talk later about the choice of the number of layers.

Anatomy of artificial neural network (ANN)



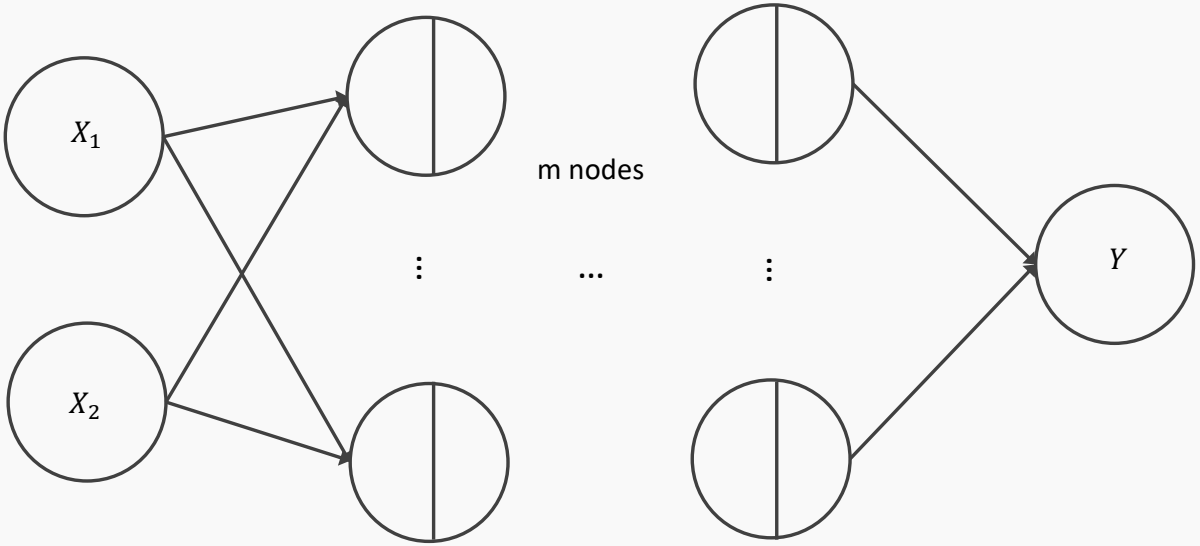
Anatomy of artificial neural network (ANN)

Input layer

hidden layer 1, m nodes

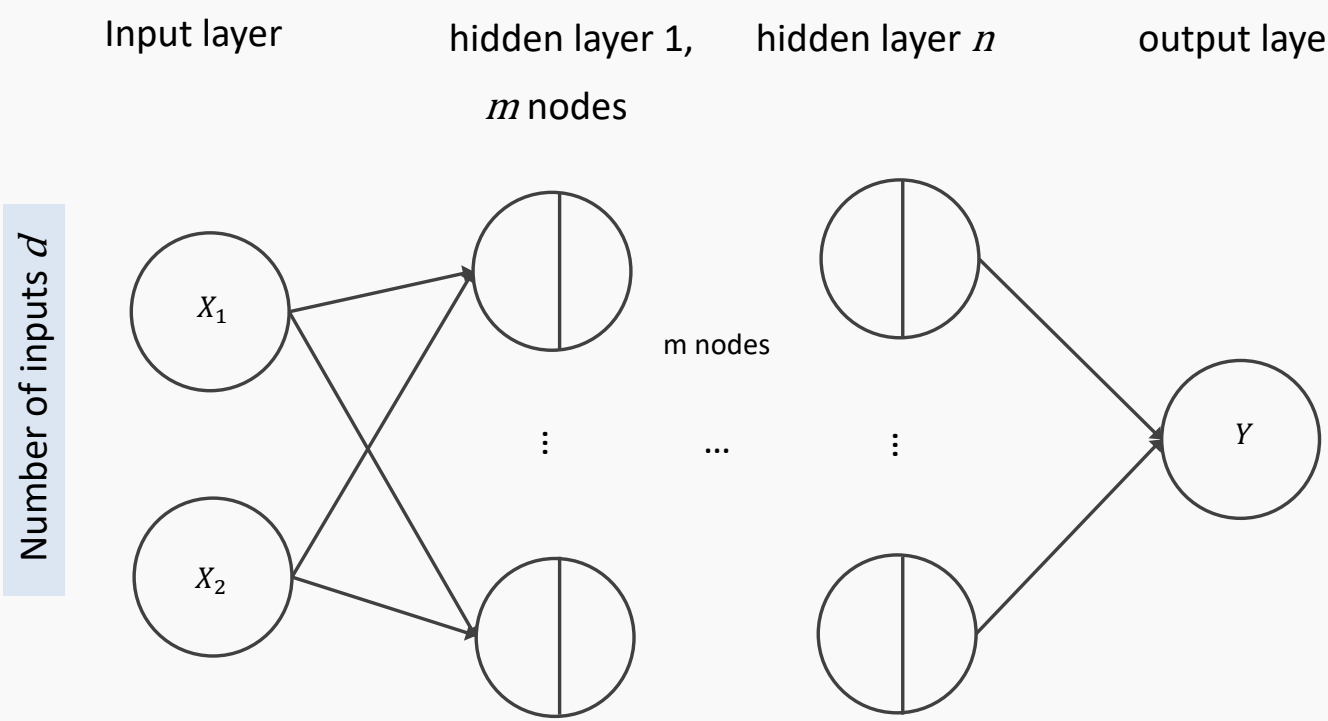
hidden layer n

output layer



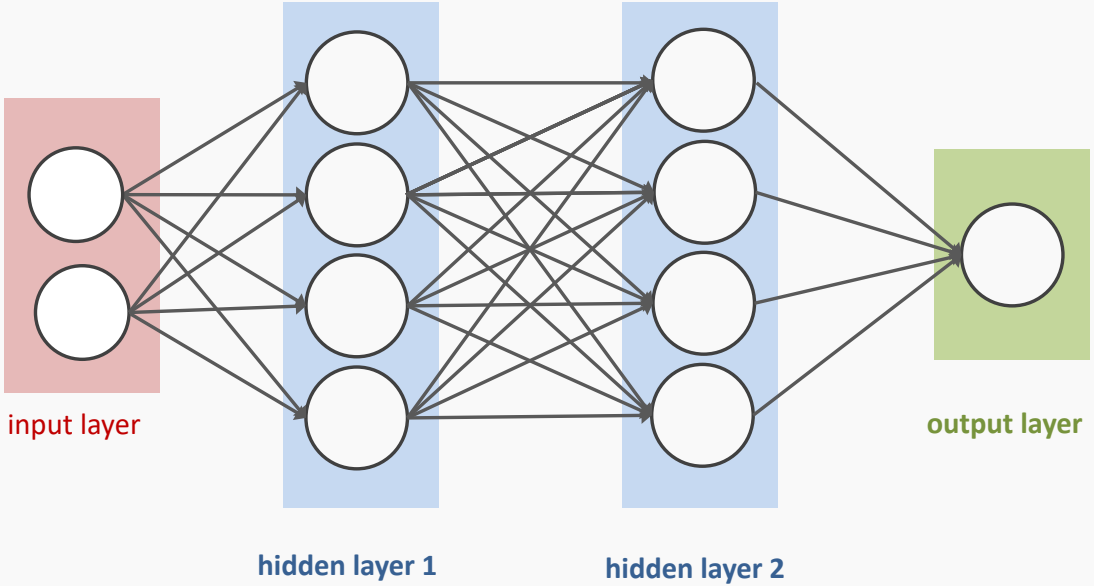
We will talk later about the choice of the number of nodes.

Anatomy of artificial neural network (ANN)

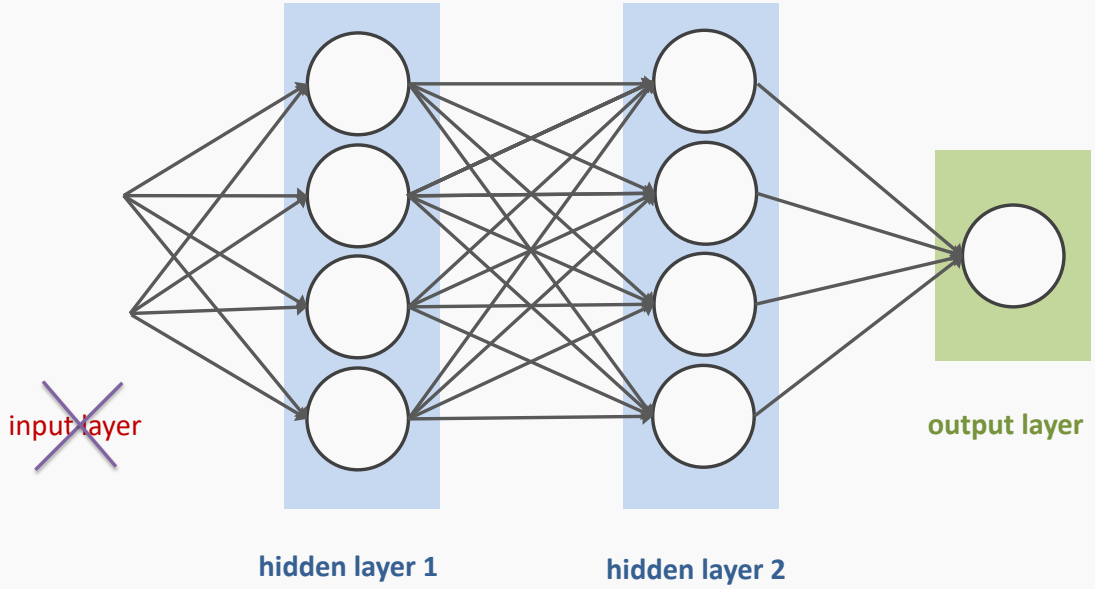


Number of inputs is specified by the data

Anatomy of artificial neural network (ANN)



Anatomy of artificial neural network (ANN)



Anatomy of a NN

Outline

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture

Outline

Anatomy of a NN

Design choices

- **Activation function**
- Loss function
- Output units
- Architecture

Activation function

$$h = f(W^T X + b)$$

The activation function should:

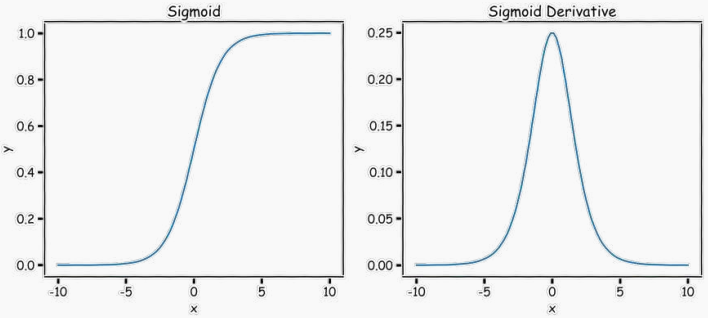
- Provide non-linearity
- Ensure gradients remain large through hidden unit

Common choices are

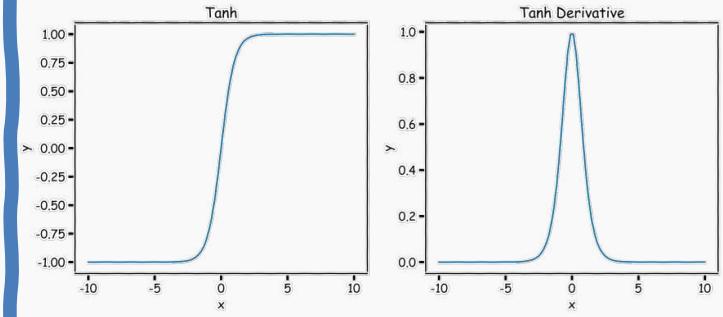
- sigmoid, tanh
- ReLU, leaky ReLU, Generalized ReLU
- softplus
- swish

Sigmoid, $\sigma()$ (aka logistic) and tanh

$$y = \frac{1}{1 + e^{-x}}$$



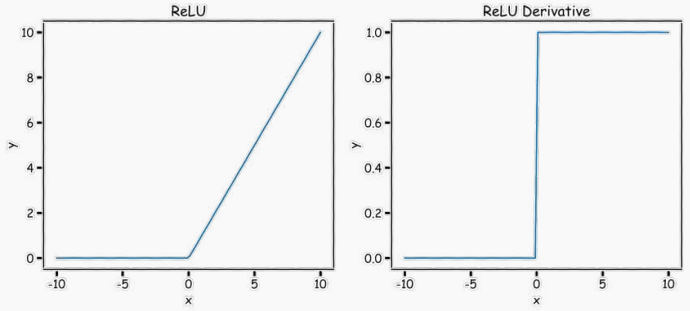
$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Derivative is **zero** for much of the domain. This leads to “vanishing gradients” in backpropagation.

Rectified Linear Unit, `ReLU()`, Exponential ReLU (ELU)

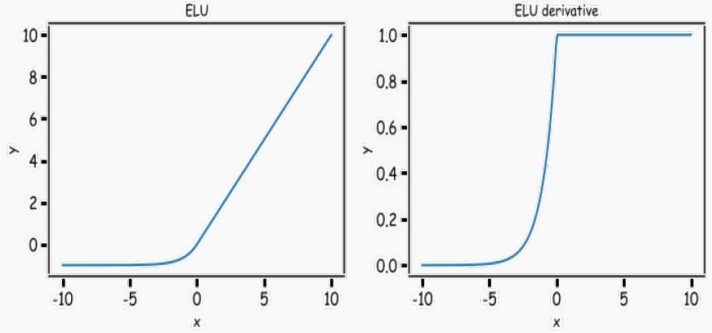
$$y = \max(0, x)$$



Two major advantages:

1. No vanishing gradient when x > 0
2. Provides sparsity (regularization) since y = 0 when x < 0

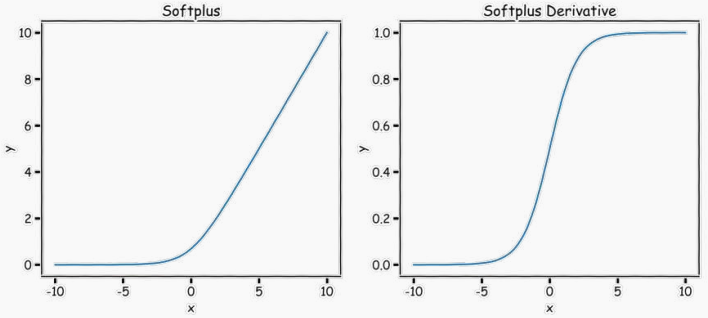
$$y = \max(0, x) + \alpha \min(0, e^x - 1)$$
where α takes a small value



No vanishing gradients and easy to calculate.

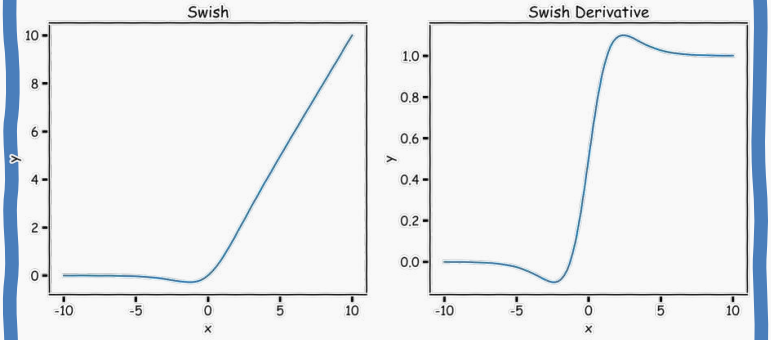
Softplus and Swish

$$y = \log(1 + e^x)$$



The derivative of the softplus is the sigmoid logistic function, which is a smooth approximation of the derivative of the rectifier. So the derivative of the softplus is continuous.

$$g(x) = x\,\sigma(x)$$



Swish tends to work better than ReLU on deeper models across a number of challenging datasets.

TL;DR (too long; didn’t read)

Use ReLU or leaky ReLU or ELU to start.

If you are concerned for a few points improvement in performance, experiment with swish and softplus and treat them as hyperparameters. In other words choose the activation that gives you the best validation loss.

Outline

Anatomy of a NN

Design choices

- Activation function
- **Loss function**
- Output units
- Architecture

Probabilistic modeling

Likelihood for a given measurement:

$$p(y_i|W;x_i)$$

Assume **independency**, likelihood for all measurements:

$$L(W;X,Y) = p(Y|W;X) = \prod_i p(y_i|W;x_i)$$

Maximize the likelihood, or equivalently minimizing the –ve log-likelihood:

$$\mathcal{L}(W;X,Y) = -\log L(W;X,Y) = \sum_i \log p(y_i|W;x_i)$$

Loss Function

Do not need to design separate loss functions if we follow the probabilistic modeling approach, i.e. minimize the –ve likelihood function.

Examples:

- Distribution is **Normal** then –ve log-likelihood is **MSE** :

$$p(y_i|W;x_i) = \frac{1}{\sqrt{\{2\pi^2\sigma\}}} e^{-\frac{(y_i-\hat{y}_i)^2}{2\sigma^2}}$$

$$\mathcal{L}(W;X,Y) = \sum_i (y_i - \hat{y}_i)^2$$

- Distribution is **Bernoulli** then –ve log-likelihood is **Binary Cross-Entropy**:

$$p(y_i|W;x_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

$$\mathcal{L}(W;X,Y) = -\sum_i [y_i \log p_i + (1-y_i)\log(1-p_i)]$$

Loss Function

- For y that follow **Multinoulli,** then likelihood is:

$$p(y_i|W;x_i) = \prod_k p\,(y_i = k)^{\mathbb{I}(y_i=k)}$$

Cross-Entropy

$$\mathcal{L}(W;X,Y) = -\sum_i \sum_k \mathbb{I}(y_i = k)\log p(y_i = k)$$

where k is the class and $\mathbb{I}(y_i = k)$ is the indicator function.

Questions:

- How do we know what distribution to use?
- Why not MSE for classification?

Design Choices

Activation function

Loss function

Output units

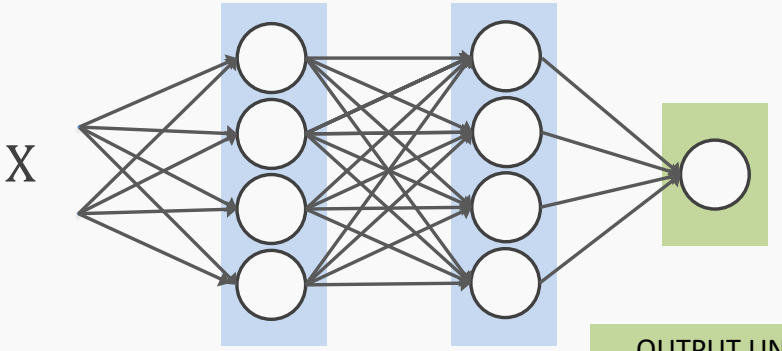
Architecture

Optimizer

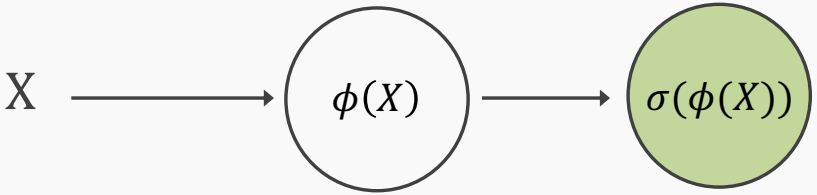
Output Units

Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli	?	Binary Cross Entropy

Output unit for binary classification



$\hat{Y} = \mathrm{P}(\mathrm{y} = 1)$



$\hat{Y} = \mathrm{P}(\mathrm{y} = 1)$

$$X \Longrightarrow \phi(X) \Longrightarrow P(y = 1) = \frac{1}{1 + e^{-\phi(X)}}$$

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete			

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli		

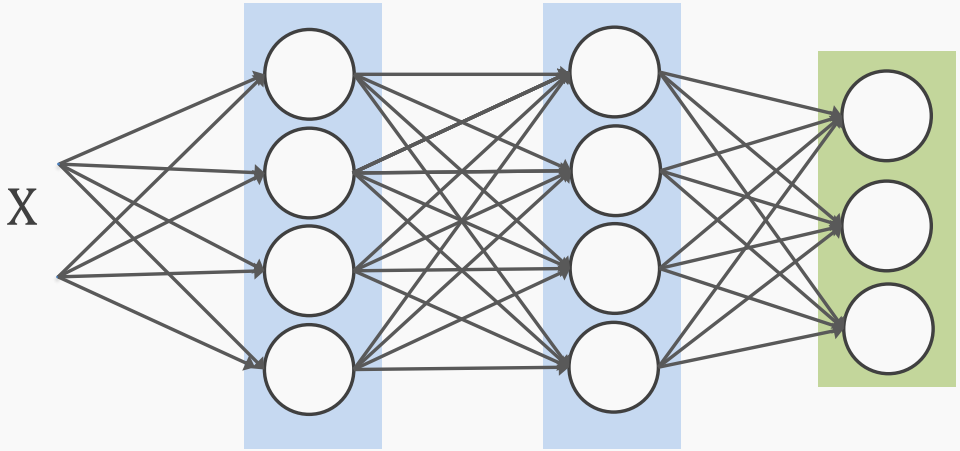
Output Units

Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinouli		Cross Entropy

Output Units

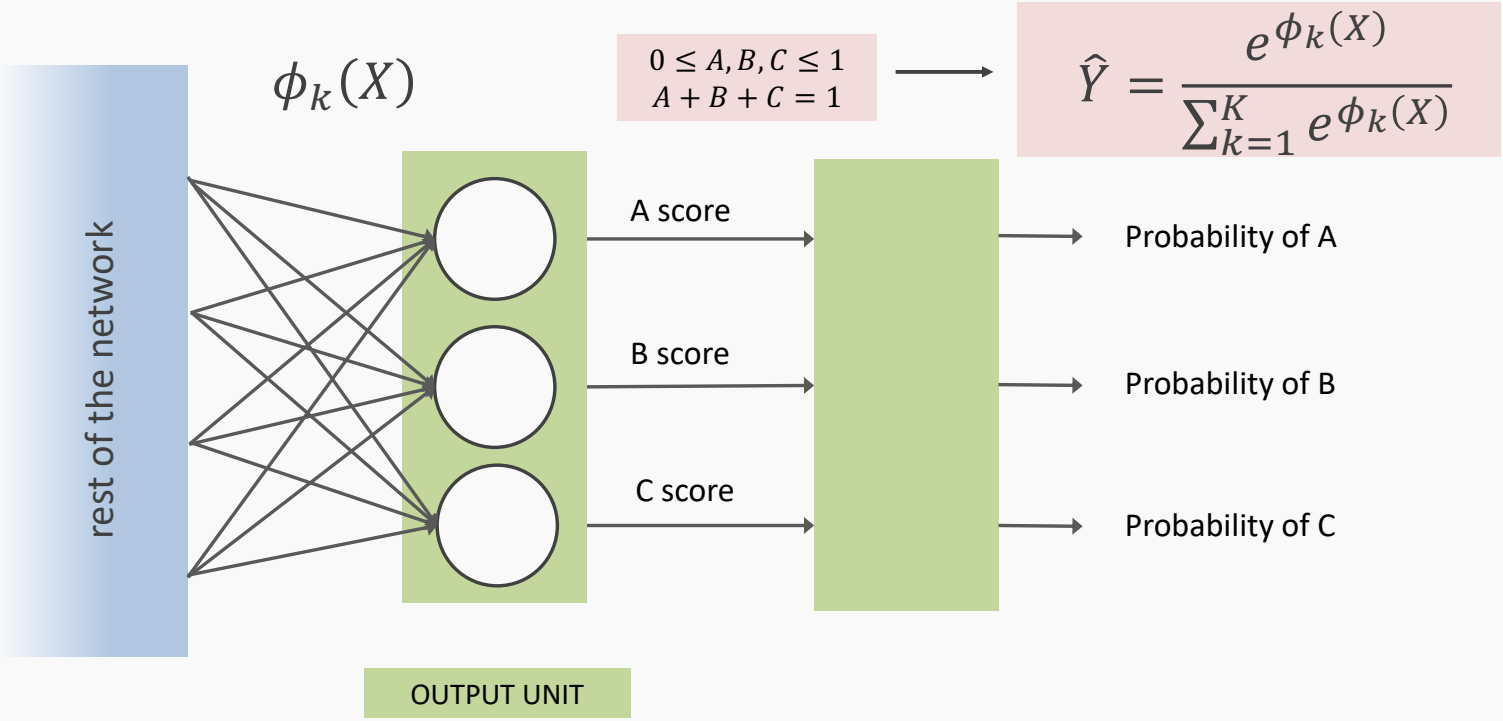
Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinouli	?	Cross Entropy

Output unit for multi-class classification



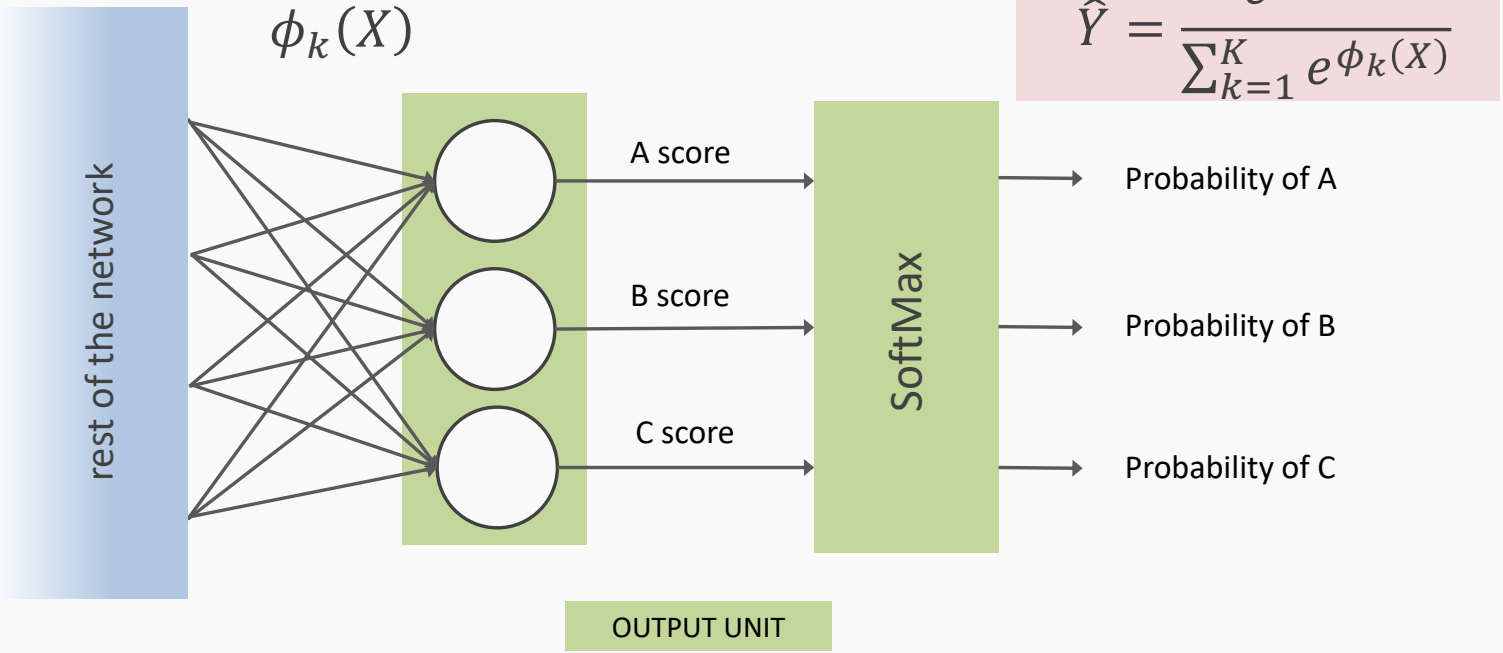
$$\hat{Y} = [P_1, P_2, P_3]$$

SoftMax


$\phi_k(X)$
$0 \le A, B, C \le 1$
$A + B + C = 1$
$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$
rest of the network
A score
B score
C score
Probability of A
Probability of B
Probability of C
OUTPUT UNIT

SoftMax

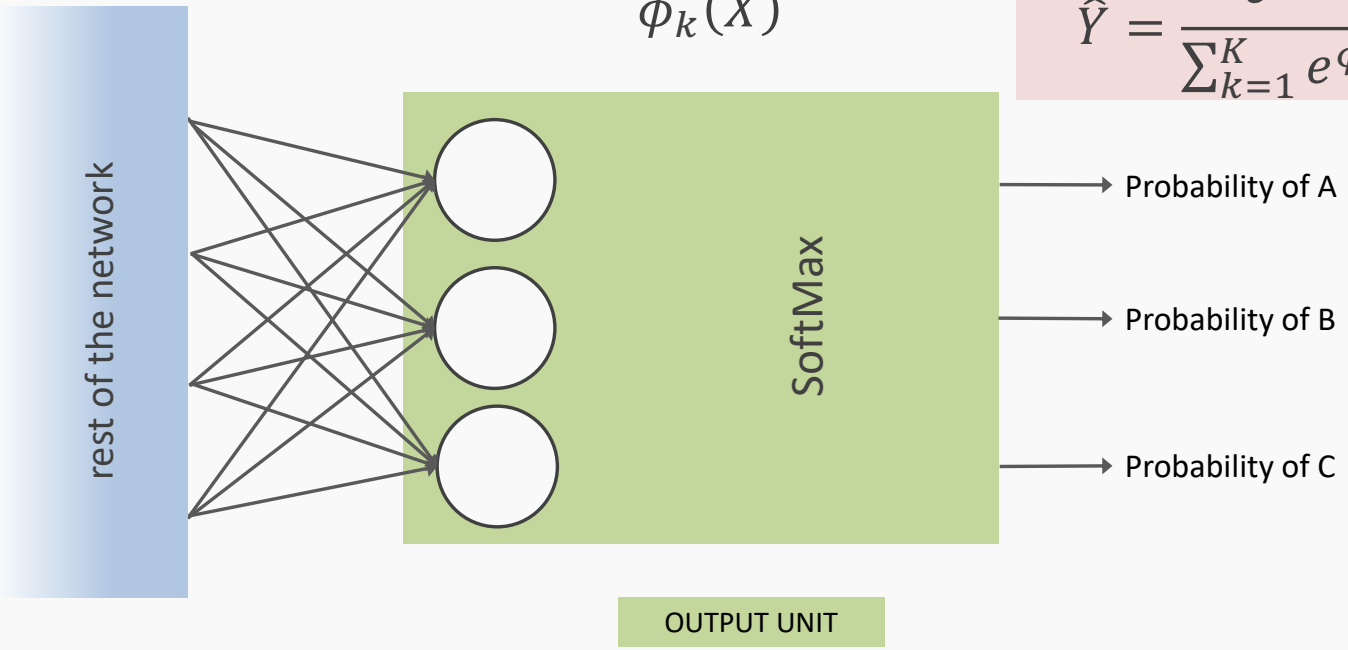
$$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$$



SoftMax

$$\phi_k(X)$$

$$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$$



Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous			

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian		

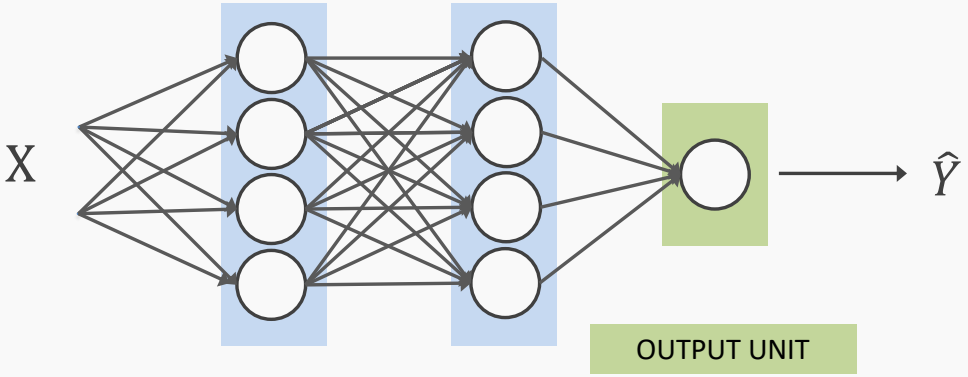
Output Units

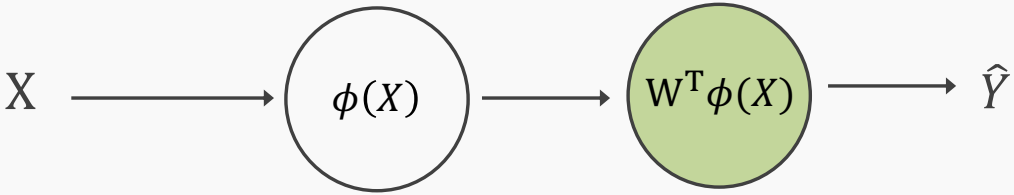
Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian		MSE

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	?	MSE

Output unit for regression





$$X \Longrightarrow \phi(X) \Longrightarrow \hat{Y} = W^T\phi(X)$$

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE
Continuous	Arbitrary	-	

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE
Continuous	Arbitrary	-	GANS

Design Choices

Activation function

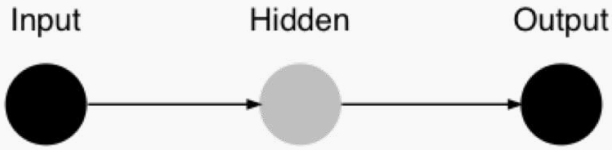
Loss function

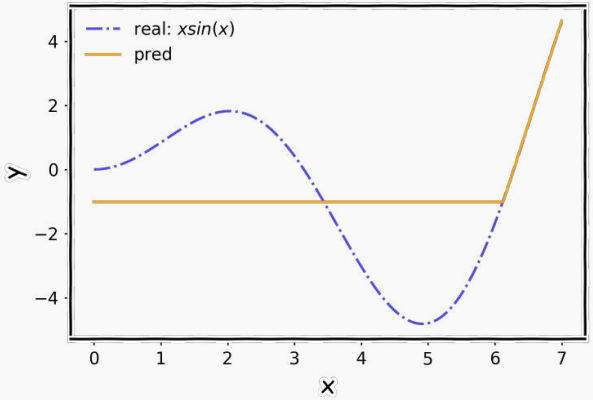
Output units

Architecture

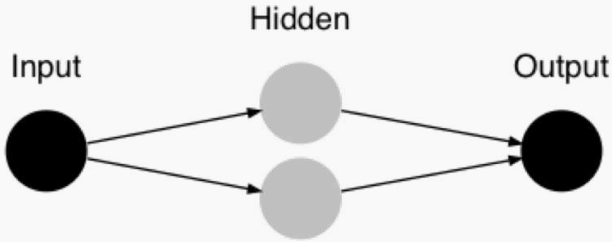
Optimizer

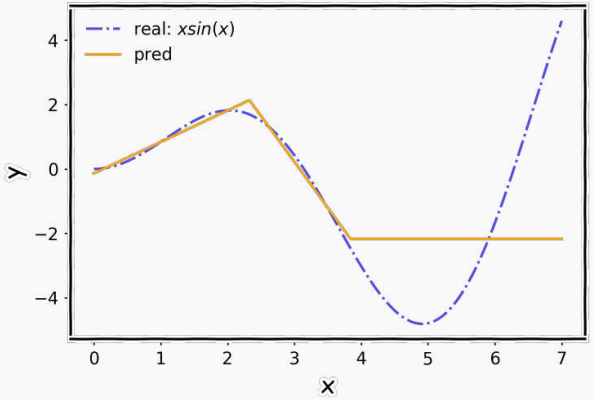
Number of nodes



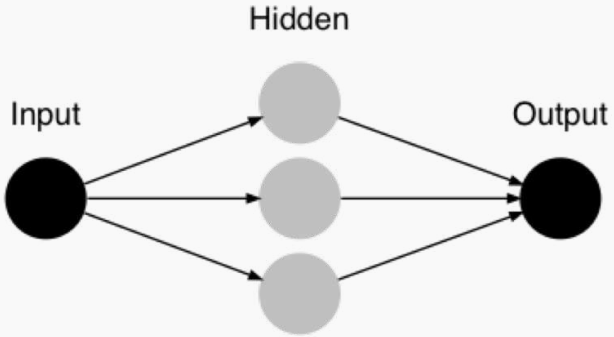


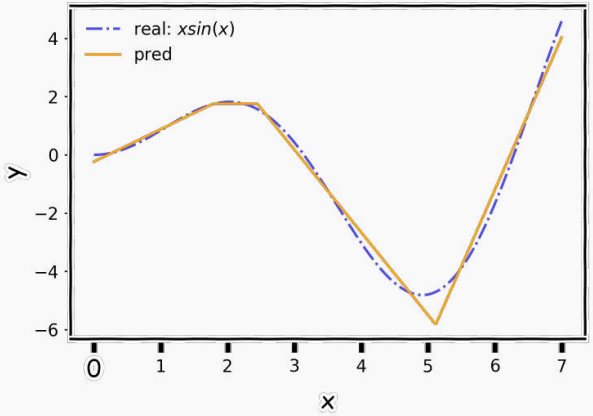
Number of nodes



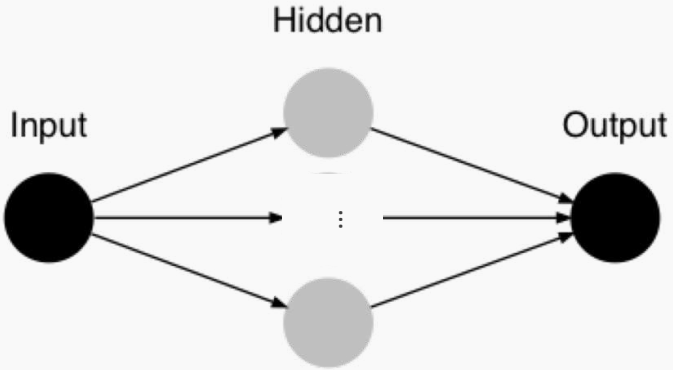


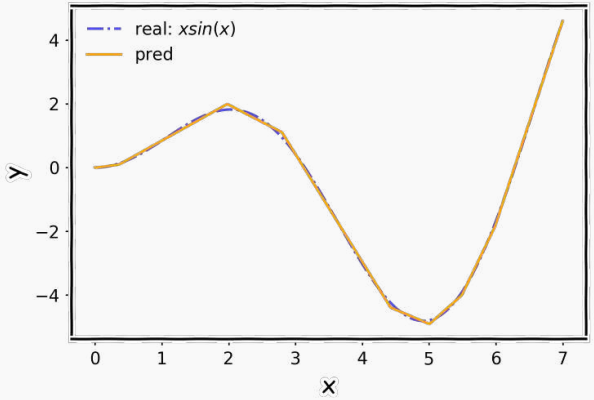
Number of nodes



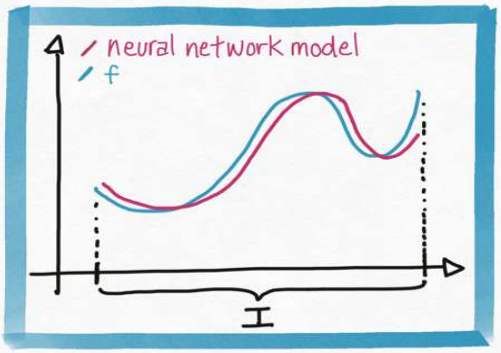


Number of nodes





Neural Networks as Universal Approximators



We have seen that neural networks can represent complex functions, but are there limitations on what a neural network can express?

Theorem:

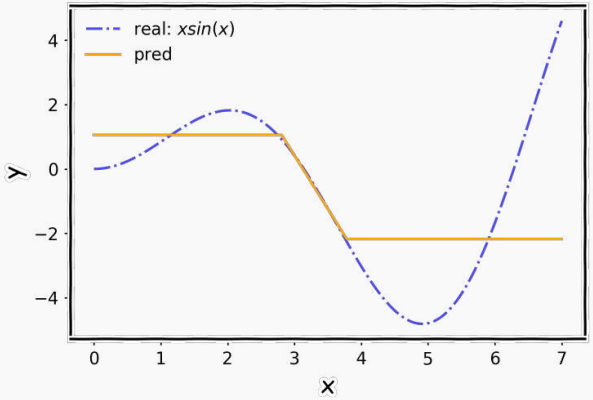
For any continuous function f defined on a bounded domain, we can find a neural network that approximates f with an arbitrary degree of accuracy.

One hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy.

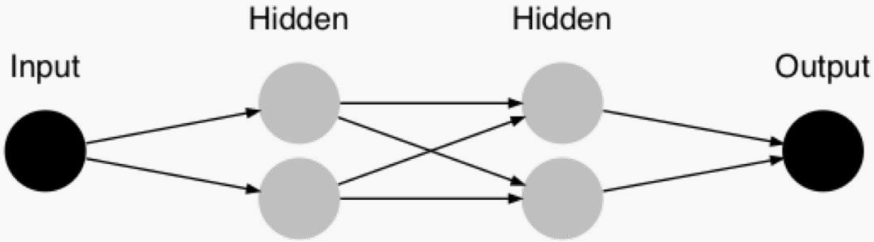
So why deeper?

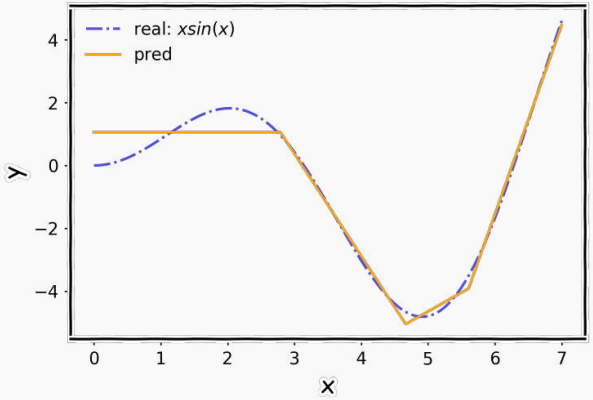
Layers


Input
Hidden
Hidden
Output

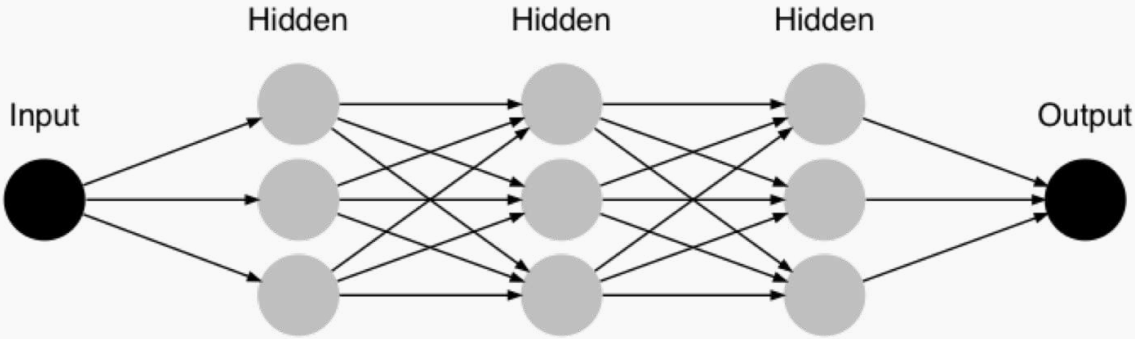

real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

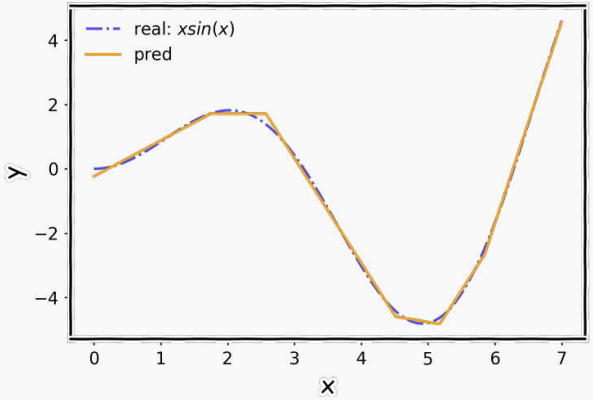
Layers


Input
Hidden
Hidden
Output


real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

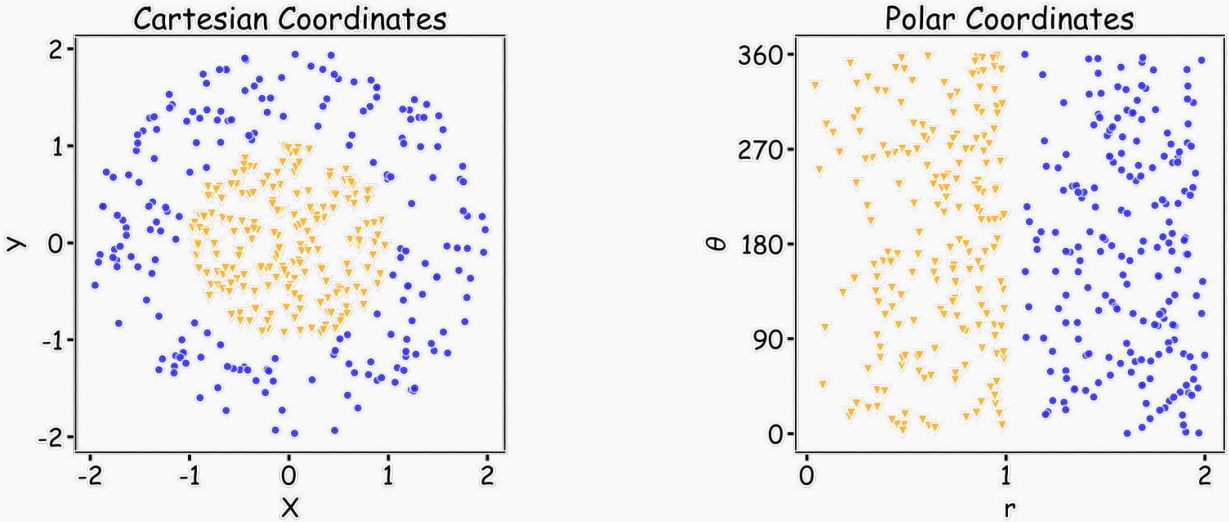
Layers


Hidden
Hidden
Hidden
Input
Output

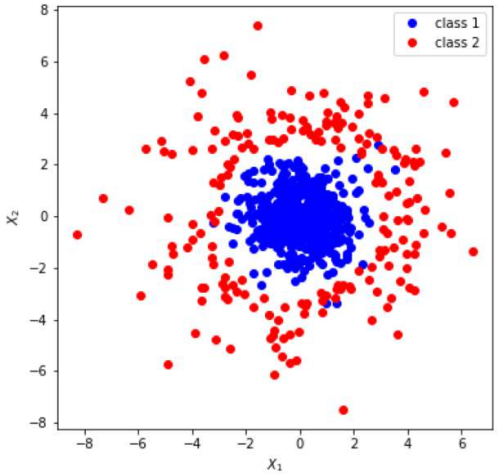

real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

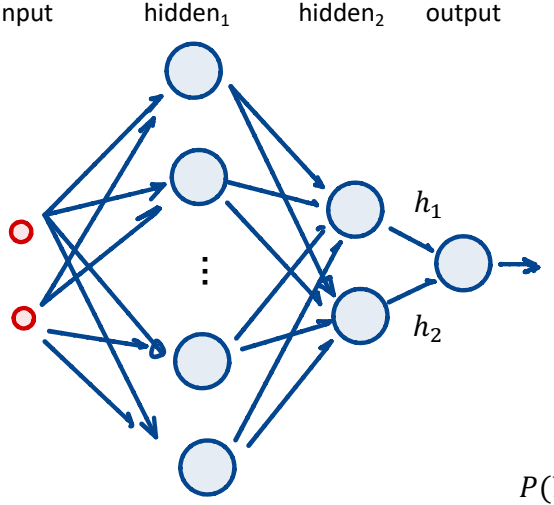
Why layers?

Representation matters!

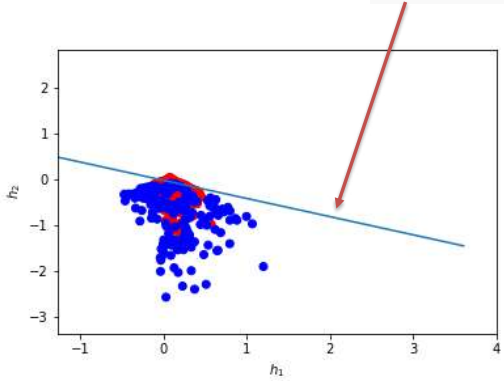


Neural networks can **learn useful representations** for the problem. This is another reason why they can be so powerful!

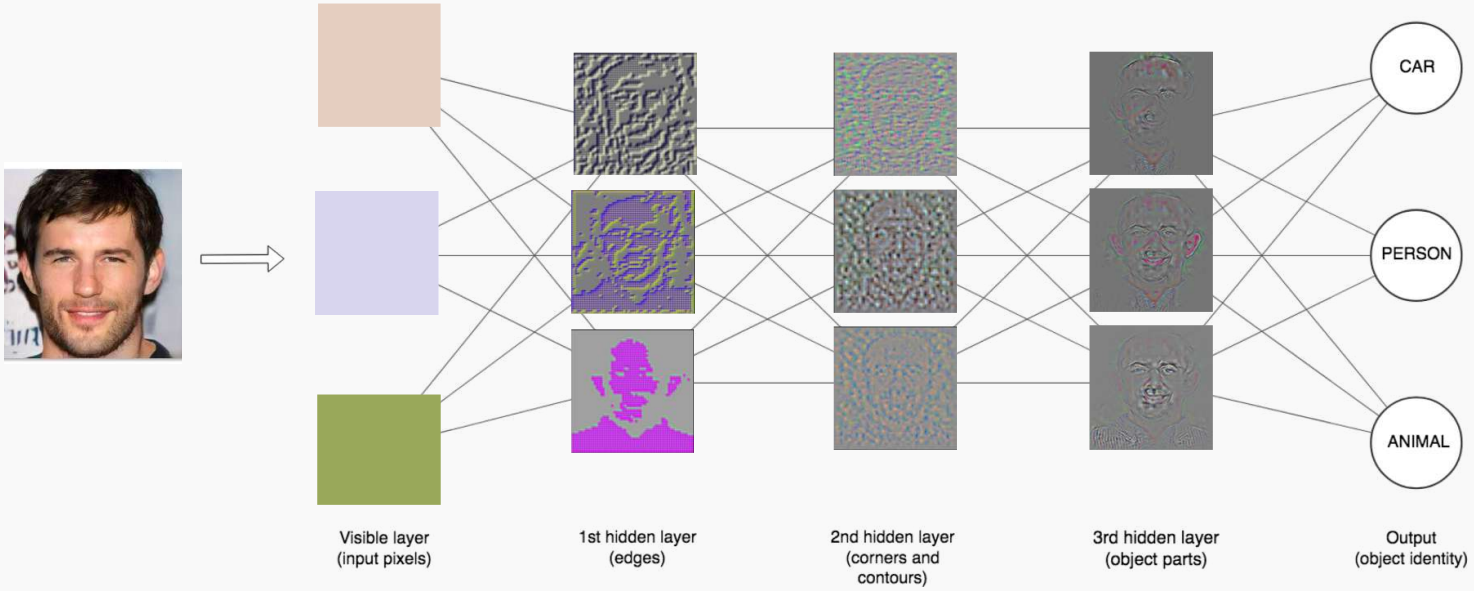

class 1
class 2
X_1
X_2


input
hidden$_1$
hidden$_2$
output
h_1
h_2
$P(Y = 1)$

$P(Y = 0.5)$


h_1
h_2

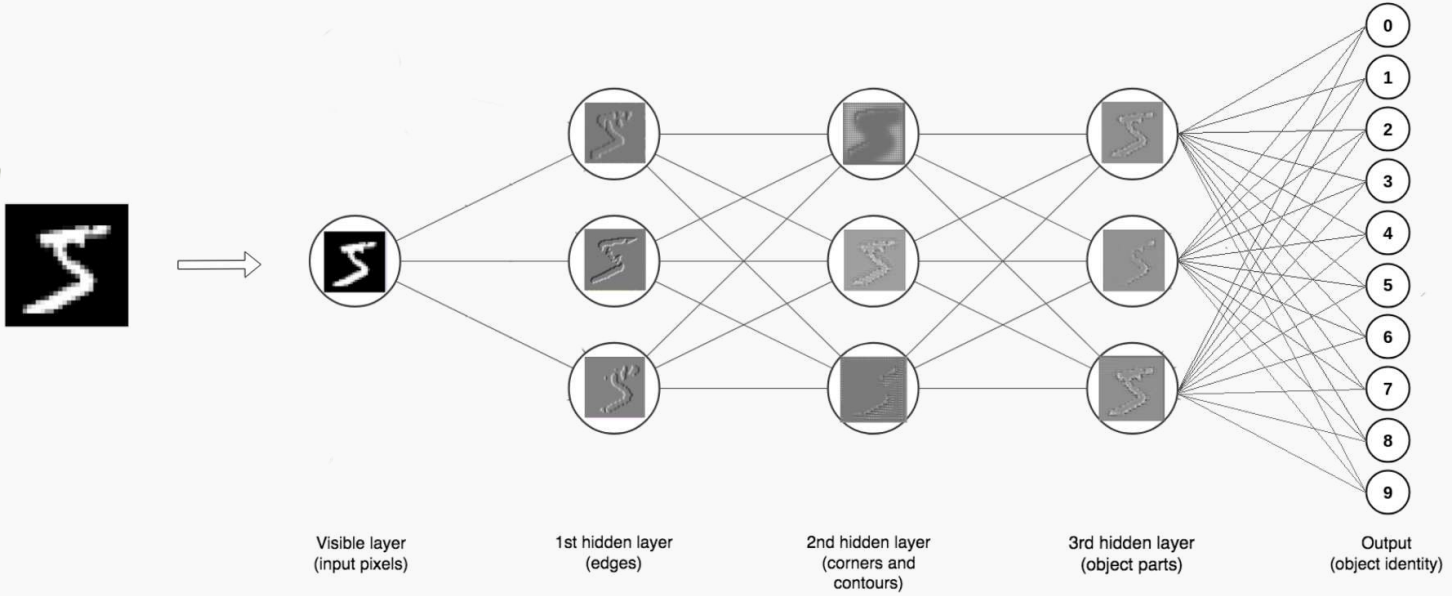
Depth = Repeated Compositions



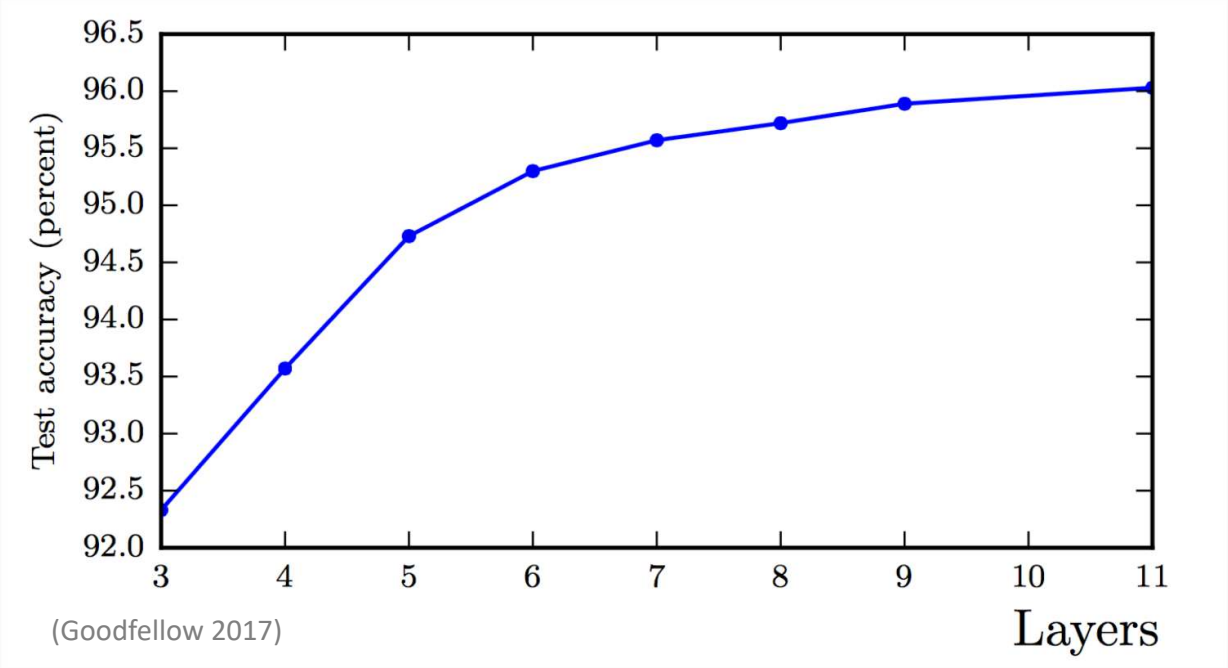
Neural Networks

Hand-written digit recognition: MNIST data

Depth = Repeated Compositions



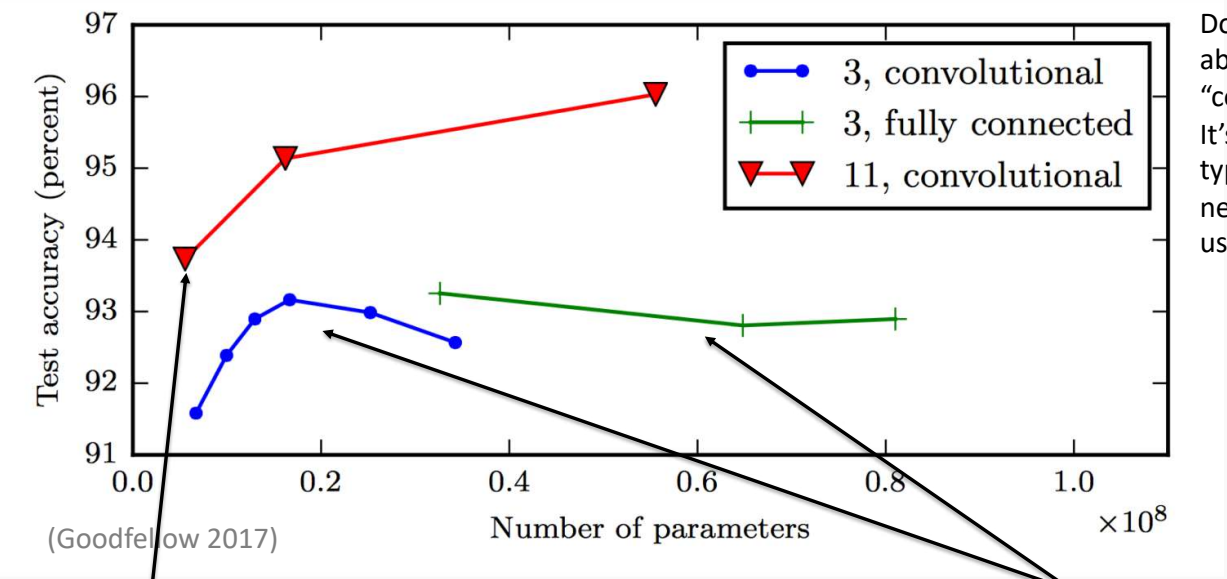
Better Generalization with Depth



(Goodfellow 2017)

Shallow Nets Overfit More

Depth helps, and it's not just because of more parameters



Don't worry about this word "convolutional". It's just a special type of neural network, often used for images.

The **11-layer net** generalizes better on the test set when controlling for number of parameters.

The 3-layer nets perform worse on the test set, even with similar number of total parameters.

Design Choices

Activation function

Loss function

Output units

Architecture

Optimizer

Learning the coefficients

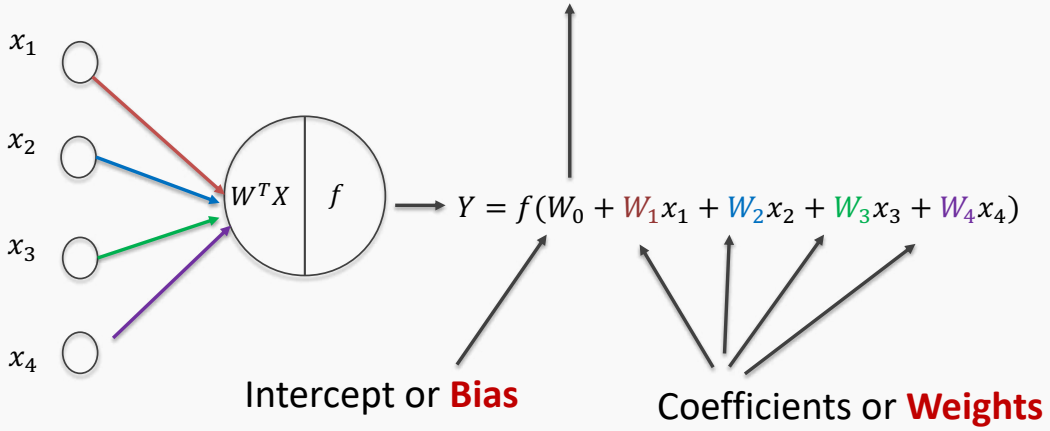
Start with single neuron

Classification

$f(X) = \frac{1}{1+e^{-W^T X}}$

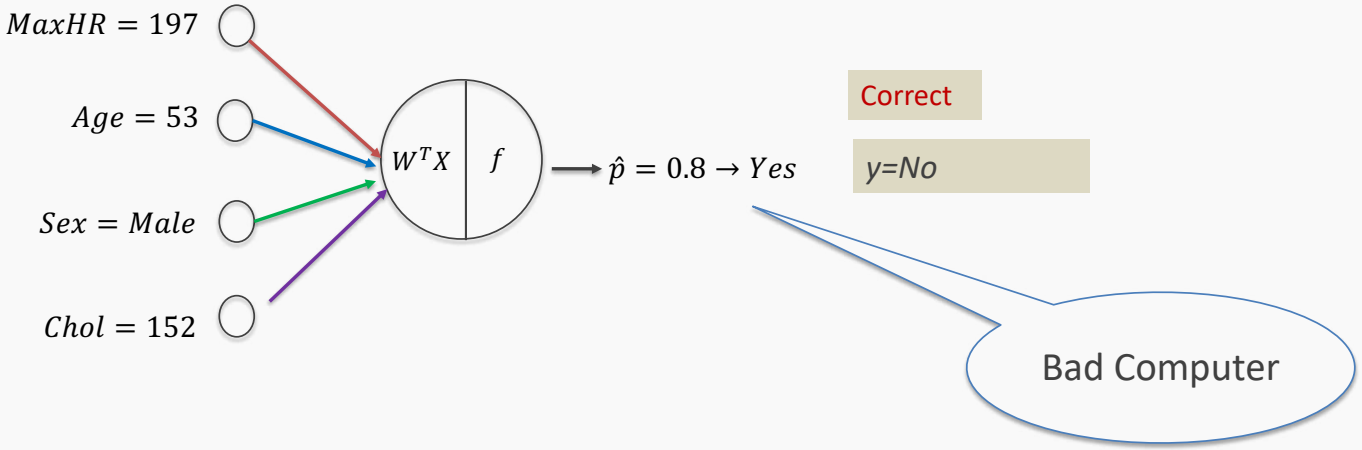
Regression

$f(X) = W^T X$



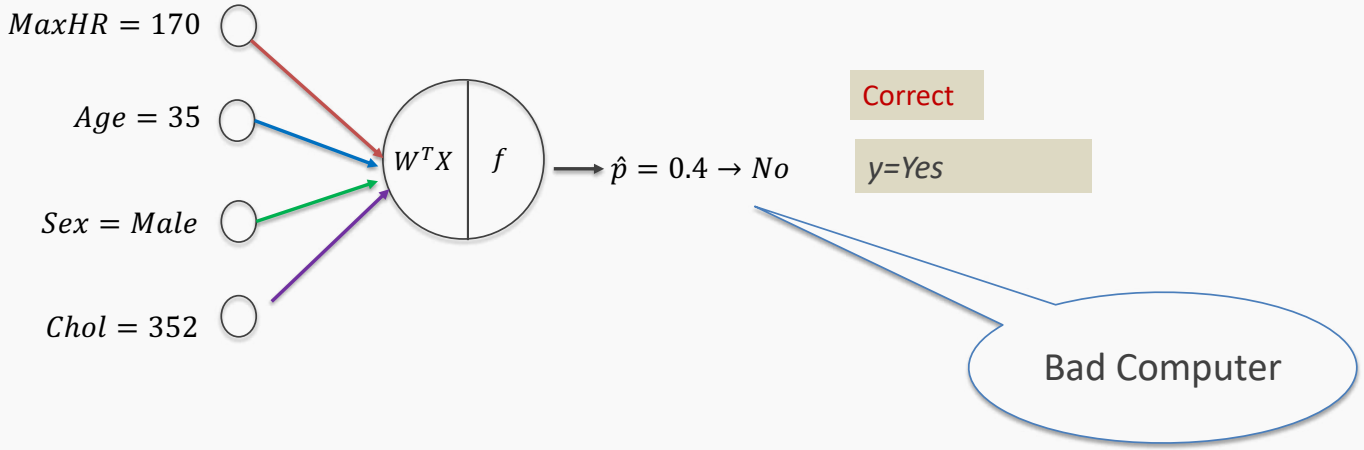
But what is the idea?

Start with all randomly selected weights. Most likely it will perform horribly.
For example, in our heart data, the model will be giving us the wrong answer.



But what is the idea?

Start with all randomly selected weights. Most likely it will perform horribly.
For example, in our heart data, the model will be giving us the wrong answer.



But what is the idea?

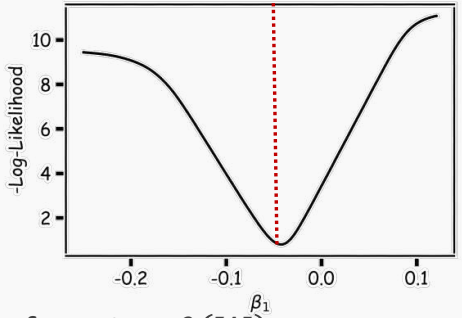
- **Loss Function:** Takes all of these results and averages them and tells us how bad or good the computer or those weights are.
- Telling the computer how **bad** or **good** is, does not help.
- You want to tell it how to change those weights so it gets better.

Loss function: $\mathcal{L}(w_0, w_1, w_2, w_3, w_4)$

For now let's only consider a single weight, $\mathcal{L}(w_1)$

Minimizing the Loss function

Ideally we want to know the value of w_1 that gives the minimal $\mathcal{L}(W)$

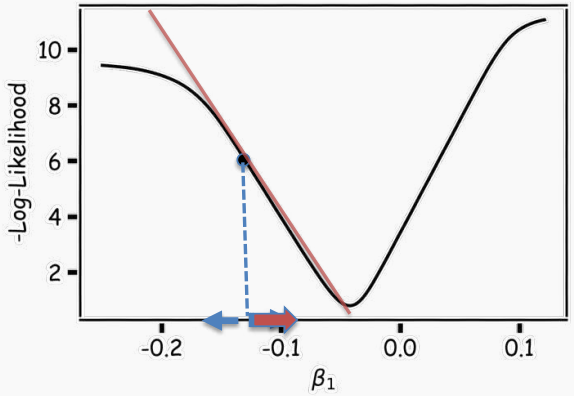


To find the optimal point of a function $\mathcal{L}(W)$

$$\frac{d\mathcal{L}(W)}{dW} = 0 \qquad \text{solve for } W^* = \text{argmin}_W \mathcal{L}(W)$$

And find the W that satisfies that equation. **Sometimes** there is no explicit solution for that.

Estimate of the regression coefficients: gradient descent



A more flexible method is

- Start from a random point
  1. Determine which direction to go to reduce the loss (left or right)
  2. Compute the slope of the function at this point and step to the right if slope is negative or step to the left if slope is positive
  3. Goto to #1

Minimization of the Loss Function

Question: What is the mathematical function that describes the slope?

Derivative

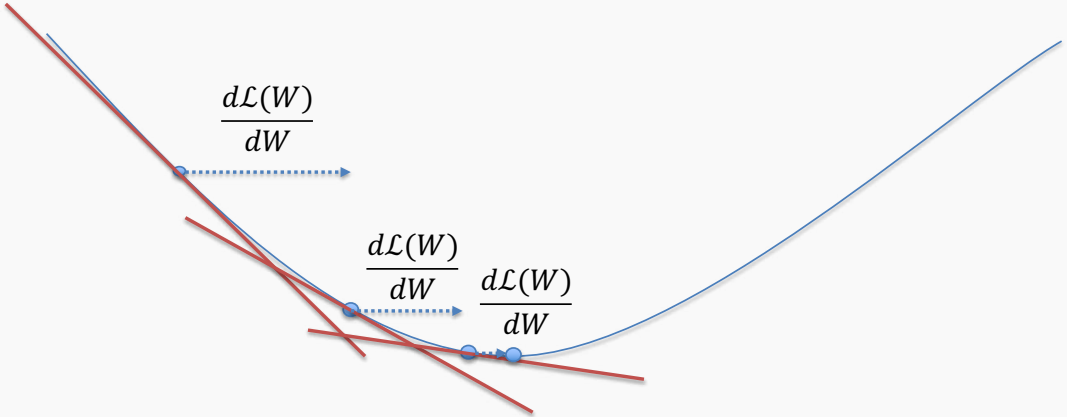
Question: How do we generalize this to more than one predictor?

Take the derivative with respect to each coefficient and do the same sequentially

Question: What do you think is a good approach for telling the model how to change (what is the step size) to become better?

Gradient Descent (cont.)

If the step is proportional to the slope then you avoid overshooting the minimum. How?



Gradient Descent (cont.)

We know that we want to go in the opposite direction of the derivative and we know we want to be making a step proportional to the derivative.

Making a step means:

$$w^{new} = w^{old} + step$$

Step size is proportional to derivative

Opposite direction of the derivative means:

$$w^{new} = w^{old} - \eta \frac{d\mathcal{L}}{dw}$$

Learning Rate

Change to more conventional notation:

$$w^{(i+1)} = w^{(i)} - \eta \frac{d\mathcal{L}}{dw}$$

Gradient Descent

- Algorithm for optimization of first order to finding a minimum of a function.
- It is an iterative method.
- *L* is decreasing much faster in the direction of the negative derivative.
- The learning rate is controlled by the magnitude of η.

$$w^{(i+1)} = w^{(i)} - \eta \frac{d\mathcal{L}}{dw}$$