CSE/STAT 416

Neural Networks

Amal Nanavati
University of Washington
July 25, 2022

Adapted from Hunter Schafer's slides

Roadmap So Far

1. Housing Prices - Regression
   - Regression Model
   - Assessing Performance
   - Ridge Regression
   - LASSO
2. Sentiment Analysis – Classification
   - Classification Overview
   - Logistic Regression
   - K-Nearest Neighbors
   - Decision Trees
   - Ensemble Methods
3. Neural Networks – Image Classification
   - Neural Networks
   - Convolutional Neural Networks

Administrivia

- Timeline:
  - This is the last week (mostly) on supervised learning
  - **This Week:** Neural Networks, Deep Learning
  - **Next Week**: Clustering
  - **Following Week**: Dimensionality Reduction, Recommender Systems
  - **Then**: Course Recap & Final
- Deadlines:
  - HW4 deadline tomorrow, 7/26 11:59PM
    - Up to 7/28 11:59PM with two late days
    - **NOTE**: Change `dummy_model` when copy-pasting the code to generate the CSV!
  - HW5 released Wed 7/27, due Tues 8/2
  - Learning Reflection 6 due Fri, 7/29 11:59PM

submit programming → Gradescope → Kaggle

Addressing LR Uncertainties

Missing Data: Idea 1

- **Idea 1**: Remove rows (datapoints) with missing values.

Train Set

Credit	Term	Income	Loan Safety
fair	5 yrs	$100K	Safe
~~excellent~~	~~3 yrs~~		~~Risky~~
poor	5 yrs	$75K	Risky

Test Set

Credit	Term	Income	Prediction
excellent	3 yrs	$100K	
fair	5 yrs	$20K	
poor	3 yrs		???

Idea 1 is a

BAD IDEA

Missing Data: Idea 2

- **Idea 2**: Remove columns (features) with missing values.

Train Set

Credit	Term	Income	Loan Safety
fair	5 yrs	$100K	Safe
excellent	3 yrs		Risky
poor	5 yrs	$75K	Risky

Test Set

Credit	Term	Income	Prediction
excellent	3 yrs	$100K	
fair	5 yrs	$20K	
poor	3 yrs		

Idea 2 works, but whether you want to use it depends on your dataset, and how important you think that feature is.

Missing Data: Idea 3

- **Idea 3**: Treat missing values as a separate value of the feature (only Decision Trees)

Train Set

Credit	Term	Income	Loan Safety
fair	5 yrs	$100K	Safe
excellent	3 yrs		Risky
poor	5 yrs	$75K	Risky

Test Set

Credit	Term	Income	Prediction
excellent	3 yrs	$100K	
fair	5 yrs	$20K	
poor	3 yrs		

Idea 3 works for Decision Trees (but is not implemented in sklearn)

Missing Data: Idea 4

(Most Commonly Used!)

- **Idea 4**: Replace missing values with a reasonable statistic (Imputation)

Sklearn ⇒ Simple Imputer

Train Set

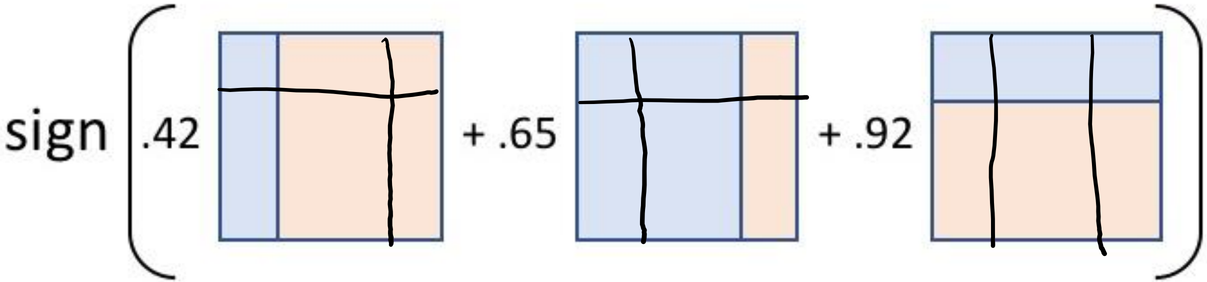
Credit	Term	Income	Loan Safety
fair	5 yrs	$100K	Safe
excellent	3 yrs	$87.5 K	Risky
poor	5 yrs	$75K	Risky
fair	3yr	$87.5K	Safe

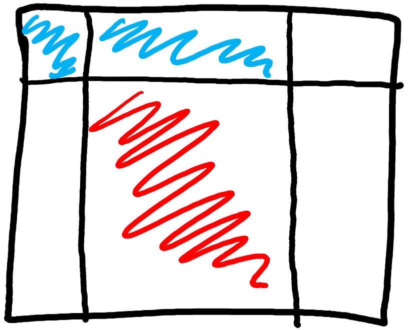
Test Set

Credit	Term	Income	Prediction
excellent	3 yrs	$100K	
fair	5 yrs	$20K	
poor	3 yrs	$87.5 K	

- mean income?
- median?
- mode?
- constant (0) → not typically good

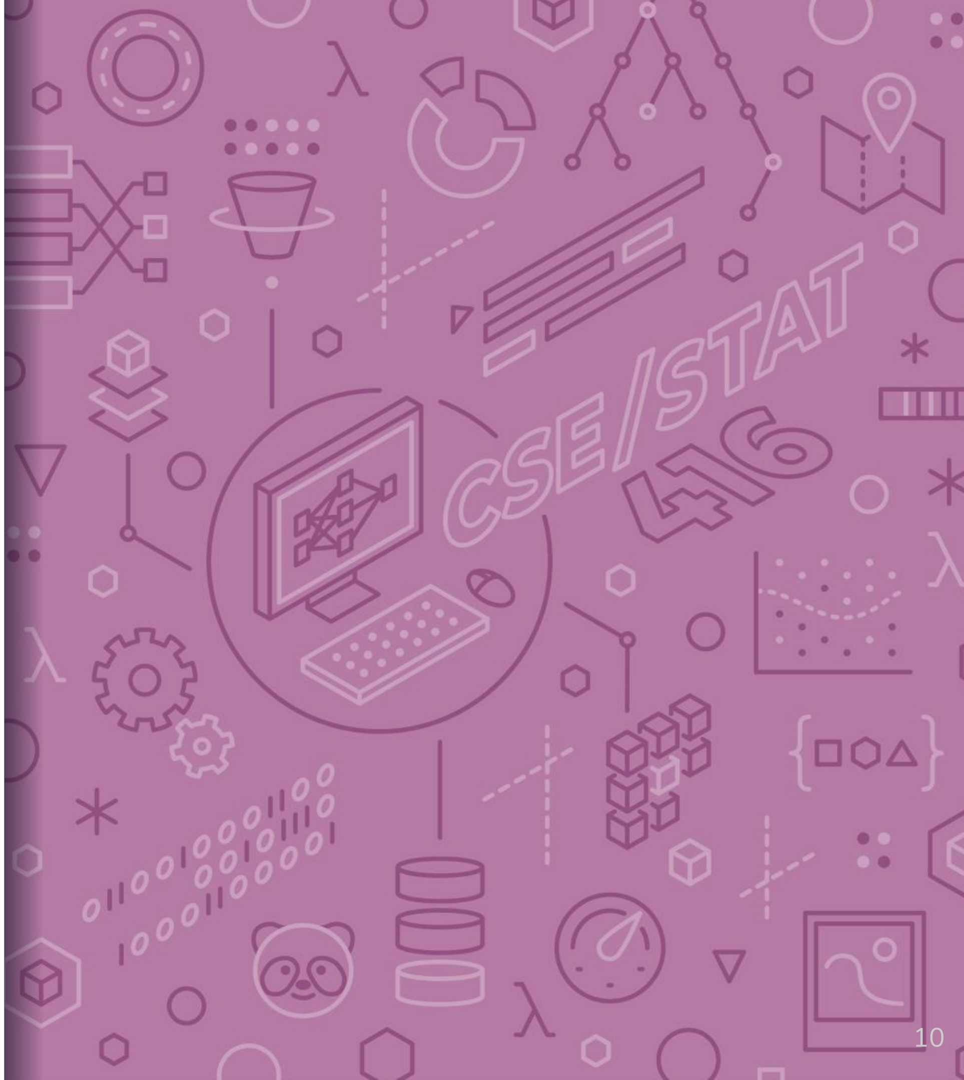
Combining Decision Boundaries





What Model Should You Use?

- 🤷‍♀️ Depends on your goal and your data ☺
- Try different models (in an informed fashion)

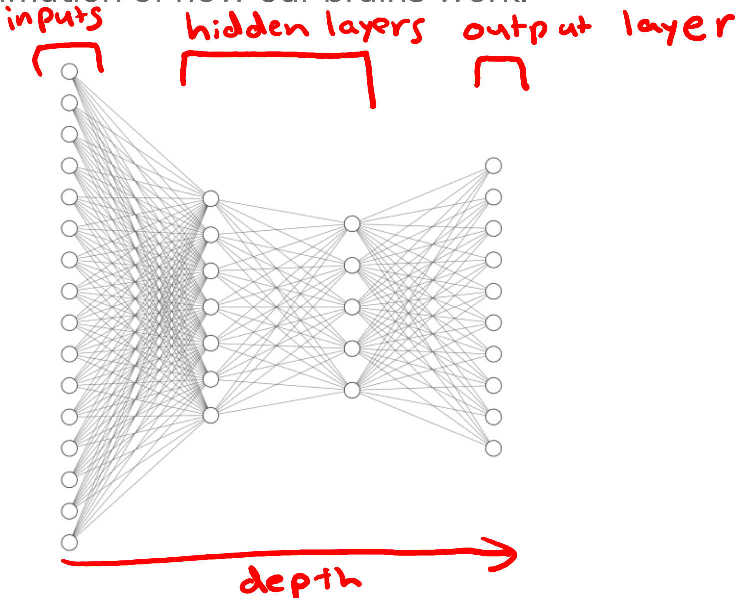


Introduction to Neural Networks

Deep Learning

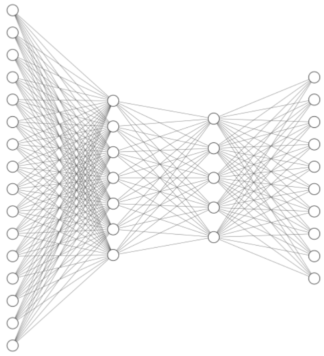
A lot of the buzz about ML recently has come from recent advancements in **deep learning.**

When people talk about "deep learning" they are generally talking about a class of models called **neural networks** that are a loose approximation of how our brains work.



History of Neural Networks

Generally layers and layers of linear models and non-linearities (activation functions).



Have been around for about 50 years

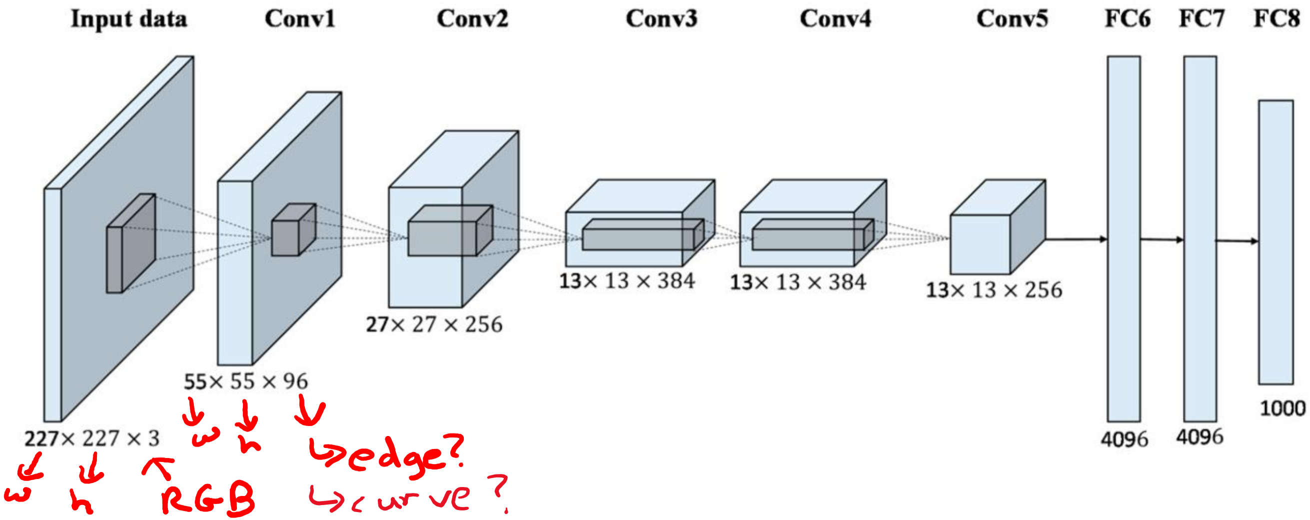
- Fell in "disfavor" in the 90s when simpler models were doing well

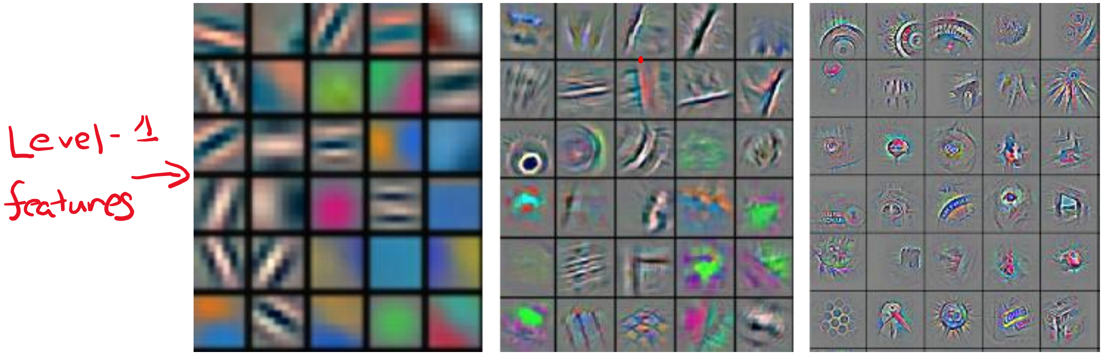
In the last ~~few years~~ decade, have had a huge resurgence

- Impressive accuracy on several benchmark problems
- Have risen in popularity due to huge datasets, GPUs, and improvements to

Popular Neural Network Architectures: CNNs

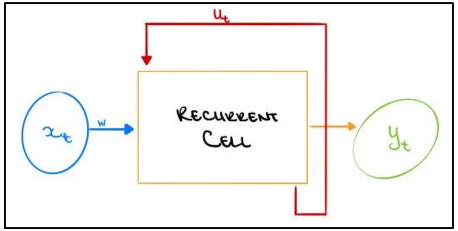
- Convolutional Neural Networks (CNNs) are commonly used in Computer Vision. We'll learn about these on Wed!

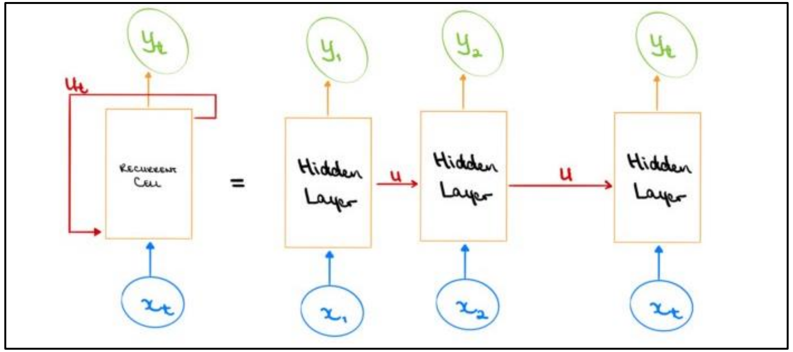


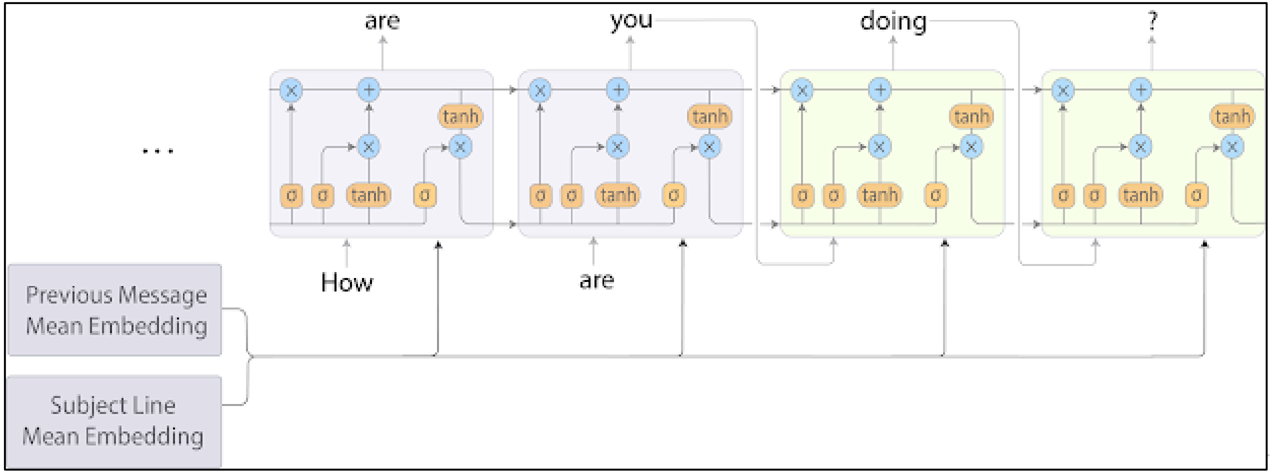


Popular Neural Network Architectures: RNNs

- Recurrent Neural Networks(RNNs) are commonly used in Natural Language Processing, where the model must remember context from earlier in the text.



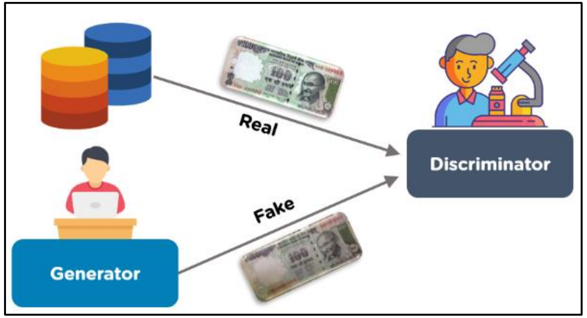


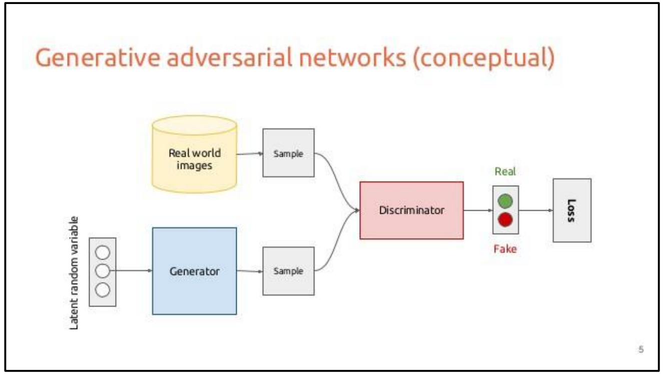


Popular Neural Network Architectures: GANs

Generative Adversarial Networks

- Train two networks together:
  - **Generator Network**: generate fake images
  - **Discriminator Network**: given a real image and a fake image, determine which is fake

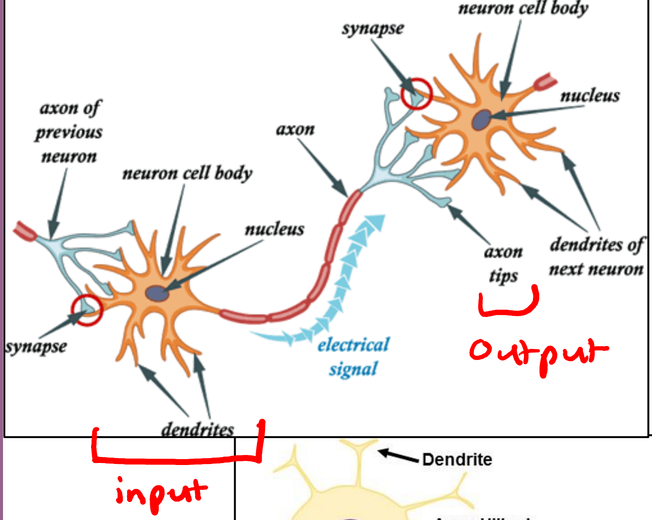


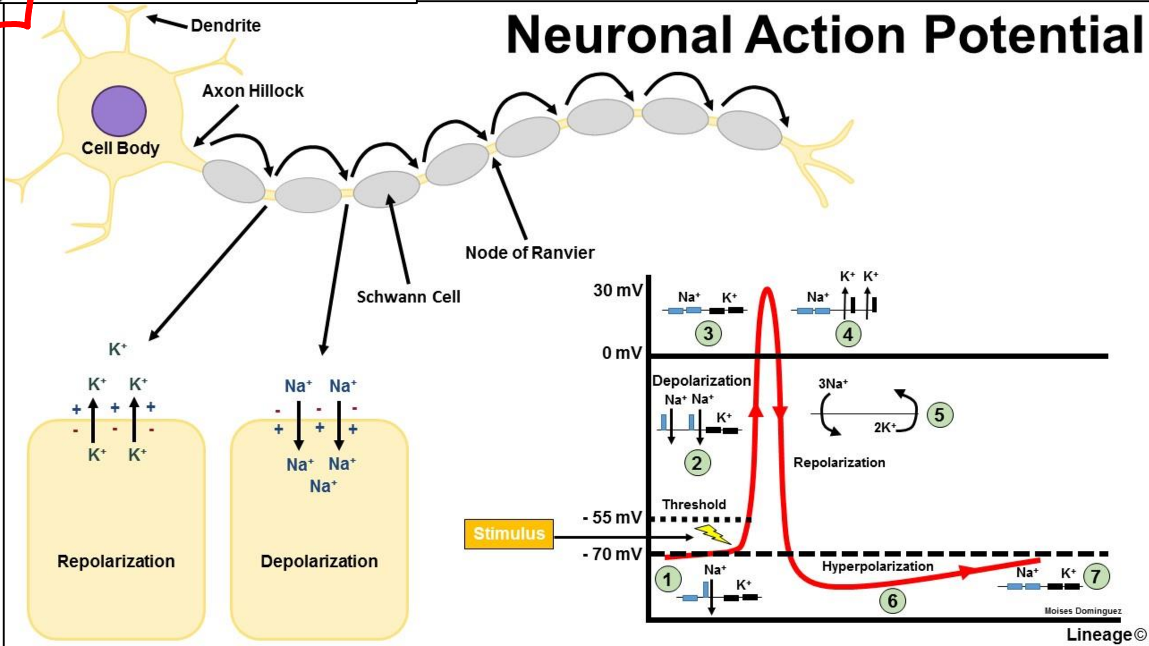


https://thispersondoesnotexist.com/

Biological Inspiration

(Artificial Neural Networks vs. Biological Neural Networks)





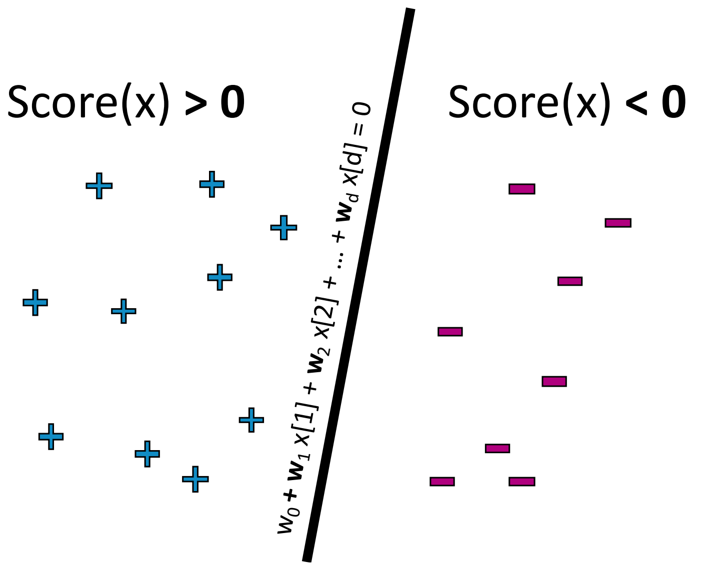
Neural Networks: Technical Details

Recall: Linear Classifier

Remember the linear classifier based on score

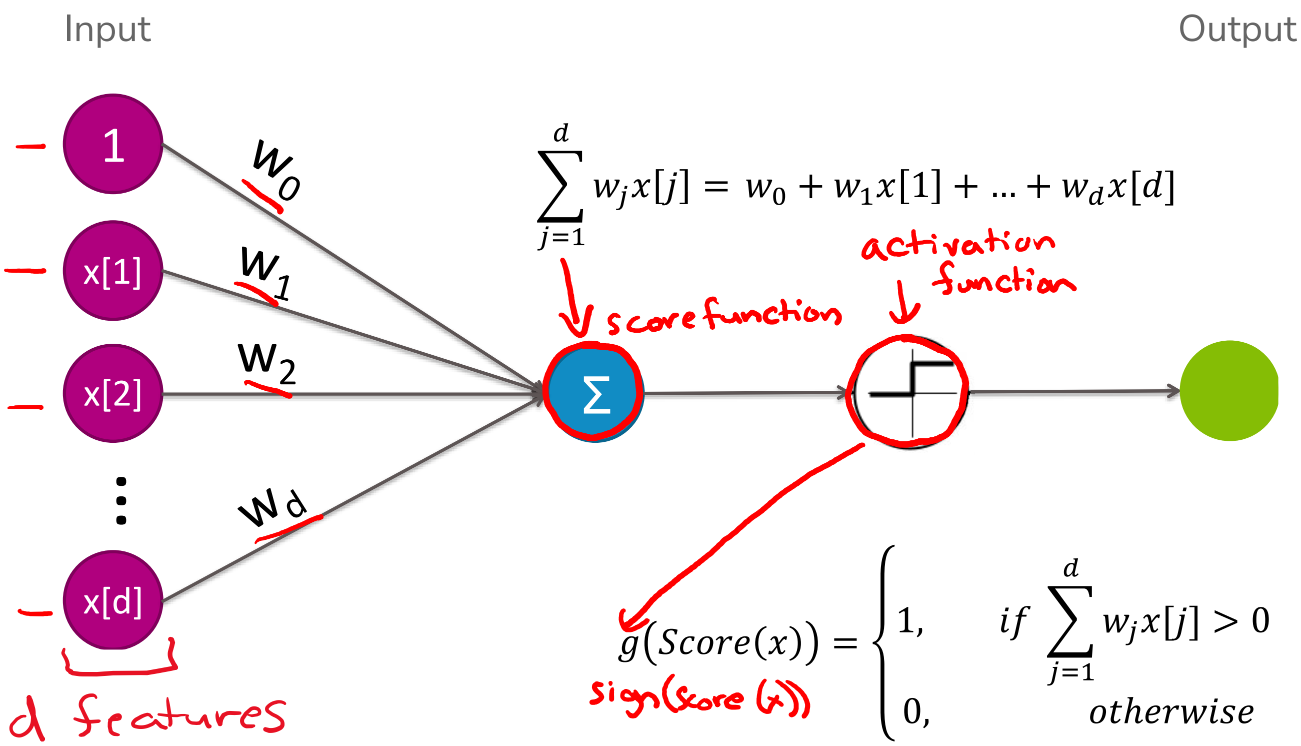
predicting sign(score(x))

$$Score(x) = w_0 + \mathbf{w_1}\, x[1] + \mathbf{w_2}\, x[2] + \ldots + \mathbf{w_d}\, x[d]$$



Perceptron

Graphical representation of this same classifier



This is called a **perceptron**

Poll Everywhere

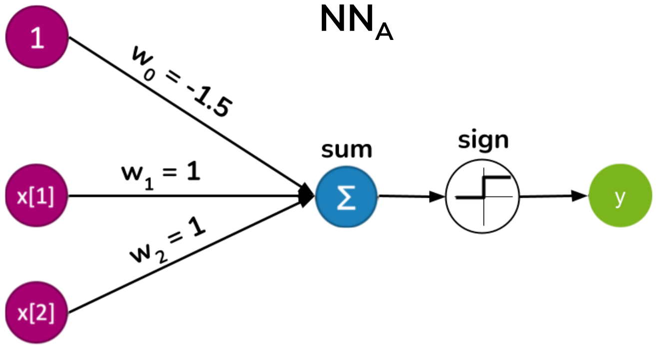
Think

2 mins

pollev.com/cs416

- Match the perceptrons below to the functions they compute.

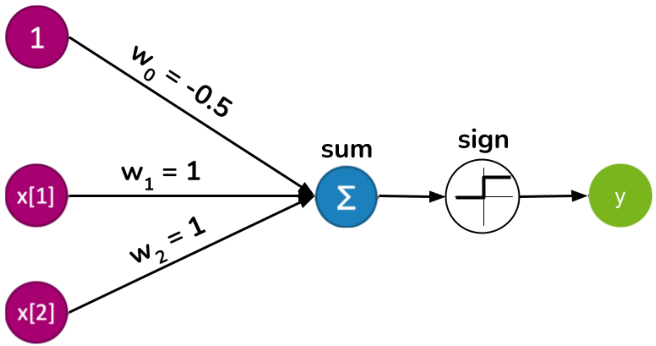
NN_A



$Function_A$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

NN_B



$Function_B$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Poll Everywhere

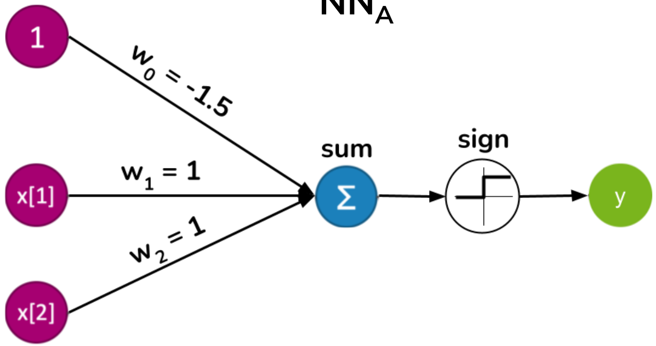
Group

2 mins

pollev.com/cs416

- Match the perceptrons below to the functions they compute.

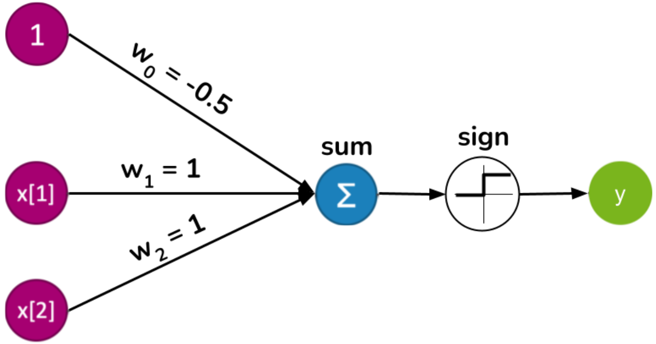
NN_A



$Function_A$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

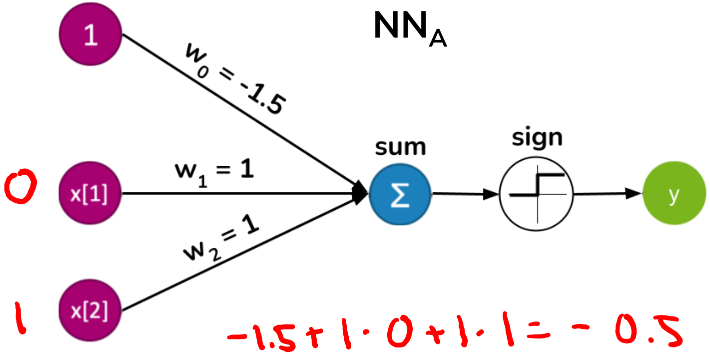
NN_B



$Function_B$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

NN_A



$-1.5 + 1 \cdot 0 + 1 \cdot 1 = -0.5$

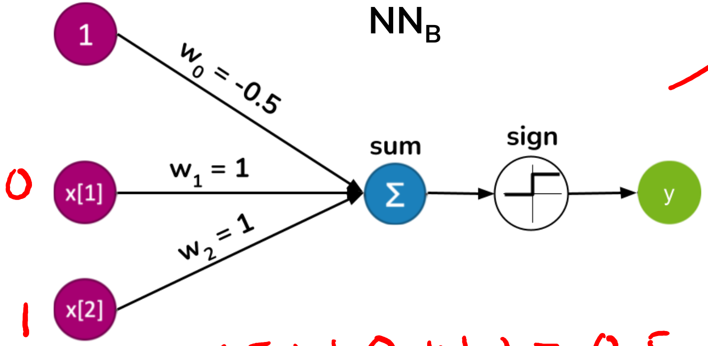
$sign(-0.5) = 0$

$Function_A$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR

NN_B



$-0.5 + 1 \cdot 0 + 1 \cdot 1 = 0.5$

$sign(0.5) = 1$

$Function_B$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

AND

AND: $\text{sign}(-1.5 + x_1 + x_2)$

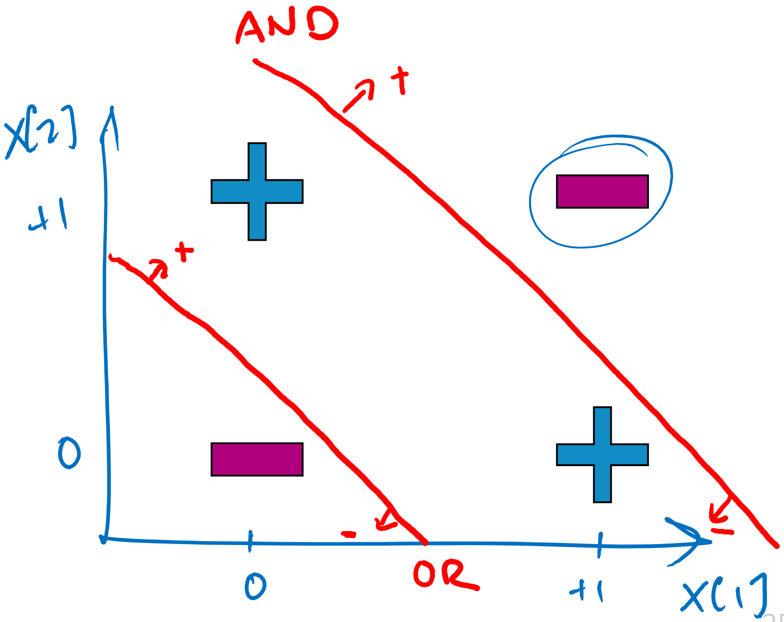
OR: $\text{sign}(-0.5 + x_1 + x_2)$

XOR

The perceptron can learn most boolean functions, but XOR always has to ruin the fun.

This data is not **linearly separable**, therefore can't be learned with the perceptron

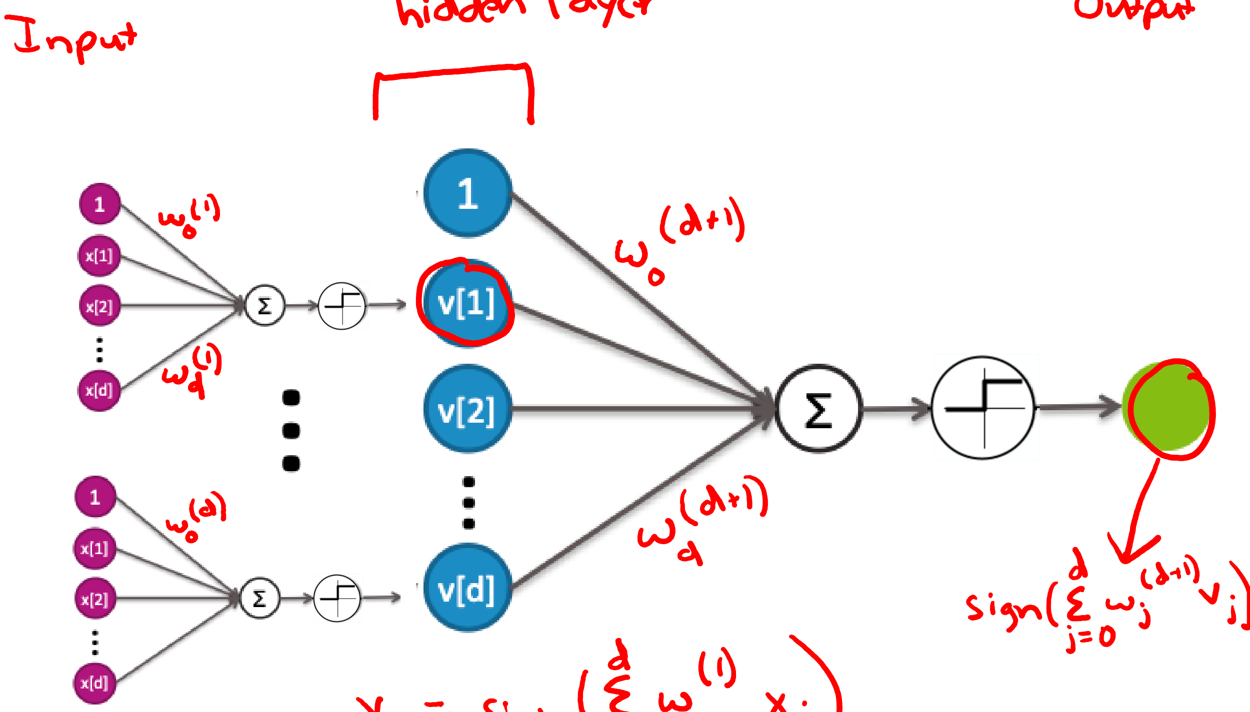
x_1	x_2	y
0	0	0
0	1	1
1	0	0
1	1	1



Multi-Layer Perceptron (Neural Network)

2-layer NN (one hidden layer NN)

Idea: Combine these perceptrons in layers to learn more complex functions.

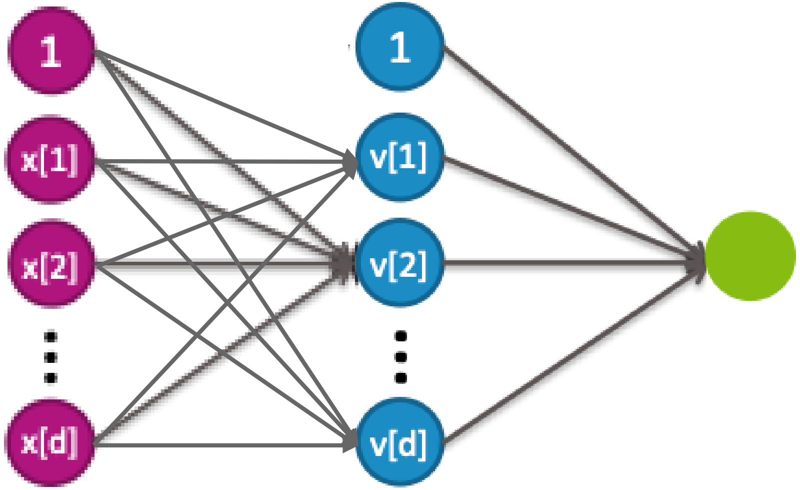


$v_1 = \text{sign}\left(\sum_{i=0}^{d} w_i^{(1)} x_i\right)$

$\text{sign}\left(\sum_{j=0}^{d} w_j^{(d+1)} v_j\right)$

Neural Network Diagram Simplified

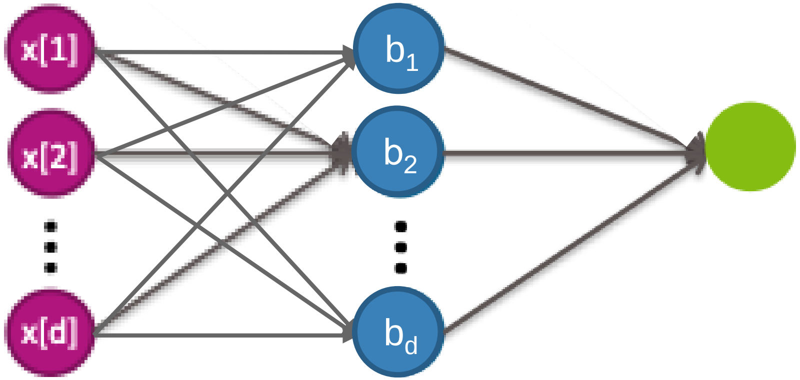
- Since the inputs are the same, typically we combine them in the diagram, with multiple arrows coming out.
- We don't explicitly show the sum and activation function – that is implicitly a part of each node.



Neural Network Diagram Simplified Further

Intercept = bias

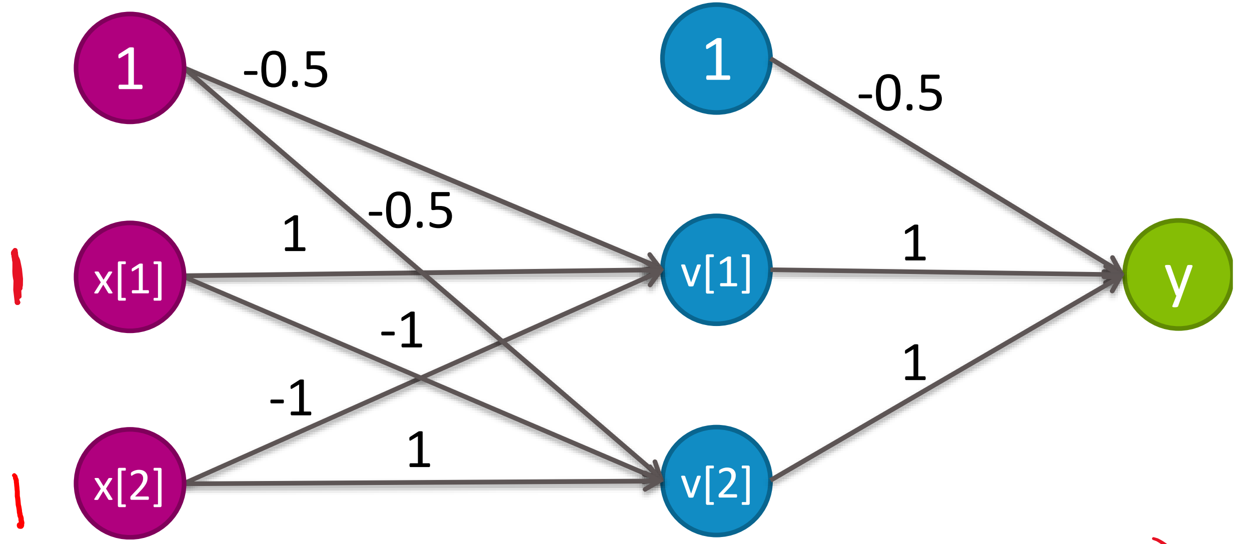
- Oftentimes, the bias is not explicitly shown as another input, and instead written on top of a node.
- You will see both types of diagrams in this course.



XOR

Notice that we can represent

$$x[1]\ XOR\ x[2] = (x[1]\ AND\ !\,x[2])\ OR\ (!\,x[1]\ AND\ x[2])$$



$v_1 = sign(-0.5 + 1\cdot 1 - 1\cdot 1) = sign(-0.5) = 0$

$v_2 = sign(-0.5 - 1\cdot 1 + 1\cdot 1) = sign(-0.5) = 0$

$y = sign(-0.5 + 1\cdot 0 + 1\cdot 0) = sign(-0.5) = 0$

XOR

This is a 2-layer neural network

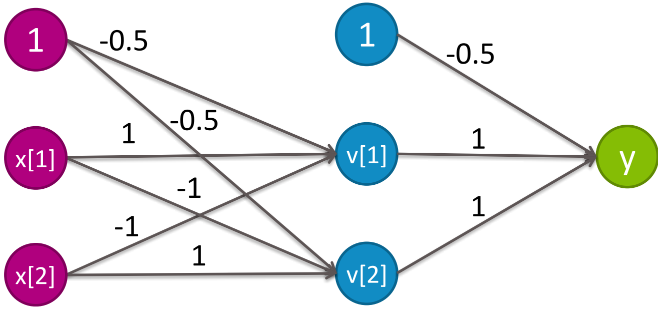
$$y = x[1]\ XOR\ x[2] = (x[1]\ AND\ !\,x[2])\ OR\ (!\,x[1]\ AND\ x[2])$$

$$v[1] = (x[1]\ AND\ !\,x[2])$$
$$= g(-0.5 + x[1] - x[2])$$

sign

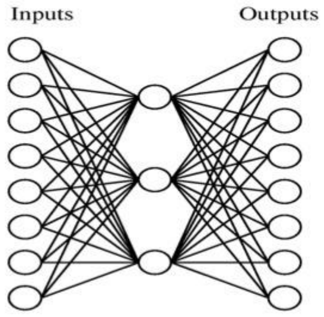
$$v[2] = (!\,x[1]\ AND\ x[2])$$
$$= g(-0.5 - x[1] + x[2])$$

$$y = v[1]\ OR\ v[2]$$
$$= g(-0.5 + v[1] + v[2])$$



Neural Network

Two layer neural network (alt. one hidden-layer neural network)



Single

$$out(x) = g\left(w_0 + \sum_j w_j x[j]\right)$$

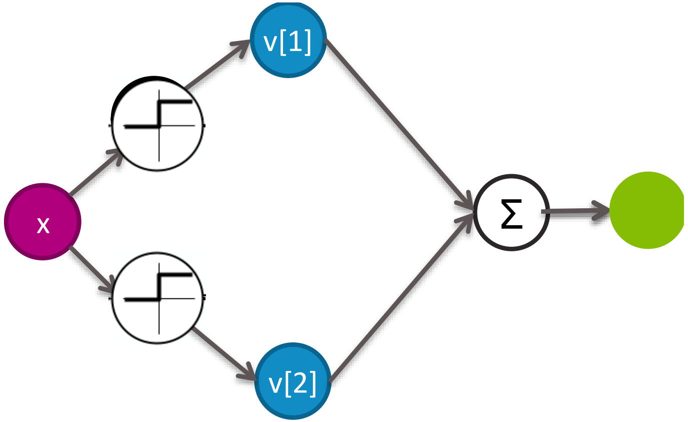
1-hidden layer

$$out(x) = g\left(w_0 + \sum_k w_k \underbrace{g\left(w_0^{(k)} + \sum_j w_j^{(k)} x[j]\right)}_{v_k}\right)$$

Power of 2-layer NN

A surprising fact is that a 2-layer network can represent any function, if we allow enough nodes in hidden layer.

For this example, consider regression function with one input.



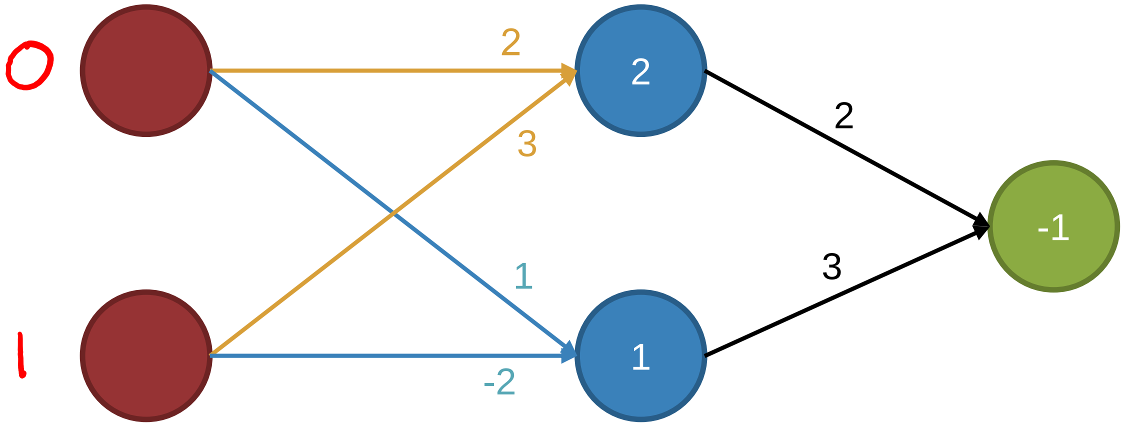
See more here: http://neuralnetworksanddeeplearning.com/chap4.html

Poll Everywhere

Think

2 mins

- Compute the output for input (0, 1). There is a sign activation function on the hidden layers and output layer.

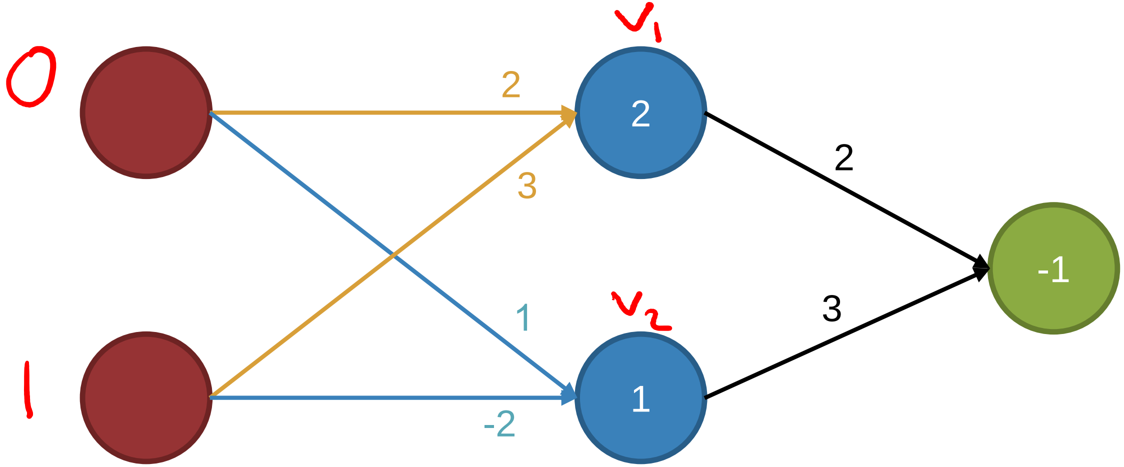


Poll Everywhere

Group

2 mins

- Compute the output for input (0, 1). There is a sign activation function on the hidden layers and output layer.

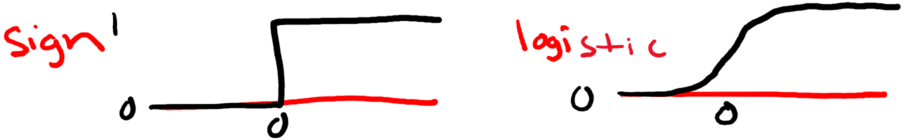


$$v_1 = \text{sign}(2 + 2\cdot 0 + 3\cdot 1) = \text{sign}(5) = 1$$

$$v_2 = \text{sign}(1 + 1\cdot 0 - 2\cdot 1) = \text{sign}(-1) = 0$$

$$y = \text{sign}(-1 + 2\cdot 1 + 3\cdot 0) = \text{sign}(1) = 1$$

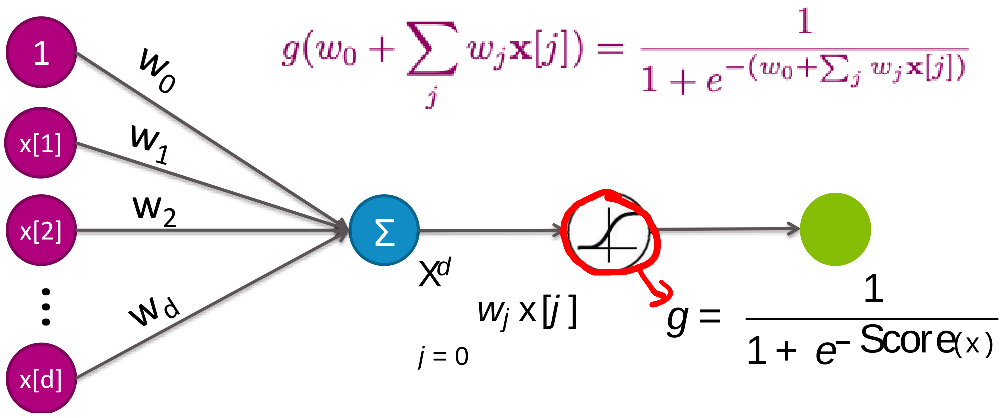
Activation Function



Before, we were using the sign activation function.

- This is not generally used in practice.
  - Not differentiable
  - No notion of confidence

What if we use the logistic function instead?



Activation Functions

Hyper-parameter

- Sigmoid (Logistic)

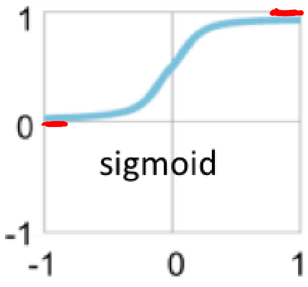
-Historically popular, but (mostly) fallen out of favor

- Neuron's activation saturates

(weights get very large -> gradients get small)

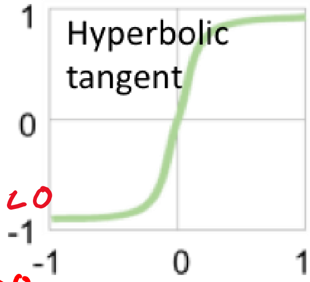
- Not zero-centered -> other issues in the gradient steps

-When put on the output layer, called "softmax" because interpreted as class probability (soft assignment)



- Hyperbolic tangent $g(x) = \tanh(x)$

-Saturates like sigmoid unit, but zero-centered

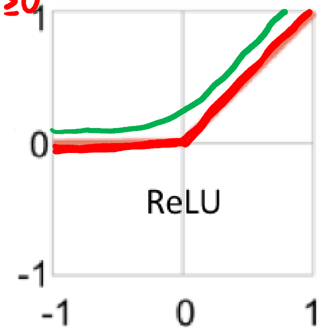


- Rectified linear unit (ReLU) $g(x) = x^+ = \max(0,x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$

-Most popular choice these days

-Fragile during training and neurons can "die off"… be careful about learning rates

-"Noisy" or "leaky" variants



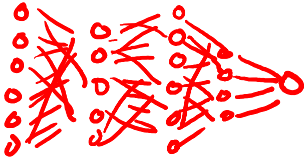
- Softplus $g(x) = \log(1+\exp(x))$

-Smooth approximation to rectifier activation

Classification or Regression

You can use neural networks for classification and regression!

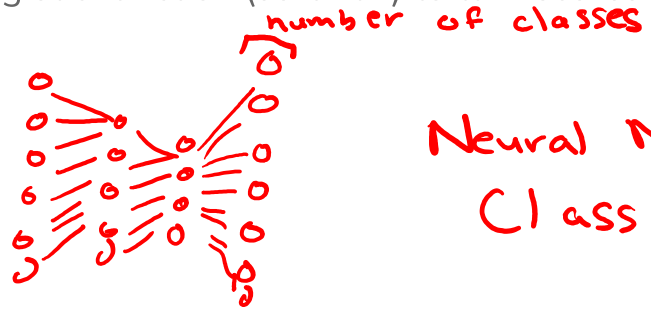
Regression



The output layer will generally have one node that is the output (outputs a single number). Don't apply activation to the last layer.

Classification

The output layer will have one node per class. Usually take the node with the highest score as the prediction for an example. Can also use the logistic function (softmax) to turn scores into probabilities!



Overfitting NNs

Are NNs likely to overfit? **YES**.

- Consequence of being able to fit any function!

How to avoid overfitting?

- Get more training data
- Few hidden nodes / better architecture
  - **Rule of thumb:** 3-layer NNs outperform 2-layer NNs, but going deeper only helps if you are very careful (different story next time with convolutional neural networks)
- Regularization
  - Dropout
- Early stopping → will revisit later in lecture

Poll Everywhere

Think

1 min

pollev.com/cs416

- The models we have seen so far have ≤ 100 parameters (weights, biases). How many parameters do you think GPT-3 has?

Class Vote: #4 #4 #2 #1 #3

(a) 0.4B (b) 1B (c) 12B (d) 90B (e) 175B

https://beta.openai.com/playground

Playground

How should I teach an introduction to neural networks?

There is no one answer to this question. Depending on the level of the students and the resources available, the approach to teaching neural networks will vary. A few potential ideas include:

-Starting with a basic introduction to artificial neural networks and how they work

-Progressing to more specific types of neural networks, such as convolutional neural networks

-Using hands-on activities or projects to help students understand how neural networks work

-Providing resources for students to explore on their own outside of class

Brain Break

3:30

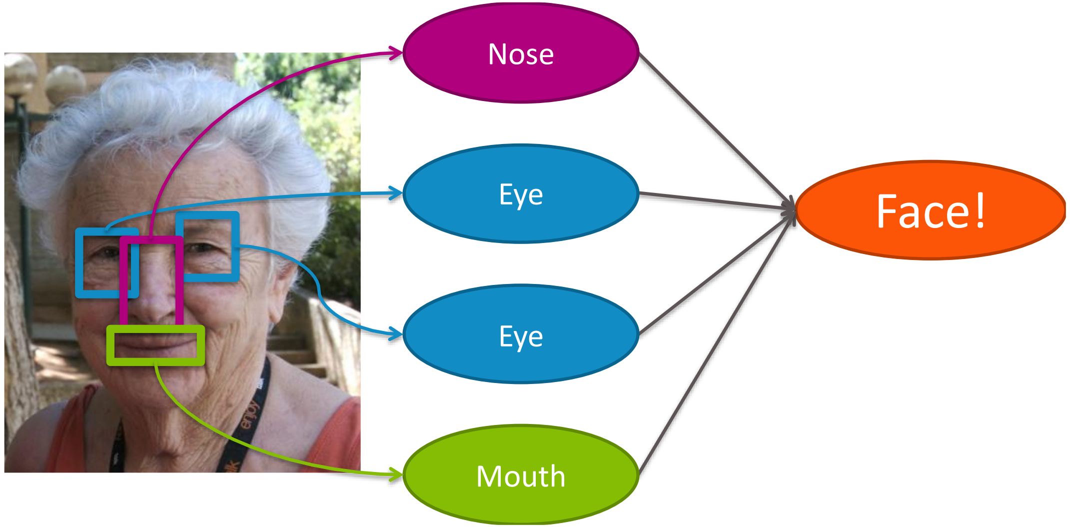
Application to Computer Vision

SKIPPED

Image Features

Features in computer vision are local detectors

- Combine features to make prediction



In reality, these features are much more low level (e.g. Corner?)

SKIPPED

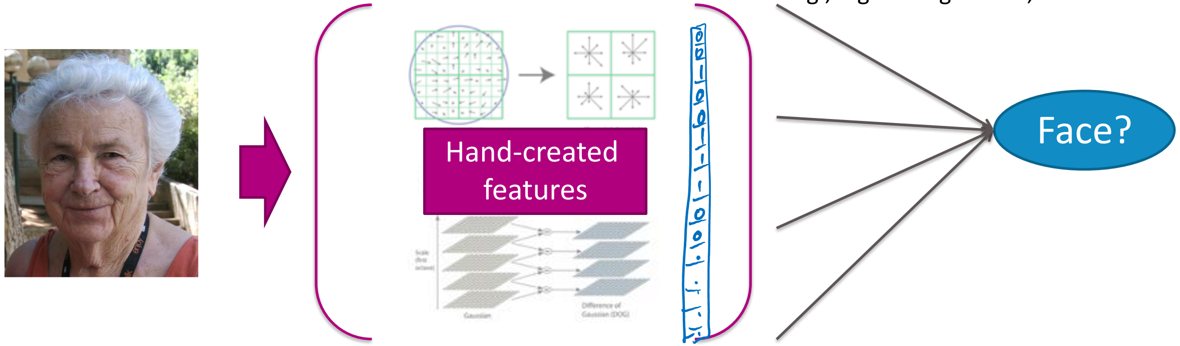
The Past

A popular approach to computer vision was to make hand-crafted features for object detection

Input

Extract features

Use simple classifier
e.g., logistic regression, SVMs

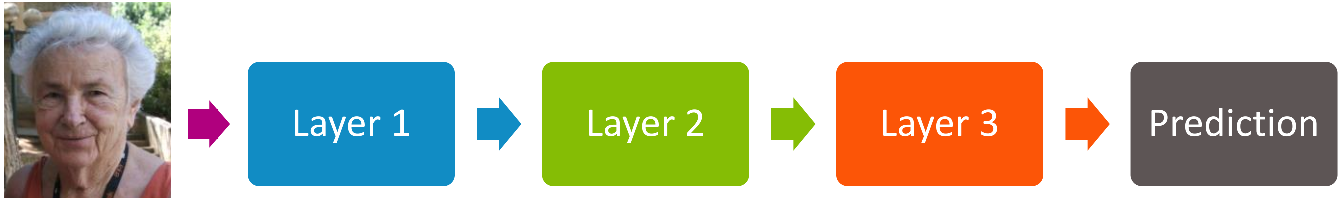


Relies on coming up with these features by hand (yuck!)

NNs to the Rescue

SKIPPED

Neural Networks implicitly find these low level features for us!



Example detectors learned			
Example interest points detected			

[Zeiler & Fergus '13]

Each layer learns more and more complex features

Poll Everywhere

Think

1 min

pollev.com/cs416

SKIPPED

- The models we have seen so far have ≤ 100 parameters (weights, biases). How many parameters do you think DALL-E Mini has?

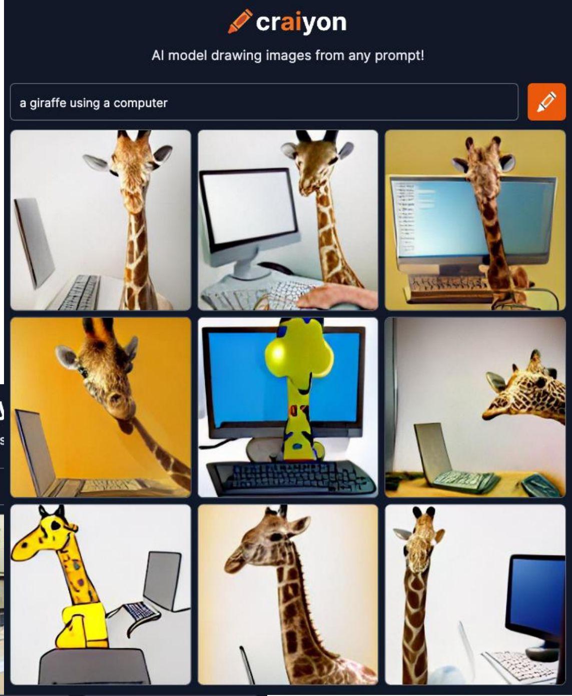
(a) 0.4B

(b) 1B

(c) 12B

(d) 90B

(e) 175B



https://www.craiyon.com/
(formerly Dall-E Mini)

Training Neural Networks

Learning Coefficients

So the idea of neural networks might make sense, but how do we actually go about learning the coefficients in the layers?

First we need to define a quality metric or cost function

- For regression, generally use MSE or RMSE
- For classification, generally use something call the Cross Entropy loss.

Can we use gradient descent here? Actually yes!

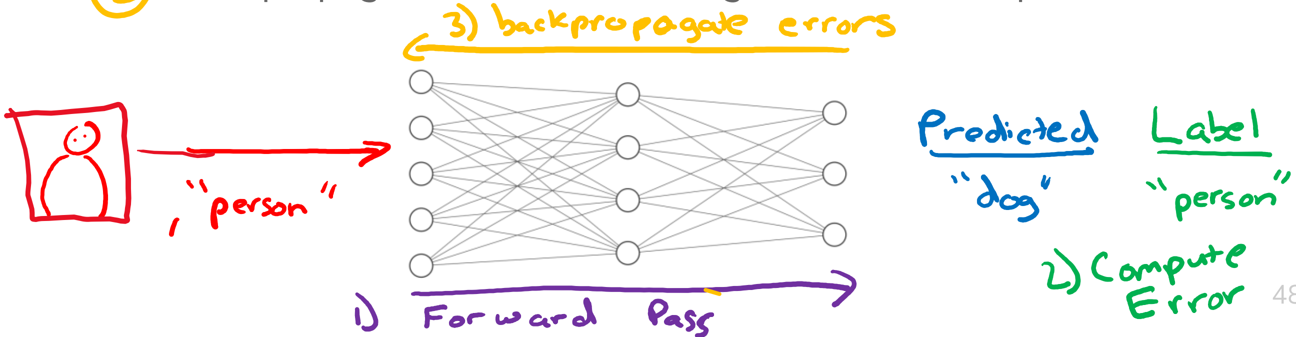
- How do we take the derivative of a network?
- Are there convergence guarantees?

Backpropagation

What does gradient descent do in general? Have the model make predictions and update the model in a special way such that the new weights have lower error.

To do gradient descent with neural networks, we generally use **backpropagation.**

1. Do a forward pass of the data through the network to get predictions
2. Compare predictions to true values
3. Backpropagate errors so the weights make better predictions



Training a NN

divide train set into batchess

It's pretty expensive to do this update for the entire dataset at once, so it's common to break it up into small batches to process individually.

However, processing each batch only once isn't enough. You generally have to repeatedly update the model parameters. We call an iteration that goes over every batch once an **epoch**.

epochs: how long to train for

```
for i in range(num_epochs):
  for batch in batches(training_data):
    preds = model.predict(batch.data) # Forward pass
    diffs = compare(preds, batch.labels)  # Compare
    model.backprop(diffs)              # Backpropagation
```

- epoch = one step of training the NN on the whole dataset
- to train the NN, use many epochs

NN Convergence

In general, loss functions with neural networks are **not** convex.

This means the backprop algorithm for gradient descent will only converge to a local optima.

This means that how you initialize the weights is really important and can impact the final result.

How should you initialize weights? ¯_(ツ)_/¯

- Usually people do random initialization

People also use adaptive ways of changing the learning rate to reduce the empirical likelihood of getting stuck in local minima.

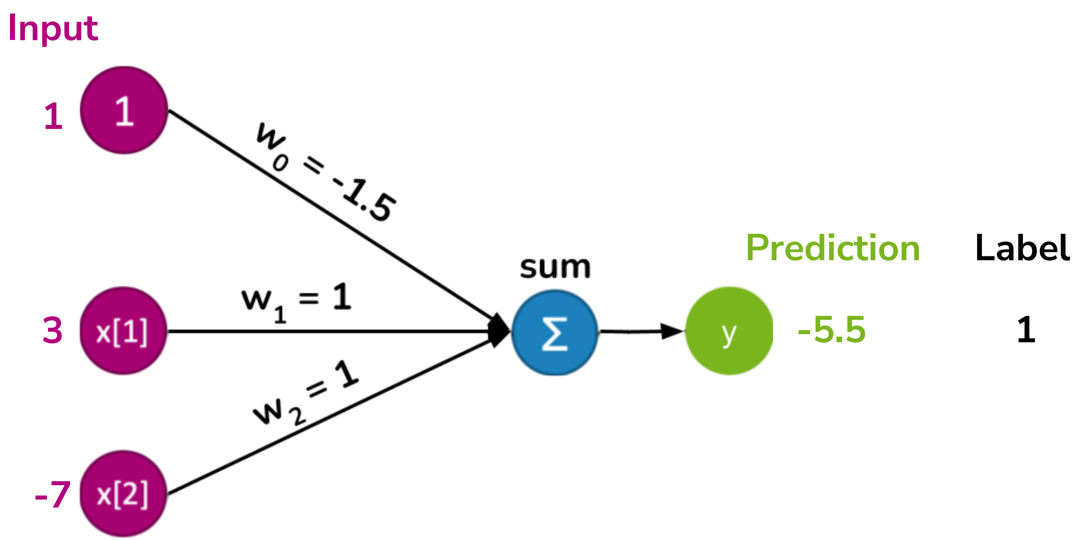
Poll Everywhere

Think

2 mins

pollev.com/cs416

- Consider the below neural network, used for regression (hence, no activation on the last layer).
- The input, prediction, and actual label are shown.
- To move the prediction slightly closer to the label, would you (**increase / decrease**) w_1?



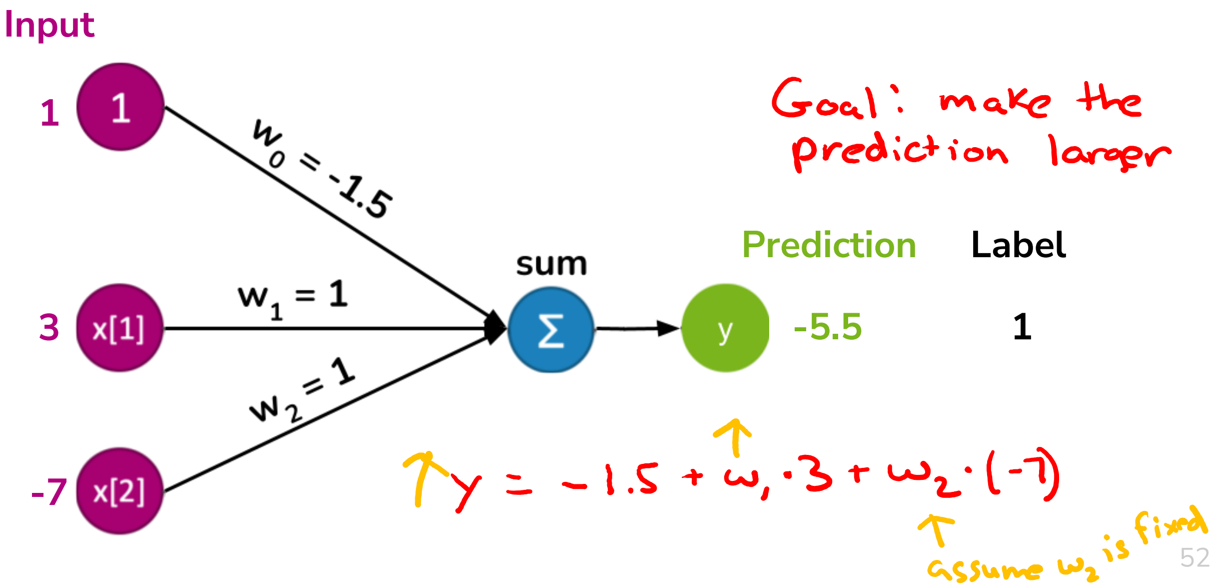
Poll Everywhere

Group

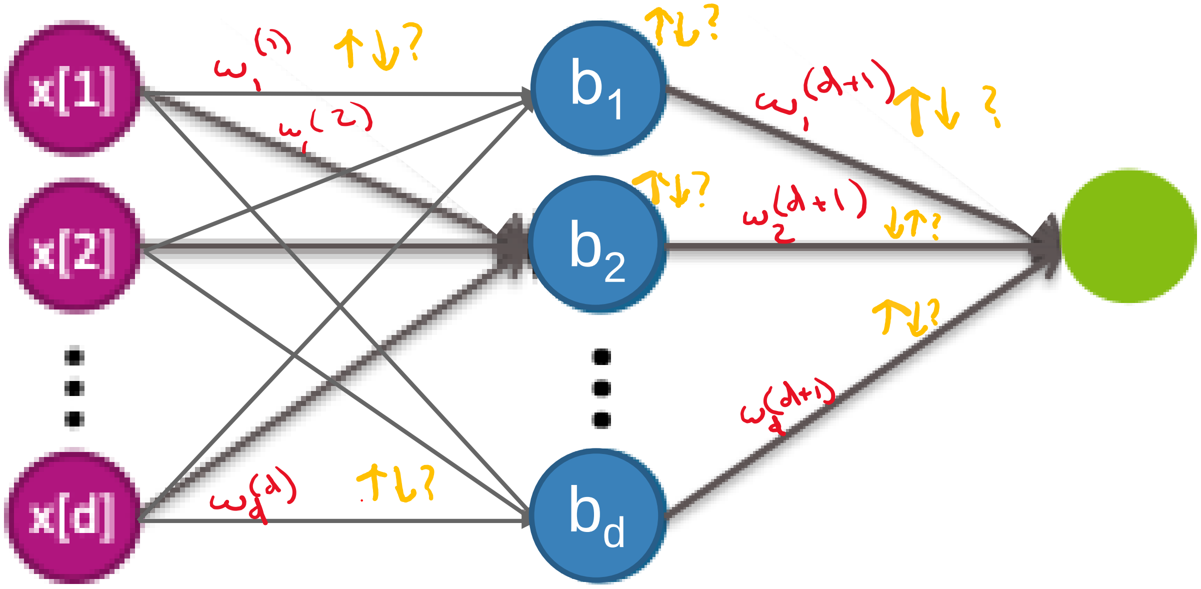
1 mins

~~pollev.com/cs416~~

- Consider the below neural network, used for regression (hence, no activation on the last layer).
- The input, prediction, and actual label are shown.
- To move the prediction slightly closer to the label, would you (**increase / decrease**) w_1?



Backpropogation Intuition on Multiple Layers



Hyper-parameter Tuning

Training NN

Neural Networks have MANY hyperparameters

- How many hidden layers and hidden neurons?
- What activation function?
- What is the learning rate for gradient descent?
- What is the batch size?
- How many epochs to train?
- And much much more!

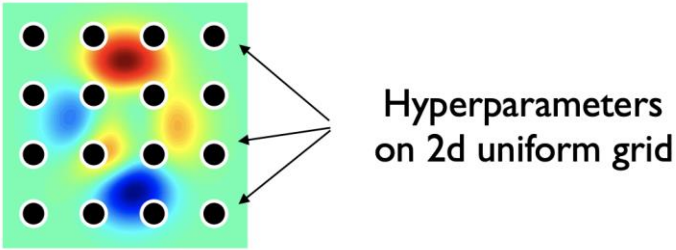
How do you decide these values should be? ¯_(ツ)_/¯

The most frustrating thing is that we don’t have a great grasp on how these things impact performance, so you generally have to try them all.

Hyperparameter Optimization

How do we choose hyperparameters to train and evaluate?

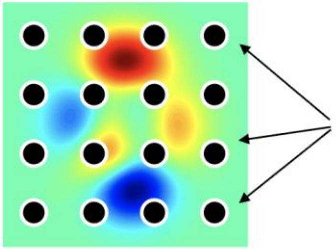
Grid search:



Hyperparameter Optimization

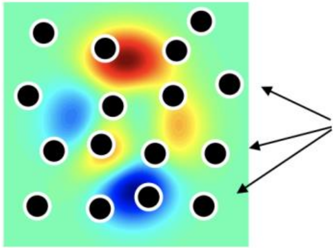
How do we choose hyperparameters to train and evaluate?

Grid search:



Hyperparameters on 2d uniform grid

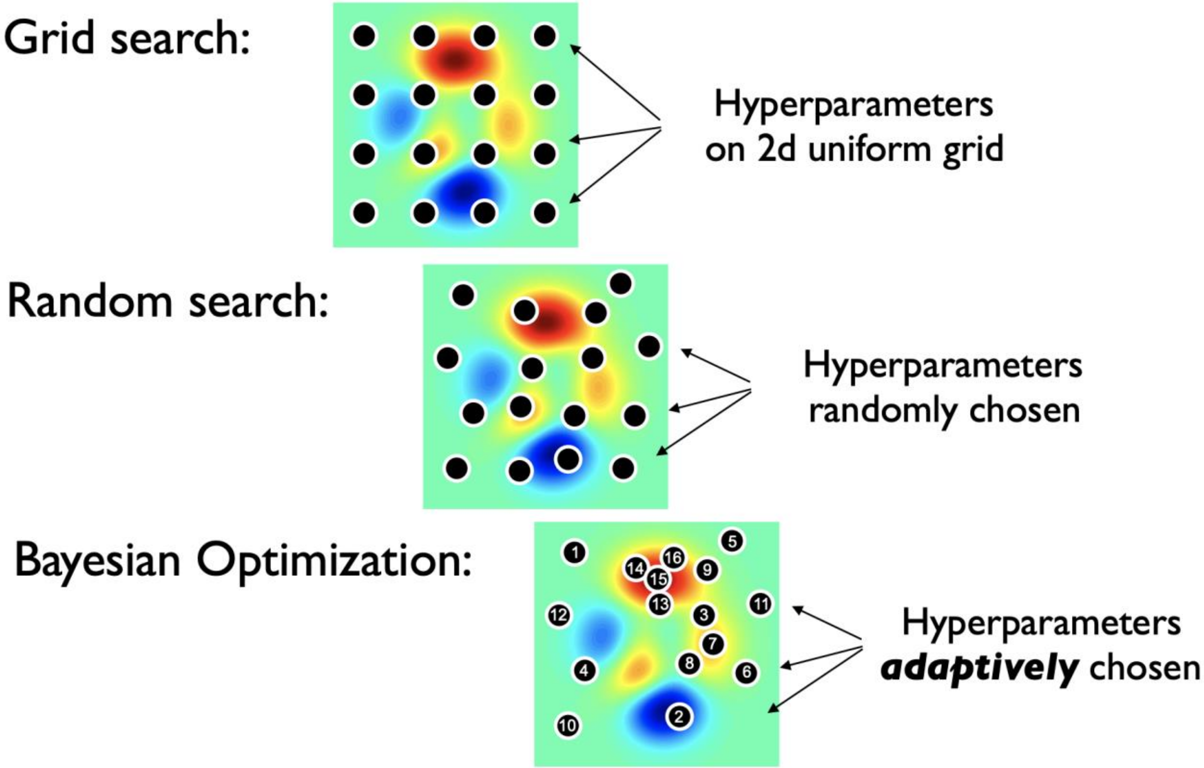
Random search:



Hyperparameters randomly chosen

Hyperparameter Optimization

How do we choose hyperparameters to train and evaluate?



Hyperparameter Optimization

Early Stopping

Recent work attempts to speed up hyperparameter evaluation by stopping poor performing settings before they are fully trained.

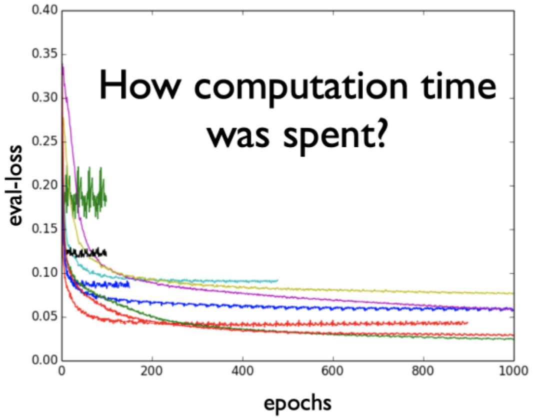
Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. arXiv:1406.3896, 2014.

Alekh Agarwal, Peter Bartlett, and John Duchi. Oracle inequalities for computationally adaptive model selection. COLT, 2012.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

András György and Levente Kocsis. Efficient multi-start strategies for local search algorithms. JAIR, 41, 2011.

Li, Jamieson, DeSalvo, Rostamizadeh, Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. ICLR 2016.



Tips on Hyperparameter Optimization

In general, hyperparameter optimization is a non-convex optimization problem where we know very little about how the function behaves.

Your time is valuable and compute time is cheap. Write your code to be modular so you can use compute time to try a range of values.   grad student descent

Tools for different purposes

- Very few evaluations: use random search (and pray)
- Few evaluations and long-run computations: See last slide
- Moderate number of evaluations: Bayesian optimization
- Many evaluations possible: Use random search. Why overthink it?

Recap

Theme: Details of neural networks and how to train them

Ideas:

Perceptron (Single-Layer Neural Network)

Neural Networks

Using the logistic function to turn Score to probability

Logistic Regression

Minimizing error vs maximizing likelihood

Gradient Ascent

Effects of learning rate

Overfitting with logistic regression

- Over-confident (probabilities close to 0 or 1)
- Regularization