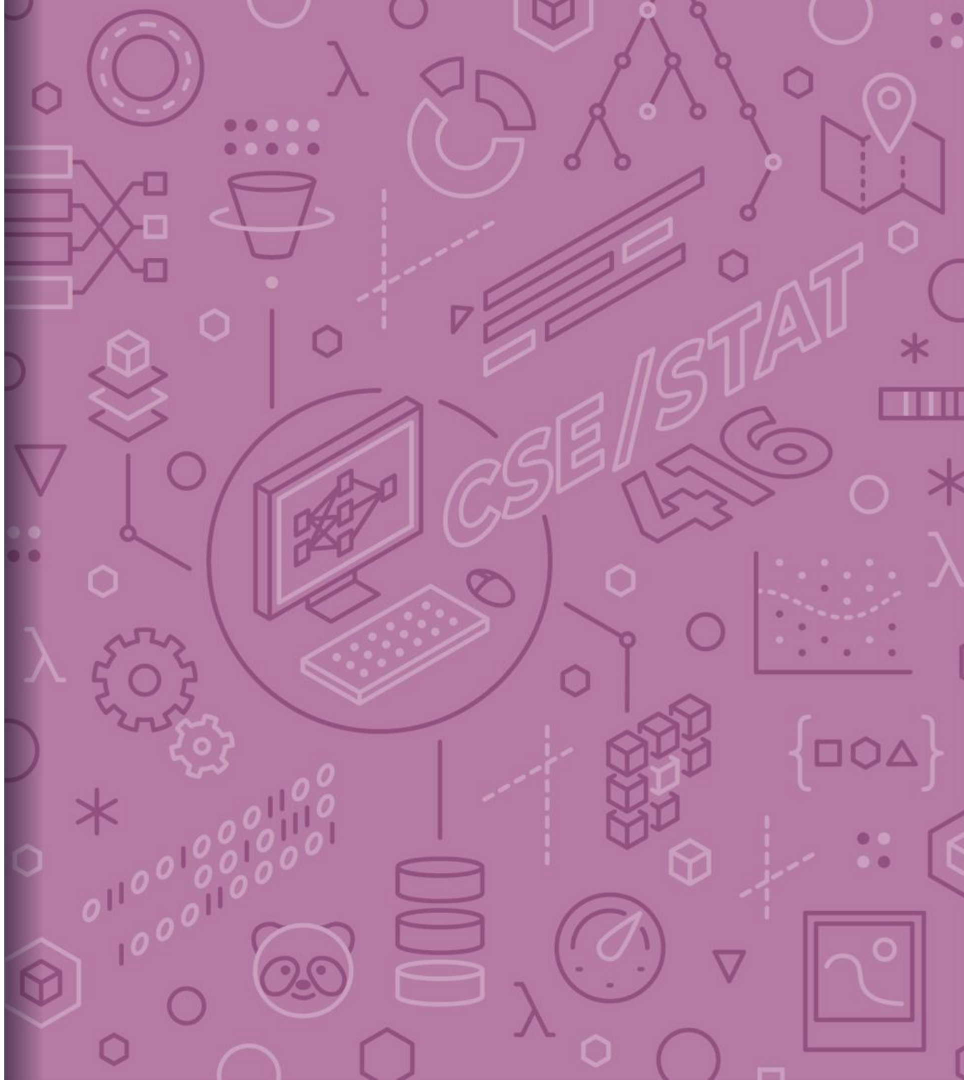
CSE/STAT 416

Neural Networks

Pre-Lecture Videos

Hunter Schafer
Paul G. Allen School of Computer Science & Engineering
University of Washington

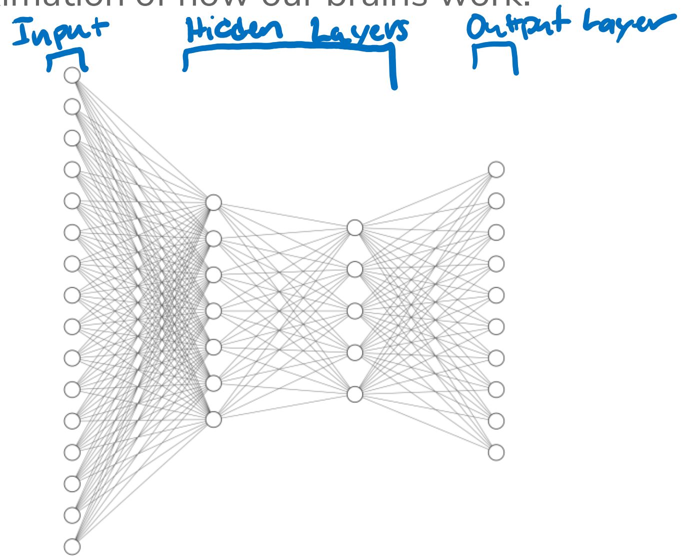
May 1 , 2023



Deep Learning

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

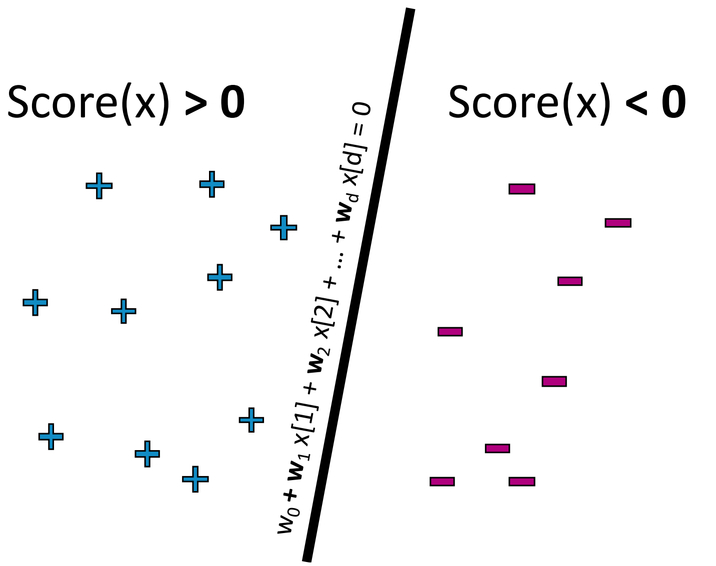
When people talk about “deep learning” they are generally talking about a class of models called **neural networks** that are a loose approximation of how our brains work.



Recall: Linear Classifier

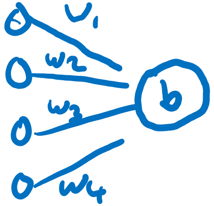
Remember the linear classifier based on score

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



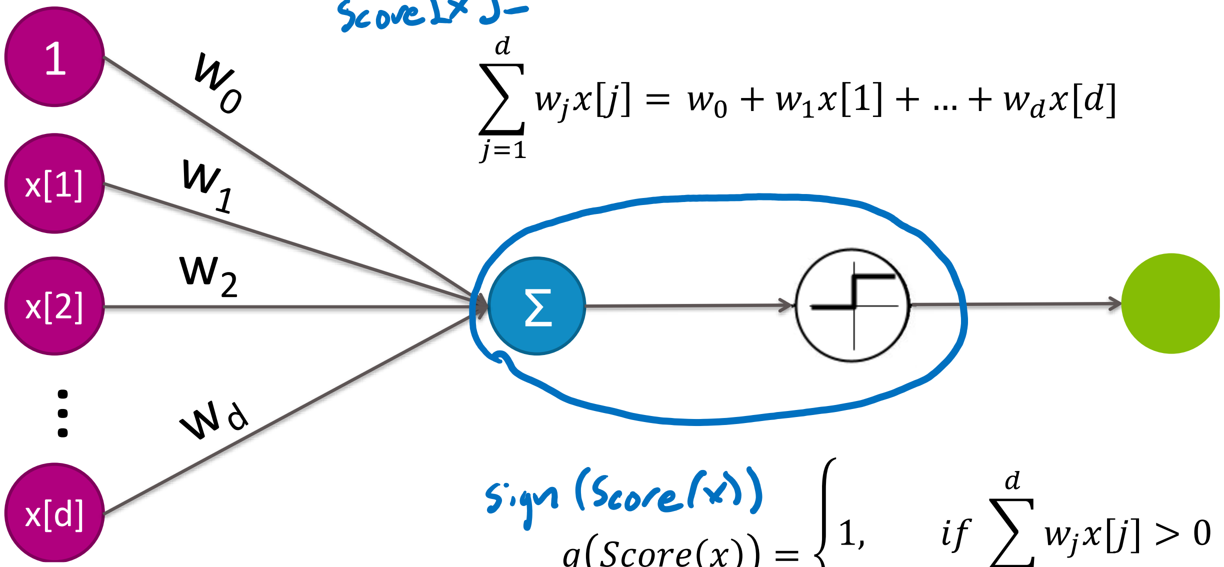
Perceptron

Graphical representation of this same classifier



Input

Output



Score[x]=

$$\sum_{j=1}^{d} w_j x[j] = w_0 + w_1 x[1] + \ldots + w_d x[d]$$

Sign(Score(x))

$$g(Score(x)) = \begin{cases} 1, & if \sum_{j=1}^{d} w_j x[j] > 0 \\ 0, & otherwise \end{cases}$$

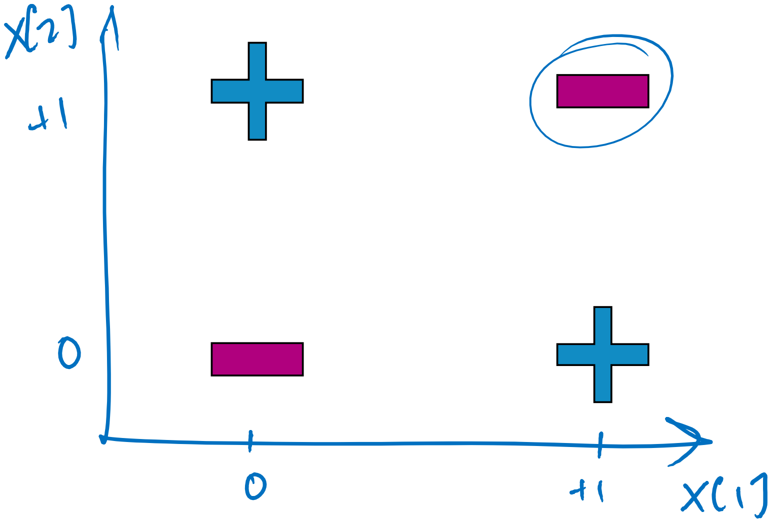
This is called a **perceptron**

XOR

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

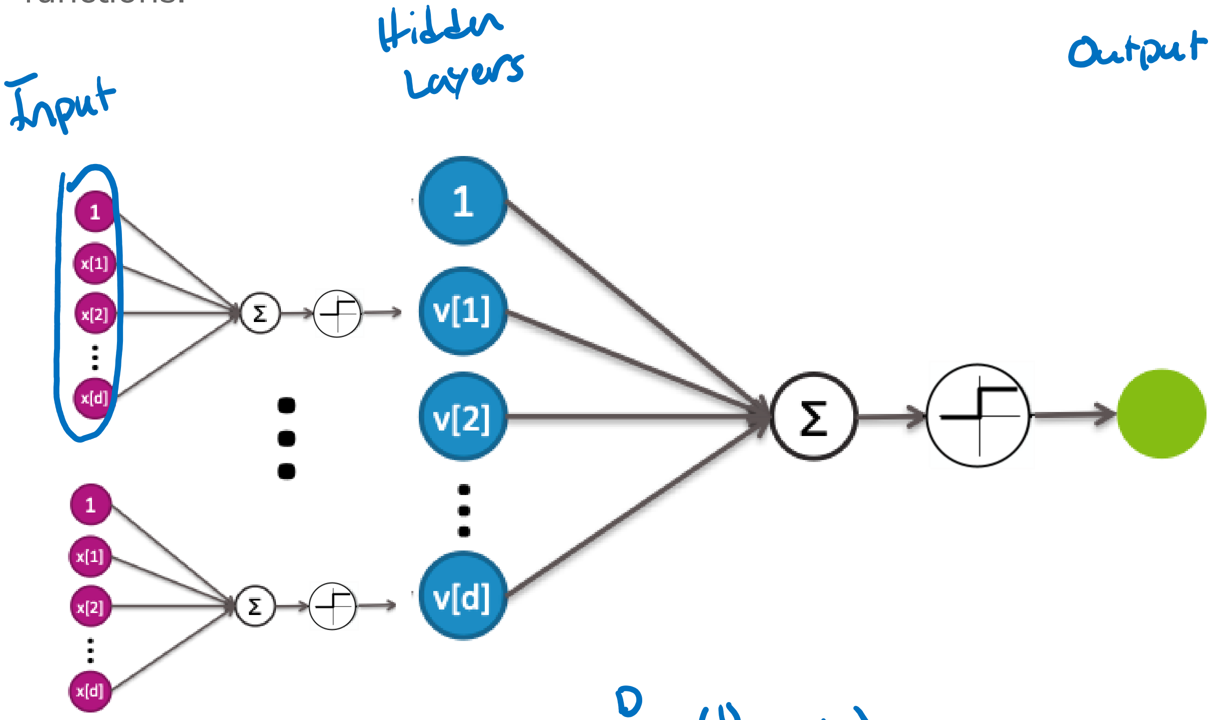
This data is not **linearly separable**, therefore can't be learned with the perceptron

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



Multi-Layer Perceptron (Neural Network)

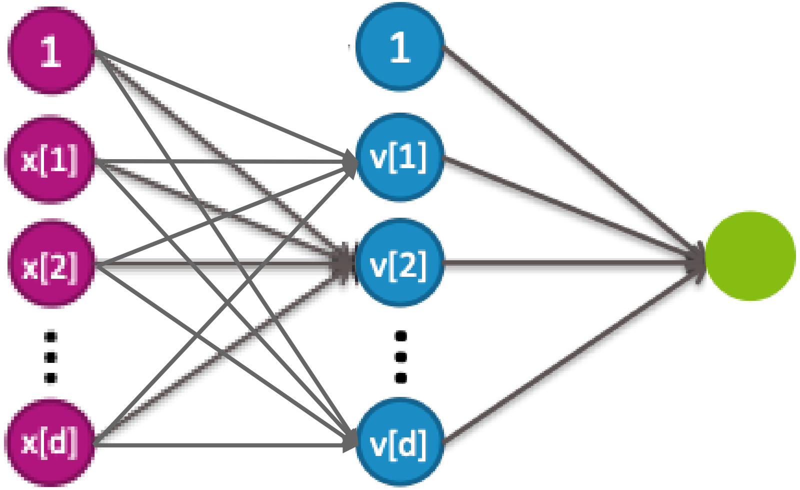
Idea: Combine these perceptrons in layers to learn more complex functions.



$$V[d] = \text{sign}\left(w_0^{(d)} + \sum_{i=1}^{D} w_i^{(d)} x[i]\right)$$

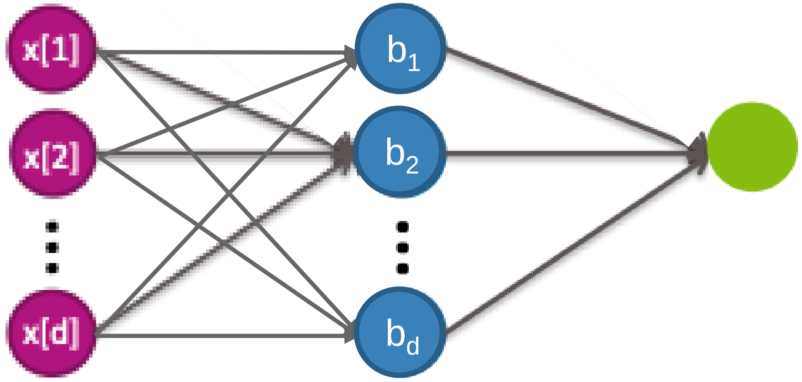
Neural Network Diagram Simplified

- 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

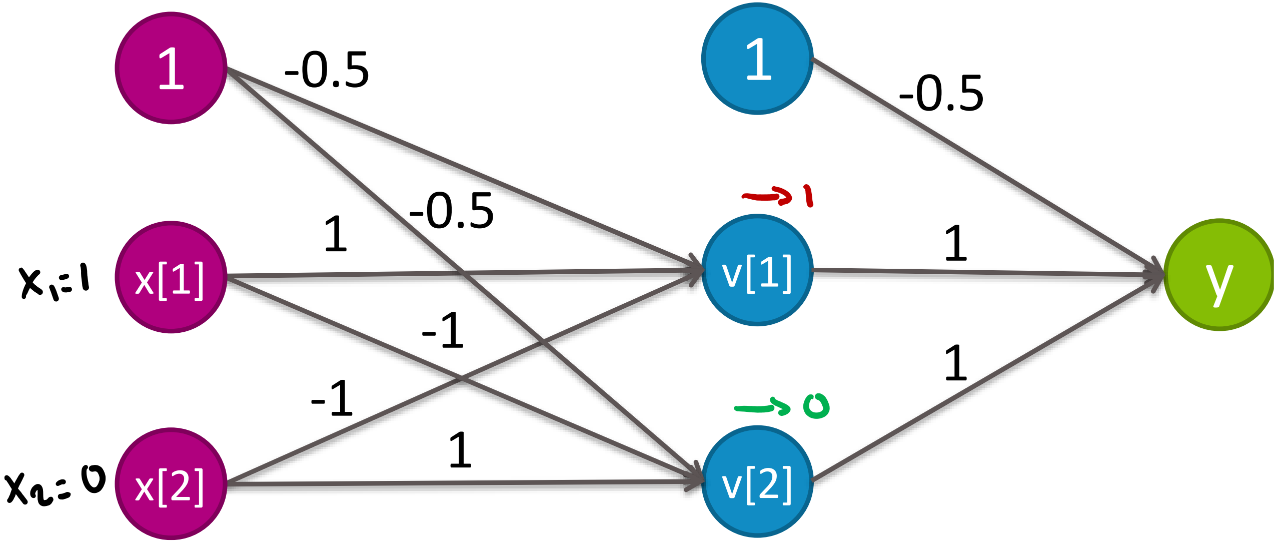
- Oftentimes, the bias is not explicitly shown as another input, and instead written on top of a node.
- You will see both types of diagrams in this course.



XOR

Notice that we can represent

$$x[1]\ XOR\ x[2] = \overbrace{\underbrace{(x[1]\ AND\ !\,x[2])}_{v_1}\ OR\ \underbrace{(!\,x[1]\ AND\ x[2])}_{v_2}}^{v_1\ OR\ v_2}$$



$v_1 = -0.5 + 1\cdot 1 - 1\cdot 0 = 0.5 \xrightarrow{AF} 1$

$v_2 = -0.5 + (-1)\cdot 1 + 1\cdot 0 = -1.5 \xrightarrow{AF} 0$

XOR

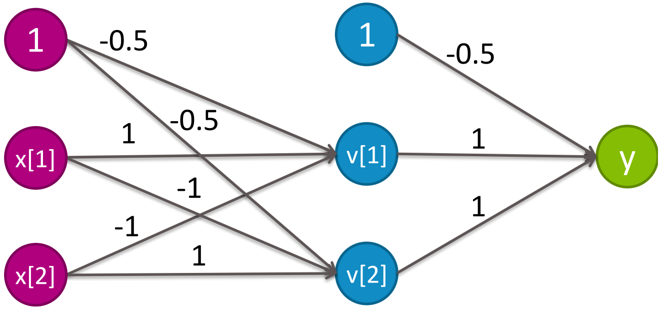
This is a 2-layer neural network

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

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

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

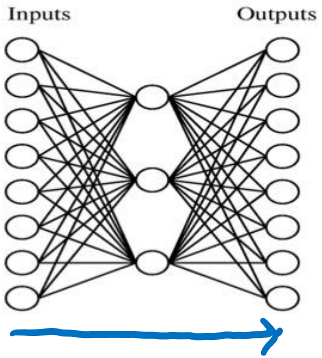
$$y = v[1]\ OR\ v[2]$$
$$= g(-0.5 + v[1] + v[2])$$



Neural Network

Prediction = Forward Pass

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



Single

Activation Function

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

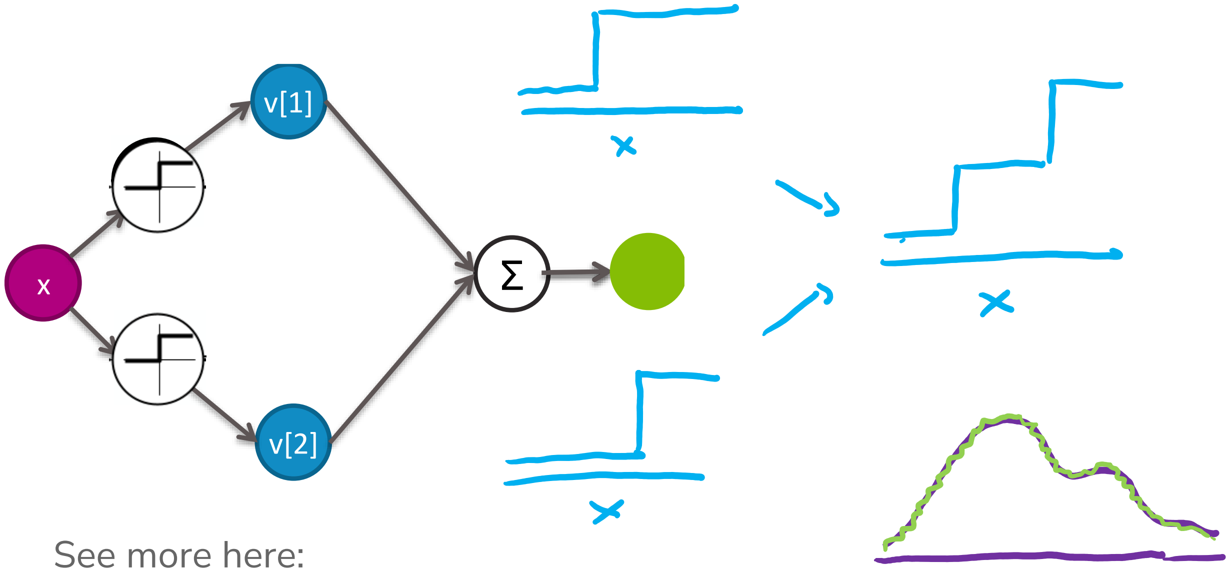
1-hidden layer

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

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

CSE/STAT 416

Neural Networks

Hunter Schafer
Paul G. Allen School of Computer Science & Engineering
University of Washington

May 1 , 2023

❓ **Questions?** Raise hand or **sli.do #cs416**
🎵 **Listening to:** Lady Lamb

Aside: Missing Data

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		???

BAD Idea

No way to deal w/ future missing data

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		

Can work, but requires domain-specific judgment on why data is missing and if removing col is okay

Missing Data: Idea 3

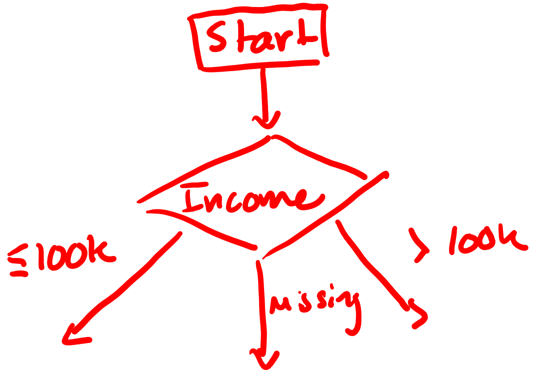
- **Idea 3**: Treat missing values as a separate value of the feature (only Decision Trees)

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		



Works for Decision Trees (in theory, but not supported by sklearn)

Missing Data: Idea 4

(Most Commonly Used!)

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

sklearn ⇒ SimpleImputer

Train Set

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

Test Set

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

- mean?
- median?
- mode?
- constant? (0?)
- Fancy modeling

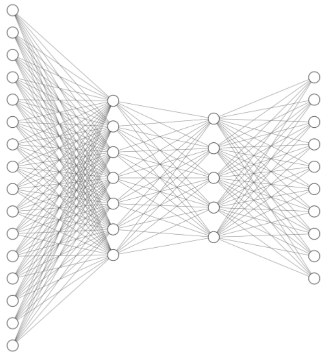
Introduction to Neural Networks

Roadmap So Far

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

History of Neural Networks

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



Have been around for about 50 years

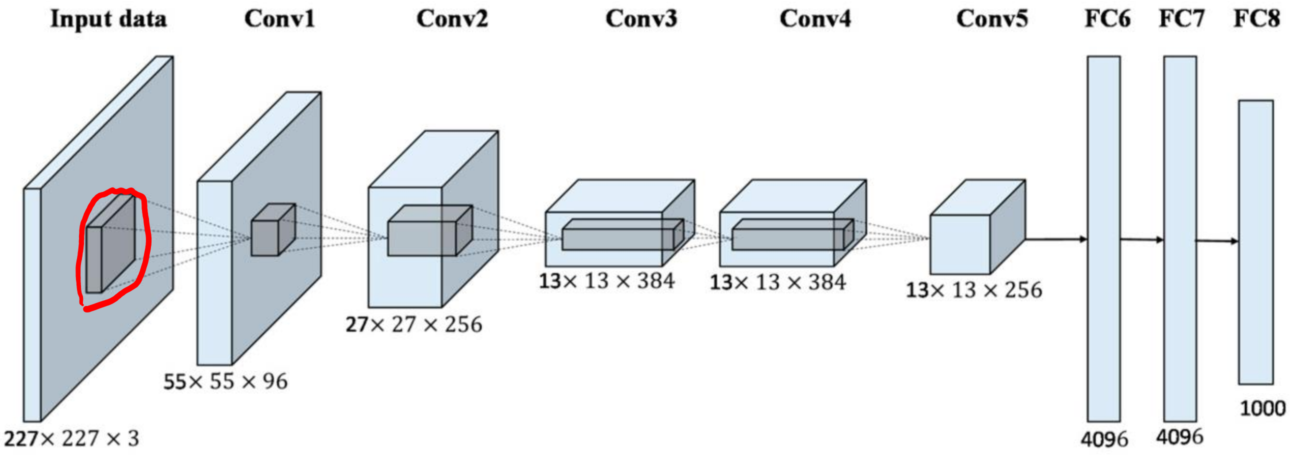
- Fell in “disfavor” in the 90s when simpler models were doing well

In the last decade(s), have had a huge resurgence

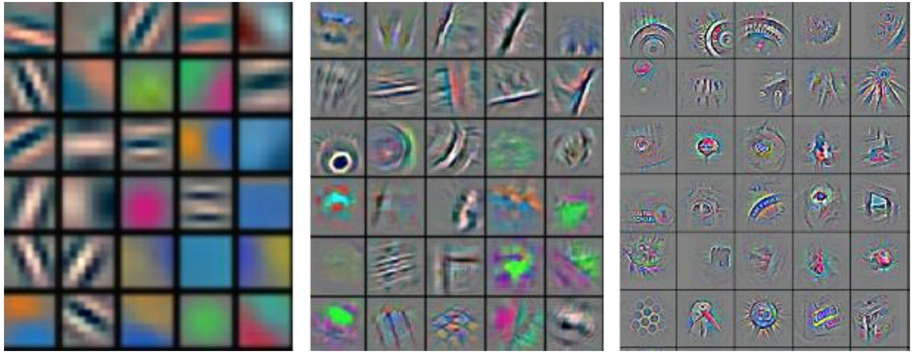
- Impressive accuracy on several benchmark problems
- Have risen in popularity due to huge datasets, GPUs, and improvements to

Popular Neural Network Architectures: CNNs

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

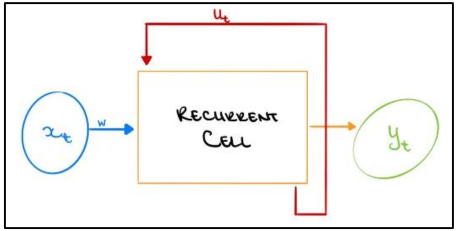


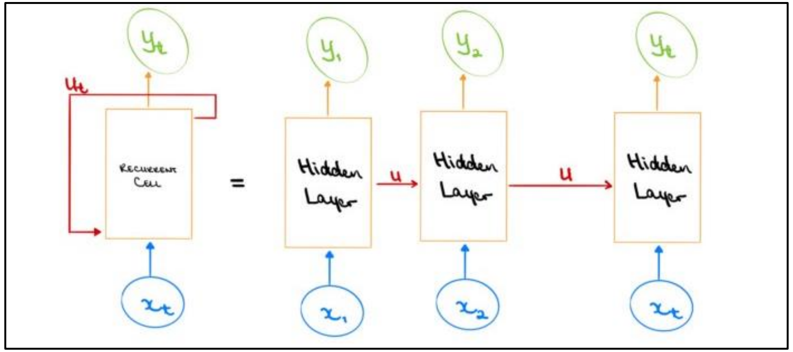
Level 1 Features

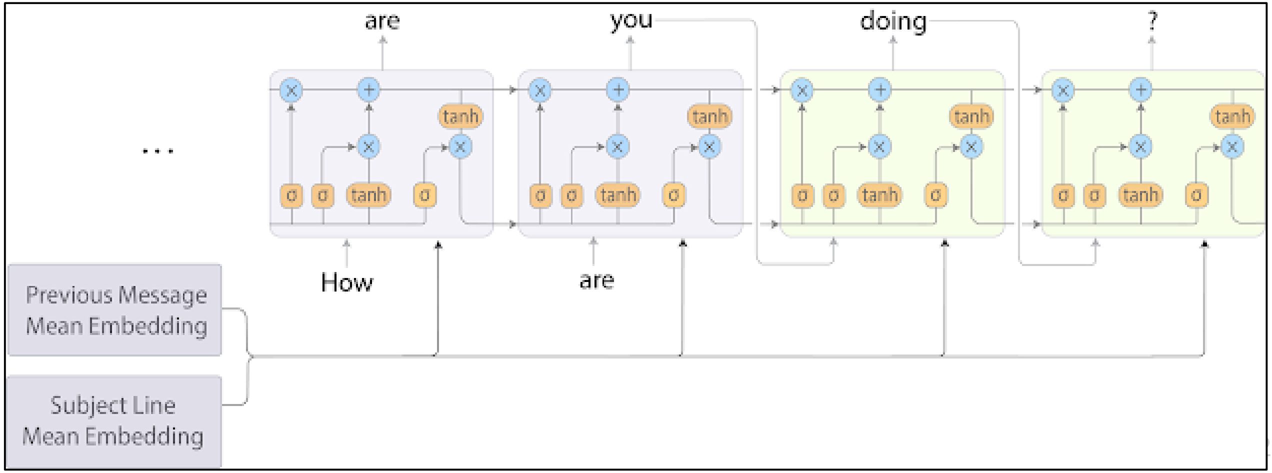


Popular Neural Network Architectures: RNNs

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



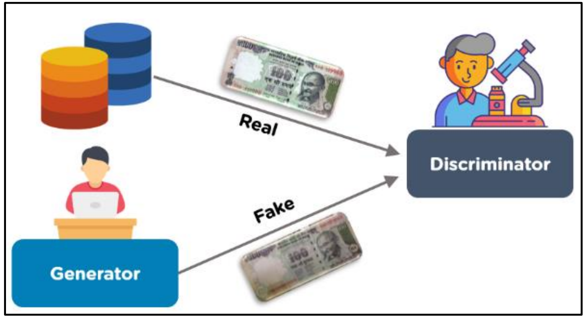


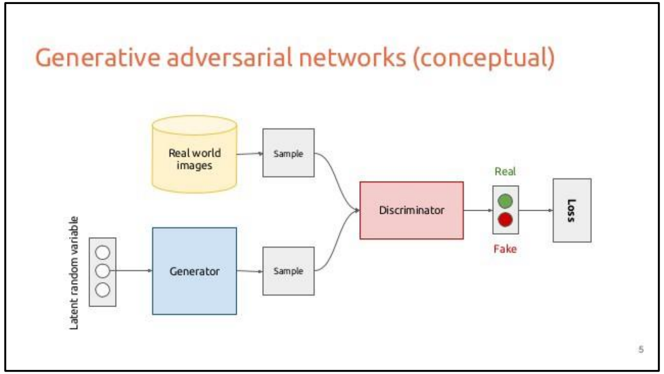


Popular Neural Network Architectures: GANs

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/

Neural Network Details

XOR

$v[1] = g(-0.5 + 1 \cdot 1 + (-1) \cdot 0) = g(0.5)$
$= 1$

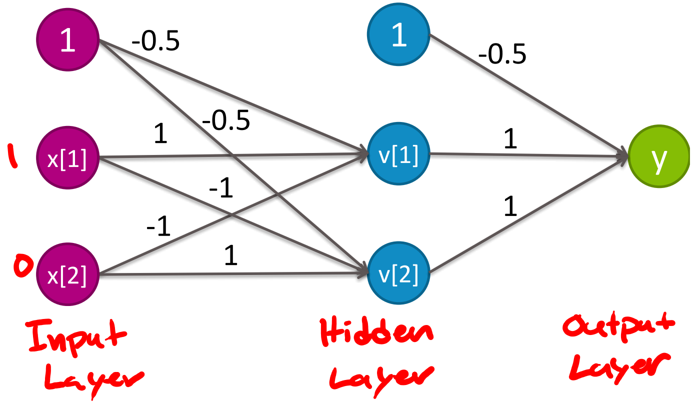
This is a 2-layer neural network

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

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

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

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



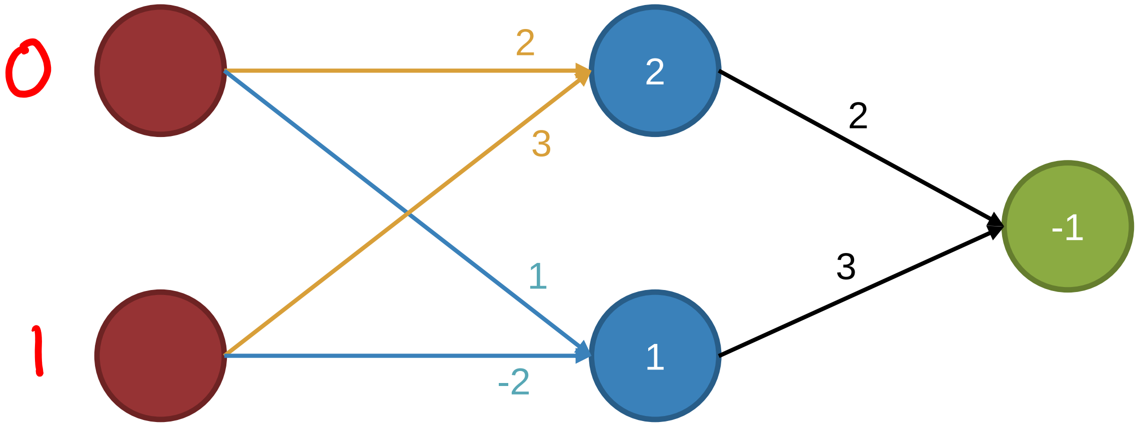
slido

Think

2 mins

sli.do #cs416

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



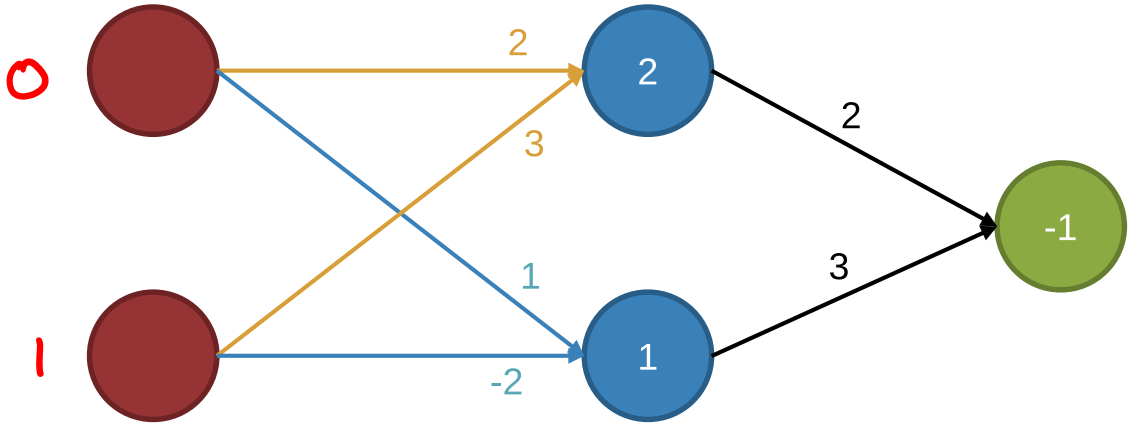
slido

Group

2 mins

sli.do #cs416

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



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

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

$$v_3 = \text{sign}(-1 + 2\cdot 1 + 3\cdot 0) = \text{sign}(1) = \boxed{1}$$

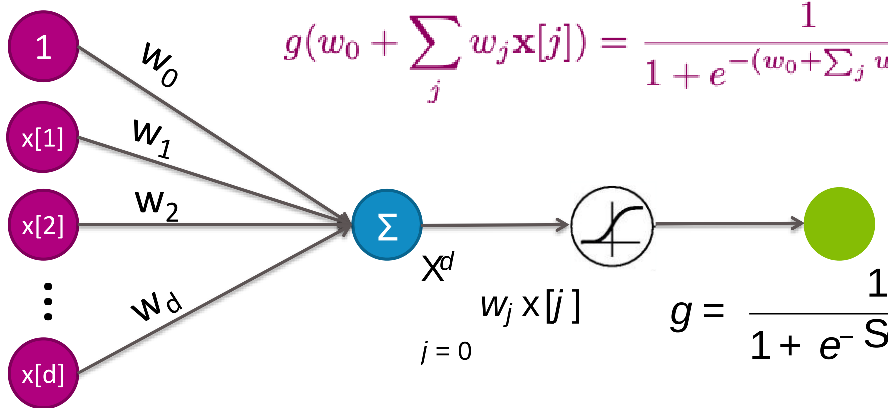
Activation Function

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?

$$g(w_0 + \sum_j w_j \mathbf{x}[j]) = \frac{1}{1 + e^{-(w_0 + \sum_j w_j \mathbf{x}[j])}}$$

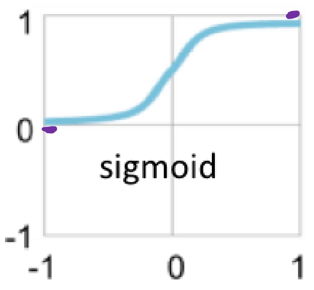


$$\sum_{j=0}^{d} w_j x[j]$$

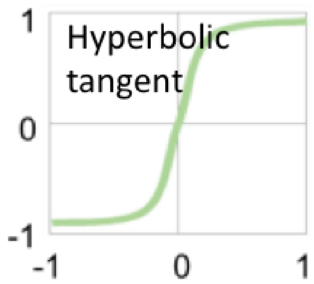
$$g = \frac{1}{1 + e^{-Score(x)}}$$

Activation Functions

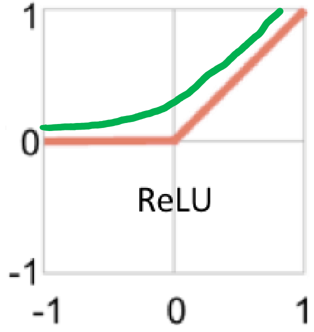
•Sigmoid (Logistic)
-Historically popular, but (mostly) fallen out of favor
•Neuron's activation saturates
(weights get very large -> gradients get small)
•Not zero-centered -> other issues in the gradient steps
-When put on the output layer, called "softmax" because interpreted as class probability (soft assignment)



•Hyperbolic tangent $g(x) = \tanh(x)$
-Saturates like sigmoid unit, but zero-centered



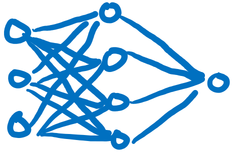
•Rectified linear unit (ReLU) $g(x) = x^+ = \max(0,x)$
-Most popular choice these days
-Fragile during training and neurons can "die off"… be careful about learning rates
-"Noisy" or "leaky" variants



•Softplus $g(x) = \log(1+\exp(x))$
-Smooth approximation to rectifier activation

Classification or Regression

You can use neural networks for classification and regression!



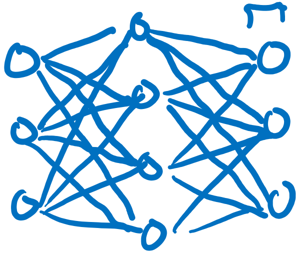
Regression

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!

number of classes



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

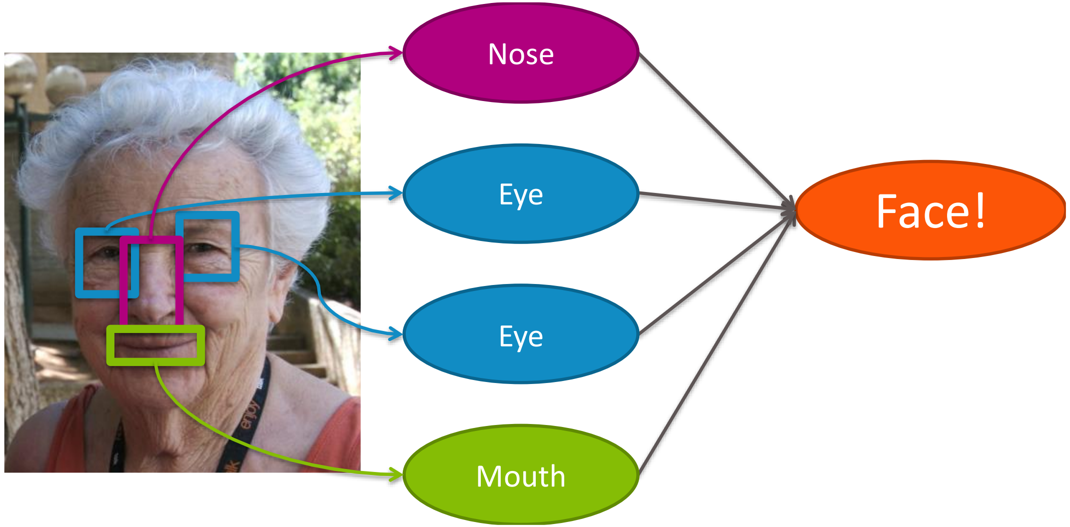
Brain Break

Application to Computer Vision

~~Image Features~~

Features in computer vision are local detectors

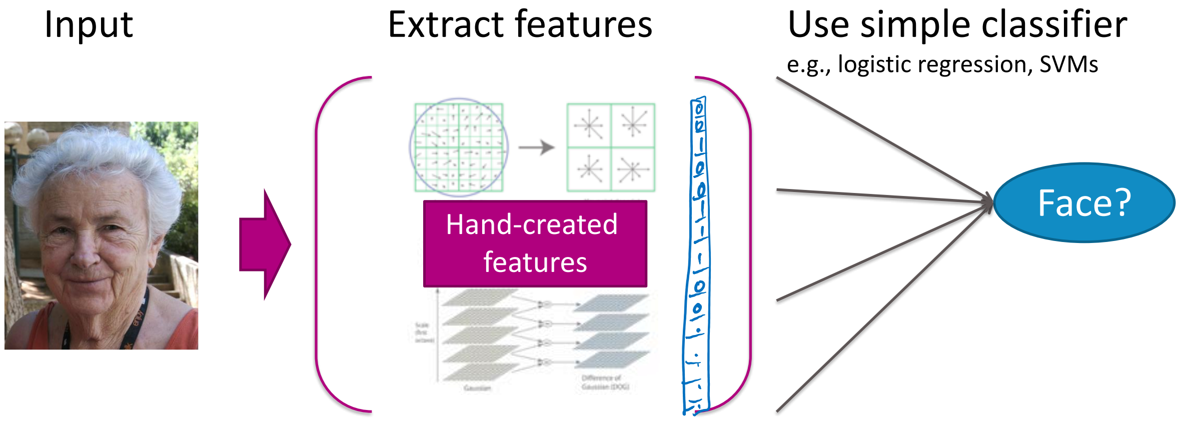
- Combine features to make prediction



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

~~The Past~~

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



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

~~NNs to the Rescue~~

Neural Networks implicitly find these low level features for us!

→ Layer 1 → Layer 2 → Layer 3 → Prediction

Example detectors learned			
Example interest points detected			

[Zeiler & Fergus '13]

Each layer learns more and more complex features

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!

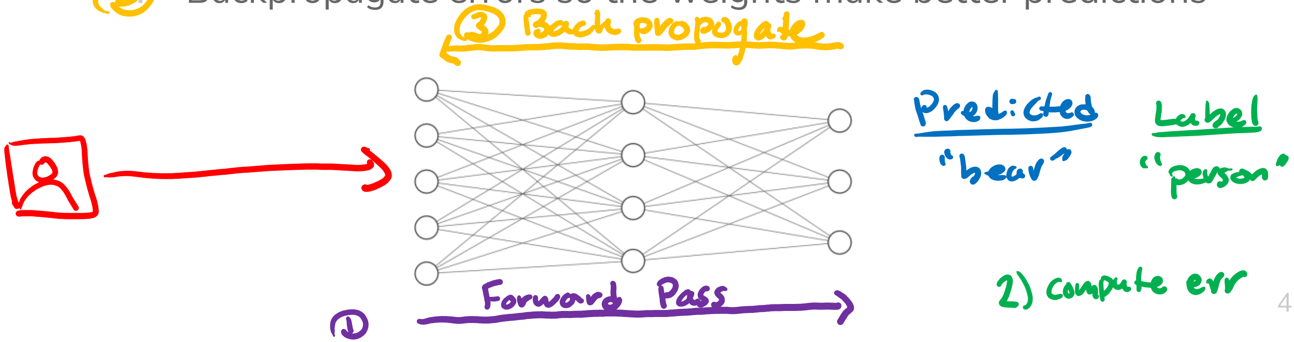
- How do we take the derivative of a network?
- Are there convergence guarantees?

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 training set into batches

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**.

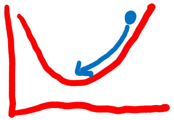
epochs: How many times we iterate over train set

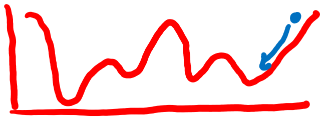
```
for i in range(num_epochs):
  for batch in batches(training_data):
    preds = model.predict(batch.data) # Forward pass
    diffs = compare(preds, batch.labels)  # Compare
    model.backprop(diffs)                 # Backpropagation
```

- epoch = one step of training NN on whole train set

NN Convergence

convex





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.

slido

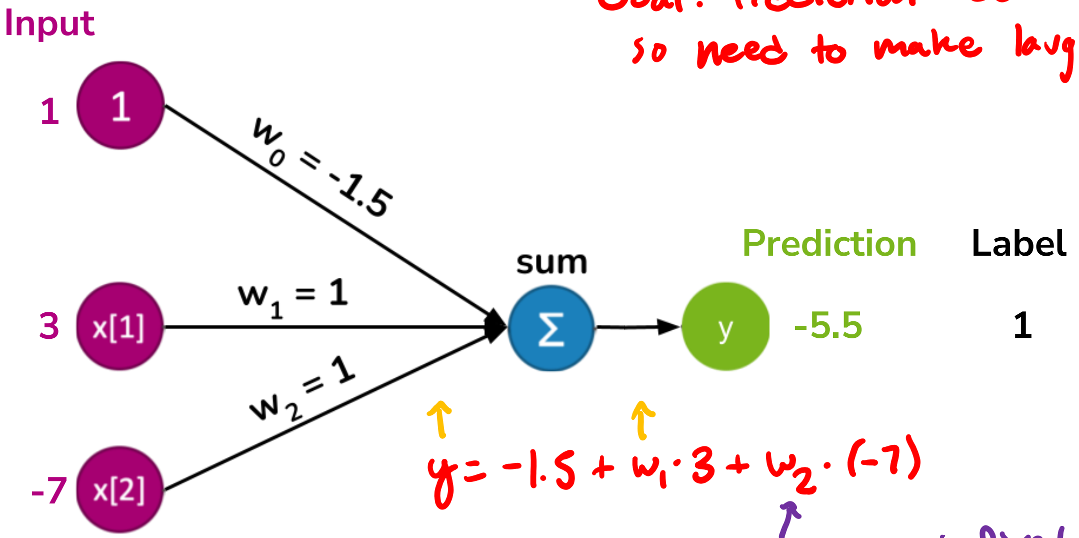
Group

1 mins

sli.do #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?

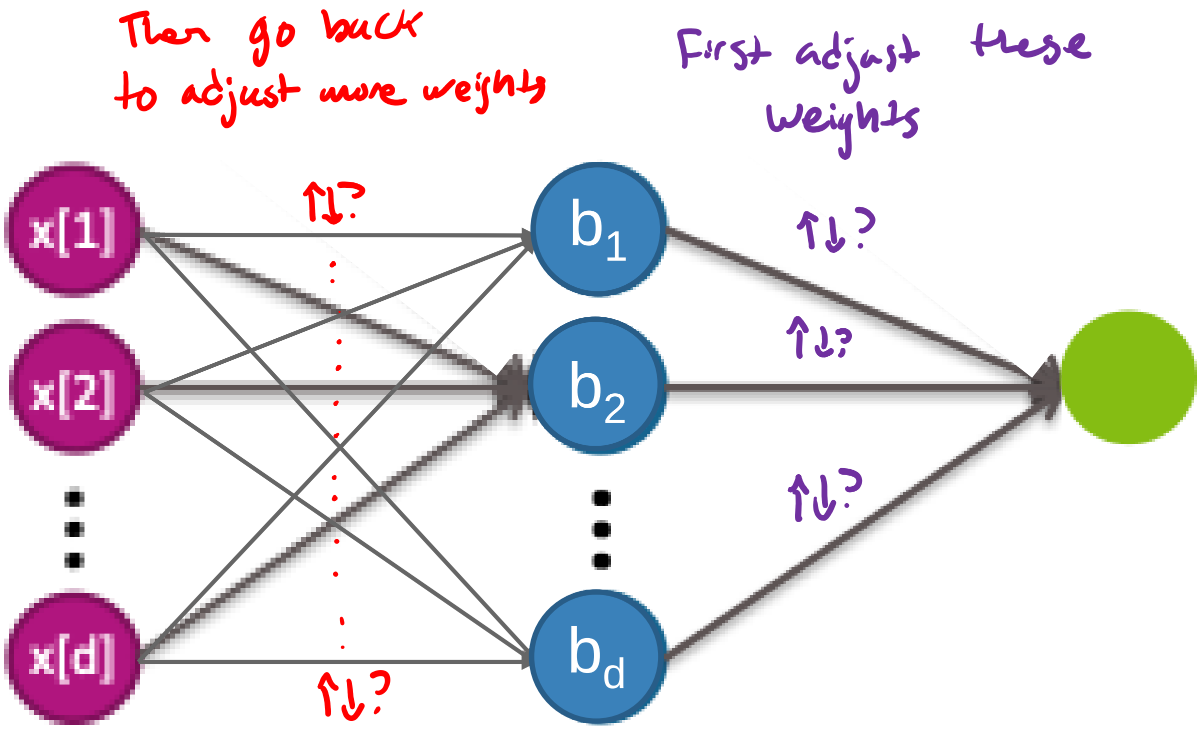
Goal: Prediction too small, so need to make larger



$y = -1.5 + w_1 \cdot 3 + w_2 \cdot (-7)$

assume w_2 is fixed

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!

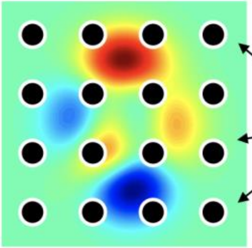
How do you decide these values should be? ¯_(ツ)_/¯

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:

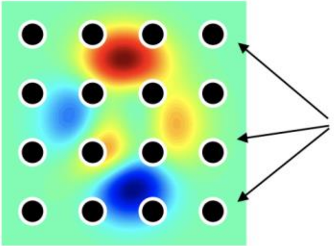


Hyperparameters on 2d uniform grid

Hyperparameter Optimization

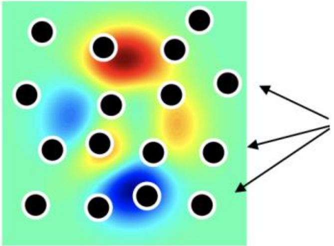
How do we choose hyperparameters to train and evaluate?

Grid search:



Hyperparameters on 2d uniform grid

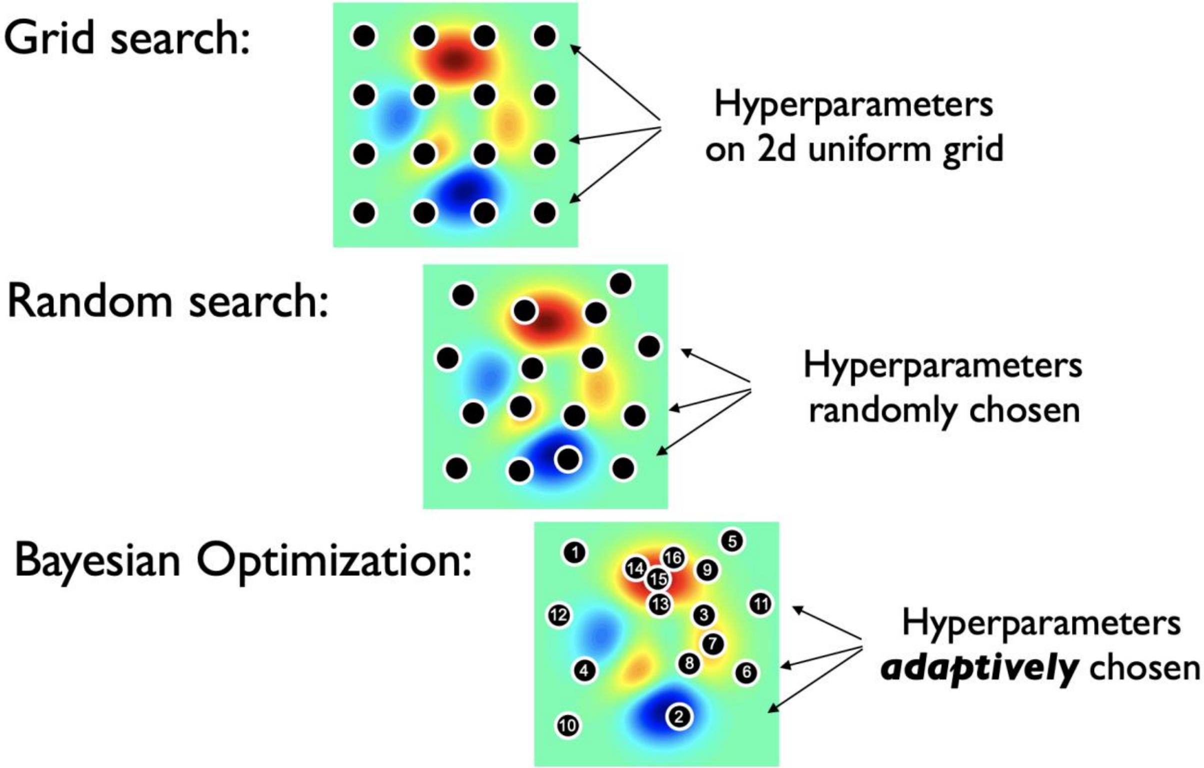
Random search:



Hyperparameters randomly chosen

Hyperparameter Optimization

How do we choose hyperparameters to train and evaluate?



Hyperparameter Optimization

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

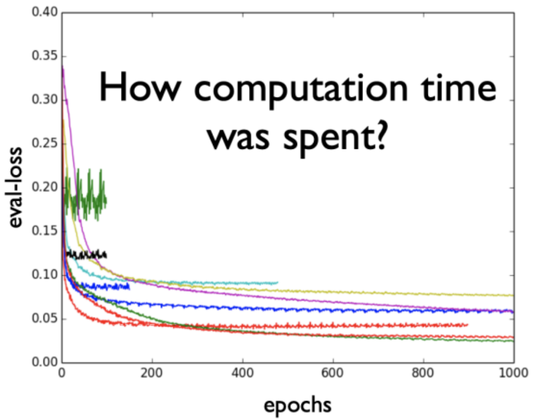
Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. arXiv:1406.3896, 2014.

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
- Activation functions
- Neural Networks and Overfitting
- Backpropagation idea
- NN Hyperparameters
- Hyperparameter optimization
- NN Convergence guarantees