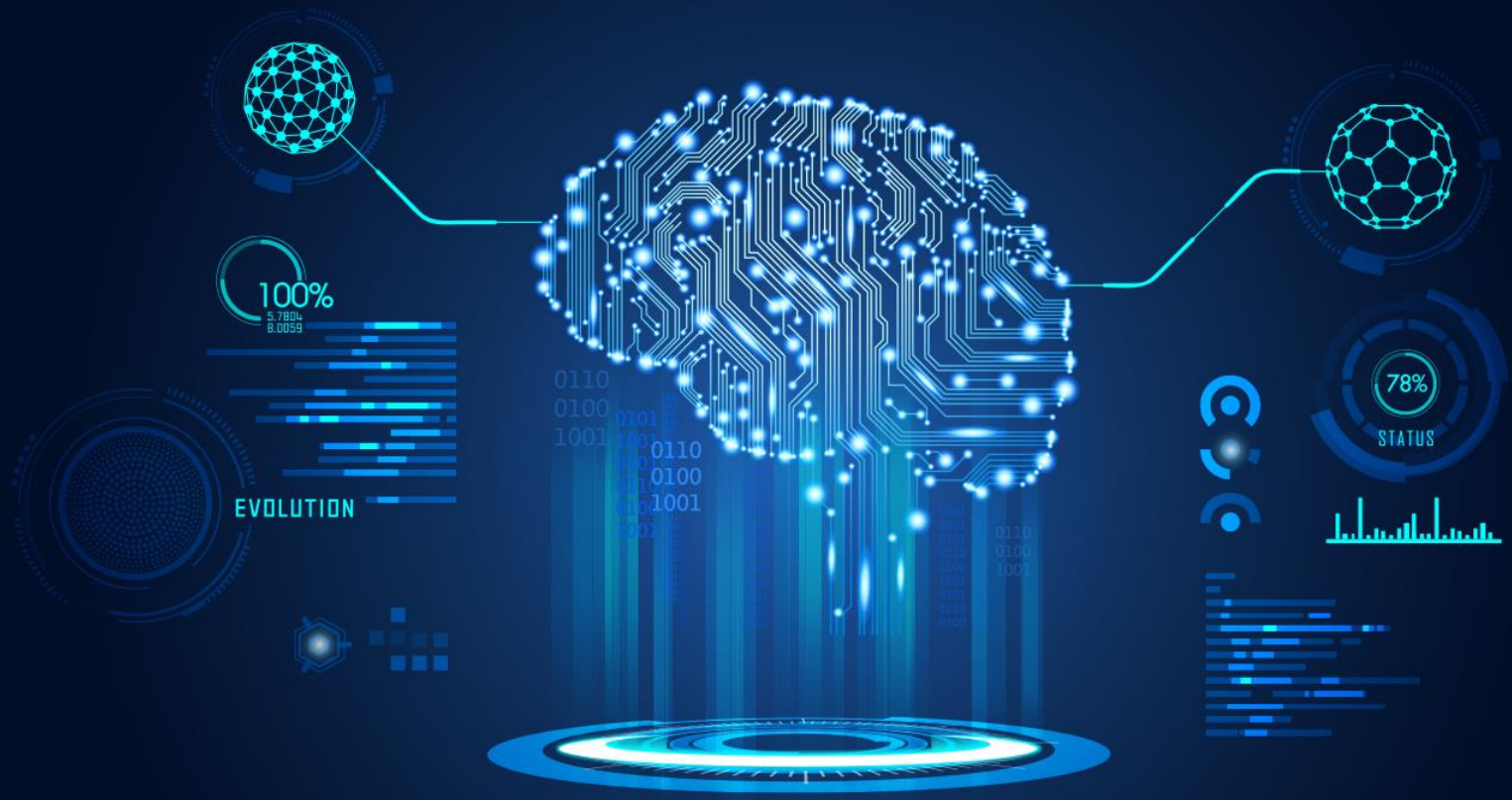
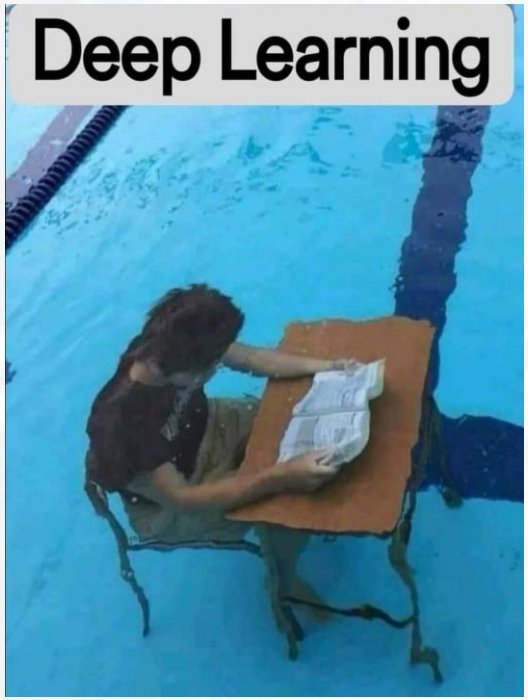


DEEP LEARNING

An Intuitive Introduction

OUTLINE



Introduction

Neural Networks

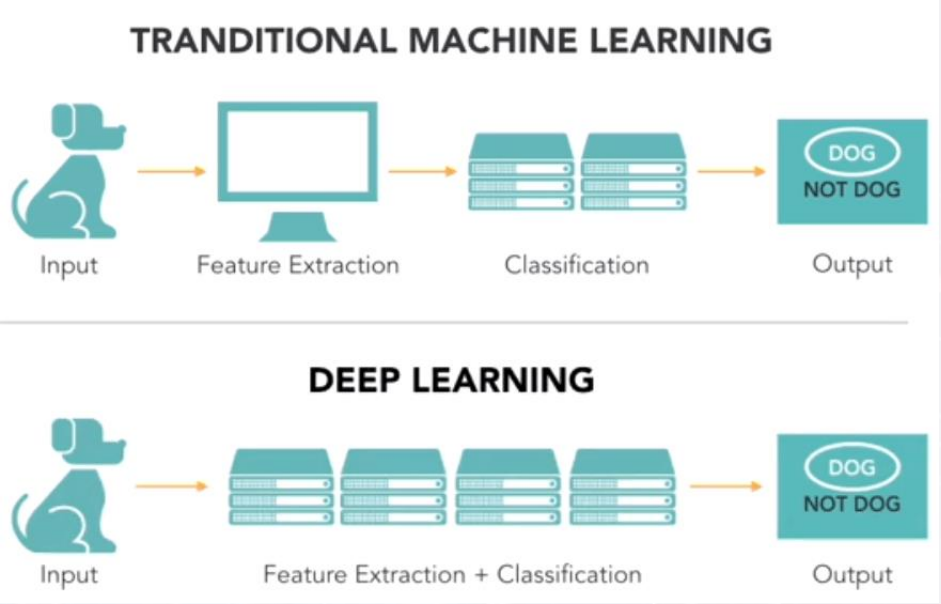
- Perceptron's
- Activation Function
- Loss Function
- Back propagation
- Gradient Descent
- Learning Rate

Tips in Deep Learning

Application Areas

Common NN Architectures

INTRODUCTION



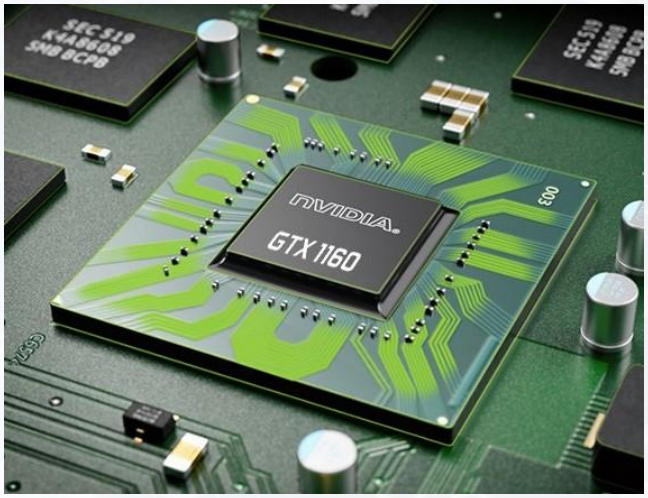
Machine Learning Vs Deep Learning

- ML: Hand crafted features
  - Time consuming
  - Brittle
  - Not scalable
- Deep Learning:
  - learn features from data itself

INTRODUCTION

Deep Learning: Why now?

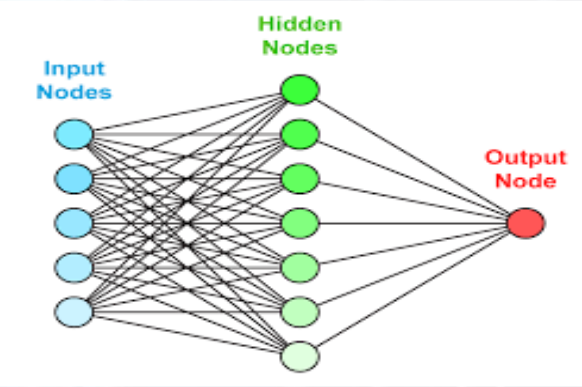
- Big Data
- Hardware:
  - These models are highly parallelable and compatible on GPU's
- Software



INTRODUCTION

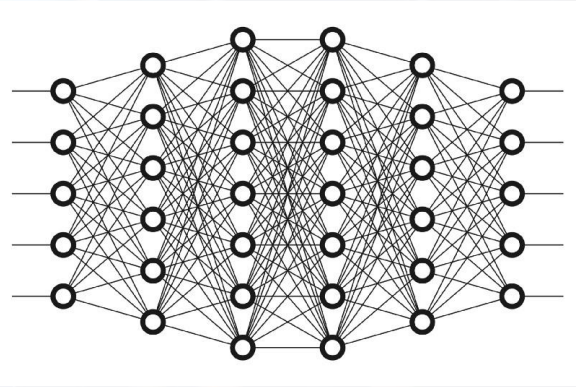
ANN:

- a supervised learning system built of perceptrons.
- can emulate almost any function, and answer practically any question, given enough training samples and computing power.
- A "shallow" neural network has only three layers of neurons: *an input layer, one hidden layer and an output layer.*



DNN:

- has a similar structure, but it has two or more "hidden layers" of neurons that process inputs.
- Additional layers are useful up to some limit, after which their predictive power starts to decline.



INTRODUCTION

Hidden Layer:

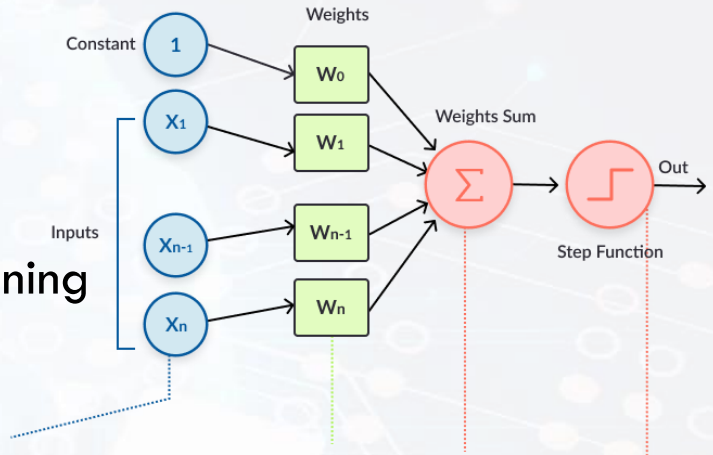
- Not observable and it's a learnt layer
- We can't explicitly enforce any behavior at this layer

Fully Connected Network:

- Dense networks
- Each input layer is connected to each node in the output layer

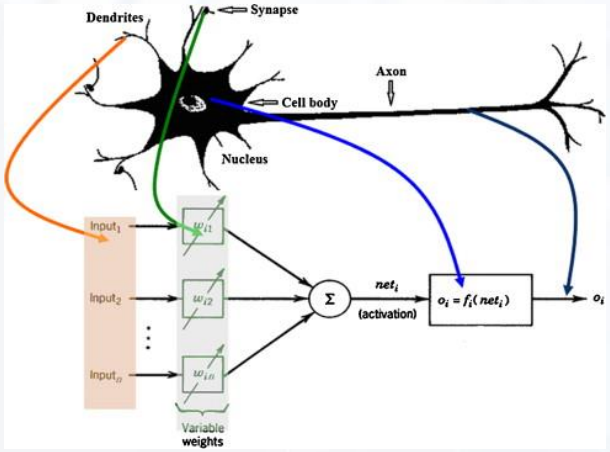
Neural Net architecture may have a single or multiple output perceptron

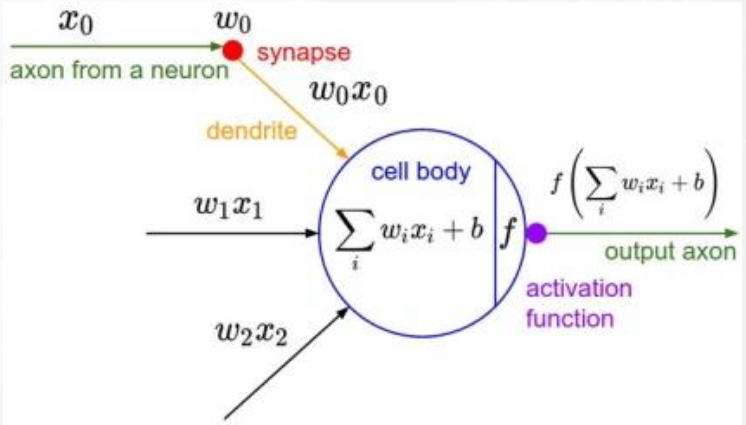
COMPONENTS: PERCEPTRON



Perceptron/neuron: Building blocks of deep learning

- takes the inputs from the input layer,
- multiplies them by their weights, and computes the sum.
- adds the number one, multiplied by a "bias weight"
- feeds the sum through the activation function
- pass it on to the next layer of neurons in the network

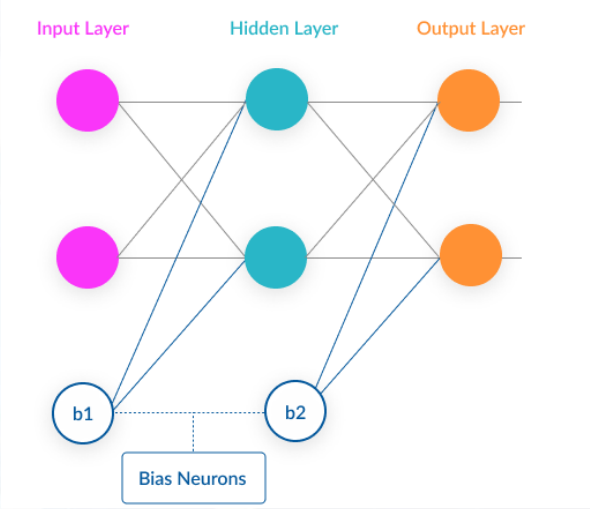




COMPONENTS: BIAS NEURON

Bias Neuron:

- special neuron added to each layer in the neural network
- shifts activation function left or right on the graph



What will happen to the network with no bias node?

- Ans: Consider the function Y = 2x + b, with out the bias value b, you can't get a value 2 given an input value 0. b shifts the graph 2x right or left.
- So b can generalize the function to represent complex inputs.

ACTIVATION FUNCTION

Decides whether a neuron should be activated or not.

Introduce non-linearity into the output of a neuron.

Allows approximation of complex functions

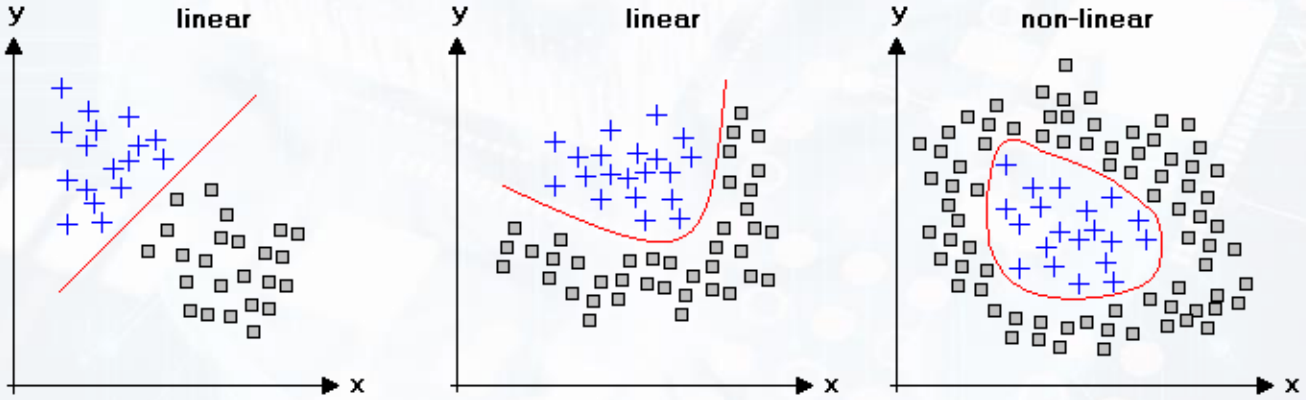


Fig X: approximation of data using linear vs non linear activation functions

Linearity:

- Linear models are linear in the parameters which have to be estimated
- It isn't about having a straight line on the boundary between the classes

ACTIVATION FUNCTION

Neural Networks use non-linear activations as linear activation function has two major problems:

- Not possible to use backpropagation (gradient descent) to train the model
  - the derivative of the function is a constant, and has no relation to the input, X.
  - So it's not possible to go back and understand which weights in the input neurons can provide a better prediction.

- All layers of the neural network collapse into one
  - with linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function).
  - So a linear activation function turns the neural network into just one layer.

A neural network with a linear activation function is simply a linear regression model. It has limited power and ability to handle complexity varying parameters of input data.

ACTIVATION FUNCTION

$$\sigma(z)j = \frac{ezj}{\sum_{k=1}^{k} ezk} \quad for\ j = 1....k.$$

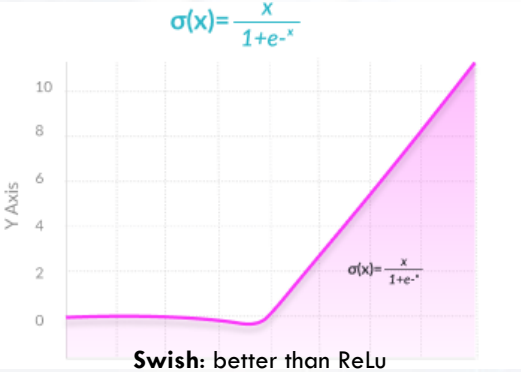
Softmax:

Able to handle multiple classes

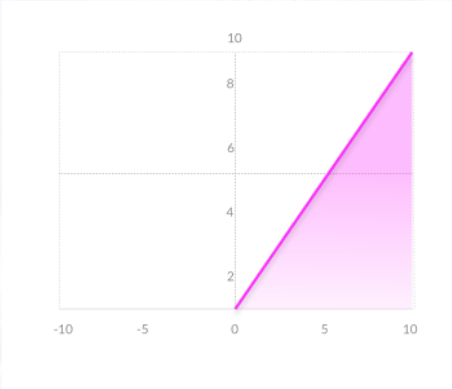
- normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.

Useful for output neurons

- is used only for the output layer, for neural networks that need to classify inputs into multiple categories.



Swish: better than ReLu

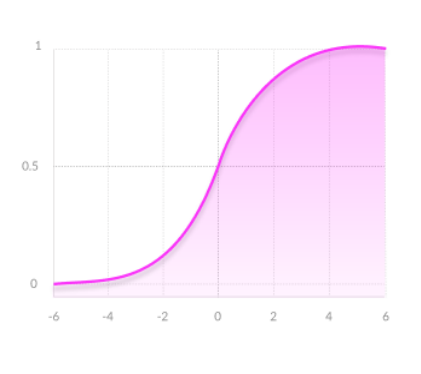


Relu:

Computationally efficient

The Dying ReLU problem

- when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.



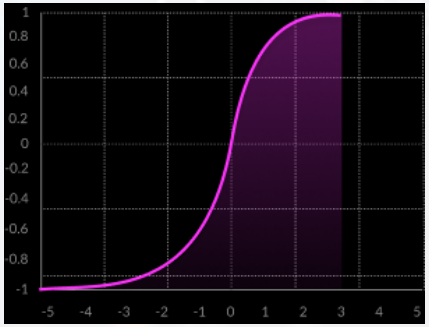
Sigmoid:

- **Smooth gradient,** preventing "jumps" in output values.
- **Output values bound** between 0 and 1, normalizing the output of each neuron.
- **Clear predictions:** close to 1 or 0.

Vanishing gradient

- for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem.
- This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.

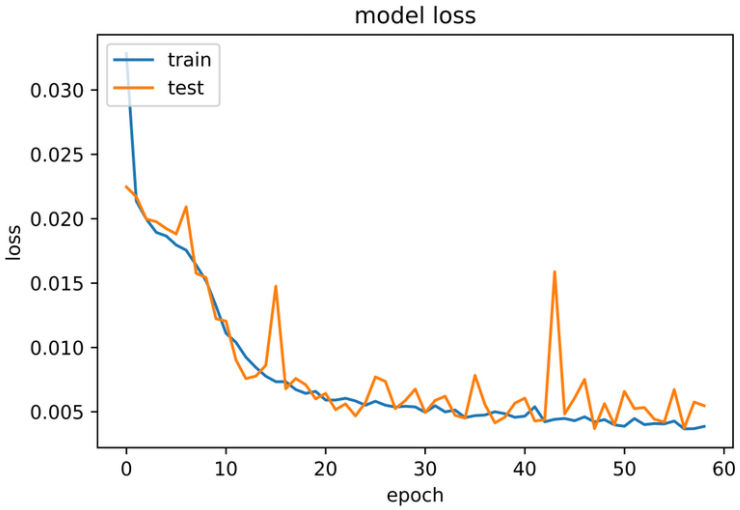
Computationally expensive



Tanh

and others . . .

LOSS FUNCTION



Loss:

- Used to calculate mistakes that the network made while learning.
- Something to be minimized through iterative update of weights with backpropagation

- Cross Entropy Loss
  - Used when the model output is a probability between 0 and 1
  - When determining class labels
  - It could be binary cross entropy or categorical cross entropy depending on the number of classes

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * log(\hat{y}_i) + (1-y) * log(1-\hat{y}_i))$$

- Mean Squared Error Loss
  - Used with regression models that output continuous real numbers

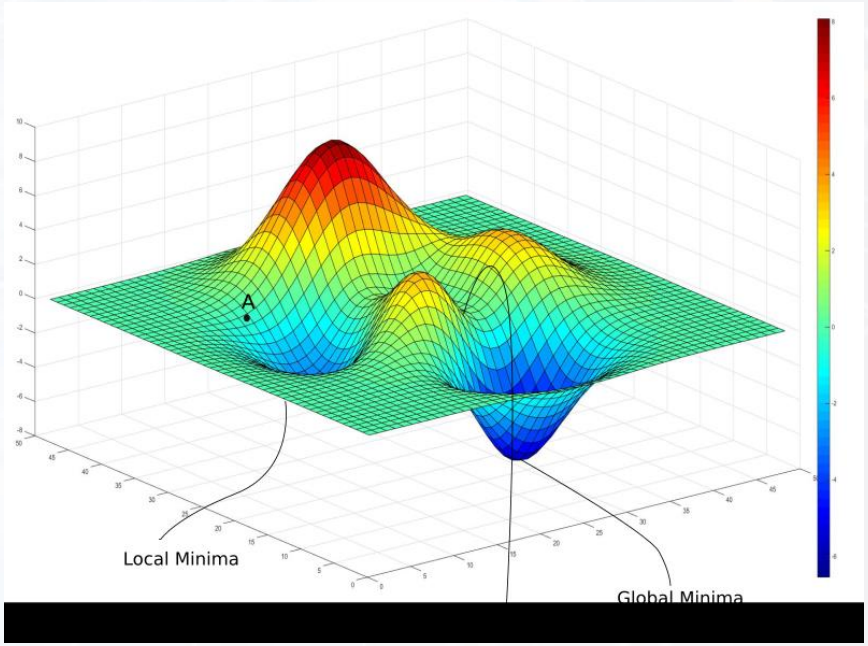
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

TRAINING: BACKPROPAGATION

Objective: finding weights that minimize the loss function

Training:

- Initialize weights randomly
- Loop:
  - Compute gradient: partial derivative [batching]
  - Update weights [propagate error back to weights]
- Return Weights



TRAINING: GRADIENT DESCENT

Stochastic Gradient Descent

- calculates the error and updates the model for each example in the training dataset.
- Updating the model so frequently is more computationally expensive

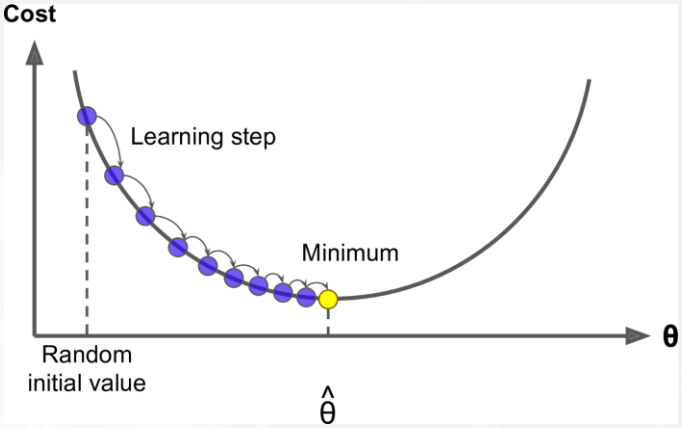
Mini-Batch Gradient Descent

- splits the training dataset into small batches that are used to calculate model error and update model coefficients.
- Implementations may choose to sum the gradient over the mini-batch which further reduces the variance of the gradient.
- Error information must be accumulated across mini-batches of training examples like batch gradient descent.

Batch Gradient Descent

- calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated.
- more computationally efficient than stochastic gradient descent.
- The decreased update frequency results in a more stable error gradient and may result in a more stable convergence on some problems.
- algorithm to parallel processing based implementations.
- The more stable error gradient may result in premature convergence of the model to a less optimal set of parameters.

LEARNING RATE



The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated.

Problem:

- Choosing the learning rate is challenging
  - as a value too small may result in a long training process that could get stuck,
  - a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.

Solution:

- Try lot of learning rates and use the best
- Design adaptive algorithm that adapts gradient decent landscape
  - Learning rates are not fixed, can be small or large depending on:
    - How large the gradient is at that location
    - How fast learning is happening
    - Size of particular weight

Adaptive LR algorithms:

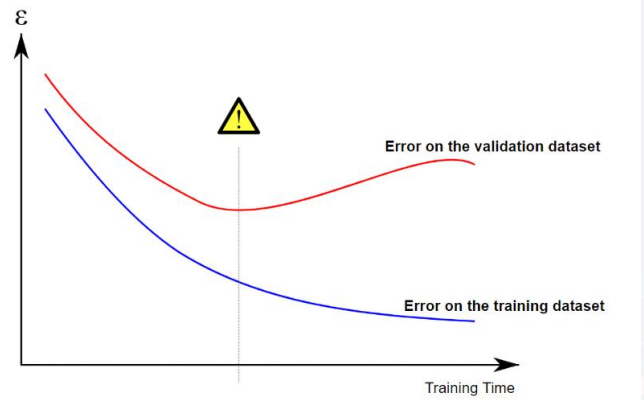
- Momentum
- Adagred
- Adadelta
- Adam
- RMSprop

EXAMPLE

Simple ANN: Regression Problem

- *backpropagation [Mathematical rules]*
- *python code:*

TIPS IN DL: REGULARIZATION

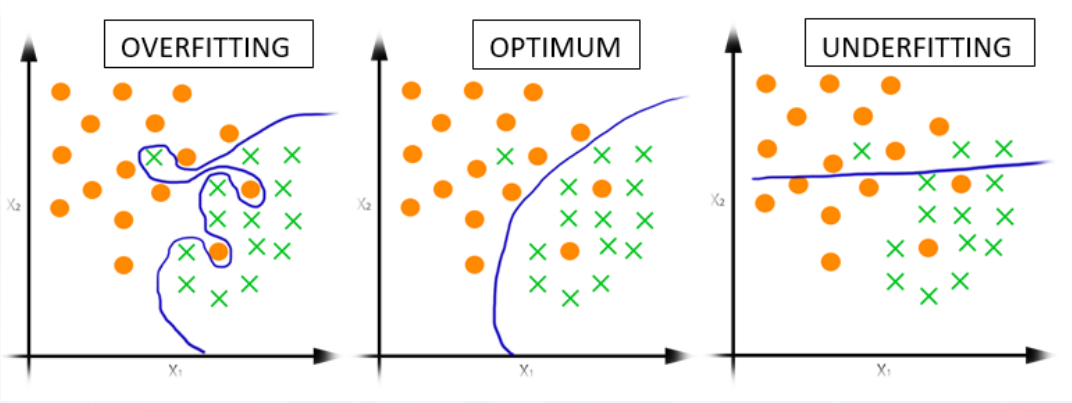


Overfitting:

- model performs very good on training data, but fails when it sees new data from the same problem domain.
- use fewer layers or neurons in the neural network. Or use regularization techniques

Underfitting:

- model fails on both types of data: training data as well as new data
- increase the complexity by adding more layers or neurons



TIPS IN DL: REGULARIZATION

Add droput and early stopping images

Dropout:

- Randomly set some of the activations to zero
- On every iteration the model doesn't know which node is going to be dropped, so it can't rely on a specific path
- Generates an ensemble of different models that better generalize on unseen data

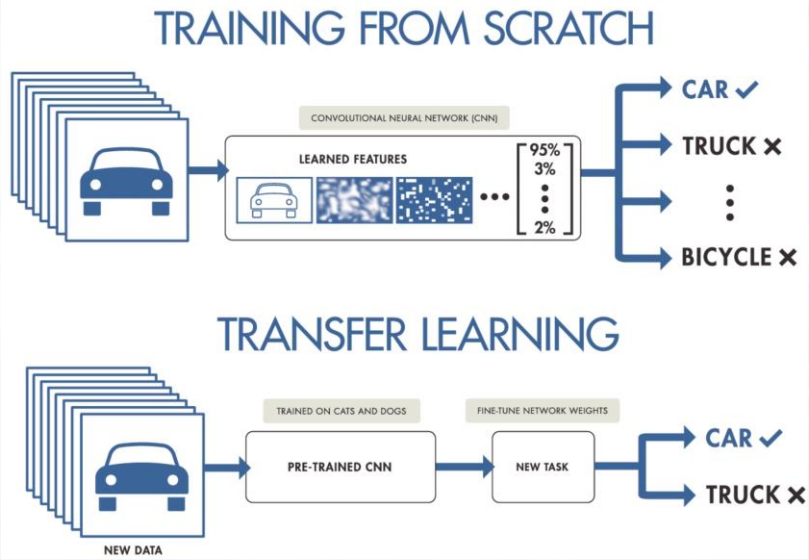
Early stopping:

- Stop learning the model when it starts overfitting visualizing the learning behavior at each iteration .

TIPS IN DL: TRANSFER LEARNING

Transfer Learning:

- model that was created and trained for one task, is reused as the starting point for a secondary task.
- used with Less training data



TIPS IN DL: PARAMETER VS HYPERPAR'S

Model parameters are internal to the neural network

- neuron weights.
- They are estimated or learned automatically from training samples.
- These parameters are also used to make predictions in a production model.

Hyperparameters are external parameters set by the operator of the neural network

- activation function, batch size, dropout, weights initialization, learning rate, epoch, and others
- have a huge impact on the accuracy of a neural network
- can be set manually or with some learning algorithms

APPLICATION AREAS:

- Self-driving cars
- Deep Learning in Healthcare
- Voice Search & Voice-Activated Assistants
- Automatically Adding Sounds To Silent Movies
- Automatic Machine Translation
- Automatic Text Generation
- Automatic Handwriting Generation
- Image Recognition
- Automatic Image Caption Generation

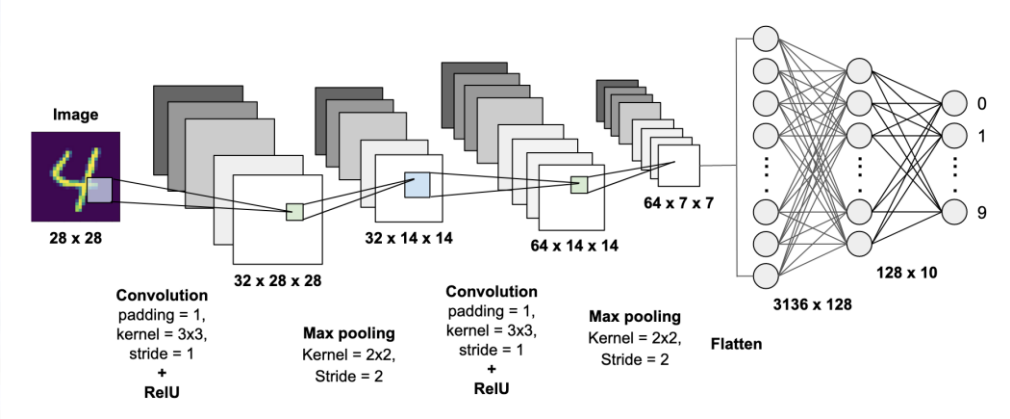
- News Aggregation and Fraud News Detection
- Natural Language Processing
- Virtual Assistants
- Entertainment
- Visual Recognition
- Healthcare
- Detecting Developmental Delay in Children
- Colourisation of Black and White images
- Adding sounds to silent movies
- Automatic Game Playing
- Pixel Restoration
- Demographic and Election Predictions
- Deep Dreaming

COMMON NN ARCHITECTURES: CNN

Convolutional Neural Networks

Applications:

- Image processing



Interesting Applications:

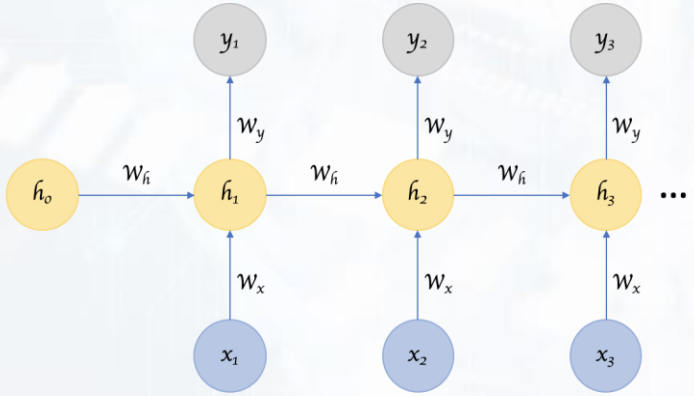
- Image Classification - Search Engines, Recommender Systems, Social Media
- Face Recognition Applications
- Legal, Banking, Insurance, Document digitization - Optical Character Recognition
- Drug Discovery

COMMON NN ARCHITECTURES: RNN

Sequential Models

Applications:

- Sequential data processing [Text, Speech, Video]



Input		Target		Use Cases
Type	Elements	Type	Elements	
Scalar	One	Trends	Many	Pattern generation
		Audio	Many	Music Generation
		Text	Many	Text Generation
		Image	Many	Image generation
Trends	Many	Scalar	One	Stock Trading decisions
				Forecasting KPI for fixed duration
		Trends	Many	DNA Sequence analysis
				Time series forecasts
Text	Many	Scalar	One	Sentiment Classification
				Topic Classification
				Answer Selection
		Text	Many	Text Summarization
				Machine translation
				Chatbots
				Name Entity Recognition
				Subject Extraction
				Part of Speech Tagging
				Textual Entailment
				Relation Classification
		Trends	Many	Path Query Answering
		Audio	Many	Speech Generation
Image	Many	Scalar	One	Facial expression tagging
				Entity classification
		Text	Many	Image Captioning
		Image	Many	Image Modification
Audio	Many	Scalar	One	Sentiment Classification
				Number of speaker tagging
				Topic Classification
		Text	Many	Speech Recognition
				Conference Summarization
		Audio	Many	Speech Assistant
Video	Many	Scalar	One	Activity Recognition
		Text	Many	Subtitle generation

COMMON NN ARCHITECTURES: GAN

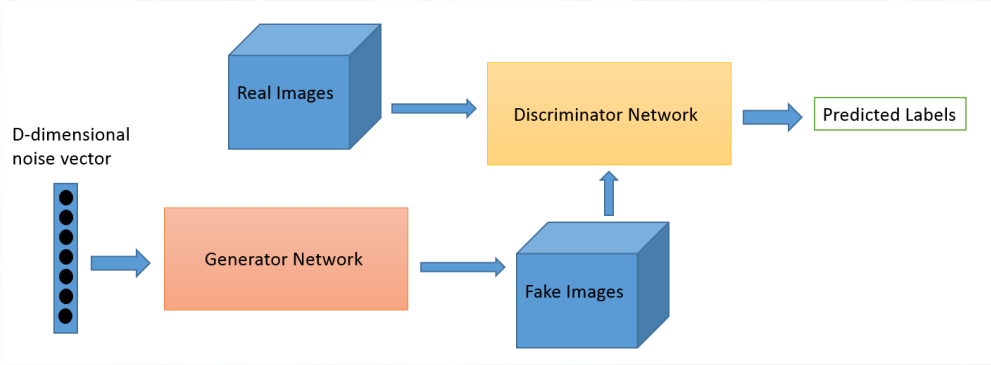
Generative Adversarial Networks

Applications:

- Mostly on images

Interesting Applications:

- Generate Examples for Image Datasets
- Generate Photographs of Human Faces
- Generate Realistic Photographs
- Generate Cartoon Characters
- Image-to-Image Translation
- Text-to-Image Translation
- Semantic-Image-to-Photo Translation
- Face Frontal View Generation
- Generate New Human Poses
- Face Aging
- Video Prediction
- 3D Object Generation

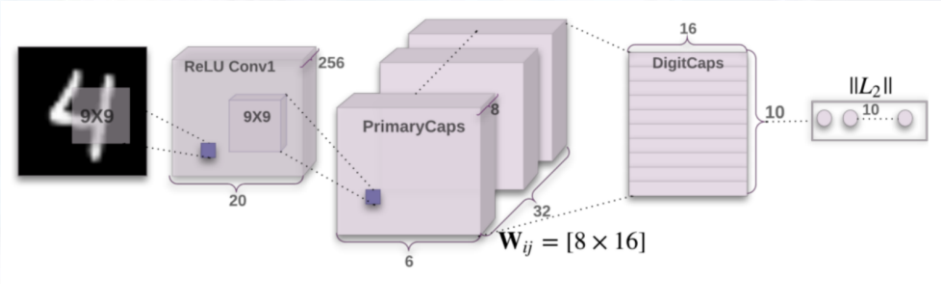


COMMON NN ARCHITECTURES: CAPSNET

CapsNet:

Applications:

- Images



CapsNet Vs CNN

- Viewpoint invariance:
  - the use of pose matrices allows capsule networks to recognize objects regardless of the perspective from which they are viewed.
- Fewer parameters:
  - Because capsules group neurons, the connections between layers require fewer parameters.
- Better generalization to new viewpoints:
- Defense against white-box adversarial attacks:
  - the Fast Gradient Sign Method (FGSM) is a typical method for attacking CNNs.
  - It evaluates the gradient of each pixel against the loss of the network, and changes each pixel by at most epsilon (the error term) to maximize the loss.
  - Although this method can drop the accuracy of CNNs dramatically (e.g.: to below 20%), capsule networks maintain accuracy above 70%.

REFERENCES

1. MIT Deep Learning summer school material
2. Andrew Ng Deep Learning Specialization Materials
3. Deep Learning by Geoffrey Hinton
4. Deep Learning A-Z from Udemy
5. https://missinglink.ai/
6. https://towardsdatascience.com/
7. https://www.newworldai.com/
8. https://www.deeplearning.ai/
9. https://machinelearningmastery.com/
10. https://www.analyticsvidhya.com/
11. https://en.wikipedia.org/
12. https://www.deeplearning-academy.com/

and others.

Thank you!

DATA SCIENCE RESEARCH GROUP

Proposal

DATA SCIENCE RESEARCH GROUP

- Why Data Science?
- Objectives?
- Focus areas?
- What are the benefits?
- Things not to expect?
- Membership requirements?

Comments and/or suggestions?