SCALER

NEURAL NETWORK CHEAT SHEET

NN VS TRADITIONAL METHODS

WHY USE NN

SIMPLEST NN

PERCEPTRON

ISSUE

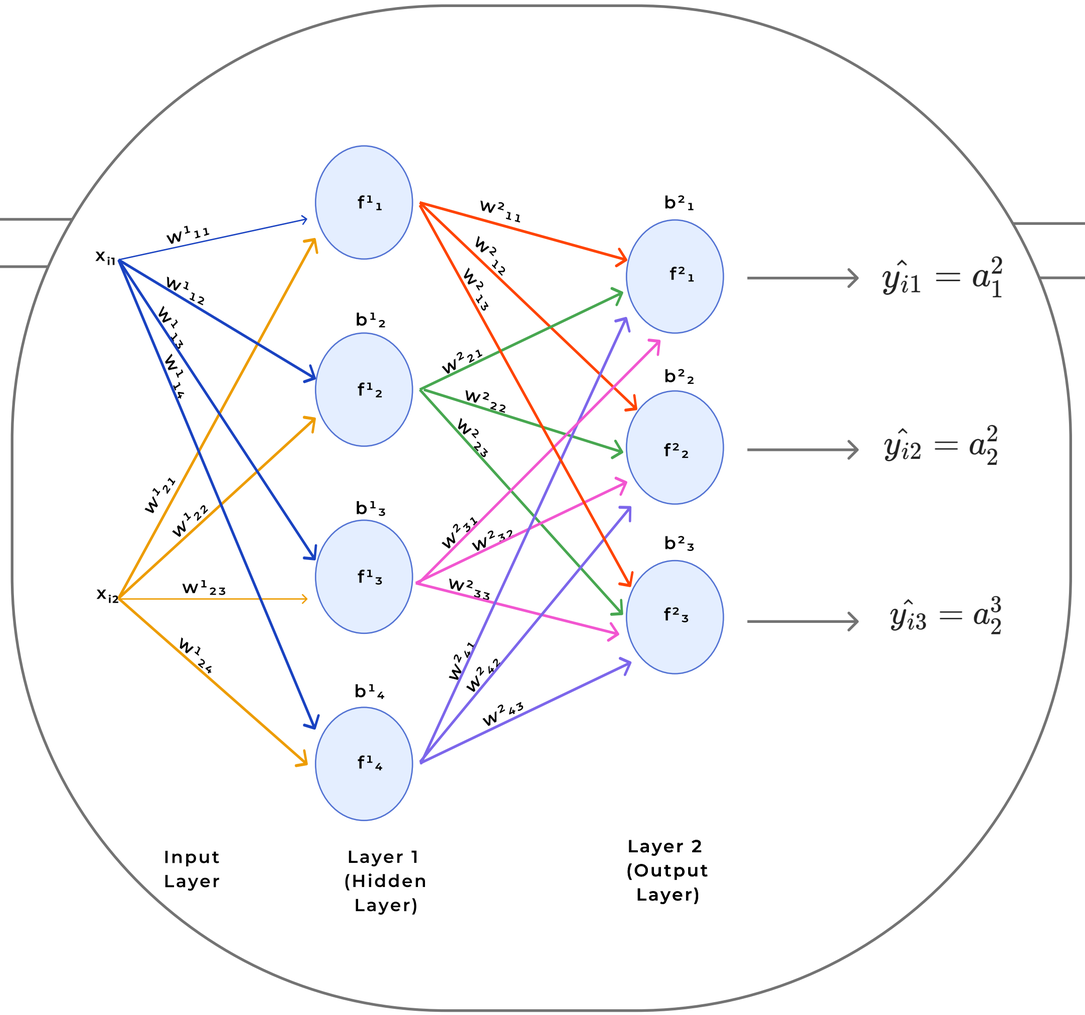
LINEAR ACTIVATION

SOLUTIONS

NON LINEAR ACTIVATION

CLASSIFICATION ACTIVATION

HYPERPARAMETERS



AUTO ENCODER

UNSUPERVISED SETTING

HOW NN TRAINS

BACKPROP & FORWARD PROP

ISSUE WHILE TRAINING

- EXPLORING GRAD
- VANISHING GRAD
- DEAD NEURON
- OVERFITTING
- TRAINING OPTIMIZATION

SOLUTIONS

EXPLORING GRAD
VANISHING GRAD
DEAD NEURON

OVERFITTING

TRAINING OPTIMIZATION

WEIGHT INITIALIZATION
GLAROT INITIALIZATION
RANDOM

REGULARIZATION
BATCH NORM
EARLY STOPPING
DROPOUT

OPTIMIZERS
LEARNING RATE
EPOCH

HYPERPARAMETERS

HYPER PARAMETER TUNING

SCALER

NEURAL NETWORK CHEAT SHEET

Neural Network Applications

- Pattern recognition and natural language processing.
- Recognition and processing of images.
- Automated translation.
- Analysis of sentiment.
- System for answering questions.
- Classification and Detection of Objects.

NN vs Traditional Machine Learning

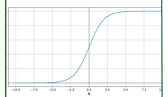
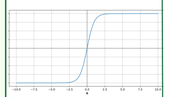
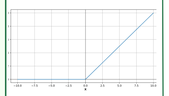
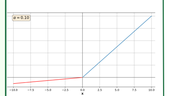
NEURAL NETWORK	TRADITIONAL ML
Neural Networks are a group of Machine Learning algorithms which mimic biological neurons	Traditional models refer to algorithms that statistically analyze and discern patterns from data
Neural Networks can adapt well to unstructured data like image and text	Traditional models work well with structures/ tabular data
Requires more computation power to train. Most NN libraries are capable of leveraging GPU for speeding the training process	Require less training time and computation resources in general
Specialized NN architectures and deep layers implicitly learn features	Manual feature engineering is required
NN are hard to interpret and often act like a black box	Traditional models are easy to interpret

PERCEPTRON

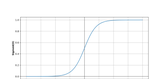
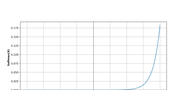
- If there is linear or no activation functions used, a neural network acts as a perceptron

- **Issue with Perceptron:**
  - A Perceptron with L layers will act same as a linear neural network without any hidden layers which makes the data information in the hidden layers redundant
  - Cannot learn non Linear patterns in data

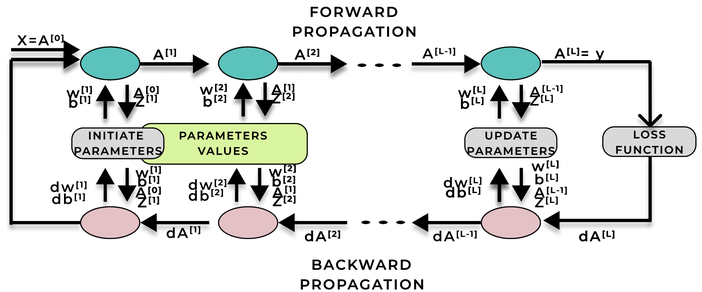
- **Non Linear Activation Function**
- Adds non-linearity to Neural Network

ACTIVATION	PLOT	EQUATION	DERIVATIVE	RANGE	PROBLEM	ADVANTAGE
SIGMOID		$\sigma(z) = \frac{1}{1+e^{-z}}$	$\sigma'(z) = \sigma(z)(1-\sigma(z))$	(0,1)	Vanishing Gradient	Adds Non Linearity
TANH		$f(z) = \frac{e^{2z}-1}{e^{2z}+1}$	$f'(z) = 1-(f(z))^2$	(-1,1)	Vanishing Gradient	· Adds non linearity · Works With Glorot Initialization
RELU		$f(z) = 0$ if $z<0$ $f(z) = z$ if $z>=0$	$f'(z) = 0$ if $z<0$ $f'(z) = 1$ if $z>=0$	$[0,\infty)$	Can Cause Dead Neuron Scenario	· To reduce possibility of vanishing Gradient and faster convergence · works with He initialization
LEAKY RELU		$f(z) = \alpha z$ if $z<0$ $f(z) = z$ if $z>=0$	$f'(z) = \alpha$ if $z<0$ $f'(z) = 1$ if $z>=0$	$(-\infty,\infty)$	-	· Solves the issue with ReLU

OUTPUT LAYER FUNCTION FOR CLASSIFICATION

OUTPUT LAYER FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE	CLASSIFICATION
SIGMOID		$\sigma(z) = \frac{1}{1+e^{-z}}$	$\sigma'(z) = \sigma(z)(1-\sigma(z))$	(0,1)	Binary Classification
SOFTMAX		$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$;	If $i == j$: $\frac{\partial f(z_i)}{\partial z_i} = f(z_i)(1-f(z_i))$ Else: $\frac{\partial f(z_i)}{\partial z_i} = -f(z_i) \times f(z_i)$	[0,1]	Multi- Class Classification

Neural Network Training

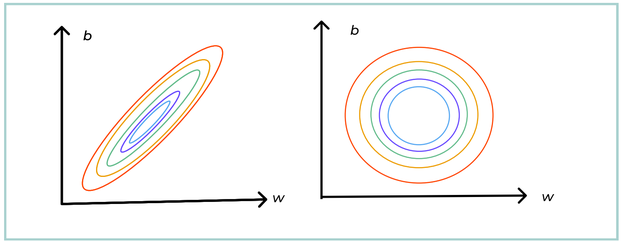


Issues while training Neural Network

Training Issue	Explanation	Reason	Solution
Exploding Gradients	If model weights grow exponentially and become unexpectedly large during training, Exploding Gradients happen.	For the L layered NN, the L derivatives are multiplied together and if the gradients have value >1 then the gradient for the Lth layer increases exponentially.	**Weight Initialization :** Vanishing/exploding gradients are caused when the value of weights are large. it can be avoided if the variance of the weights can be kept small Glorot/Xavier Initialization: Glorot Normal
Vanishing Gradients	When back-propagating to the initial layers, the derivatives of L hidden layers are multiplied together and if the gradients are small, then the gradient value decreases exponentially which slows the model's learning process	It is caused when the gradient values become < 1	$w^k_{ij} \sim N(0, \sigma_{ij})$, where, $\sigma_{ij} = \sqrt{\frac{2}{fan_{in}+fan_{out}}}$ Glorot Uniform $w^k_{ij} \sim Uniform\left[\frac{-\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}\right]$ **He Initialization :** He Normal $N(0, \sigma)$, where $\sigma = \sqrt{\frac{2}{fan_{in}}}$ He Uniform $Uniform\left[\sqrt{\frac{-6}{fan_{in}}}, \sqrt{\frac{6}{fan_{in}}}\right]$
Dead Neurons	If all the weights of the NN are initialized as 0 or any constant value , the gradient value for these neurons will be zero irrespective of the input, making the neurons dead which results in no learning of the Neural Network	Initializing all weights to zeros will cause the derivatives to remain the same for every weight in the Weight matrix because of which neurons will learn the same features in every iteration.	**Random Weight Initialization:** If all the weights have different values, then the gradients will be different for every layer of NN which makes every neuron useful

Optimizers	Weight Updation	How it works	Advantages	Disadvantages
Batch GD	$w_{t+1} = w_t - \frac{\alpha}{n}\sum_{i=1}^{n}\Delta w_t$ Where, t is the iteration, n is the number of samples $\Delta w_t = \frac{\partial L(w,x,y)}{\partial w}$	Takes the whole dataset for weight updation	• Easy computation. • Easy to implement. • Easy to implement.	• May trap at local minima. • If the dataset is very large, then the time for convergence will be very high. • Requires large memory to calculate gradient on the whole dataset.
Stochastic GD	$w_{t+1} = w_t - \alpha\Delta w_t$ Where $\Delta w_t = \frac{\partial L(w,x_i,y_i)}{\partial w}$; $i \in [1,n]$	Updates the weight on a random sample of the dataset.	• Frequent updates of model weights hence, converges in less time. • Requires less memory.	• Due to frequent updates, the optimizer steps will be noisy, hence high variance in weight value. • May shoot even after achieving global minima. • To get the same convergence as Batch gradient descent, it needs to slowly reduce the value of learning rate per epoch.
Mini Batch GD	$w_{t+1} = w_t - \alpha\Delta w_t$ Where $\Delta w_t = \frac{\partial L(w,x_{B(i)},y_{B(i)})}{\partial w}$;. $B(i)$ is the batch	Updates the weights based on a random batch of the dataset.	• Frequently updates the weight value with less variance. • Requires medium amount of memory. • Improvement over Batch GD and SGD.	• Choosing an optimum value of the learning rate. • May get trapped at local minima.
Momentum	$v_t = \beta v_{t-1} + (1-\beta)\Delta w_t$ $w_t = w_t - \alpha v_t$	• Updates the weights based on a random batch of the dataset. • Uses previous batch gradients to boost the momentum of GD in the direction of convergence.	• Reduces the noisy steps by gaining information from previous steps. • Further reduces the variance of the weight. • Converges faster than gradient descent.	One more hyperparameter β is added.
RMSprop	$s_t = \beta s_{t-1} + (1-\beta)(\Delta w_t)^2$ $w_t = w_t - \alpha\frac{\Delta w_t}{\sqrt{s_t+\epsilon}}$	• Updates the weights based on a random batch of the dataset. • Further reduces the noisy steps, by penalizing the gradients of the weight which causes it.	• The algorithm converges quickly. • Requires lesser convergence time than gradient descent.	• The learning rate has to be defined manually and may not work for every application.
Adam	Momentum: $v_t = \beta_1 v_{t-1} + (1-\beta_1)\Delta w_t$ RMSprop: $s_t = \beta_2 s_{t-1} + (1-\beta_2)(\Delta w_t)^2$ Bias Correction: $v_t^{corr} = \frac{v_t}{1-\beta_1}, s_t^{corr} = \frac{s_t}{1-\beta_2}$	• Updates the weights based on a random batch of the dataset. • Uses both the Momentum and RMSprop techniques to reduce training time and variance. • Removes biasness so to have a better moving average at the starting.	• The method is too fast and converges rapidly. • Rectifies vanishing learning rate, high variance.	• Computationally costly.

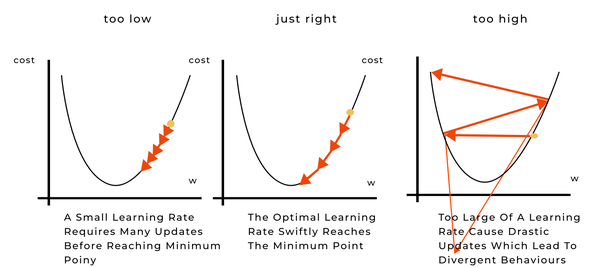
Batch Normalization



- Batch Normalization normalizes the input by performing scaling and shifting in every layer of the NN which not only prevents internal covariate shift but also helps in faster convergence as normalized data tends to have a circular shaped loss function plot (which helps the optimizer to reach global minima faster) as compared to elliptical one.
- It's a good practise to use Batch Normalization before activation function.
- **Scaling in Batch Normalization.**
  - mean$(\mu) = \frac{1}{m}\sum_{i=1}^{m} z_i$
  - variance $(\sigma^2) = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu)^2$
  - $znorm_i = \frac{z_i - \mu}{\sqrt{\sigma^2+\epsilon}}$
- **Shifting in Batch Normalization.**
  - Shifting: $\hat{z_i} = \gamma znorm_i + \beta$

Impact of learning rate on training

- Learning rate has a huge impact on training time of a NN as:



- Priori selection of Learning rate is hard, hence:
  - Learning Rate Decay is used

OVERFITTING IN NEURAL NETWORK:

- Updating Weights With High Dimensional Matrix For Deep Neural Network Is What Makes The Model Overfit On Data

RESOLVING OVERFITTING:

Frobenius Norm Regularization:

- Due to hidden layers in NN, L2-Regularization cannot be used, hence to regularize the weights, Frobenius Norm is used

$$\text{Regularization} = \frac{\lambda}{2n} \sum_{k=1}^{k=L} ||W^k||_F^2$$

$$\text{where } ||W^k||_F^2 = \sum_{i=1}^{n^{k-1}} \sum_{j=1}^{n^k} (w_{ij}^k)^2$$

Here:

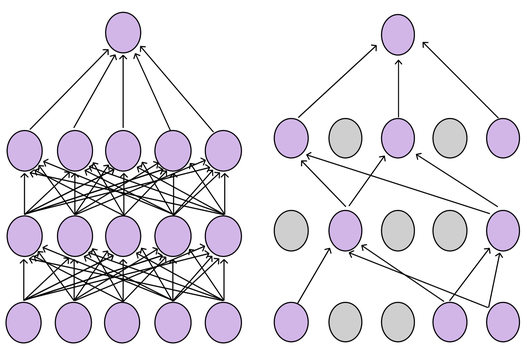
n : number of samples in the data

n^{k-1} : number of neurons in layer $k-1$

n^k : number of neurons in layer k

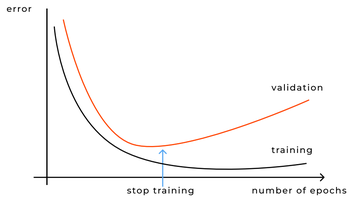
L : total number of layers in the neural network

Dropout:



- Randomly drops neurons/edges during training
- Rate at which dropout happens is called Dropout Rate (r)
- During test time, no dropout takes place, but weight values are multiplied by a factor of p = 1-r
- Dropout causes a single neural network to have different network architecture while training, which improves generalization errors as the NN cannot put a higher weight value to any of the features.

Early Stopping



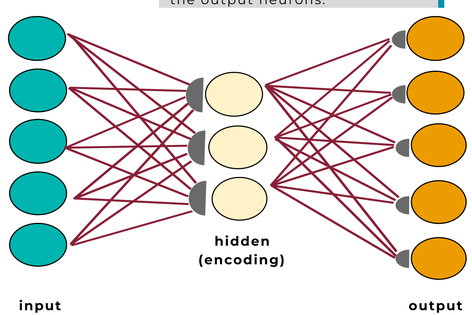
- After maximum validation performance is reached, the Neural Network is trained for a few epochs and if there is no improvement in performance, then training stops

Neural Network in Unsupervised setting

Autoencoders

It is a neural network that has 3 layers

The network is trained to resconstruct its inputs. Here, the input neurons are equal to the output neurons.



The network's target output is same as the input. It uses dimensionality reduction to restructure the input.

WHY USE AUTOENCODERS?

- Curse of Dimensionality causes lots of difficulties while training a model because it requires training a lot of parameters on a scarce dataset
- PCA and autoencoders are used to tackle these issues.

AUTOENCODERS VS PCA

AUTOENCODERS	PCA
Autoencoders are NN which are used for compressing the data into a low dimensional latent space and then try to reconstruct the actual high dimensional data.	The main idea of PCA is to find the best value of vector u, which is the direction of maximum variance (or maximum information) and along which we should rotate our existing coordinates. The eigenvector associated with the largest eigenvalue indicates the direction in which the data has the most variance.
Autoencoders are usually preferred when there is a need for modeling non-linearities and relatively complex relationships.	PCA essentially learns a linear transformation.

HYPERPARAMETERS

1. Learning Rate

Learning rate controls how quickly the Neural Network reaches global minima. Discussed in detail in impact of Learning Rate.

2. Momentum

Momentum helps know the direction of the next step with the knowledge of the previous step. This will help prevent oscillations(slow convergence). Discussed in detail in Optimizer

3. Number Of Epochs

- Controls the number of iterations a Neural Network be trained for
- Too many Epochs can lead to Overfitting
- Fewer Epochs may lead to Underfitting

4. Batch Size

- Controls the number of samples passed in one iteration
- Increased Batch size, hinders the model to generalize better

5. Initialization Of Weights

- Use Glorot Normal or Uniform Distribution
- Or use He Normal or Uniform Distribution

6. Setting The Number Of Hidden Layers

With the increasing number of hidden layers and neurons, the number of parameters increases which makes the neural network not only overfit but also increases the number of computations and training time.