

Cheat Sheet – Bias-Variance Tradeoff

What is Bias?

- Error between average model prediction and ground truth
- The bias of the estimated function tells us the capacity of the underlying model to predict the values

$$\text{bias} = \mathbb{E}[f'(x)] - f(x)$$

What is Variance?

- Average variability in the model prediction for the given dataset
- The variance of the estimated function tells you how much the function can adjust to the change in the dataset

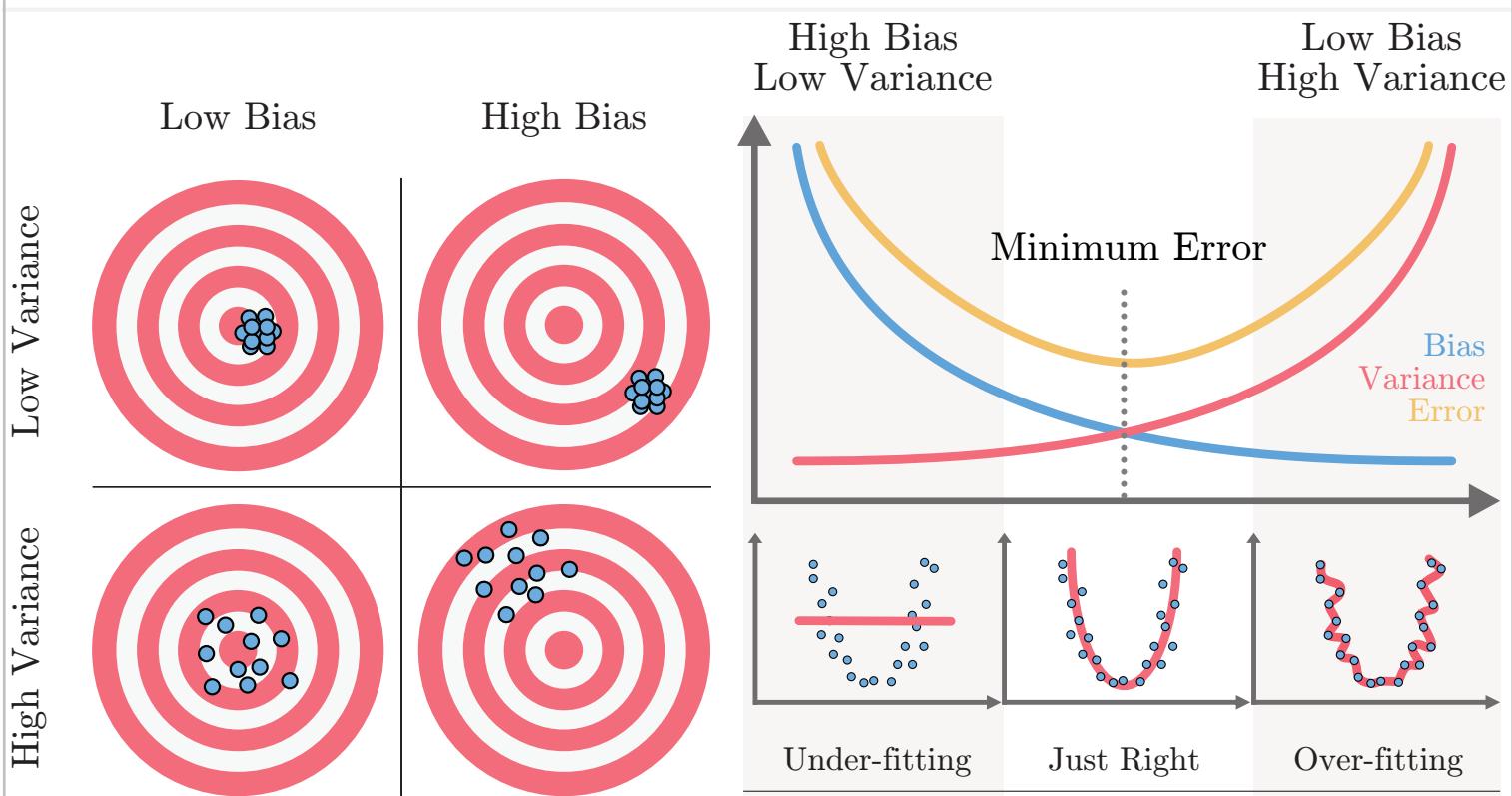
$$\text{variance} = \mathbb{E}[(f'(x) - \mathbb{E}[f'(x)])^2]$$

High Bias

- Overly-simplified Model
- Under-fitting
- High error on both test and train data

High Variance

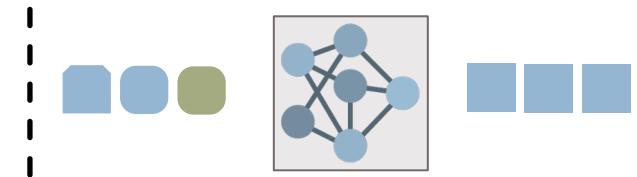
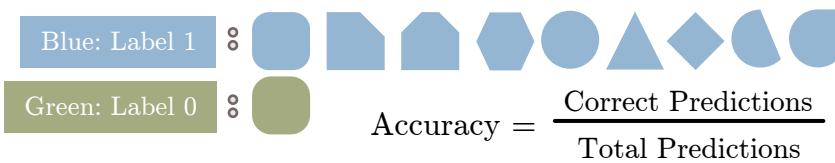
- Overly-complex Model
- Over-fitting
- Low error on train data and high on test
- Starts modelling the noise in the input



Bias variance Trade-off

- Increasing bias (not always) reduces variance and vice-versa
- Error = bias² + variance + irreducible error
- The best model is where the error is reduced.
- Compromise between bias and variance

Cheat Sheet – Imbalanced Data in Classification



Classifier that always **predicts label blue** yields prediction accuracy of 90%

Accuracy doesn't always give the correct insight about your trained model

Accuracy: %age correct prediction
Precision: Exactness of model

Correct prediction over total predictions
 From the detected cats, how many were actually cats

One value for entire network
 Each class/label has a value

Recall: Completeness of model
F1 Score: Combines Precision/Recall

Correctly detected cats over total cats
 Harmonic mean of Precision and Recall

Each class/label has a value
 Each class/label has a value

Performance metrics associated with Class 1

		Actual Labels	
		1	0
Predicted Labels	1	True Positive	False Positive
	0	False Negative	True Negative

(Is your prediction correct?) (What did you predict)

True	Negative
(Your prediction is <u>correct</u>)	(You predicted <u>0</u>)

Precision = $\frac{\text{TP}}{\text{TP} + \text{FP}}$

F1 score = $2x \frac{(\text{Prec} \times \text{Rec})}{(\text{Prec} + \text{Rec})}$

Specificity = $\frac{\text{TN}}{\text{TN} + \text{FP}}$

False +ve rate = $\frac{\text{FP}}{\text{TN} + \text{FP}}$

Accuracy = $\frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}$

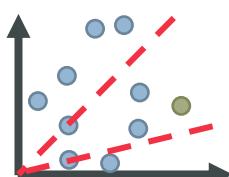
Recall, Sensitivity = $\frac{\text{TP}}{\text{TP} + \text{FN}}$
True +ve rate

Possible solutions

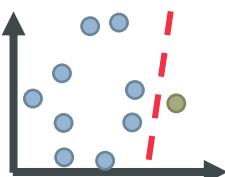
- Data Replication:** Replicate the available data until the number of samples are comparable
- Synthetic Data:** Images: Rotate, dilate, crop, add noise to existing input images and create new data
- Modified Loss:** Modify the loss to reflect greater error when misclassifying smaller sample set
- Change the algorithm:** Increase the model/algorithim complexity so that the two classes are perfectly separable (Con: Overfitting)



$$\text{loss} = a * \text{loss}_{\text{green}} + b * \text{loss}_{\text{blue}} \quad a > b$$



Increase model complexity



Straight line ($y=ax+b$) can perfectly separate data.
 Green class will no longer be predicted as blue

No straight line ($y=ax$) passing through origin can perfectly separate data. **Best solution:** line $y=0$, predict all labels blue

Cheat Sheet – PCA Dimensionality Reduction

What is PCA?

- Based on the dataset find a new set of orthogonal feature vectors in such a way that the data spread is maximum in the direction of the feature vector (or dimension)
- Rates the feature vector in the decreasing order of data spread (or variance)
- The datapoints have maximum variance in the first feature vector, and minimum variance in the last feature vector
- The variance of the datapoints in the direction of feature vector can be termed as a measure of information in that direction.

Steps

- Standardize the datapoints
- Find the covariance matrix from the given datapoints
- Carry out eigen-value decomposition of the covariance matrix
- Sort the eigenvalues and eigenvectors

$$X_{new} = \frac{X - \text{mean}(X)}{\text{std}(X)}$$

$$C[i, j] = \text{cov}(x_i, x_j)$$

$$C = V\Sigma V^{-1}$$

$$\Sigma_{sort} = \text{sort}(\Sigma) \quad V_{sort} = \text{sort}(V, \Sigma_{sort})$$

Dimensionality Reduction with PCA

- Keep the first m out of n feature vectors rated by PCA. These m vectors will be the best m vectors preserving the maximum information that could have been preserved with m vectors on the given dataset

Steps:

- Carry out steps 1-4 from above
- Keep first m feature vectors from the sorted eigenvector matrix $V_{reduced} = V[:, 0 : m]$
- Transform the data for the new basis (feature vectors) $X_{reduced} = X_{new} \times V_{reduced}$
- The importance of the feature vector is proportional to the magnitude of the eigen value

Figure 1

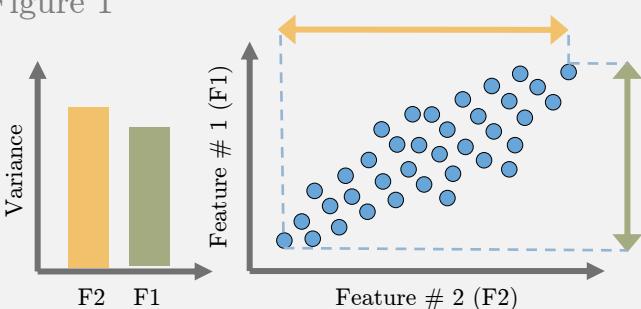


Figure 2

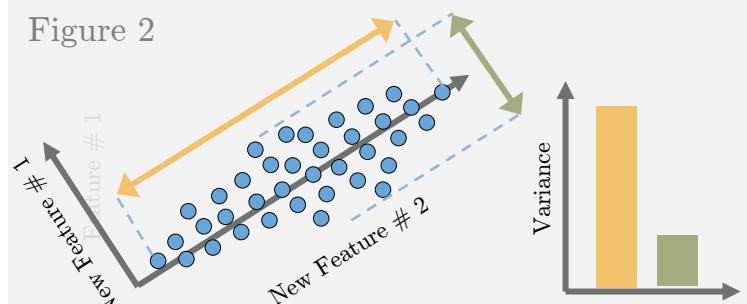


Figure 3

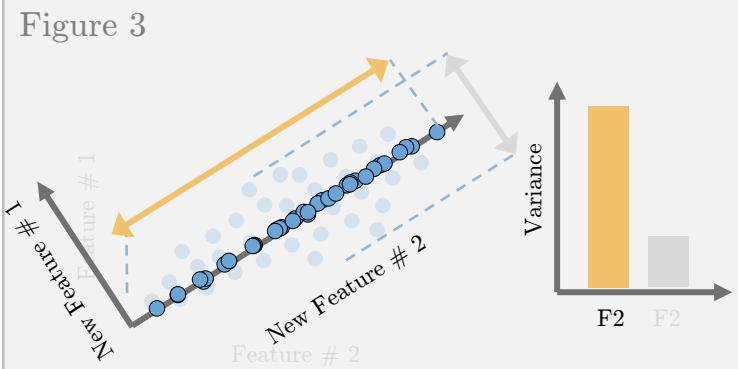


Figure 1: Datapoints with feature vectors as x and y-axis

Figure 2: The cartesian coordinate system is rotated to maximize the standard deviation along any one axis (new feature # 2)

Figure 3: Remove the feature vector with minimum standard deviation of datapoints (new feature # 1) and project the data on new feature # 2

Cheat Sheet – Bayes Theorem and Classifier

What is Bayes' Theorem?

- Describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

$$P(A|B) = \frac{P(B|A)(\text{likelihood}) \times P(A)(\text{prior})}{P(B)(\text{evidence})}$$

- How the probability of an event changes when we have knowledge of another event

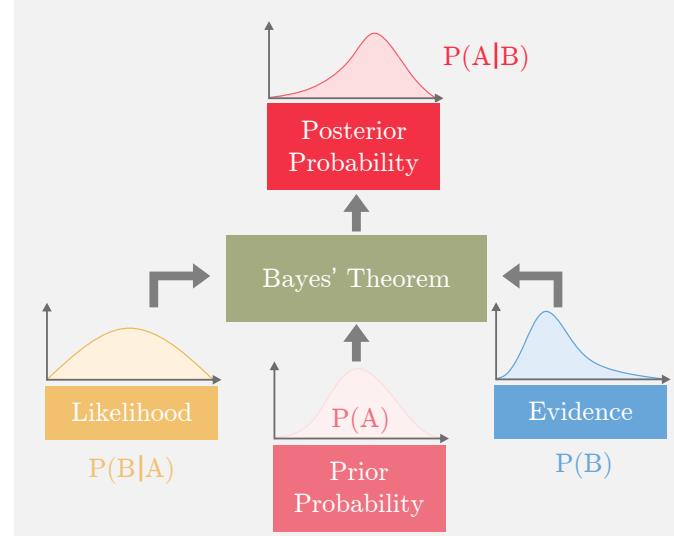
$$P(A) \longrightarrow P(A|B)$$

Usually, a better estimate than $P(A)$

Example

- Probability of fire $P(F) = 1\%$
- Probability of smoke $P(S) = 10\%$
- Prob of smoke given there is a fire $P(S|F) = 90\%$
- What is the probability that there is a fire given we see a smoke $P(F|S)?$

$$P(F|S) = \frac{P(S|F) \times P(F)}{P(S)} = \frac{0.9 \times 0.01}{0.1} = 9\%$$



Maximum Aposteriori Probability (MAP) Estimation

The MAP estimate of the random variable y , given that we have observed iid (x_1, x_2, x_3, \dots) , is given by. We try to accommodate our prior knowledge when estimating.

$$\hat{y}_{MAP} = \operatorname{argmax}_y P(y) \prod_i P(x_i|y)$$

y that maximizes the product of prior and likelihood

Maximum Likelihood Estimation (MLE)

The MAP estimate of the random variable y , given that we have observed iid (x_1, x_2, x_3, \dots) , is given by. We assume we don't have any prior knowledge of the quantity being estimated.

$$\hat{y}_{MLE} = \operatorname{argmax}_y \prod_i P(x_i|y)$$

y that maximizes only the likelihood

MLE is a special case of MAP where our prior is uniform (all values are equally likely)

Naïve Bayes' Classifier (Instantiation of MAP as classifier)

Suppose we have two classes, $y=y_1$ and $y=y_2$. Say we have more than one evidence/features (x_1, x_2, x_3, \dots) , using Bayes' theorem

$$P(y|x_1, x_2, x_3, \dots) = \frac{P(x_1, x_2, x_3, \dots | y) \times P(y)}{P(x_1, x_2, x_3, \dots)}$$

Naïve Bayes' theorem assumes the features (x_1, x_2, \dots) are i.i.d. i.e $P(x_1, x_2, x_3, \dots | y) = \prod_i P(x_i|y)$

$$P(y|x_1, x_2, x_3, \dots) = \prod_i P(x_i|y) \frac{P(y)}{P(x_1, x_2, x_3, \dots)}$$

$$\hat{y} = y_1 \text{ if } \frac{P(y_1|x_1, x_2, x_3, \dots)}{P(y_2|x_1, x_2, x_3, \dots)} > 1 \text{ else } \hat{y} = y_2$$

Cheat Sheet – Regression Analysis

What is Regression Analysis?

Fitting a function $f(\cdot)$ to datapoints $y_i = f(x_i)$ under some error function. Based on the estimated function and error, we have the following types of regression

1. Linear Regression:

Fits a **line** minimizing the **sum of mean-squared error** for each datapoint.

$$\min_{\beta} \sum_i \|y_i - f_{\beta}^{\text{linear}}(x_i)\|^2$$

$$f_{\beta}^{\text{linear}}(x_i) = \beta_0 + \beta_1 x_i$$

2. Polynomial Regression:

Fits a **polynomial** of order k ($k+1$ unknowns) minimizing the **sum of mean-squared error** for each datapoint.

$$\min_{\beta} \sum_{i=0}^m \|y_i - f_{\beta}^{\text{poly}}(x_i)\|^2$$

$$f_{\beta}^{\text{poly}}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_k x_i^k$$

3. Bayesian Regression:

For each datapoint, fits a **gaussian distribution** by minimizing the **mean-squared error**. As the number of data points x_i increases, it converges to point estimates i.e. $n \rightarrow \infty, \sigma^2 \rightarrow 0$

$$\mathcal{N}(\mu, \sigma^2) \rightarrow \text{Gaussian with mean } \mu \text{ and variance } \sigma^2$$

4. Ridge Regression:

Can fit either a **line**, or **polynomial** minimizing the **sum of mean-squared error** for each datapoint **and** the **weighted L2 norm** of the function parameters beta.

$$\min_{\beta} \sum_{i=0}^m \|y_i - f_{\beta}(x_i)\|^2 + \sum_{j=0}^k \beta_j^2$$

$$f_{\beta}(x_i) = f_{\beta}^{\text{poly}}(x_i) \text{ or } f_{\beta}^{\text{linear}}(x_i)$$

5. LASSO Regression:

Can fit either a **line**, or **polynomial** minimizing the **sum of mean-squared error** for each datapoint **and** the **weighted L1 norm** of the function parameters beta.

$$\min_{\beta} \sum_{i=0}^m \|y_i - f_{\beta}(x_i)\|^2 + \sum_{j=0}^k |\beta_j|$$

$$f_{\beta}(x_i) = f_{\beta}^{\text{poly}}(x_i) \text{ or } f_{\beta}^{\text{linear}}(x_i)$$

6. Logistic Regression:

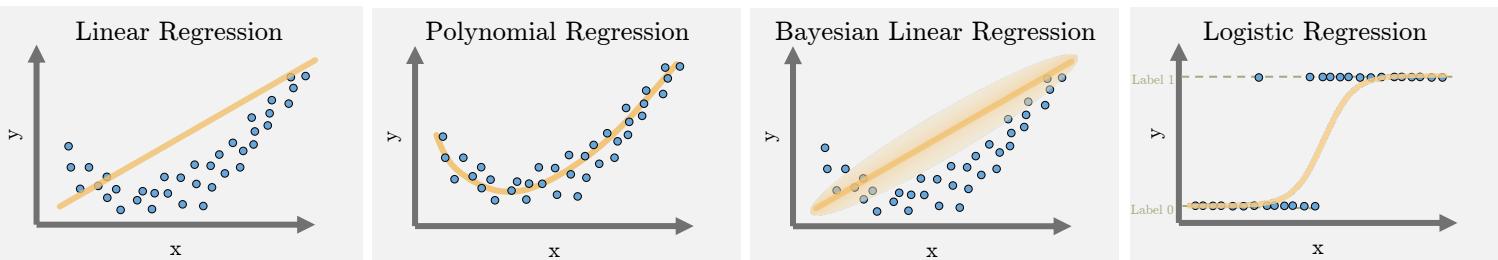
Can fit either a **line**, or **polynomial with sigmoid activation** minimizing the **binary cross-entropy loss** for each datapoint. The labels y are binary class labels.

$$\min_{\beta} \sum_i -y_i \log(\sigma(f_{\beta}(x_i))) - (1 - y_i) \log(1 - \sigma(f_{\beta}(x_i)))$$

$$f_{\beta}(x_i) = f_{\beta}^{\text{poly}}(x_i) \text{ or } f_{\beta}^{\text{linear}}(x_i)$$

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Visual Representation:



Summary:

	What does it fit?	Estimated function	Error Function
Linear	A line in n dimensions	$f_{\beta}^{\text{linear}}(x_i) = \beta_0 + \beta_1 x_i$	$\sum_{i=0}^m \ y_i - f_{\beta}(x_i)\ ^2$
Polynomial	A polynomial of order k	$f_{\beta}^{\text{poly}}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots$	$\sum_{i=0}^m \ y_i - f_{\beta}(x_i)\ ^2$
Bayesian Linear	Gaussian distribution for each point	$\mathcal{N}(f_{\beta}(x_i), \sigma^2)$	$\sum_i \ y_i - \mathcal{N}(f_{\beta}(x_i), \sigma^2)\ ^2$
Ridge	Linear/polynomial	$f_{\beta}^{\text{poly}}(x_i) \text{ or } f_{\beta}^{\text{linear}}(x_i)$	$\sum_{i=0}^m \ y_i - f_{\beta}(x_i)\ ^2 + \sum_{j=0}^n \beta_j^2$
LASSO	Linear/polynomial	$f_{\beta}^{\text{poly}}(x_i) \text{ or } f_{\beta}^{\text{linear}}(x_i)$	$\sum_{i=0}^m \ y_i - f_{\beta}(x_i)\ ^2 + \sum_{j=0}^n \beta_j $
Logistic	Linear/polynomial with sigmoid	$\sigma(f_{\beta}(x_i))$	$\min_{\beta} \sum_i -y_i \log(\sigma(f_{\beta}(x_i))) - (1 - y_i) \log(1 - \sigma(f_{\beta}(x_i)))$

Cheat Sheet – Regularization in ML

What is Regularization in ML?

- Regularization is an approach to address over-fitting in ML.
- Overfitted model fails to generalize estimations on test data
- When the underlying model to be learned is low bias/high variance, or when we have small amount of data, the estimated model is prone to over-fitting.
- Regularization reduces the variance of the model

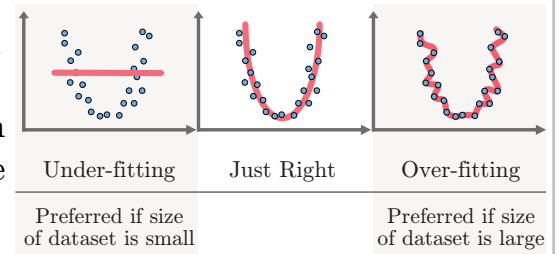


Figure 1. Overfitting

Types of Regularization:

1. Modify the loss function:

- **L2 Regularization:** Prevents the weights from getting too large (defined by L2 norm). Larger the weights, more complex the model is, more chances of overfitting.

$$\text{loss} = \text{error}(y, \hat{y}) + \lambda \sum_j \beta_j^2 \quad \lambda \geq 0, \lambda \propto \text{model bias}, \lambda \propto \frac{1}{\text{model variance}}$$

- **L1 Regularization:** Prevents the weights from getting too large (defined by L1 norm). Larger the weights, more complex the model is, more chances of overfitting. L1 regularization introduces sparsity in the weights. It forces more weights to be zero, than reducing the average magnitude of all weights

$$\text{loss} = \text{error}(y, \hat{y}) + \lambda \sum_j |\beta_j| \quad \lambda \geq 0, \lambda \propto \text{model bias}, \lambda \propto \frac{1}{\text{model variance}}$$

- **Entropy:** Used for the models that output probability. Forces the probability distribution towards uniform distribution.

$$\text{loss} = \text{error}(p, \hat{p}) - \lambda \sum_i \hat{p}_i \log(\hat{p}_i) \quad \lambda \geq 0, \lambda \propto \text{model bias}, \lambda \propto \frac{1}{\text{model variance}}$$

2. Modify data sampling:

- **Data augmentation:** Create more data from available data by randomly cropping, dilating, rotating, adding small amount of noise etc.
- **K-fold Cross-validation:** Divide the data into k groups. Train on (k-1) groups and test on 1 group. Try all k possible combinations.

3. Change training approach:

- **Injecting noise:** Add random noise to the weights when they are being learned. It pushes the model to be relatively insensitive to small variations in the weights, hence regularization
- **Dropout:** Generally used for neural networks. Connections between consecutive layers are randomly dropped based on a dropout-ratio and the remaining network is trained in the current iteration. In the next iteration, another set of random connections are dropped.

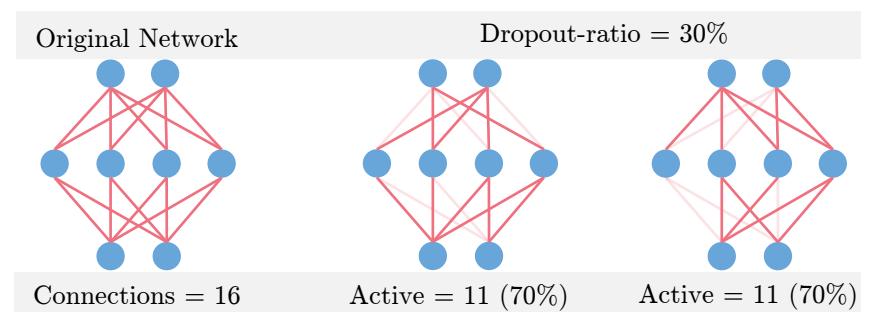
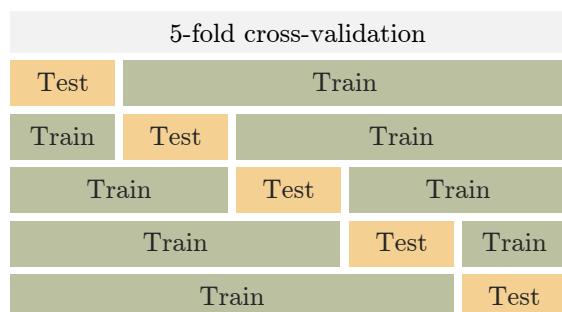


Figure 2. K-fold CV

Figure 3. Drop-out

Cheat Sheet – Convolutional Neural Network

Convolutional Neural Network:

The data gets into the CNN through the input layer and passes through various hidden layers before getting to the output layer. The output of the network is compared to the actual labels in terms of loss or error. The partial derivatives of this loss w.r.t the trainable weights are calculated, and the weights are updated through one of the various methods using backpropagation.

CNN Template:

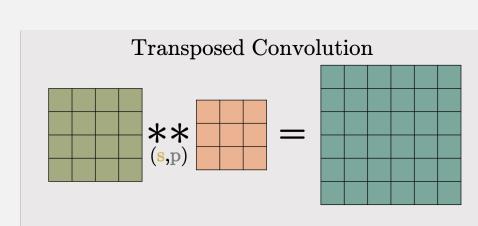
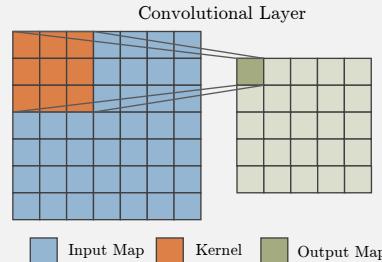
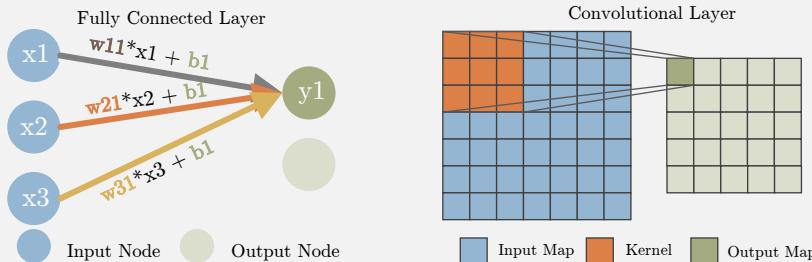
Most of the commonly used hidden layers (not all) follow a pattern

1. Layer function: Basic transforming function such as convolutional or fully connected layer.

a. Fully Connected: Linear functions between the input and the

a. Convolutional Layers: These layers are applied to 2D (3D) input feature maps. The trainable weights are a 2D (3D) kernel/filter that moves across the input feature map, generating dot products with the overlapping region of the input feature map.

b. Transposed Convolutional (DeConvolutional) Layer: Usually used to increase the size of the output feature map (Upsampling) The idea behind the transposed convolutional layer is to undo (not exactly) the convolutional layer



2. Pooling: Non-trainable layer to change the size of the feature map

a. Max/Average Pooling: Decrease the spatial size of the input layer based on selecting the maximum/average value in receptive field defined by the kernel

b. UnPooling: A non-trainable layer used to increase the spatial size of the input layer based on placing the input pixel at a certain index in the receptive field of the output defined by the kernel.

3. Normalization: Usually used just before the activation functions to limit the unbounded activation from increasing the output layer values too high

a. Local Response Normalization LRN: A **non-trainable layer** that square-normalizes the pixel values in a feature map within a local neighborhood.

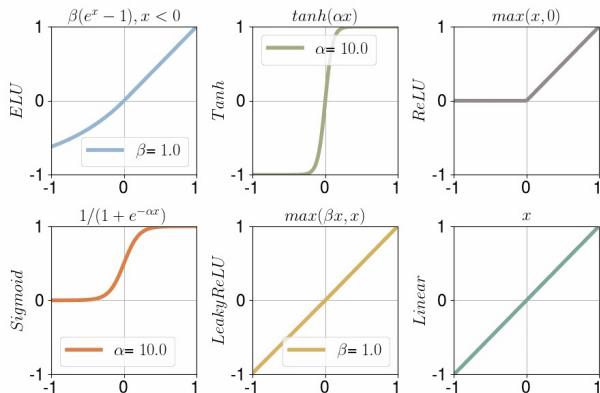
b. Batch Normalization: A trainable approach to normalizing the data by learning scale and shift variable during training.

3. Activation: Introduce non-linearity so CNN can efficiently map non-linear complex mapping.

a. Non-parametric/Static functions: Linear, ReLU

b. Parametric functions: ELU, tanh, sigmoid, Leaky ReLU

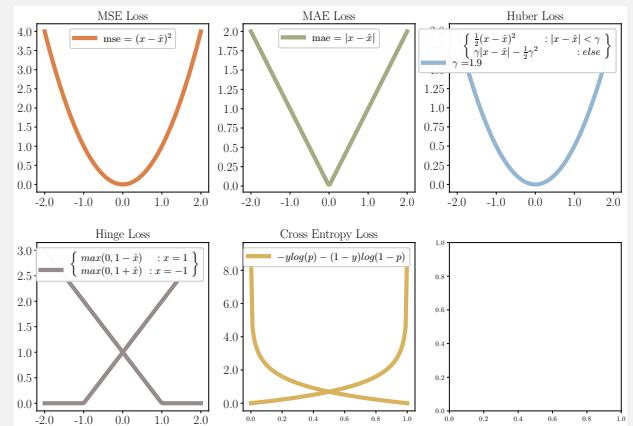
c. Bounded functions: tanh, sigmoid



5. Loss function: Quantifies how far off the CNN prediction is from the actual labels.

a. Regression Loss Functions: MAE, MSE, Huber loss

b. Classification Loss Functions: Cross entropy, Hinge loss



Cheat Sheet – Famous CNNs

AlexNet – 2012

Why: AlexNet was born out of the need to improve the results of the ImageNet challenge.

What: The network consists of 5 Convolutional (CONV) layers and 3 Fully Connected (FC) layers. The activation used is the Rectified Linear Unit (ReLU).

How: Data augmentation is carried out to reduce over-fitting, Uses Local response normalization.

VGGNet – 2014

Why: VGGNet was born out of the need to reduce the # of parameters in the CONV layers and improve on training time

What: There are multiple variants of VGGNet (VGG16, VGG19, etc.)

How: The important point to note here is that all the conv kernels are of size 3x3 and maxpool kernels are of size 2x2 with a stride of two.

ResNet – 2015

Why: Neural Networks are notorious for not being able to find a simpler mapping when it exists. ResNet solves that.

What: There are multiple versions of ResNetXX architectures where 'XX' denotes the number of layers. The most used ones are ResNet50 and ResNet101. Since the vanishing gradient problem was taken care of (more about it in the How part), CNN started to get deeper and deeper

How: ResNet architecture makes use of shortcut connections to solve the vanishing gradient problem. The basic building block of ResNet is a Residual block that is repeated throughout the network.

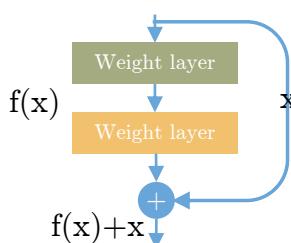


Figure 1 ResNet Block

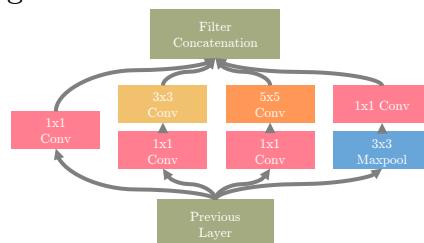


Figure 2 Inception Block

Inception – 2014

Why: Larger kernels are preferred for more global features, on the other hand, smaller kernels provide good results in detecting area-specific features. For effective recognition of such a variable-sized feature, we need kernels of different sizes. That is what Inception does.

What: The Inception network architecture consists of several inception modules of the following structure. Each inception module consists of four operations in parallel, 1x1 conv layer, 3x3 conv layer, 5x5 conv layer, max pooling

How: Inception increases the network space from which the best network is to be chosen via training. Each inception module can capture salient features at different levels.

AlexNet Network - Structural Details										
Input	Output	Layer	Stride	Pad	Kernel size	in	out	# of Param		
227x227x3	55x55x96	conv1	4	0	11	11	3	96	34944	
55x55x96	27x27x96	maxpool1	2	0	3	96	256	614656		
27x27x96	13x13x96	conv2	1	2	5	96	256	614656		
13x13x96	13x13x256	maxpool2	2	0	3	256	256	0		
13x13x256	13x13x384	conv3	1	1	3	3	256	384	885120	
13x13x384	13x13x384	conv4	1	1	3	3	384	384	1327488	
13x13x384	13x13x256	conv5	1	1	3	3	384	256	884992	
13x13x256	6x6x256	maxpool5	2	0	3	3	256	256	0	
		f6				1	1	9216	4096	37752832
		f7				1	1	4096	4096	16781312
		f8				1	1	4096	1000	4097000
		Total								62378.344

VGG16 - Structural Details										
#	Input	Image	output	Layer	Stride	Kernel	in	out	Param	
1	224	224	3	224x224	64	conv3-64	1	3	3	64
2	224	224	64	224x224	64	conv3064	1	3	3	64
3	112	112	64	112x112	128	conv3-128	1	3	3	64
4	112	112	128	112x112	128	conv3-128	2	2	2	128
5	56	56	128	56x56	128	maxpool	2	2	2	128
6	56	56	256	56x56	256	conv3-256	1	3	3	256
7	56	56	256	56x56	256	conv3-256	1	3	3	256
8	28	28	256	28x28	512	maxpool	2	2	2	256
9	28	28	512	28x28	512	conv3-512	1	3	3	512
10	28	28	512	28x28	512	conv3-512	3	3	3	512
11	14	14	512	14x14	512	conv3-512	2	2	2	512
12	14	14	512	14x14	512	conv3-512	1	3	3	512
13	14	14	512	7	512	maxpool	2	2	2	512
14	1	1	25088	1	1	4096	fc	1	1	25088
15	1	1	4096	1	1	4096	fc	1	1	4096
16	1	1	1000	1	1	1000	fc	1	1	1000
							Total			138,423,208

ResNet18 - Structural Details										
#	Input	Image	output	Layer	Stride	Pad	Kernel	in	out	Param
1	227x227	3	112x112	64	conv1	2	1	7	7	64
2	112	112	64	56	64	maxpool	2	0	5	3
3	56	56	64	56	64	conv2-1	1	1	3	64
4	56	56	64	56	64	conv2-2	1	1	3	64
5	56	56	64	56	64	conv2-4	1	1	3	64
6	56	56	64	28	28	conv3-1	2	0	5	3
7	28	28	128	28	128	conv3-2	1	1	3	128
8	28	28	128	28	128	conv3-3	1	1	3	128
9	28	28	128	28	128	conv3-4	1	1	3	128
10	28	28	128	14	128	conv3-5	2	0	5	256
11	14	14	256	14	128	conv3-6	2	0	5	256
12	14	14	256	7	512	conv5-1	2	0	5	3
13	7	7	512	7	512	conv5-2	1	1	3	512
14	7	7	512	7	512	conv5-3	1	1	3	512
15	7	7	512	7	512	conv5-4	1	1	3	512
16	7	7	512	1	1	512	avg pool	7	0	7
17	1	1	1000	1	1	1000	fc	1	1	1000
							Total			11,511,784

GoogLeNet - Structural Details											
#	Input	Image	output	Layer	Input Layer	Stride	Pad	Kernel	in	out	Param
1	227x227x3	3	110x110x64	conv1	conv1	2	0	3	64	64	0
2	110x110x64	64	56x56x64	maxpool	conv1	2	0	3	64	64	0
3	56x56x64	64	56x56x64	conv1x1	maxpool	1	0	1	1	64	64
4	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	1	64	64
5	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
6	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
7	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
8	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
9	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
10	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
11	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
12	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
13	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
14	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
15	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
16	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
17	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
18	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
19	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
20	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
21	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
22	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
23	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
24	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
25	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
26	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
27	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
28	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
29	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
30	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
31	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
32	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
33	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
34	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
35	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
36	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
37	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
38	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
39	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
40	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
41	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
42	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	64	64
43	56x56x64	64	56x56x64	conv1x1	conv1x1	1	0	1	3	6	

Cheat Sheet – Ensemble Learning in ML

What is Ensemble Learning? Wisdom of the crowd

Combine multiple weak models/learners into one predictive model to reduce bias, variance and/or improve accuracy.

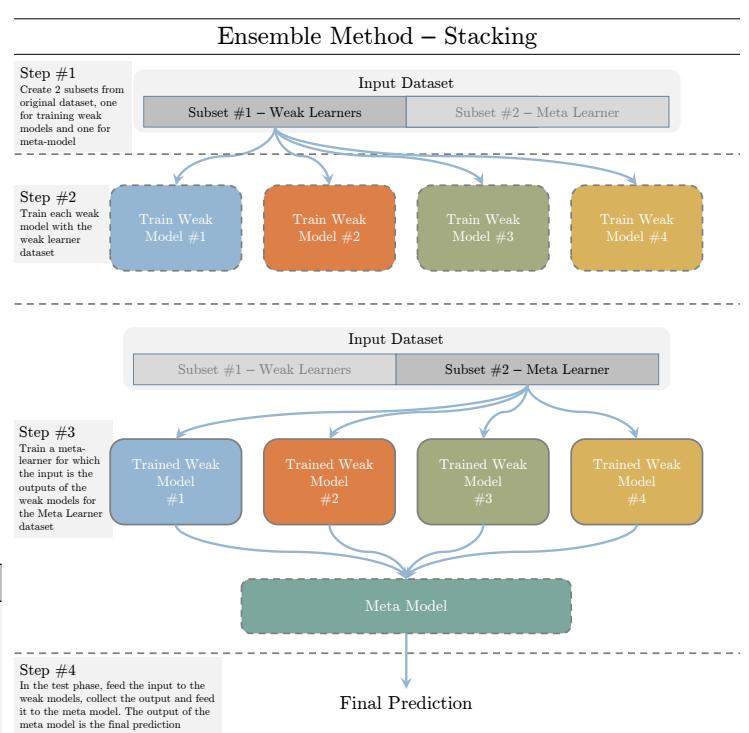
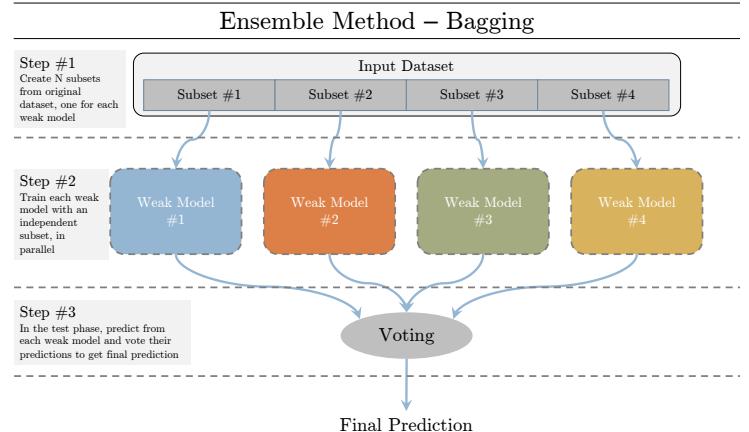
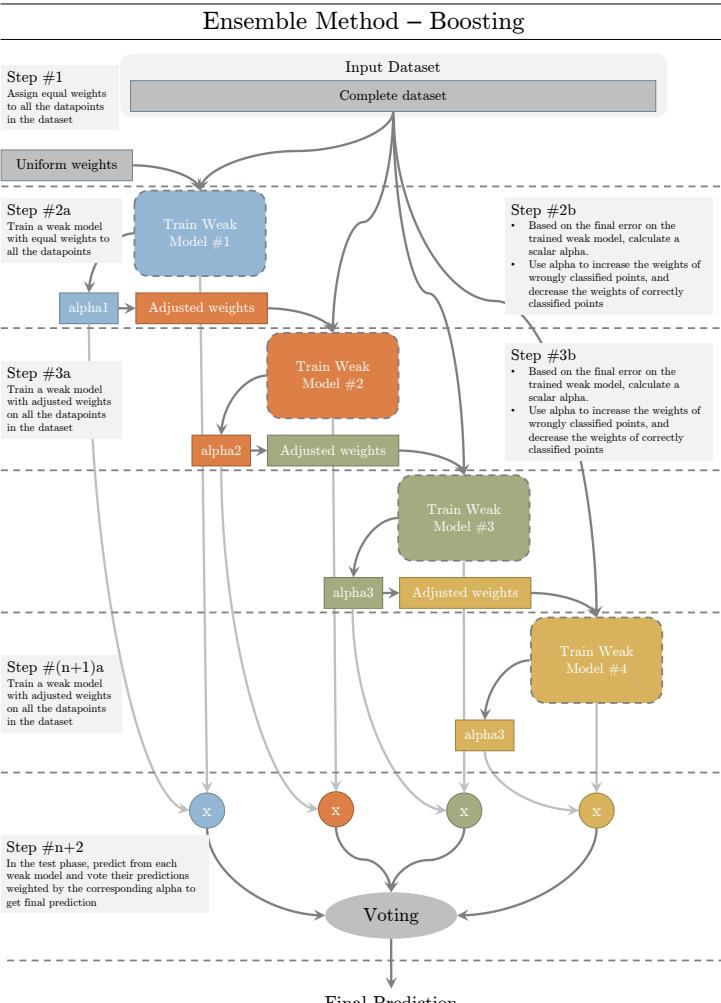
Types of Ensemble Learning: N number of weak learners

1.Bagging: Trains N different weak models (usually of same types – homogenous) with N non-overlapping subset of the input dataset in parallel. In the test phase, each model is evaluated. The label with the greatest number of predictions is selected as the prediction. Bagging methods reduces variance of the prediction

2.Boosting: Trains N different weak models (usually of same types – homogenous) with the complete dataset in a sequential order. The datapoints wrongly classified with previous weak model is provided more weights to that they can be classified by the next weak leaner properly. In the test phase, each model is evaluated and based on the test error of each weak model, the prediction is weighted for voting. Boosting methods decreases the bias of the prediction.

3.Stacking: Trains N different weak models (usually of different types – heterogenous) with one of the two subsets of the dataset in parallel. Once the weak learners are trained, they are used to trained a meta learner to combine their predictions and carry out final prediction using the other subset. In test phase, each model predicts its label, these set of labels are fed to the meta learner which generates the final prediction.

The block diagrams, and comparison table for each of these three methods can be seen below.



Parameter	Bagging	Boosting	Stacking
Focuses on	Reducing variance	Reducing bias	Improving accuracy
Nature of weak learners is	Homogenous	Homogenous	Heterogenous
Weak learners are aggregated by	Simple voting	Weighted voting	Learned voting (meta-learner)

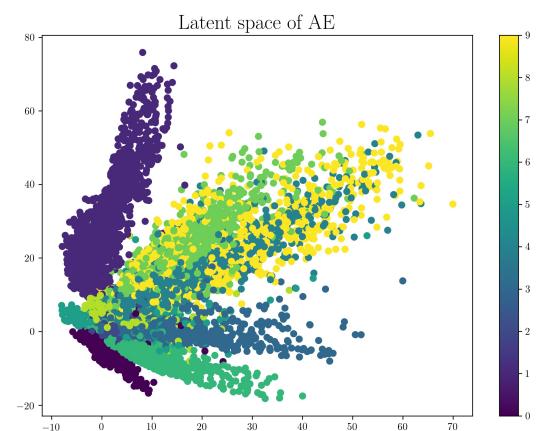
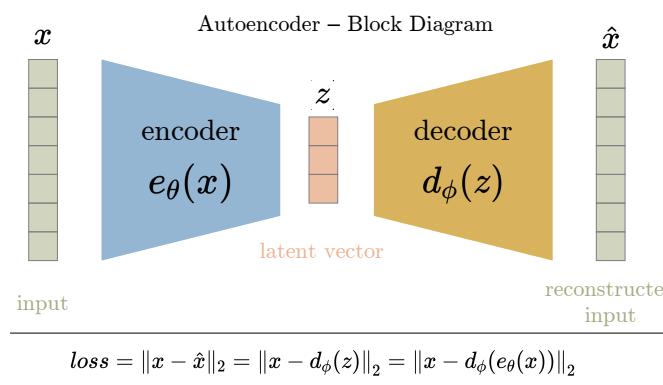
Cheat Sheet – Autoencoder & Variational Autoencoder

Context – Data Compression

- Data compression is an essential phase in training a network. The idea is to compress the data so that the same amount of information can be represented by fewer bits.

Auto Encoder (AE)

- Autoencoder is used to learn efficient embeddings of unlabeled data for a given network configuration. It consists of two parts, an **encoder**, and a **decoder**.
- The encoder compresses the data from a higher-dimensional space to a lower-dimensional space (also called the latent space), while the decoder converts the latent space back to higher-dimensional space.
- The entire encoder-decoder architecture is collectively trained on the loss function which encourages that the input is reconstructed at the output. Hence the loss function is the mean squared error between the encoder input and the decoder output.
- The latent variable is not regularized. Picking a random latent variable will generate garbage output.
- Latent variable is deterministic values and the space lacks the generative capability



Variational Auto Encoder (VAE)

- Variational autoencoder addresses the issue of non-regularized latent space in autoencoder and provides the generative capability to the entire space.
- Instead of outputting the vectors in the latent space, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input.
- The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution.
- The latent variable in the compressed form is mean and variance
- The training loss of VAE is defined as the sum of the reconstruction loss and the similarity loss (the KL divergence between the unit gaussian and decoder output distribution).
- The latent variable is smooth and continuous i.e., random values of latent variable generates meaningful output at the decoder, hence the latent space has generative capabilities.
- The input of the decoder is sampled from a gaussian with mean/variance of the output of encoder.

