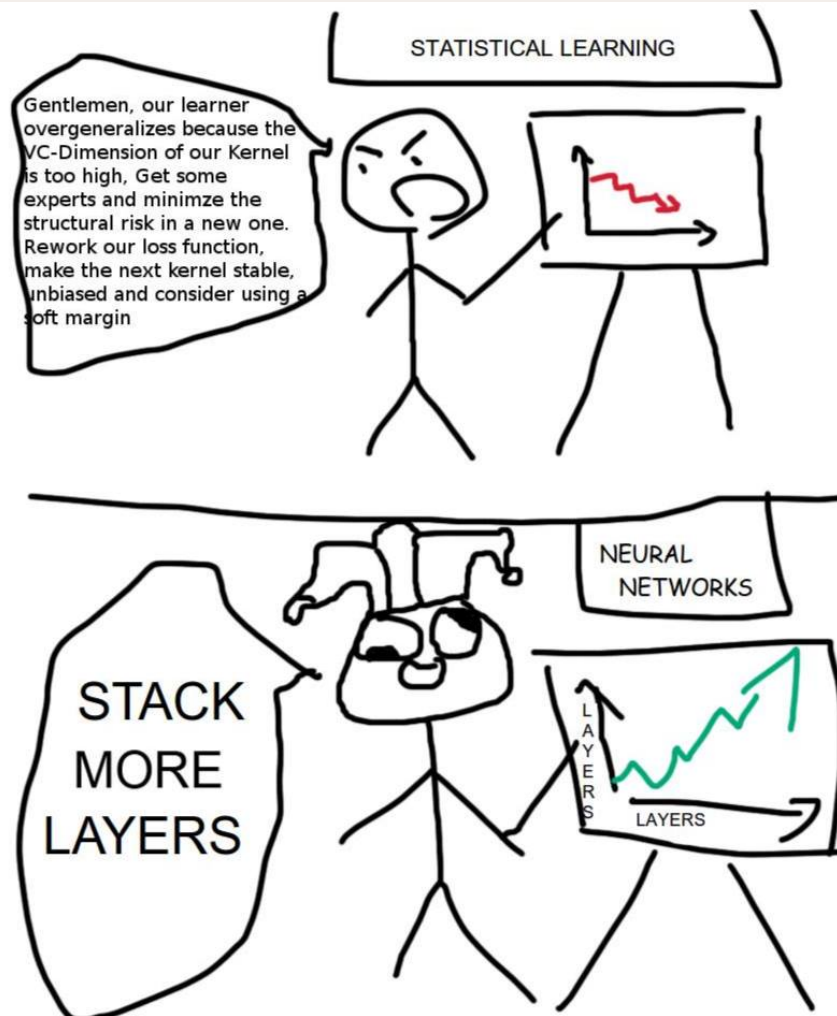
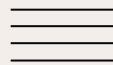
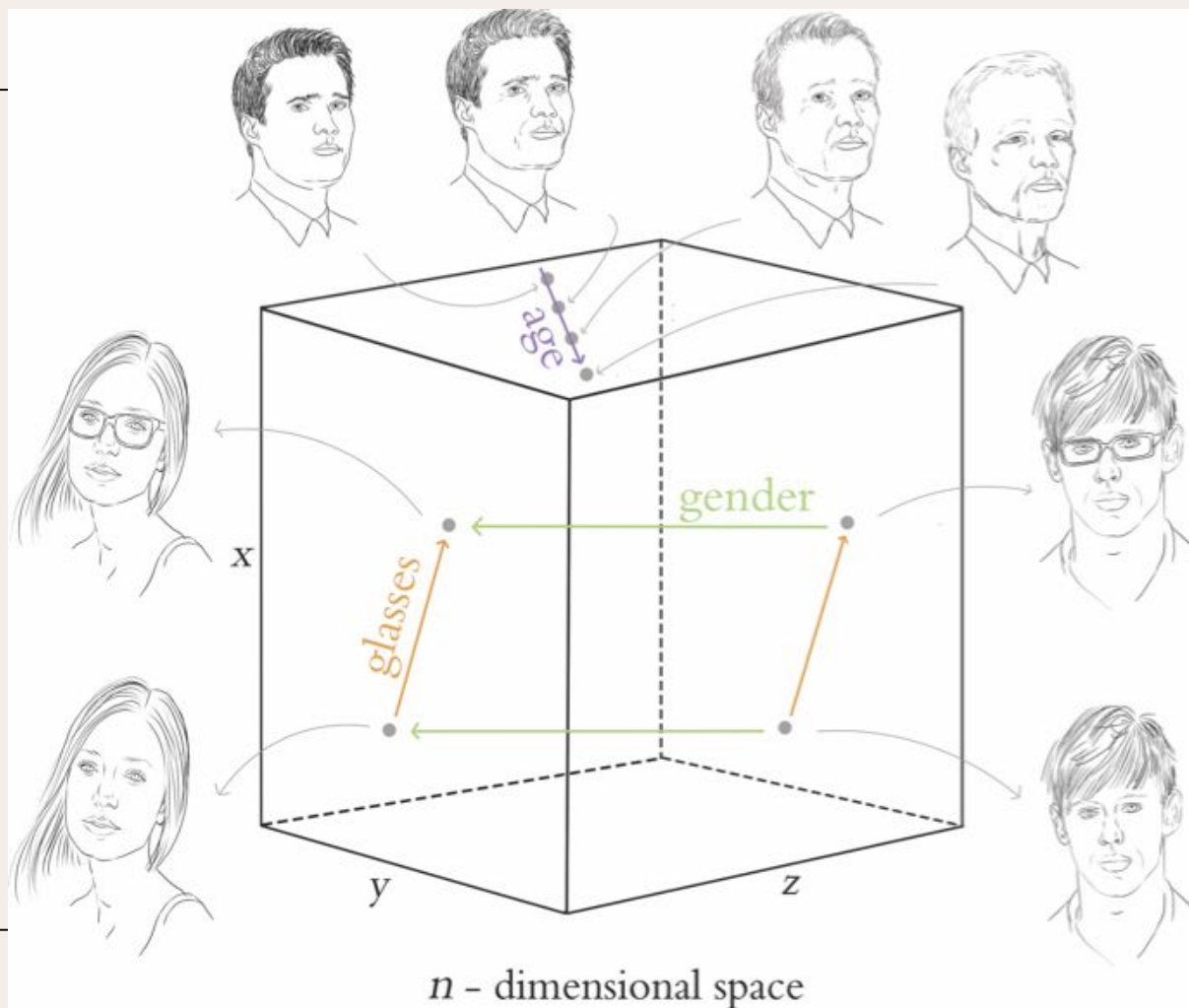
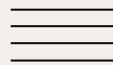


<welcome to>

SIG AI

Meeting 3: 04/04/24







(ooga booga) What is Deep Learning?

- Deep learning is a subset of machine learning that uses multi-layered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain. Some form of deep learning powers most of the artificial intelligence (AI) in our lives today.
- By strict definition, a deep neural network, or DNN, is a neural network with three or more layers. In practice, most DNNs have many more layers. DNNs are trained on large amounts of data to identify and classify phenomena, recognize patterns and relationships, evaluate possibilities, and make predictions and decisions.
- While a single-layer neural network can make useful, approximate predictions and decisions, the additional layers in a deep neural network help refine and optimize those outcomes for greater accuracy.

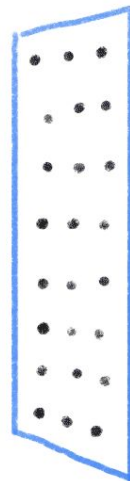
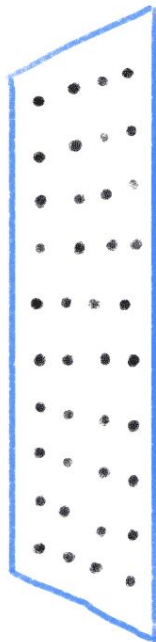
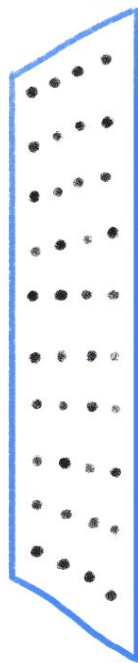
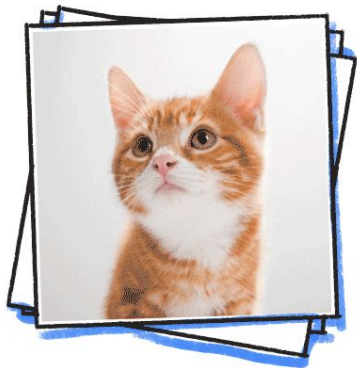


**What use-case can you think of,
where using unlabelled data
might be beneficial, with more
than just input-compute-output?**

CAT

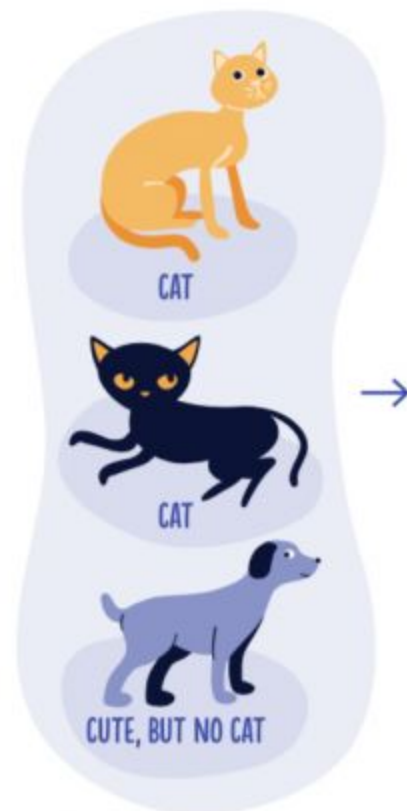
(Labeled
PHOTOS)

DOG

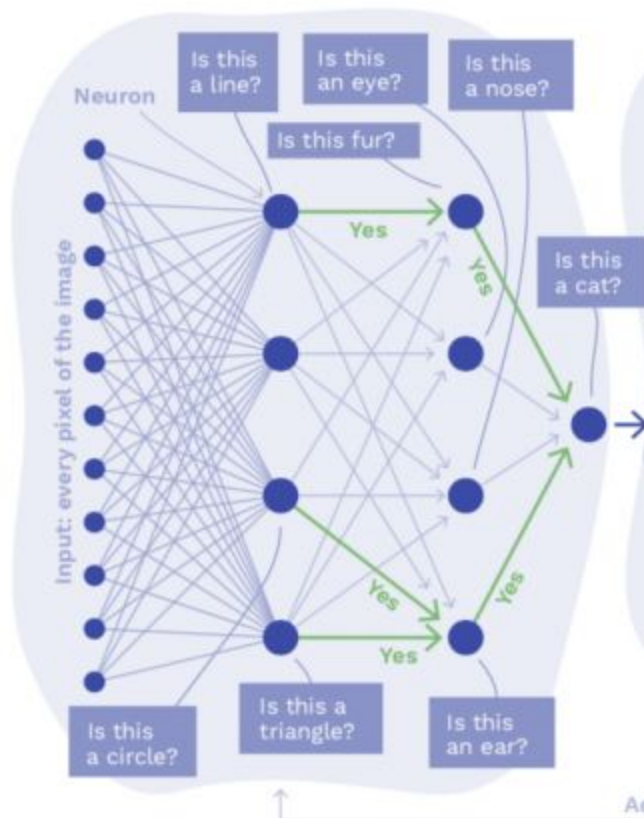


OUTPUT

LABELS / IMAGES



ALGORITHM



PREDICTION



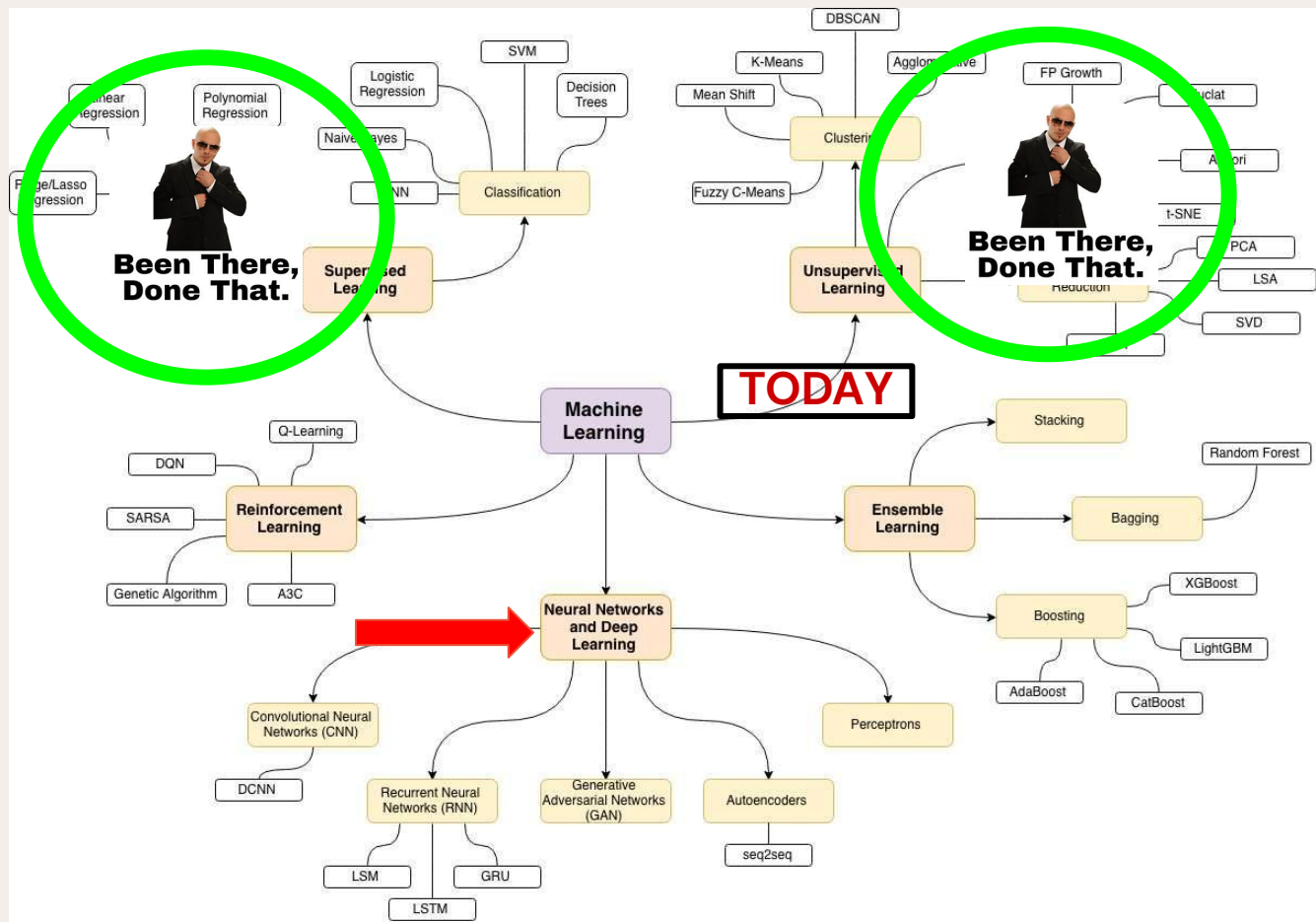
TRAINING





Is this a cat or a dog?





What are we tackling today?

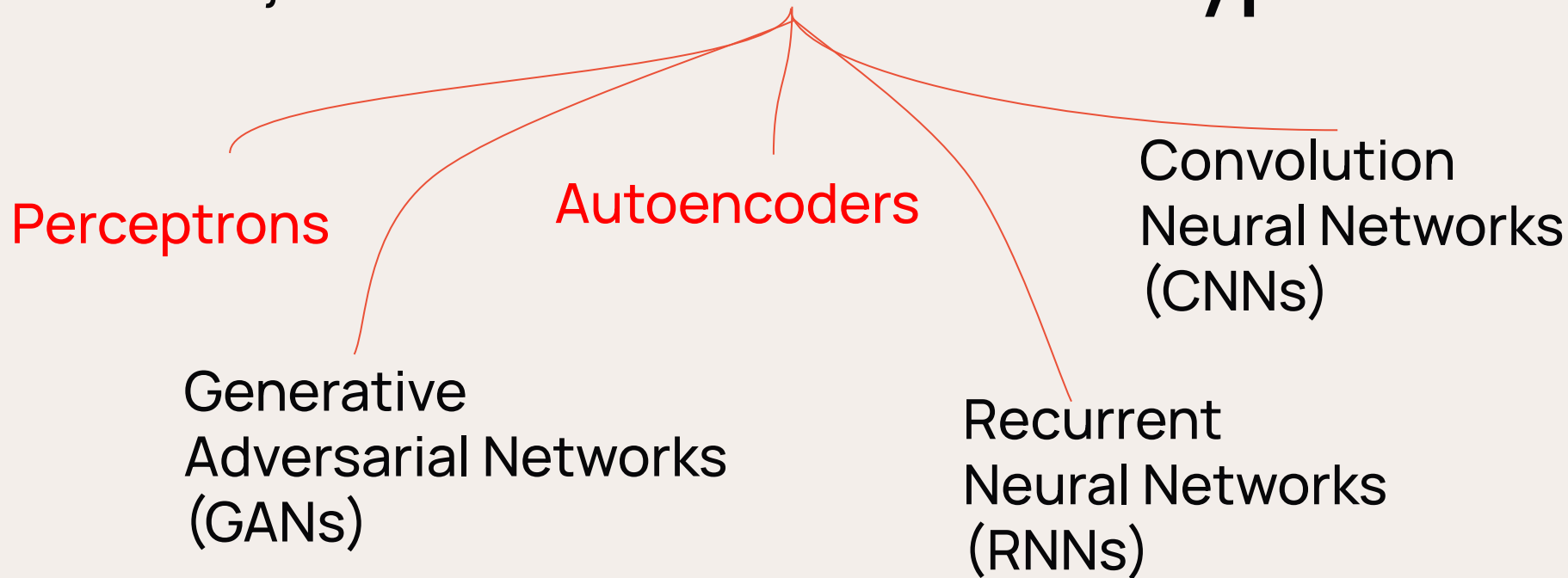
Machine Learning

- Deep Learning

- Perceptrons

- Autoencoders

Ok, what are the different types? *



Ok, what are the different types? *



Perceptrons



The basis of the idea of the perceptron is rooted in the words perception (the ability to sense something) and neurons (nerve cells in the human brain that turn sensory input into meaningful information).

Autoencoders



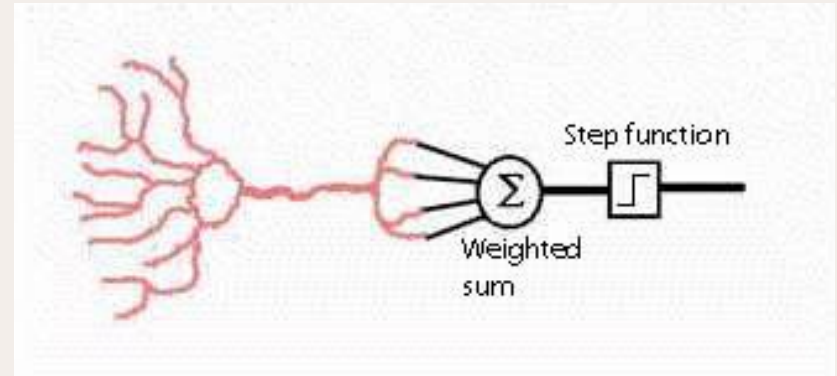
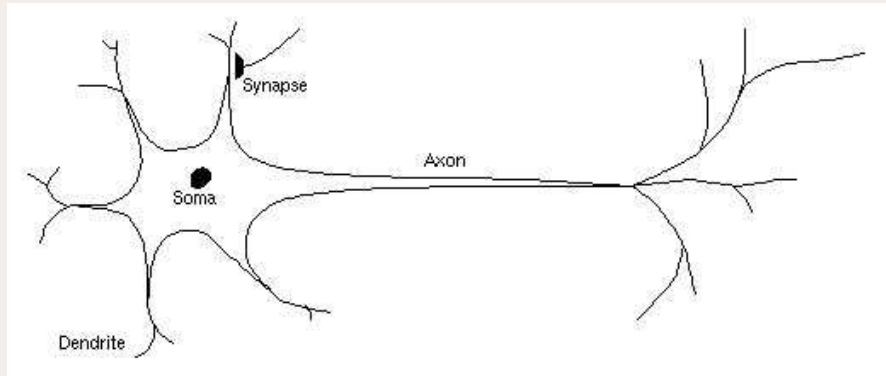
PCA

PCA is a dimensionality reduction technique that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.



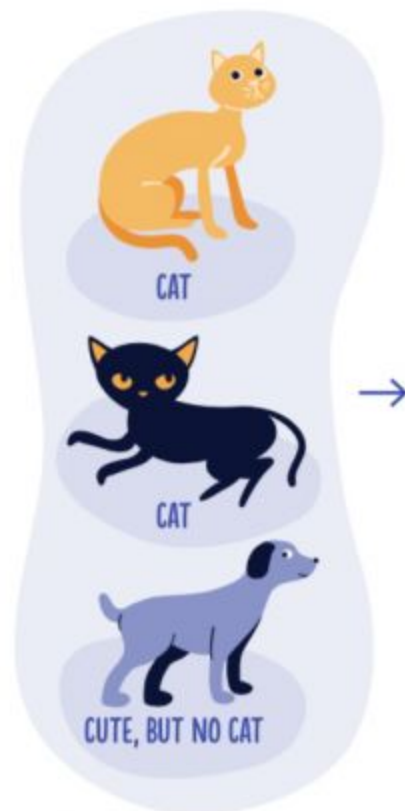
Perceptrons

- *Perceptrons* are the building blocks of *neural networks*. They are artificial models of biological neurons that simulate the task of decision-making. Perceptrons aim to solve binary classification problems given their input.
- Lore of perceptrons and biological neurons

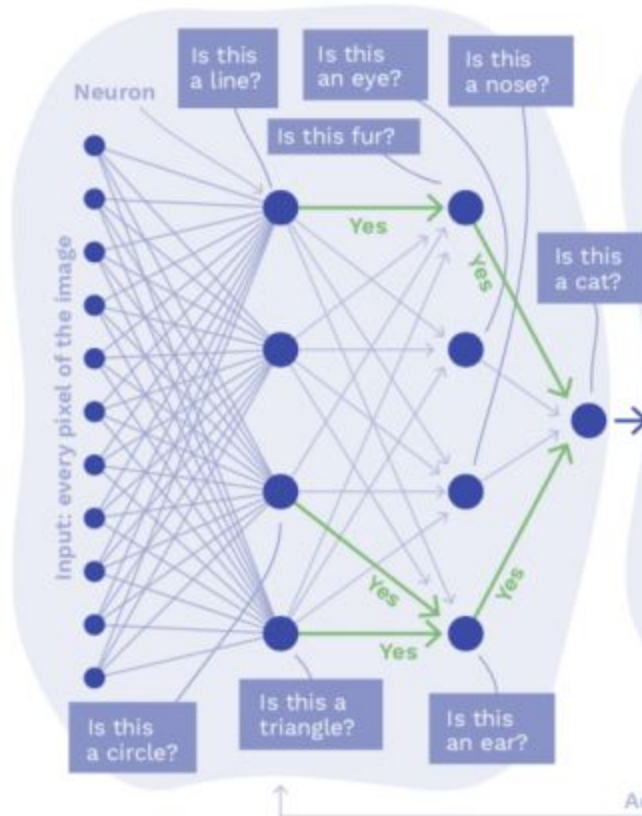




LABELS / IMAGES



ALGORITHM



PREDICTION



TRAINING







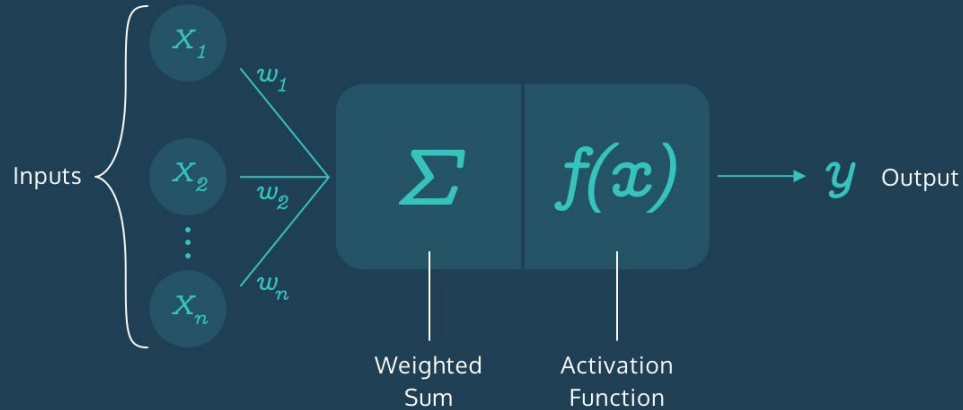
Perceptron Main Components

Perceptrons use three main components for classification:

- Input: Numerical input values correspond to features. i.e. [22, 130] could represent a person's age & weight features.
- Weights: Each feature has a weight assigned; this determines the feature's importance. i.e. In a class, homework might be weighted 30%, but a final exam 50%. Therefore the final is more important to the overall grade (output).
- Output: This is computed using inputs and weights. Output is either binary (1,0) or a value in a continuous range (70-90).



Perceptron: Weights





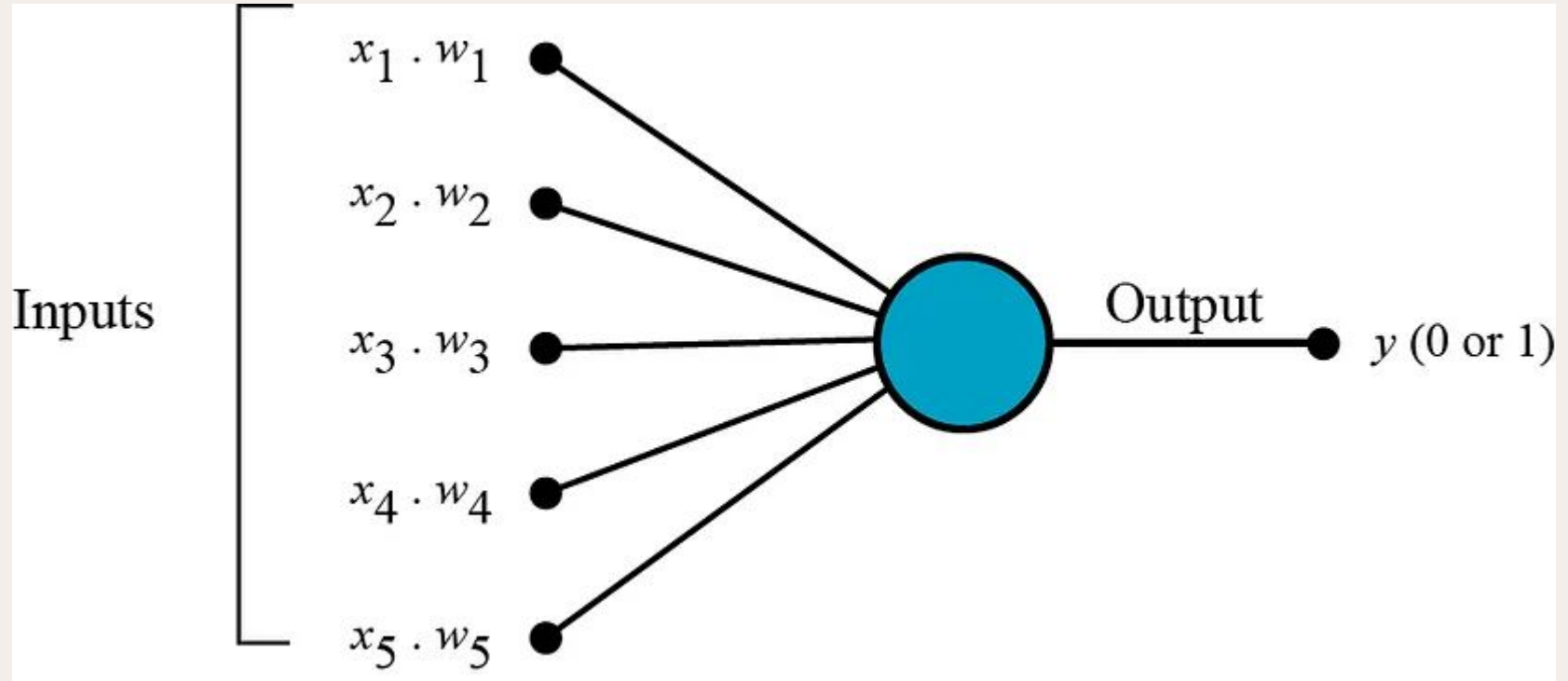
Perceptron: Weights

1. The first step in the perceptron classification process is calculating the weighted sum of the perceptron's inputs and weights.
2. To do this, multiply each input value by its respective weight and then add all of these products together. **This sum gives an appropriate representation of the inputs based on their importance.**

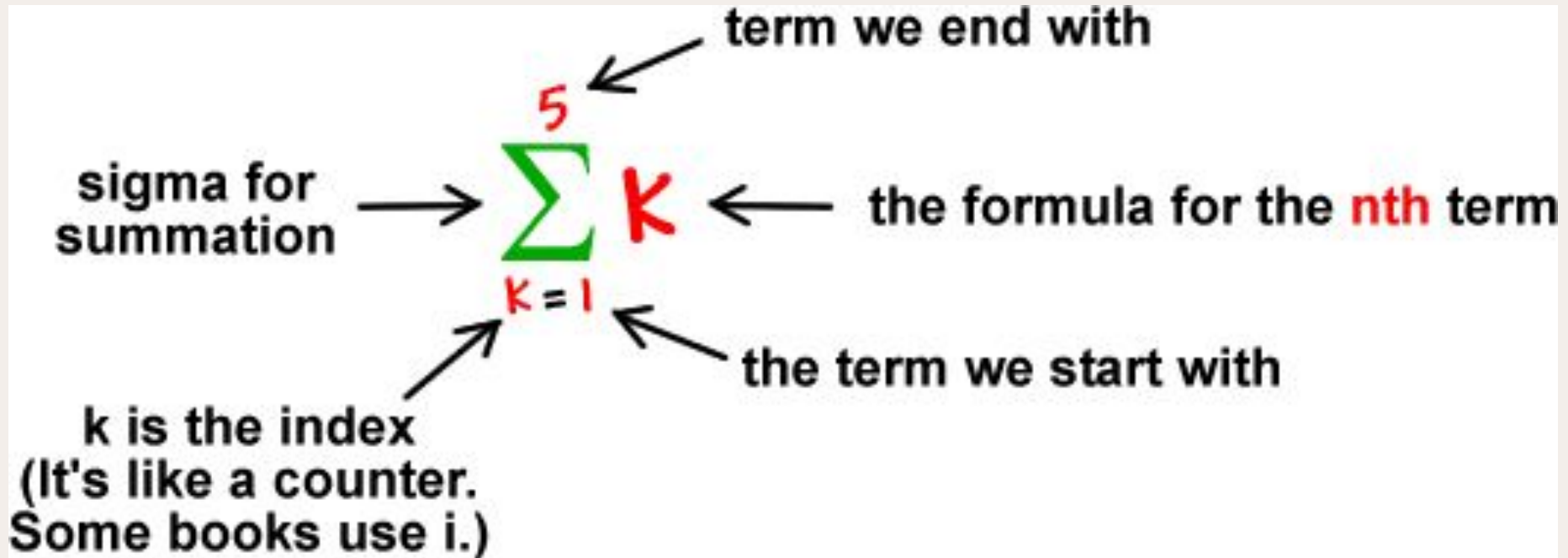
```
inputs = [x1, x2, x3]  
weights = [w1, w2, w3]
```

```
weighted_sum = x1*w1 + x2*w2 + x3*w3
```

Perceptron: Weights



Perceptron: Weighted Sum





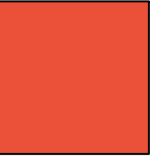
Perceptron: Optimisation Weights

The expression $\text{error} * \text{input}$ computes the adjustment to be made to the weight. If the perceptron correctly classified the example, the error is 0 and the weight is not adjusted. If the perceptron misclassified the example, the error is non-zero and the weight is adjusted proportionally to the value of $\text{error} * \text{input}$.

The weights are adjusted in the direction that reduces the error.

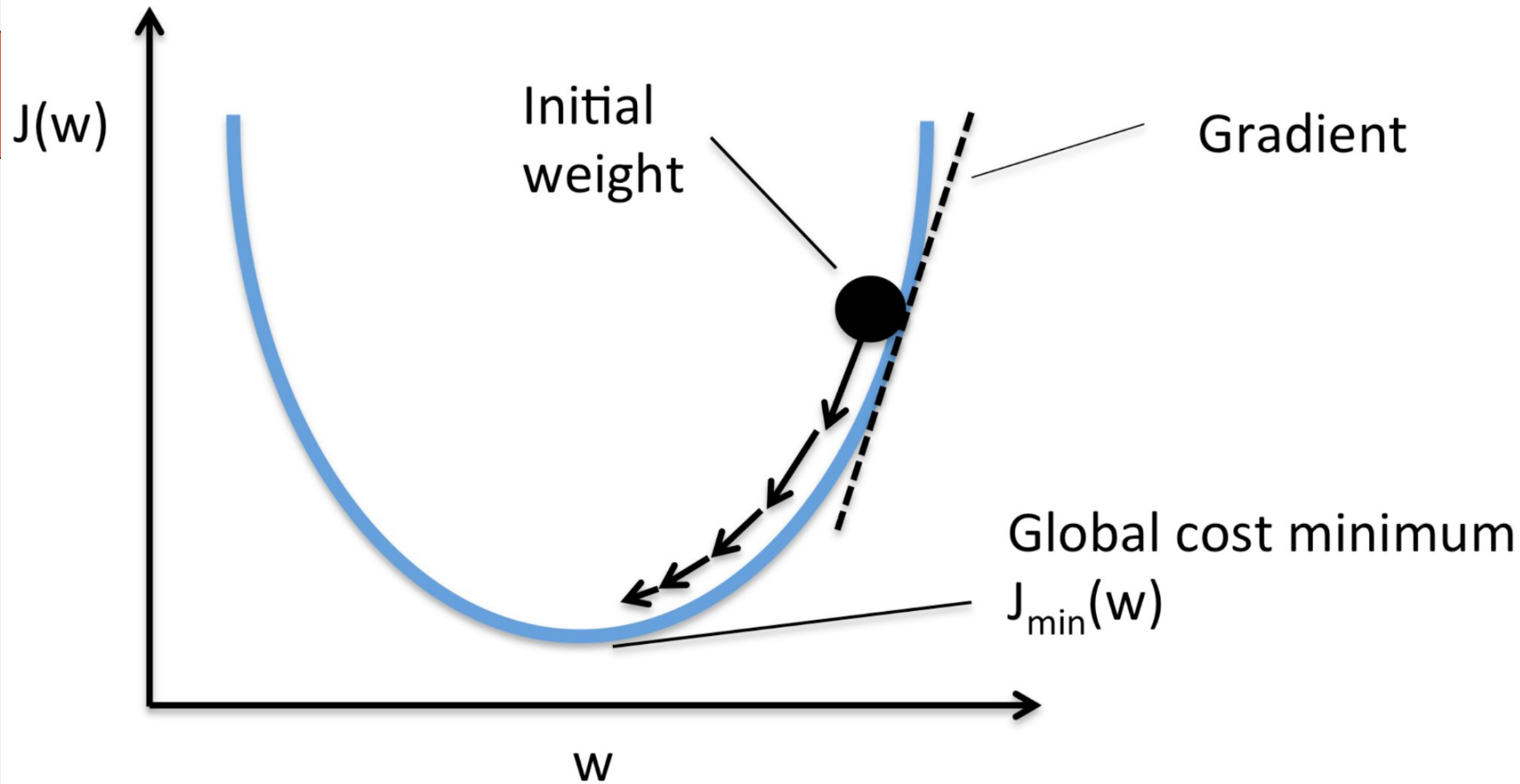
The updated weight is then given by $\text{weight} = \text{weight} + (\text{error} * \text{input})$. This process is repeated for each training example, which gradually adjusts the weights to minimize the error on the training examples. This is known as gradient descent.

In summary, the goal of optimizing perceptron weights is to find the best set of weights that minimizes the error on the training data, thereby increasing the accuracy of the perceptron's classifications. This is a fundamental concept in machine learning and forms the basis for many more complex algorithms and models.



Why are we 'optimising weights' AFTER calculating the weighted sum?

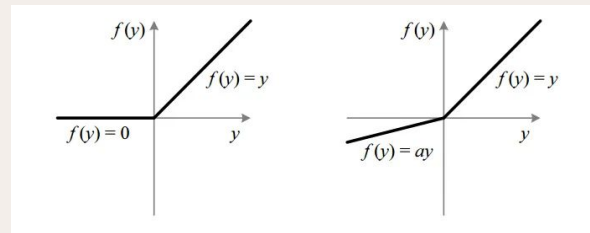
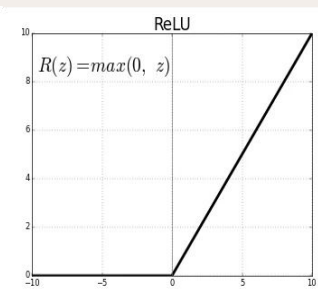
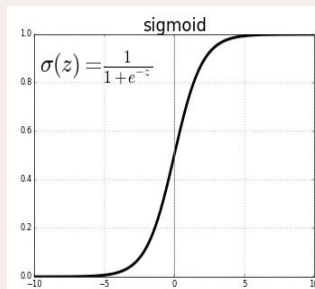
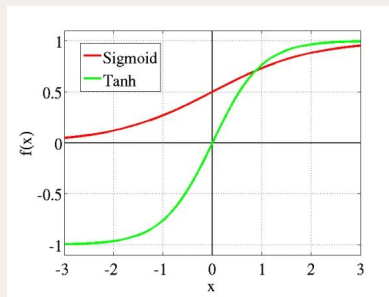
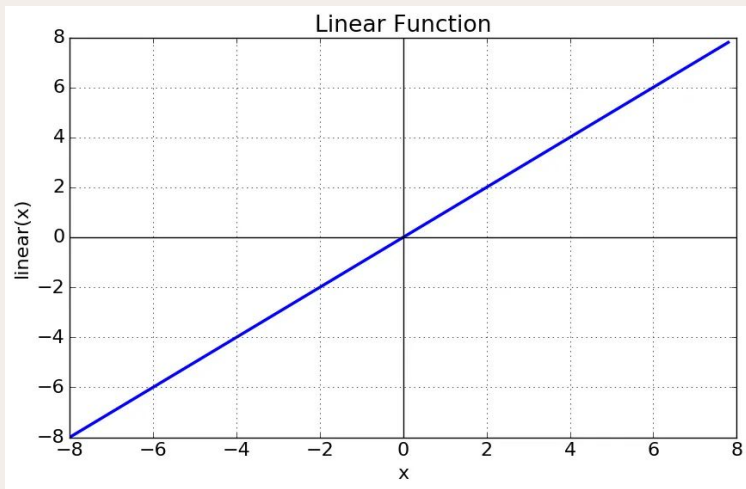
Shouldn't we take into account the error per input, BEFORE we add it all?



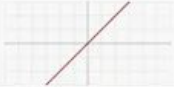










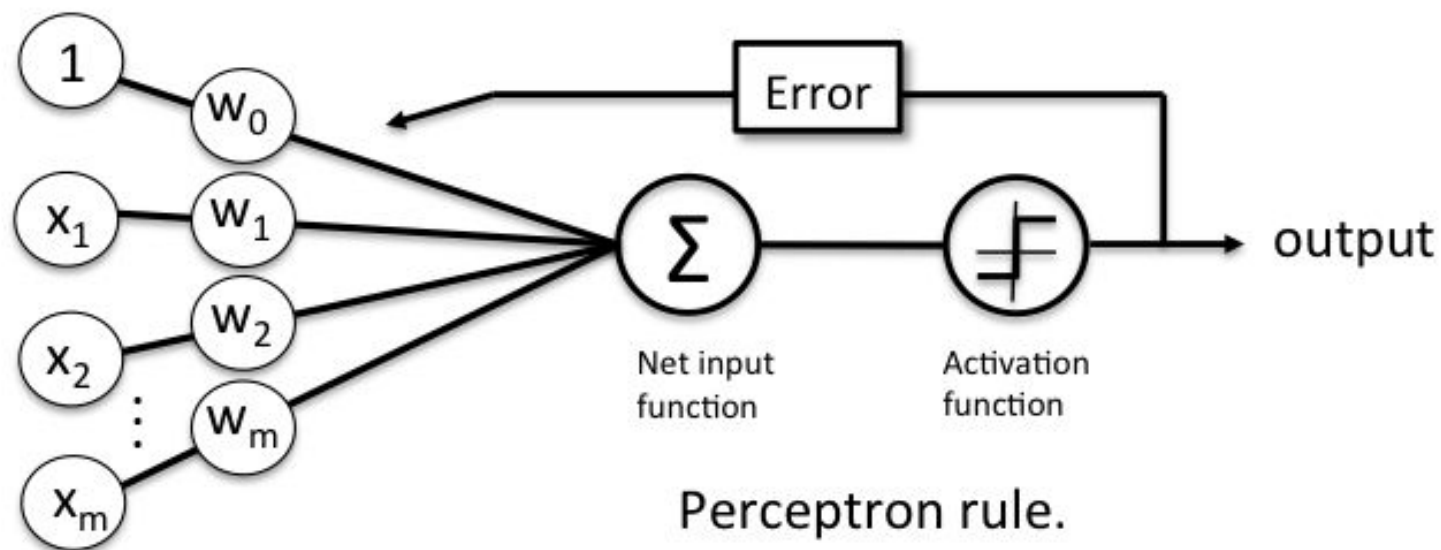
Perceptron: Activation Sum

It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).



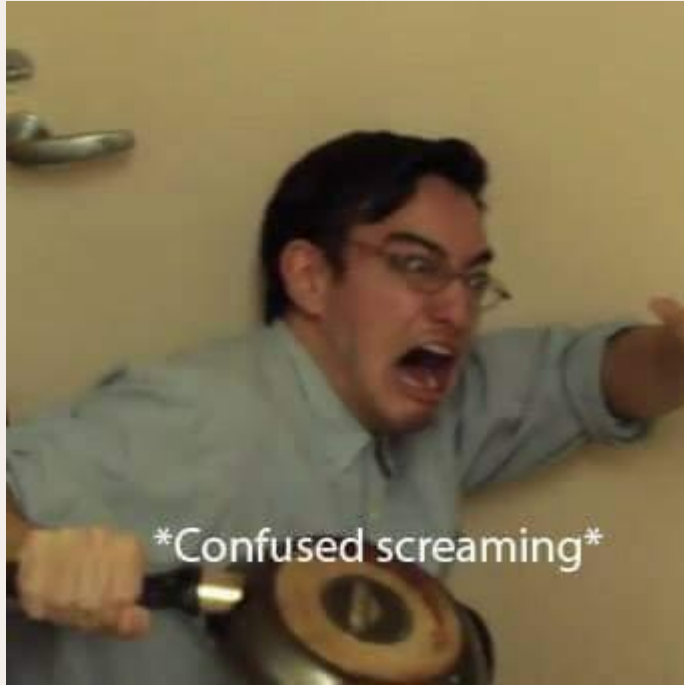


Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$





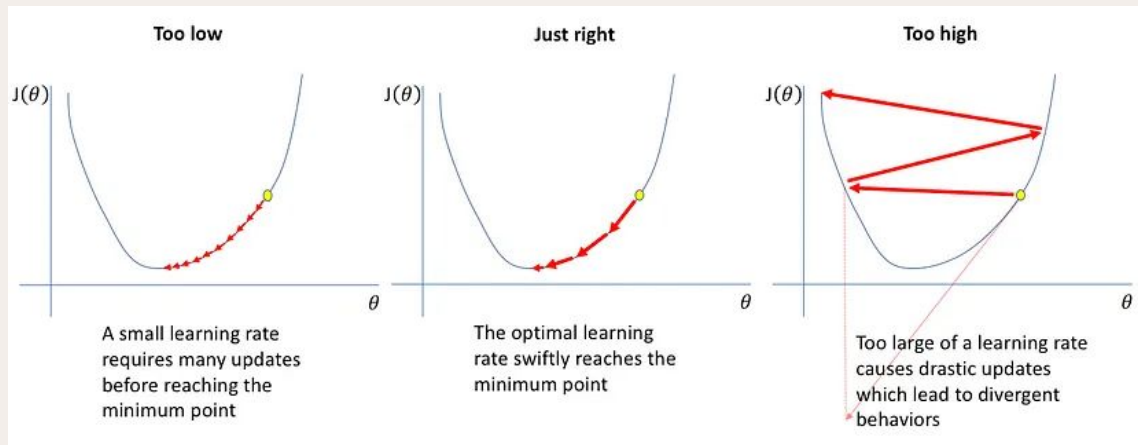
Perceptron: Differentiation?





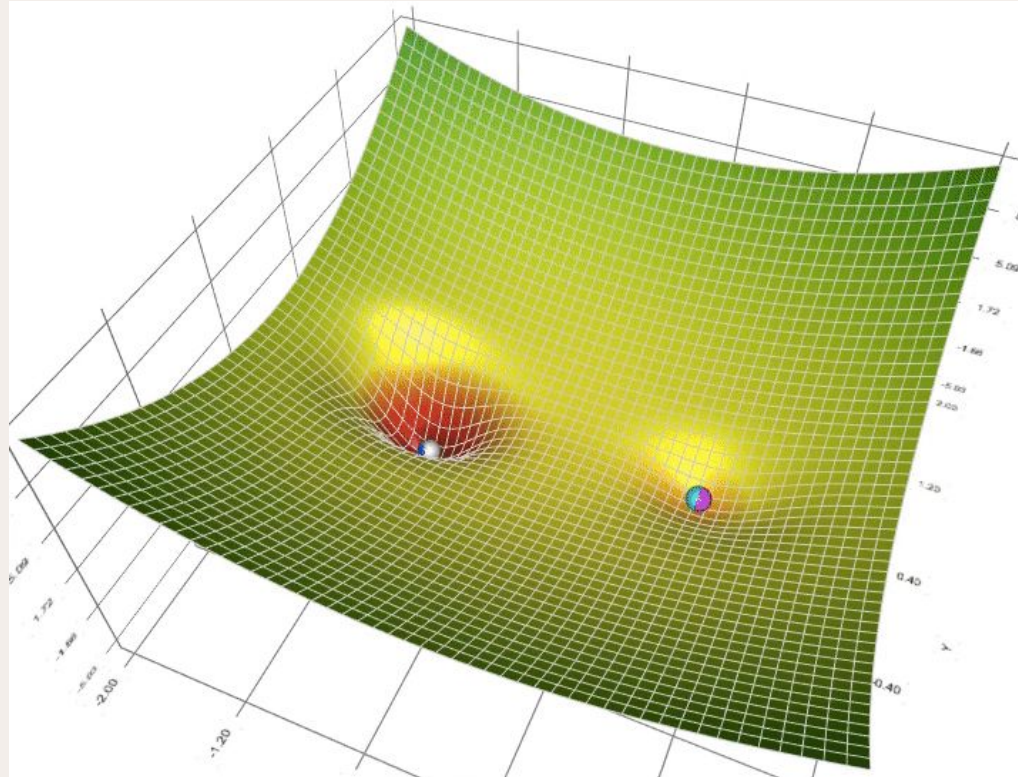
Perceptron: Differentiation?

When updating the curve, to know in which **direction** and **how much** to change or update the curve depending upon the slope.



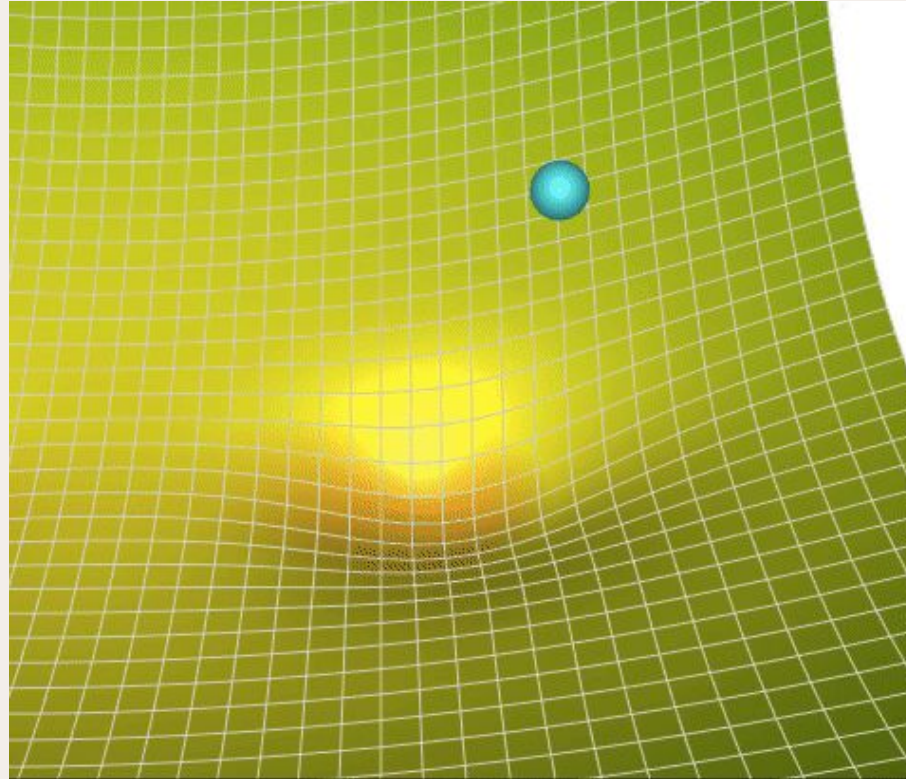


Perceptron: Differentiation?



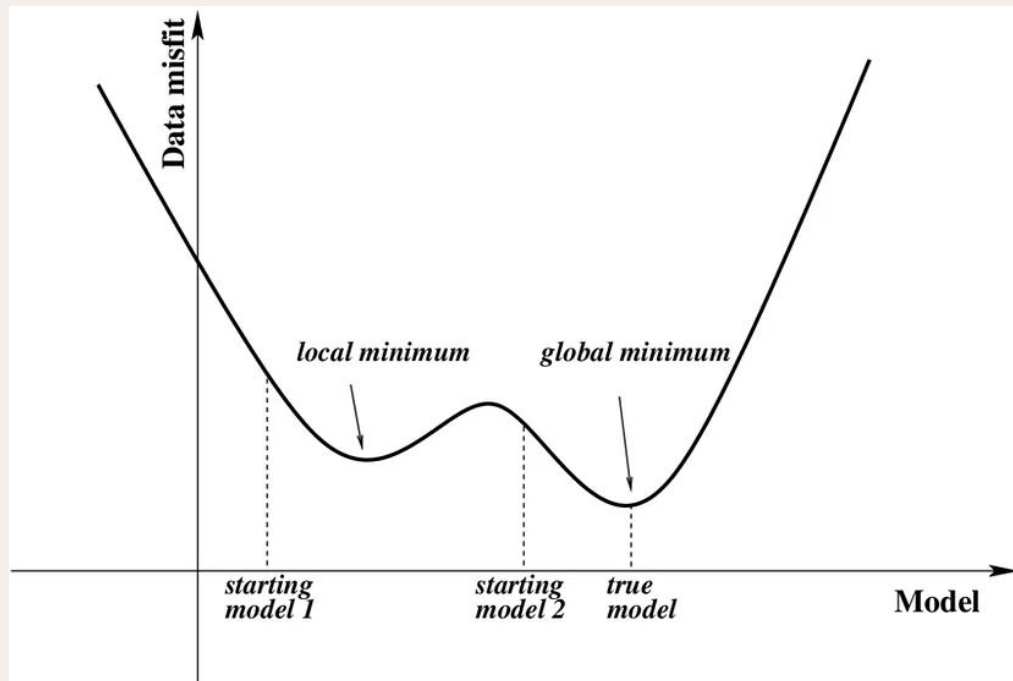
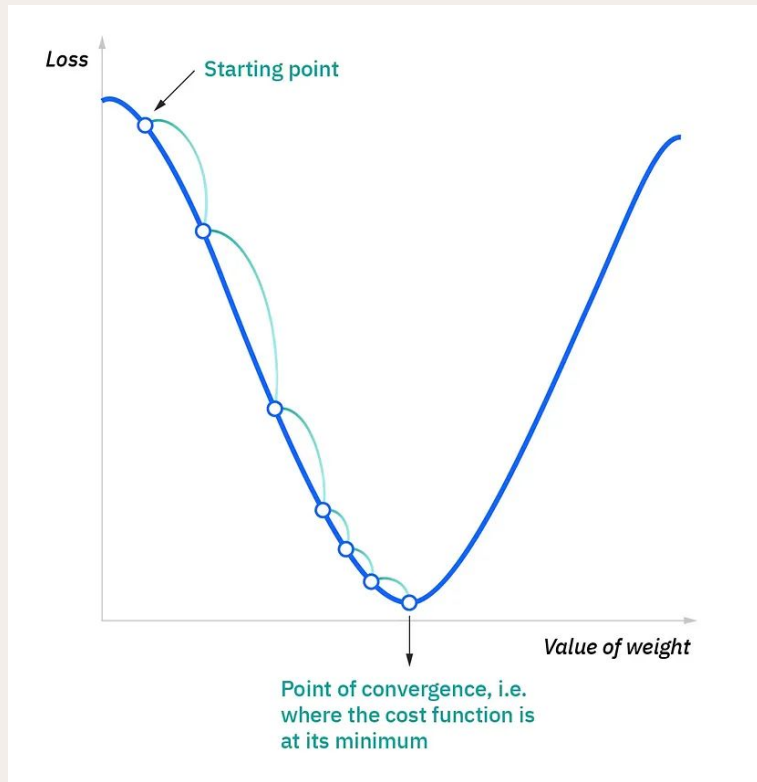


Perceptron: Differentiation?





Perceptron: Differentiation?





Perceptron Training Error

Training error is the measure of how accurate a perceptron classification is for a specific training data sample. It essentially measures “how bad” the perceptron is performing and helps determine what adjustments need to be made to the weights of that sample to increase classification accuracy.

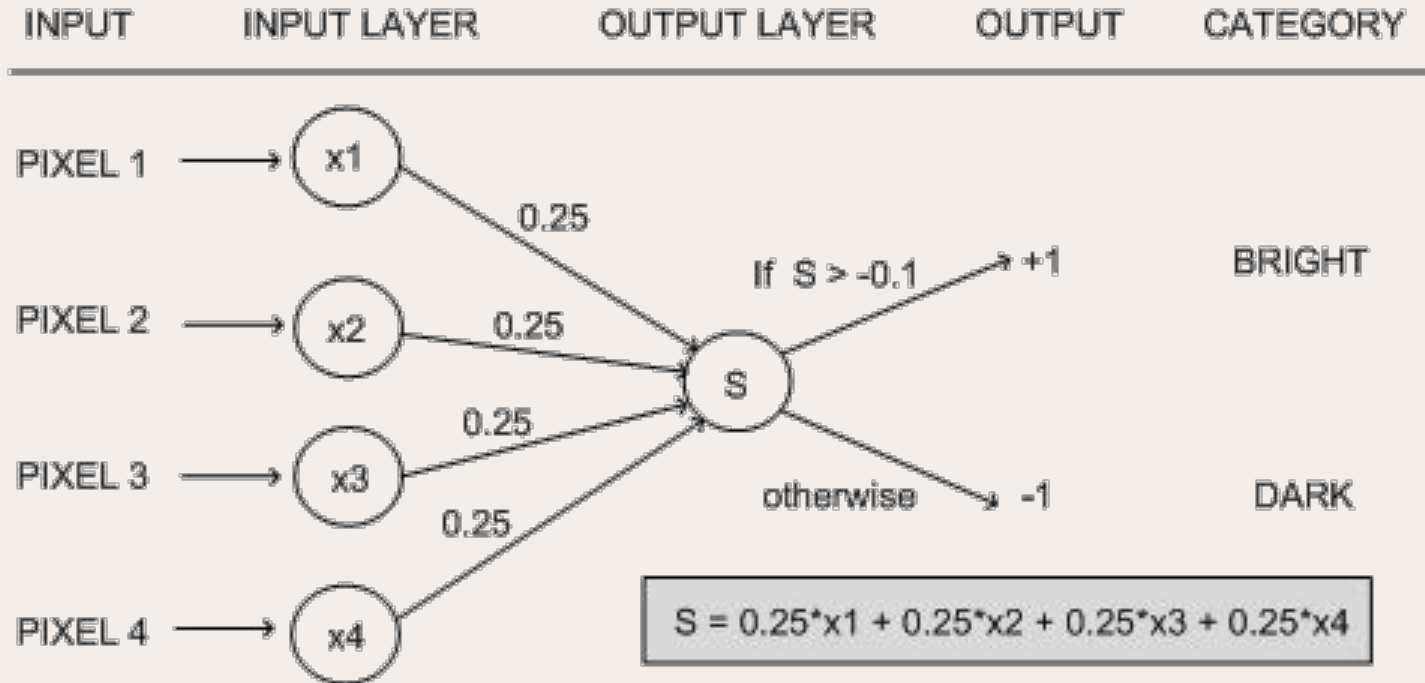
Training error = actual label - predicted label

*obviously

Actual Label	Predicted Label	Training Error
+1	+1	0
+1	-1	2
-1	-1	0
-1	+1	-2



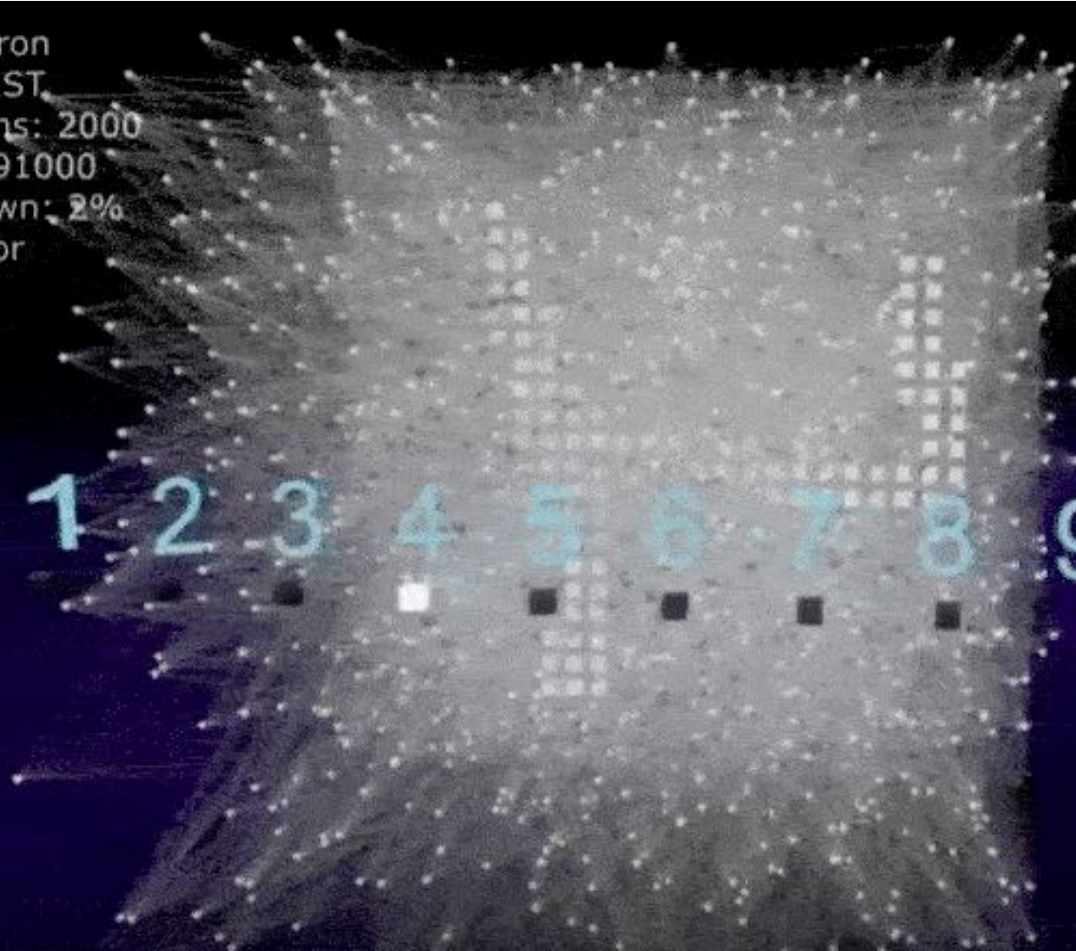
Perceptron: Goodbye Habibi





Type: Perceptron
Data Set: MNIST
Hidden Neurons: 2000
Synapses: 1191000
Synapses shown: 2%
Learning: WCor

0 1 2 3 4 5 6 7 8 9



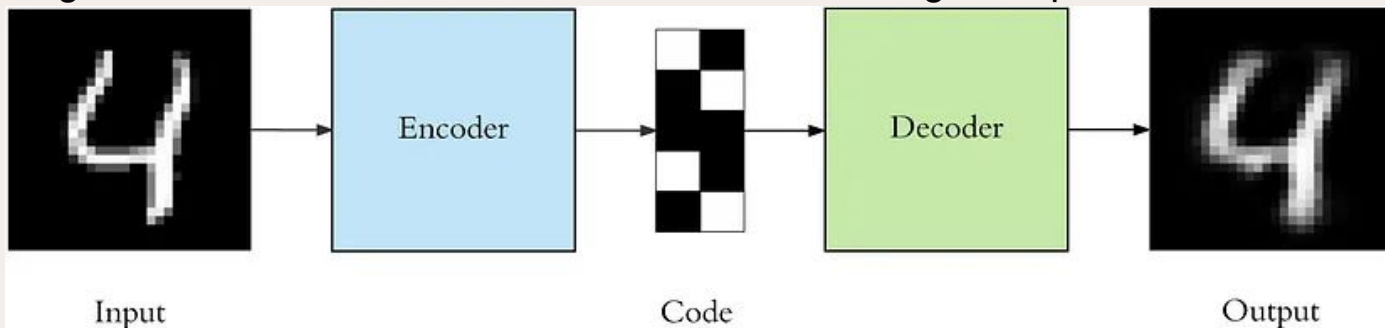


Autoencoders

An autoencoder is a type of neural network architecture designed to efficiently compress (encode) input data down to its essential features, then reconstruct (decode) the original input from this compressed representation.

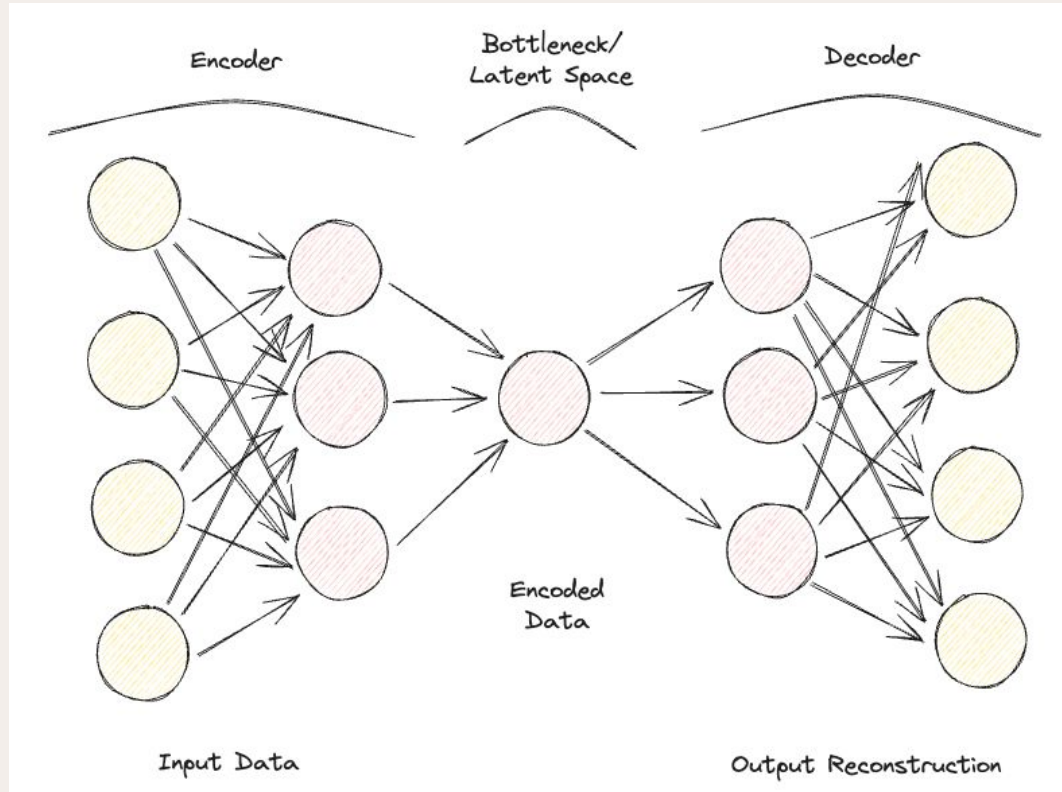
Here's a simple breakdown of how autoencoders work:

1. **Encoding:** The input data is compressed into a lower-dimensional code by the encoder. This code is a representation that retains the essential features of the input data.
2. **Decoding:** The decoder then attempts to reconstruct the original input data from this encoded representation. The goal is to minimize the difference between the original input and the reconstruction.





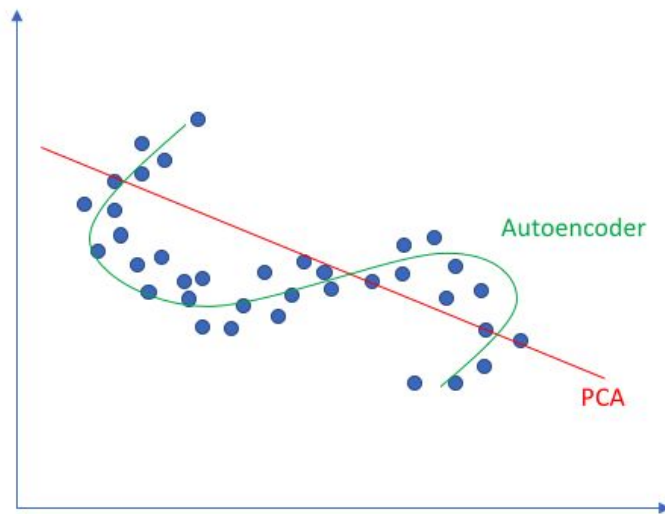
Autoencoders





Autoencoders

Linear vs nonlinear dimensionality reduction

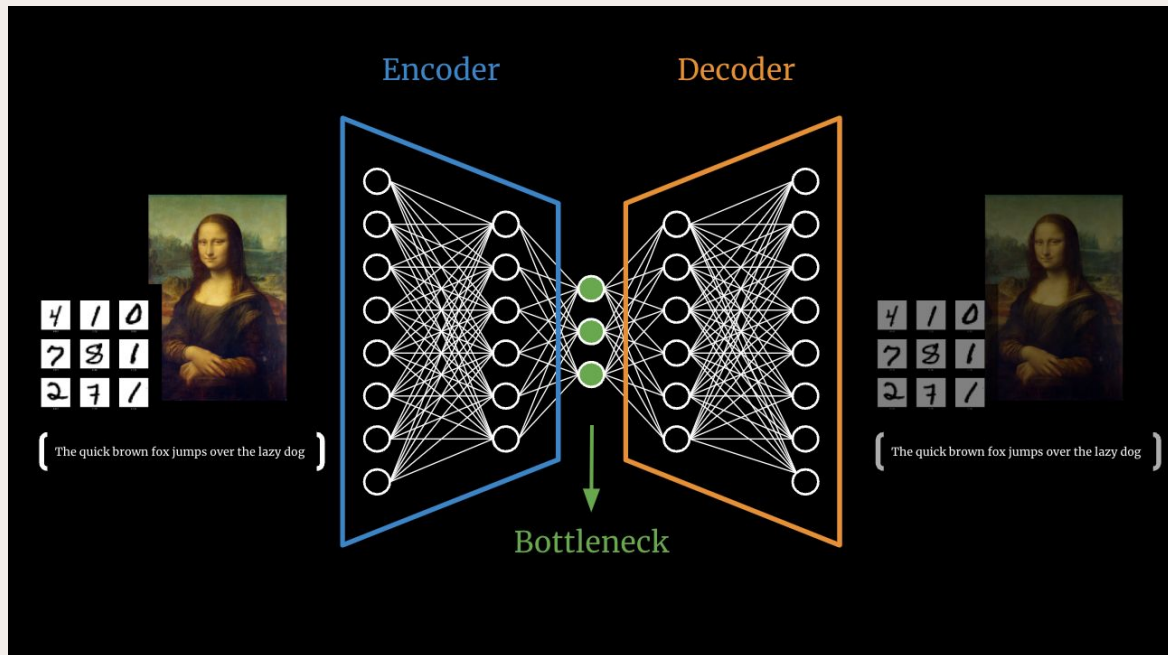




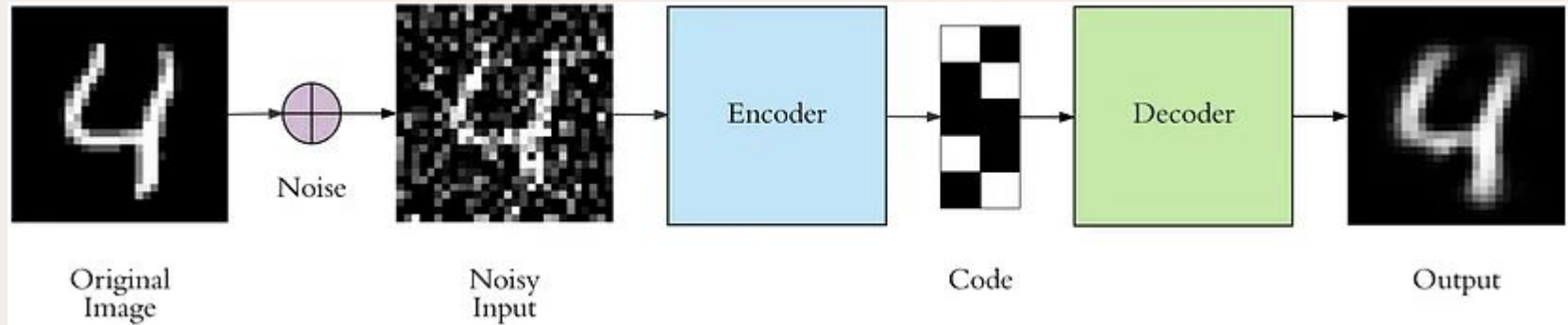
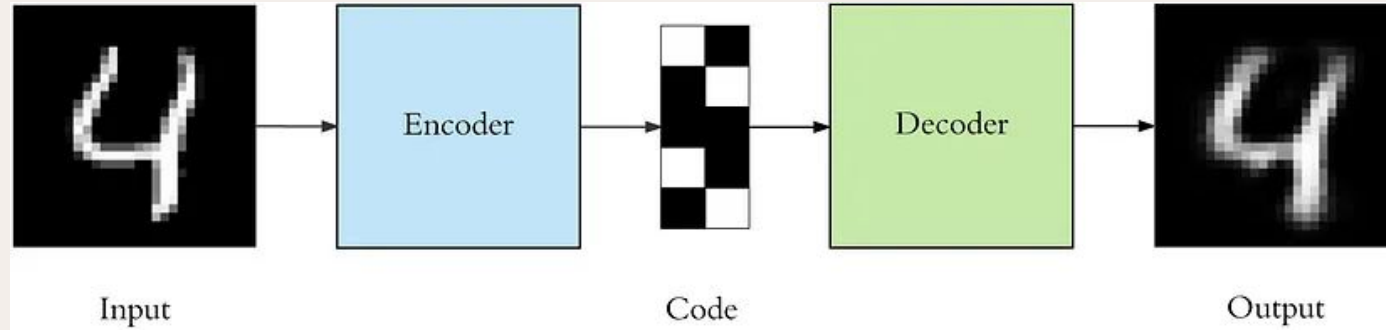
Autoencoders: Types

Over the years, different types of Autoencoders have been developed:

- Undercomplete Autoencoder
- Sparse Autoencoder
- Denoising Autoencoder
- Convolutional Autoencoder



Autoencoders: Denoising





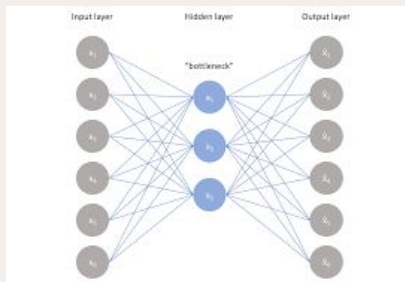
Autoencoders: Regularization

- We can take an unlabeled dataset and frame it as a supervised learning problem tasked with outputting a reconstruction of the original input. This network can be trained by minimizing the reconstruction error, which measures the differences between our original input and the consequent reconstruction.
- The bottleneck is a key attribute of our network design; without the presence of an information bottleneck, our network could easily learn to simply memorize the input values by passing these values along through the network.
- For most cases, this involves constructing a loss function where one term encourages our model to be sensitive to the inputs (ie. reconstruction loss) and a second term discourages memorization/overfitting (ie. an added regularizer).
- ***Loss function = Reconstruction + Regularizer***

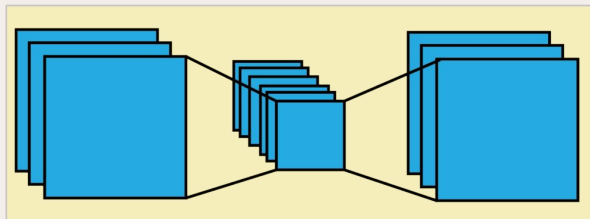
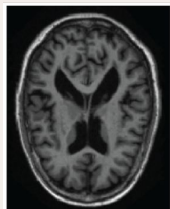


Non-regularized

- Undercomplete



NOISY
INPUT



COMPRESSION/
CLEANING

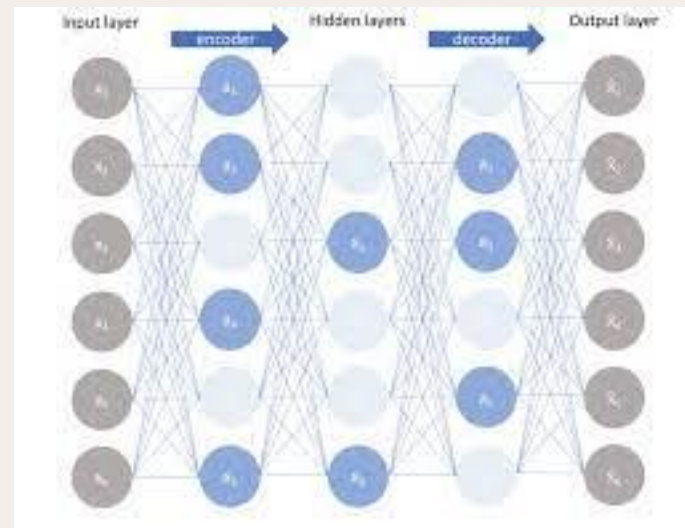
DECOMPRESSION

DENOISED
OUTPUT



Regularized

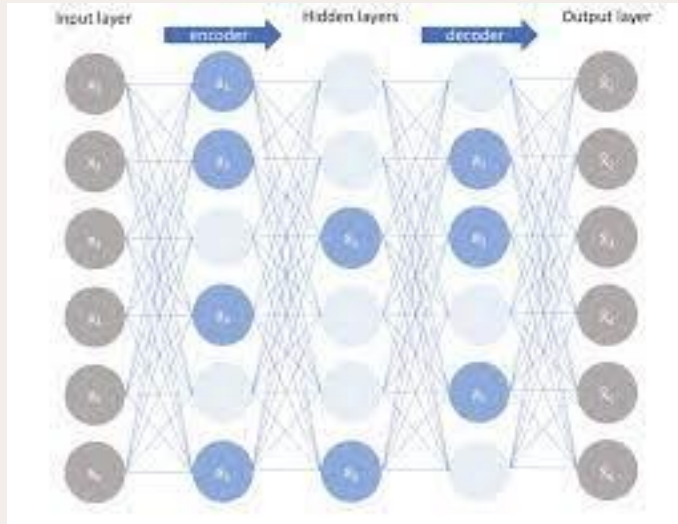
- Sparse Autoencoder
- Denoising Autoencoder





Regularized

- Sparse Autoencoder



- We don't necessarily have to reduce the dimensions of the bottleneck, but we use a loss function that tries to penalize the model from using all its neurons in the different hidden layers.
- This penalty is commonly referred to as a sparsity function, and it's quite different from traditional regularization techniques since it doesn't focus on penalizing the size of the weights but the **number of nodes activated**.
- In this way, different nodes could specialize for different input types and be activated/deactivated depending on the specifics of the input data.

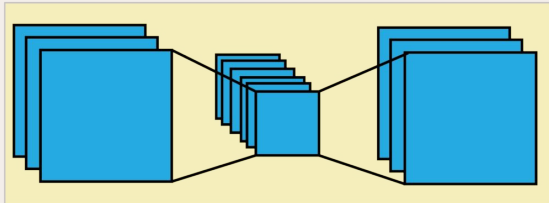
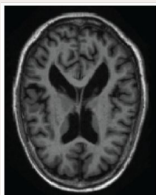


Regularized

- Denoising Autoencoder

- With Denoising Autoencoders, the input and output of the model are no longer the same. For example, the model could be fed some low-resolution corrupted images and work for the output to improve the quality of the images.
- We'll construct our loss function such that we penalize activations within a layer. For any given observation, we'll encourage our network to learn an encoding and decoding which only relies on activating a small number of neurons. It's worth noting that this is a different approach towards regularization, as we normally regularize the weights of a network, not the activations.
- In order to assess the performance of the model and improve it over time, we would then need to have some form of labeled clean image to compare with the model prediction.

NOISY
INPUT



COMPRESSION/
CLEANING

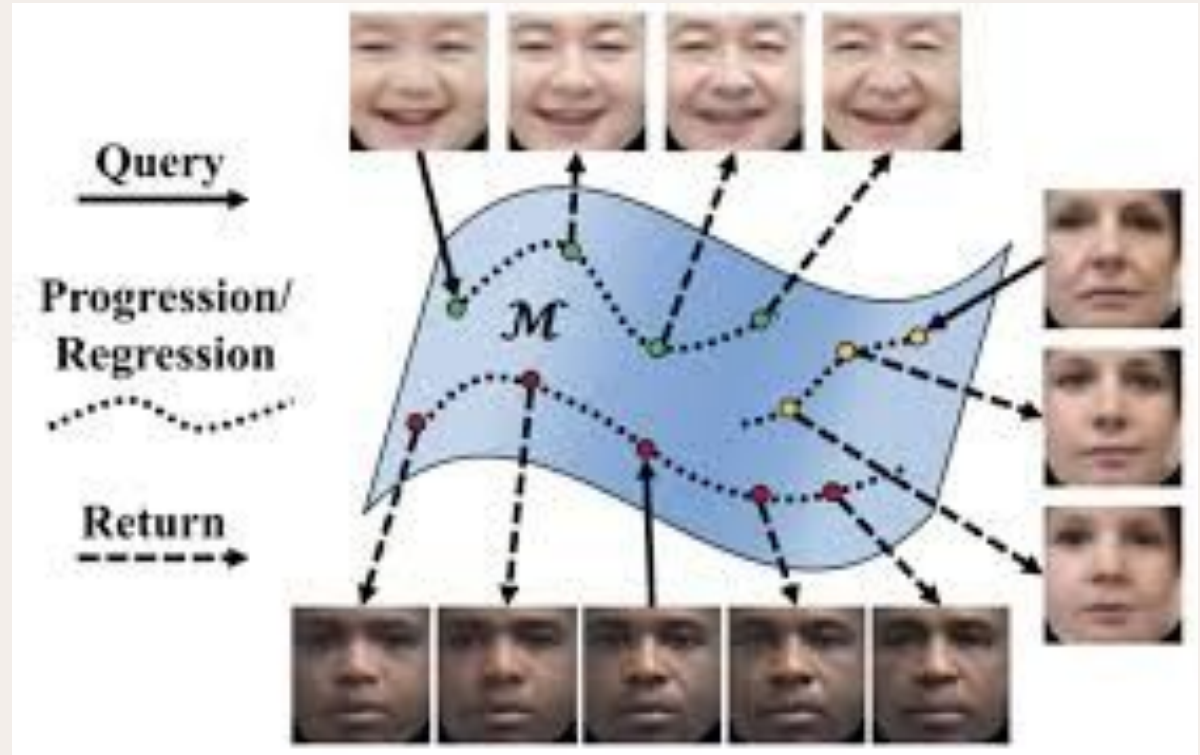
DECOMPRESSION

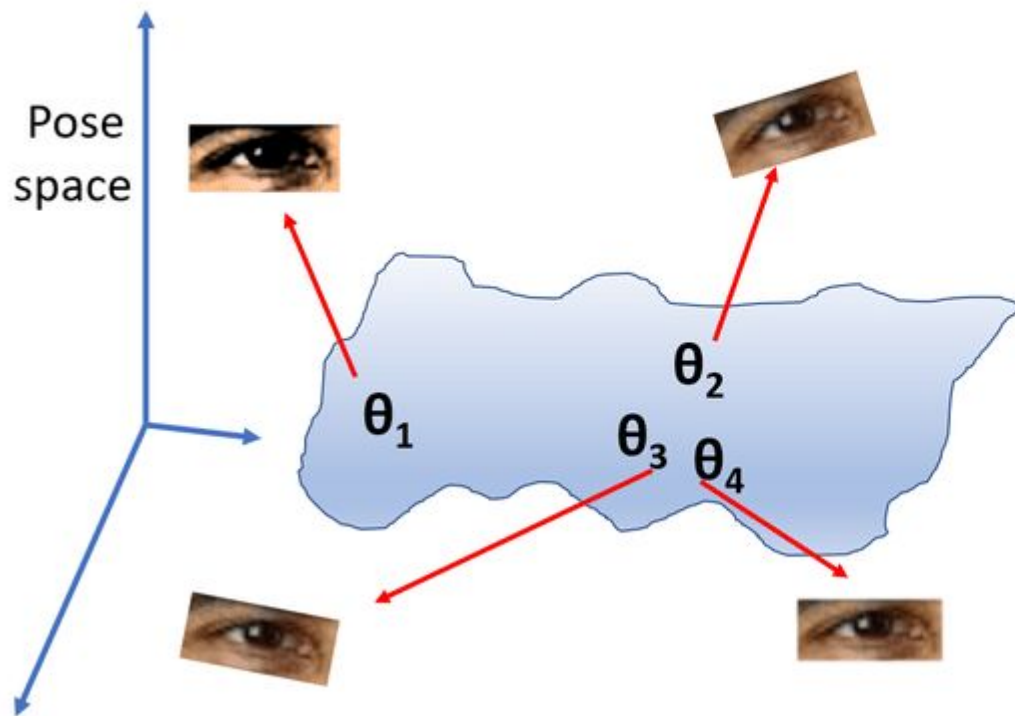
DENOISED
OUTPUT



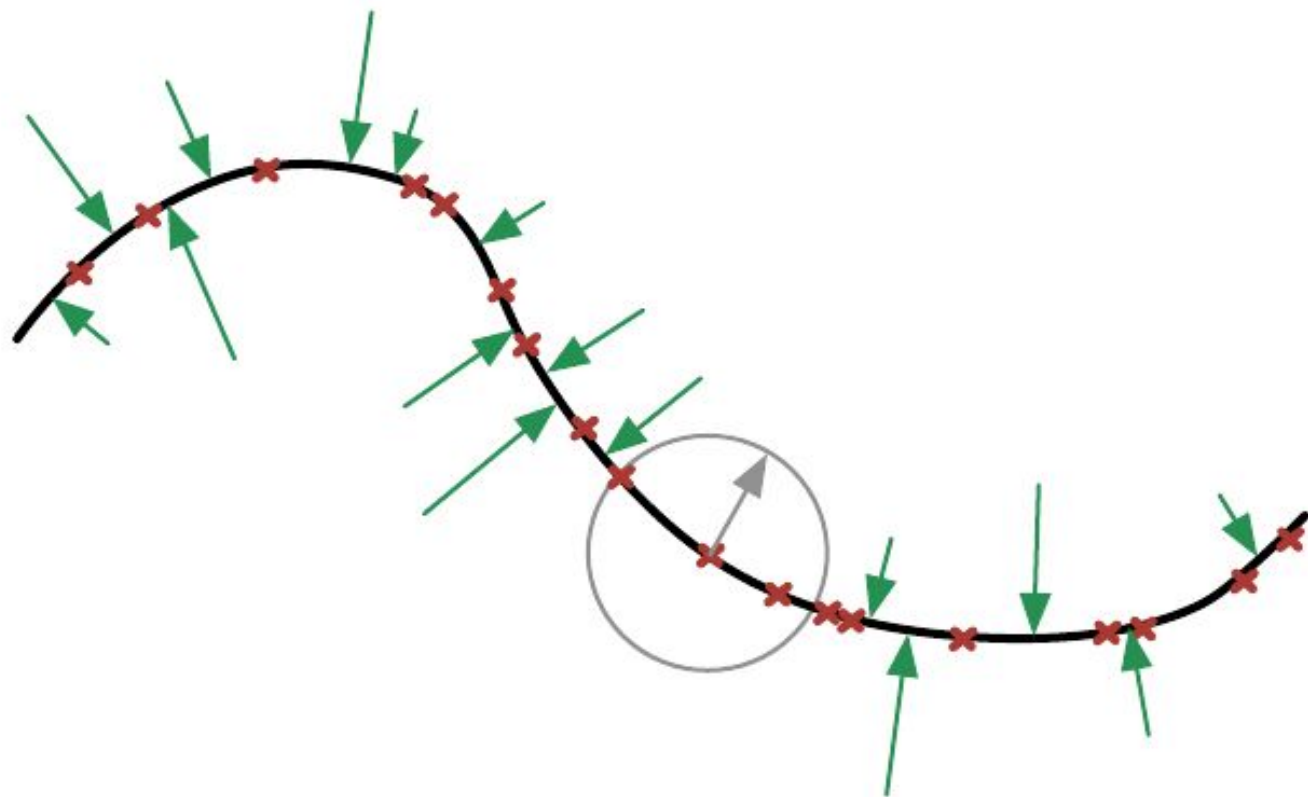
Regularized

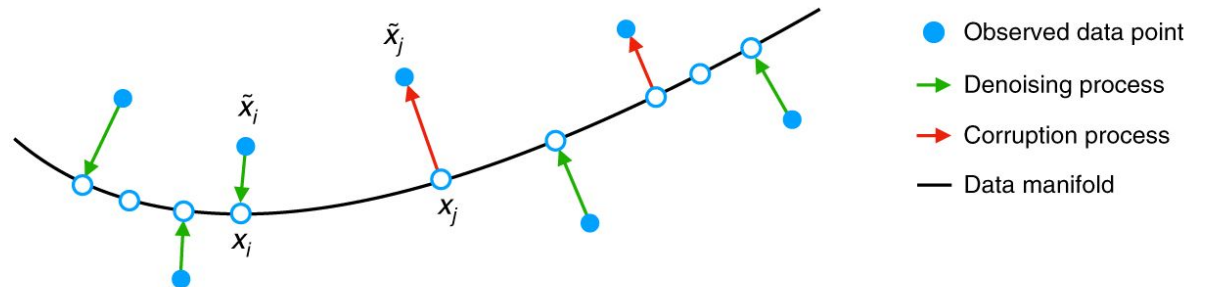
- Denoising
Autoencoder:
Manifolds





Human Eye Image Manifold. Each point in the manifold corresponds to particular set of pose parameters θ . The Manifold is topologically an f -dimensional object embedded in R^n .



a**b**