Generative Artificial Intelligence

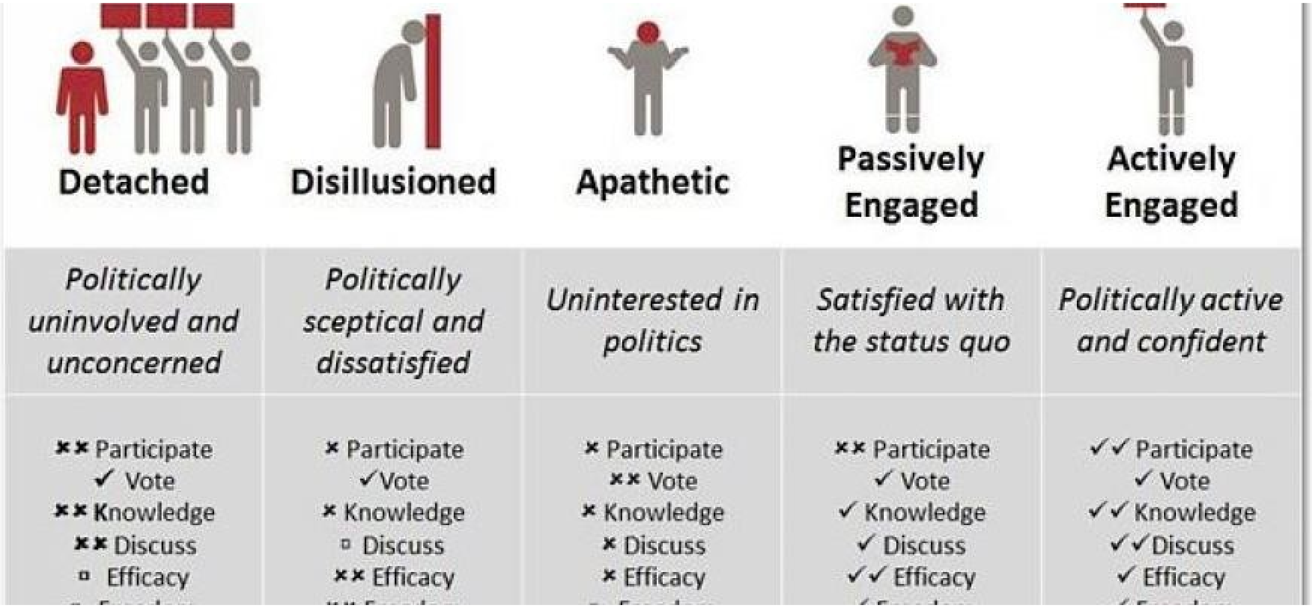
Module Two: Generative and Discriminative ML

Discriminative AI

- Traditional ML and Deep Learning are discriminative
- That means that they are trained on data to do one of two things
- Make decisions
  - Classify data into categories (or make predictions)
  - The model *discriminates* between possible choices
  - Often used to re-engineer a logical process that was once used but is no longer available
  - Or to emulate the result of a process that cannot be represented logically
  - Eg. Replicate a banker’s successful “gut feeling” for credit risk
  - Often experts make good decisions but they can’t describe what they do so we emulate it with machine learning
  - But we need to select the features of parameters to use to make the decision

Discriminative AI

- Find relationships among data data points
  - Clustering data – find data that is near to each other using some measure of nearness
  - We have to determine which features will result in a useful clustering
  - The clustering is often a basis for making decisions

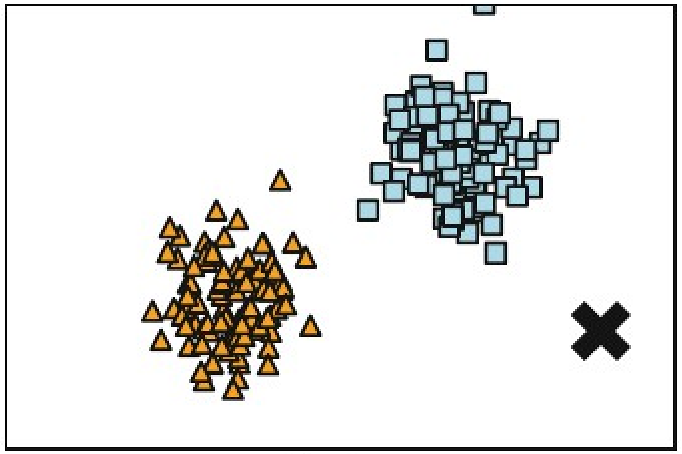


Discriminative Probability Models

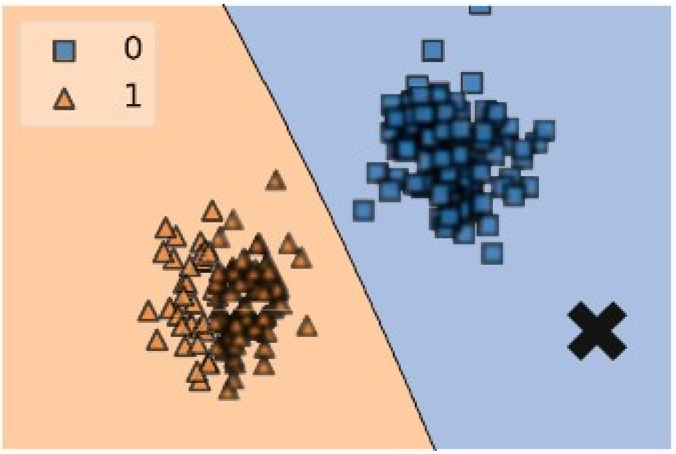
- Traditional ML produce probability distributions
  - However they lack context
  - Probability of classification is taken as a measure of certainty but this is limited to the probability distribution produces on the basis of the training data
  - It doesn’t take into account the center of mass of the probability distribution in context
  - A data point may be in one class based on the model but be quite different from the other members of the class (ie. the classification is spurious)
  - We might catch this mistake if we understood the probability distribution of the whole space across all the features.
- Example
  - Chairs might be classified in a distribution of furniture features based on appearance, size, the presence of a seat and other factors
  - A hologram of chair would be misclassified as a chair because it differs on other features

Discriminative Probability Models

- In the image below the X would be classified as blue
  - But it is distant from both of the centers of mass of classifications
  - This fact should increase the uncertainty of classifying it as blue



Data



$p(y|\mathbf{x})$

Regression

- The blue dots are measurements
  - There appears to be a linear relation among the points
  - If it were an exact linear relationship, no need for ML
  - The line is our model that predicts the y value given an an x value
- ML regression algorithm is intended to find which line that is the best fit to the data
  - What “best fit” means will be defined later

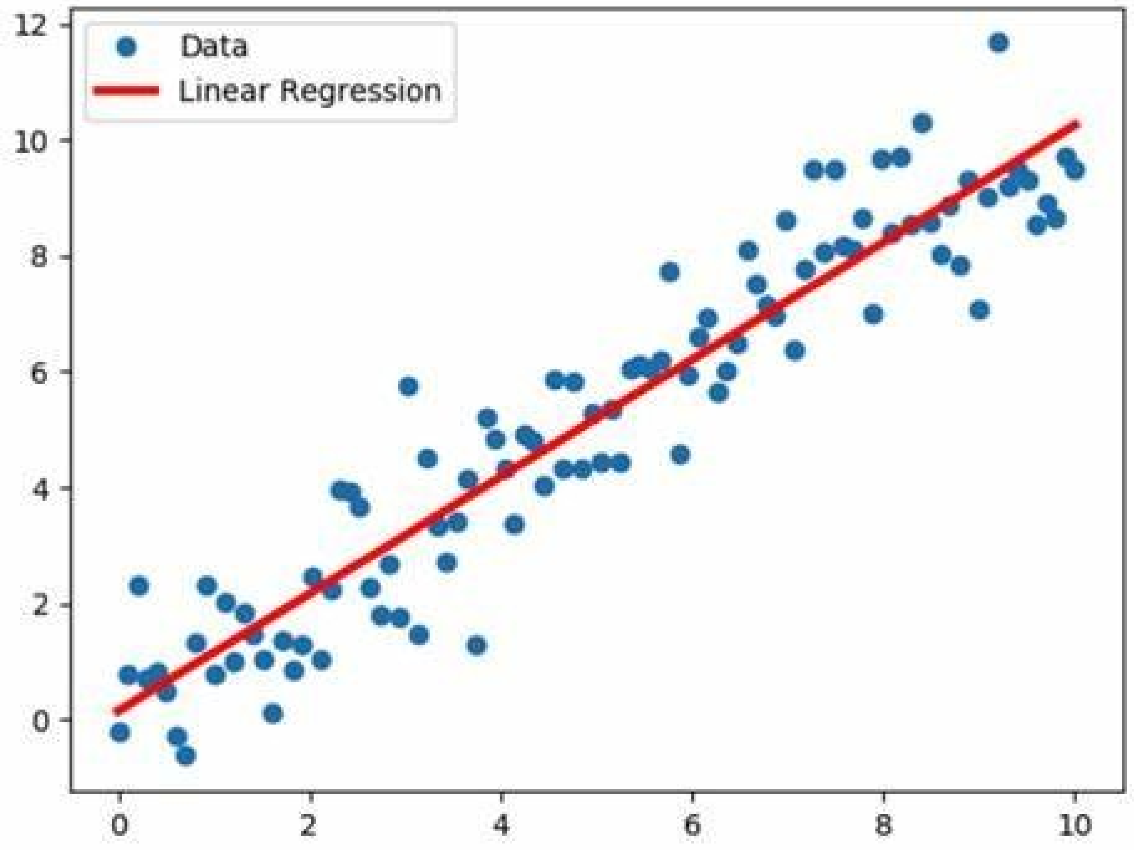


Image Credit: https://www.researchgate.net/figure/Linear-Regression-model-sample-illustration_fig3_333457161

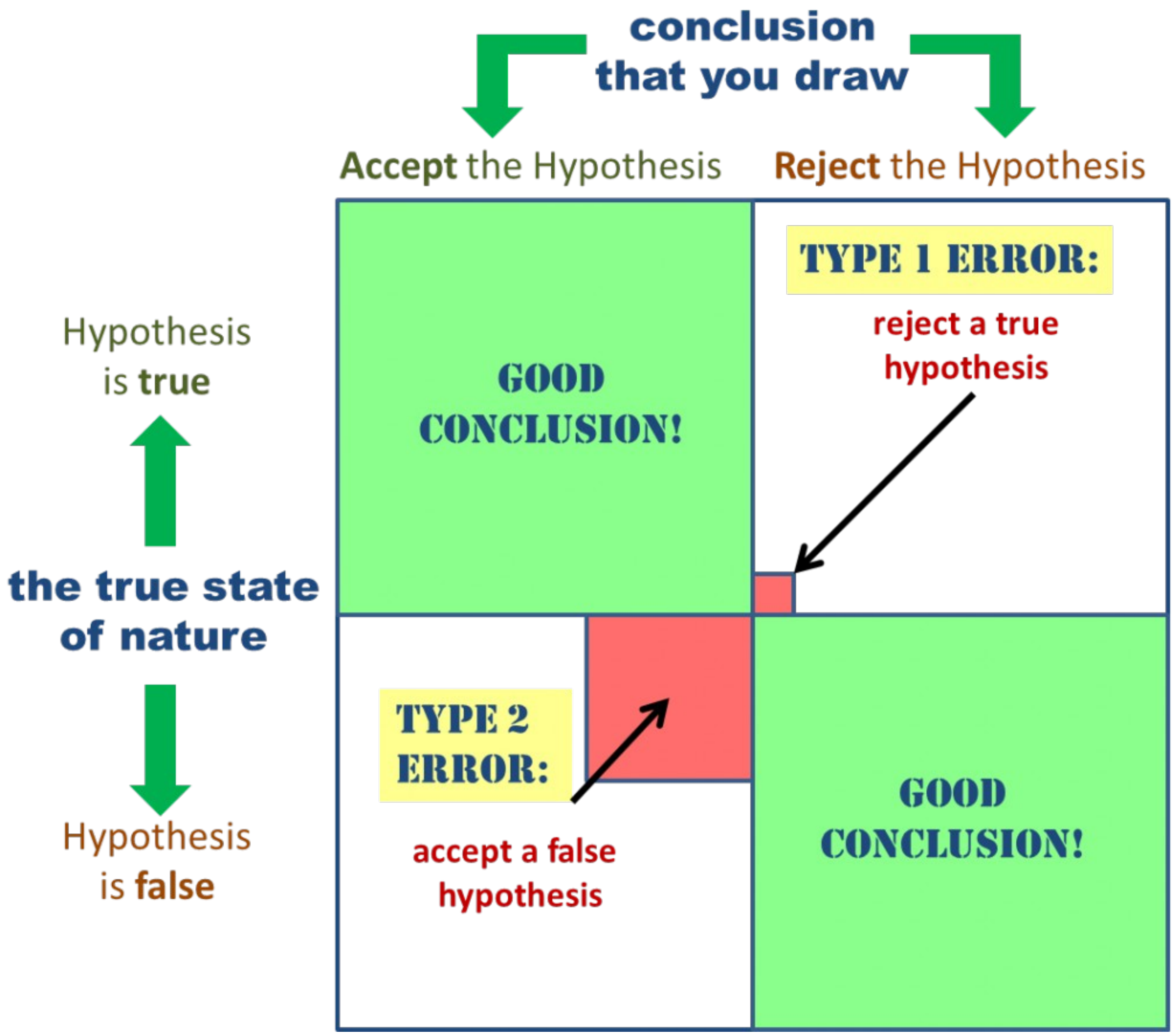
Correlation

- Correlation describes an association between variables
  - When one variable changes, so does the other
  - Correlation is a an observed relationship between variables
  - When variables change together, they are said to exhibit “co-variance”
  - Co-variance does **not** imply any underlying relationship between the variables
- Correlation is strictly observational
  - The ML regressor-predictor hypothesis is a *description* of the relationship between variables that co-vary
  - Our co-variant ML model has predictive value and but has no explanatory power
  - We may have no idea why our variables are co-variant, but we can still measure the relationship

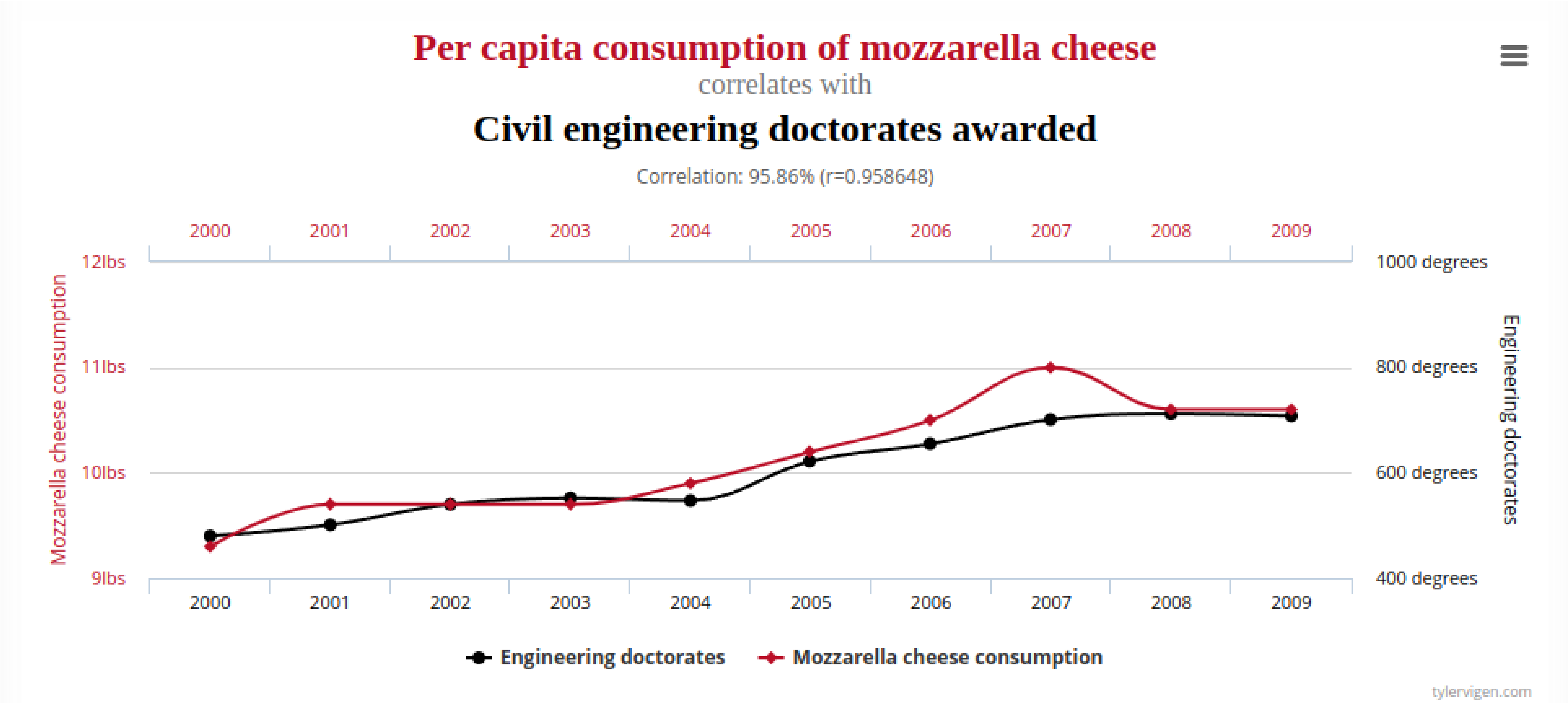
Causation

- Implies a cause and effect relationship
  - Changes in the independent variable *always* produce changes in the dependent variable
  - Causation *always* implies correlation
  - There may be a proposed explanation as to what the causation mechanism is
- We cannot infer causation from correlation
  - The variables may both be influenced by a confounding third variable
    - Heat stroke and ice cream sales may be correlated
    - But they are both affected independently by the outdoor temperature
    - Temperature affects both ice cream sales and the number of cases of heat stroke independently
    - To assume ice cream sales cause heat stroke is not a valid hypothesis
    - Also called a Type I error or a false positive or failing to reject the null hypothesis

Type I and II Errors



Spurious Correlations



Feature Engineering

- Features or parameters are the properties of our data points
- Feature engineering is selecting and modifying features
  - There are potentially an infinite number of features
  - In ML we have to select only a few to build a model on
- We can also think of each possible feature as a clustering of the data points
  - The problem of feature extraction and selection is a significant one
  - It directly impacts the usability and reality of the model we create
- If we have a large enough training set and enough features
  - We have probability distributions for all the features over data that closely resembles the population from which it is drawn

Generative AI

- Assuming we have a large enough training set and enough features
  - Each data point maps to a very large feature vector, maybe billions of features
  - Assume the data points have labels
  - Then we can generate a new data point from a set of features that are similar to other data points with that label
- Synthetic data
  - The use of GAI to create new data points that are similar to existing data points, x-rays of cancer tumors for example, is called synthetic data
  - Very useful for training models when not enough training data exists

People that Don’t Exist

- GAI can produce faces of people that don’t exist by generating them from the model

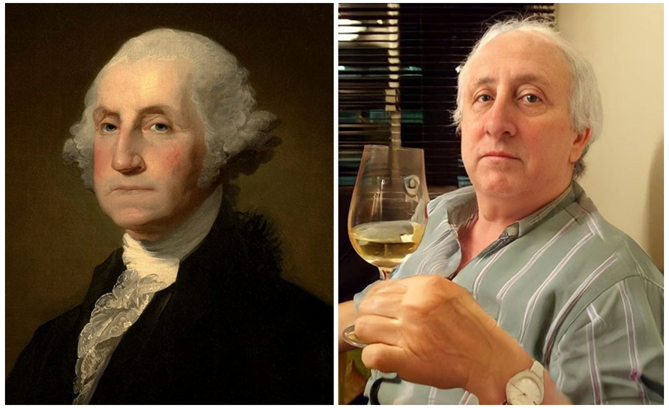
Generative AI

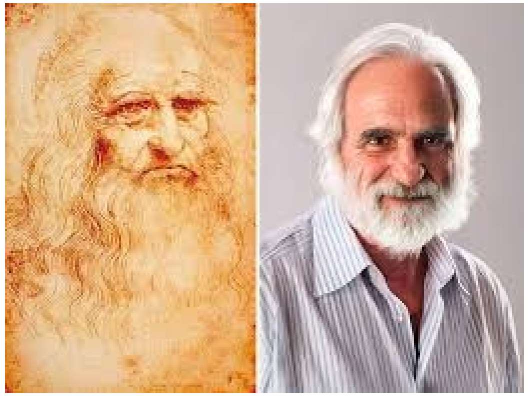
- This allows for predictive capabilities
  - Given enough data in an input structure (often a sequence like a string of word) it allows for prediction of what comes next
  - This can also be used to generate text string or other sequences
- Transformational capabilities
  - Given two different types of data (images of faces and paintings by Picasso) it is possible to create a new data point with features common to both
  - “Your selfie as if it was painted by Picasso”
  - Create a photo of George Washington drinking wine

Domain Transfer

- Converting depictions of historical people to photographs





Generative AI

- We pointed out earlier that generative AI required three things to happen
  - Access to enough data to build really big models on petabytes of data and billions of features
  - Enough compute power to train the models
  - A theoretical approach that enables us to build the models
- The rest of this module will focus on the theory that enables generative AI
  - GAI builds on Neural Networks
  - Nns build on traditional ML
  - Details on specific traditional ML techniques are provided in the ancillary materials

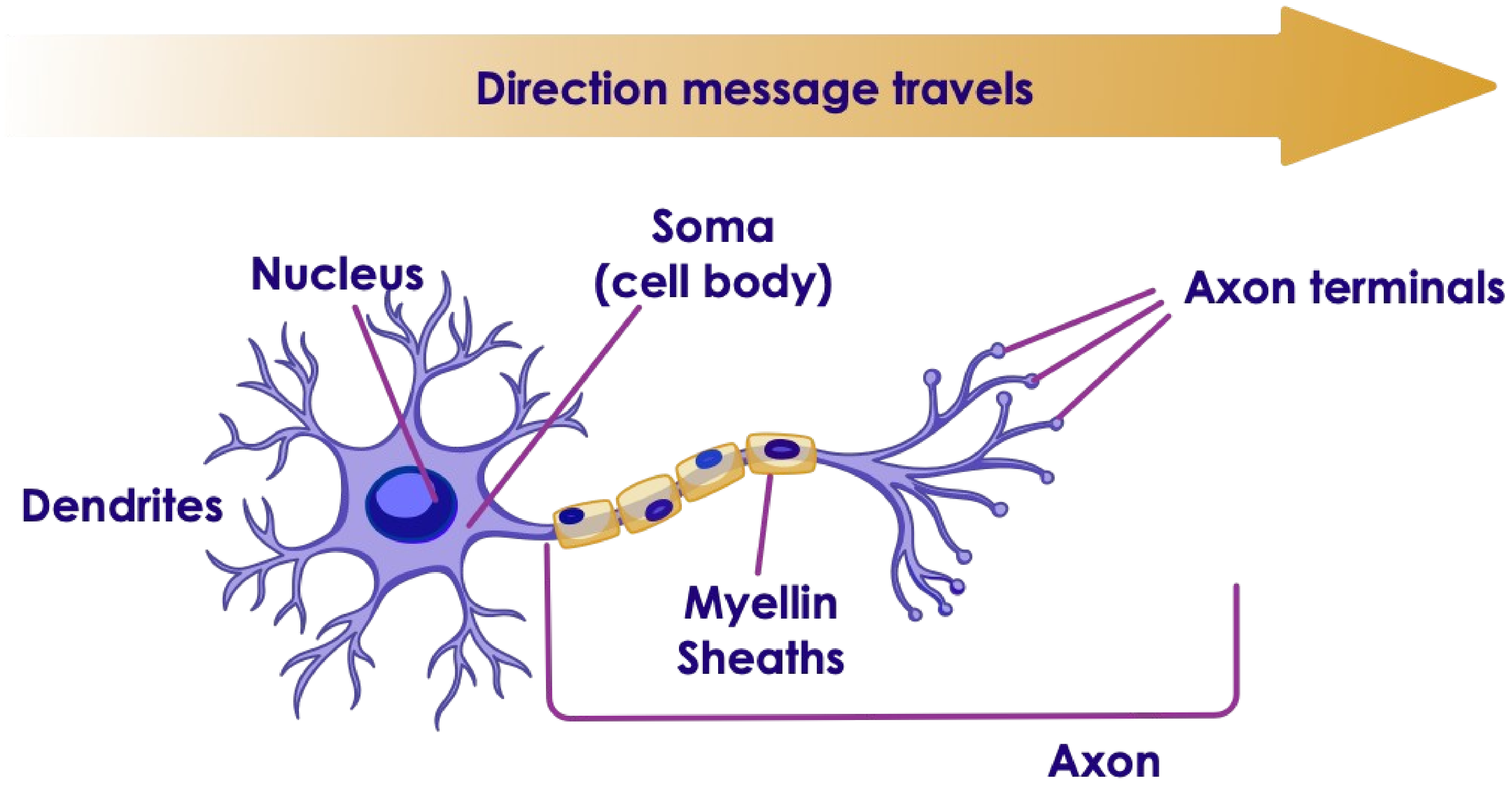
Artificial Neural Networks (ANN)

- ANNs are at the core of Deep Learning
  - Powerful, scalable and can solve complex problems like classifying billions of images (Google Images)
- ANNs were inspired by neurons in human brain
  - Original theoretical formulation was in the 1940s
  - First working models in the 1960s
  - Very limited in power and could not simulate actual neurons
- But early work provided an architectural model for ML
  - No longer intended to simulate human neurons

History of ANN

- 1943: McCulloch Pitts Neural model
- 1962: Frank Rosenblatt invented the Perceptron:
- 1969: Marvin Minsky's paper killed interest in ANNs.
  - He demonstrated the ANNs can't solve a simple XOR problem
- 1970s: No work on ANNs - first AI winter
- 1980s: Some revival in ANNs (new models + training techniques)
- 1986: Rumelhart introduce the backpropagation training algorithm.
- 1990s: Second AI winter (Methods like SVMs were producing better results)
  - Limited by processing power
- 2010s: Huge revival in AI after some promising results

Prototype Neuron Architecture



The Perceptron Model

- Basic architectural model for a neural net
  - Simplest type of a Feed Forward neural network
- Efficient classifier on linearly separable data
  - Linearly separable means a hyperplane can be drawn that totally divides the data into the label classes
  - Stochastic classifiers perform better on non-linearly separable data

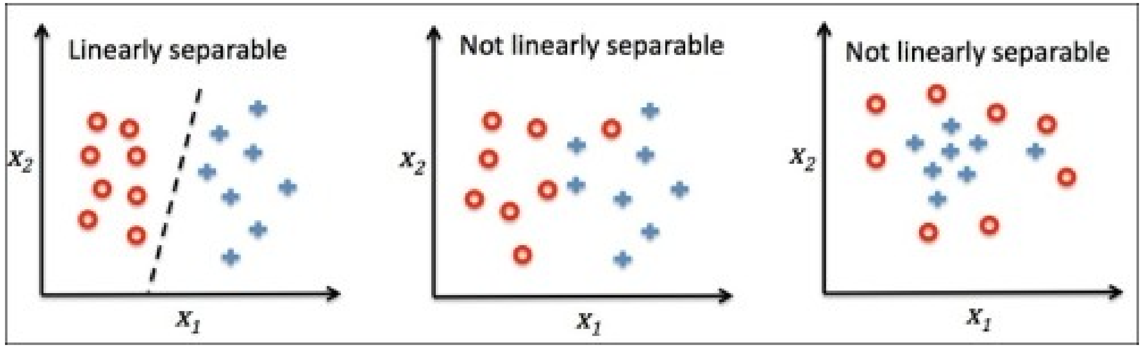
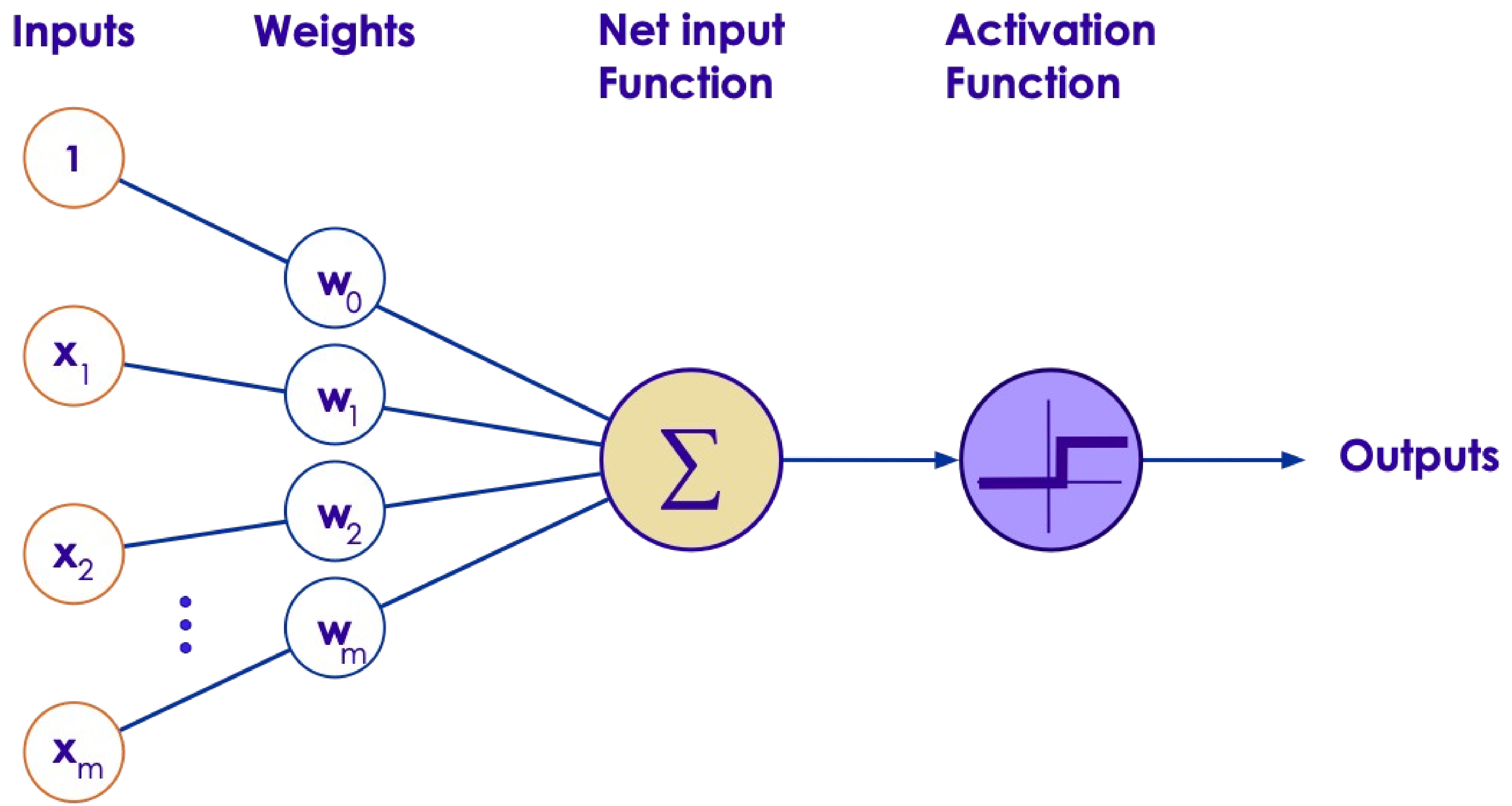


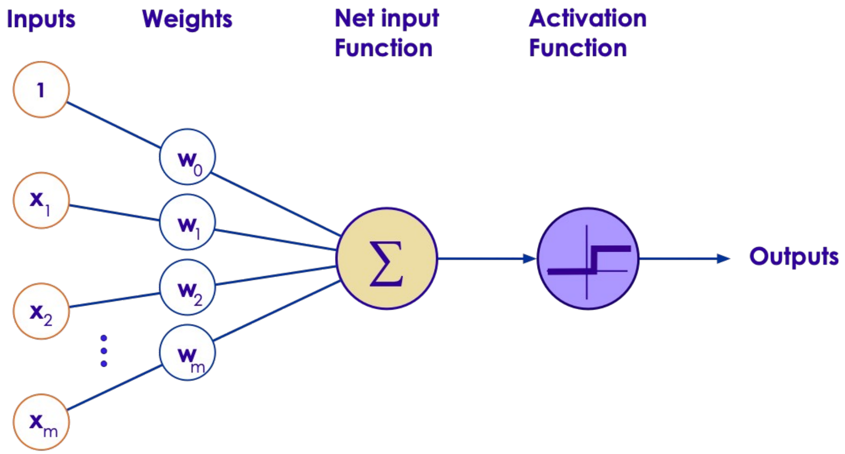
Image Credit: http://www.jeannicholashould.com/what-i-learned-implementing-a-classifier-from-scratch.html

The Perceptron Model

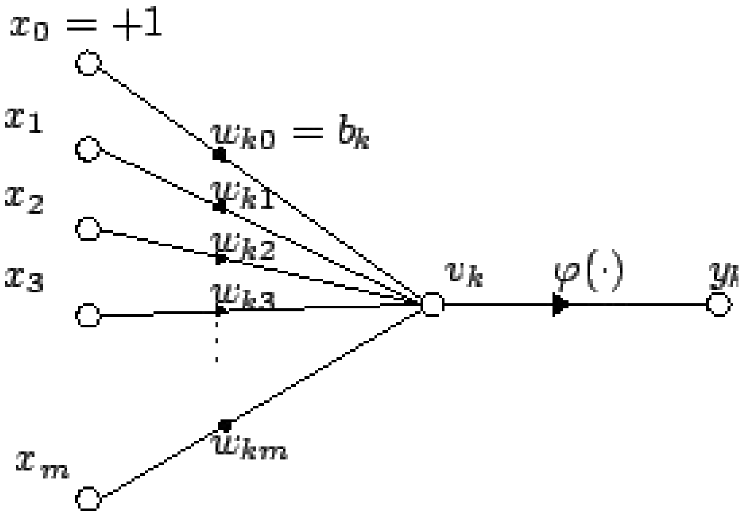


The Perceptron Model

- For each input x_i is assigned a weight w_i
  - The net input function produces a single computation from all the inputs
  - The activation function maps the computation to a category
  - Activation functions are often a step function

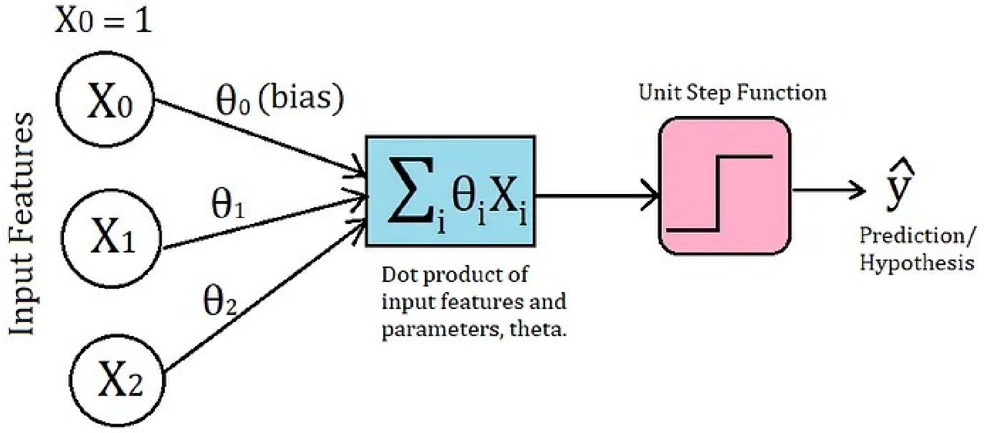


$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$



The Perceptron Algorithm

- An arbitrary set of weights is selected
  - The error of for each input is calculated and the weights adjusted if the point is miss-classified
  - This process is repeated a number of times
  - Eventually the weights will converge to an optimal solution
  - But only for linearly separable data



$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}.$$

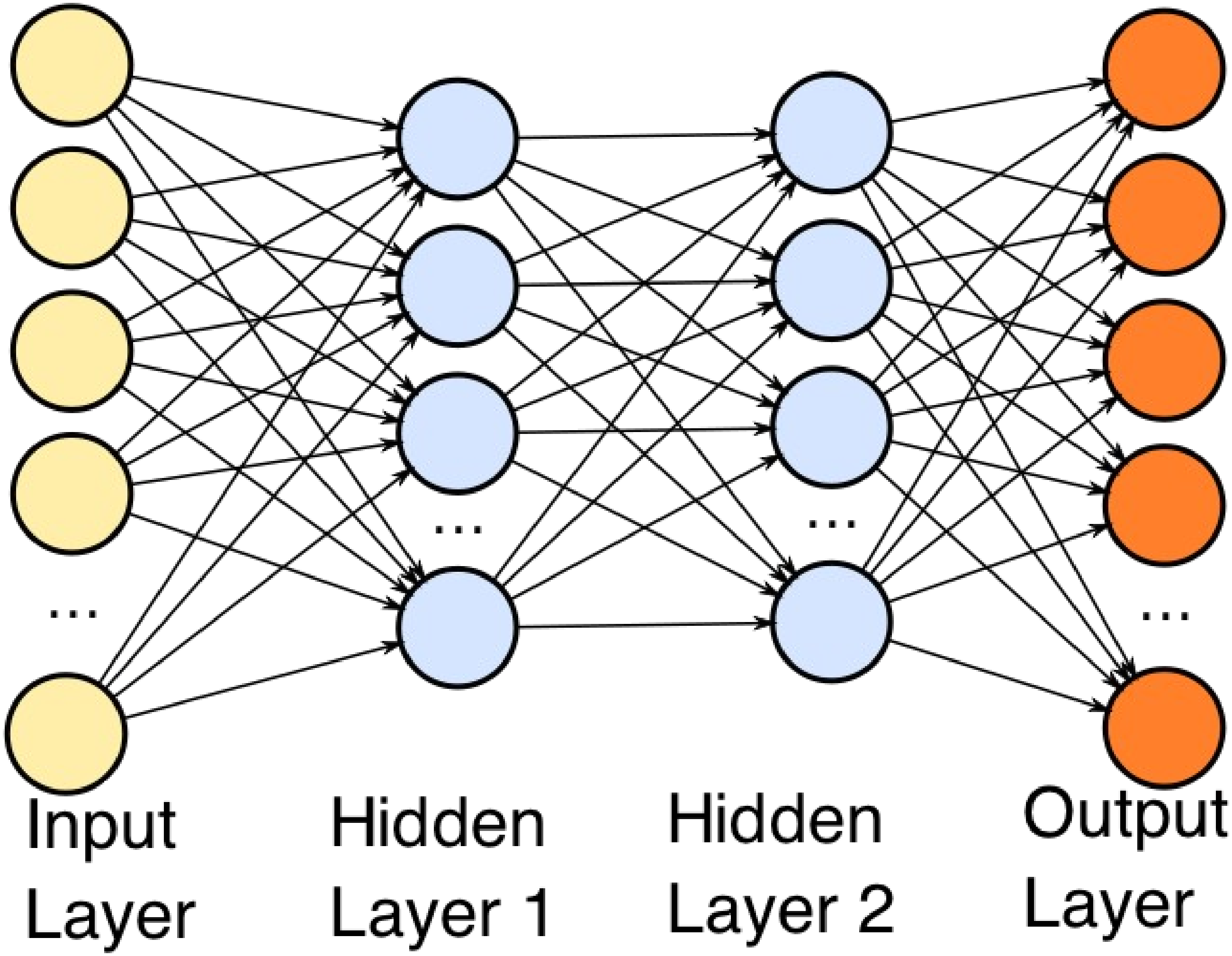
Intuitive Analogy

- A dev team is hiring a new programmer based on
  - Code inspections of previous work
  - Evaluation of previous performance reviews
  - Results of an in person performance test
- The hiring decision is a classifier based on some weighted combination of these three inputs
  - But each of these inputs is a the result of a weighted evaluation of raw data
    - Some performance reviews are weighted higher than others for example
    - Some previous work is weighted differently depending on the programming language

Feed Forward Neural Networks

- Also known as *Multi-Layer Perceptrons* (MLP) or *Deep Feedforward Neural Networks* (DFNN)
- Feedforward Network Design
  - There are multiple layers
  - Each layer has many neurons (previously called perceptrons)
  - Each neuron is connected to neurons on previous layer
  - Information flows through ONE-WAY (no feedback loop)
  - Composed of: Input, Output and Middle (Hidden) layers
  - Nets with more than one hidden layer are called deep neural nets

Feed Forward Neural Networks



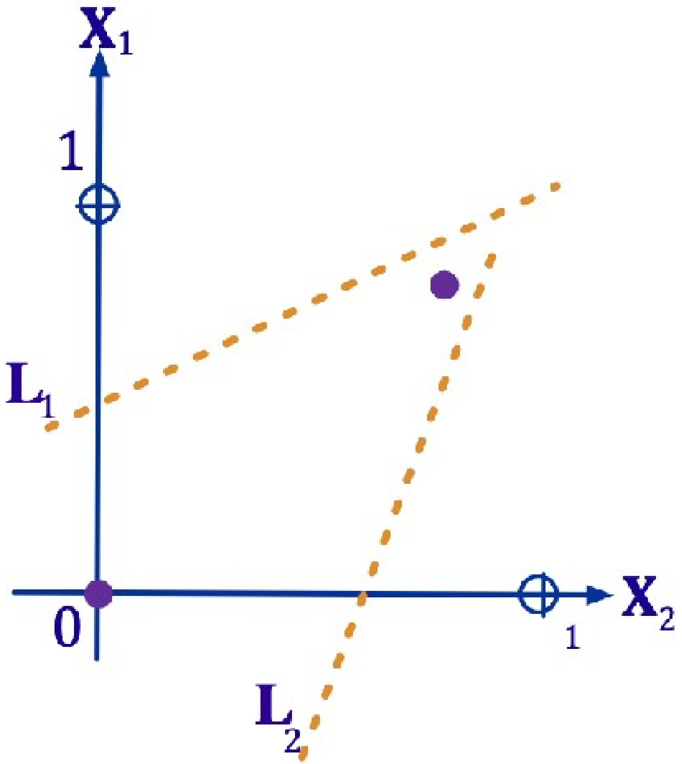
Hidden Layers

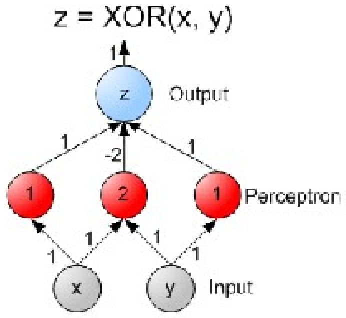
- Hidden Layers allow us to solve the "XOR" and related problems by creating a nonlinear decision boundary
- How Many hidden layers?
  - Many nonlinear problems solvable with one hidden layer
  - Multiple hidden layers allow for more complex decision boundaries
- One Hidden Layer can be enough
  - It has been proven that any function can be represented by a sufficiently large neural network with one hidden layer
  - Training that network may be difficult
  - Modern training methods mean that more than one layer is required in many cases.
- Each layer can be thought of as a step in solving the problem
  - Although we don't know what is being done at each step, at least in general
  - That is being decided by the NN when it is being trained

Trivial XOR Example

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$Y = X_1 \oplus X_2$$





How Hidden Layers Work

- In the XOR example we had two linear classifies in the hidden layer
  - Each of the classifiers solved part of the classification problem
  - The hidden layer produced two classifier results
- The final step produced a classifier which was a linear combination of the two classifier results from the hidden layer
- In deep learning, each hidden layer takes the outputs of classifiers from the previous layer and produces a combination of those inputs
  - This can be thought of as a linear combination of linear combinations of the original inputs

Intuitive Analogy

- A dev team is hiring a new programmer based on
  - Code inspections of previous work
  - Evaluation of previous performance reviews
  - Results of an in person performance test
- The hiring decision is a classifier based on some weighted combination of these three inputs
  - But each of these inputs is a the result of a weighted evaluation of raw data
    - Some performance reviews are weighted higher than others for example
    - Some previous work is weighted differently depending on the programming language

Intuitive Analogy

- To represent this a neural net
  - The input layer has three neurons representing each of the three evaluation criteria
  - The hidden layer can be thought of as a set of judges, each of which is evaluating one input
    - Each judge is a classifier
    - There can be an arbitrary number of judges
  - The output layer is the hiring decision based on a weighted input of each judge's decision to produce final classification

The Neuron

- Each node can be represented as a linear combination inputs and an activation function
  - Training the network is choosing the optimal weights for all the layers though some sort of iterative process

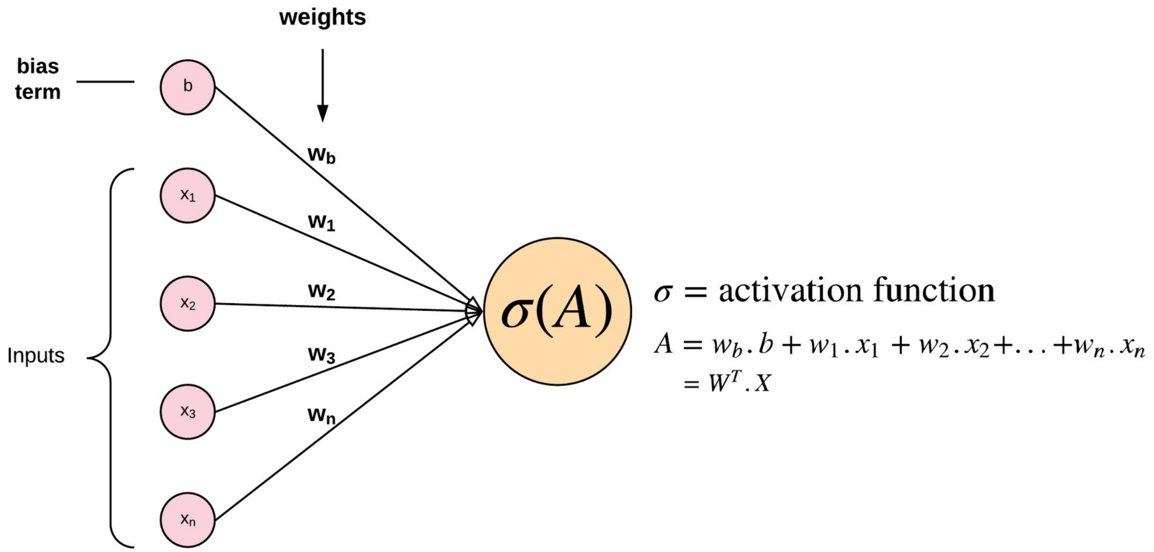
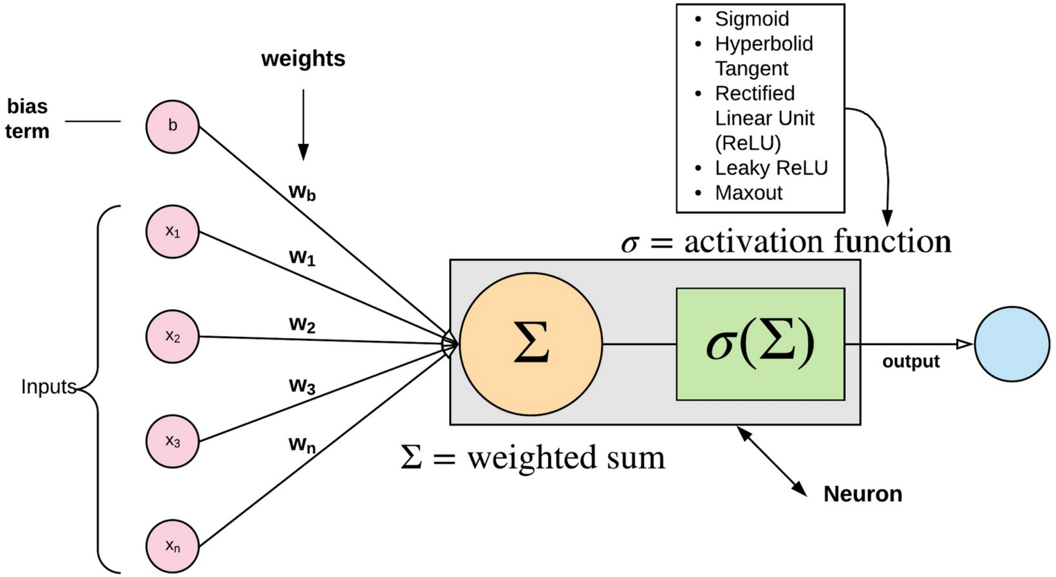


Image Credit: https://ekababisong.org/gcp-ml-seminar/deep-learning/

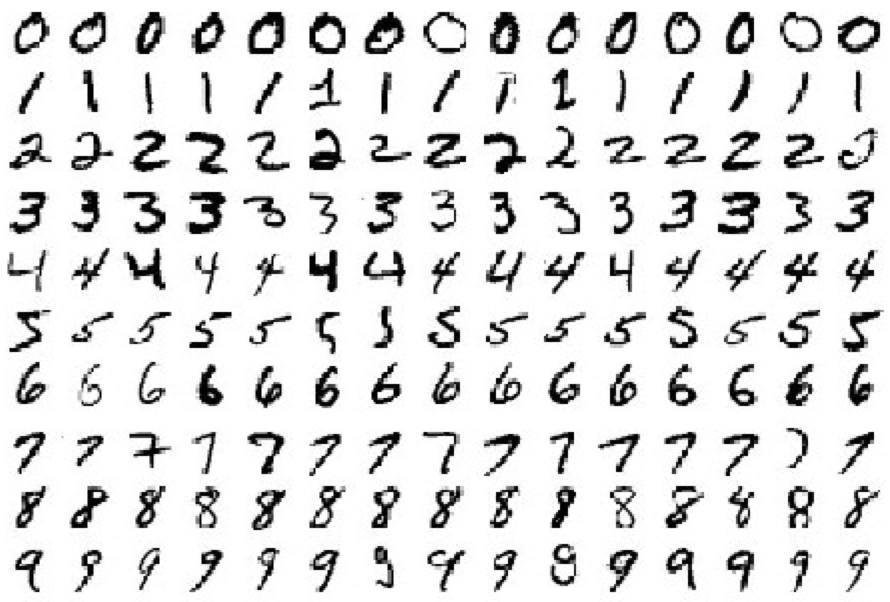
The Neuron

- To get successful training, the activation functions are generally non-linear



Example Handwritten Digit Recognition

- How to identify handwritten digits with a neural - an intuitive explanation
  - We are just emulating a NN, not claiming this is exactly what an NN does

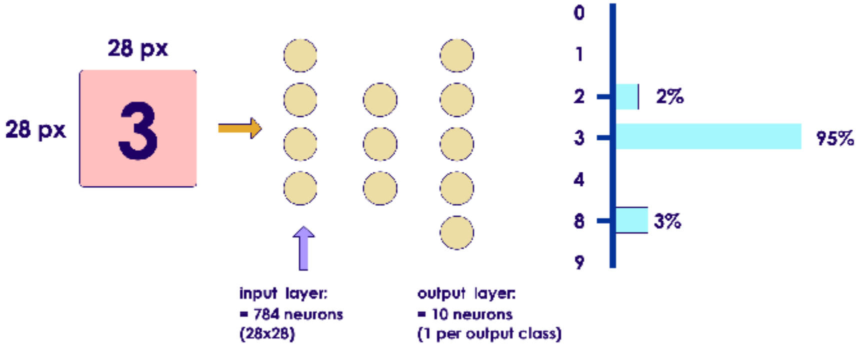


Example Handwritten Digit Recognition

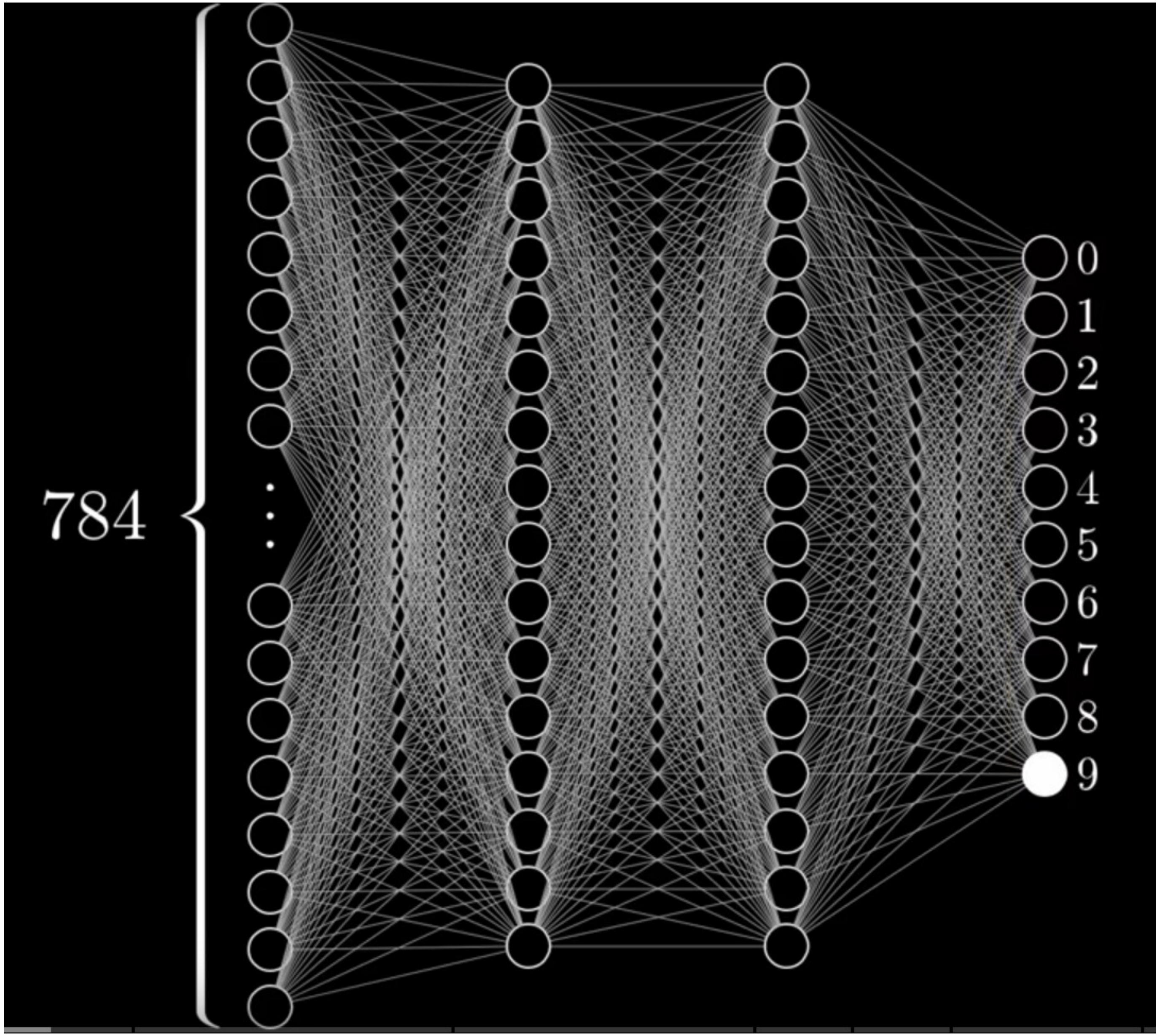
- We assume each input is an image
  - Each pixel of the image represents one input
- The first layer can be thought of as identifying features
  - Does the image contain a horizontal line?
  - Does the image have a circle in it?
  - Think of each neuron identifying one feature
- The second layer then makes a decision based on the features
  - If we get a horizontal line in the upper part of the image and a vertical line attached at the right side then we probably have a seven… or maybe a one

Example Handwritten Digit Recognition

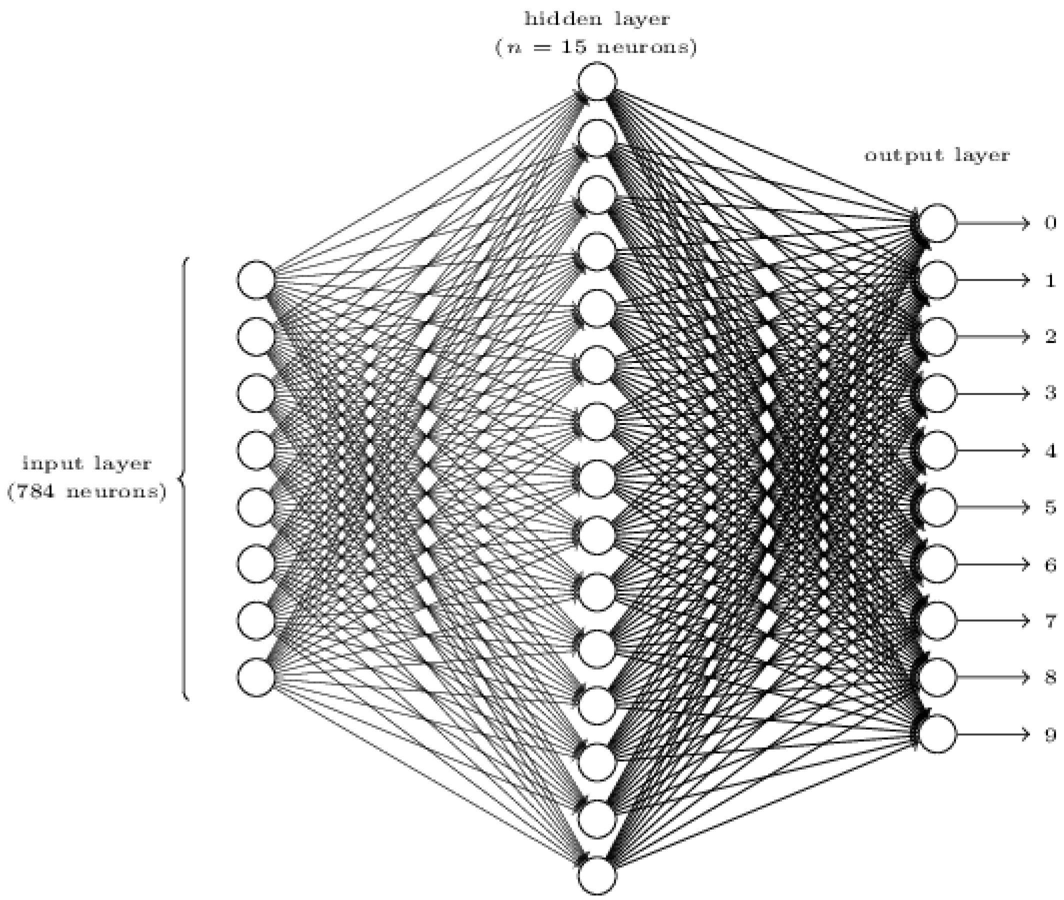
- Input layer sizing
  - Match input dimensions: 784 = 28 x 28 pixels
- Output layer sizing
  - One neuron per output class - 10 (one for each digit; 0, 1, ..8,9)
- Hidden layer sizing is flexible

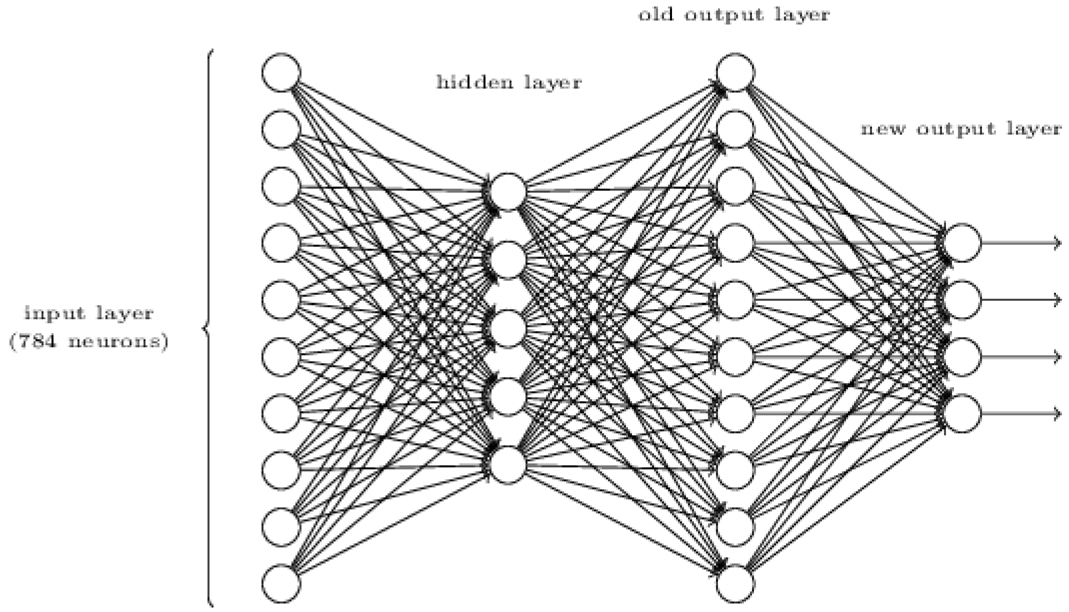


Possible NN Architecture



Several Different Possible NN Architectures





Caveat

- We don’t really know what each layer is doing
  - This is not AI, this is ML
  - The neural network does not actually emulate how you recognize a handwritten digit
- The intuitive explanation we gave might be similar to what really happens
- But probably not

Sizing Neural Nets

- Input Layer
  - Size: Equal to Number of Input Dimensions plus bias term
- Hidden Layer(s)
  - Size depends on training sample, input features, outputs
- Output Layer
  - Regression: 1 single neuron (continuous output)
  - Binary Classification: 1 single neuron (binary output)
  - Multi-class Classification: 1 node per class label

Learning with Neural Nets

- So far, we have only looked at feed forward nets
  - For a given set of weights at each layer we get a prediction for each data point
  - But we also have a label for that point
  - So we can create a cost function in a number of standard ways
  - For example, the average of the sum of the squares of the difference between the prediction and label
- This now gives a cost function that can be minimized to find the best set of weights in the neural nets
  - We saw this before with gradient descent

Back Propagation Algorithm

- The back propagation algorithm can be intuitively explained in a two hidden layer neural net by:
  - Find the weights necessary for the inputs to the second layer to minimize the error on the output
    - This is our optimal set of weights for the second layer
    - We will use a loss function like gradient descent
  - Then go backwards and find the weights necessary for the inputs to the first layer that will produce the optimal inputs for the second layer
- We repeat this for all the layers we have in the neural net
  - This is a single training iteration
  - We do as many more iterations as necessary

Back Propagation Algorithm

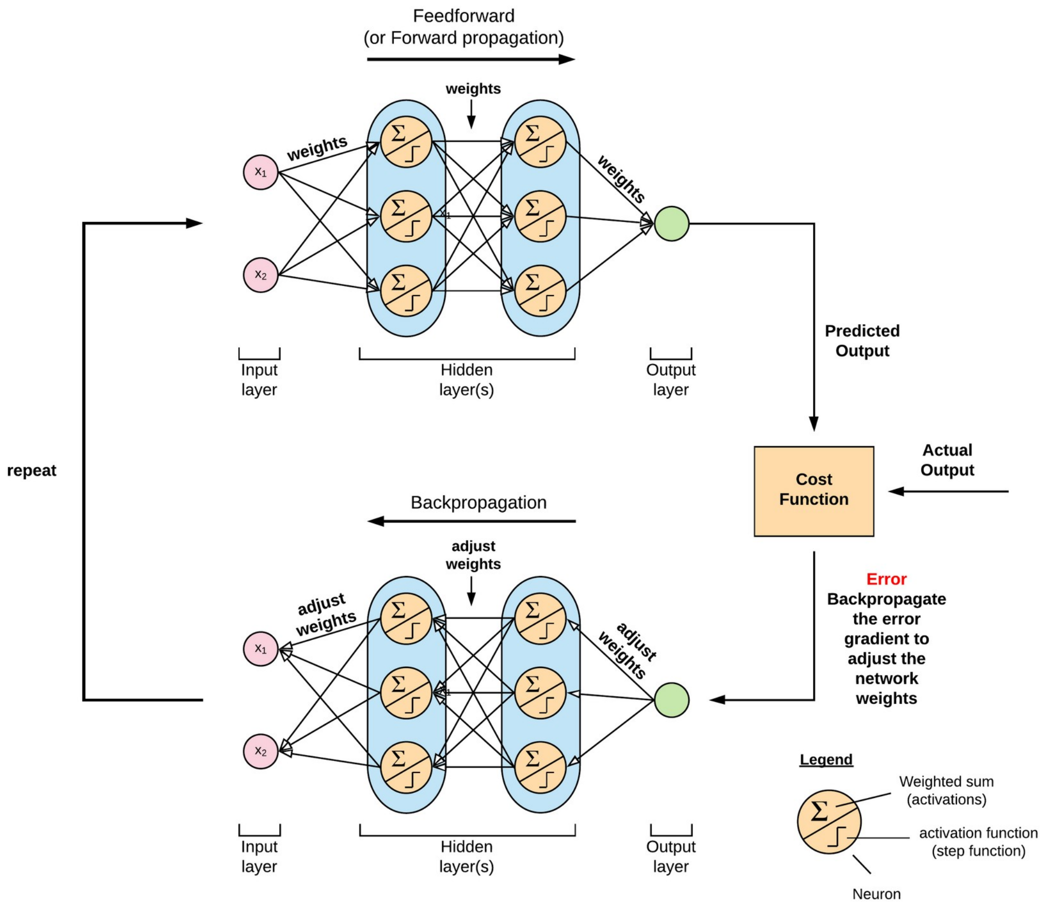


Image Credit: https://ekababisong.org/gcp-ml-seminar/deep-learning/

Back Propagation Algorithm

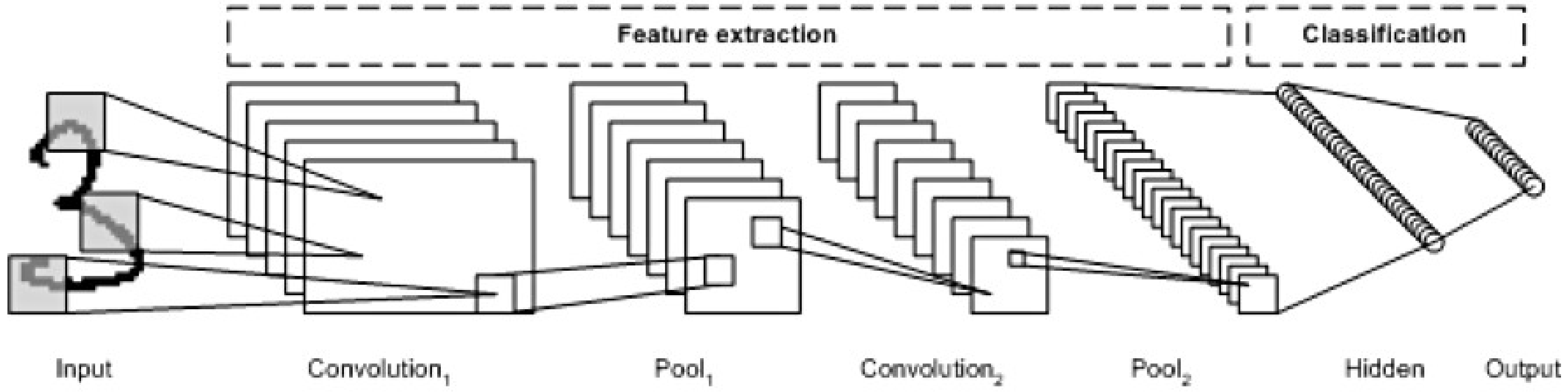
- Once we have the amount we want to change each parameter by for each data point, we can use gradient descent to improve the net
- This can be incredibly computationally intensive
  - We will often use smaller batches of data to get approximations
  - This is stochastic gradient descent
  - It will also eventually converge like gradient descent
  - But each step may not be optimal

Convolutional Neural Nets (CNNs)

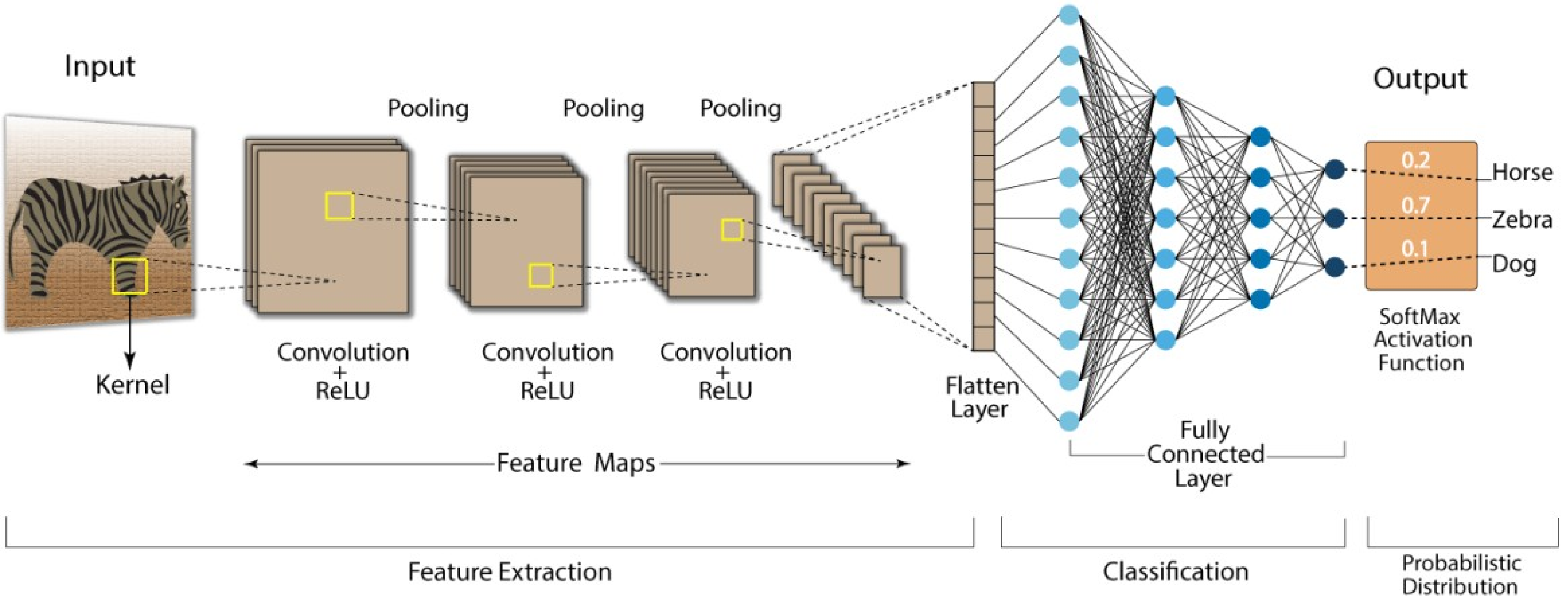
- Used in image processing
- Imagine a small patch being slid across the input image
  - This sliding is called convolving
  - Similar to a flashlight moving from the top left end progressively scanning the entire image
  - This patch is called the filter/kernel
  - The area under the filter is the receptive field
- The idea is to detect local features in a smaller section of the input space, section by section to eventually cover the entire image

Convolutional Neural Nets (CNNs)

- CNNs are a sequence of layers:
  - Input layer
  - Convolutional Layer
  - ReLU (Rectified Linear Unit) Activation
  - Pooling Layer
  - Fully Connected Layer(s)
- We usually have more than one sequence of layers



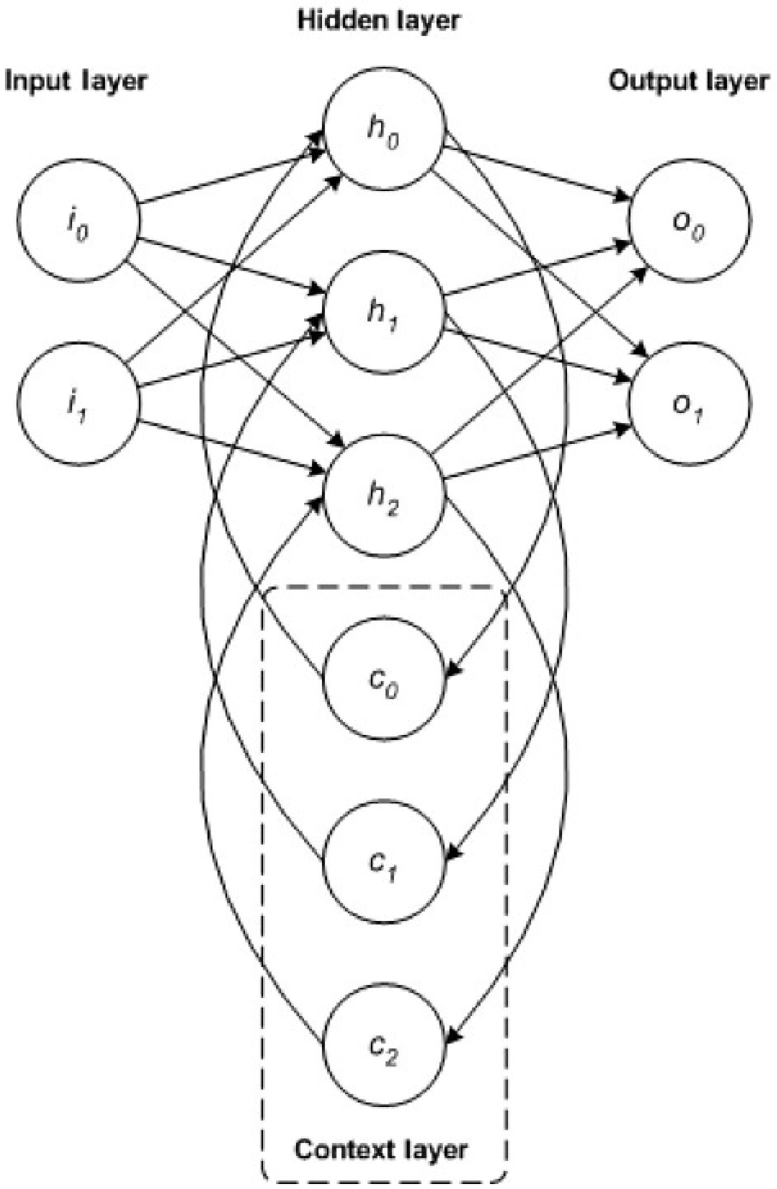
Convolutional Neural Nets (CNNs)

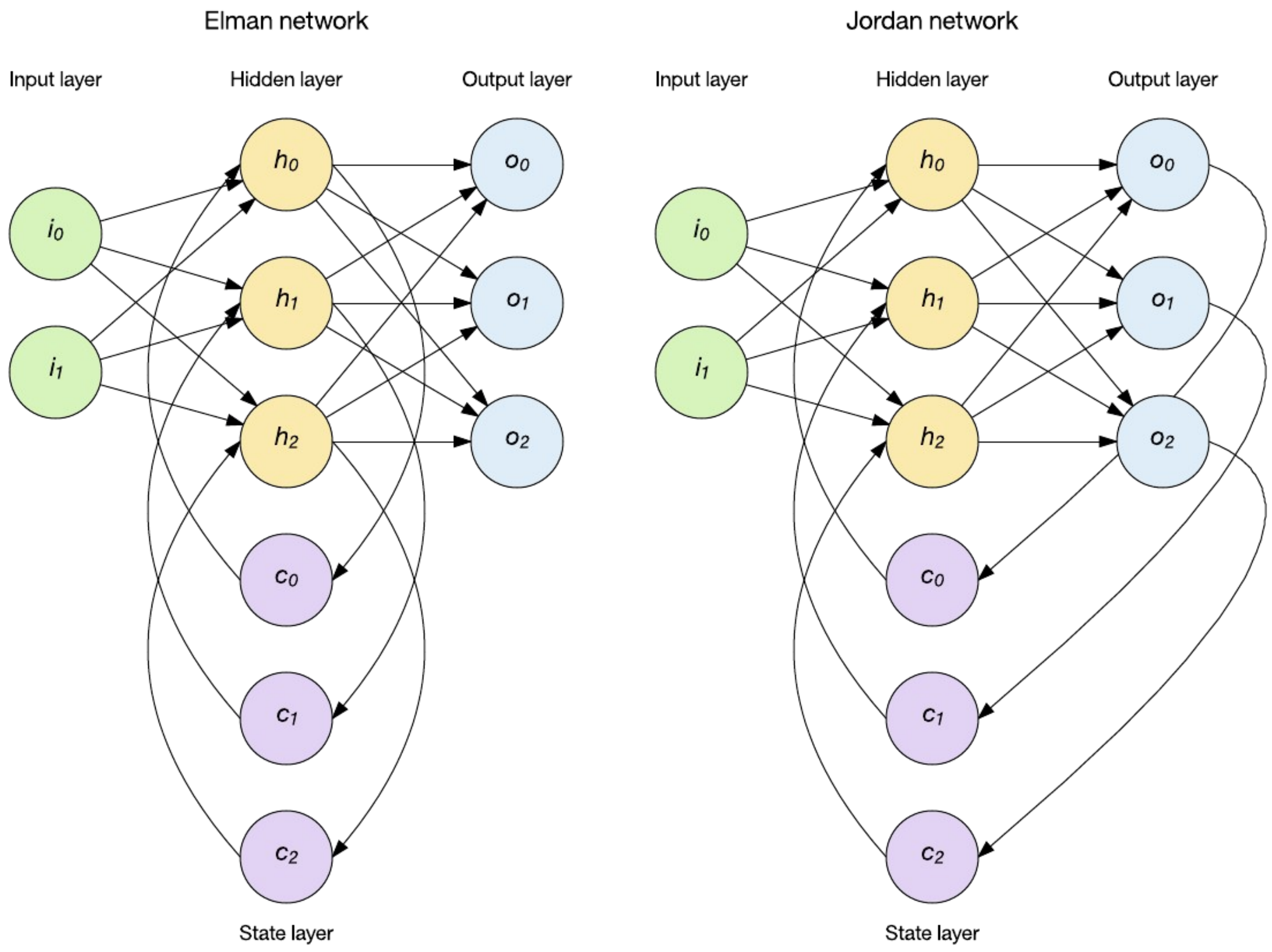


Recurrent Neural Nets (RNNs)

- A problem with Feedforward Neural Networks
  - Feedforward Neural Networks can model any relationship between input and output.
  - However they can’t keep/remember state
  - The only state retained is weight values from training.
  - They don’t remember previous input!
  - In Feedforward Networks, data flows one way, it has no state or memory
  - RNNs have a ‘loop back’ mechanism to pass the current state to the next iteration
- In many problems, like language processing
  - What was processed before has an impact on what is being processed now
  - For example, the meaning of a word in a sentence depends on the words preceding it

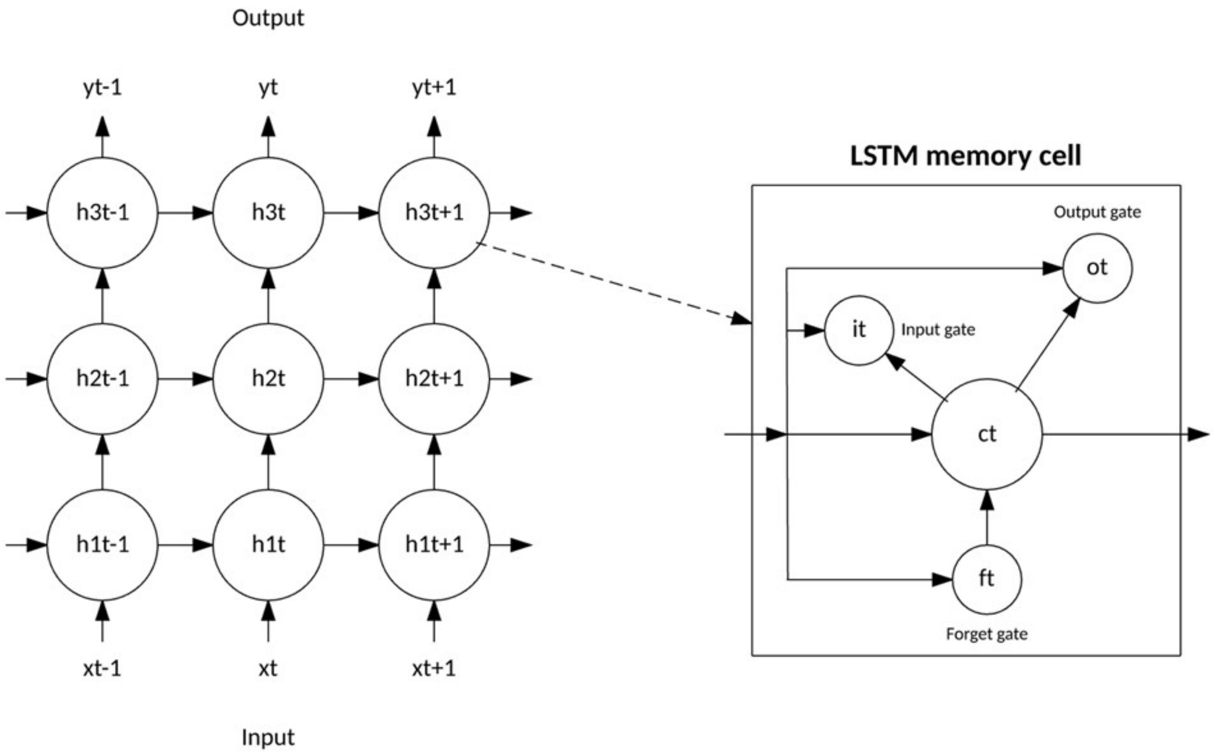
Recurrent Neural Nets (RNNs)





Long short-term memory

- An improvement on RNN
  - Has a memory cell in addition to neurons
  - Remembers what is important not just what it did last

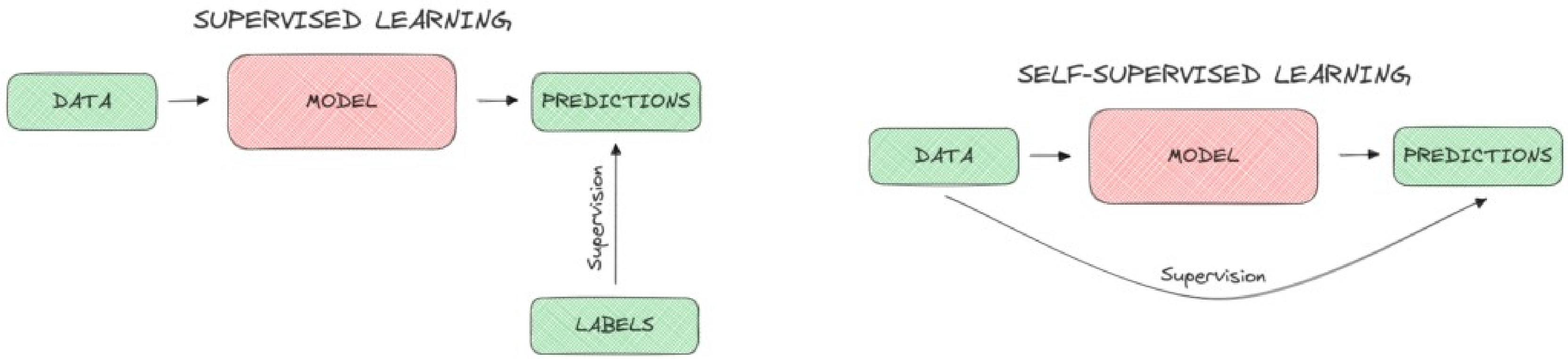


Reaching the Limits

- All the NNs we have looked at so far have been supervised machine learning
  - This is the bottleneck in scaling up ML models
  - We have the compute power
  - We have the data thanks to big data
  - But the existing models did not scale
- Two significant problems
  - It becomes prohibitive to label data for training
  - Feature engineering become a major stumbling block

Self Supervised Learning

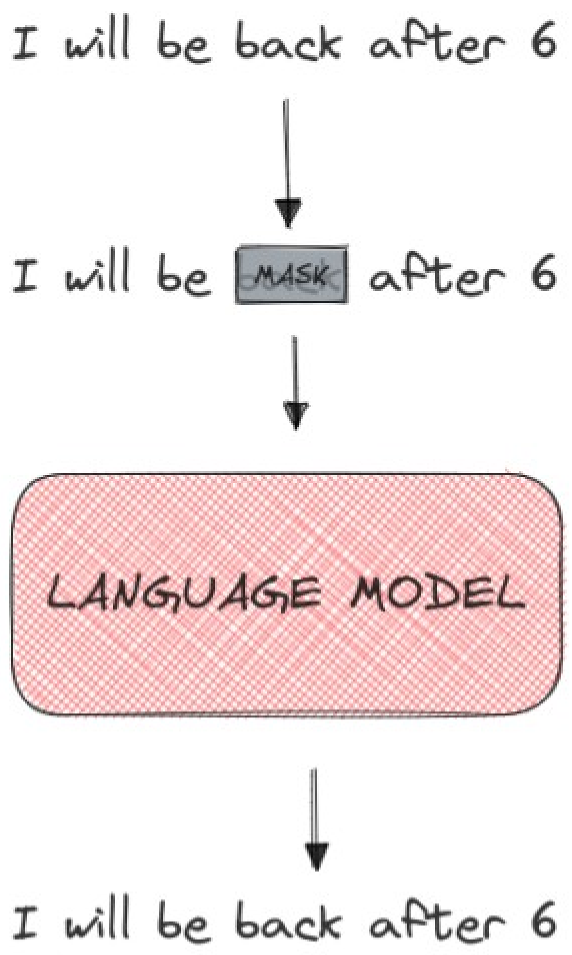
- An extension of unsupervised learning
- Similar to how people learn
- In self supervised learning
  - The model is not trained using external labels
  - The model generates labels and features from the data
  - These are then used to make predictions
  - How good the predictions are is based on *recovery error rate* or how well the model can predict patterns in the original data

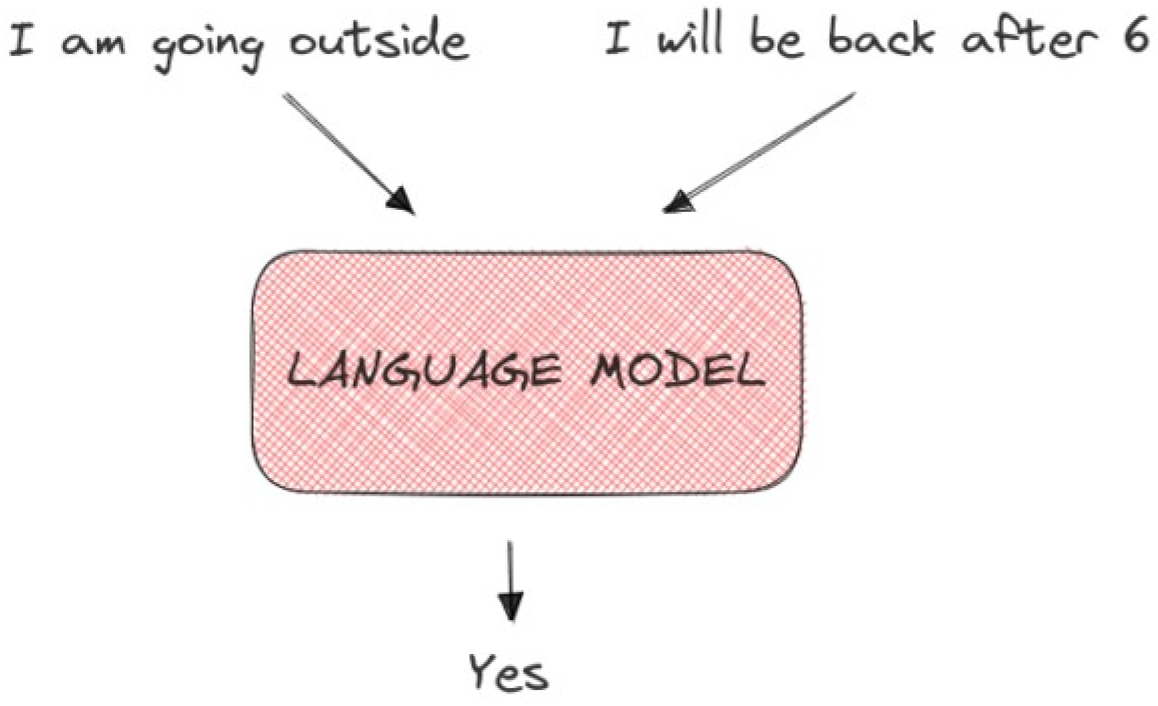


SSL for Text

- Two strategies commonly used are
- MaskedLM
  - Some words are are masked out of an input sentence
  - The model is trained to predict the missing words and train the language model to predict these hidden words
  - Used in in techniques like word2vec
- Next Sentence Prediction
  - Model takes as input a pair of sentences and learns their relationship
  - Predicts if the second sentence comes after the first sentence for example

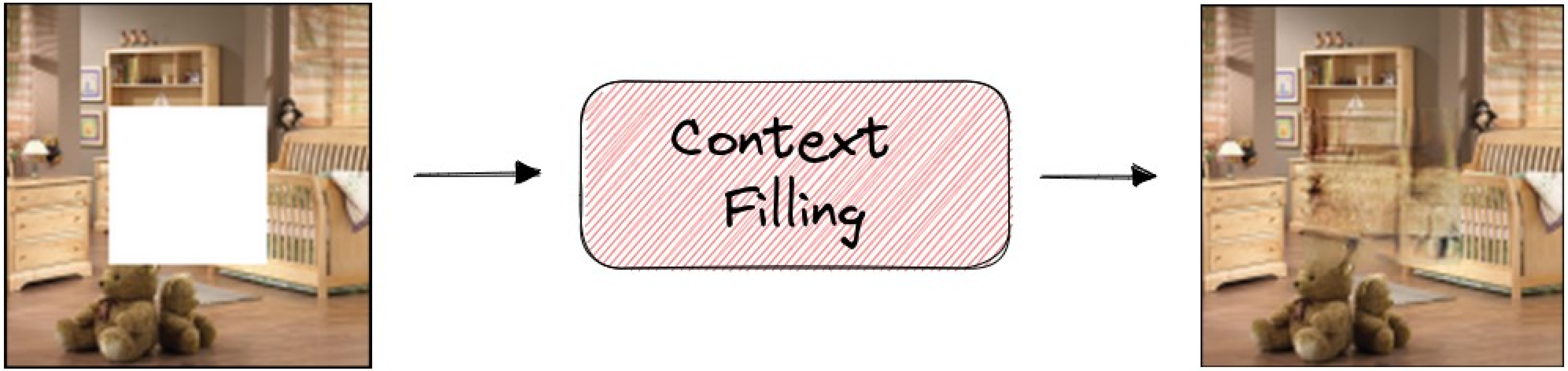
SSL for Text


I will be back after 6
I will be MASK after 6
LANGUAGE MODEL
I will be back after 6


I am going outside
I will be back after 6
LANGUAGE MODEL
Yes

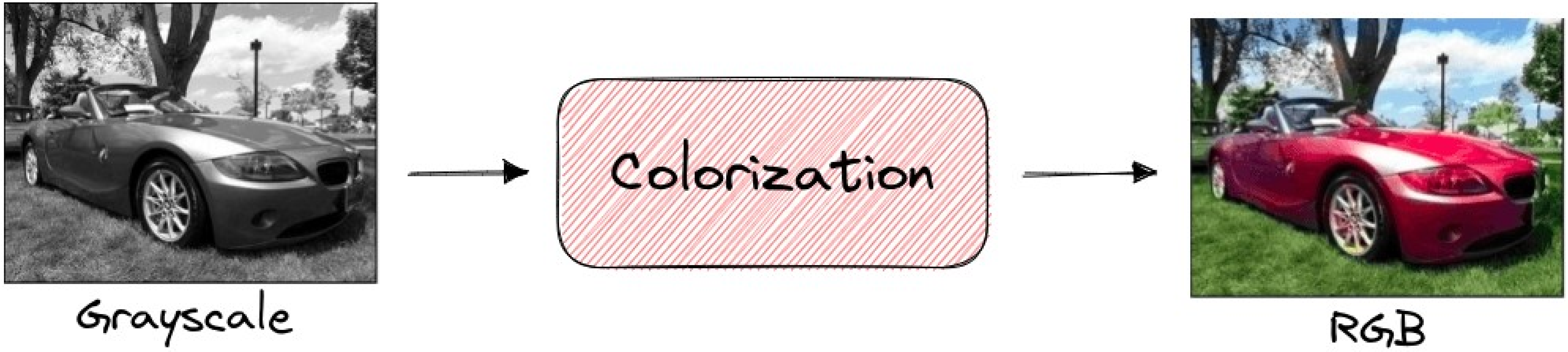
SSL for Images

- Image inpainting
  - Various parts of an image are masked out
  - The model tries to reconstruct the missing pixels in context



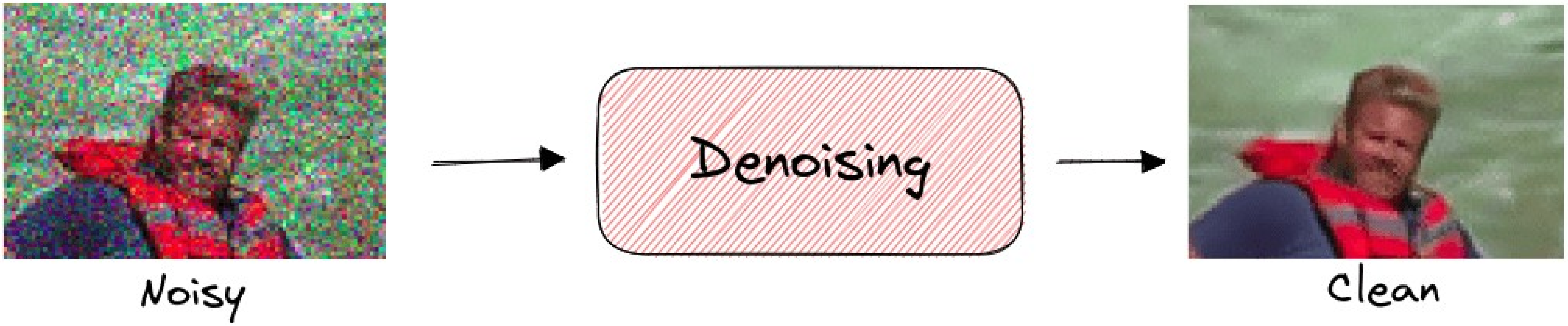
SSL for Images

- Colorization
  - Tasked with coloring an image that has been greyscaled
  - Often captures important semantic information



SSL for Images

- Denoiseing
  - Model learns to recover an image from a corrupted or distorted version



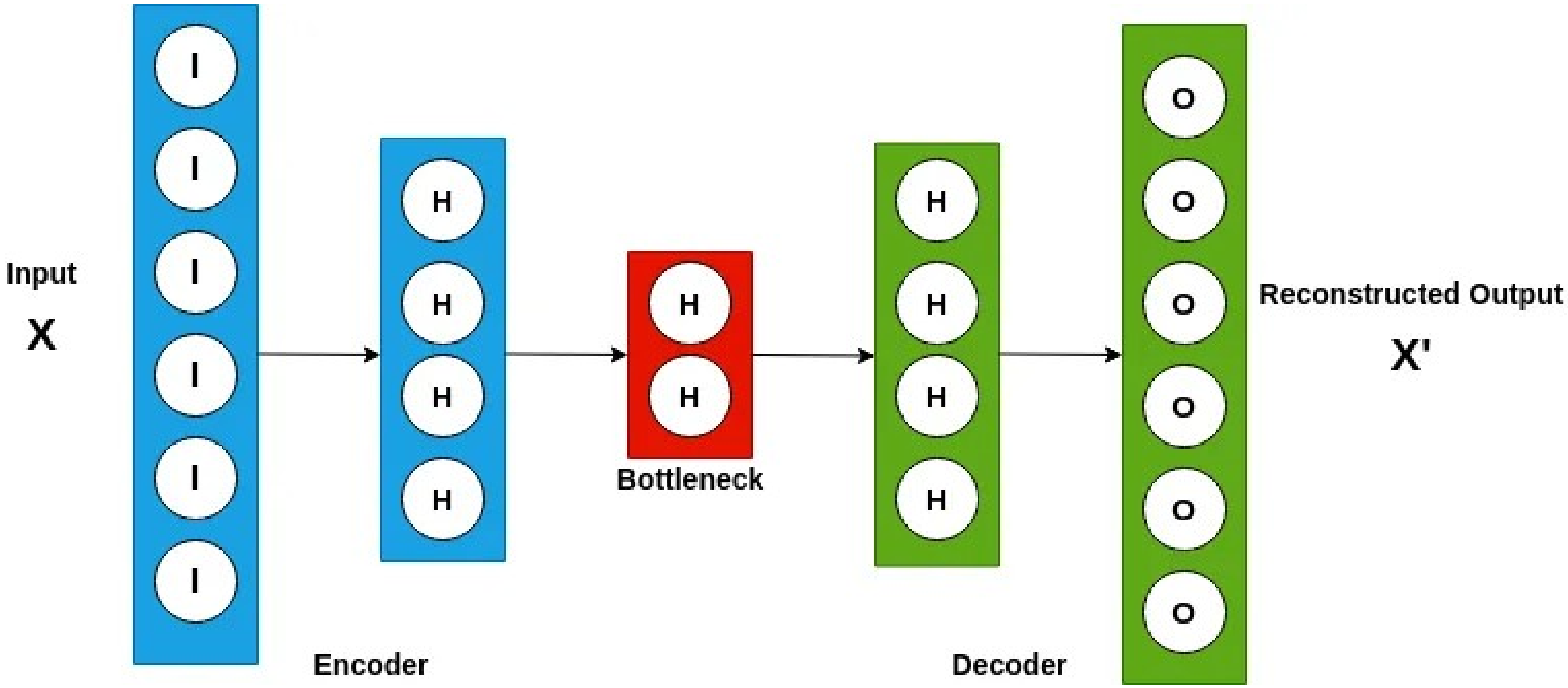
Autoencoders

- Self supervised learning but focusing on features
- The encoder learns two things
  - A encoding function that maps a data point to an encoded representation
  - A decoder that reconstructs a data point from an encoded representation
- The objective is to learn an efficient feature representation
  - Learning takes place my minimizing the *reconstruction loss* or how different the reconstructed data is from the original data
- The reason for using an autoencoder is to understand only the deep correlations and relationships among the data
  - We do this by forcing dimensionality reduction to force the decoder to actually have to reconstruct the data
  - Otherwise, the trivial autoencoder would just return the original data point unmodified

Autoencoders

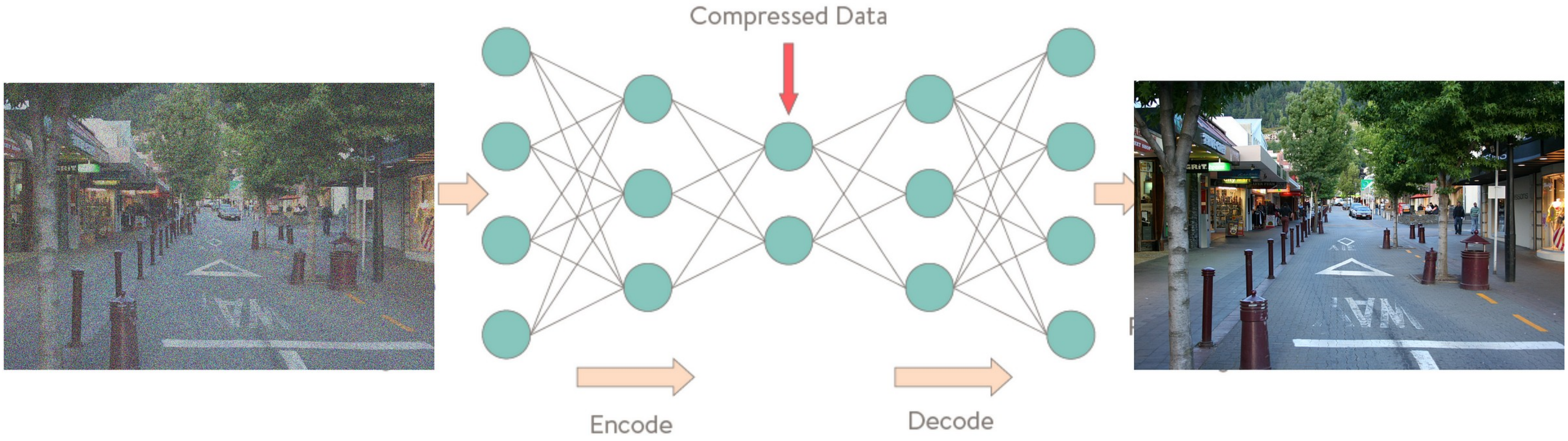
- Uses a bottleneck layer
  - Has a lower number of nodes and the number of nodes in the bottleneck layer also gives the dimension of the encoding of the input
  - The representation from the bottleneck layer is called the latent space
  - This forces the model to explore relationships among the features – exploits the natural structure of the data
  - However, it does require that relationships do exist, no encoding can be done if all the features are independent
  - Can learn non-linear relationships
  - Want it to be sensitive enough to minimize reconstruction loss but not to overfit the data
  - We use back propagation to train the encoder

Autoencoders



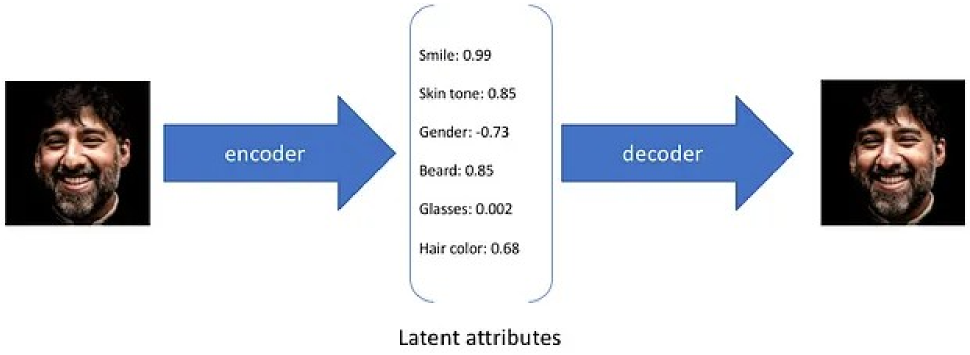
Denoise Autoencoders

- Uses encoding and decoding to remove noise from an image



Standard Autoencoders

- The features in the latent space are discrete vectors
  - In the example below, the face image is reduced to six features
  - For applications like removing noise, the autoencoder looks at the features defined in the latent space and ignores the noise
  - The autoencoder is not sensitive to small variations in the input



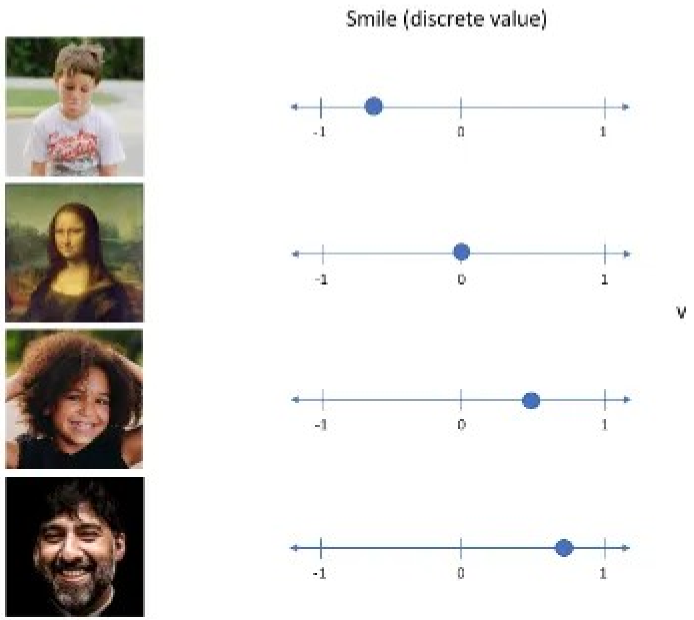
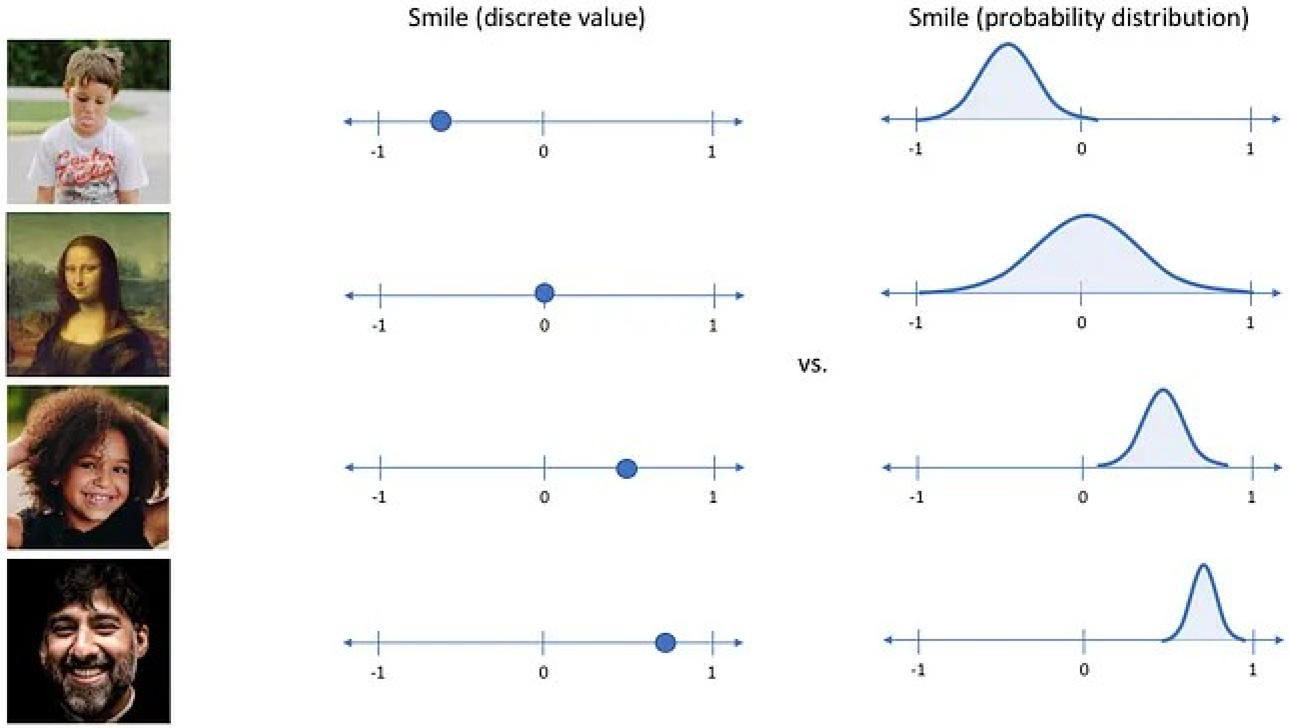


Image Credit: https://towardsdatascience.com/introduction-to-autoencoders-7a47cf4ef14b

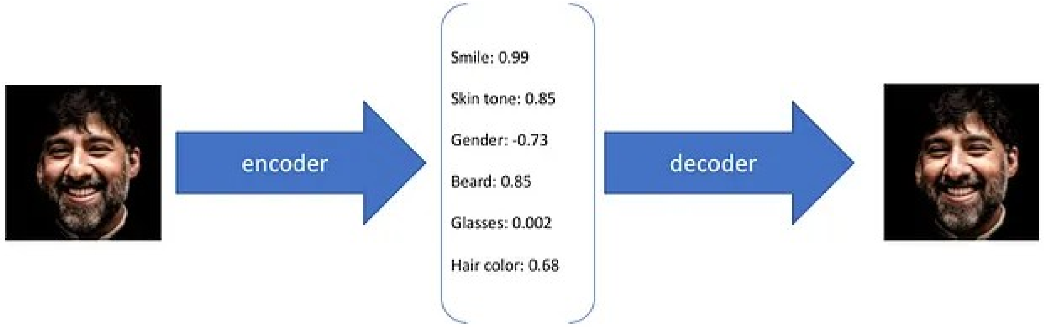
Variational Autoencoders

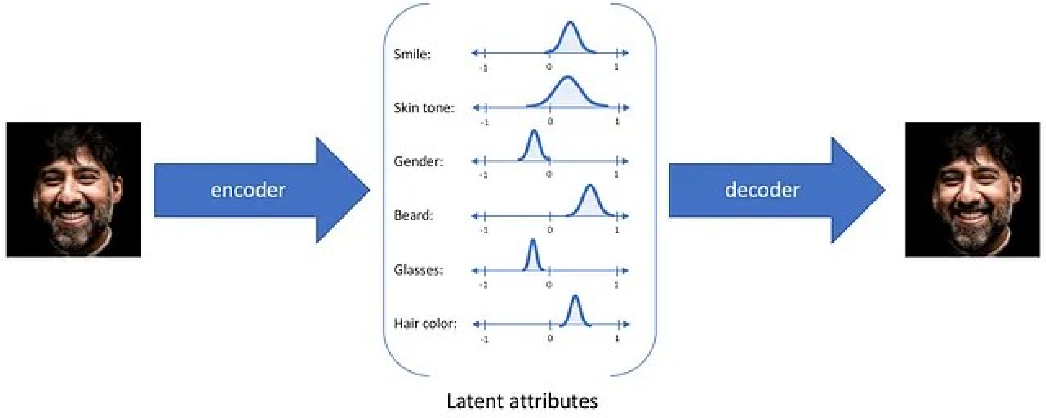
- Standard autoencoders are not good at generating new data
  - They are prone to overfitting which limits the generation of novel data
- Variational encoders replace discrete values with probablility distributions



Variational Autoencoders

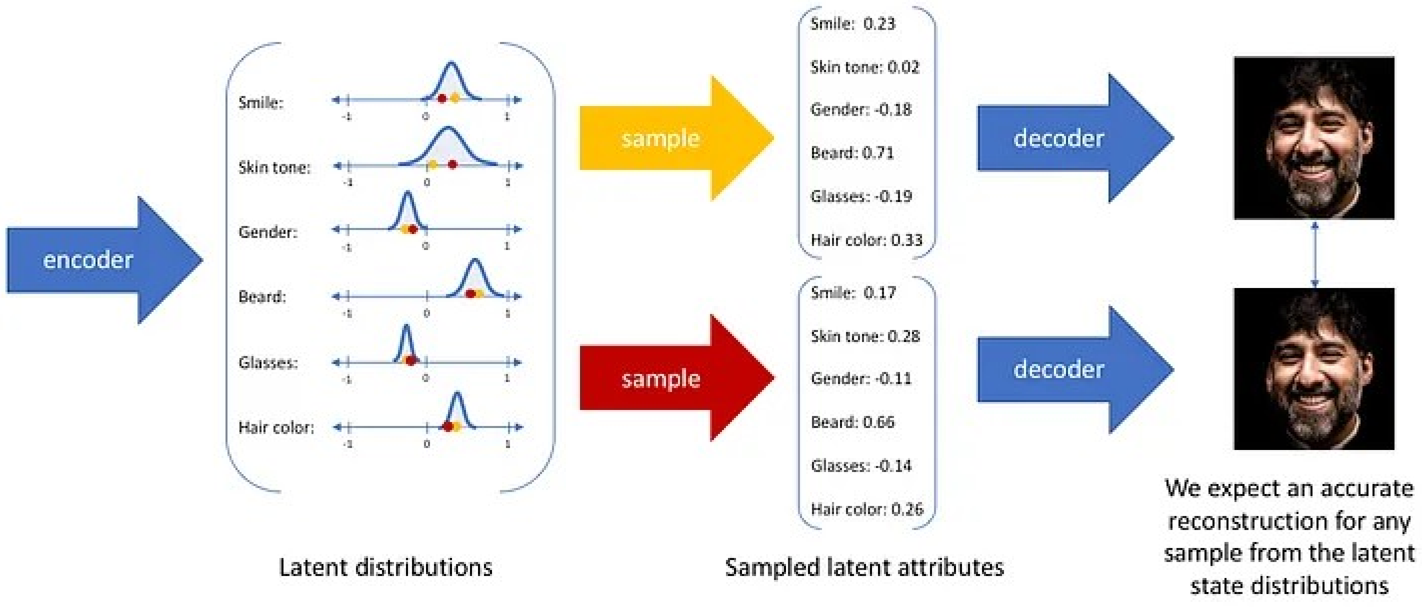
- This changes how the latent features are stored





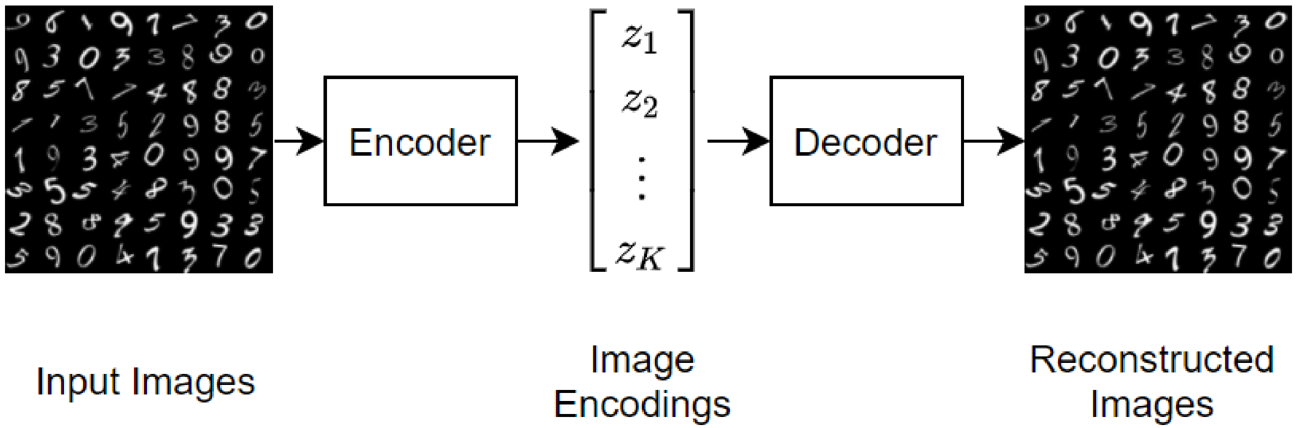
Variational Autoencoders

- Now the decoder samples from the probability distribution for each latent feature
  - Since the sampling is random the generated image is similar to the original but not the same as the original input image
  - The output is a novel generated data item that resembles the original



Variational Autoencoders

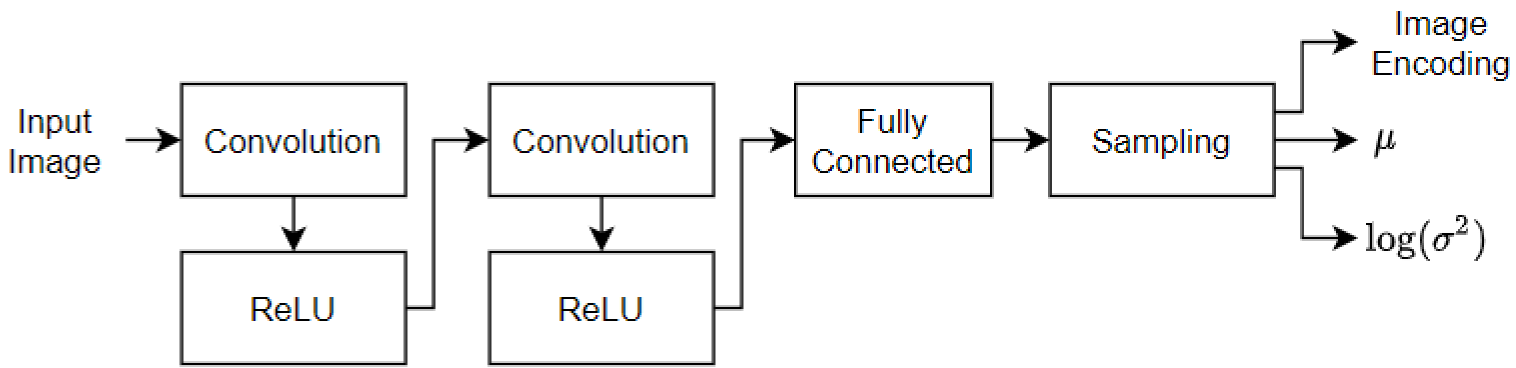
- To generate new images
  - We only use the decoder
  - Supply a random input representing representing something from the latency space
  - The NN of the decoder then constructs the image from the input
- Example – Generate new handwritten digits
  - Define the autoencoder



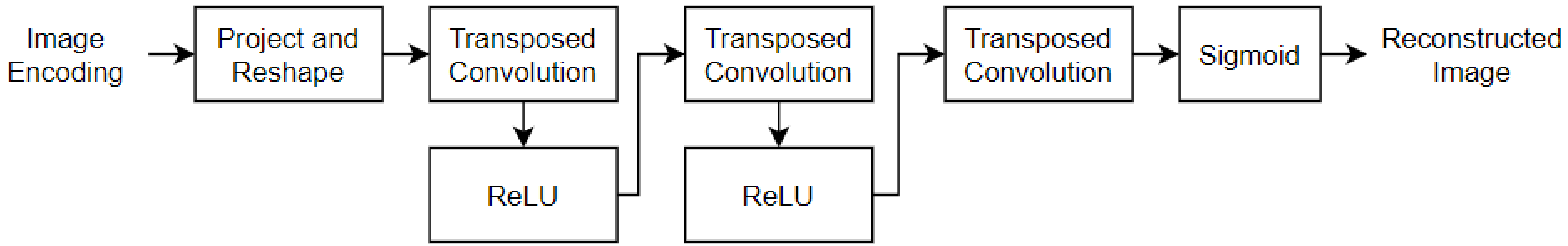
Credit: https://www.mathworks.com/help/deeplearning/ug/train-a-variational-autoencoder-vae-to-generate-images.html

Variational Autoencoders

- Encoder architecture



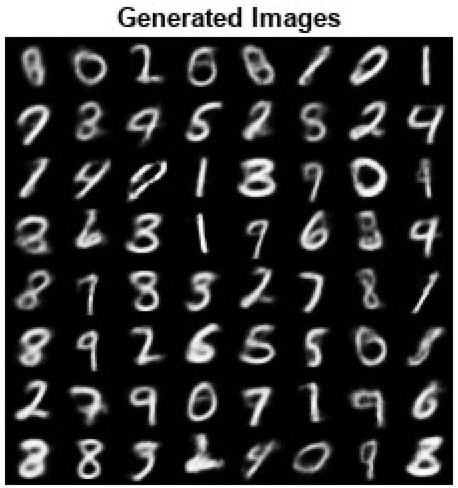
- Decoder Architecture



Credit: https://www.mathworks.com/help/deeplearning/ug/train-a-variational-autoencoder-vae-to-generate-images.html

Variational Autoencoders

- Supplying random inputs to the decoder produces these images



Credit: https://www.mathworks.com/help/deeplearning/ug/train-a-variational-autoencoder-vae-to-generate-images.html

Questions?

End Module