Lecture 08: Artificial Neural Networks

EE5253: Machine Learning (TE)

Mr. M.W.G Charuka Kavinda Moremada

Lecturer, Department of Electrical and Information Engineering, Faculty of Engineering, University of Ruhuna.

Lecture Overview

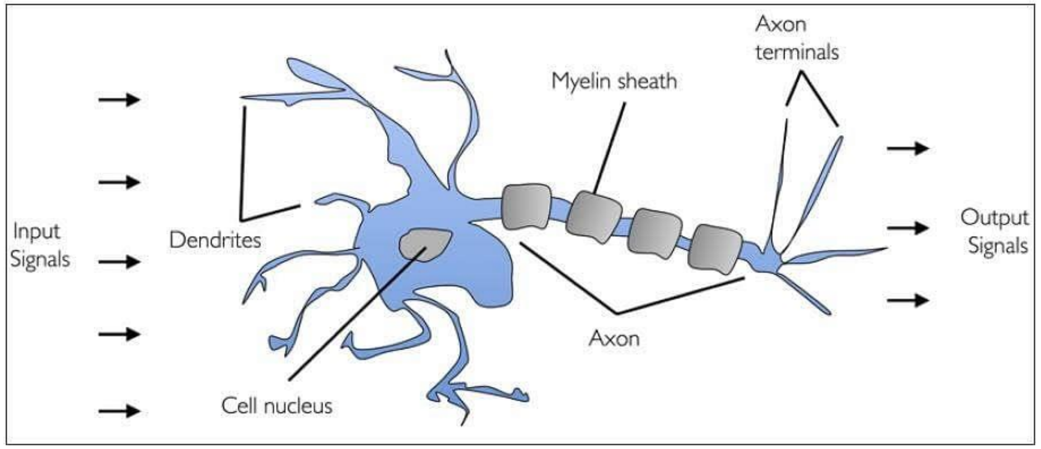
A text generated image from Bing image creator powered by DALLE-3

Perceptron

Biological Neurons

Multiple signals arrive at the dendrites and are then integrated into the cell body, and, if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

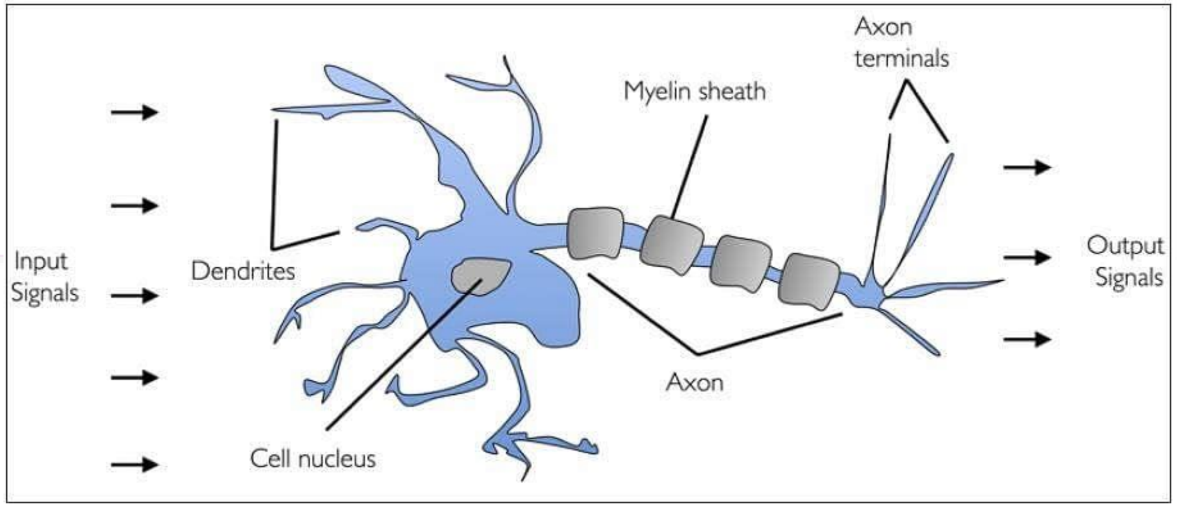
- Human brain consist of billons of neurons.
- Neurons are interconnected nerve cells in the human brain that are involved in processing and transmitting chemical and electrical signals.
- There are different components in a neuron:
  - **Dendrites:** Received information from other neurons.
  - **Cell Nucleus/Soma:** Processes the information received from dendrites.
  - **Axon:** Cable that is used by neurons to send information.
  - **Synapse:** Connection between an axon and other neuron dendrites.



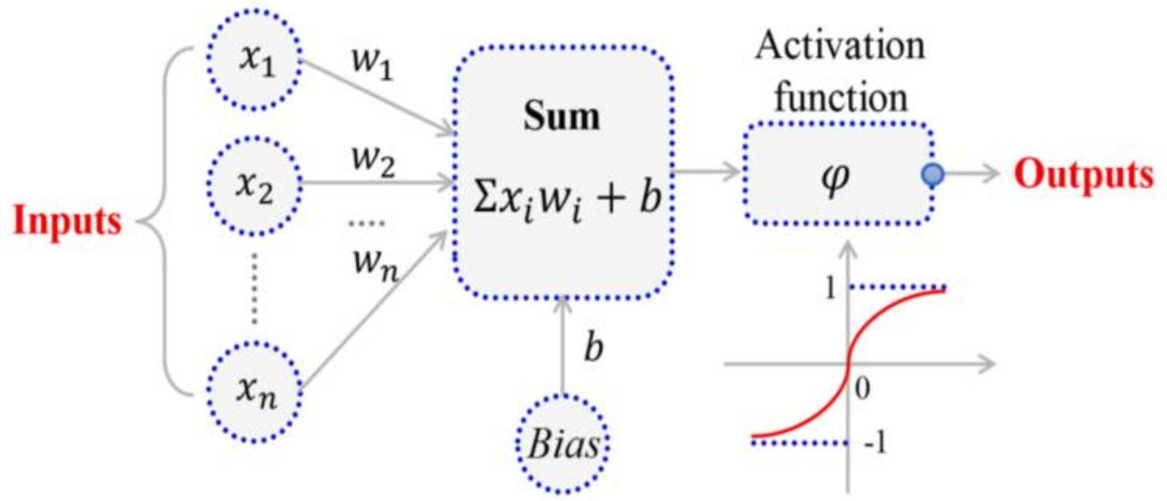
Artificial Neurons

- Warren McCullock and Walter Pitts published their first concept of simplified brain cell in 1943.
- They introduced the **McCullock-Pitts (MCP) neuron.**
- An artificial neuron is **a mathematical function based on a model of biological neurons**, where each neuron takes inputs, weighs them separately, sums them up and passes this sum through a nonlinear function to produce output.

Biological Neuron

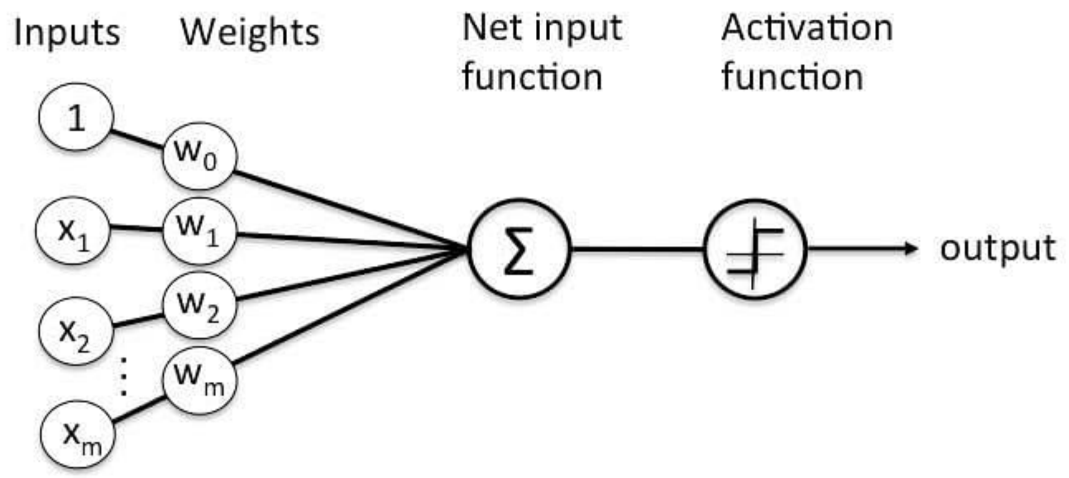


Artificial Neuron



Perceptron

- Perceptron is one of the first and most straightforward models of artificial neural networks.
- Perceptron was introduced by **Frank Rosenblatt in 1957s,** and it is the simplest type of feedforward neural networks.
- Frank Rosenblatt introduced the **perceptron learning rule** based on the original MCP neuron mentioned in one of the previous slides.
- **A Perceptron is an algorithm for supervised learning of binary classifiers.**



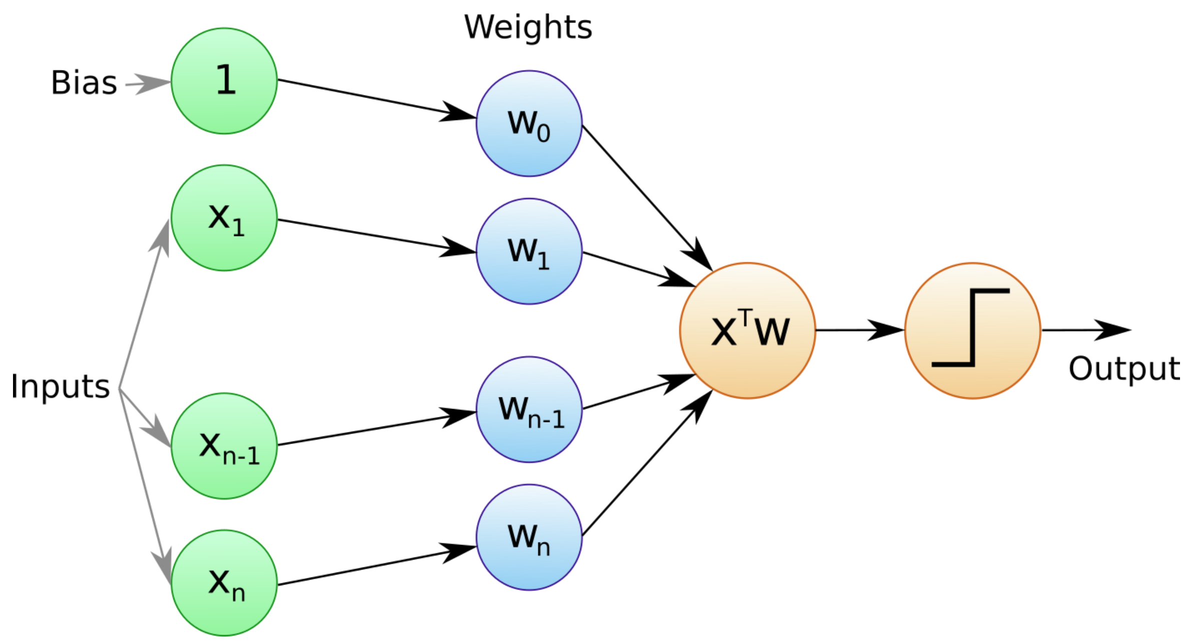
Perceptron: Basic Components

- **Input Features:** The perceptron takes multiple input features; each input feature represents a characteristic or attribute of the input data.
- **Weights:** Each input feature is associated with a weight, determining **the significance of each input feature in influencing the perceptron's output**. During training, these weights are adjusted to learn the optimal values.
- **Bias:** A bias term is often included in the perceptron model. The bias allows the model to make adjustments that are independent of the input. It is an additional parameter that is learned during training.
- **Activation Function:** The activation function determines the output of the perceptron based on the weighted sum of the inputs and the bias term. Common activation functions used in perceptron include the step function, sigmoid function, and ReLU function

Perceptron: Basic Components

- **Output:** The final output of the perceptron, is determined by the activation function's result. For example, in binary classification problems, the output might represent a predicted class (0 or 1).
- **Learning Algorithm (Weight Update Rule):** During training, the perceptron learns by adjusting its weights and bias based on a learning algorithm. A common approach is the **perceptron learning algorithm**, which updates weights based on the difference between the predicted output and the true output.

The bias term set to 1 by default.



Info

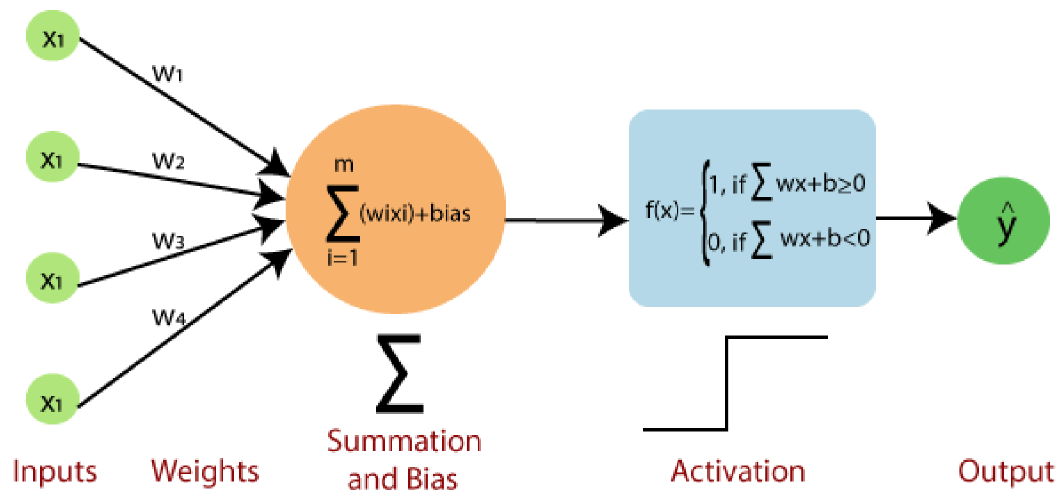
Perceptron: Bias

- **Handling zero inputs :** Bias terms are added to neural networks to help the models shift the activation function towards the positive or negative side. By adding bias term, it is ensured that that a neuron can activate even when all of its input values are zero.
- **Assists in achieving a better data fit and learning complex patterns:** The bias allows the perceptron to make adjustments to its output independently of the inputs. Bias is like the intercept added in a linear equation, which helps the model in a way that it can fit best for the given data. It allows the network to learn and represent more complex relationships between the input and output variables.
- **Additional Parameter:** The bias term serves as another model parameter (in addition to weights) that can be tuned to make the model's performance on training data as good as possible.

Perceptron: Types

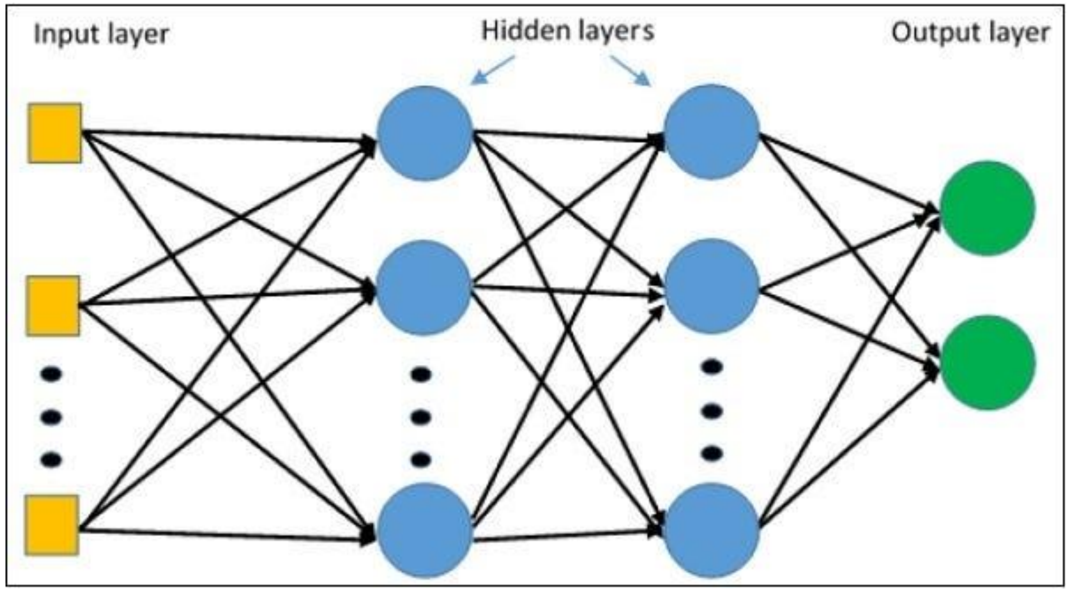
- **Single-Layer Perceptron:** This type of perceptron is limited to learning linearly separable patterns. Effective for tasks where the data can be divided into distinct categories through a straight line.
- **Multilayer Perceptron:** Multilayer perceptron possess enhanced processing capabilities as they consist of two or more layers, adopt to handle more complex patterns and relationships within the data.

Single Layer/Basic Perceptron



x1
x1
x1
x1
w1
w2
w3
w4
$\sum_{i=1}^{m}$ (wixi)+bias
f(x)= 1, if ∑ wx+b≥0 ; 0, if ∑ wx+b<0
ŷ
Inputs
Weights
Summation and Bias
Activation
Output

Multi-Layer Perceptron

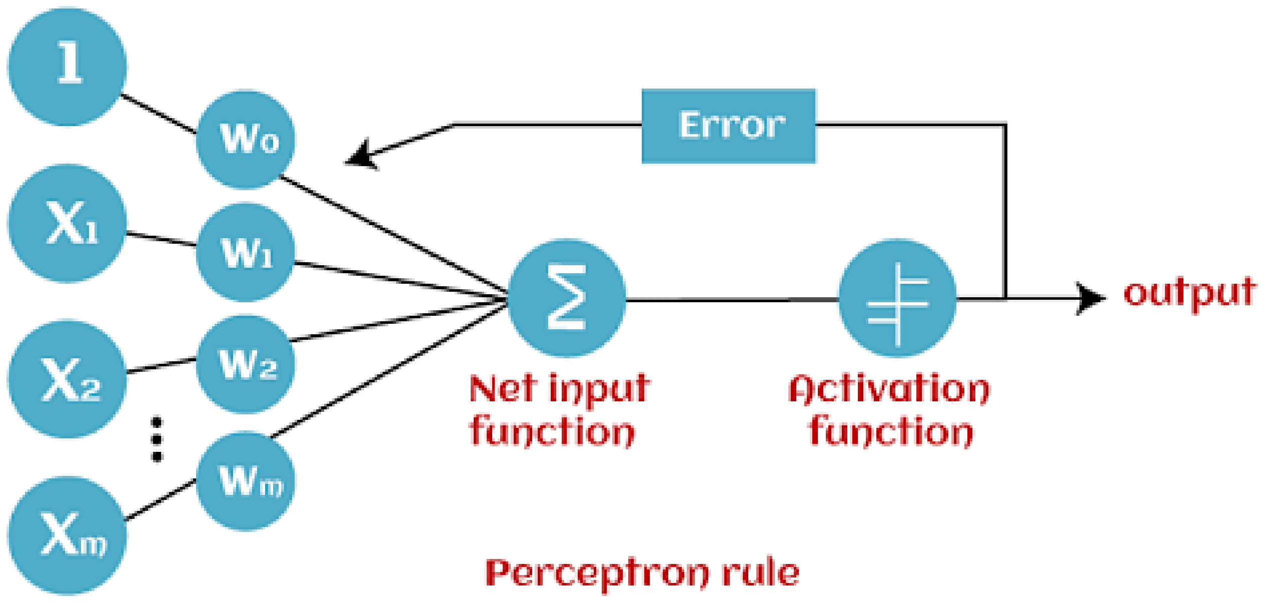


Perceptron: Working

The summation function multiplies the input features $\boldsymbol{X}$ $(\boldsymbol{Where, X} = [\boldsymbol{1, x_1, x_2, \dots, x_m}]$ with weights $\boldsymbol{W}$ $(\boldsymbol{Where\ W} = [\boldsymbol{w_0, w_1, \dots, w_m}]$ and sum them up as follows.

$$Z = w_0 + w_1x_1 + w_2x_2 + \ \dots + w_mx_m$$

In this case w_0 will act as the **threshold or the bias term** and the x_0 is always put as +1.

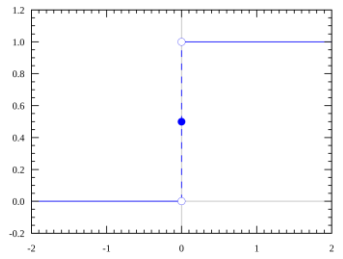


Perceptron: Working

- Perceptron Learning Rule states that the **algorithm would automatically learn the optimal weight coefficients.** The input features are then multiplied with these weights to determine **if a neuron fires or not.**
- The Perceptron receives multiple input signals, and **if the sum of the input signals exceeds a certain threshold**, it either outputs a signal or does not return an output.

Assuming the Heaviside Step Function,

$$h(z) = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; Otherwise \end{cases}$$



Without the bias term,

$$h(z) = \begin{cases} 1 \; if \; w_1x_1 + w_2x_2 \ldots + w_3x_3 > \theta \\ 0 \; Otherwise \end{cases}$$

For convenience we can bring the θ the threshold term to left and $\mathrm{w}_0 = -\theta$ and then,

$$h(z) = \begin{cases} 1 \; if \; w_0 + w_1x_1 + w_2x_2 \ldots + w_3x_3 > 0 \\ 0 \; Otherwise \end{cases}$$

Perceptron: Working

- The output of the perceptron can be (0, 1) or (-1, +1) based on the **activation function** incorporated.
- **Furthermore, during training, the perceptron's weights are adjusted to minimize the difference between the predicted output and the actual output.** Usually, supervised learning algorithms like the **delta rule** (delta rule is a gradient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network) or **perceptron learning rule** are used for this.

$$w_j := w_j + \alpha\left(y^{(i)} - h\left(x^{(i)}\right)\right)x_j^{(i)}$$

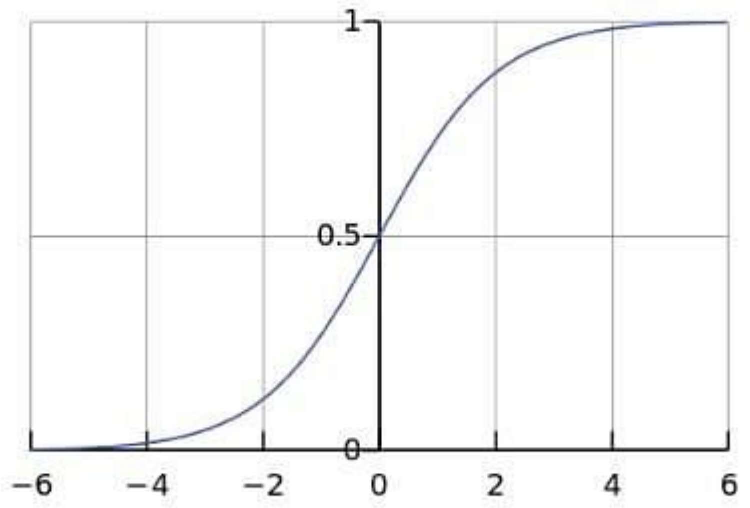
Where $x_j^{(i)}$ is the input value and α is the learning rate, and the $y^{(i)}$ and $h(x^{(i)})$ are actual and predicted values, respectively.

Perceptron: Activation Functions

- **Sigmoid Activation Function**
  - Also known as logistic function.
  - Useful as an activation function when one is interested in probability mapping.
  - Can lead to slow learning and the model getting trapped in local minima during training.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

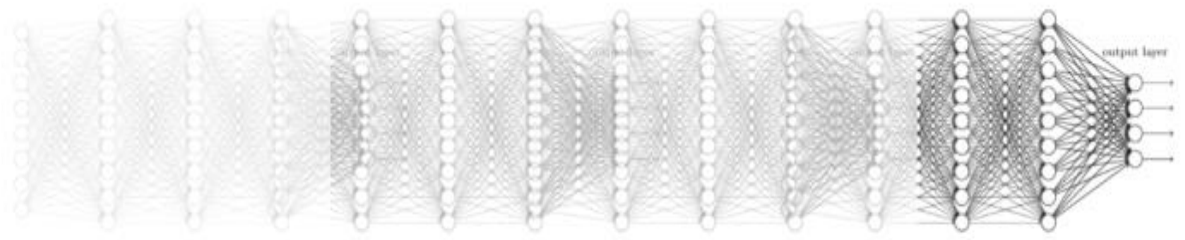
  - Furthermore, this is subjected to vanishing gradient problem in deep neural networks and not a zero-centric function (always give positive values)



Info

Vanishing Gradient Problem

- The vanishing gradient problem is a phenomenon that occurs during the training of deep neural networks, **where the gradients that are used to update the network become extremely small or "vanish" as they are backpropagated from the output layers** to the earlier layers.

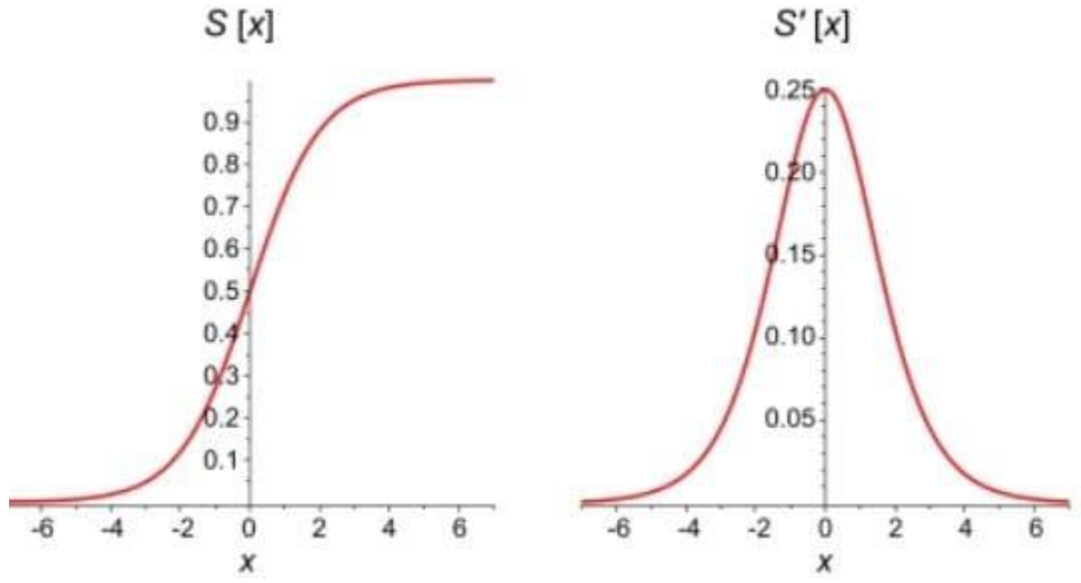


During the training process of the NNs are trying to minimize the loss function by adjusting the weights and the **backpropagation algorithm** computes these gradients by propagating error from the output layer to the input.

Because of the vanishing gradient problem, the NNs may display slow convergence and may get stuck at local minima while indicating impaired learning of deep representations.

Perceptron: Activation Functions

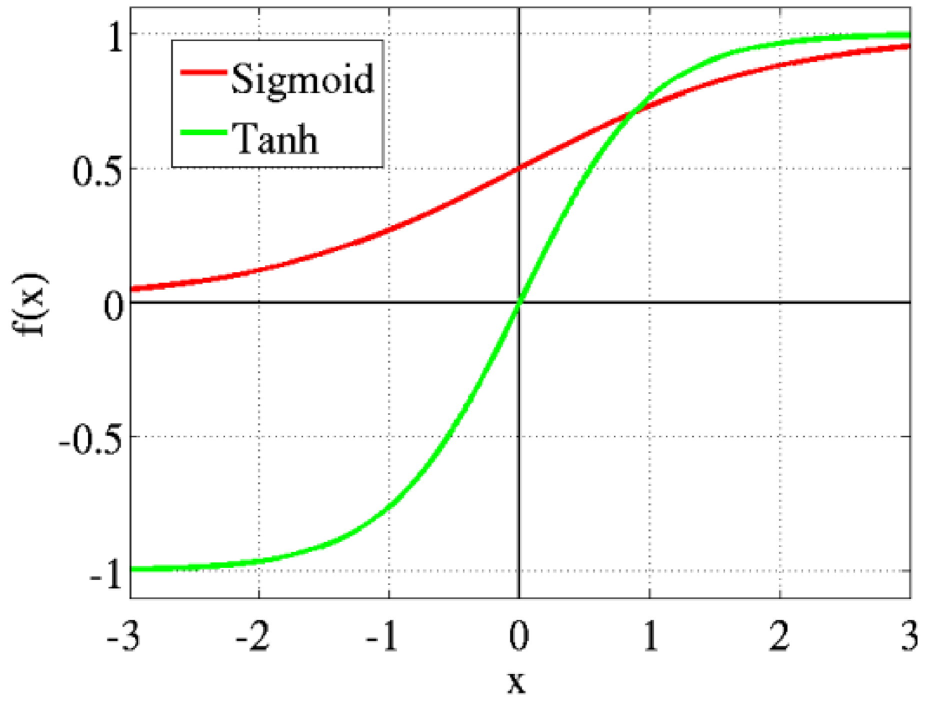
Sigmoid Activation Function Cont.



Derivative of the Sigmoid Function

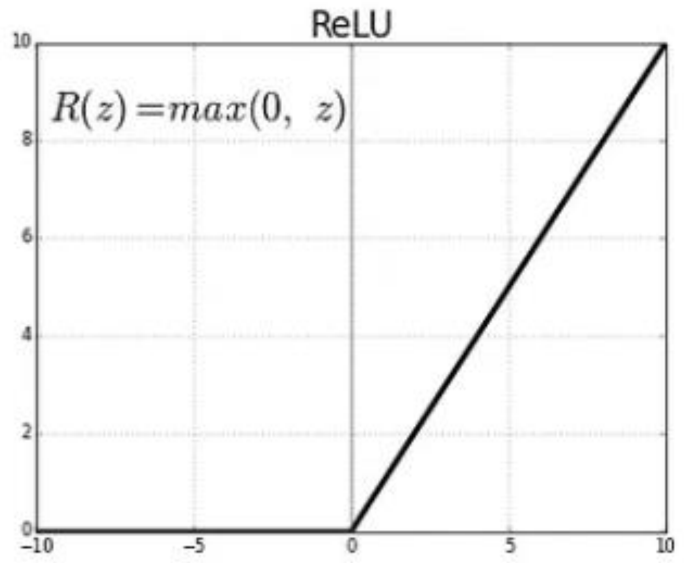
Perceptron: Activation Functions

- **Tanh or hyperbolic tangent Activation Function**
  - The range of the tanh function is from (-1 to 1).
  - This is a zero-centric function which removes the requirement for center the data under certain cases.
  - However, this can be still subjected to the vanishing gradient issue and can be computationally more expensive.



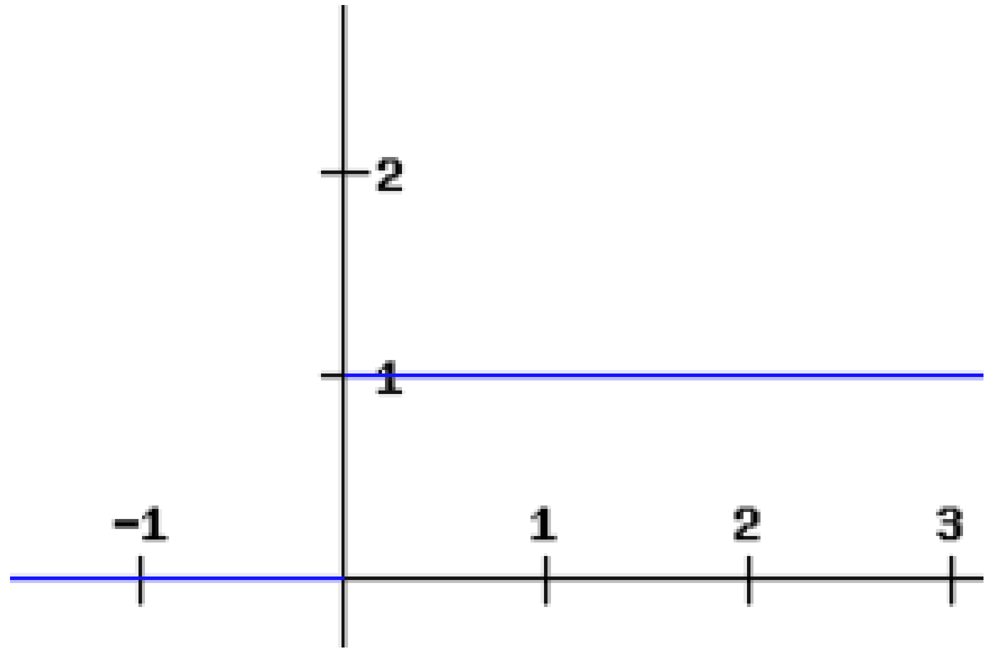
Perceptron: Activation Functions

- **ReLU (Rectified Linear Unit) Activation Function.**
  - The ReLU is the most used activation function in the ANN at the moment.
  - As of the graph, this function ranges from 0 to +infinity.
  - The ReLU function is non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
  - It does not activate all the neurons at the same time. Since the output of some neurons is zero, only a few neurons are activated making the network sparse, efficient, and easy for computation.
  - **Reduce the vanishing gradient issue.**



Perceptron: Activation Functions

ReLU Activation Function Cont.

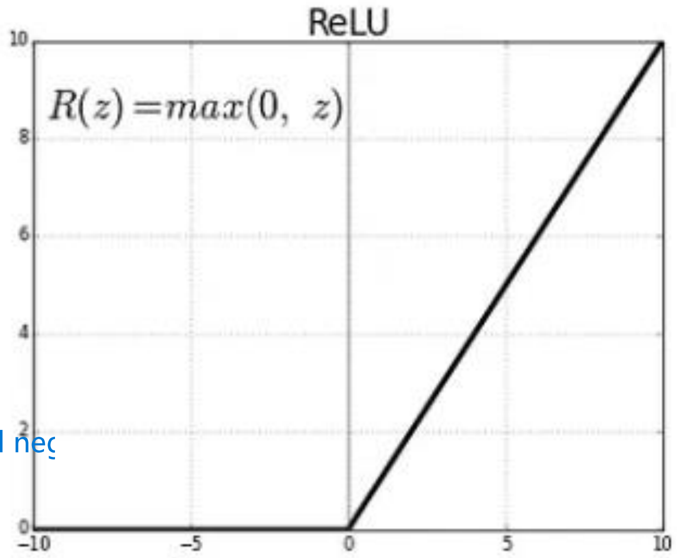


Derivative of the ReLU Function

Perceptron: Activation Functions

- **ReLU (Rectified Linear Unit) Activation Function Cont.**
  - Non-differentiable at zero.
  - **The gradients for negative input are zero**, which means for activations in that region, the weights are not updated during backpropagation. This can create **dead neurons** that never get activated. **This can be handled by reducing the learning rate and bias.**
  - ReLU output is not **zero-centered and it does hurt the neural network performance**.
  - The mean value of activation is not zero.

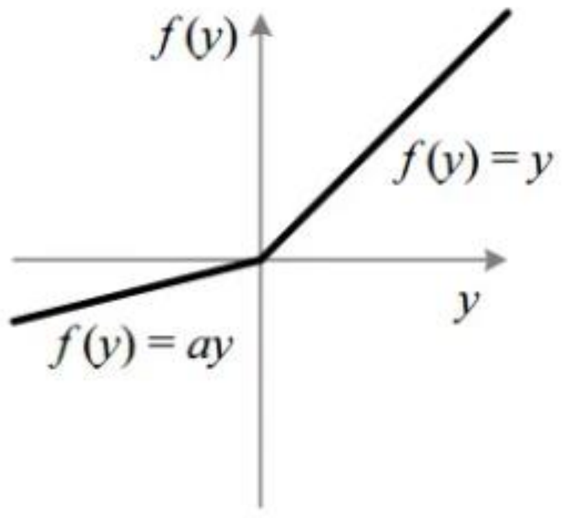
batchnorm : Keeps activations near zero mean rather than being all neg



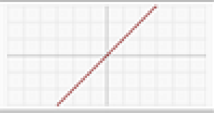
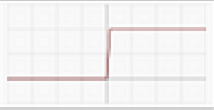
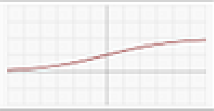
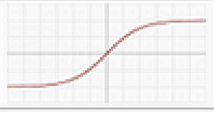
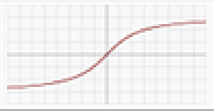
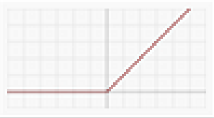
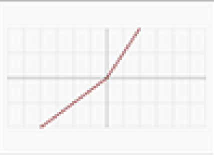
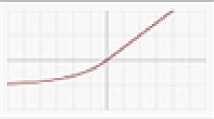
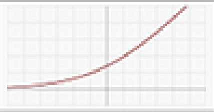
In practice, the non–zero-centered issue is less damaging than vanishing gradients, hence pre
over sigmoid/tanh.

Perceptron: Activation Functions

- **Leaky ReLU Activation Function**
  - With this function it has been attempted to solve the dying ReLU problem.
  - Range: -Infinity to +infinity.
  - If the output of a ReLU is consistently 0 (for example, if the ReLU has a large negative bias), then the gradient through it will consistently be 0. **The error signal backpropagated from later layers gets multiplied by this 0, so no error signal ever passes to earlier layers. The ReLU has died.**
  - **No dying ReLU units** in this case and speed up the training process, however, this can be saturate for large negative values.



Perceptron: Activation Functions

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

Perceptron

- Research and identify the advantages, disadvantages and limitations of the single layer and multi-layer Perceptions.

Learning AND OR XOR gates with single layer perceptron and
layer perceptron

Minimum number of perceptrons to learn XOR?

Try to find the corresponding weights of the multilayer percept

Requirement of the Non-Linear Activation Functions

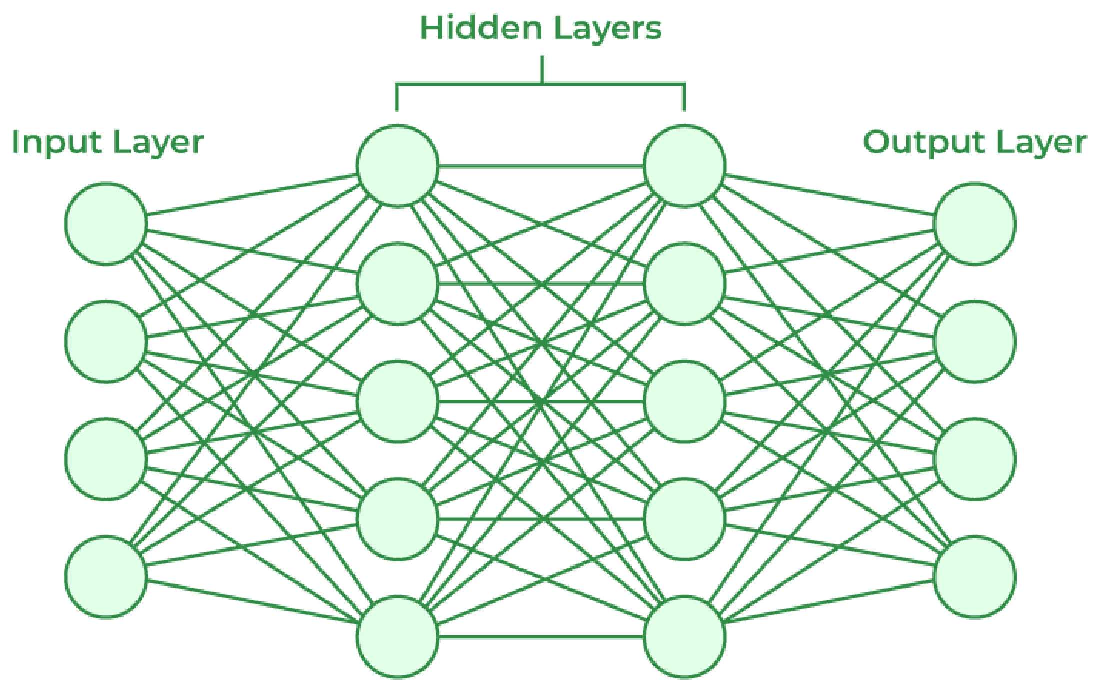
- If you are utilizing only the linear/identity activation functions, the NN also outputs a linear function.
- It will be not capable of learning non-linearities in data even with hundreds of hidden layers.

A text generated image from Bing image creator powered by DALLE-3

Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN)

- A neural network is **a machine learning program, or model, that makes decisions in a manner similar to the human brain**, by using processes that **mimic the way biological neurons work together to identify phenomena**, weigh options, and arrive at conclusions.
- ANNs consist of artificial neurons/nodes/units, and these are arranged in a series of layers.
- A layer may consist of up to millions of artificial neurons.
- In general, the ANNs have an input layer, output layer, and hidden layer/s.



ANN: Working

- The structures and operations of human neurons serve as the basis for artificial neural networks.
- As mentioned in the previous slide there are three types of layers:
  - **Input Layer:** Information from the outside world enters the artificial neural network from the input layer. Input nodes process the data, analyze or categorize it, and pass it on to the next layer.
  - **Hidden Layer/s:** Hidden layers take their input from the input layer or other hidden layers. Artificial neural networks can have a large number of hidden layers. Each hidden layer analyzes the output from the previous layer, processes it further, and passes it on to the next layer.
  - **Output Layer:** The output layer gives the final result of all the data processing by the artificial neural network. It can have single or multiple nodes. For instance, if we have a binary (yes/no) classification problem, the output layer will have one output node, which will give the result as 1 or 0. However, if we have a multi-class classification problem, the output layer might consist of more than one output node.

ANN: Working

- The neurons in one layer are usually connected with the neurons of the subsequent layer. These connections are assigned with **weights** which indicates **the influence of one neuron on the other.**
- **As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.**
- Most deep neural networks are feedforward, **meaning data flows in one direction only, from input to output.**
- However, **the learning in ANN** happens through an algorithm known as **backpropagation**; that is, it moves in the opposite direction from output to input. **Backpropagation allows us to calculate and attribute the error associated with each neuron**, allowing us to adjust and fit the parameters of the model(s) appropriately.

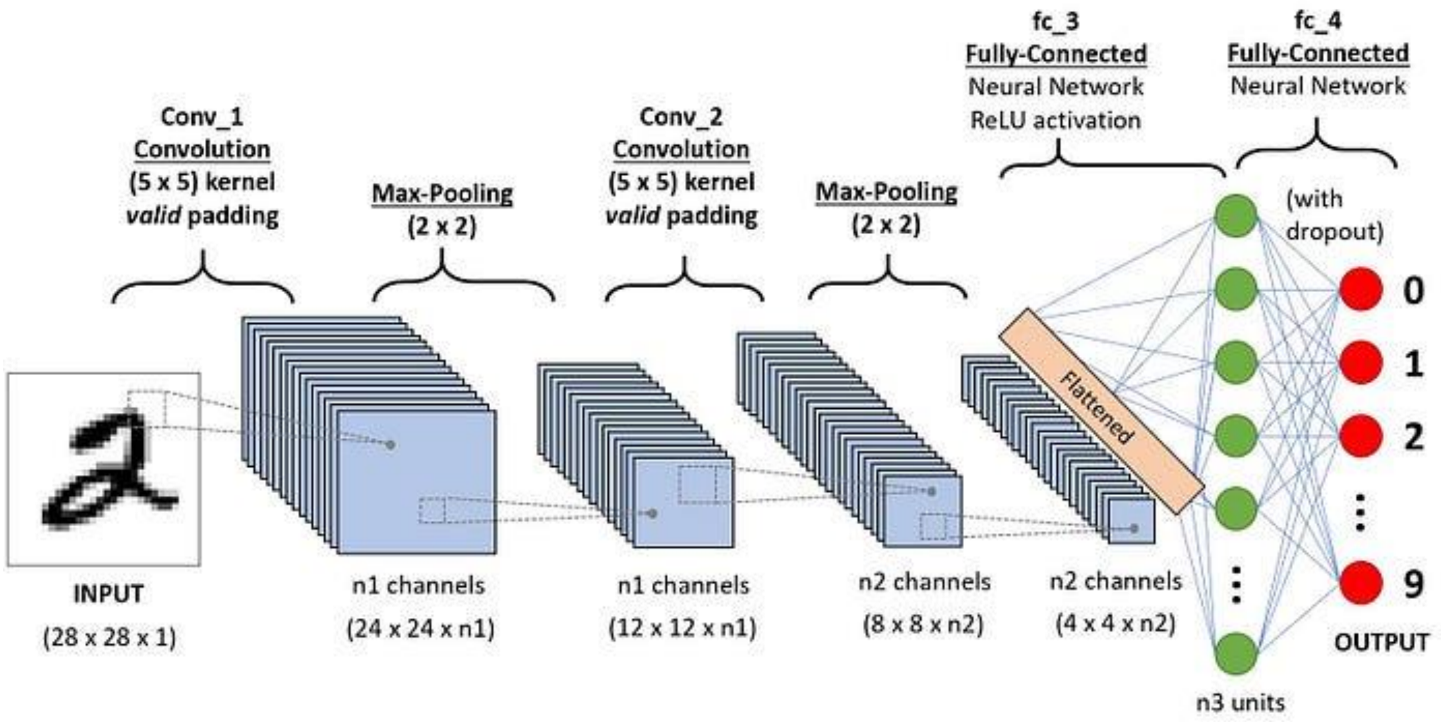
ANN: Types

- Perceptron is the oldest and most primary type of neural network which consist of only one neuron.
- Up until now we are talking about the **feedforward neural networks,** and these are commonly known as the **multi-layer perceptron** or MLPs.

ANN: Types – Convolutional Neural Networks

Mainly utilized in **Computer Vision** related applications.

More information will be provided under the Artificial Intelligence module next semester…

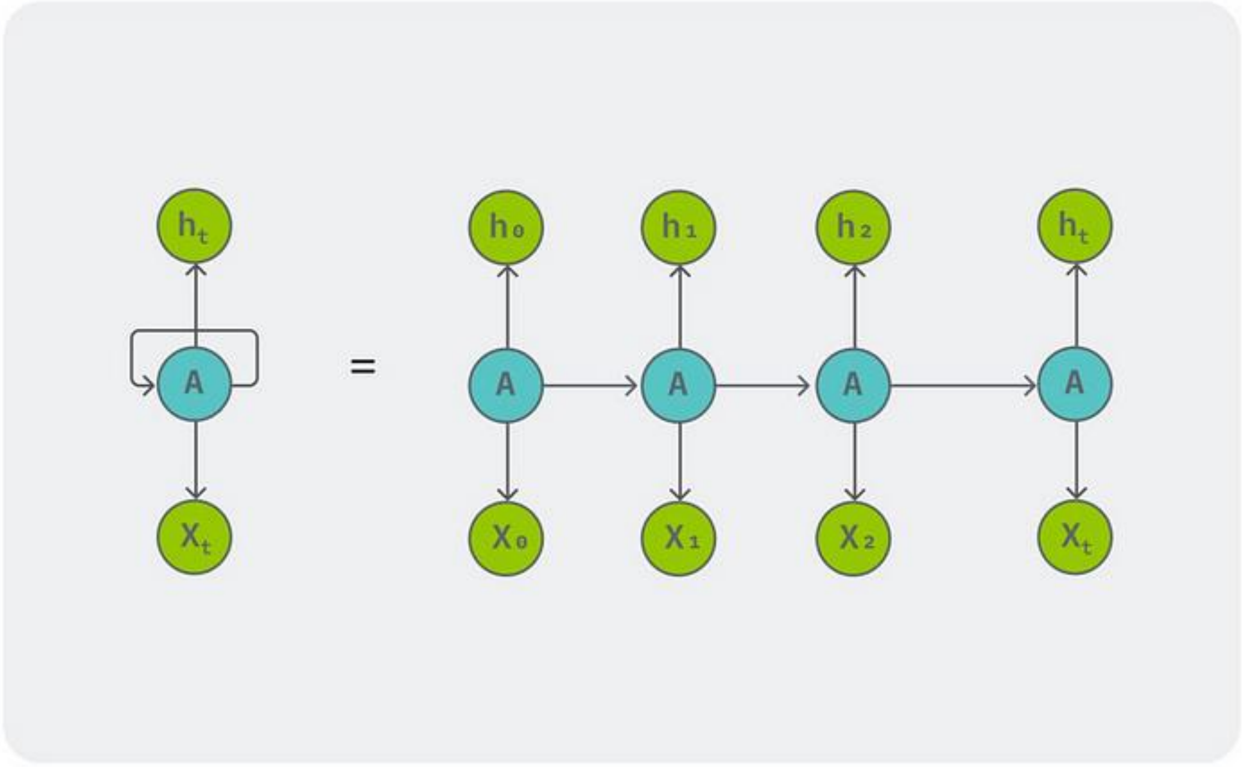


ANN: Types - Recurrent Neural Networks (RNN)

RNN are mainly utilized to process **sequential data** like, text, speech or music.

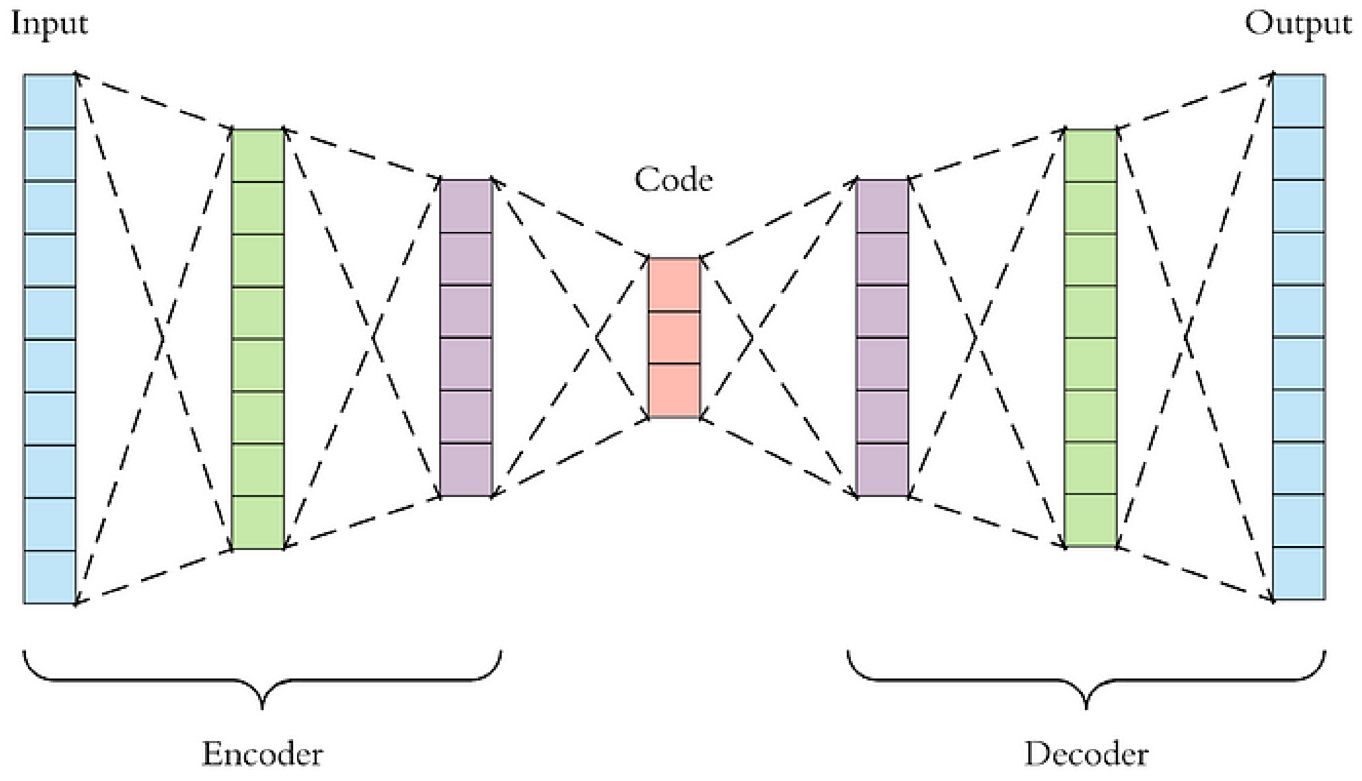
- Other RNN Architectures):
  - **BRNN** (Bidirectional recurrent neural networks)
  - **LSTM** (Long Short-Term Memory).
  - **GRU** (Gated Recurrent Units)

More information will be provided under the Artificial Intelligence module next semester…



ANN: Types – Autoencoders

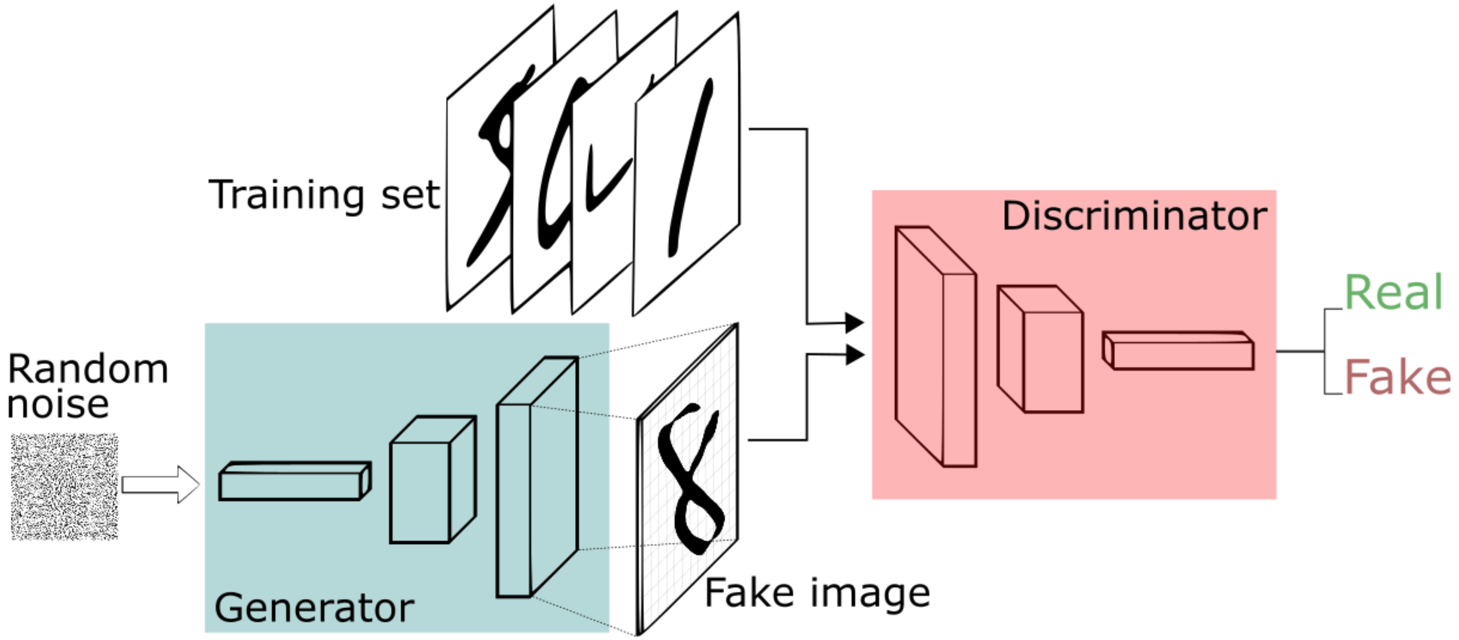
Autoencoders are a specific type of feedforward neural networks where the **input is the same as the output (Not exactly! There are losses…)**. They compress the input into a lower-dimensional code and then reconstruct the output from this representation. The code is a compact "summary" or "compression" of the input, also called the latent-space representation.



ANN: Types - Generative Adversarial Networks (GANs)

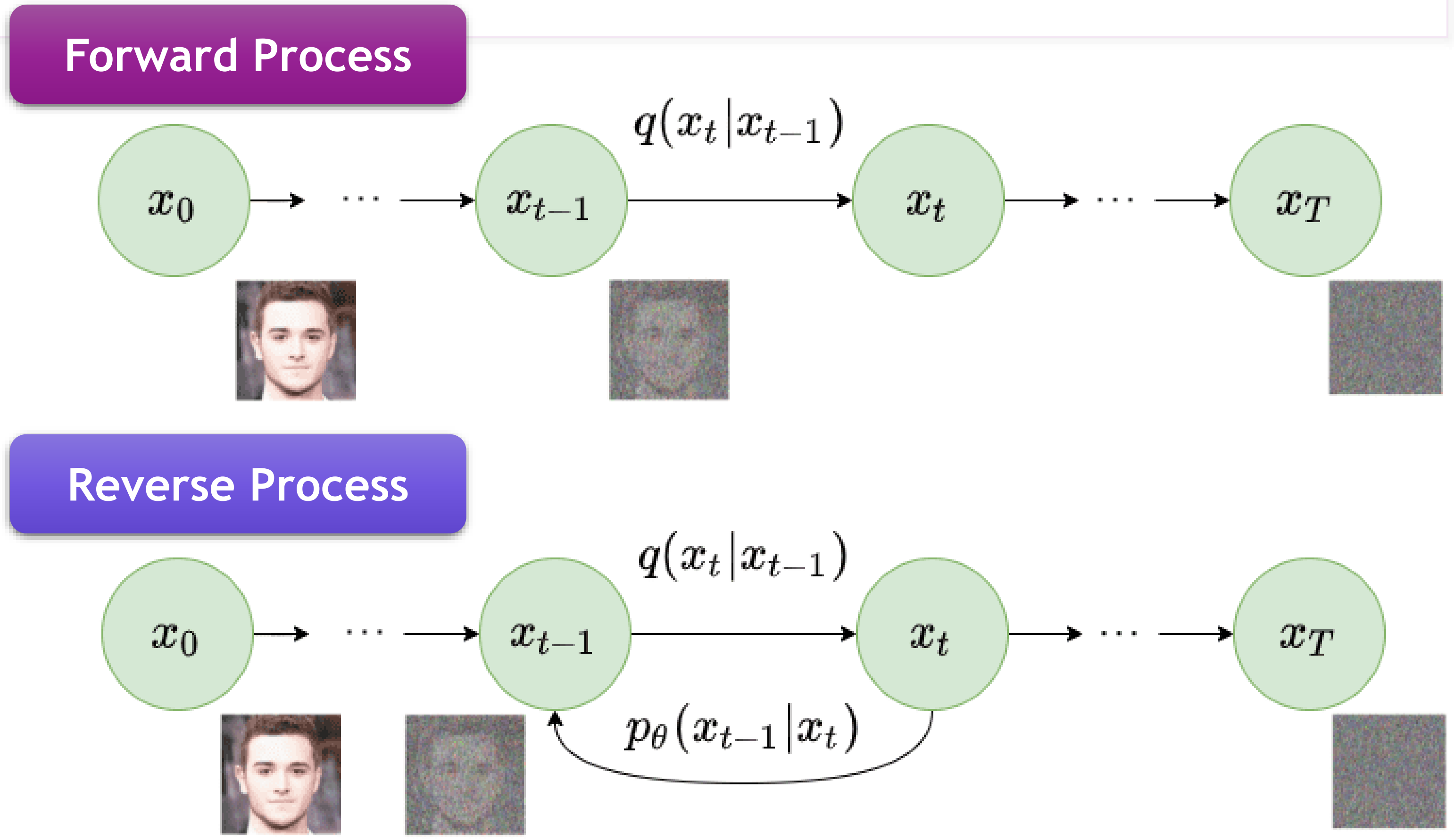
A generative adversarial network (GAN) has two parts:

- **The generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- **The discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.



ANN: Types - Diffusion Networks

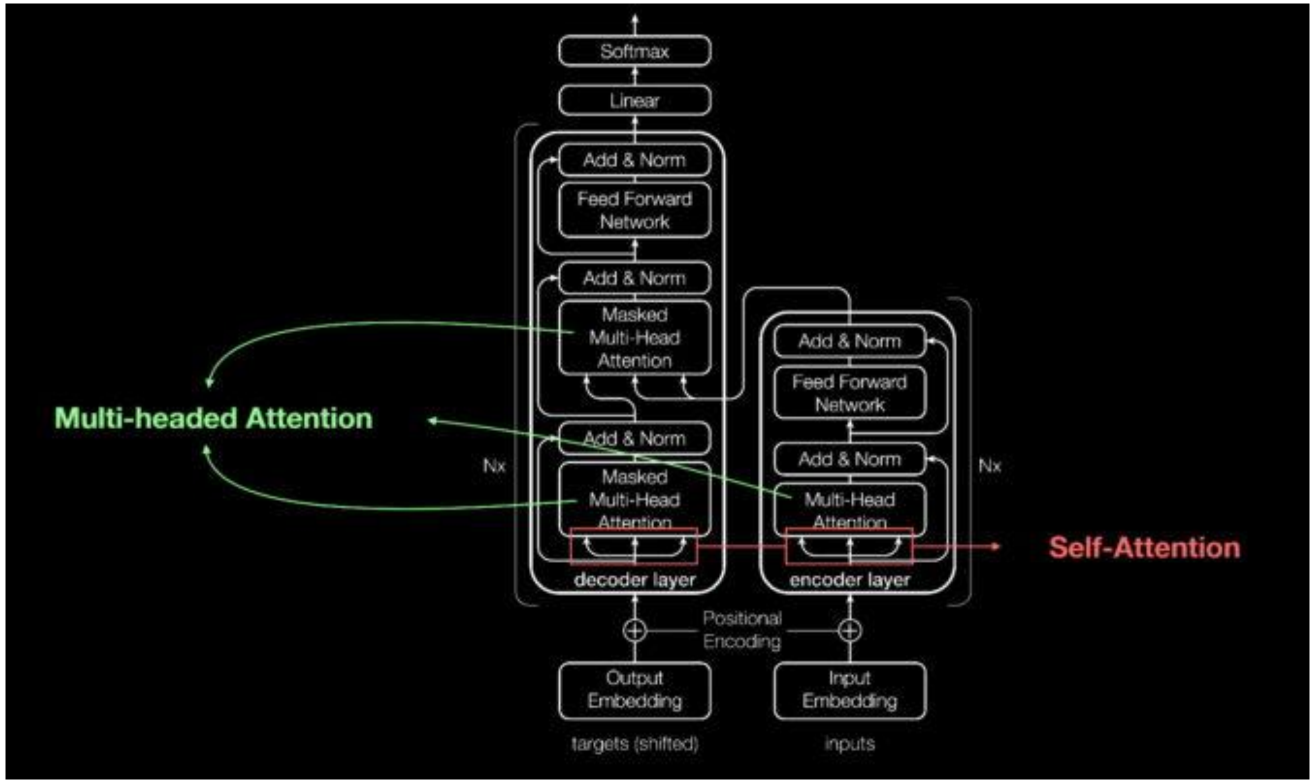
- Under the forward process of the diffusion networks, Gaussian noise is added to an image through a series of steps.
- Then under the backward process, the neural network is trained to recover the original image by reversing the noising process.
- Being able to model the reverse process we will be able to generate new data e.g., images).



ANN: Types – Transformers

A transformer model is a neural network **that learns context and thus meaning by tracking relationships in sequential data** like the words in this sentence. Transformer models apply an evolving set of mathematical techniques, called **attention or self-attention**, to detect subtle ways even distant data elements in a series influence and depend on each other.

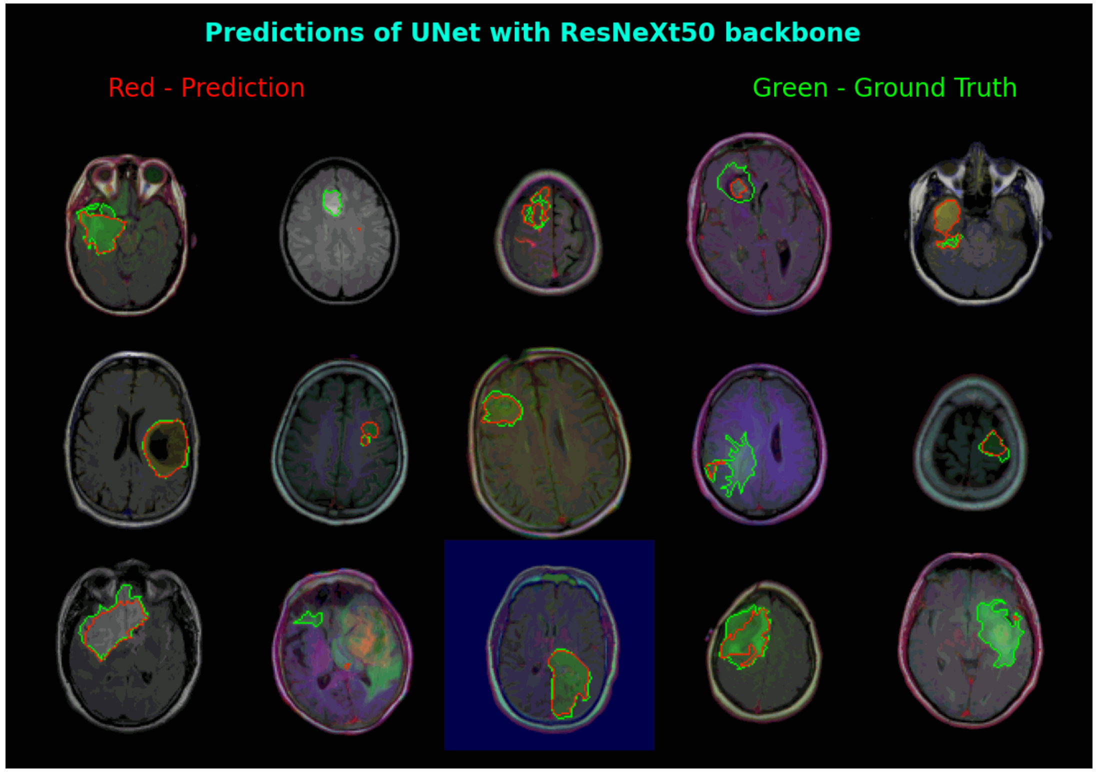
Can apply for the images with other flavors of transformers such as **Vision Transformer**



ANN: Applications

- Nowadays, the neural networks utilized in a vast spectrum of industrial and academical use-cases. Some of them can me mentioned as follows:
  - Medical diagnosis by medical image classification.
  - Targeted marketing by social network filtering and behavioral data analysis.
  - Financial predictions by processing historical data of financial instruments.
  - Electrical load and energy demand forecasting.
  - Process and quality control.
  - Chemical compound identification.

Example: Medical Image Segmentation with Deep Neural Networks



ANN: Applications - Computer Vision

system detects facial landmarks (e.g., eye corners, nose tip, mouth edges — typically 68 o
468 points), it often visualizes them using a triangulated mesh.

468-point landmarks from MediaPipe Face Mesh

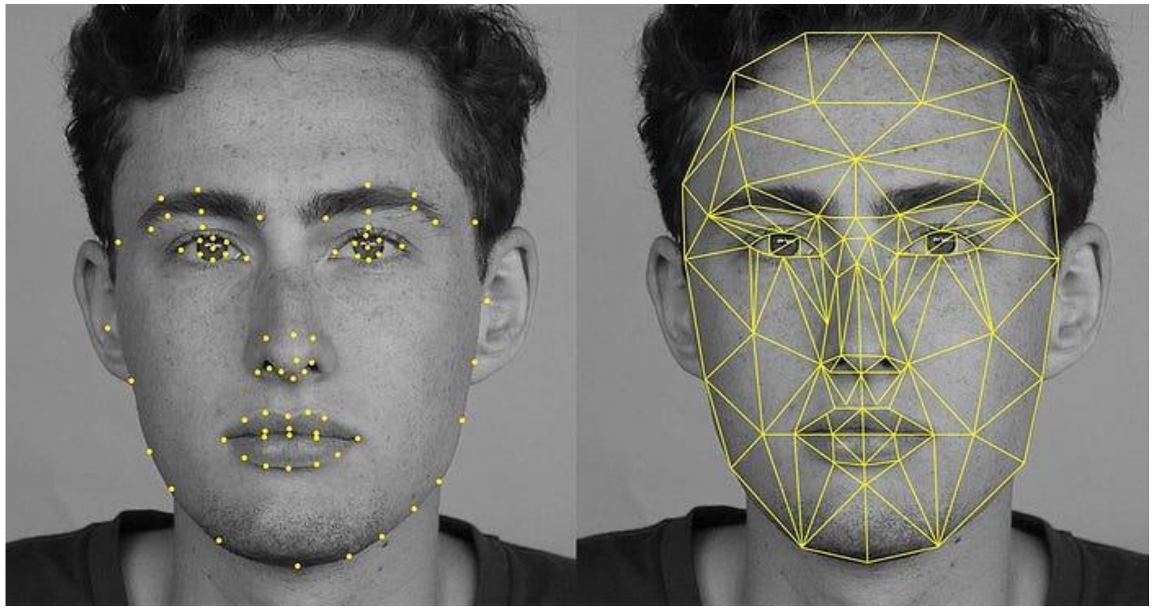
those triangles define the 2D topology that corresponds to a 3D face model.

(3D reconstruction (estimating face shape from a single image), Pose estimation (yaw, pit
roll), Expression transfer, Texture mapping (projecting skin texture onto a 3D mesh)

- Computer vision is the ability of computers to extract information and insights from images and videos. With neural networks, computers can distinguish and recognize images.
  - Self-driving cars.
  - Facial recognition.
  - Image labeling.

3D face reconstruction

-Detect facial landmarks in 2D.
- them to match a canonical 3D model (like the Basel Face Model).
-Estimate the camera projection matrix (so the 3D → 2D mapping fits the ir
- Reconstruct 3D vertex positions by optimizing shape and pose parameter



ANN: Applications – Natural Language Processing & Speech recognition

- **Natural language processing (NLP)** is the ability to process natural, human-created text. Neural networks help computers gather insights and meaning from text data and documents. This has been used in:
  - Automated virtual agents and chatbots.
  - Document summarization and article generation for a given topic.
  - Indexing of key phrases that indicate sentiment, like positive and negative comments on social media.
- Neural networks can analyze human speech despite varying speech patterns, pitch, tone, language, and accent. Virtual assistants like Amazon Alexa and automatic transcription software use **speech recognition** to do tasks like these:
  - Assist call center agents and automatically classify calls.
  - Convert clinical conversations into documentation in real-time.
  - Accurately subtitle videos and meeting recordings for wider content reach.

ANN: Applications – Recommendation Engines

- Neural networks can track user activity to develop personalized recommendations. They can also analyze all user behavior and discover new products or services that interest a specific user.

Good as an exam qustion?

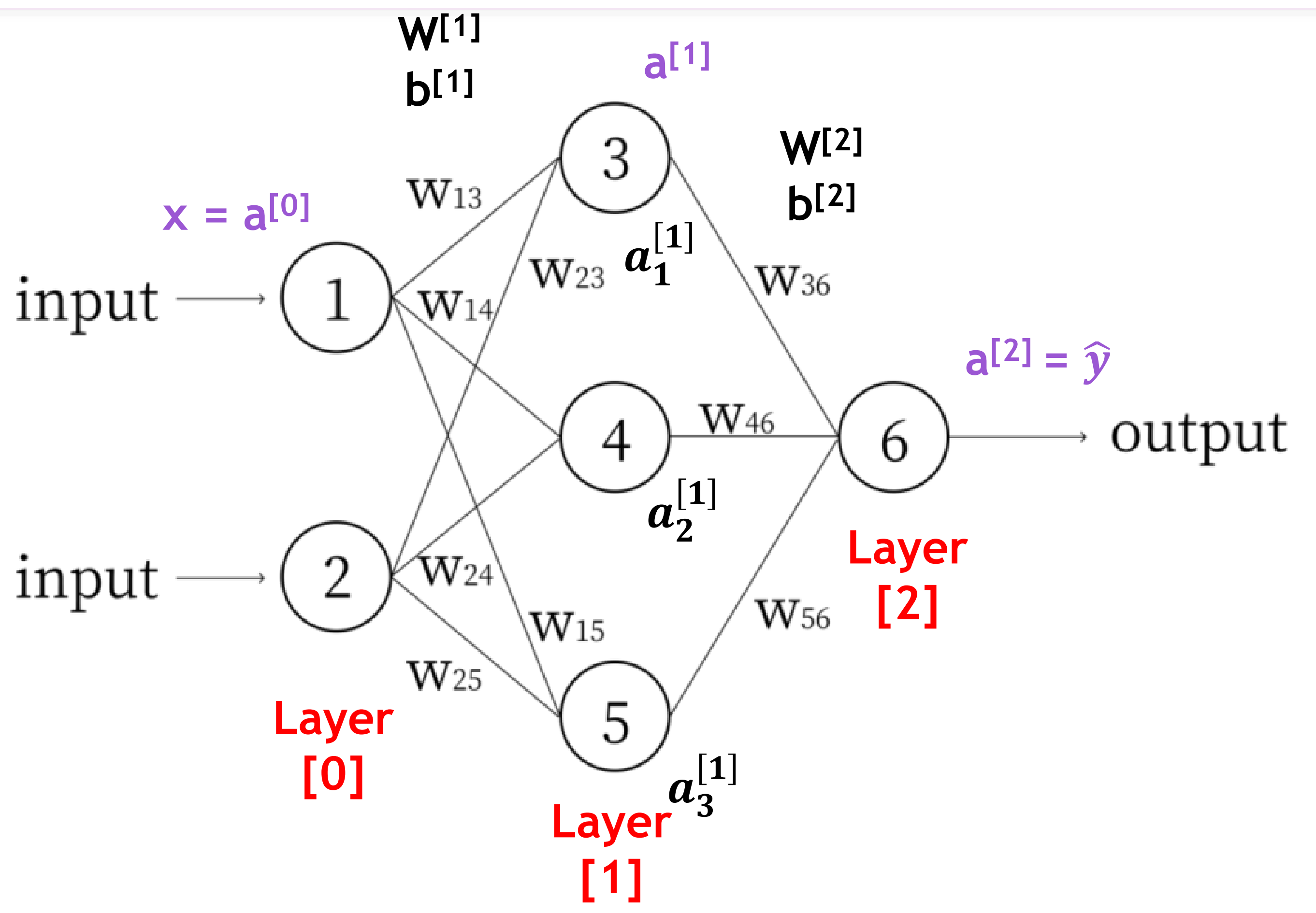
* Feature based user embedding (age , geo-location, gender)
* Behavior based user embedding (previous n items , RNN-sequence based)

A text generated image from Bing image creator powered by DALLE-3

ANN: Forward Propagation

This chapter has been mainly composed with the contents presented in Prof. Andrew Ng’s lecture series on **Deep Learning Specialization** which is available in deeplearning.ai website

Forward Propagation: Notations



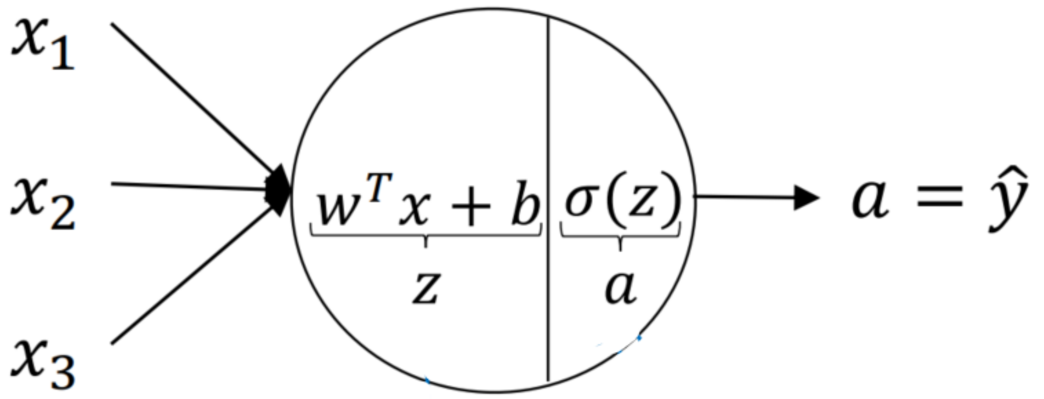
For the case of having one hidden layer and two inputs

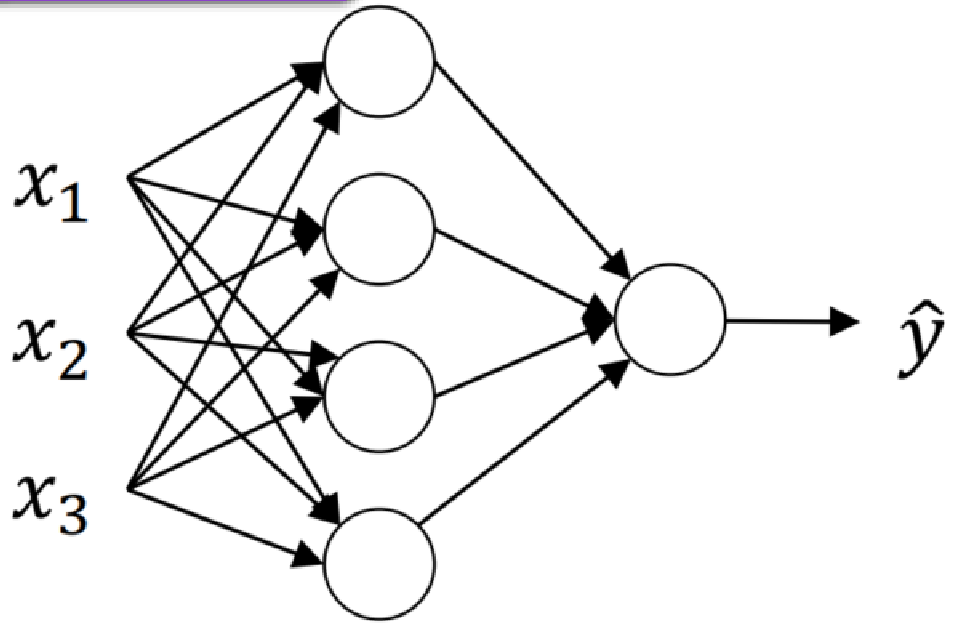
- 'W' indicates the weights
- 'b' indicates the bias under each case.
- The 'a' indicates the output of each node also known as the **activations (not the activation function).**

This can be named as a two-layer neural network by ignoring the input layer for the layer count as a convention.

Forward Propagation: Inner Working

Same thing is happening multiple time inside the NN

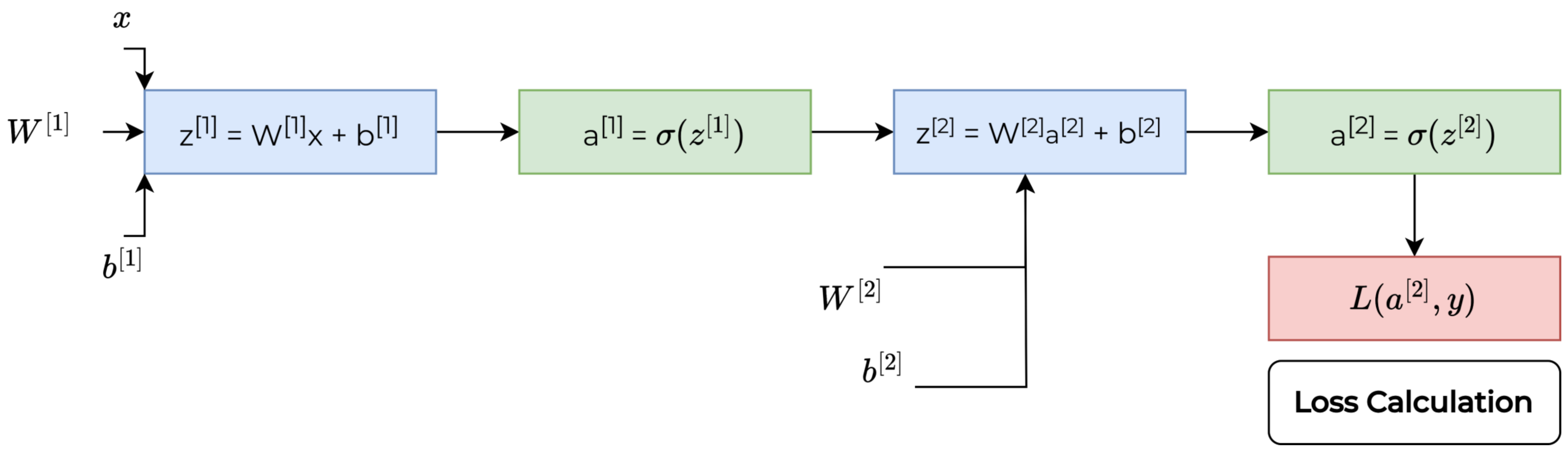




$$z = w^T x + b$$
$$a = \sigma(z)$$

This chapter has been mainly composed with the contents presented in Prof. Andrew Ng's lecture series on **Deep Learning Specialization** which is available in deeplearning.ai website.

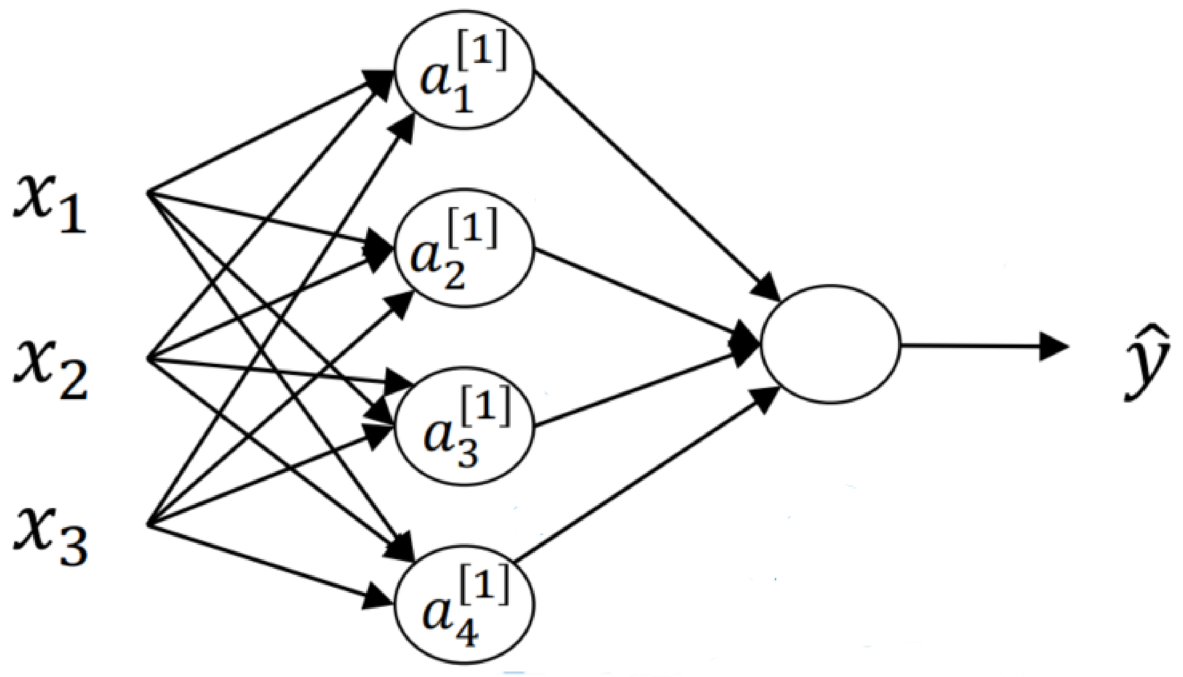
Forward Propagation



x
$W^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = W^{[2]}a^{[2]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$W^{[2]}$
$b^{[2]}$
$L(a^{[2]}, y)$
Loss Calculation

This chapter has been mainly composed with the contents presented in Prof. Andrew Ng's lecture series on **Deep Learning Specialization** which is available in deeplearning.ai website.

Forward Propagation



$\boldsymbol{W^{[1]}}$ $\boldsymbol{b^{[1]}}$ $\boldsymbol{z^{[1]}}$

$$z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}, \qquad a_1^{[1]} = \sigma\left(z_1^{[1]}\right)$$
$$z_2^{[1]} = w_2^{[1]T}x + b_2^{[1]}, \qquad a_2^{[1]} = \sigma\left(z_2^{[1]}\right)$$
$$z_3^{[1]} = w_3^{[1]T}x + b_3^{[1]}, \qquad a_3^{[1]} = \sigma\left(z_3^{[1]}\right)$$
$$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}, \qquad a_4^{[1]} = \sigma\left(z_4^{[1]}\right)$$

$\boldsymbol{z^{[1]}}$ $\boldsymbol{x = a^{[0]}}$ $\boldsymbol{a^{[1]}}$

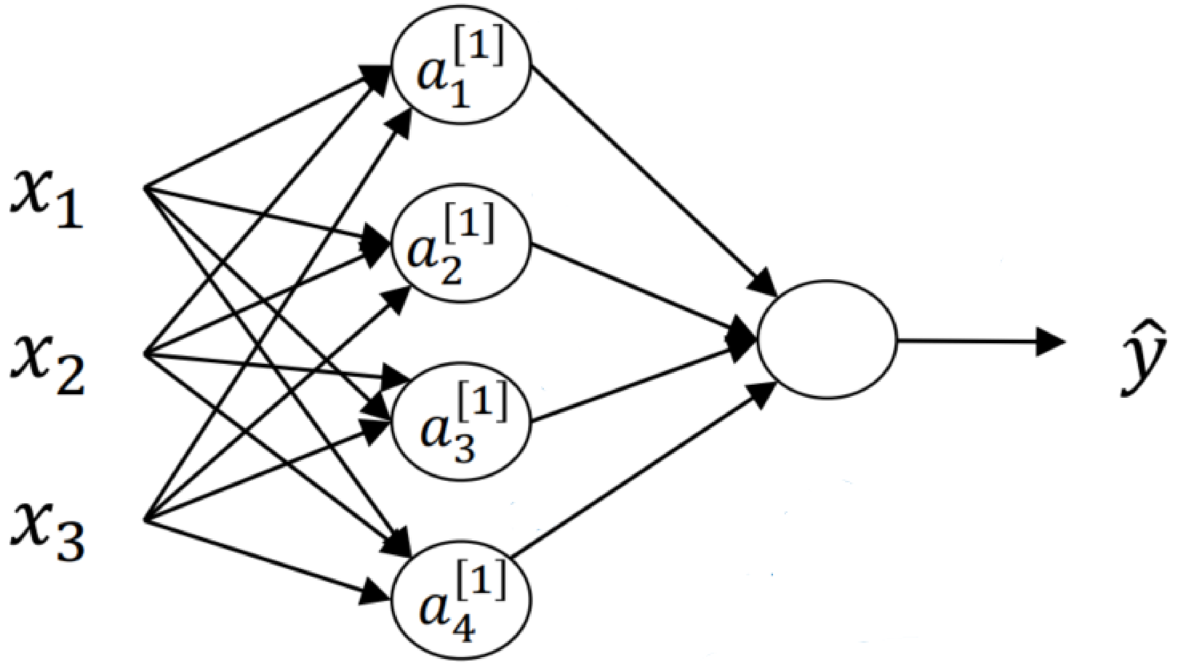
This chapter has been mainly composed with the contents presented in Prof. Andrew Ng's lecture series on **Deep Learning Specialization** which is available in deeplearning.ai website.

Forward Propagation

Considering a single training example

$\boldsymbol{a}^{[0]}$

- $z^{[1]} = W^{[1]}x + b^{[1]}$
- $a^{[1]} = \sigma(z^{[1]})$
- $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
- $a^{[2]} = \sigma(z^{[2]})$



Forward Propagation: Vectorizing Across Multiple Examples

Considering the un-vectorized case:

- *for i = 1 to m:*

$$z^{[1](i)} = W^{[1](i)} x^{(i)} + b^{[1](i)}$$

$$a^{[1](i)} = \sigma\left(z^{[1](i)}\right)$$

$$z^{[2](i)} = W^{[2](i)} a^{[1](i)} + b^{[2](i)}$$

$$a^{[2](i)} = \sigma\left(z^{[2](i)}\right)$$

After vectorizing across m training examples

$$\boldsymbol{Z}^{[1]} = \boldsymbol{W}^{[1]} \boldsymbol{X} + \boldsymbol{b}^{[1]}$$

$$\boldsymbol{A}^{[1]} = \boldsymbol{\sigma}\left(\boldsymbol{Z}^{[1]}\right)$$

$$\boldsymbol{Z}^{[2]} = \boldsymbol{W}^{[2]} \boldsymbol{A}^{[1]} + \boldsymbol{b}^{[2]}$$

$$\boldsymbol{A}^{[2]} = \boldsymbol{\sigma}\left(\boldsymbol{Z}^{[2]}\right)$$

Notice the upper-case annotations for the vectors

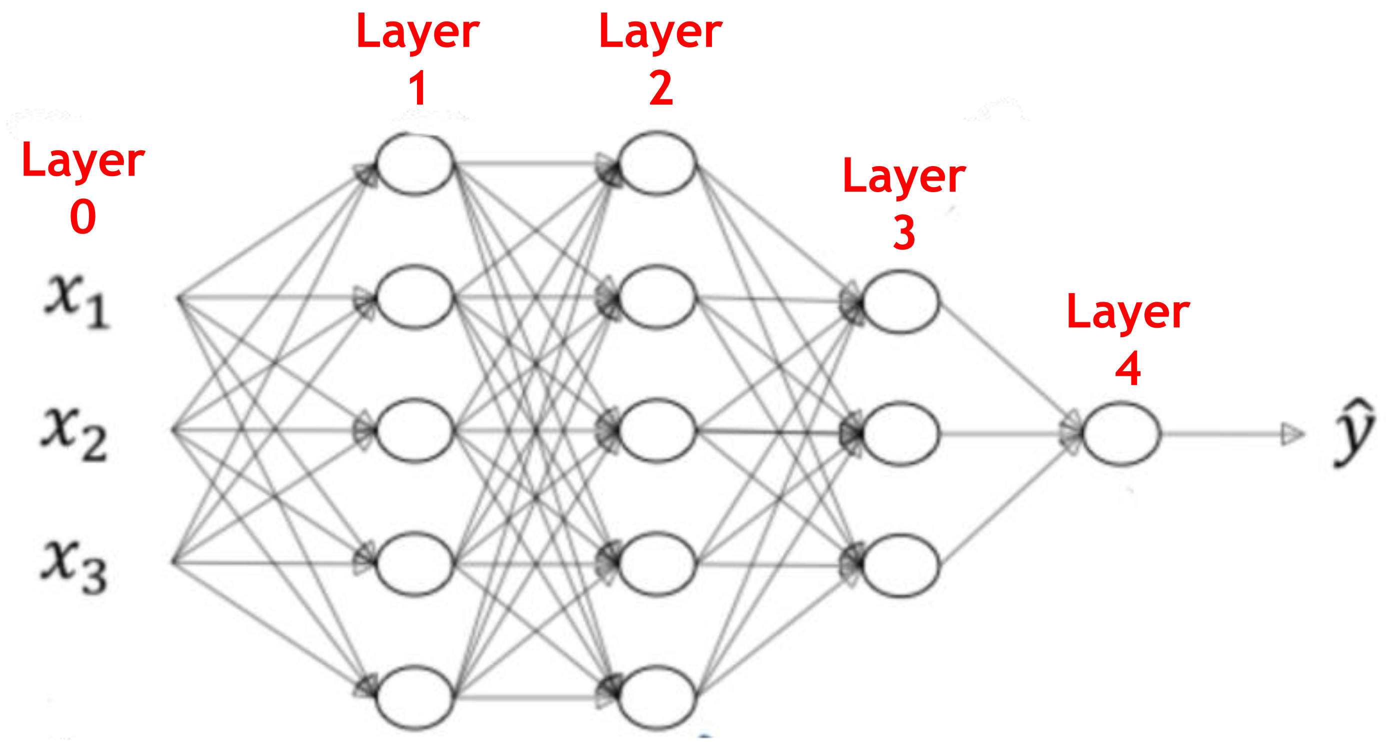
Forward Propagation: Vectorizing Across Multiple Examples

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](1)} & \ldots & z^{[1](1)} \\ | & | & & | \end{bmatrix}$$

Forward Propagation: For MLP



Number of Layers = 3

$n^{[l]}$= **Number of units in layer l**

($n^{[0]}$=3, $n^{[1]}$=5, $n^{[2]}$=5, $n^{[3]}$=3, $n^{[4]}$=$n^{[L]}$=1)

$a^{[l]}$= **Activations in layer l**

This chapter has been mainly composed with the contents presented in Prof. Andrew Ng's lecture series on **Deep Learning Specialization** which is available in deeplearning.ai website.

Forward Propagation: For MLP

Un-Vectorized

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma\left(z^{[1]}\right)$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma\left(z^{[2]}\right)$$

…

$$z^{[4]} = W^{[4]}a^{[3]} + b^{[4]}$$

$$a^{[4]} = \sigma\left(z^{[4]}\right) = \hat{y}$$

Vectorized

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma\left(Z^{[2]}\right)$$

…

$$\hat{Y} = g\left(Z^{[4]}\right) = A^{[4]}$$

A text generated image from Bing image creator powered by DALLE-3

ANN: Backward Propagation

This chapter has been mainly composed with the contents presented in Prof. Andrew Ng's lecture series on **Deep Learning Specialization** which is available in deeplearning.ai website

(Homework)

Backward Propagation

- **Backpropagation** is a **widely used algorithm for training feedforward neural networks.** It computes the gradient of the loss function with respect to the network weights.
- It is very efficient, rather than naively directly computing the gradient concerning each weight.
- This efficiency makes it **possible to use gradient methods to train multi-layer networks** and update weights to minimize loss; variants such as gradient descent or stochastic gradient descent are often used.
- **The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight via the chain rule,** computing the gradient layer by layer, and iterating backward from the last layer to avoid redundant computation of intermediate terms in the chain rule.

Backward Propagation: Stages

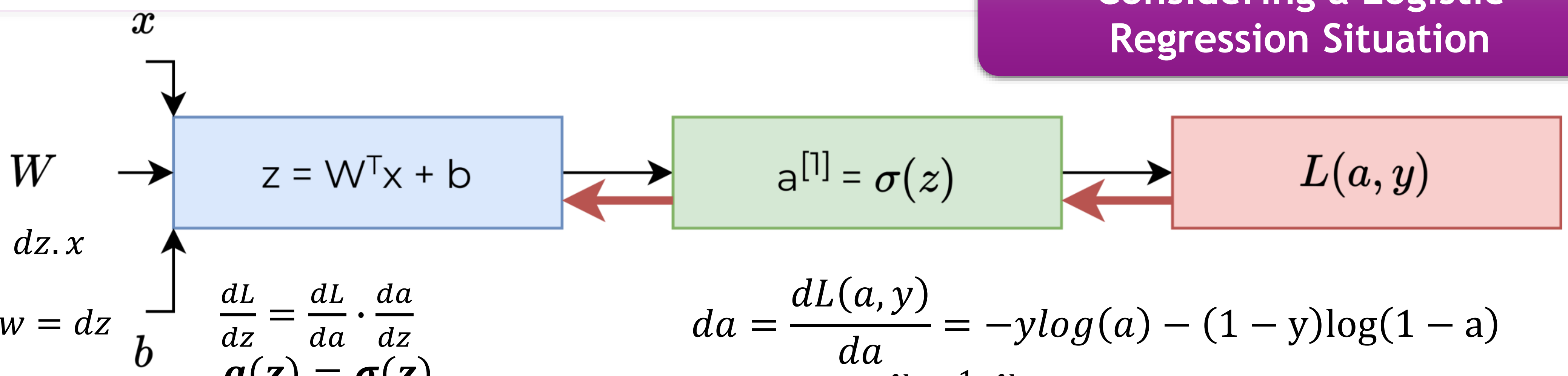
- Three stages in the learning:
  - The feed-forward of input training pattern.
  - The calculation and backpropagation of the error.
  - Updating of the weight

Backward Propagation: Algorithm

1. Inputs X, arrive through the preconnected path.
2. The input is modeled using true weights W. Weights are usually chosen randomly (initially).
3. Calculate the output of each neuron from the input layer to the hidden layer/s to the output layer.
4. Calculate the error in the outputs:
$$Backpropagation\ Error = Actual\ Output\ - Desired\ Output$$
5. From the output layer, go back to the hidden layer to adjust the weights to reduce the error.
6. Repeat the process until the desired output is achieved.

Backward Propagation

Considering a Logistic Regression Situation



$dw = dz.x$

$dw = dz$

$$\frac{dL}{dz} = \frac{dL}{da} \cdot \frac{da}{dz}$$

$\boldsymbol{g(z) = \sigma(z)}$

(Notation for the Activation Function)

$$dz = da \cdot \frac{dg(z)}{d(z)}$$

$$dz = da \cdot g'(z)$$

$dz = a - y$ (After obtaining the derivative of the complete equation using chain rule)

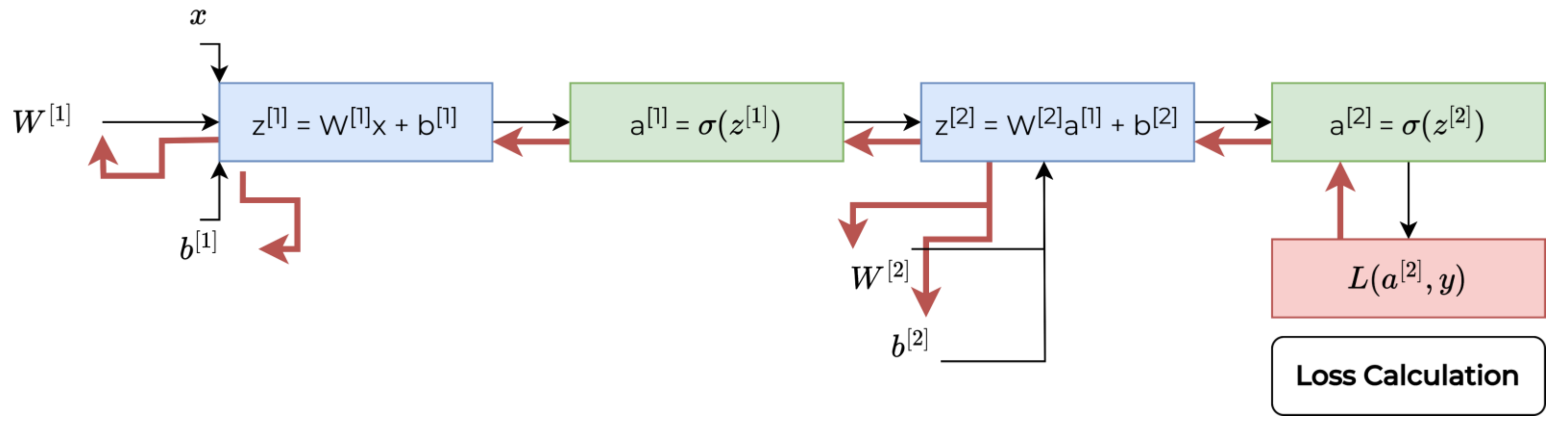
$$da = \frac{dL(a,y)}{da} = -y\log(a) - (1-y)\log(1-a)$$

$$da = -\frac{y}{a} + \frac{1-y}{1-a}$$

This chapter has been mainly composed with the contents presented in Prof. Andrew Ng's lecture series on **Deep Learning Specialization** which is available in deeplearning.ai website.

Backward Propagation: For NN

For Un-Vectorized Case



x
$W^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$W^{[2]}$
$b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$L(a^{[2]}, y)$
Loss Calculation

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]}a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]}x^T$$

$$db^{[1]} = dz^{[1]}$$

Backward Propagation: For NN

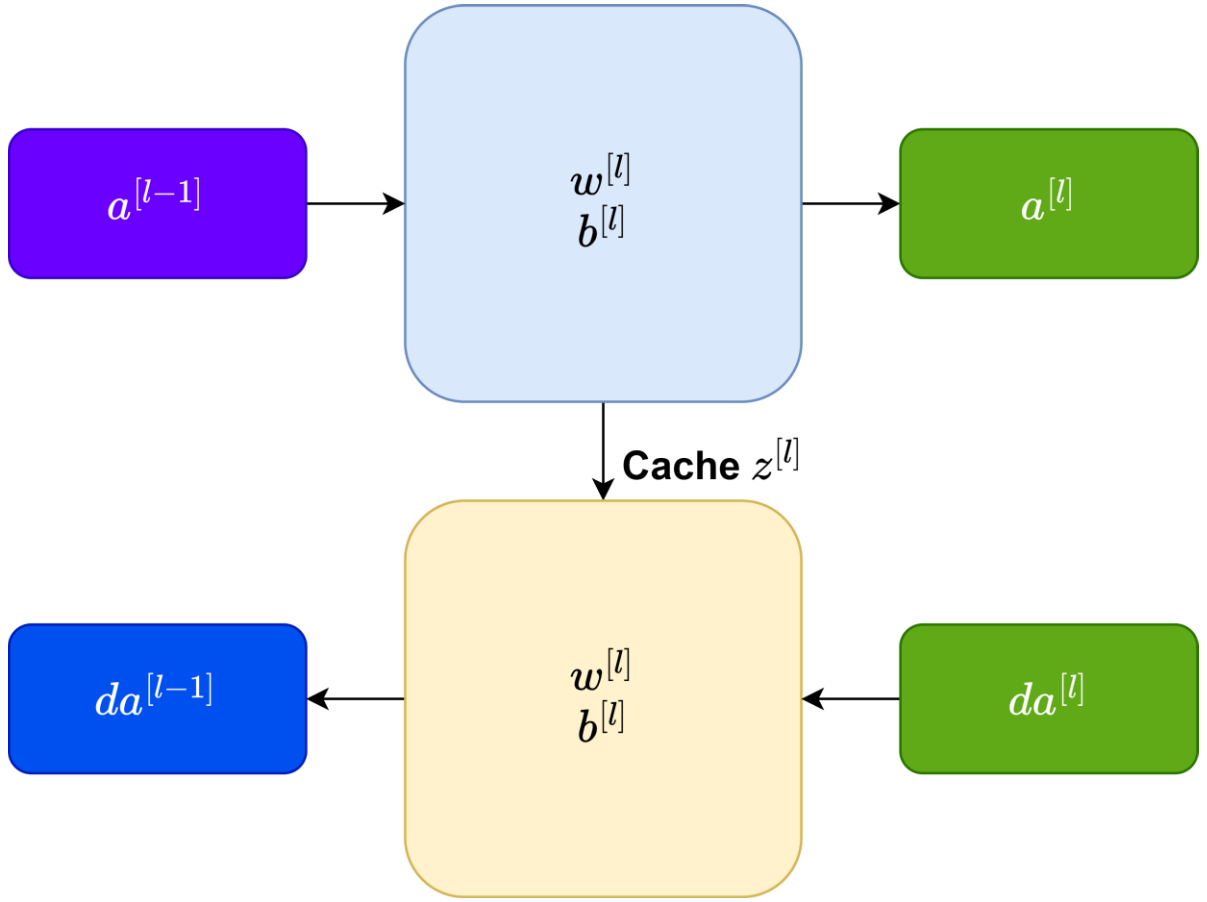
- Forward Function for layer l: $W^{[l]}, b^{[l]}$

  Input: $a^{[l-1]}$, Output: $a^{[l]}$

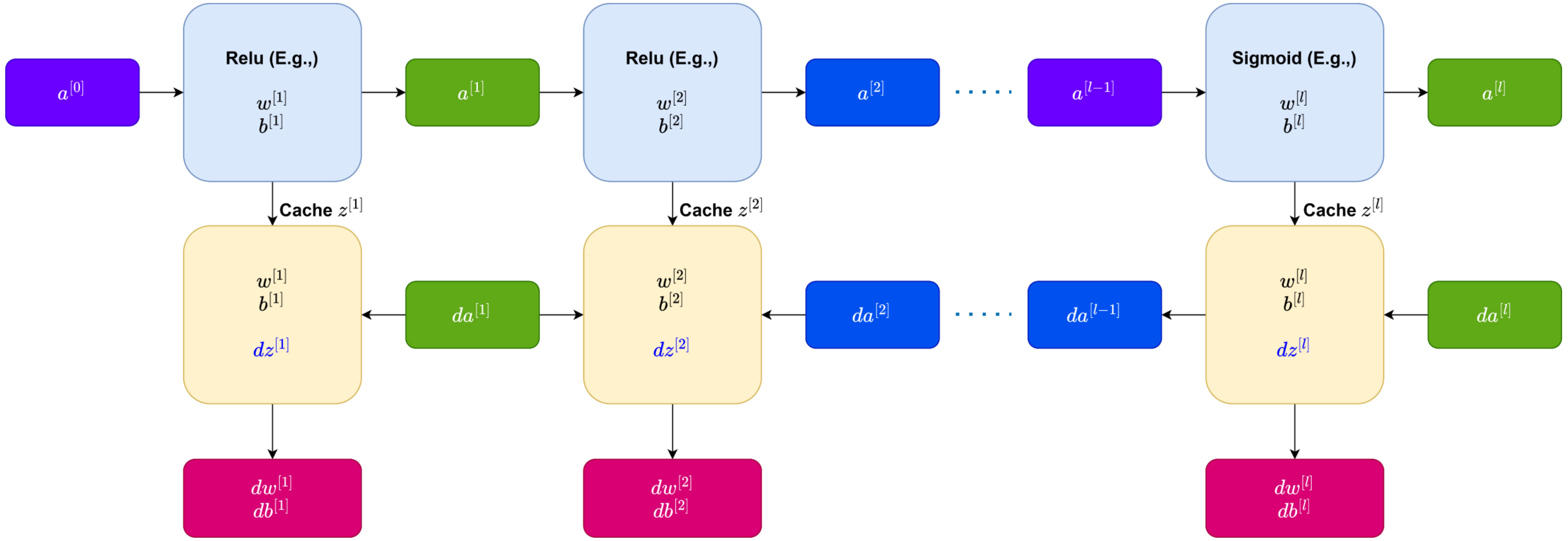
  $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$, cache $z^{[l]}$

  $a[l] = g^{[l]}(z^{[l]})$

- Backward Function for layer l:
  Input $da^{[l]}$ & cache $z^{[l]}$
  Output $da^{[l-1]}, dw^{[l]}, db^{[l]}$



Backward Propagation: For NN



Backward Propagation: For NN

- Based on the calculated gradients the parameters are updating as follows (using gradient decent in this case):

$$W^{[l]} := W^{[l]} - \alpha dw^{[l]}$$
$$b^{[l]} := b^{[l]} - \alpha db^{[l]}$$

Backward Propagation: For NN

- Summary (Un-Vectorized Version)

$$dz^{[l]} = da^{[l]} * g^{[l]'}\left(z^{[l]}\right)$$

$$dW^{[l]} = dz^{[l]} . a^{[l-1]}$$

$$db^{[l]} = dz^{[l]}$$

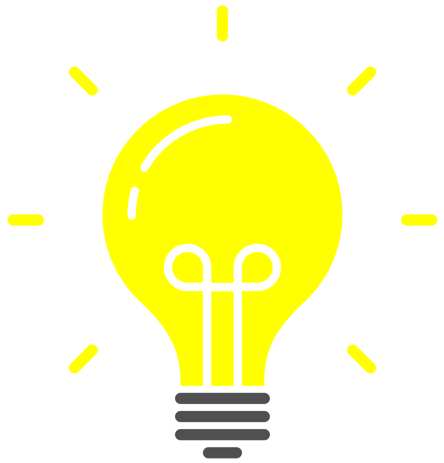
$$da^{[l-1]} = W^{[l]T} . dz^{[l]}$$

ANN: Additional Resources

- Watch this playlist from DeepLearningAI by Prof. Andrew NG to obtain further understanding regarding the concepts: https://www.youtube.com/watch?v=CS4cs9xVecg&list=PLkDaE6sCZn6Ec-XTbcX1uRg2_u4xOEky0
- Additionally watch the following playlist to gain an extensive understanding of the neural networks: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

ANN

- Investigate how to avoid overfitting and ensure the generalization in ANNs?



References

1. “What is Perceptron | The Simplest Artificial neural network,” GeeksforGeeks, Jul. 31, 2023. https://www.geeksforgeeks.org/what-is-perceptron-the-simplest-artificial-neural-network/
2. “What is Perceptron? A Beginners Guide for 2023 | Simplilearn,” Simplilearn.com, May 26, 2021. https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron#activation_functions_at_a_glance (accessed Feb. 29, 2024).
3. X. Wang, Y. Liu, and H. Xin, “Bond strength prediction of concrete-encased steel structures using hybrid machine learning method,” Structures, vol. 32, pp. 2279-2292, Aug. 2021, doi: https://doi.org/10.1016/j.istruc.2021.04.018.
4. AIML.com, “What is a Perceptron? What is the role of bias in a perceptron (or neuron)?,” AIML.com, Jun. 27, 2022. https://aiml.com/what-is-a-perceptron/ (accessed Feb. 29, 2024).
5. “Introduction to Artificial Neural Networks and the Perceptron | QuantStart,” www.quantstart.com. https://www.quantstart.com/articles/introduction-to-artificial-neural-networks-and-the-perceptron/
6. A. Hange, “Flux Prediction using Single-layer Perceptron and Multilayer Perceptron,” Nerd For Tech, Apr. 18, 2021. https://medium.com/nerd-for-tech/flux-prediction-using-single-layer-perceptron-and-multilayer-perceptron-cf82c1341c33
7. S. SHARMA, “Activation Functions in Neural Networks,” Medium, Sep. 06, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
8. “Top 10 Activation Function’s Advantages & Disadvantages,” www.linkedin.com. https://www.linkedin.com/pulse/top-10-activation-functions-advantages-disadvantages-dash.
9. “Artificial Neural Networks and its Applications,” GeeksforGeeks, Jun. 24, 2020. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/

References

10. IBM, “What Are Neural Networks? | IBM,” www.ibm.com, 2023. https://www.ibm.com/topics/neural-networks

11. AWS, “What is a Neural Network? AI and ML Guide - AWS,” Amazon Web Services, Inc., 2023. https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a

12. Google, “Overview of GAN Structure | Generative Adversarial Networks,” Google Developers, 2019. https://developers.google.com/machine-learning/gan/gan_structure

13. S. Saha, “A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way,” Towards Data Science, Dec. 15, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

14. V. Gadre, “Recurrent Neural Networks: A Beginner’s Guide,” Medium, Oct. 03, 2023. https://vijaygadre.medium.com/recurrent-neural-networks-a-beginners-guide-16333bd2eeb1

15. A. Dertat, “Applied Deep Learning - Part 3: Autoencoders,” Medium, Oct. 03, 2017. https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

16. “A Short Introduction to Generative Adversarial Networks - Thalles’ blog,” sthalles.github.io. https://sthalles.github.io/intro-to-gans/

17. S. K. Adaloglou Nikolas, “How diffusion models work: the math from scratch,” AI Summer, Sep. 29, 2022. https://theaisummer.com/diffusion-models/

18. R. Merritt, “What Is a Transformer Model?,” NVIDIA Blog, Mar. 25, 2022. https://blogs.nvidia.com/blog/what-is-a-transformer-model/

References

19. "Deep Learning Architectures for Medical Image Segmentation," Eda AYDIN, Feb. 02, 2023. https://edaaydinea.home.blog/2023/02/03/deep-learning-architectures-for-medical-image-segmentation/ (accessed Mar. 01, 2024).
20. A. L. Chandra, "Tutorial: Selfie Filters Using Deep Learning And OpenCV (Facial Landmarks Detection)," Medium, Sep. 27, 2019. https://towardsdatascience.com/facial-keypoints-detection-deep-learning-737547f73515
21. R. Chua, "A simple way to explain the Recommendation Engine in AI," Medium, Jun. 26, 2019. https://medium.com/voice-tech-podcast/a-simple-way-to-explain-the-recommendation-engine-in-ai-d1a609f59d97
22. N. Sardana, "Neural Networks: Forward and Backpropagation," 2017.
23. "Semi Supervised Classification in Data Mining," GeeksforGeeks, Aug. 05, 2022. https://www.geeksforgeeks.org/semi-supervised-classification-in-data-mining/?ref=ml_lbp (accessed Mar. 03, 2024).
24. "Delta rule," Wikipedia, Oct. 27, 2023. https://en.wikipedia.org/wiki/Delta_rule#:~:text=In%20machine%20learning%2C%20the%20delta (accessed Mar. 07, 2024).
25. "Vanishing gradient problem," Engati. https://www.engati.com/glossary/vanishing-gradient-problem#:~:text=Vanishing%20gradient%20problem%20is%20a
26. "Vanishing Gradient: 기울기 소실," Velog.io, 2022. https://velog.io/@yunyoseob/Gradient-Vanishing-%EA%B8%B0%EC%9A%B8%EA%B8%B0-%EC%86%8C%EC%8B%A4 (accessed Mar. 07, 2024).
27. T. Jacob, "Vanishing Gradient Problem, Explained," KDnuggets, Feb. 25, 2022. https://www.kdnuggets.com/2022/02/vanishing-gradient-problem.html.

Thank You