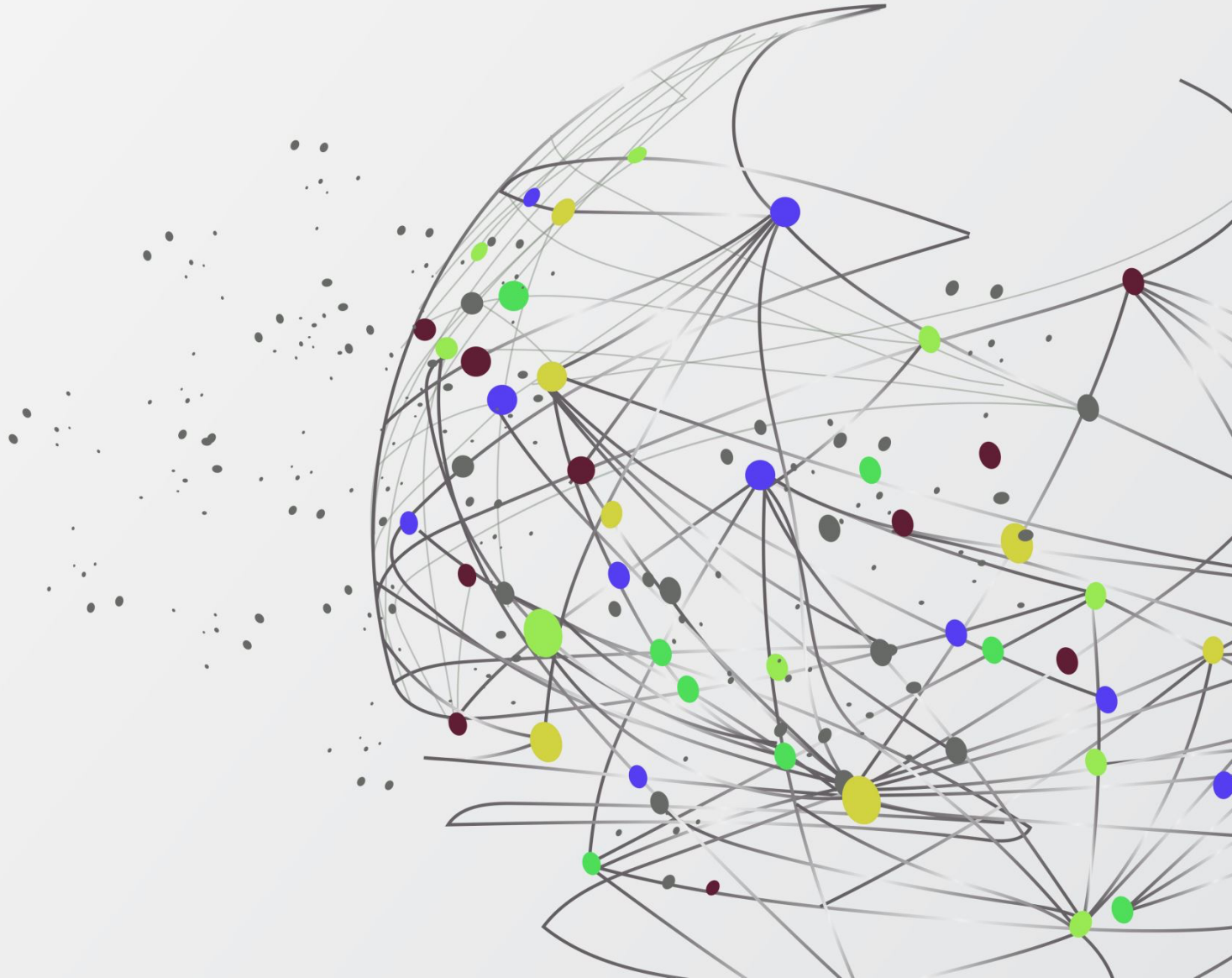
Mentor Session
Deep Learning Part 1

Machine Learning &
Data Science II

Yingjie(Chelsea) Wang



Today's Plan

- Some concepts on Deep Learning
  - Background
  - Artificial Neural Networks (ANN)
    - Feedforward Neural Network
    - Backpropagation
  - Recurrent Neural Networks
  - Convolutional Neural Networks
- Use Case
  - Develop a Neural Network With Keras
  - Evaluate The Performance of Deep Learning Models
  - Regularization and Parameter Tuning
  - Save Your Models and Reload the Model
  - Understand Model Behavior During Training By Plotting History
- Examples and exercise with Keras and Tensorflow 2
  - MLP, CNN, RNN

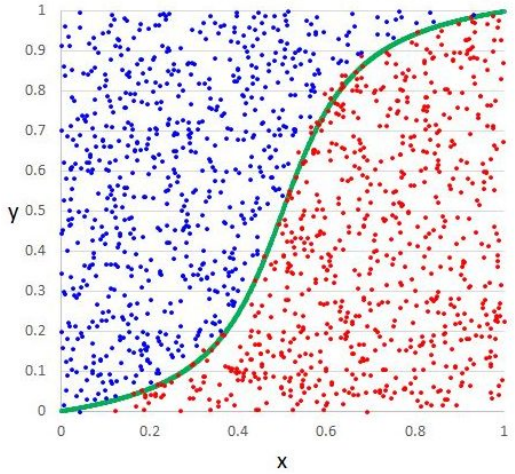
Most Popular Deep Learning Frameworks

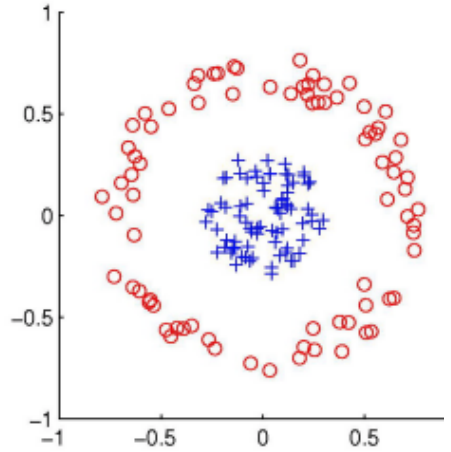
- Keras
  - Keras focuses on being modular, user-friendly, and extensible. It doesn't handle low-level computations.
  - Keras was adopted and integrated into TensorFlow in mid-2017. Users can access it via the tf.keras module. However, the Keras library can still operate separately and independently.
- TensorFlow
  - TensorFlow is an end-to-end open-source deep learning framework developed by Google and released in 2015.
  - It is known for documentation and training support, scalable production and deployment options, multiple abstraction levels, and support for different platforms, such as Android.
- PyTorch
  - PyTorch is a relatively new deep learning framework based on Torch. Developed by Facebook's AI research group and open-sourced on GitHub in 2017, it's used for natural language processing applications.
  - PyTorch has a reputation for simplicity, ease of use, flexibility, efficient memory usage, and dynamic computational graphs.

	Keras	TensorFlow	PyTorch
Level of API	high-level API[1]	Both high & low level APIs	Lower-level API[2]
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex[3]
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook[4]
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs[5]

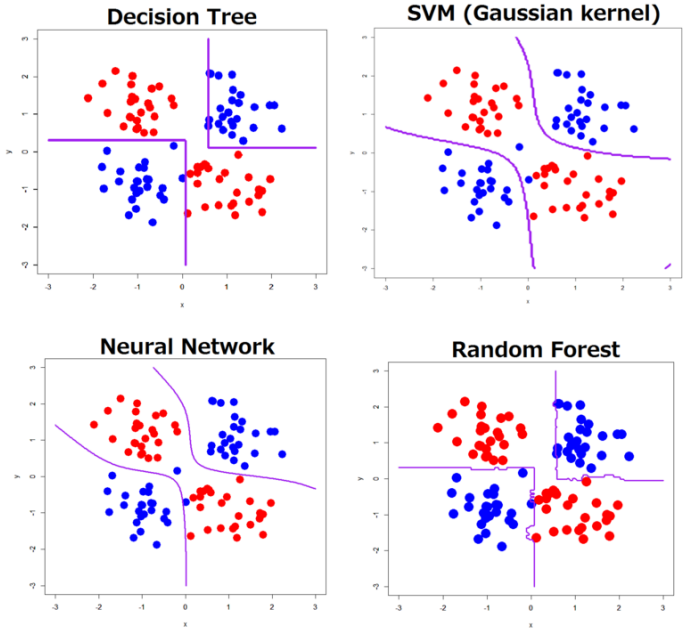
Machine Learning: Decision Boundary

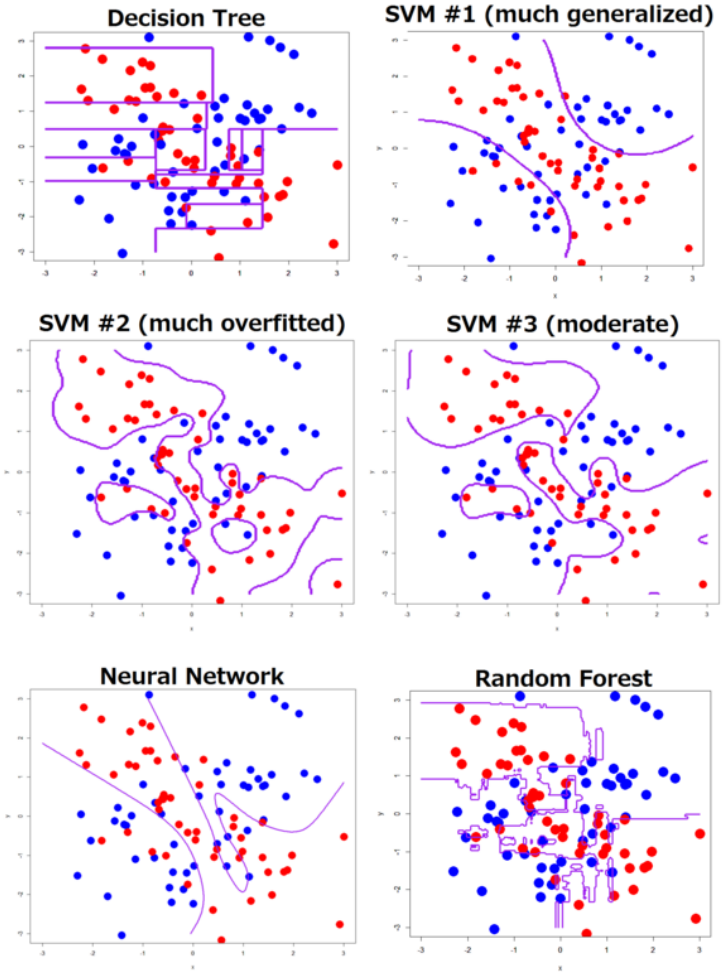
- Every Machine Learning algorithm learns the mapping from an input to output. In case of parametric models, the algorithm learns a function with a few sets of weights:
  - Input -> $f(w_1, w_2 \ldots . . w_n)$ -> Output
- In the case of classification problems, the algorithm learns the function that separates 2 classes – this is known as a Decision boundary. A decision boundary helps us in determining whether a given data point belongs to a positive class or a negative class.
- The Machine Learning algorithm (logistic regression in this case) is not capable of learning all the functions. This limits the problems these algorithms can solve that involve a complex relationship.





Machine Learning: Decision Boundary

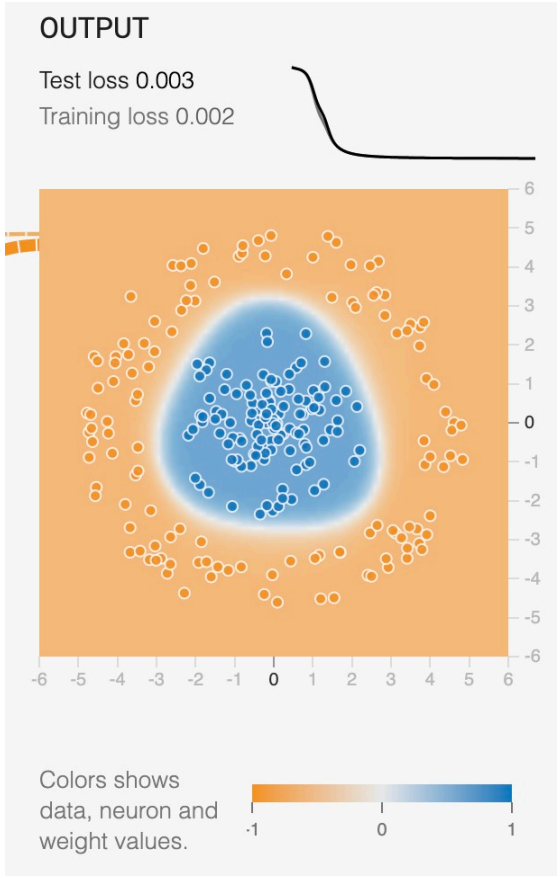


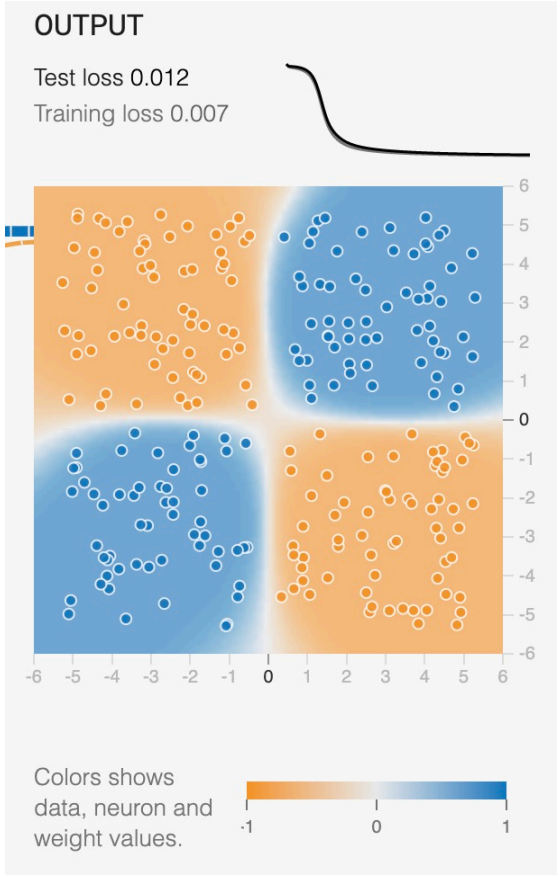


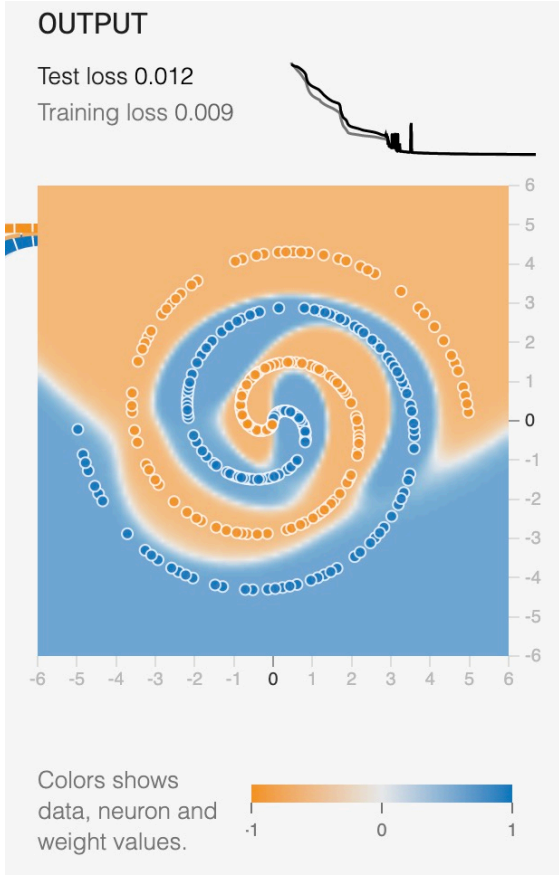
XOR pattern

- Decision tree, neural network and random forest estimated much more complicated boundaries than the true boundaries, although SVM with well generalized by specific parameters gave natural and well smoothed boundaries (but classification accuracy was not good).

Deep Learning: Decision Boundary

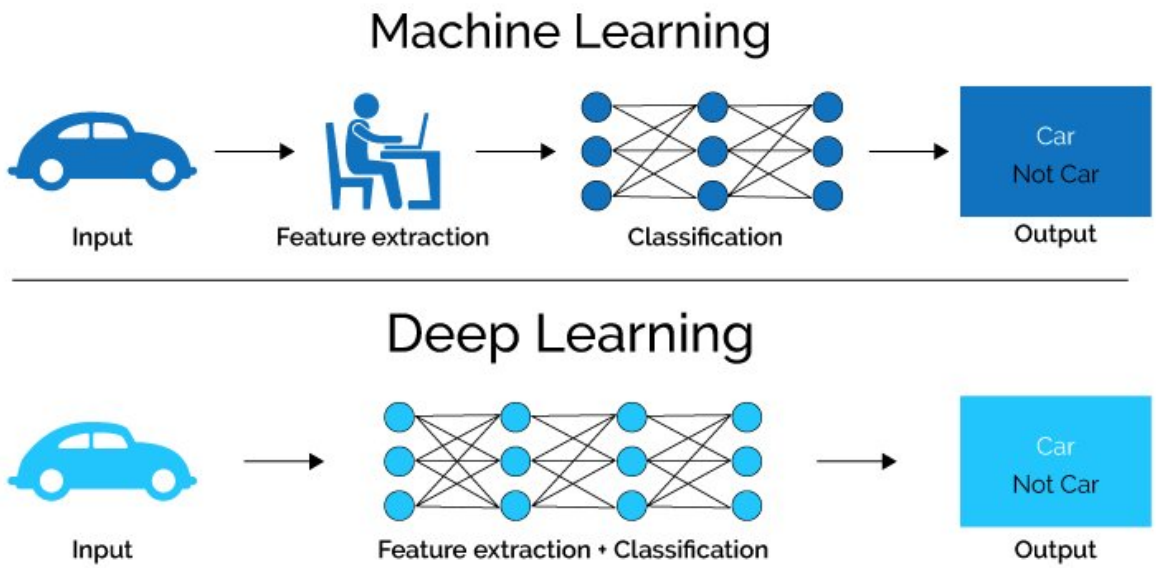






Machine Learning vs. Deep Learning: Feature Extraction

- In feature extraction, we extract all the required features for our problem statement and in feature selection, we select the important features that improve the performance of our machine learning.
- Consider an image classification problem. Extracting features manually from an image needs strong knowledge of the subject as well as the domain. It is an extremely time-consuming process. Thanks to Deep Learning, we can automate the process of Feature Engineering!

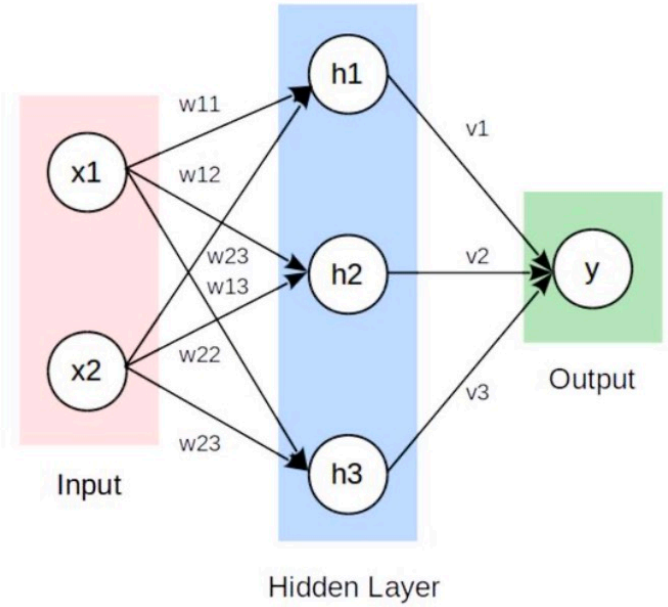


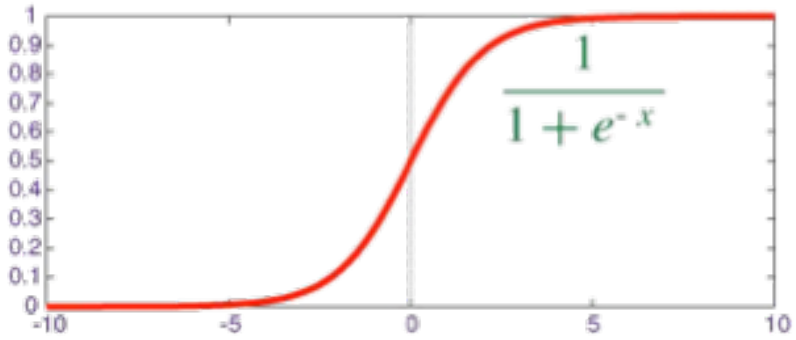
Types of Neural Networks in Deep Learning

- Three important types of neural networks that form the basis for most pre-trained models in deep learning:
  - Artificial Neural Networks (ANN)
  - Convolution Neural Networks (CNN)
  - Recurrent Neural Networks (RNN)

	MLP	RNN	CNN
Data	Tabular data	Sequence data (Time Series,Text, Audio)	Image data
Recurrent connections	No	Yes	No
Parameter sharing	No	Yes	Yes
Spatial relationship	No	No	Yes
Vanishing & Exploding Gradient	Yes	Yes	Yes

Feedforward Neural Network





- Each unit can be computed as two parts:
  - Linear part: weighted sum of inputs (plus bias)
    - $a_i = w_{1i}x_1 + w_{2i}x_2 + b_i$
  - Non-linear part: transformation of that sum by a nonlinearity of our choosing.
    - $h_i = f(a_i)$

$$\vec{x}^T = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

$$\vec{a} = \vec{x}W + \vec{b}$$

$$\vec{h} = f(\vec{x}W + \vec{b})$$

$$\begin{aligned} y &= g(v_1h_1 + v_2h_2 + v_3h_3 + c) \\ &= g(v_1 f(w_{11}x_1 + w_{12}x_2 + b_1) \\ &+ v_2 f(w_{21}x_1 + w_{22}x_2 + b_2) \\ &+ v_3 f(w_{31}x_1 + w_{32}x_2 + b_3) \\ &+ c) \end{aligned}$$

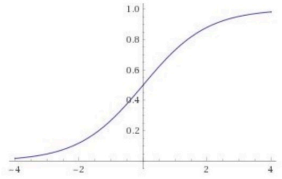
Activation Functions

- The purpose of the activation function is to **introduce non-linearity** into the output of a neuron.

- Sigmoid: not blowing up activation
- Relu: not vanishing gradient
- Relu: More computationally efficient to compute than Sigmoid like functions since Relu just needs to pick max(0, x) and not perform expensive exponential operations as in Sigmoids.
- Relu : In practice, networks with Relu tend to show better convergence performance than sigmoid.

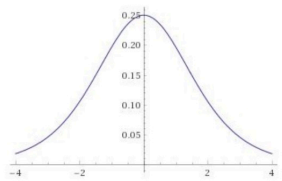
Sigmoid

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



- This function is familiar to us
- Logistic activation, easy to reason about

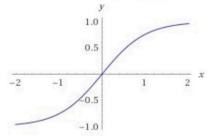
$$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$$



- This function saturates and its gradients are small
- This function isn't 0-centered
- Useful for understanding ANNs, but not used much in practice anymore

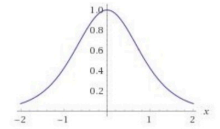
Tanh

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



- This function is 0-centered, behaves a little better
- Derivative is little stronger

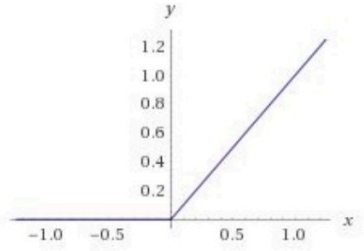
$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$



- If you're looking for a smooth S-shaped curve, try this one instead of sigmoid

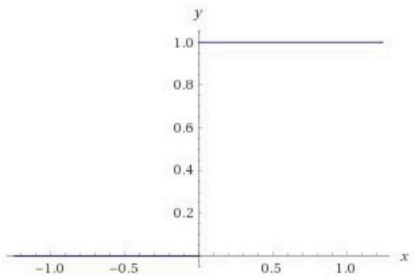
ReLU

$$\mathrm{ReLu}(x) = \max(0, x)$$



- This function doesn't saturate at large values of x
- The derivative does not saturate at large values of x

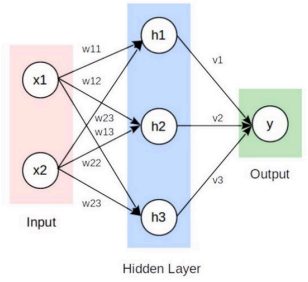
$$\frac{d}{dx}(\max(0, x)) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$



- ReLU should be your go-to activation function

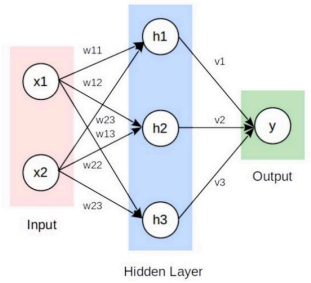
Output Functions

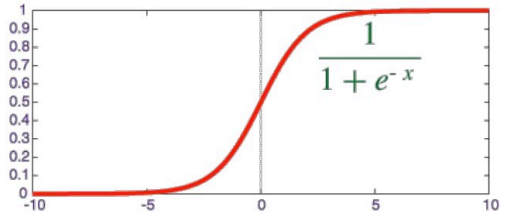
- Linear



- For continuous target variable
  - $\hat{y} = \vec{v}^T \vec{h} + c$
- Accompanying Loss Function
- Squared Error Loss
  - $L = \frac{1}{N}\frac{1}{2}\sum_i((y_i - \hat{y}_i)^2)$
  - $\frac{\partial L}{\partial \hat{y}_i} = y_i - \hat{y}_i$

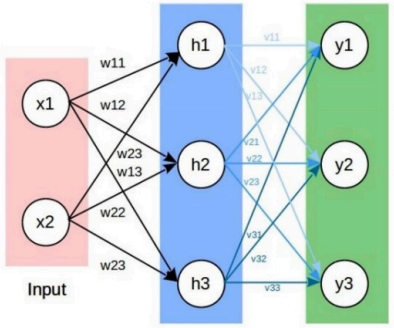
- Sigmoid





- For binary target variables (Bernoulli)
  - $\hat{y} = \sigma(\vec{v}^T \vec{h} + c)$
- Intended to represent a probability over the two classes. It "squashes" a real-valued scalar to lie within [0,1].
- Binary Cross Entropy
  - $L = y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i)$
  - $\frac{dL}{d\hat{y}_i} = \frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}$

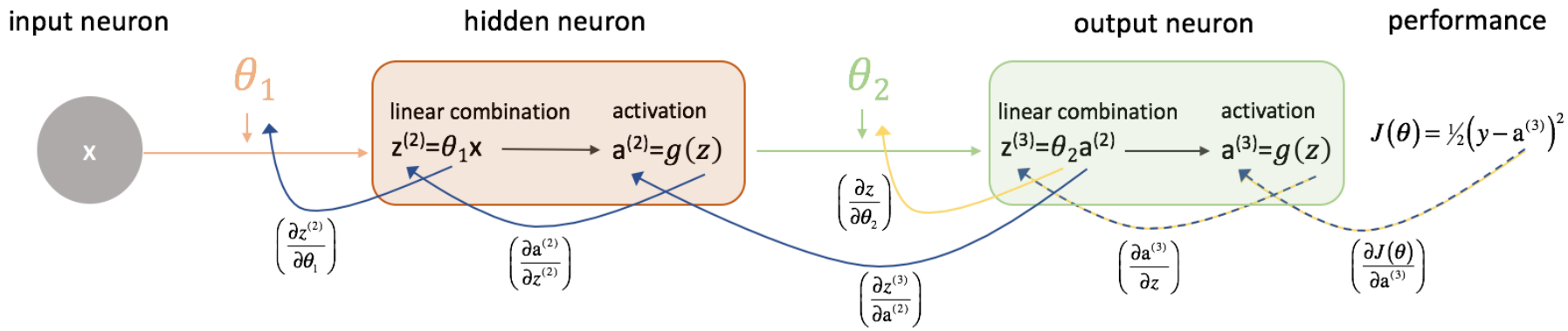
- Softmax



- For categorical outputs
  - $\vec{z} = W^T \vec{h} + \vec{b}$
  - $\hat{y}_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$
- Converts real-valued vector (logits) into a probability vector over K classes. Similar to Sigmoid, it squashes input into a valid probability vector.
- Categorical Cross Entropy
  - $L = -\sum_j y_i \log(\hat{y}_i) = -\vec{y}^T \log(\vec{\hat{y}_i})$
  - $\frac{dL}{d\hat{y}_i} = \vec{\hat{y}_i} - \vec{y}$

Backpropagation

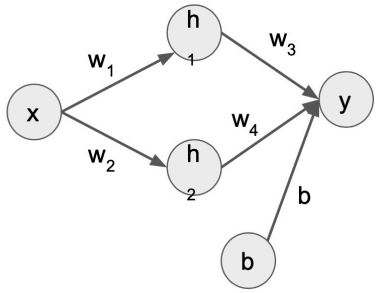
- The algorithm is used to effectively train a neural network through a method called **chain rule**. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while **adjusting the model's parameters** (weights and biases).

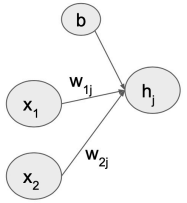


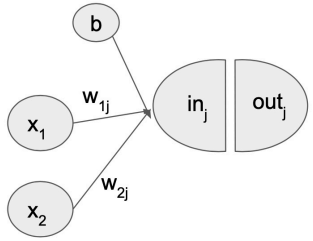
Backpropagation

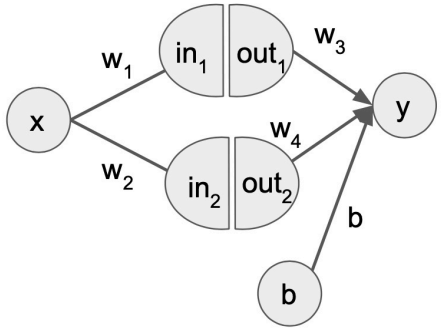
Example - Regression Network with sigmoid activation

- This notation makes it clear how we will compute derivatives of composite functions. Some useful properties:









$$L = \frac{1}{2}(y_i - \hat{y}(x_i))^2$$

$$\theta = \{w_1, w_2, w_3, w_4, b\}$$

$$\nabla_\theta = \{\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial b}\}$$

$$\text{in}_j = x_1 w_{1j} + x_2 w_{2j} + b$$

$$\text{out}_j = f(\text{in}_j) = \frac{1}{1 + \exp(-\text{in}_j)}$$

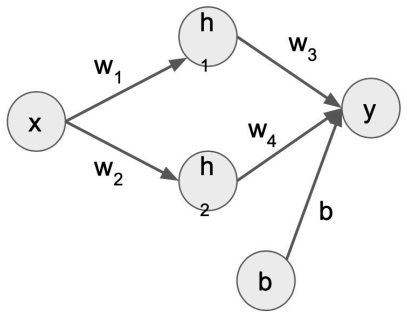
$$\frac{d\text{out}_j}{d\text{in}_j} = \frac{d}{dx} f(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

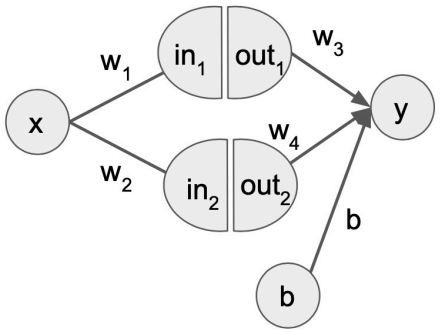
$$\frac{d\text{in}_j}{dw_k} = \frac{d}{dw_k}[b + w_1 x_1 + w_2 x_2] = x_k$$

$$\frac{d\text{out}_j}{dw_k} = \frac{d\text{out}_j}{d\text{in}_j}\frac{d\text{in}_j}{dw_k}$$
$$= \sigma(\text{in}_j)(1 - \sigma(\text{in}_j))x_k$$

Backpropagation

Example - Regression Network with sigmoid activation





$$L = \frac{1}{2}(y_i - \hat{y}(x_i))^2$$

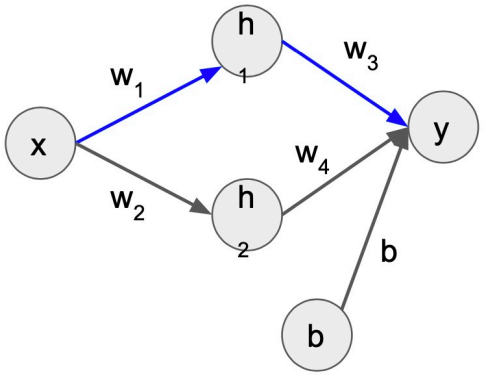
$$\theta = \{w_1, w_2, w_3, w_4, b\}$$

$$\nabla_\theta = \{\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial b}\}$$

$$\begin{aligned}
\frac{\partial L}{dw_3} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_3} \\
&= \frac{\partial L}{\partial \hat{y}} \frac{\partial}{\partial w_3}(w_3 \text{out}_1 + w_4 \text{out}_2 + b) \\
&= \frac{\partial L}{\partial \hat{y}} \text{out}_1 \\
&= \text{out}_1 \frac{\partial}{\partial \hat{y}}(\frac{1}{2}(y_i - \hat{y}(x_i))^2) \\
&= \text{out}_1(y_i - \hat{y}(x_i))(0-1) \\
&= -\text{out}_1(y_i - \hat{y}(x_i))
\end{aligned}$$

$$\frac{\partial L}{\partial w_3} = -\text{out}_1 e_i \qquad \frac{\partial L}{\partial w_4} = -\text{out}_2 e_i$$

$$\begin{aligned}
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b} \\
&= -(y_i - \hat{y}(x_i)) \frac{\partial}{\partial b} \hat{y} \\
&= -e_i \frac{\partial}{\partial b}(w_3 \text{out}_1 + w_4 \text{out}_2 + b) \\
&= -e_i
\end{aligned}$$



$$\begin{aligned}
\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \text{out}_1} \frac{d\text{out}_1}{d\text{in}_1} \frac{\partial \text{in}_1}{\partial w_1} \\
\frac{\partial L}{\partial \hat{y}} &= -(y_i - \hat{y}(x_i)) \\
\frac{\partial \hat{y}}{\partial \text{out}_1} &= w_3 \\
\frac{d\text{out}_1}{d\text{in}_1} &= \sigma(\text{in}_1)(1 - \sigma(\text{in}_1)) \\
\frac{\partial \text{in}_1}{\partial w_1} &= \frac{d}{dw_1}(xw_1) = x \\
\frac{\partial L}{\partial w_1} &= -x_i e_i w_3 \sigma(\text{in}_1)(1 - \sigma(\text{in}_1))
\end{aligned}$$

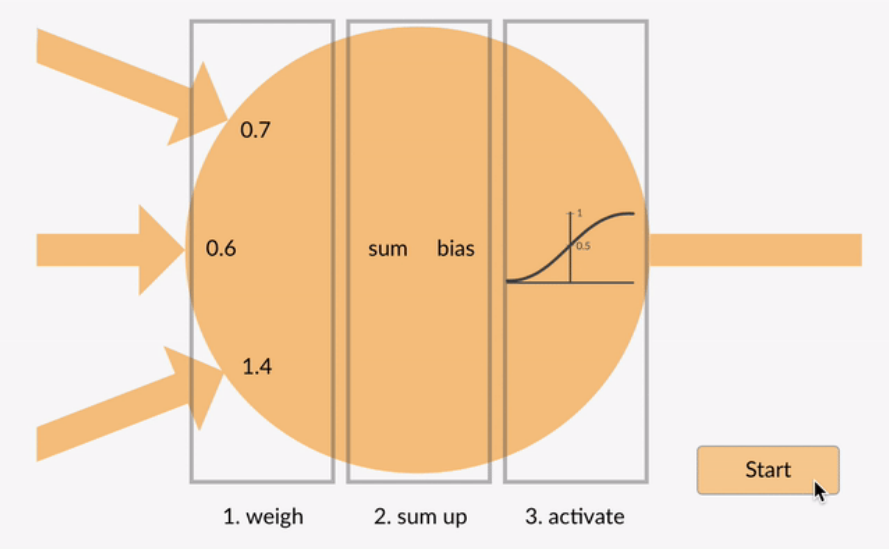
$$\frac{\partial L}{\partial w_2} = -x_i e_i w_4 \sigma(\text{in}_2)(1 - \sigma(\text{in}_2))$$

6 Stages of Neural Network Learning

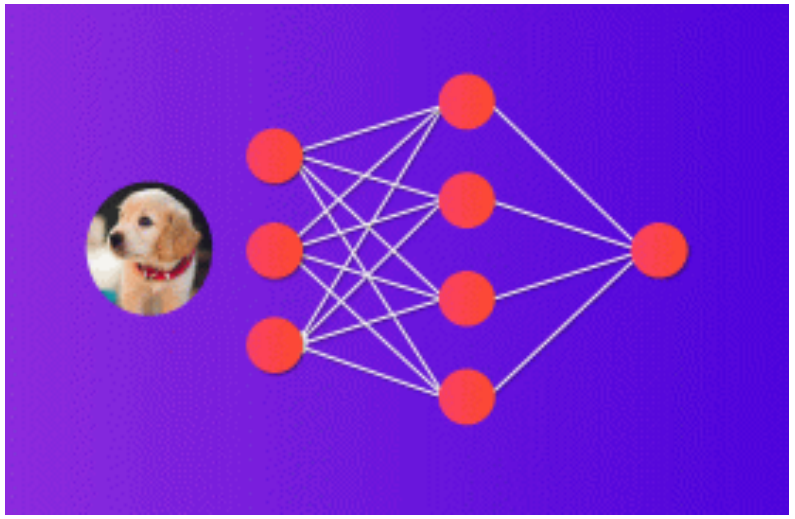
- **Initialization** – initial weights are applied to all the neurons.
- **Forward propagation** – the inputs from a training set are passed through the neural network and an output is computed.
- **Error function** – because we are working with a training set, the correct output is known. An error function is defined, which captures the delta between the correct output and the actual output of the model, given the current model weights (in other words, “how far off” is the model from the correct result).
- **Backpropagation** – the objective of backpropagation is to change the weights for the neurons, in order to
- **Weight update –** weights are changed to the optimal values according to the results of the backpropagation algorithm.
- **Iterate until convergence** – because the weights are updated a small delta step at a time, several iterations are required in order for the network to learn. After each iteration, the gradient descent force updates the weights towards less and less global loss function. The amount of iterations needed to converge depends on the learning rate, the network meta-parameters, and the optimization method used.
- At the end of this process, the model is ready to make predictions for unknown input data. New data can be fed to the model, a forward pass is performed, and the model generates its prediction.

Advantages of Artificial Neural Network (ANN)

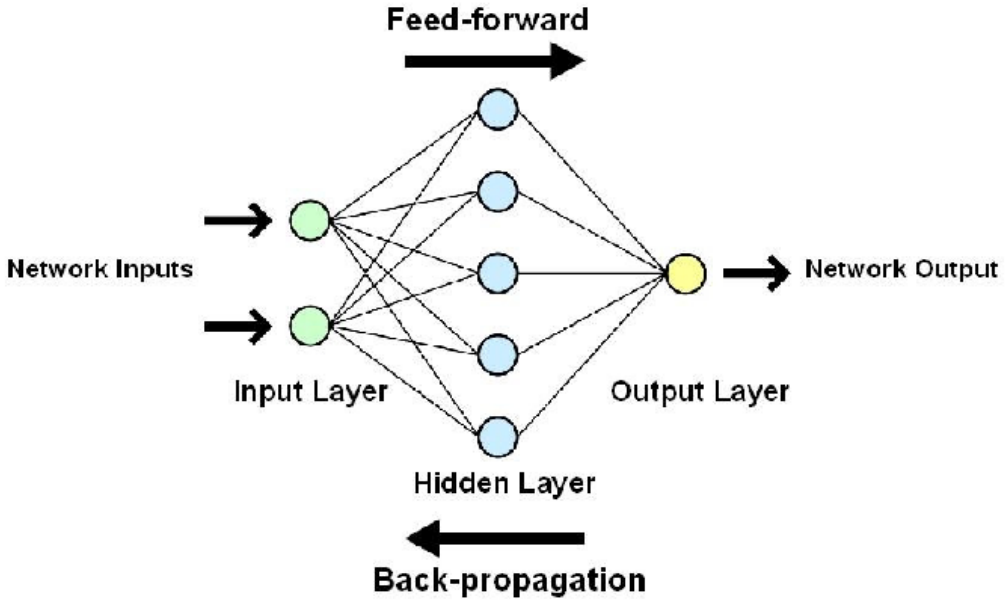
- Artificial Neural Network is capable of learning any nonlinear function. Hence, these networks are popularly known as Universal Function Approximators. ANNs have the capacity to learn weights that map any input to the output.
- One of the main reasons behind universal approximation is the activation function. Activation functions introduce nonlinear properties to the network. This helps the network learn any complex relationship between input and output.



Challenges with Artificial Neural Network (ANN)

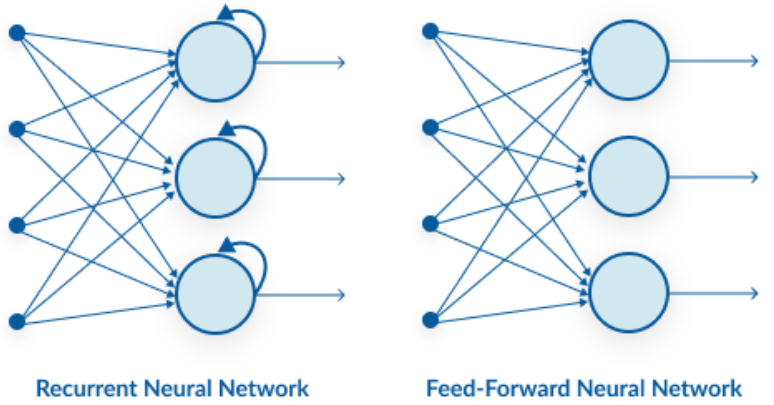


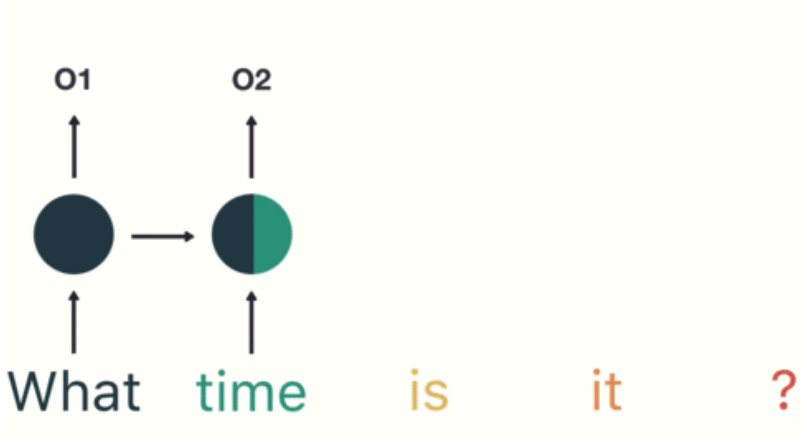
- While solving an image classification problem using ANN, the first step is to convert a 2-dimensional image into a 1-dimensional vector prior to training the model. This has two drawbacks:
  - The number of trainable parameters increases drastically with an increase in the size of the image. In the example, if the size of the image is 224*224, then the number of trainable parameters at the first hidden layer with just 4 neurons is 224*224*3*4 = 602112. That's huge!
  - ANN loses the spatial features of an image. Spatial features refer to the arrangement of the pixels in an image.
- One common problem in all these neural networks is the Vanishing and Exploding Gradient. This problem is associated with the backpropagation algorithm. The weights of a neural network are updated through this backpropagation algorithm by finding the gradients:
  - In the case of a very deep neural network (network with a large number of hidden layers), the gradient vanishes or explodes as it propagates backward which leads to vanishing and exploding gradient.



Recurrent Neural Network (RNN)

- Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. This looping constraint ensures that sequential information is captured in the input data.
- We can use recurrent neural networks to solve the problems related to:
  - Time Series data
  - Text data
  - Audio data
- RNN models are mostly used in the fields of natural language processing and speech recognition.

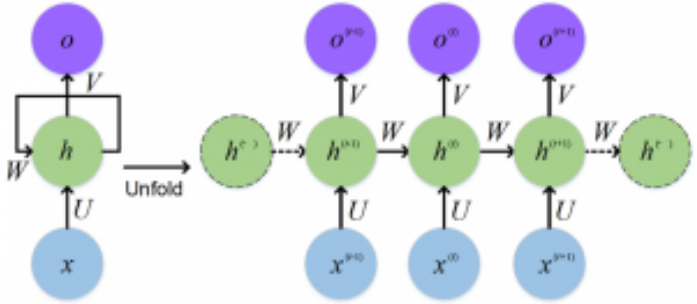




Advantages and Challenges of Recurrent Neural Network (RNN)

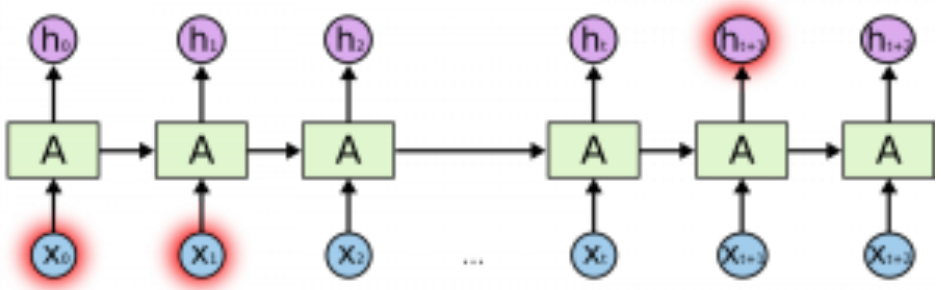
Advantages

- RNN captures the sequential information present in the input data i.e., dependency between the words in the text while making predictions.
- RNNs share the parameters across different time steps. This is popularly known as **Parameter Sharing**. This results in fewer parameters to train and decreases the computational cost.
  - Weight matrices (U, W, V) are the weight matrices that are shared across all the time steps.



Challenges

- The **vanishing and exploding gradient** phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

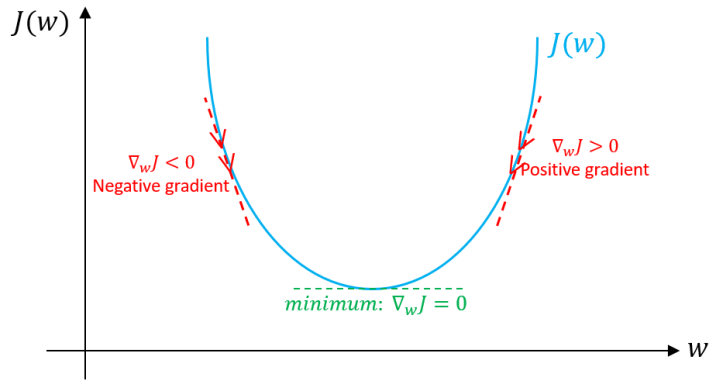


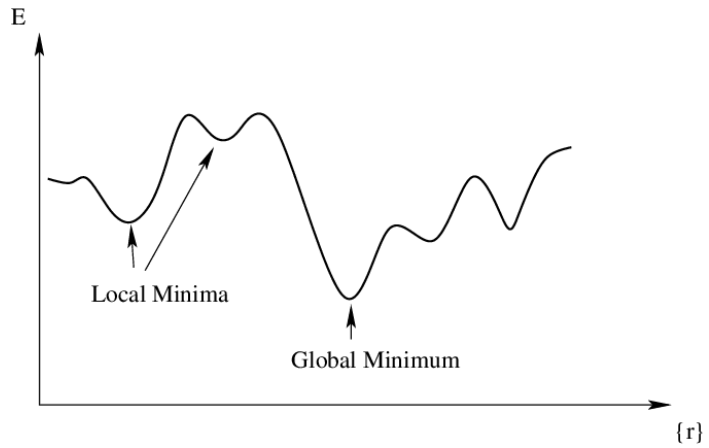
Use Cases

- Use Case
  - Develop a Neural Network With Keras
  - Evaluate The Performance of Deep Learning Models
  - Regularization and Parameter Tuning
  - Save Your Models and Reload the Model
  - Understand Model Behavior During Training By Plotting History
- Examples and exercise with Keras and Tensorflow 2
  - MLP, CNN, RNN

Gradient Descent Algorithm

- Gradient Descent (Optimization algorithm)
  - It is a first-order optimization algorithm: taking the first derivative when performing the updates on the parameters.
  - On each iteration, update the parameters in the opposite direction of the gradient of the objective function with respect to the parameters where the gradient gives the direction of the steepest ascent.
  - Overall, it follows the direction of the slope downhill until we reach a local minimum.
- Stochastic Gradient Descent
  - Shuffle the training data set to avoid pre-existing order of examples.
  - Partition the training data set into m examples.

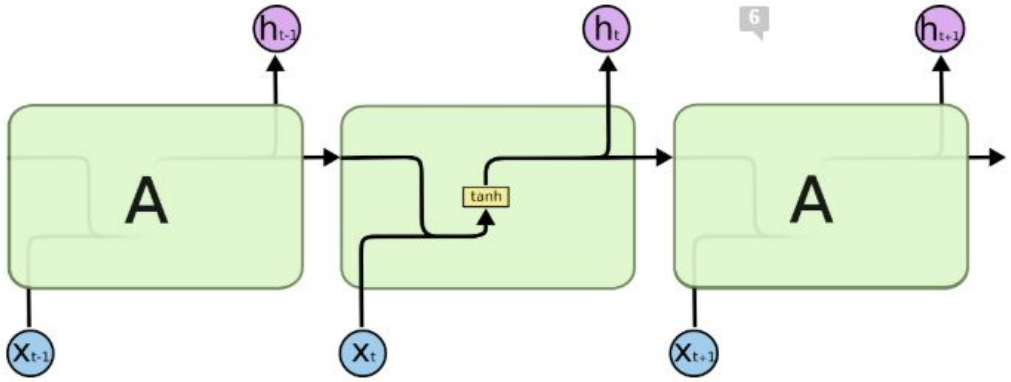




Long Short-Term Memory

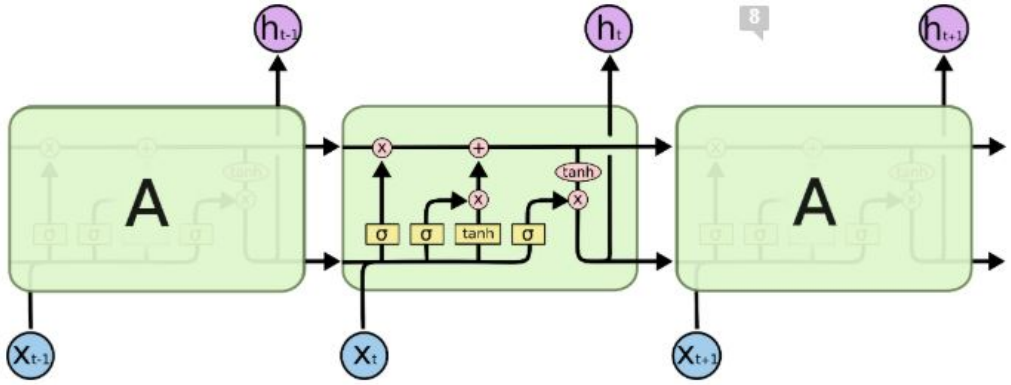
- Long Short-Term Memory (LSTM) networks are a type of **recurrent neural network** capable of learning order dependence in sequence prediction problems.
- Standard RNNs fail to learn in the presence of time lags greater than 5 – 10 discrete time steps between relevant input events and target signals..
- LSTMs (Long Short-Term Memory) deals with these problems by introducing new **gates**, such as input and forget gates, which allow for a better control over the gradient flow and enable better preservation of “long-range dependencies”. LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through “constant error carrousels” (CECs) within special units, called cells.
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

LSTM APPLICATIONS

- Time Series Prediction
  - Stock Price Prediction
  - House Price Prediction
  - https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- Text Data
  - Sentiment Analysis
  - Document Classification
  - Text Generation
  - https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
- Audio Data
  - Music Genre Classification
  - Speech Recognition

More than LSTM

- Bidirectional LSTMs
  - The basic idea of bidirectional recurrent neural nets is to present each training sequence **forwards and backwards** to two separate recurrent nets, both of which are connected to the same output layer. ... This means that for every point in a given sequence, the BRNN has complete, sequential information about all points before and after it. Also, because the net is free to use as much or as little of this context as necessary, there is no need to find a (task-dependent) time-window or target delay size.
- Attention-Based LSTM
  - Attention is the idea of freeing the encoder-decoder architecture from the fixed-length internal representation.
  - This is achieved by keeping the intermediate outputs from the encoder LSTM from each step of the input sequence and training the model to learn to **pay selective attention** to these inputs and relate them to items in the output sequence.