

Deep Learning and Neuron Network Overview



Liu Da

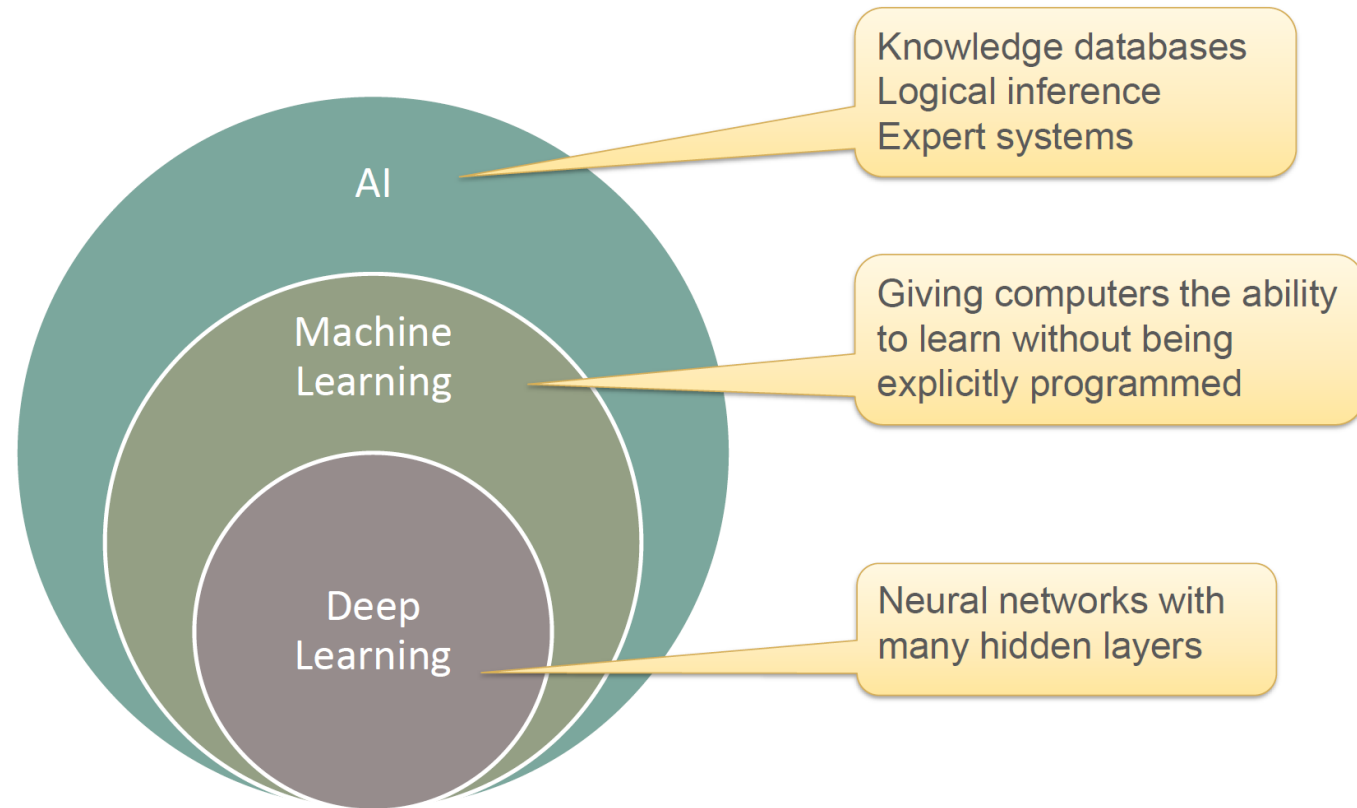
da.liu@ericsson.com

BCSS Learning Service

AI, machine learning and deep learning



- AI (Artificial Intelligence) is simply understood as the desire to enable machines to have the ability to judge and reason, and the specific application can be:
 - Predict System
 - Natural Language Processing (NLP)
 - Image, video, audio recognition
- Machine learning or deep learning is one of the goals that AI hopes to achieve, and it is also an important tool to achieve it.
- As a subset of machine learning, deep learning has its own unique processing methods and capabilities, which differ from relatively "classical" machine learning computing in various ways.

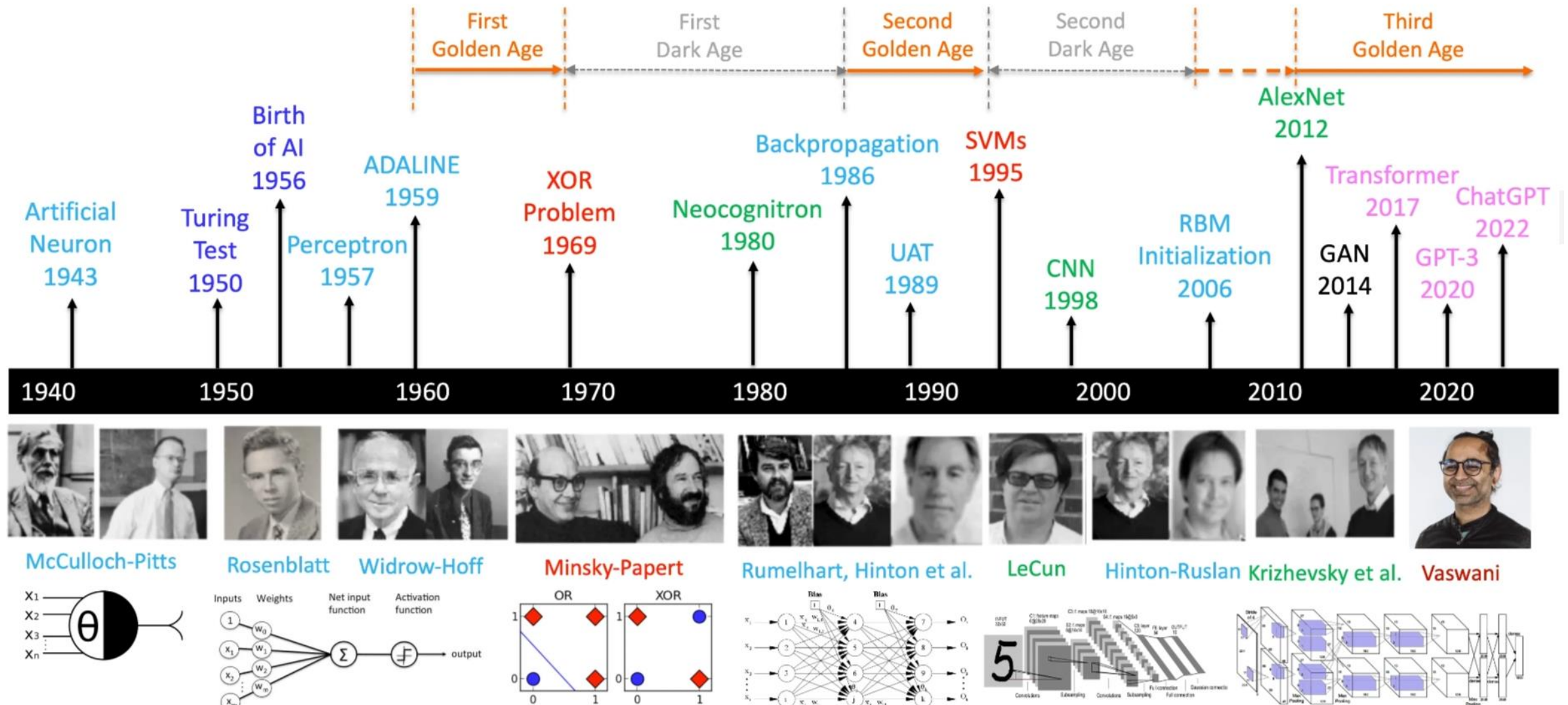


What is Deep Learning?



- Deep learning is the subset of machine learning methods based on neural networks with **representation learning**. The adjective “deep” refers to the use of multiple layers in the network. Methods used can be either supervised, semi-supervised or unsupervised. ([Wikipedia](#))
- Deep learning models can recognize complex patterns in pictures, text, sounds, and other data to produce accurate insights and predictions. You can use deep learning methods to automate tasks that typically require human intelligence, such as describing images or transcribing a sound file into text. ([AWS](#))
- The chief difference between deep learning and traditional machine learning is the structure of the underlying neural network architecture. “Nondeep,” traditional machine learning models use simple neural networks with one or two computational layers. Deep learning models use three or more layers—but typically hundreds or thousands of layers—to train the models. ([IBM](#))
- Deep learning is a subset of machine learning that's based on artificial neural networks. The learning process is deep because the structure of artificial neural networks consists of multiple input, output, and hidden layers. Each layer contains units that transform the input data into information that the next layer can use for a certain predictive task. Thanks to this structure, a machine can learn through its own data processing. ([Microsoft](#))

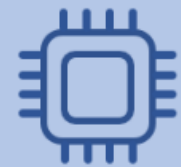
Brief history of Deep Learning



Hardware for Deep Learning

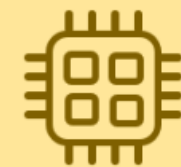


- Since the 2010s, advances in both machine learning algorithms and computer hardware have led to more efficient methods for training deep neural networks that contain many layers of non-linear hidden units and a very large output layer.
- By 2019, graphics processing units (GPUs), often with AI-specific enhancements, had displaced CPUs as the dominant method for training large-scale commercial cloud AI.
- Special electronic circuits called deep learning processors were designed to speed up deep learning algorithms. Deep learning processors include neural processing units (NPUs) in Huawei cellphones and cloud computing servers such as tensor processing units (TPU) in the Google Cloud Platform.



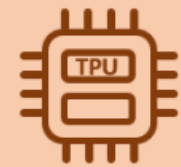
CPU

- Small models
- Small datasets
- Useful for design space exploration



GPU

- Medium-to-large models, datasets
- Image, video processing
- Application on CUDA or OpenCL



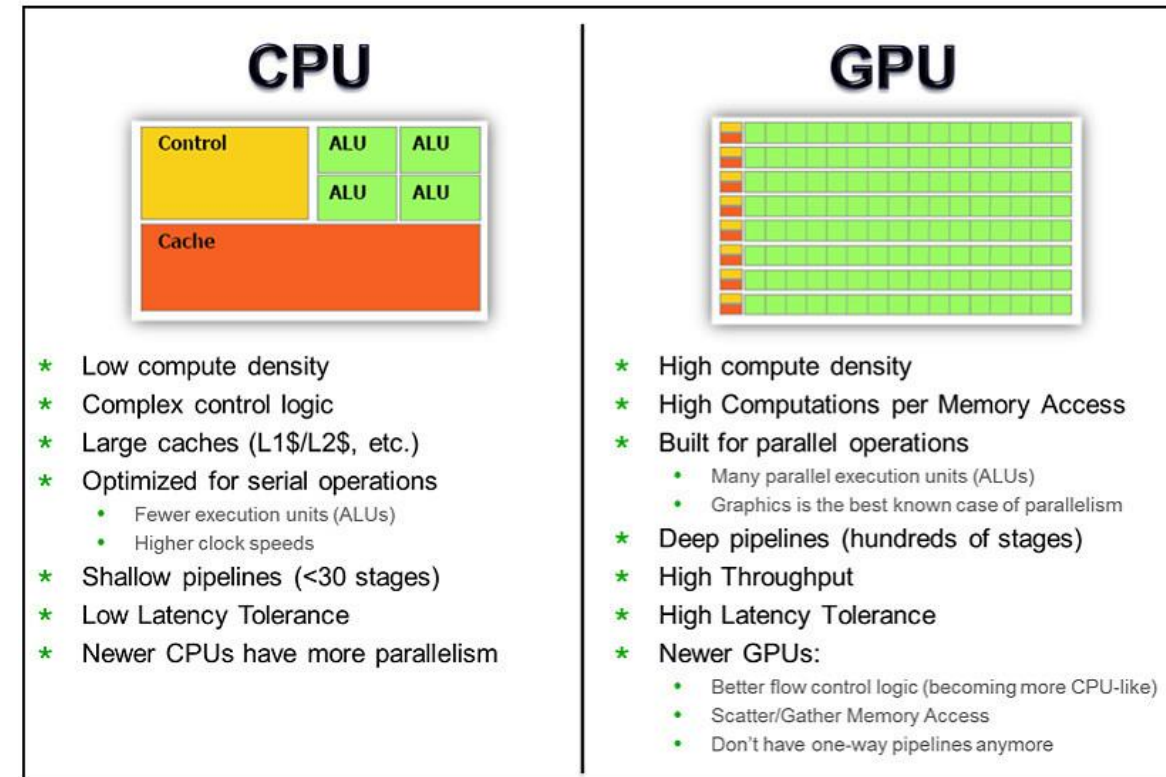
TPU

- Matrix computations
- Dense vector processing
- No custom TensorFlow operations

How GPU works?



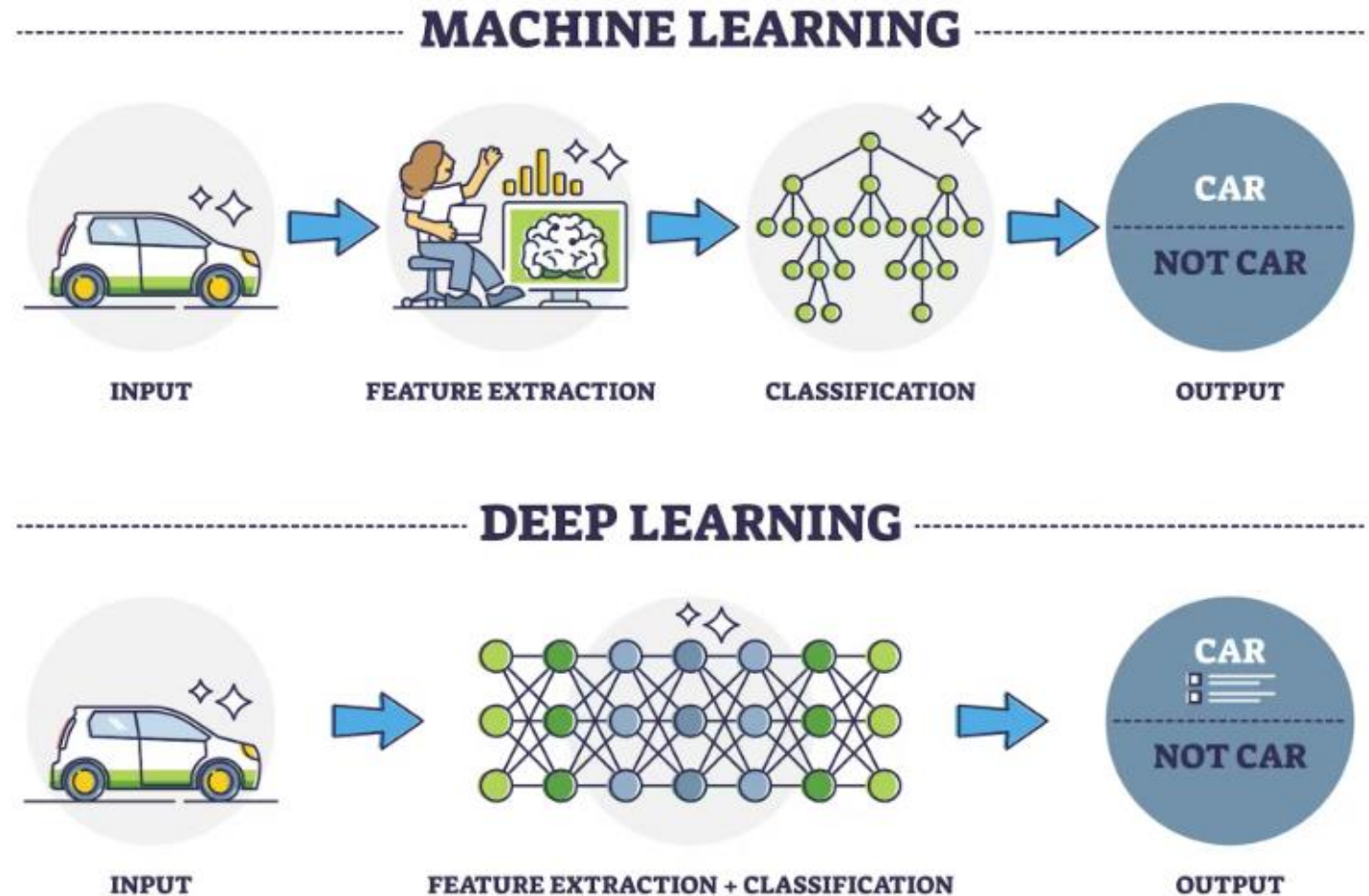
- A graphics processing unit (GPU) is an electronic circuit that can perform mathematical calculations at high speed. Computing tasks like graphics rendering, machine learning (ML), and video editing require the application of similar mathematical operations on a large dataset.
- A GPU's design allows it to perform the same operation on multiple data values in **parallel**. This increases its processing efficiency for many compute-intensive tasks.
- Modern GPUs typically contain a few multiprocessors. Each has a shared memory block, plus a few processors and corresponding registers. The GPU itself has constant memory, plus device memory on the board it is housed on.
- Each GPU works slightly differently depending on its purpose, the manufacturer, the specifics of the chip, and the software used for coordinating the GPU. For instance, Nvidia's CUDA parallel processing software allows developers to specifically program the GPU with almost any general-purpose parallel processing application in mind.



Traditional Machine Learning VS Deep Learning



- Machine learning uses statistical learning algorithms to find patterns in available data and perform predictions and classifications on new data. ML also comprises both supervised and unsupervised learning.
- Deep learning relies on multi-layered neural network models to perform complex tasks. There is a significant difference in the capabilities and applications of both. Understanding them is essential to knowing which to use in projects and get the best results.



Traditional Machine Learning VS Deep Learning

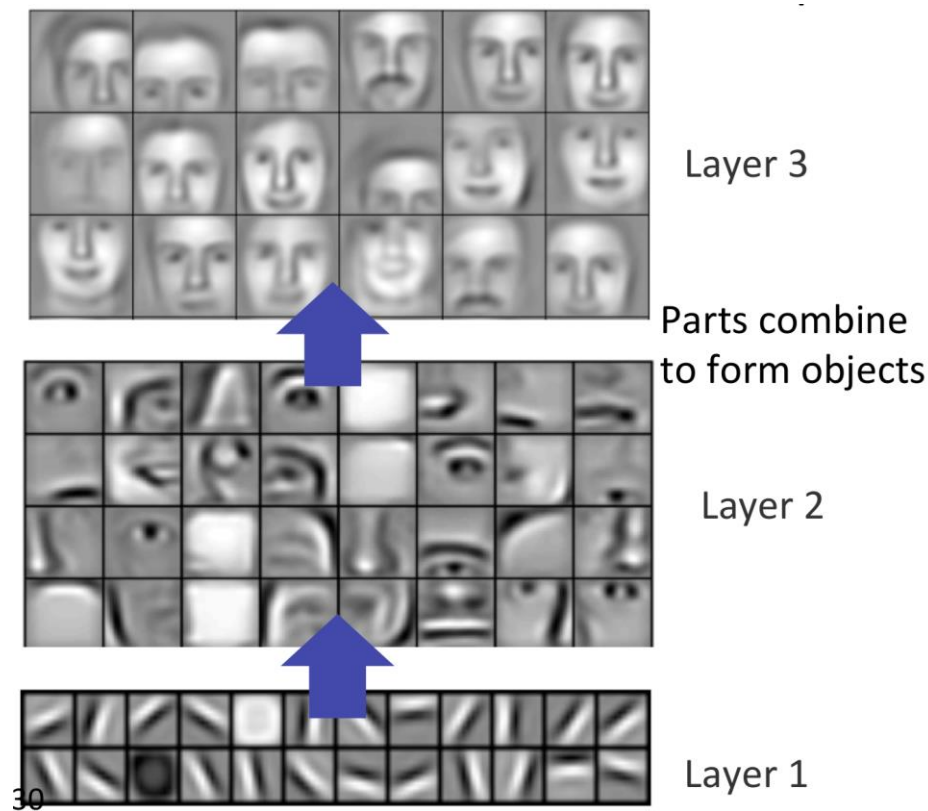
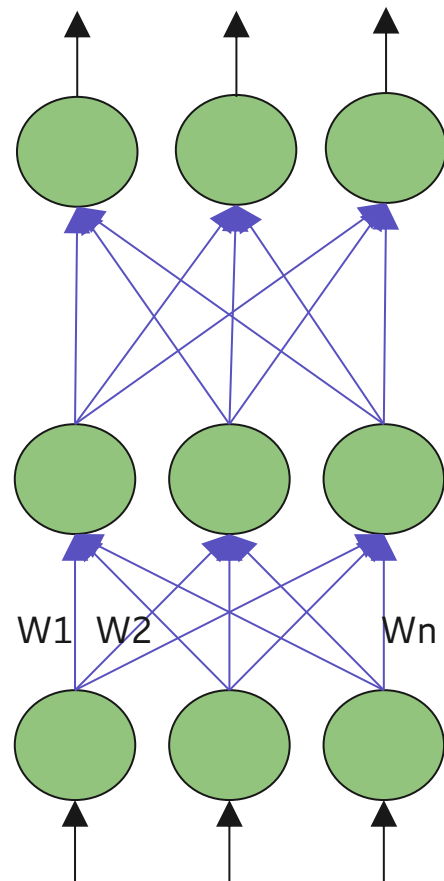


	Traditional Machine Learning	Deep Learning
The amount of training data	Different levels of data volume can be supported	A lot of data is required
Feature processing requirements	Need for clear characteristic information (supervision)	Can learn the characteristics of the data on their own (unsupervised)
The types of models available	Lot	Few
Training time	Short	Very long
Dependency on hardware	Weak	Strong
Algorithm complexity	Normal	Very complicated
Explainability	Yes	No
The number of processing logics	Single	Multilayer

How does deep learning work?



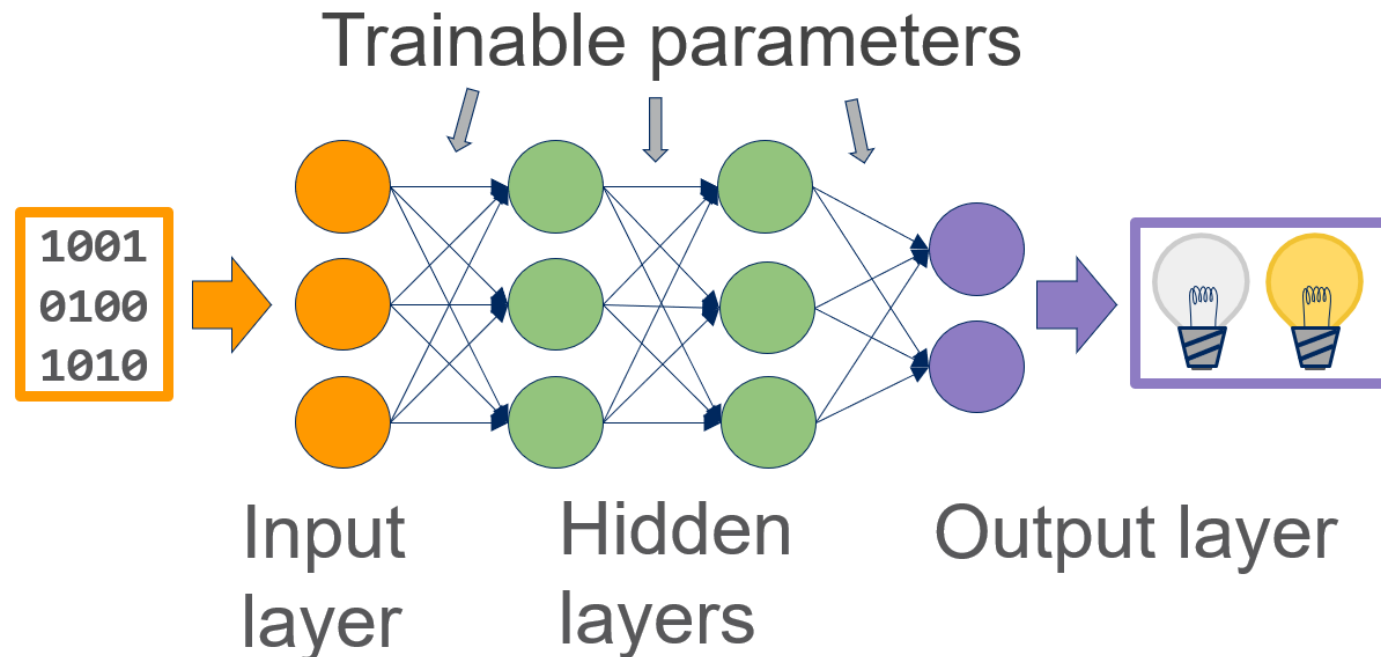
- Deep learning works by using **artificial neural networks (ANN)** to learn from data. Neural networks are made up of **layers** of interconnected **nodes (Neuron, Perceptron)**, and each node is responsible for learning a specific feature of the data.
 - Ex: In an image recognition network, the first layer of nodes might learn to identify edges, the second layer might learn to identify shapes, and the third layer might learn to identify objects
- As the network learns, the weights on the connections between the nodes are adjusted so that the network can better classify the data. This process is called training, and it can be done using a variety of techniques, such as supervised learning, unsupervised learning, and reinforcement learning.



Neuron Network(ANN or DNN)



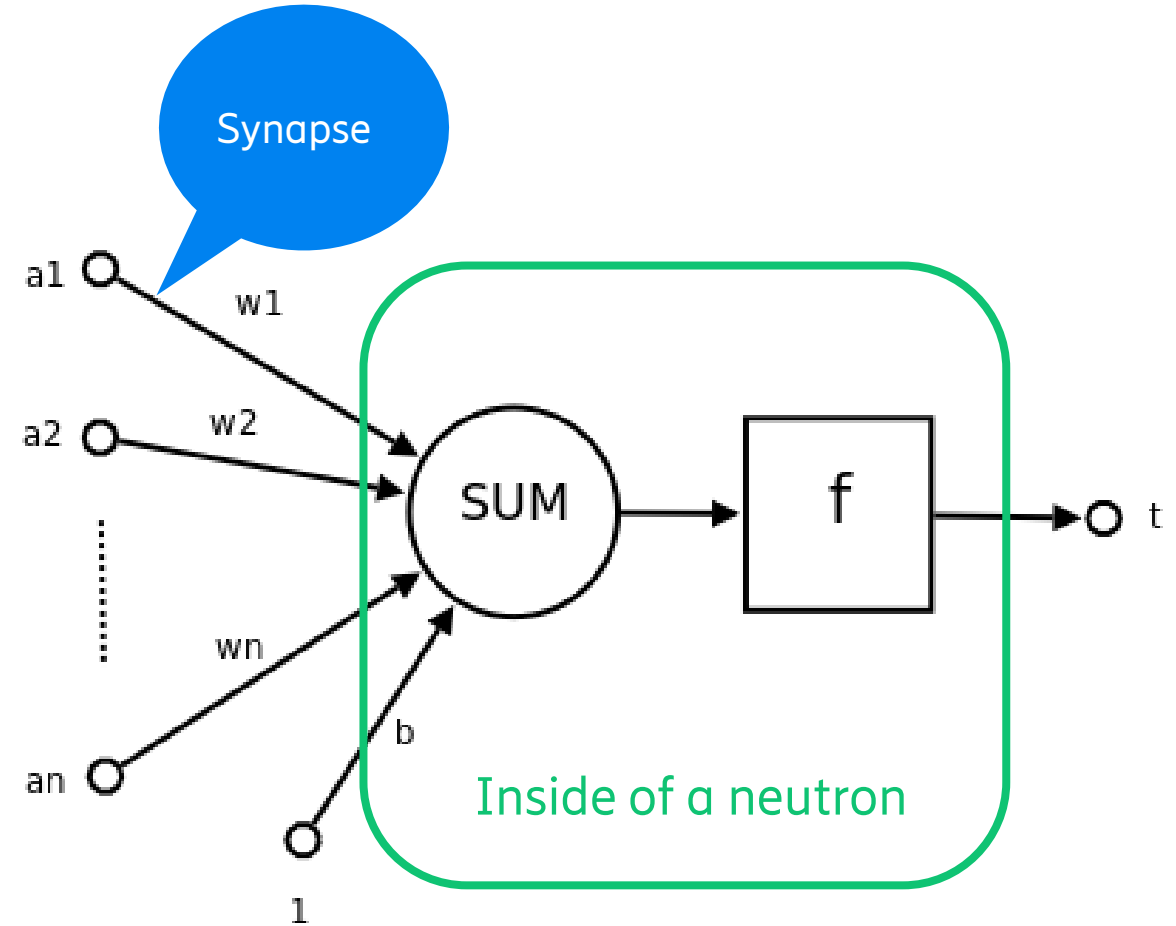
- Artificial neurons are software modules called nodes, which use mathematical calculations to process data.
- It attempt to mimic the human brain through a combination of data inputs, **weights and bias**—all acting as silicon neurons.
- Artificial Neuron Network(ANN) only consists 2 layers(input and output layers), but Deep Neuron Network (DNN) have added one or more hidden layers between input and output.



Neuron



- The composition of a typical neuron:
 - Synapses and their weights ($w_1, w_2 \dots w_n$)
 - Input bias: b
 - Linear function (**SUM**)
 - Non-linear function: **Sigmoid, ReLU ...**
 - Output result
- The function of a neuron is to obtain the inner product of the input vector (a) and the weight vector (w) and then obtain a scalar result through a nonlinear activation function.



Activation function









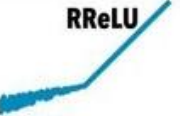
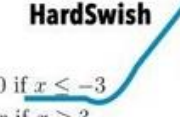
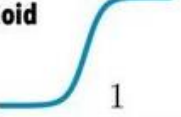
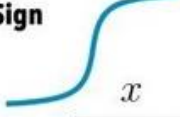

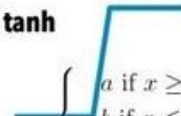
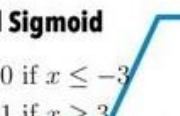
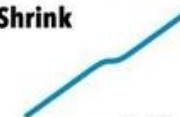
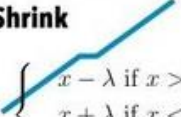
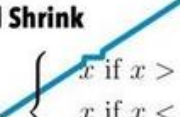


- In artificial neural networks, the activation function of a neuron defines the output of the neuron given an input or a set of inputs.
- The activation function can be linear or nonlinear, but nonlinear functions are generally chosen to solve more complex problems.
- Activation functions can be divided into:
 - Ridge activation functions
 - Linear
 - ReLU
 - Logistic
 - Radial activation functions
 - Gaussian
 - Multiquadratics
 - Folding activation functions
 - Softmax

Activation function properties



- Activation functions typically have the following properties:
 - **Non-linear** : In linear regression we're limited to a prediction equation that looks like a straight line. This is nice for simple datasets with a one-to-one relationship between inputs and outputs, but what if the patterns in our dataset were non-linear? (e.g. x^2 , \sin , \log). To model these relationships, we need a non-linear prediction equation. Activation functions provide this non-linearity.
 - **Continuously differentiable**: To improve our model with gradient descent, we need our output to have a nice slope so we can compute error derivatives with respect to weights. If our neuron instead outputted 0 or 1 (perceptron), we wouldn't know in which direction to update our weights to reduce our error.
 - **Fixed Range**: Activation functions typically squash the input data into a narrow range that makes training the model more stable and efficient.

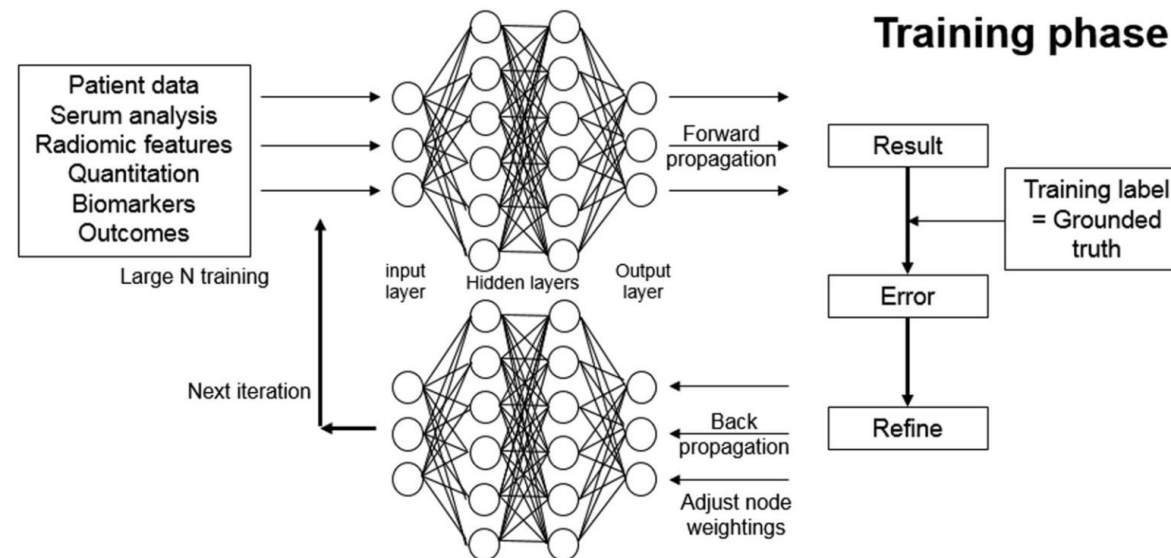
ReLU  $\max(0, x)$	GELU  $\frac{x}{2} \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + ax^3) \right) \right)$	PReLU  $\max(0, x)$
ELU  $\begin{cases} x & \text{if } x > 0 \\ \alpha(x \exp x - 1) & \text{if } x < 0 \end{cases}$	Swish  $\frac{x}{1 + \exp -x}$	SELU  $\alpha(\max(0, x) + \min(0, \beta(\exp x - 1)))$
SoftPlus  $\frac{1}{\beta} \log(1 + \exp(\beta x))$	Mish  $x \tanh \left(\frac{1}{\beta} \log(1 + \exp(\beta x)) \right)$	RReLU  $\begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \text{ with } a \sim \mathcal{R}(l, u) \end{cases}$
HardSwish  $\begin{cases} 0 & \text{if } x \leq -3 \\ x & \text{if } x \geq 3 \\ x(x+3)/6 & \text{otherwise} \end{cases}$	Sigmoid  $\frac{1}{1 + \exp(-x)}$	SoftSign  $\frac{x}{1 + x }$
Tanh  $\tanh(x)$	Hard tanh  $\begin{cases} a & \text{if } x \geq a \\ b & \text{if } x \leq b \\ x & \text{otherwise} \end{cases}$	Hard Sigmoid  $\begin{cases} 0 & \text{if } x \leq -3 \\ 1 & \text{if } x \geq 3 \\ x/6 + 1/2 & \text{otherwise} \end{cases}$
Tanh Shrink  $x - \tanh(x)$	Soft Shrink  $\begin{cases} x - \lambda & \text{if } x > \lambda \\ x + \lambda & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$	Hard Shrink  $\begin{cases} x & \text{if } x > \lambda \\ x & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$

Neural network learning (training) process



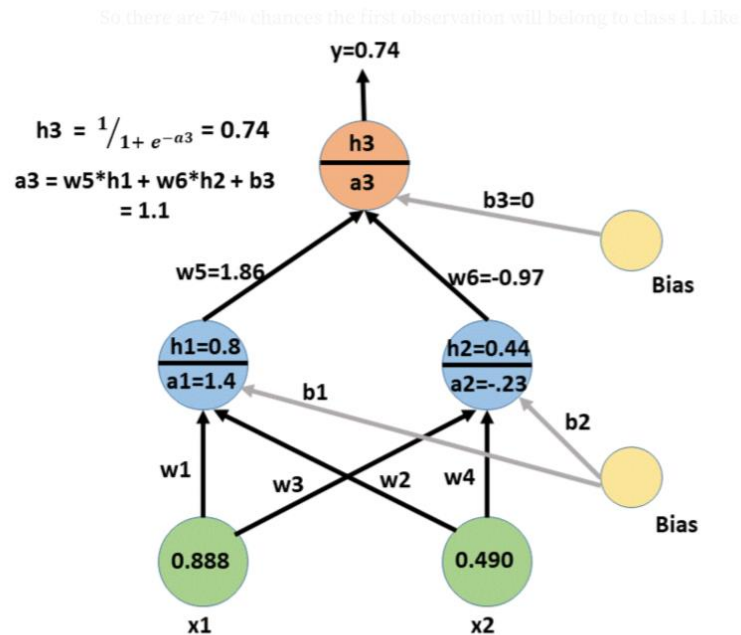
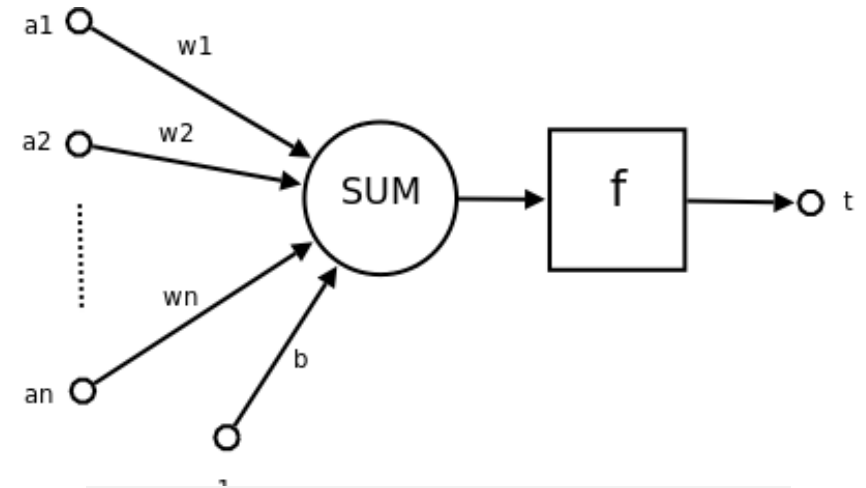
— Take **supervised learning** as an example:

- Each training data contains a known "input" and "outcome", forming a probability-weighted association between the two, which is stored in the data structure of the network itself.
- Training a neural network is usually done by determining the difference between the processed output and the target output. This difference is the error.
- The network then adjusts its weighted associations according to the learning rule and uses this error value. Successive adjustments will cause the neural network to produce outputs that are increasingly similar to the target output.
- After enough of these adjustments, training can be terminated according to certain criteria.



Forward propagation

- Forward propagation is the process of calculate the result or output of the FNN by doing the SUM and activation of each neuron in hidden layers and output layers.
 - **SUM**: $a_1*w_1+a_2*w_2 \dots a_n*w_n + b$
 - Activation Function take the out put of SUM and output **ONE** value.
 - Where **wn** will be random number and b will be zero at initial phase,.



```
class FeedForwardNetwork:

    def __init__(self):
        np.random.seed(0)
        self.w1 = np.random.randn()
        self.w2 = np.random.randn()
        self.w3 = np.random.randn()
        self.w4 = np.random.randn()
        self.w5 = np.random.randn()
        self.w6 = np.random.randn()
        self.b1 = 0
        self.b2 = 0
        self.b3 = 0

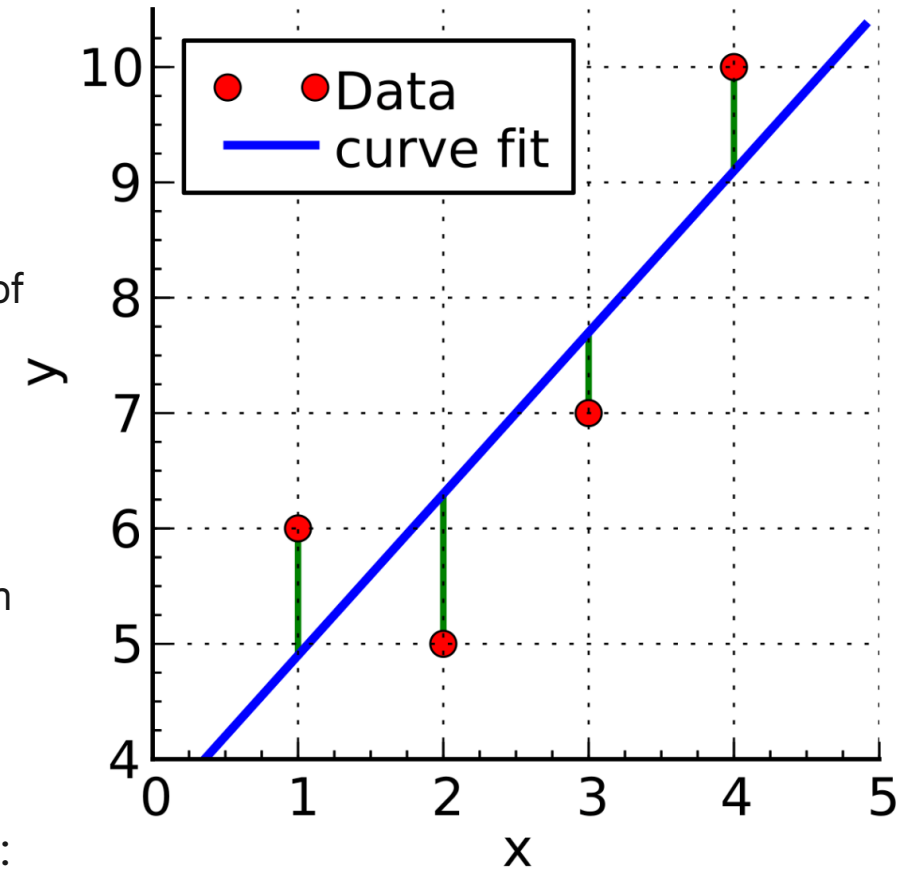
    def sigmoid(self, x):
        return 1.0/(1.0 + np.exp(-x))

    def forward_pass(self, x):
        self.x1, self.x2 = x
        self.a1 = self.w1*self.x1 + self.w2*self.x2 + self.b1
        self.h1 = self.sigmoid(self.a1)
        self.a2 = self.w3*self.x1 + self.w4*self.x2 + self.b2
        self.h2 = self.sigmoid(self.a2)
        self.a3 = self.w5*self.h1 + self.w6*self.h2 + self.b3
        self.h3 = self.sigmoid(self.a3)
        forward_matrix = np.array([[0,0,0,0,self.h3,0,0,0],
                                   [0,0,(self.w5*self.h1),
                                   (self.w6*self.h2),self.b3,self.a3,0,0],
                                   [0,0,0,self.h1,0,0,0,self.h2],
                                   [(self.w1*self.x1), (self.w2*self.x2),
                                   self.b1, self.a1,(self.w3*self.x1),(self.w4*self.x2), self.b2,
                                   self.a2]])
        forward_matrices.append(forward_matrix)
        return self.h3
```

Loss Functions



- The machine learning algorithm is to find the optimal model parameters
- The main implementation method is to optimize the loss function (Loss Function or Cost Function)
 - The function of the difference between the predicted value and the actual value
 - Once the loss function is obtained, calculating its minimum value is the process of narrowing the gap between the predicted value and the actual value
- Main loss function
 - Square loss function (least squares method)
 - LogLoss loss function: used for classification problems such as logistic regression
 - Cross-Entropy Loss Function
 - 0-1 loss function
- Finding the minimum value of the loss function is an optimization process:
gradient descent

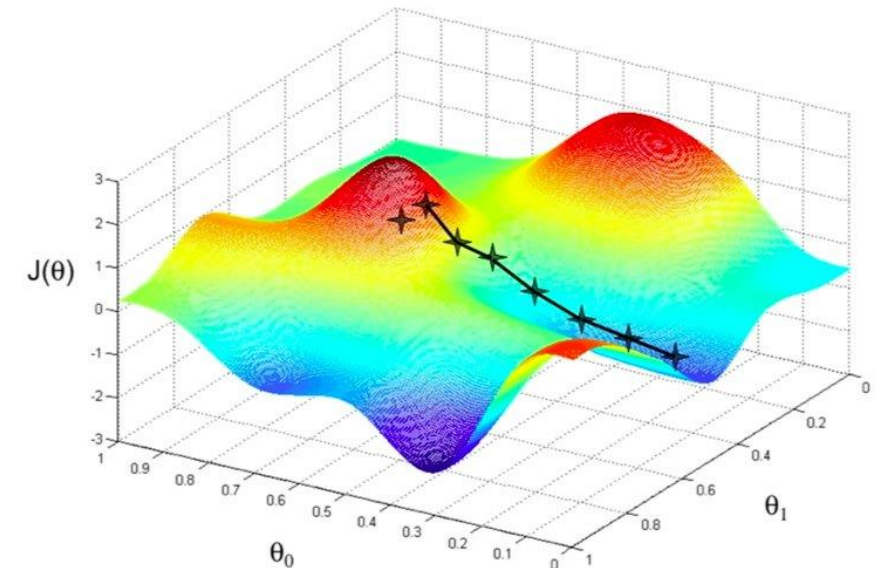


Gradient descent



- The most used algorithm for optimizing loss functions (model algorithms) is gradient descent.
- Gradient descent :
 - To find the local minimum of a function, it is necessary to iteratively search for points at a specified step distance in the opposite direction of the gradient (or approximate gradient) corresponding to the current point on the function.
 - Any starting point
 - Step size (learning rate):
 - Longer will speed up learning, but may miss the lowest point
 - Shorter will extend the learning time, but the learning accuracy will be higher
- Types of gradient descent algorithms:
 - Batch Gradient Descent (BGD)
 - Stochastic Gradient Descent (SGD)
 - Mini-batch Gradient Descent (MBDG)

Gradient Descent

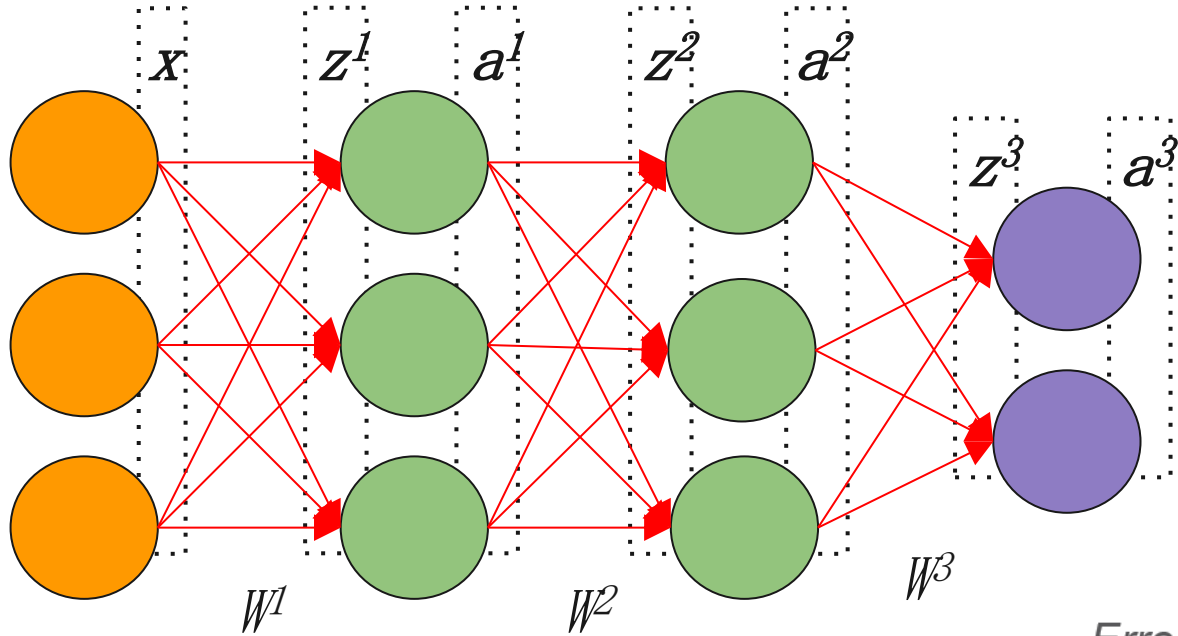


Backward propagation – backpropagation



- Backpropagation is a gradient estimation method commonly used for training neural networks to compute the network parameter updates.
- It is an efficient application of the chain rule to neural networks.
- Backpropagation computes the **gradient of a loss function** with respect to the **weights and bias** of the network for a single input–output example, and does so efficiently, computing the gradient **one layer at a time**, iterating backward **from the last layer** to avoid redundant calculations of intermediate terms in the **chain rule**; this can be derived through dynamic programming.
- Strictly speaking, the term backpropagation refers only to an algorithm for efficiently computing the gradient, not how the gradient is used; but the term is often used loosely to refer to the entire learning algorithm – including:
 - how the gradient is used, such as by stochastic gradient descent,
 - or as an intermediate step in a more complicated optimizer, such as Adam

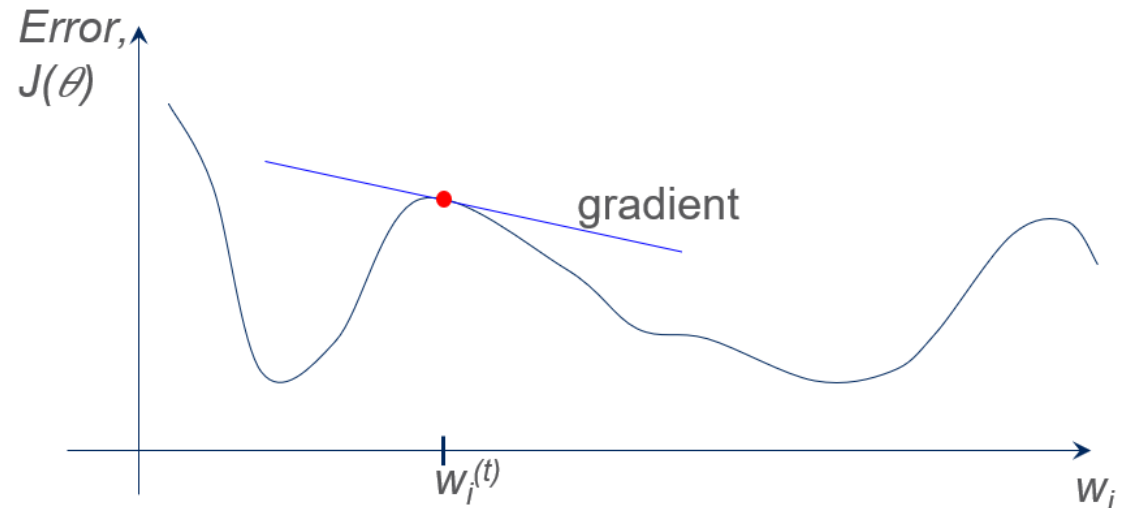
Backpropagation in equations



$J(\theta)$

$$J(\theta) = \frac{1}{N} \sum_{n=1}^N L(\mathbf{y}_n, \mathbf{a}_n^3)$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta \frac{\partial J(\theta)}{\partial w_i^{(t)}}$$



Optimizer for Gradient descent

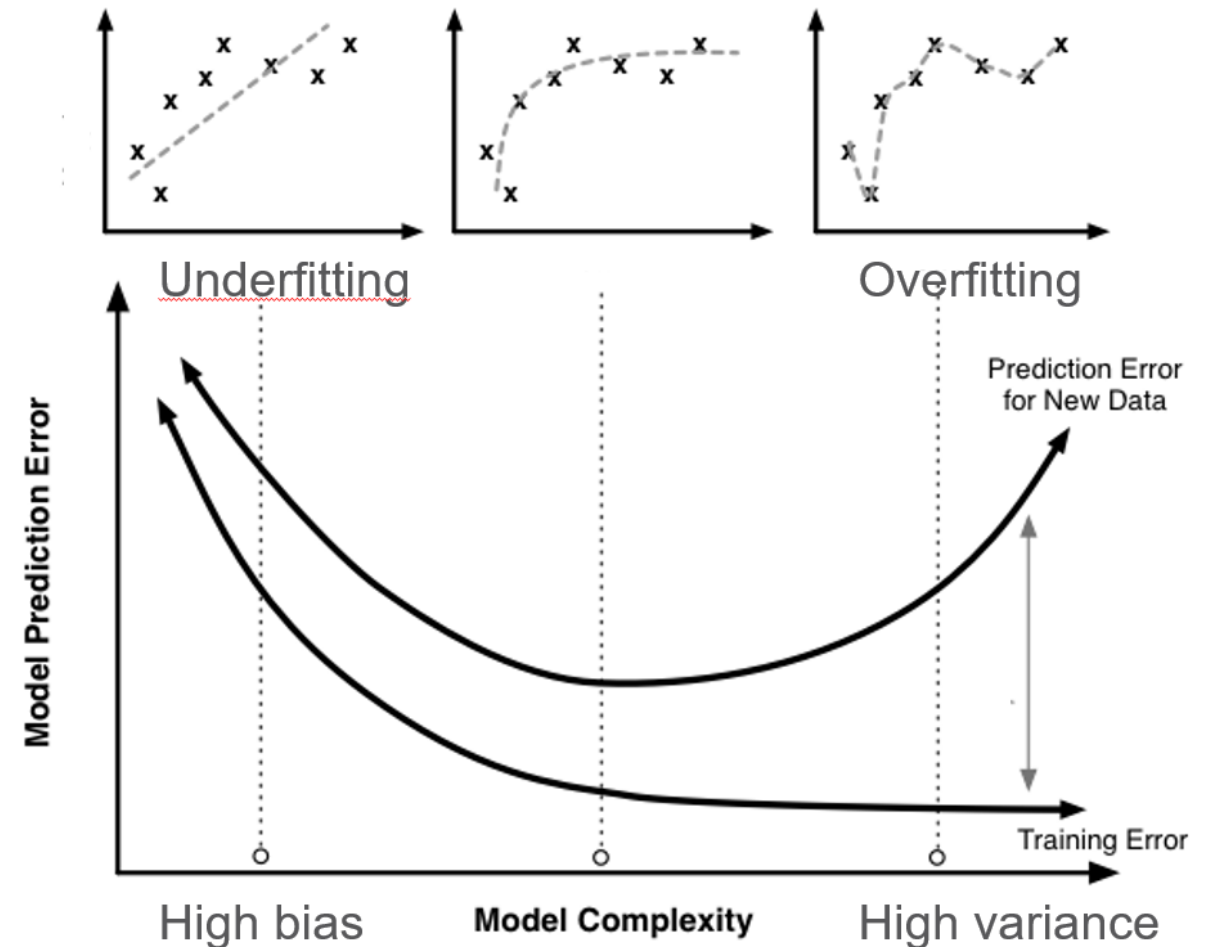


- A conceptually simple extension of stochastic gradient descent makes the learning rate a decreasing function η of the iteration number t , giving a learning rate schedule, so that the first iterations cause large changes in the parameters, while the later ones do only fine-tune.
- Momentum Optimization:
 - SGD+Momentum: Momentum is introduced based on SGD, which accelerates the convergence of SGD in the relevant direction by accumulating the previous gradient direction and suppresses oscillation.
 - NAG (Nesterov Accelerated Gradient): An improved momentum optimization method, which estimates the parameters one step before calculating the current gradient, to more accurately guide the parameter update direction.
- Adaptive(Ada) learning rate optimization algorithm:
 - AdaGrad: Adjusts the learning rate of each parameter by accumulating the sum of squares of all previous gradients, which is particularly effective for sparse data scenarios.
 - AdaDelta: An improved version of AdaGrad, which avoids the problem of too fast decay of learning rate by limiting the accumulation speed of the sum of squared gradients.
 - Adam: Combines the first-order momentum of SGDM and the second-order momentum of RMSProp and adds two correction terms to correct the deviation on this basis, with faster convergence speed and better robustness.

Overfitting and Underfitting



- An important indicator for evaluating supervised machine learning models is Generalization Error or Prediction Error, which refers to the error rate of the trained model for unknown test data.
- If:
 - The error rate of the training process is low, but the Generalization Error is high, it is called Overfitting, or High variance
 - If the error rate of the training process is very high, it is called Underfitting, or High bias
- The problem that often needs to be solved is overfitting. Because there is always a tendency to reduce the error rate of the training process, which will make the model too "complex", too suitable for training data, and very poor "adaptability" to unknown data.
- The more complex the model, the more likely it is overfit



Regularization



- Regularization is a set of methods for reducing overfitting in machine learning models. Typically, regularization trades a marginal decrease in training accuracy for an increase in generalizability.
- In machine learning, a key challenge is enabling models to accurately predict outcomes on **unseen data**, not just on familiar training data. Regularization is crucial for addressing overfitting—where a model memorizes training data details but can't generalize to new data—and underfitting, where the model is too simple to capture the training data's complexity.
- Common methods of regularization:
 - Early Stopping
 - Stops training when validation performance deteriorates, preventing overfitting by halting before the model memorizes training data.
 - L1 and L2 Regularization
 - L1 regularization (also called LASSO) leads to sparse models by adding a penalty based on the absolute value of coefficients.
 - L2 regularization (also called ridge regression) encourages smaller, more evenly distributed weights by adding a penalty based on the square of the coefficients.
 - Dropout
 - Randomly ignores a subset of neurons during training, simulating training multiple neural network architectures to improve generalization.

Python-based machine (deep) learning tools



— scikit-learn

- Scikit-learn is an open source (Python) machine learning library that supports supervised and unsupervised learning. It also provides a variety of tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities.



— TensorFlow

- TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive and flexible ecosystem of tools, libraries, and community resources that enable researchers to advance the state-of-the-art in machine learning and developers to easily build and deploy machine learning-driven applications.



— PyTorch

- PyTorch is an open source Python machine learning library based on Torch. It is implemented in C++ and is used in artificial intelligence fields such as computer vision and natural language processing. It is mainly developed by the artificial intelligence research team of Meta Platforms.



— Keras

- Keras is an open source neural network library written in Python, is designed to quickly implement deep neural networks, focusing on user-friendliness, modularity, and extensibility.



Classic Deep learning models

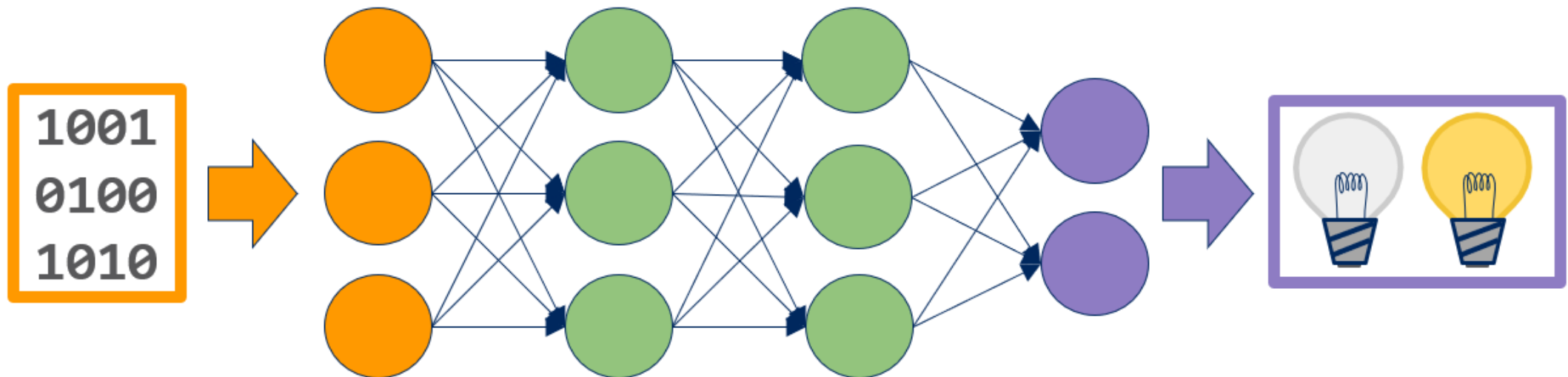


- From a historical perspective, the theoretical basis of deep learning appeared very early, but due to the limitation of computing power, it was not until after 2010, when CPU and GPU were widely adopted and matured, that machine (deep) learning was applied to more practical cases.
 - Convolutional neural network (CNN) : Image Recognition
 - Recurrent Neural Network (RNN) : Sequential data, NLP
 - Long short-term memory (LSTM)
 - gated recurrent units (GRU)
 - Generative Model:
 - generative adversarial network (GAN) : Unsupervised learning, two models against each other
 - Transformer:
 - Generative Pre-trained Transformer (GPT)
 - generative diffusion model (GDM)

Feedforward neural network



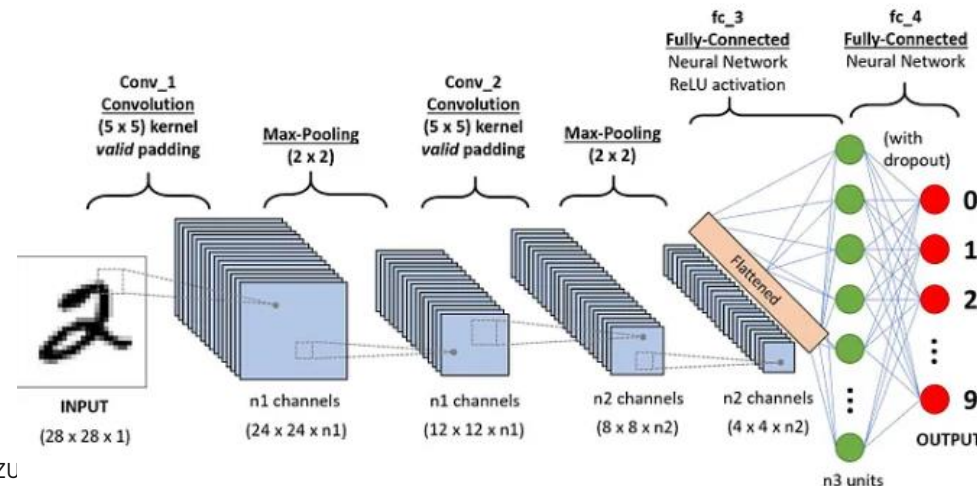
- A feedforward neural network (FNN) is one of the two broad types of artificial neural network, characterized by direction of the flow of information between its layers.
- Its flow is uni-directional, meaning that the information in the model flows in only one direction—forward—from the input nodes, through the hidden nodes (if any) and to the output nodes, without any cycles or loops, in contrast to recurrent neural networks, which have a bi-directional flow.
- Modern feedforward networks are trained using the backpropagation method and are colloquially referred to as the "vanilla" neural networks.



Convolutional neural network - CNN



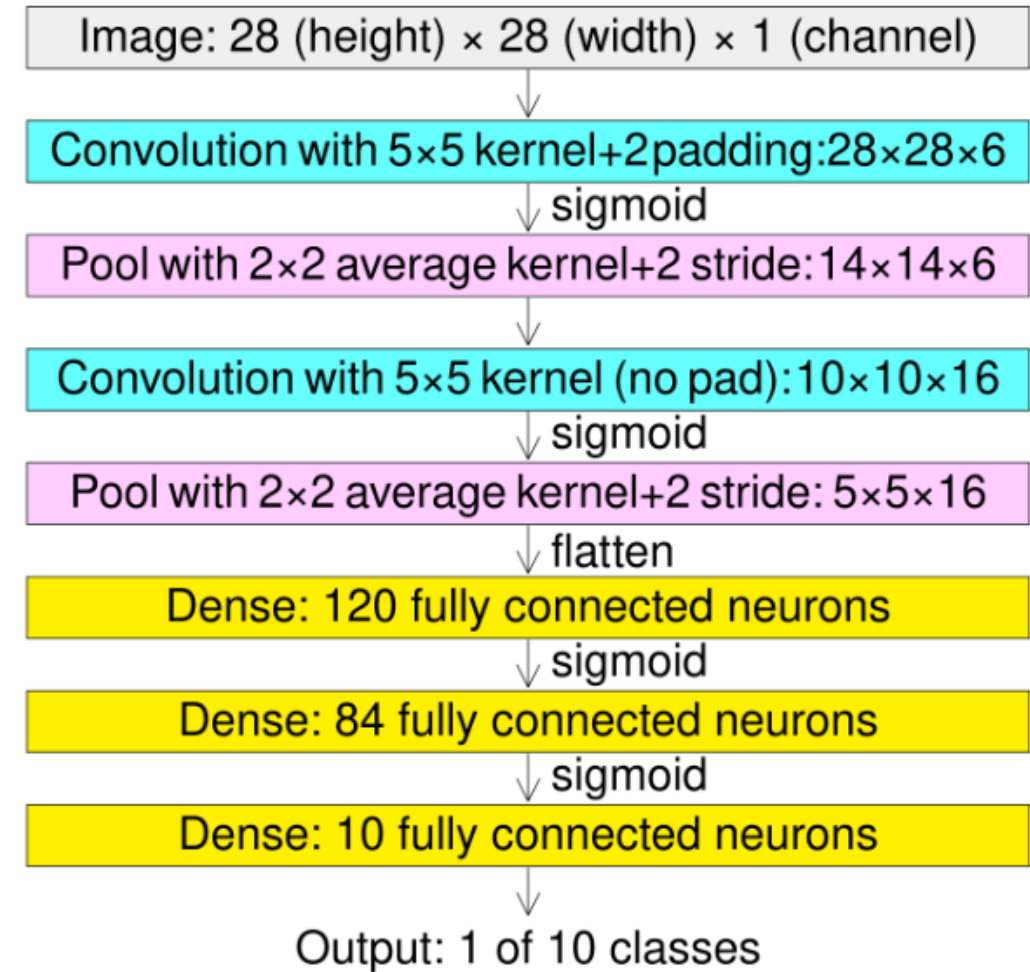
- A convolutional neural network (CNN) is a regularized type of feed-forward neural network **that learns features by itself via filter (or kernel) optimization.**
- This type of deep learning network has been applied to process and make predictions from many different types of data including text, images and audio.
- Vanishing gradients and exploding gradients, seen during backpropagation in earlier neural networks, are prevented by using regularized weights over fewer connections.
- CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered.



Architecture of CNN



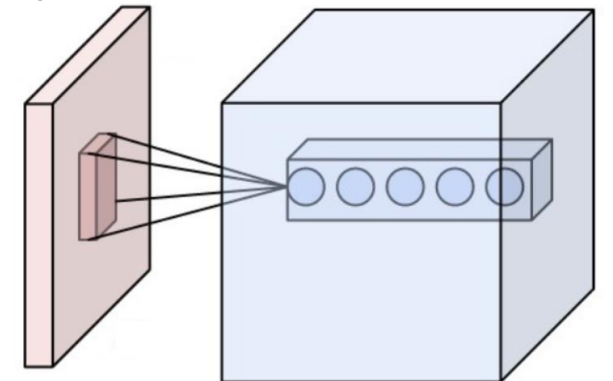
- In a convolutional neural network, the hidden layers include one or more layers that perform convolutions.
- Typically, this includes a layer(ConvNet) that performs a **dot product of the convolution kernel** with the layer's input matrix. This product is usually the Frobenius inner product, and its activation function is commonly ReLU.
- As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a **feature map**, which in turn contributes to the input of the next layer. This is followed by other layers such as **pooling layers**, fully connected layers, and normalization layers. Here it should be noted how close a convolutional neural network is to a matched filter



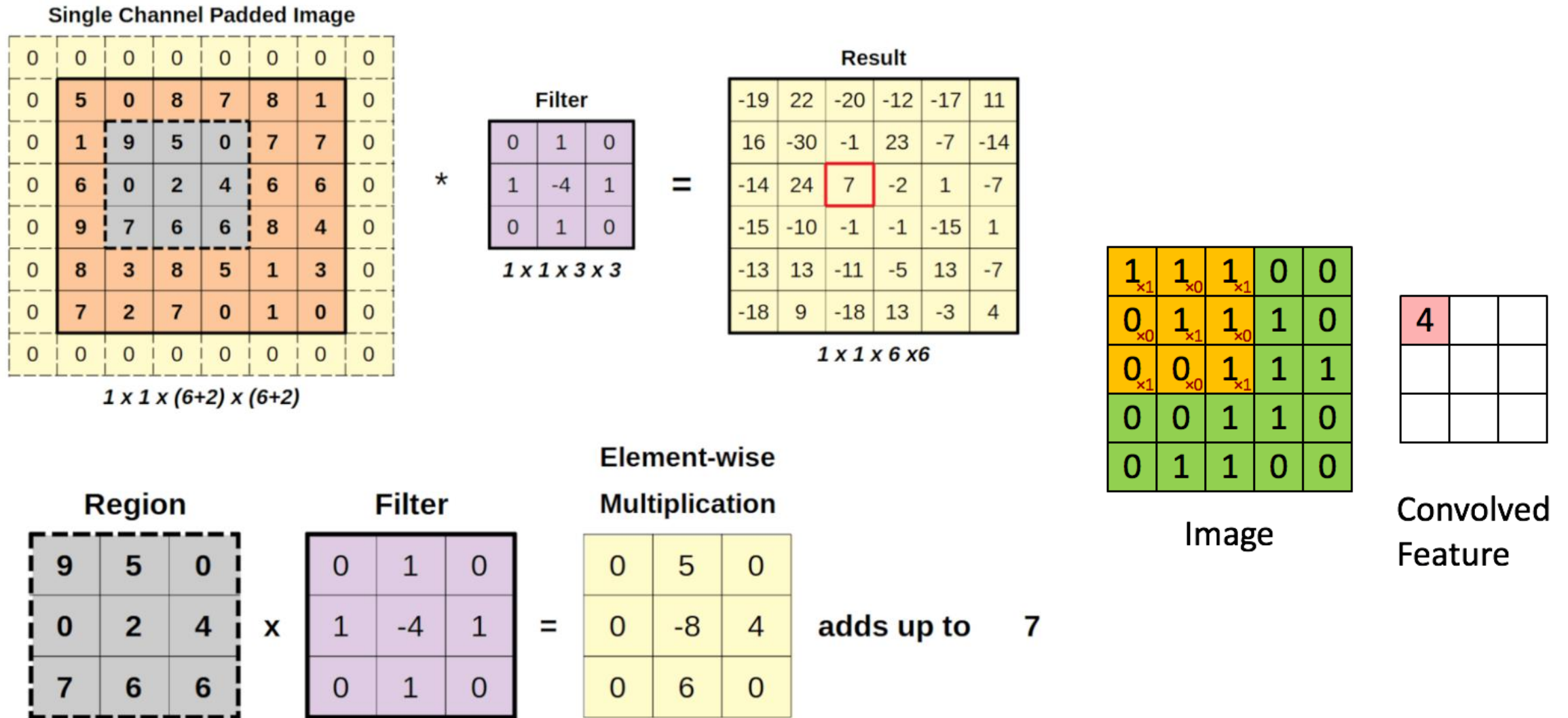
Convolutional layers - ConvNet



- The role of ConvNet is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction. This is important when we are to design an architecture that is not only good at learning features but also scalable to massive datasets.
- The layer's parameters consist of a set of learnable **filters (or kernels)**, which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the filter entries and the input, producing a 2-dimensional activation map of that filter.
- Convolutional networks exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume.
- The extent of this connectivity is a hyperparameter called the receptive field of the neuron. The connections are local in space (along width and height) but always extend along the entire depth of the input volume. Such an architecture ensures that the learned filters produce the strongest response to a spatially local input pattern.



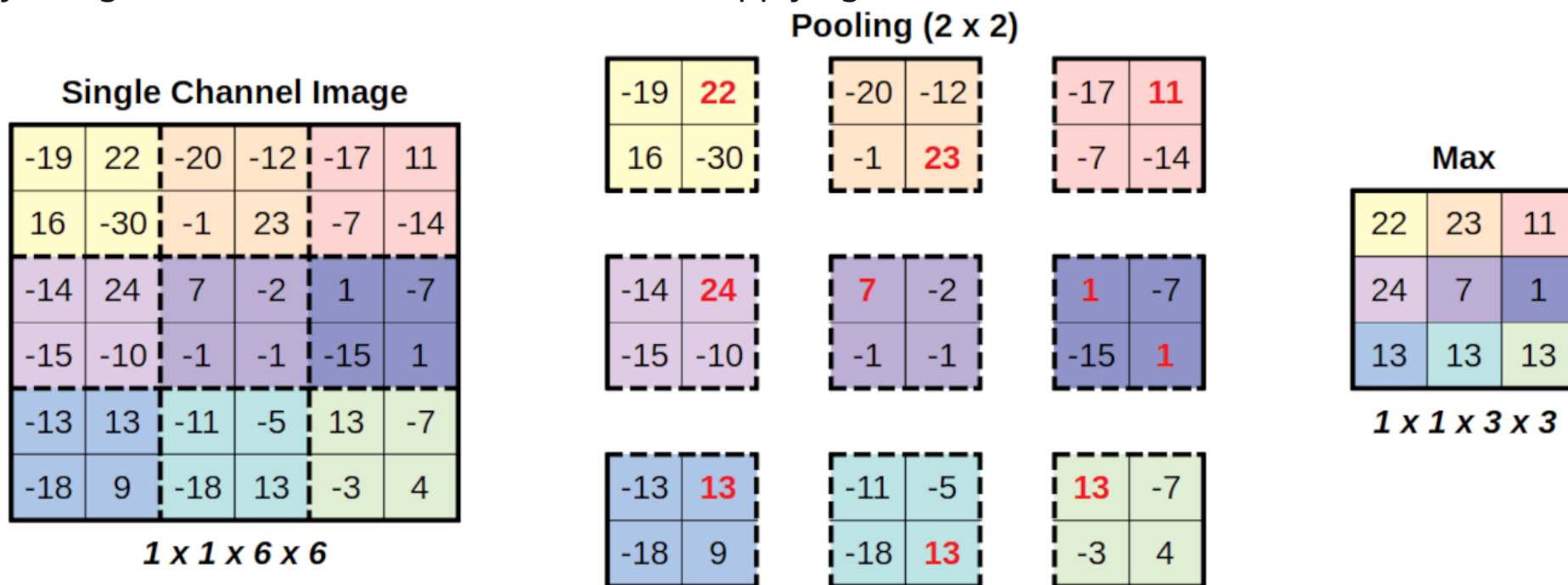
How convolution perform in ConvNet



Pooling Layer in CNN



- Another important concept of CNNs is pooling, which is used as a form of non-linear down-sampling.
- There are several non-linear functions to implement pooling, where **max pooling** and **average pooling** are the most common.
- Pooling aggregates information from small regions of the input creating partitions of the input feature map, typically using a fixed-size window (like 2x2) and applying a stride (often 2) to move the window across the input.



Other layers in CNN

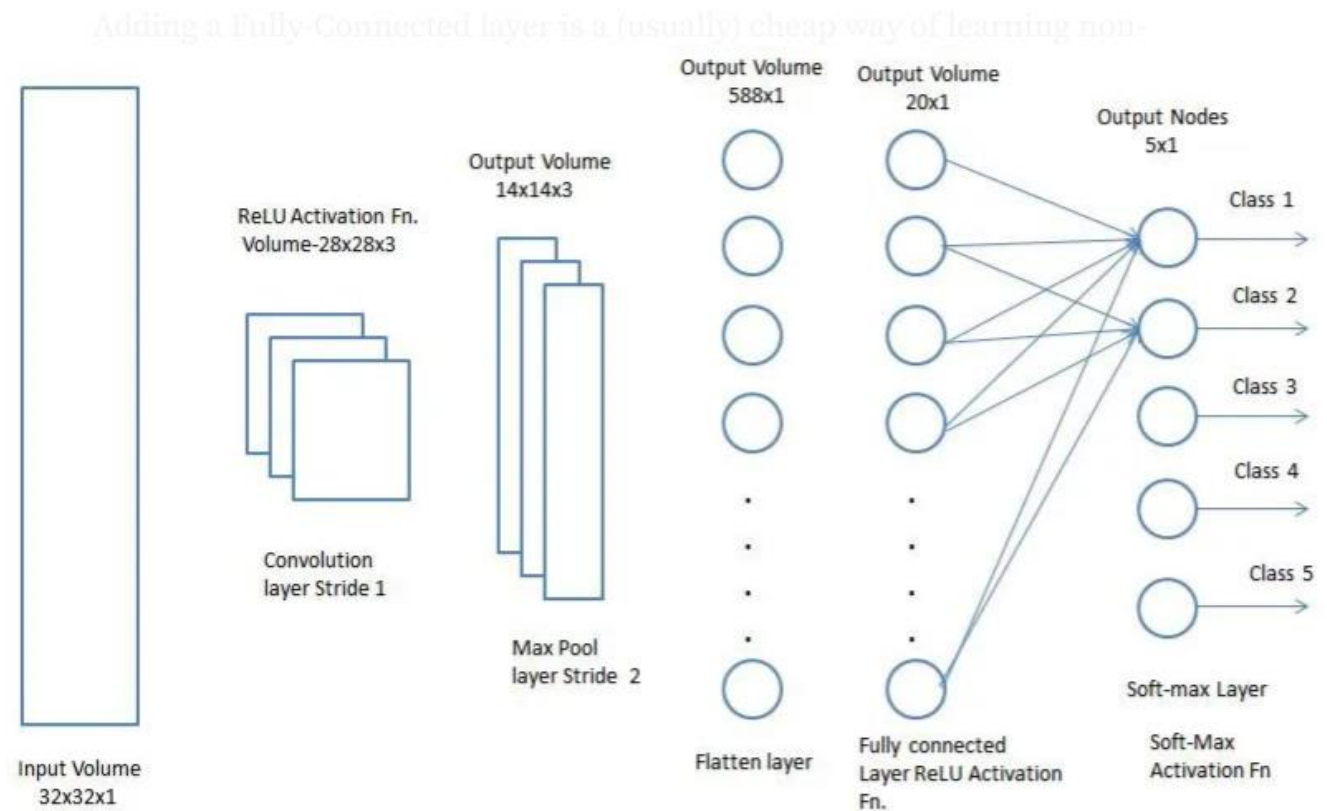


— ReLU layer

- ReLU effectively removes negative values from an activation map by setting them to zero. It introduces nonlinearity to the decision function and in the overall network without affecting the receptive fields of the convolution layers.

— Fully connected layer

- After several convolutional and max pooling layers, the final classification is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks.
- This layer performs the task of classification based on the features extracted through the previous layers and their different filters, mostly using SoftMax.



Recurrent neural networks - RNN

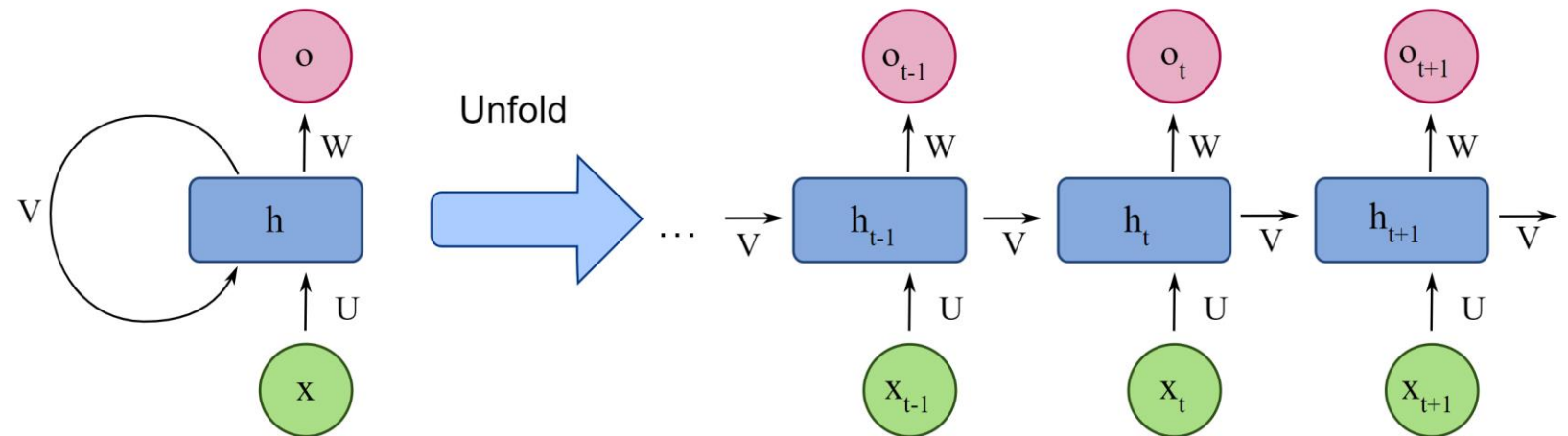


- RNNs are a class of artificial neural network commonly used for sequential data processing. Unlike feedforward neural networks, which process data in a single pass, RNNs process data across multiple time steps, making them well-adapted for modelling and processing text, speech, and time series.
- The building block of RNNs is the **recurrent unit**. This unit maintains a hidden state, essentially a form of **memory**, which is updated at each time step based on the current input and the previous hidden state. This **feedback loop** allows the network to learn from past inputs and incorporate that knowledge into its current processing.
- Early RNNs suffered from the vanishing gradient problem, limiting their ability to learn long-range dependencies. This was solved by the long short-term memory (LSTM) variant in 1997, thus making it the standard architecture for RNN.
- RNNs have been applied to tasks such as unsegmented, connected handwriting recognition, speech recognition, natural language processing, and neural machine translation.

RNN Configurations(1/4)



- An RNN-based model can be factored into two parts: **configuration** and **architecture**. Multiple RNN can be combined in a data flow, and the data flow itself is the configuration. Each RNN itself may have any architecture, including LSTM, GRU, etc.
- Standard RNN:
 - an RNN is a function f of $(x_t, h_t) \rightarrow (y_t, h_{t+1})$ where
 - x_t : input vector
 - h_t : hidden vector
 - y_t : output vector

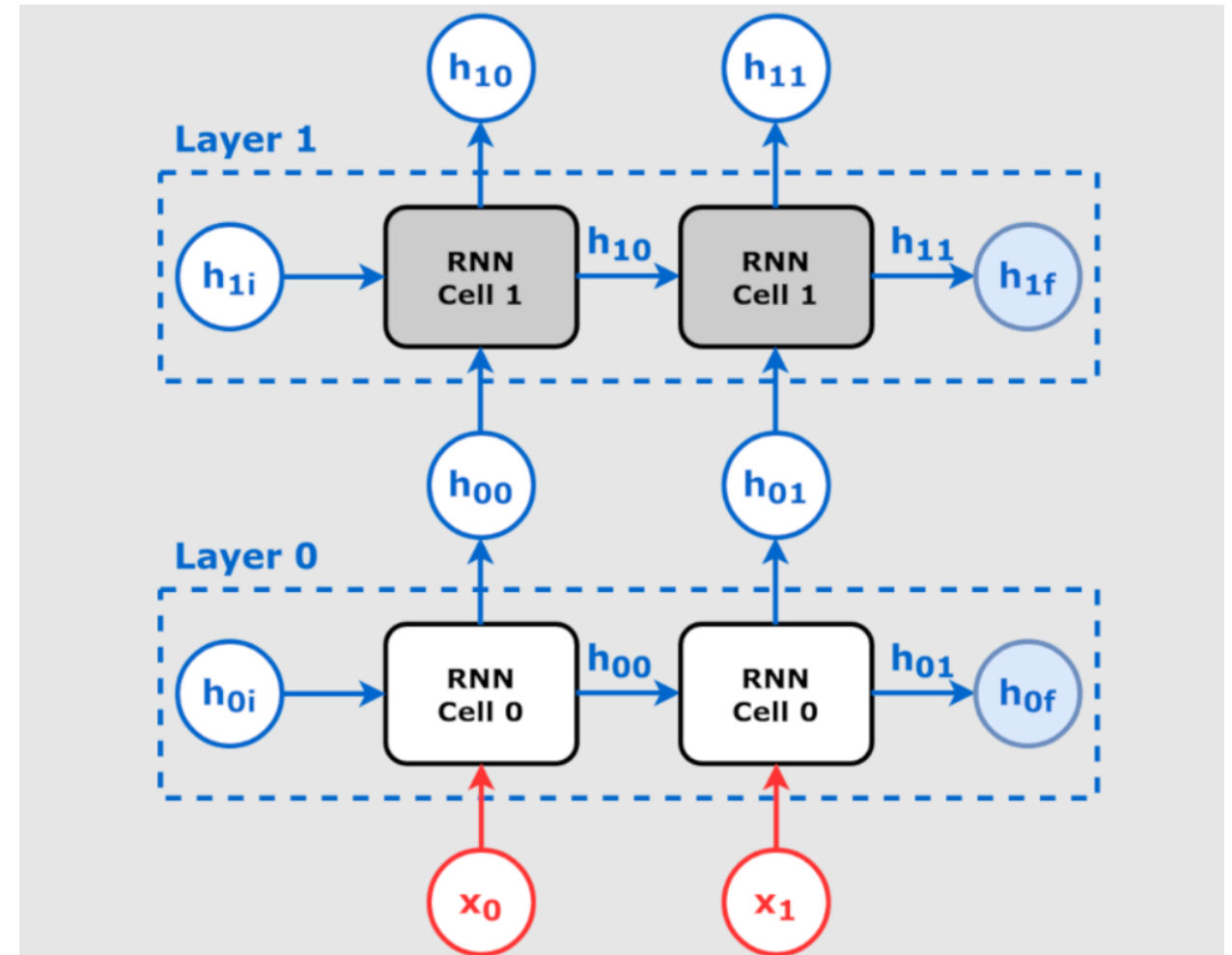


RNN Configurations (2/4)



— Stacked RNN:

- A stacked RNN, or deep RNN, is composed of multiple RNNs stacked one above the other.
- Each layer operates as a stand-alone RNN, and each layer's output sequence is used as the input sequence to the layer above. There is no conceptual limit to the depth of the RNN.



RNN Configurations (3/4)



— Bidirectional RNN:

- A bidirectional RNN (biRNN) is composed of two RNNs, one processing the input sequence in one direction, and another in the opposite direction. Abstractly, it is structured as follows:

- The forward RNN processes in one direction:

$$f_{\theta}(x_0, h_0) = (y_0, h_1), f_{\theta}(x_1, h_1) = (y_1, h_2), \dots$$

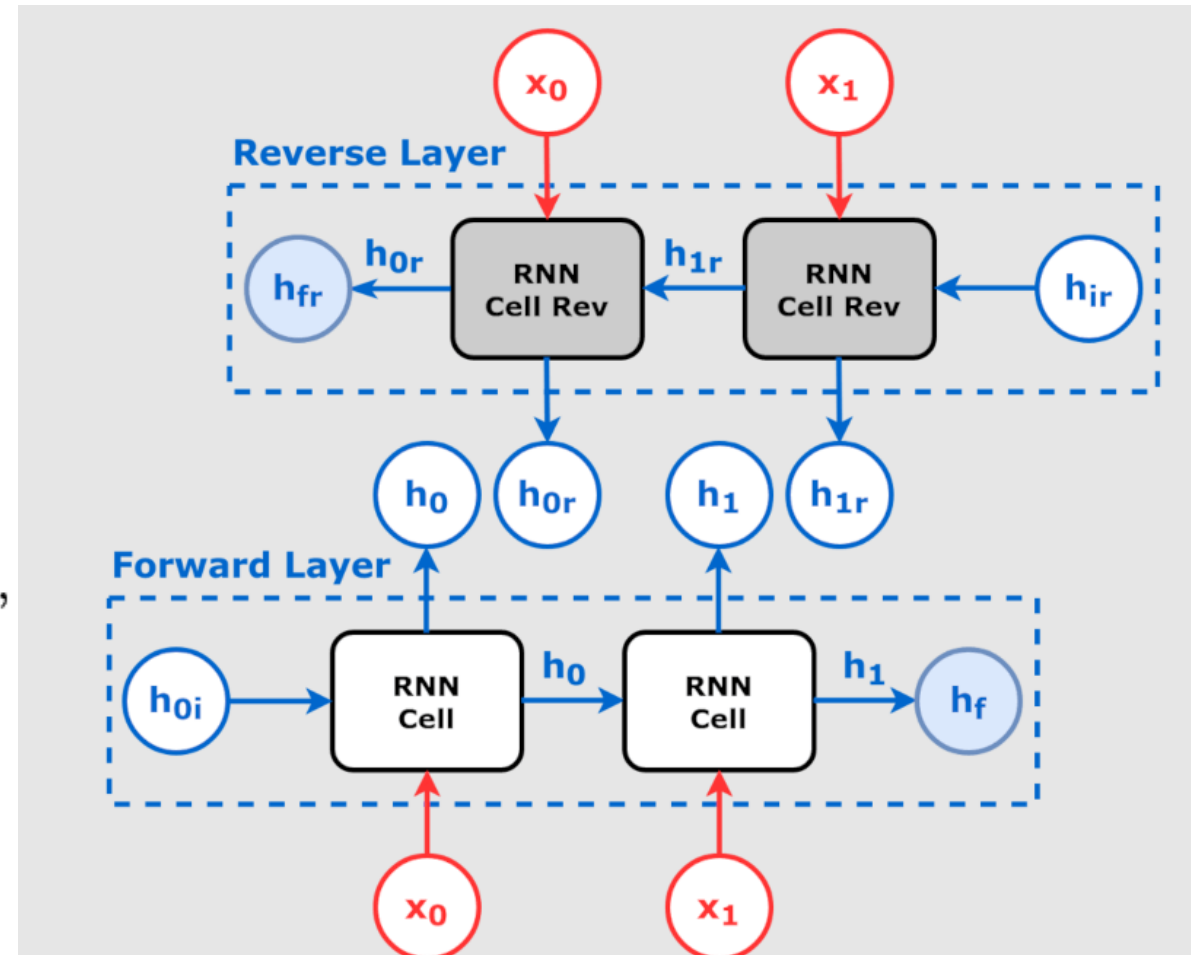
- The backward RNN processes in the opposite direction:

$$f'_{\theta'}(x_N, h'_N) = (y'_N, h'_{N-1}), f'_{\theta'}(x_{N-1}, h'_{N-1}) = (y'_{N-1}, h'_{N-2}), \dots$$

- The two output sequences are then concatenated to give the total output:

$$((y_0, y'_0), (y_1, y'_1), \dots, (y_N, y'_N)).$$

- Bidirectional RNN allows the model to process a token both in the context of what came before it and what came after it. By stacking multiple bidirectional RNNs together, the model can process a token increasingly contextually.

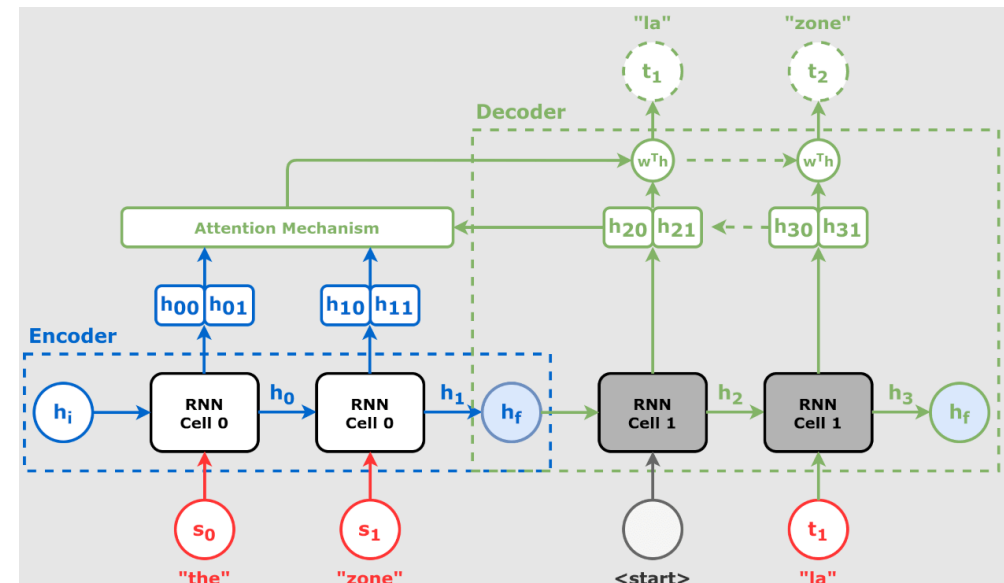
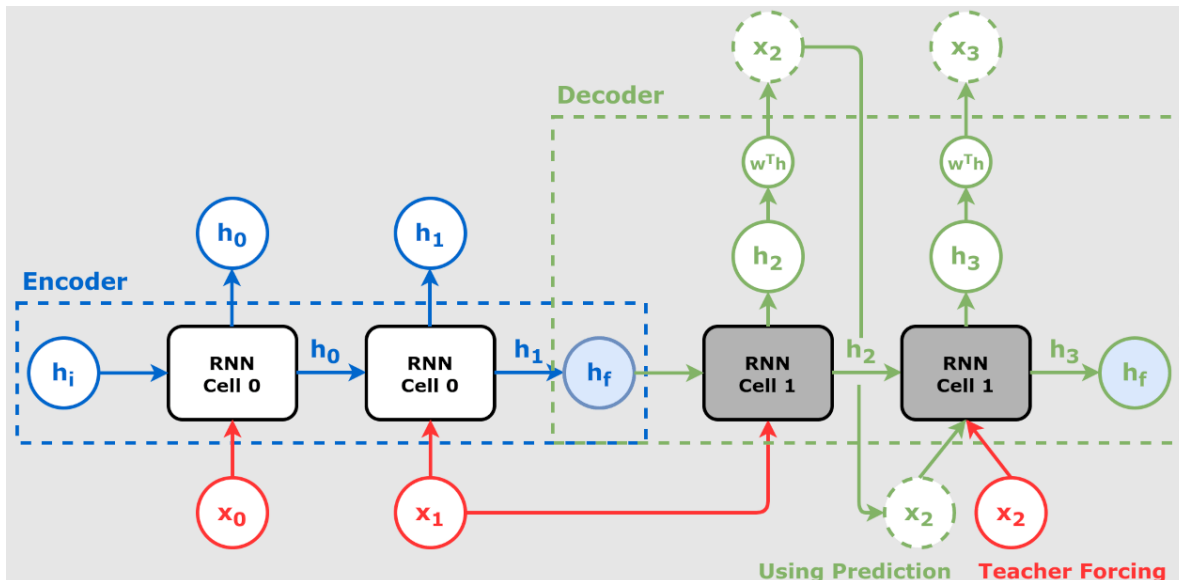


RNN Configurations (4/4)



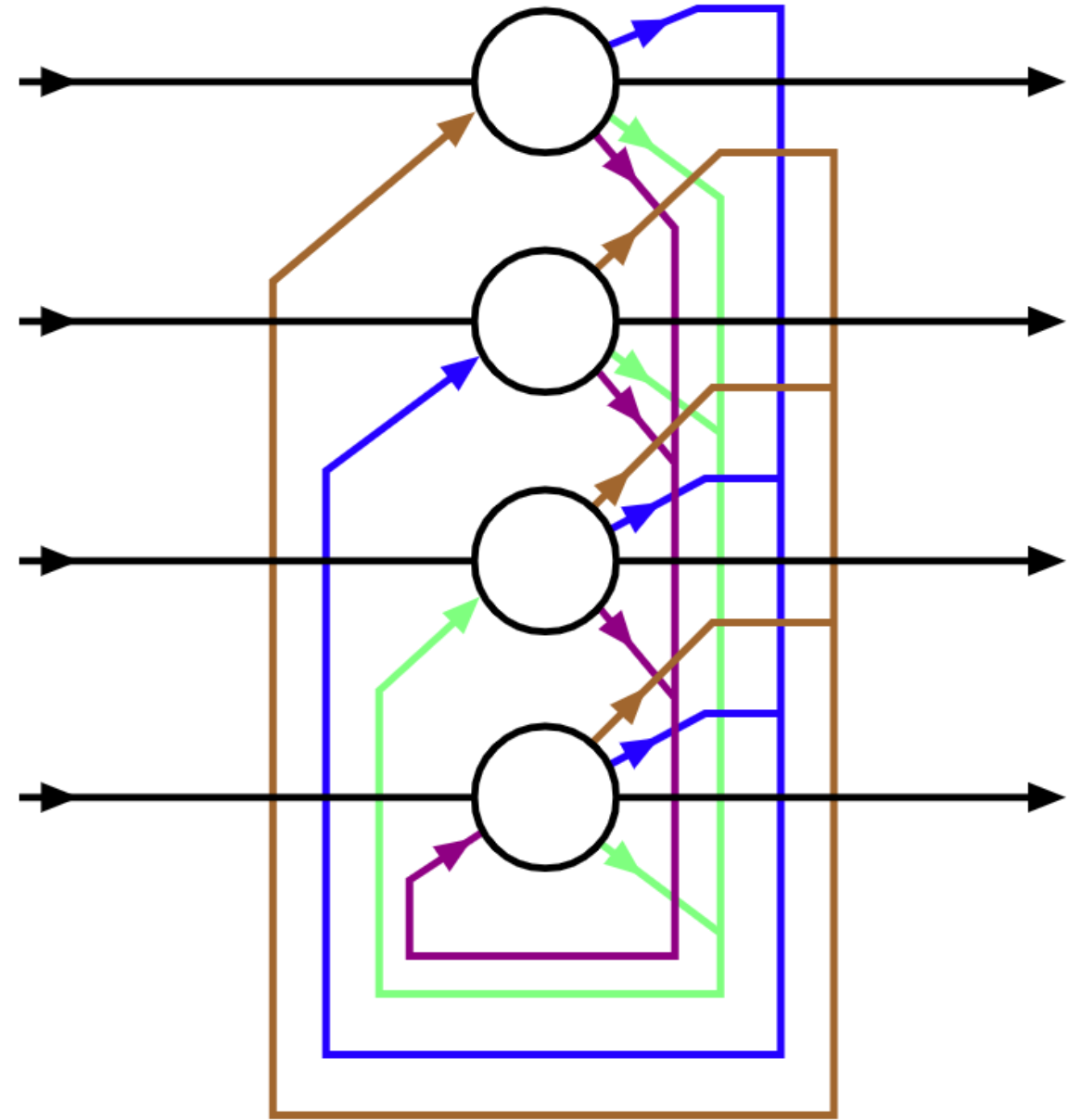
— Encoder-decoder(seq2seq)

- Two RNN can be run front-to-back in an encoder-decoder configuration.
- The encoder RNN processes an input sequence into a sequence of hidden vectors, and the decoder RNN processes the sequence of hidden vectors to an output sequence, with **an optional attention mechanism**.
- This was used to construct state of the art neural machine translators during the 2014–2017 period. This was an instrumental step towards the development of **Transformers**.



RNN Architectures (1/3)

- Fully recurrent neural networks (FRNN) connect the outputs of all neurons to the inputs of all neurons.
- In other words, it is a fully connected network. This is the most general neural network topology, because all other topologies can be represented by setting some connection weights to zero to simulate the lack of connections between those neurons.

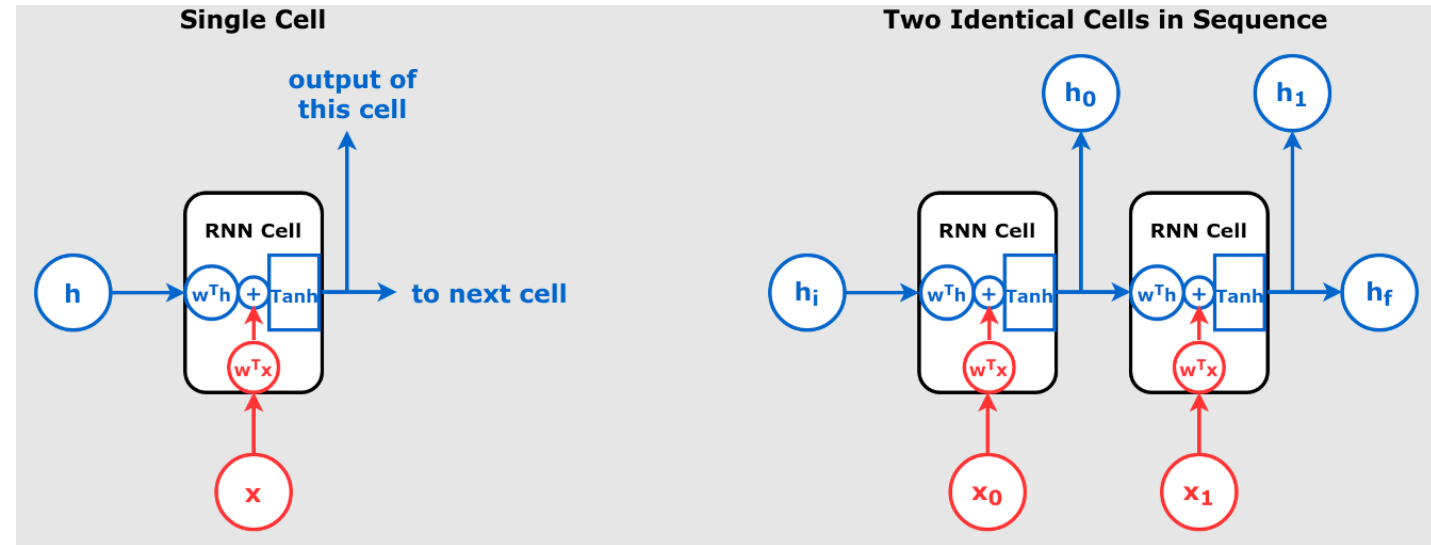


RNN Architectures (2/4)



— Elman network:

- Most common RNN architecture
- At each time step, the input is fed forward and a learning rule is applied.
- The fixed back-connections save a copy of the previous values of the hidden units in the context units (since they propagate over the connections before the learning rule is applied).
- Thus, the network can maintain a sort of state, allowing it to perform tasks such as sequence-prediction that are beyond the power of a standard multilayer perceptron.

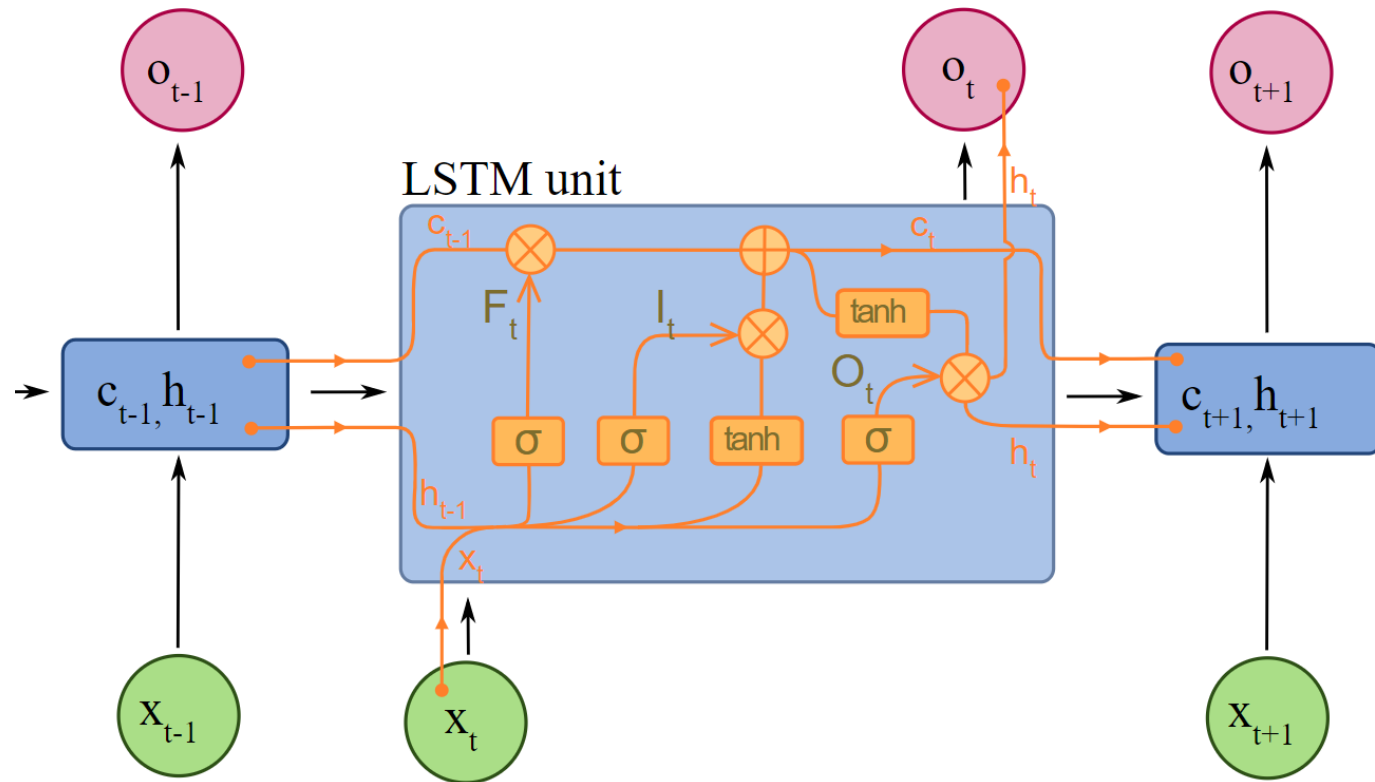


$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

RNN Architectures (3/4)



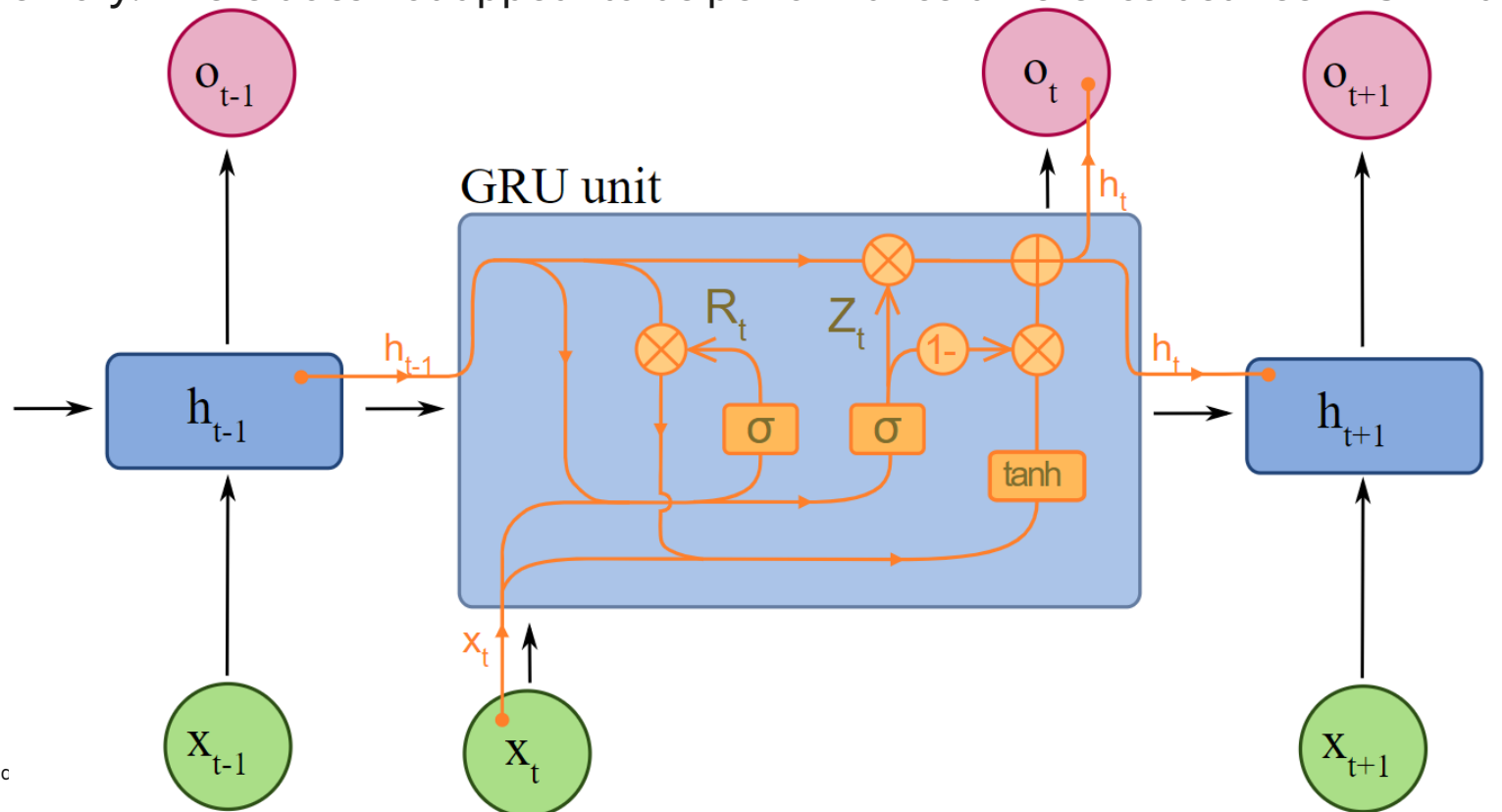
- Long short-term memory (LSTM) is the most widely used RNN architecture. It was designed to solve the vanishing and exploding gradient problem.
- LSTM is normally augmented by recurrent gates called "forget gates". LSTM prevents backpropagated errors from vanishing or exploding. Instead, errors can flow backward through unlimited numbers of virtual layers unfolded in space.
- LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier. Problem-specific LSTM-like topologies can be evolved. LSTM works even given long delays between significant events and can handle signals that mix low and high-frequency components.



RNN Architectures (4/4)



- Gated recurrent unit (GRU), introduced in 2014, was designed as a simplification of LSTM. They are used in the full form and several further simplified variants. They have fewer parameters than LSTM, as they lack an output gate.
- Their performance on polyphonic music modeling and speech signal modeling was found to be like that of long short-term memory. There does not appear to be performance difference between LSTM and GRU.



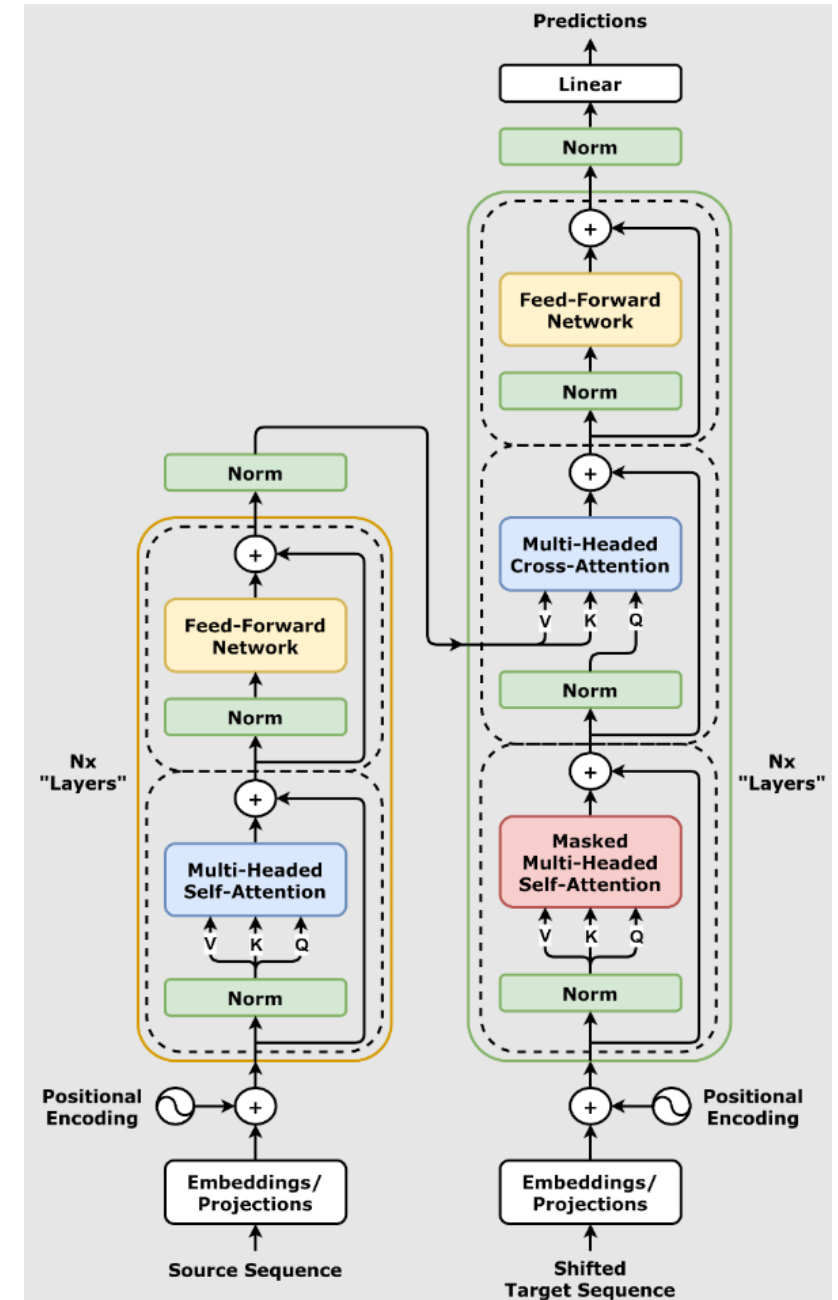
Transformer – Foundation of LLM



- A transformer is a deep learning architecture developed by researchers at Google and based on the multi-head attention mechanism. Text is converted to numerical representations called **tokens**, and each token is converted into a **vector** via lookup from a word **embedding** table.
- At each layer, each token is then contextualized within the scope of the context window with other tokens via a parallel multi-head attention mechanism, allowing the signal for key tokens to be amplified and less important tokens to be diminished.
- Transformers have the advantage of having no recurrent units, therefore requiring less training time than earlier recurrent neural architectures (RNNs) such as long short-term memory (LSTM). Later variations have been widely adopted for training large language models (LLM) on large (language) datasets
- Transformers were first developed as an improvement over previous architectures for machine translation, but have found many applications in large-scale natural language processing, computer vision (vision transformers), reinforcement learning, audio, etc. It has also led to the development of pre-trained systems, such as generative pre-trained transformers (GPTs) and BERT (bidirectional encoder representations from transformers).

Transformer Architecture

- All transformers have the same primary components:
 - Tokenizers, which convert text into tokens.
 - Embedding layer, which converts tokens and positions of the tokens into vector representations.
 - Transformer layers, which carry out repeated transformations on the vector representations, extracting more and more linguistic information.
 - These consist of alternating attention and feedforward layers. There are two major types of transformer layers: encoder layers and decoder layers, with further variants.
 - Un-embedding layer, which converts the final vector representations back to a probability distribution over the tokens.





ericsson.com/training