Deep Learning

CSIP5403 – Research Methods and AI Applications

Dr. Nathanael L. Baisa

Lecture Content

- Introduction
- Perceptron
- Multi-Layer Perceptron (MLP)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs) and Its Variants
- Transformer
- Conclusion
- References

Session Outcomes

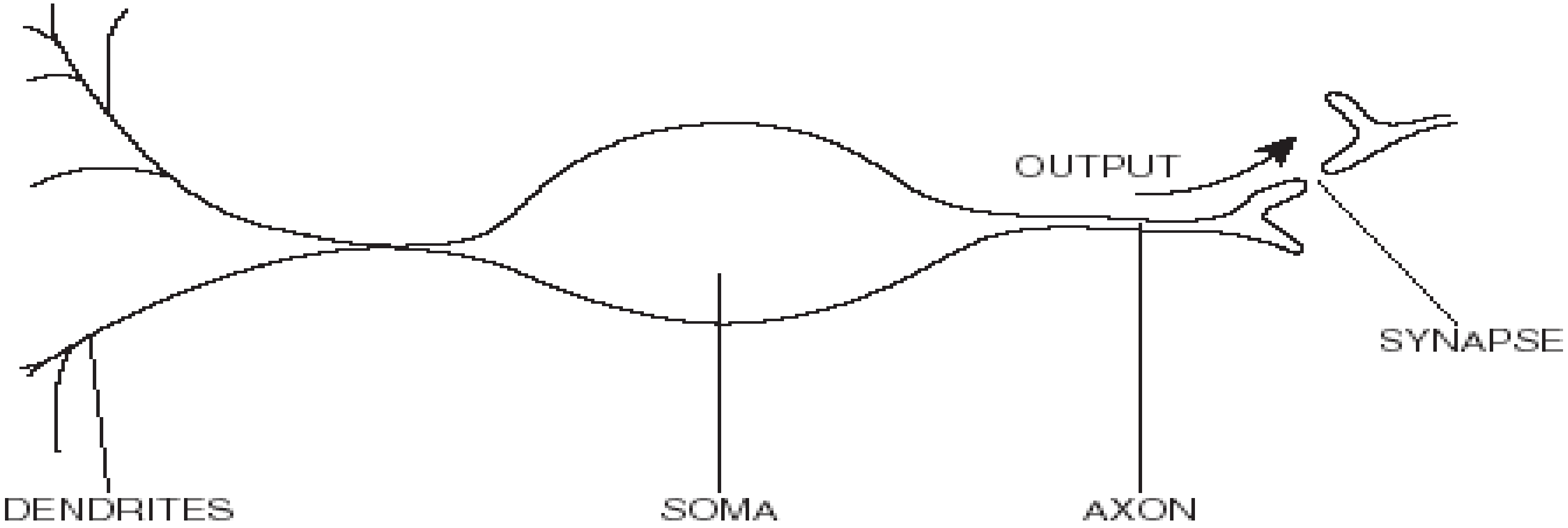
- Understand what artificial neural networks are in details.
- Acquire knowledge of perceptron and multilayer perceptron.
- Gain knowledge of deep learning.
- Understand how to apply neural networks including deep learning to different applications.

Introduction

- **Artificial neural networks (ANNs),** usually simply called neural networks (NNs), are computing systems inspired by the biological neural networks that constitute human brains.
- **ANNs** provide a practical method for ***learning*** utilizing:
  - real-valued functions
  - discrete-valued functions
  - vector-valued functions
- ANNs are **robust to errors** in training data (noisy, complex data). They can ***generalize*** on the learned datasets.

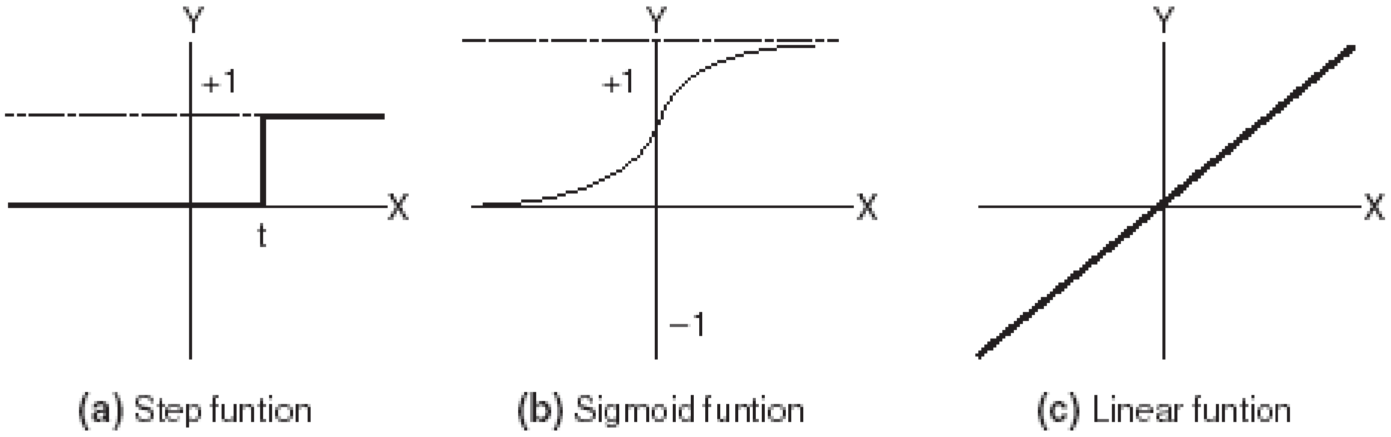
Introduction

- **The Human Brain** is made up of **billions of simple processing units,** called ***neurons*** **i.e. biological neurons.**
- **Inputs** are received on ***dendrites***, and if the **input levels** are **over** a ***threshold***, the ***neuron fires***, passing a **signal** through the ***axon*** to the ***synapse*** which then connects to another neuron.



Introduction

- Artificial neurons are based on **biological neurons.**
- **Each neuron** in the network receives **one or more inputs.**
- An **activation function** is applied to the inputs which determines the **output of the neuron** – ***the activation level***.
- The plots below show three typical activation functions.



(a) Step funtion (b) Sigmoid funtion (c) Linear funtion

Introduction

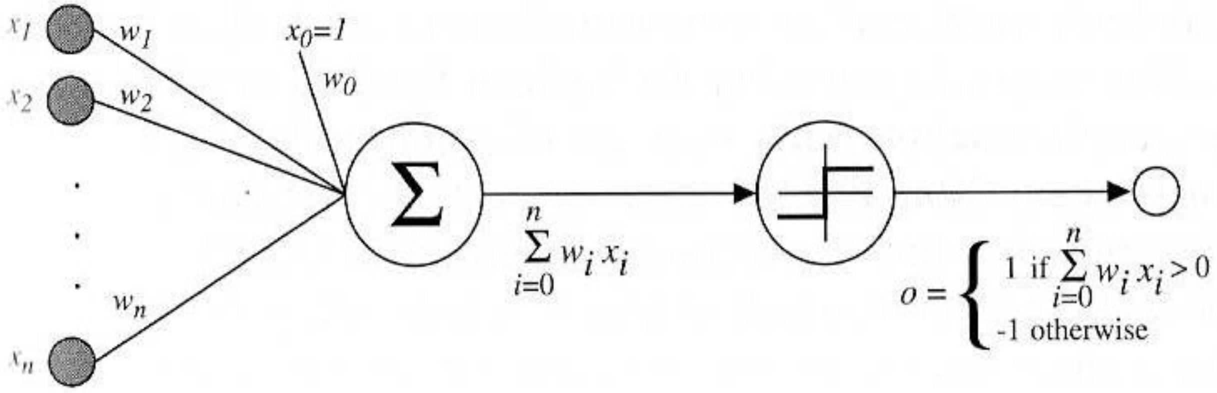
- A **typical activation function** works as follows:

$$X = \sum_{i=1}^{n} w_i x_i \qquad Y = \begin{cases} +1 & for\ X > t \\ 0 & for\ X \le t \end{cases}$$

- Each node *i* has a weight w_i associated with it.
  - The input to node *i* is x_i
  - *t* is the threshold.
  - So, if the weighted sum of the inputs to the neuron is above the threshold, then the **neuron** fires.

Perceptron: Model

- A **perceptron** is a single neuron that classifies a set of inputs into one of two categories (usually **1 or -1**).
- If the inputs are in the form of a matrix, a perceptron can be used to recognize visual images of shapes.
- The perceptron usually uses a ***step function***, which returns **1** if the weighted sum of inputs exceeds a threshold, and **-1** otherwise.


x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Perceptron: Training

- ***Learning*** involves choosing values for the weights**.**
- **Perceptron** is trained as follows:
  - First, inputs are given random weights (usually between –0.5 and 0.5).
  - An **item** of training data is presented. If the perceptron **miss-classifies** it, the **weights are modified** according to the following:

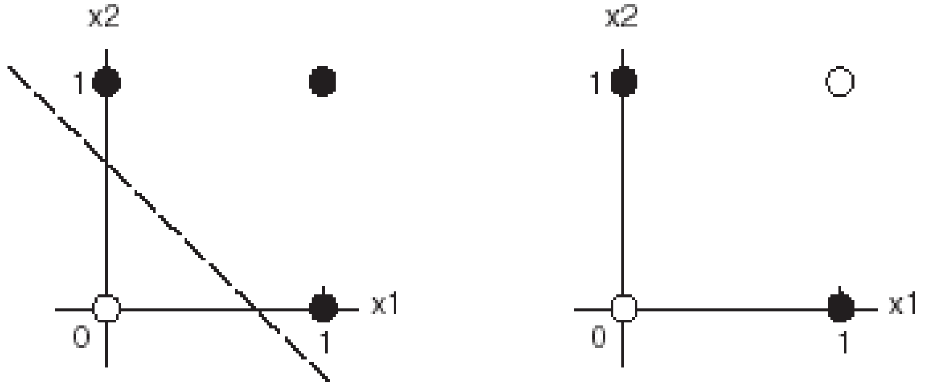
$$w_i \leftarrow w_i + \left(a \times x_i \times \left(t - o\right)\right)$$

where ***t*** is the **target output** for the **training example**, ***o*** is the **output** generated by the perceptron and ***a*** is the **learning rate**, between 0 and 1 (usually small value, e.g. 0.1)

- ***Cycle*** through **training examples** until successfully classify all examples (iterate for each training example).
  - **Each cycle** is known as an ***epoch*.**

Perceptron: Bias

- Perceptron can only **classify linearly separable functions**.
- The first of the plots below shows a **<u>linearly</u> separable function (OR).**
- The second plot below is **<u>not</u> linearly separable** (**XOR**).



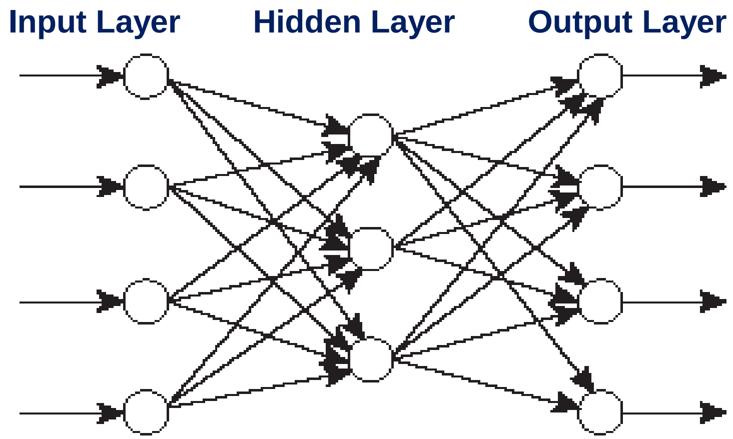
Perceptron: Convergence

- **Perceptron** ***training rule*** only ***converges*** when training examples are **linearly separable** and a has a small learning constant.

- Another approach uses the ***delta rule*** and **gradient descent:**
  - Same basic rule for finding update value.
  - **Changes** happening in the **weights**:
    - Do not incorporate the threshold in the output value (***unthresholded*** perceptron).
    - Wait to update weight until cycle is complete.
  - **Converges asymptotically** towards the **minimum error hypothesis**, possibly requiring unbounded (additional) time, but converges regardless of whether the training data are **linearly separable** or **not**.

Multilayer Perceptron (MLP)

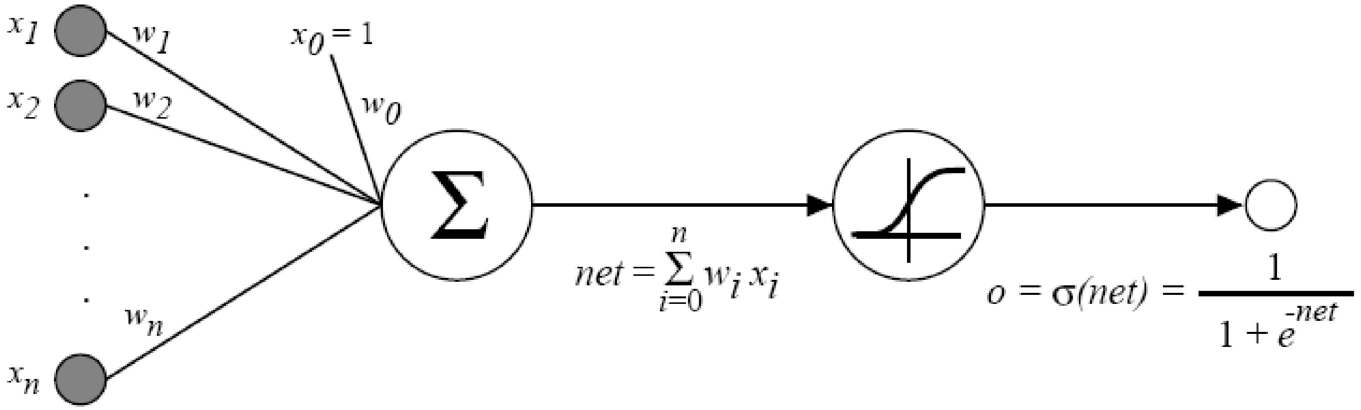
- **Multilayer perceptron (MLP),** also called feed-forward **multilayer neural network**, can **classify a range of functions**, including ***non-linearly separable ones.***
- Each input layer neuron connects to all neurons in the hidden layer.
- The neurons in the hidden layer connect to all neurons in the output layer. Hence, MLP is made up of several fully connected layers.



A feed-forward network

Multilayer Perceptron (MLP): Sigmoid Unit

- Sigmoid unit:



$net = \sum_{i=0}^{n} w_i x_i$

$o = \sigma(net) = \dfrac{1}{1 + e^{-net}}$

$\sigma(x)$ is **the sigmoid function** $\dfrac{1}{1+e^{-x}}$

It is **differentiable**: $\dfrac{\partial\sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$

Derive gradient descent rules to train:

- One sigmoid unit – ***node***.
- Multilayer networks of ***sigmoid units***.

Multilayer Perceptron (MLP): Backpropagation Algorithm

- The **training algorithm** of the multilayer neural networks is also called, *feedforward backpropagation learning***.**
- Multilayer neural networks learn in the same way as perceptrons**.**
- However, there are many more weights, and it is important to adjust weights by increasing their values when converging towards correct output or penalize the multilayer neural network when converging away from the correct output by decreasing the values of the weights.
- *E* sums the **errors** over **all** of the network output units:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

Multilayer Perceptron (MLP): Backpropagation Algorithm

- ***Create*** a **feedforward network** with ***n_{in}*** **inputs, *n_{hidden}* hidden units, and *n_{out}* output units**.
- ***Initialize*** all network **weights** to small random numbers.
- **Until termination condition is met, Do**
  - For **each <*x*, *t*> in training examples, Do**

    Propagate *the input* ***forward*** *through the network:*

    1. **Input** the instance ***x*** to the network and **compute** the **output *o_u* of every unit *u*** in the network

    Propagate *the* ***errors backward*** *through the network:*

    2. For **each** network **output unit *k*, calculate** its **error term *δ_k***
    3. For **each hidden unit *h*,** calculate its **error term *δ_h***
    4. **Update each** network **weight *w_{ji}***

    where

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k$$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = \alpha\delta_j x_{ji}$$

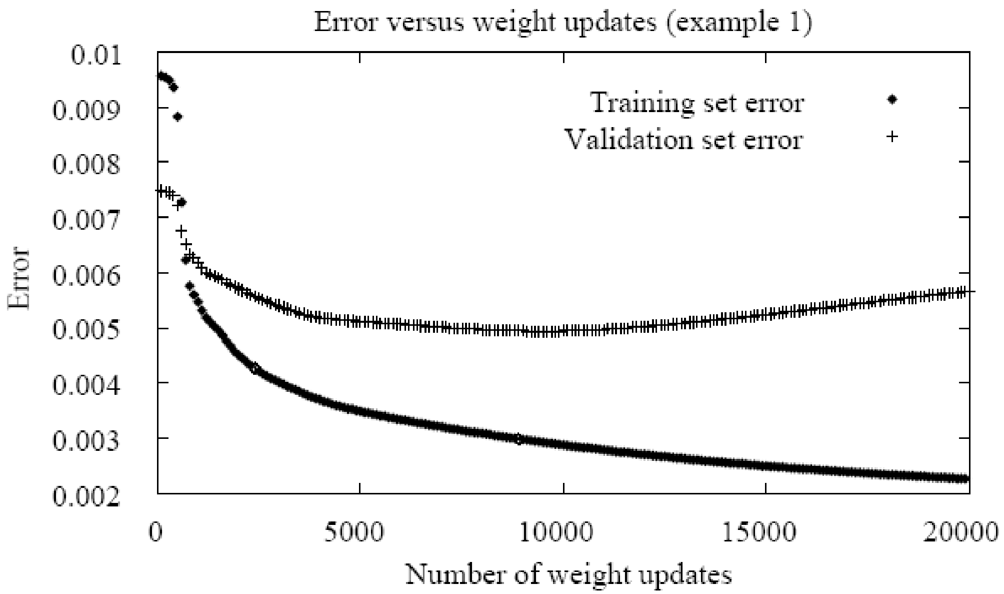
Multilayer Perceptron (MLP): Momentum

- **Momentum**: One of many different **training algorithms** for improving convergence of the multilayer neural network.
- Modify the **update rule** by making the weight update on the nth iteration depend partially on the update that occurred in the (n-1)th iteration.
- Minimizes error over training examples.
- Speeds up training.

$$\Delta w_{ji}(n) = \alpha \delta_j x_{ji} + \beta \Delta w_{ji}(n-1)$$

Multilayer Perceptron (MLP): When to Stop Training

- Continue until **error falls** below some **predefined threshold:**
  - Note that this can lead to ***overfitting*** i.e. the multilayer neural network will not be able to **generalize** well over unseen data.
  - To avoid overfitting, use Error estimate calculation.



Multilayer Perceptron (MLP): Overfitting – Cross Validation

- Common approach to **avoid overfitting**: Cross Validation
  - Reserve part of the training data for testing:
    - m examples are partitioned into k disjoint subsets.
  - Run the procedure k times.
  - Each time a different one of these subsets is used as validation.
  - Determine the number of iterations that yield the best performance by estimating the Error between the input and the output target.
  - Mean of the number of iterations is used to train all m examples.

Multilayer Perceptron (MLP): Video Tutorials

- Video Tutorial on how to create your own ANNs from scratch with Python: https://www.youtube.com/watch?v=Wo5dMEP_BbI

- Video Tutorial in depth about ANNs: https://www.youtube.com/watch?v=aircAruvnKk

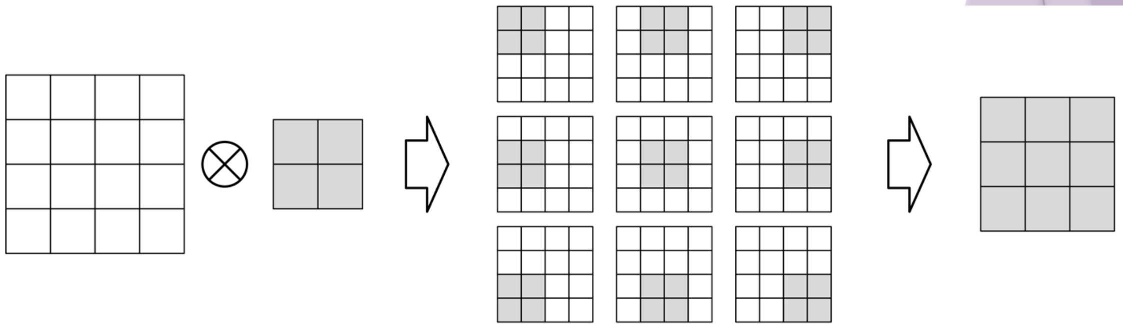
The second video tutorial about ANNs has a controversy at the end of the video talking about the ReLU vs the sigmoid unit. Actually, the closest to **nature** and to the biological neuron is the sigmoid function rather than the ReLU. However, the ReLU improves the convergence of ANNs during training!!

MLP vs Deep Learning

- ANNs are at the heart of deep learning algorithms.
- Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers.
- Different deep learning types exist:
  - Convolutional Neural Networks (CNNs)
  - Recurrent Neural Networks (RNNs) and its variants such as LSTM and GRU.
  - Transformer (based on self-attention mechanisms)

Convolutional Neural Networks (CNNs)

- Well suited ANN to detect spatial substructures.
- CNN Hyperparameters which control behaviour of CNN.
  - Controlling shape:
    - Kernel size
    - Stride -> Step size between Convolutions.
    - Padding -> Length expansion of input data tensor.
    - Dilation -> Controls how the convolutional kernel is applied to input matrix.
    - Channels -> input and output channels.



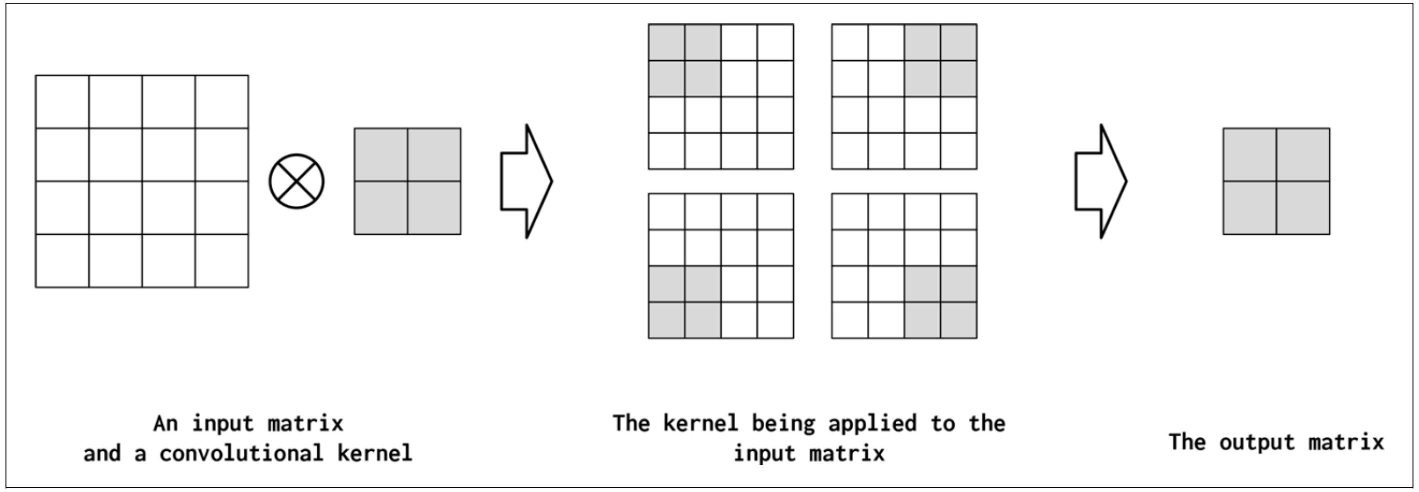
An input matrix
and a convolutional kernel

The kernel being applied to the
input matrix

The output matrix
(feature map)

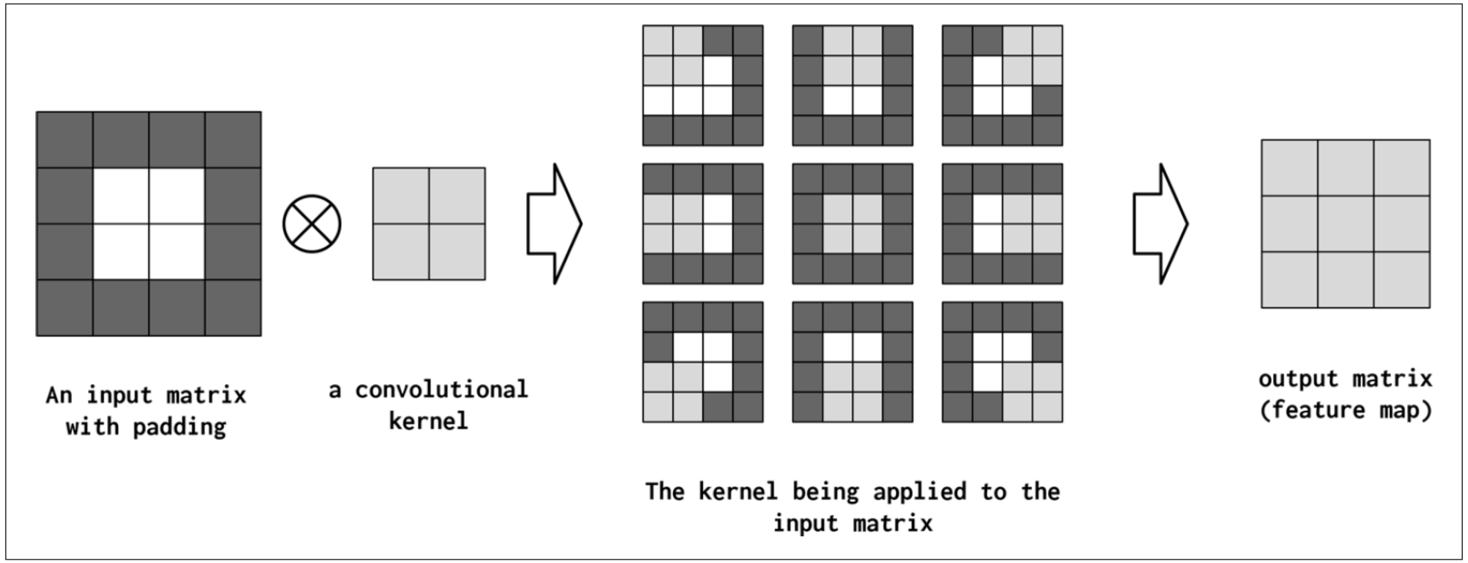
Convolutional Neural Networks (CNNs)

- **Stride:**
  - A convolutional kernel with kernel_size=2 applied to an input (multiply and add values) with the hyperparameter stride equal to 2. This has the effect that the kernel takes larger steps, resulting in a smaller output matrix. This is useful for subsampling the input matrix more sparsely.



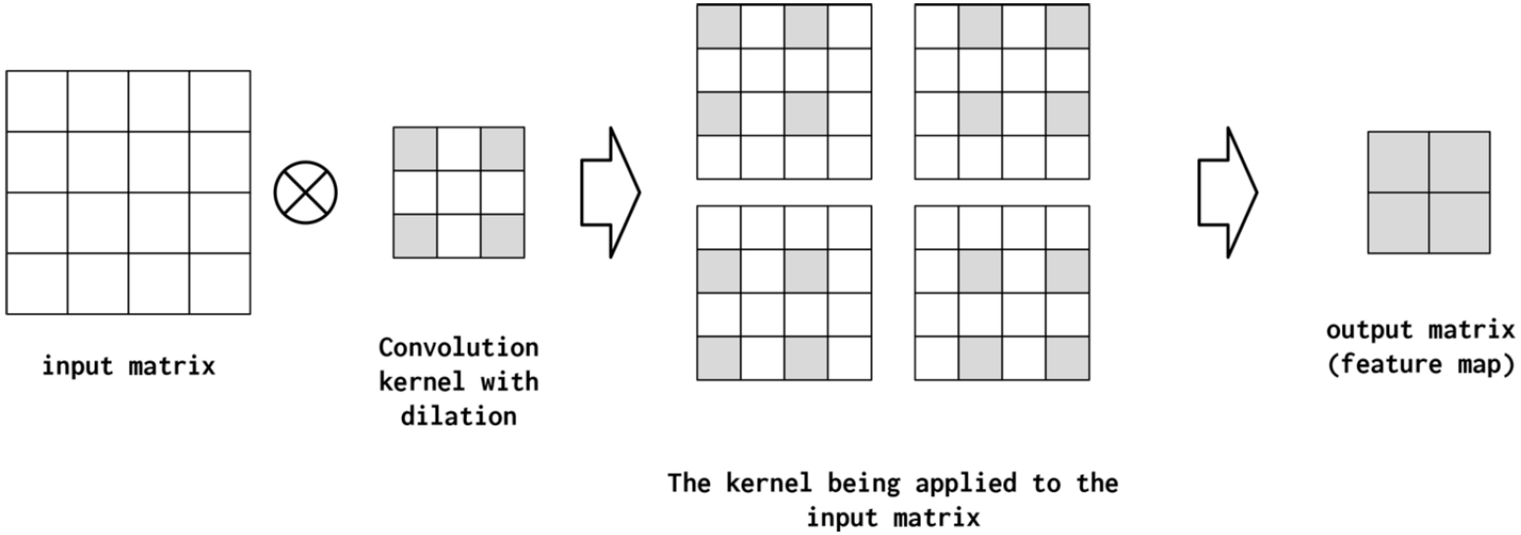
Convolutional Neural Networks (CNNs)

- **Padding:**
  - A convolution with kernel_size=2 applied to an input matrix that has height and width equal to 2. However, because of padding (indicated as dark-gray squares), the input matrix's height and width can be made larger. This is most commonly used with a kernel of size 3 so that the output matrix will exactly equal the size of the input matrix.


An input matrix with padding
a convolutional kernel
The kernel being applied to the input matrix
output matrix (feature map)

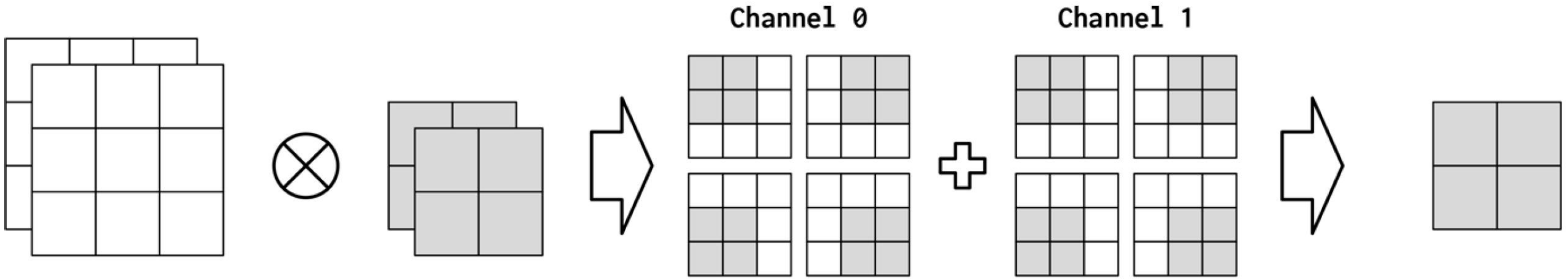
Convolutional Neural Networks (CNNs)

- **Dilation:**
  - A convolution with kernel_size=2 applied to an input matrix with the hyperparameter dilation=2. The increase in dilation from its default value means the elements of the kernel matrix are spread further apart as they multiply the input matrix. Increasing dilation further would accentuate this spread.


input matrix
Convolution kernel with dilation
output matrix (feature map)
The kernel being applied to the input matrix

Convolutional Neural Networks (CNNs)

- **Input channels:**
  - A convolution operation is shown with two input matrices (two input channels). The corresponding kernel also has two layers; it multiplies each layer separately and then sums the results.
  - Configuration: input_channels=2, output_channels=1, kernel_size=2, stride=1, padding=0, and dilation=1.



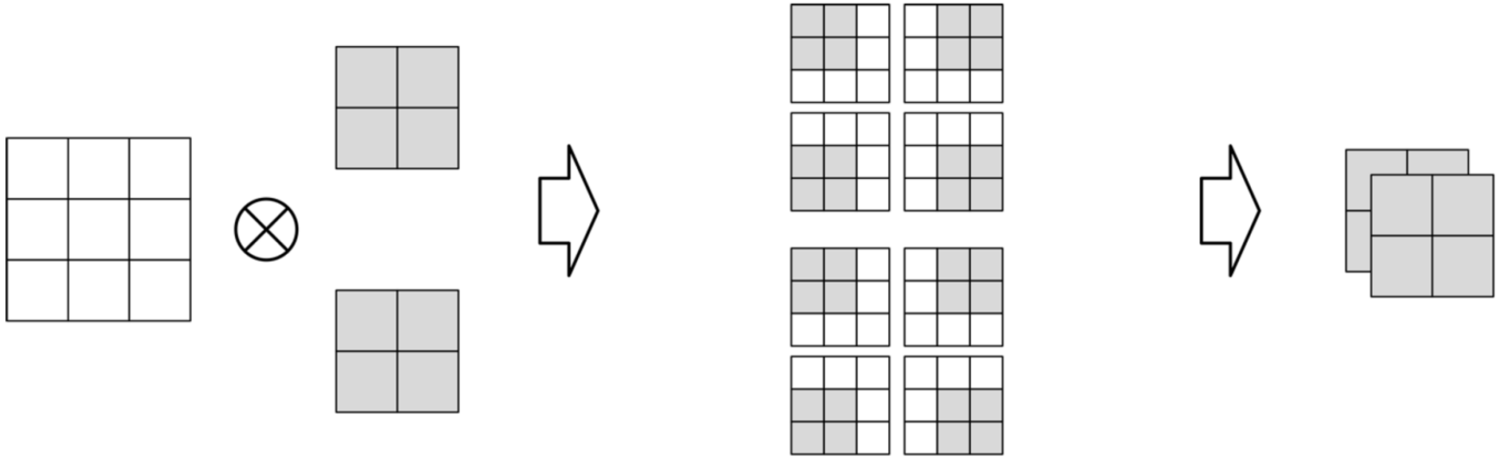
Input tensor with 2 channels and a convolutional kernel

The kernel being applied to the input tensor

The output matrix

Convolutional Neural Networks (CNNs)

- **Output channels:**
  - A convolution operation with one input matrix (one input channel) and two convolutional kernels (two output channels). The kernels apply individually to the input matrix and are stacked in the output tensor.
  - Configuration: input_channels=1, output_channels=2, kernel_size=2, stride=1, padding=0, and dilation=1.



Input tensor with 1 channel and two convolutional kernels

The kernels being applied to the input tensor

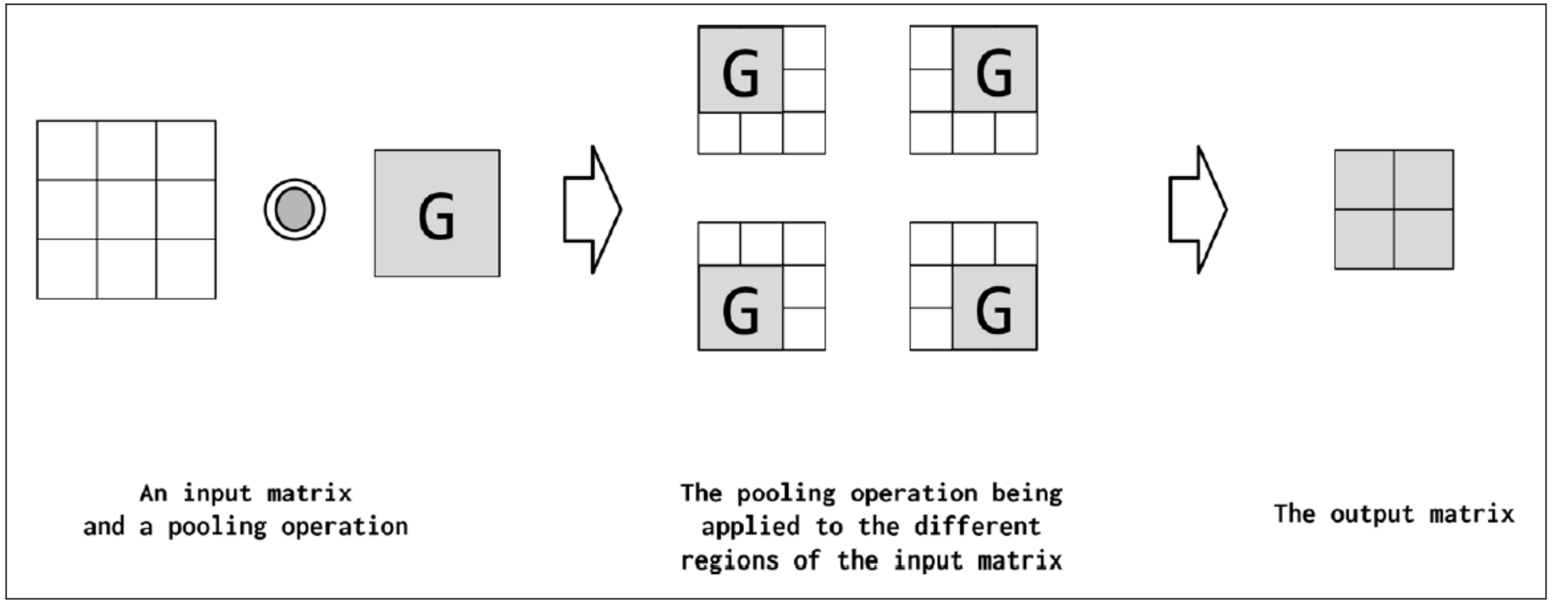
The output tensor

Convolutional Neural Networks (CNNs)

- For **CNN design**, you may consider the following:
  - Pooling: Operation used to summarize a higher dimensional feature map to a lower dimensional feature map.
  - Batch Normalization: Applies a transformation to the output of a CNN layer by scaling the activations (feature maps) to have zero mean and unit variance.
  - Residual connections: Method for adding the original matrix to the output of a convolution.

Convolutional Neural Networks (CNNs)

- **Pooling:**
  - The pooling operation as shown here is functionally identical to a convolution: it is applied to different positions in the input matrix. However, rather than multiply and sum the values of the input matrix, the pooling operation applies some function G that pools the values. G can be any operation, but summing, finding the max, and computing the average are the most common.



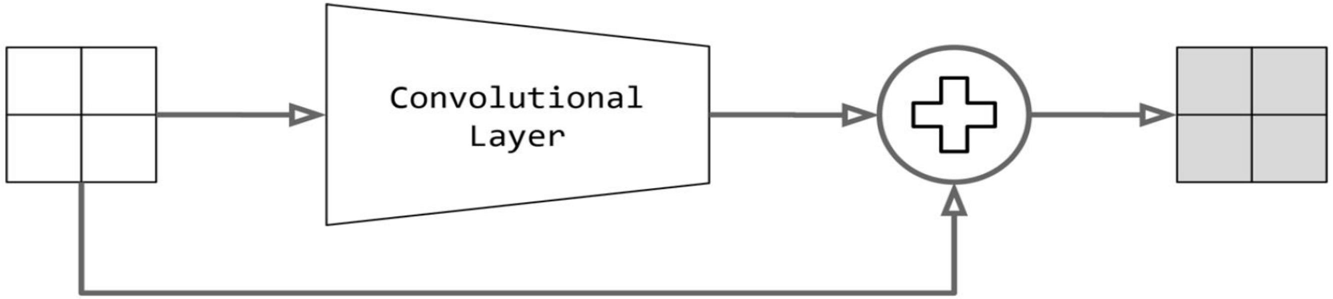
An input matrix and a pooling operation

The pooling operation being applied to the different regions of the input matrix

The output matrix

Convolutional Neural Networks (CNNs)

- **Residual Connection:**
  - A residual connection is a method for adding the original matrix to the output of a convolution. This is described visually below as the convolutional layer is applied to the input matrix and the result is added to the input matrix. A common hyperparameter setting to create outputs that are the same size as the inputs is let kernel_size=3 and padding=1.



A convolutional layer is applied to the input matrix to create a matrix of the same size.

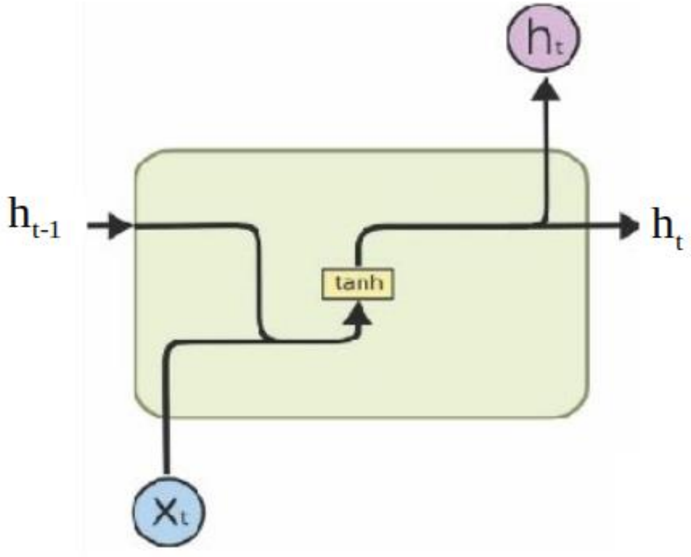
The input matrix is added to the output of the convolutional layer.

Recurrent Neural Networks (RNNs)

- In contrast to the uni-directional **feedforward neural network**, it is a bi-directional artificial neural network, meaning that it allows the output from some nodes to affect subsequent input to the same nodes.
- Their ability to use internal state (memory) to process arbitrary sequences of inputs makes them applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition, machine translation, time series prediction, etc.
- There are different recurrent neural network architectures such as vanilla RNN, Long short-term memory (LSTM) and Gated recurrent unit (GRU).

Recurrent Neural Networks (RNNs): Vanilla RNN

- **RNN cell:**
  - Difficult to train due to gradient vanishing problem.
  - Can’t learn long-range dependencies (connections).

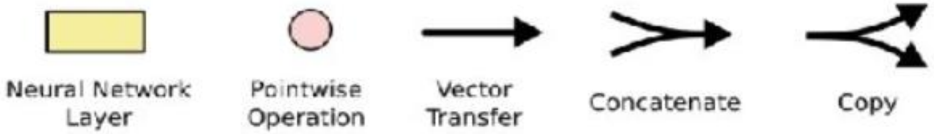


The architecture of RNN cell

Hidden state update: $h_t = \textit{tanh}(W^{(ih)}x_t + W^{(hh)}h_{t-1} + b_h)$

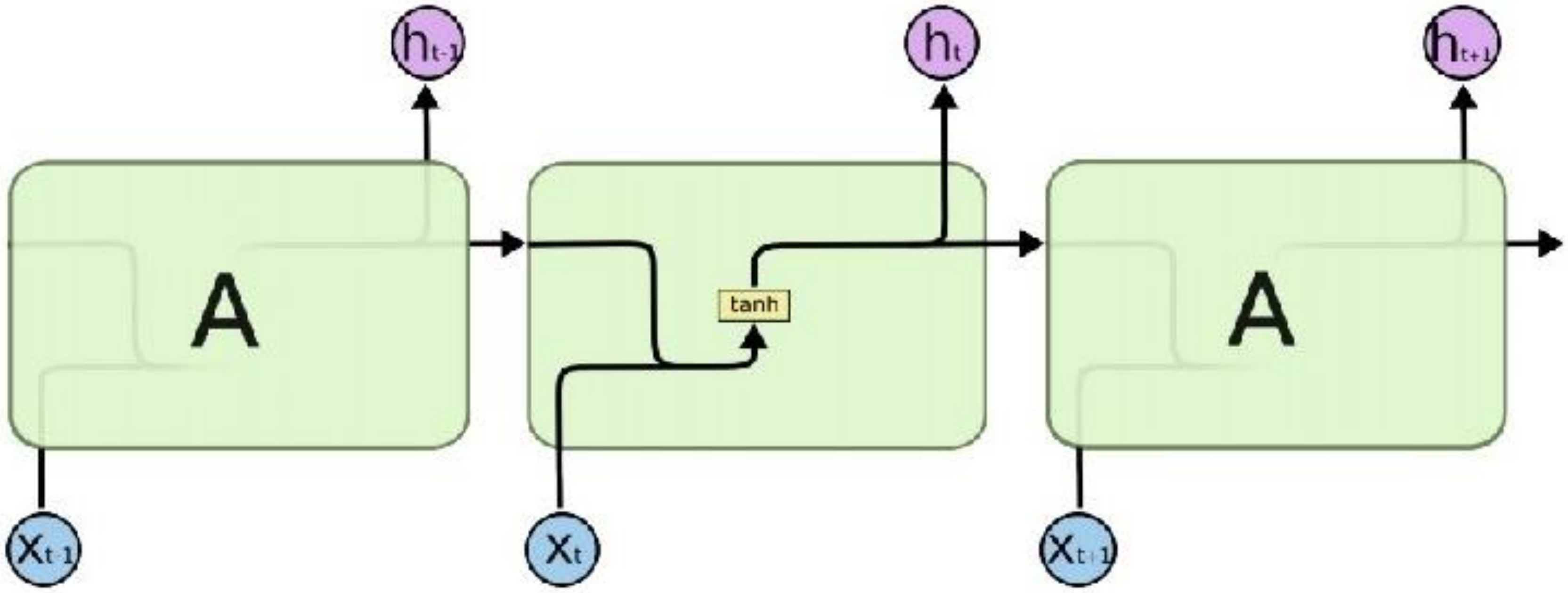
Output: $y_t = \sigma_y(W^{(hy)}h_t + b_y)$

Where σ_y is activation function e.g. softmax.



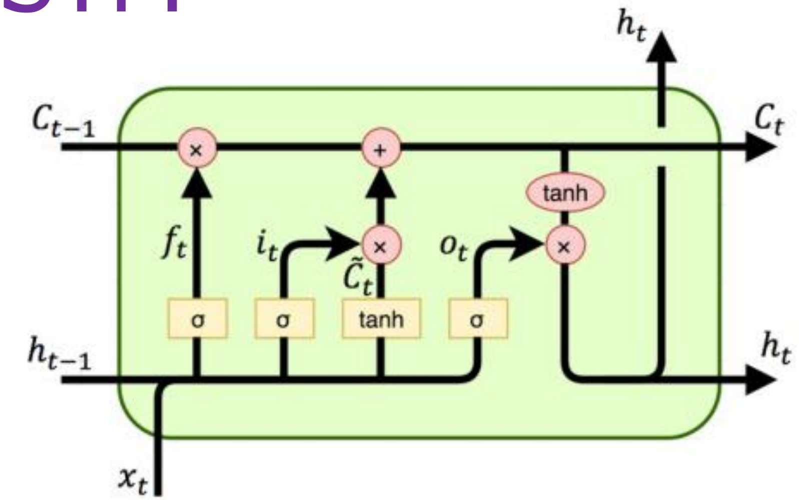
Recurrent Neural Networks (RNNs):
Vanilla RNN

- **Multilayer standard (vanilla) RNN:**



Recurrent Neural Networks (RNNs): LSTM

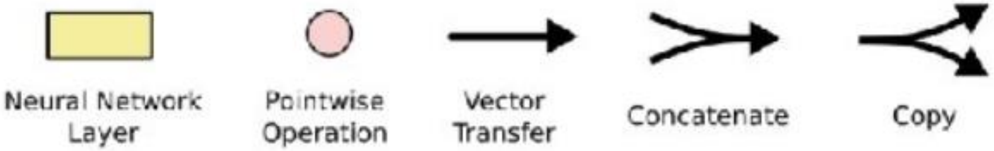
- **LSTM cell:**
  - Has memory cell C_t.
  - Has three gates:
    1. i_t – writes to memory.
    2. f_t – resets memory.
    3. o_t – reads from memory.



The architecture of LSTM cell

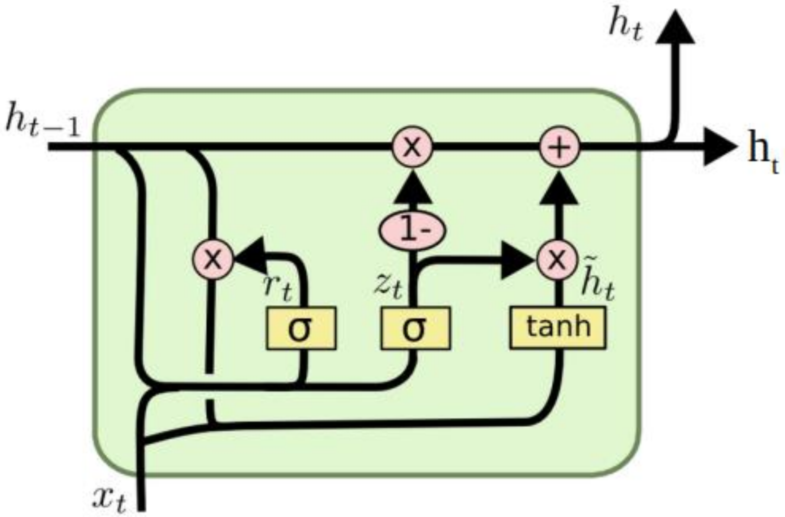
$$
\begin{aligned}
&\text{Input gate:} && i_t = \sigma\left(W^{(ii)}\bar{x}_t + W^{(hi)}h_{t-1}\right)\\
&\text{Forget gate:} && f_t = \sigma\left(W^{(if)}\bar{x}_t + W^{(hf)}h_{t-1}\right)\\
&\text{Output gate:} && o_t = \sigma\left(W^{(io)}\bar{x}_t + W^{(ho)}h_{t-1}\right)\\
&\text{Process input:} && \tilde{C}_t = tanh\left(W^{(i\tilde{c})}\bar{x}_t + W^{(h\tilde{c})}h_{t-1}\right)\\
&\text{Cell update:} && C_t = f_t * C_{t-1} + i_t * \tilde{C}_t\\
&\text{Output:} && y_t = h_t = o_t * \tanh\left(C_t\right)
\end{aligned}
$$

In fact, $y_t = \text{softmax}(W^{(hy)}h_t + b_y)$; σ is a sigmoid function.



Recurrent Neural Networks (RNNs): GRU

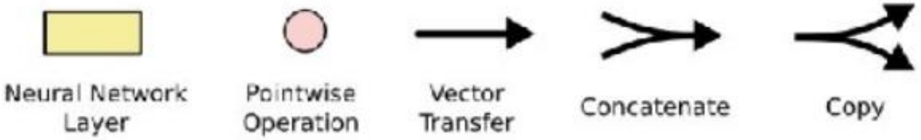
- **GRU cell:**
  - Has two gates:
    1. r_t – resets hidden state.
    2. z_t – updates hidden state.



The architecture of GRU cell

Reset gate:	$r_t = \sigma\,(W^{(ir)}\bar{x}_t + W^{(hr)}h_{t-1})$
Update gate:	$z_t = \sigma\,(W^{(iz)}\bar{x}_t + W^{(hz)}h_{t-1})$
Process input:	$\tilde{h}_t = tanh\,(W^{(i\tilde{h})}\bar{x}_t + W^{(h\tilde{h})}h_{t-1})$ $\longrightarrow + r_t * W^{(h\tilde{h})}h_{t-1}$ (Correction!)
Hidden state update:	$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$
Output:	$y_t = h_t$

In fact, $y_t = \text{softmax}(W^{(hy)}h_t + b_y)$; σ is a sigmoid function.



Transformer

- A **transformer** is a deep learning architecture based on the multi-head attention mechanism[1].
- It has no recurrent units, and thus requires less training time than previous recurrent neural architectures, such as LSTM, and its later variation has been prevalently adopted for training large language models (LLM) on large (language) datasets.
- This architecture is now used not only in NLP and computer vision[2], but also in audio and multi-modal processing. It has also led to the development of pre-trained systems, such as generative pre-trained transformers (GPTs) and Bidirectional Encoder Representations from Transformers (BERT).

- [1]. Vanilla transformer: Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, ; Jones, ; Gomez, A.; Kaiser, Ł.; Polosukhin, I. (2017). "Attention is All you Need", NIPS.
- [2]. Vision transformer: Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". arXiv:2010.11929 [cs.CV].

Transformer

- **Vanilla transformer:**

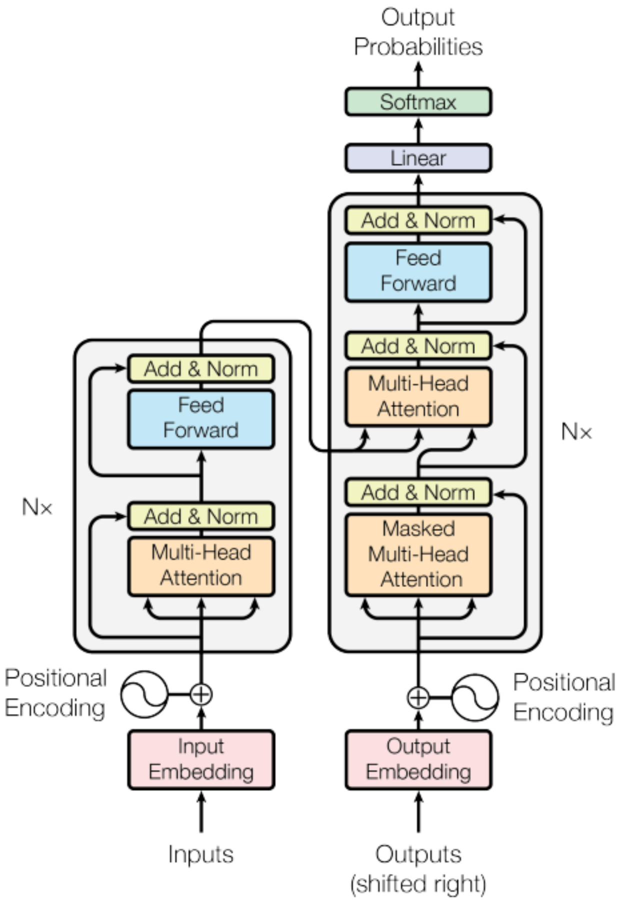


Figure 1: The Transformer - model architecture.

- Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, ; Jones, ; Gomez, A.; Kaiser, Ł.; Polosukhin, I. (2017). "Attention is All you Need", NIPS.

Transformer

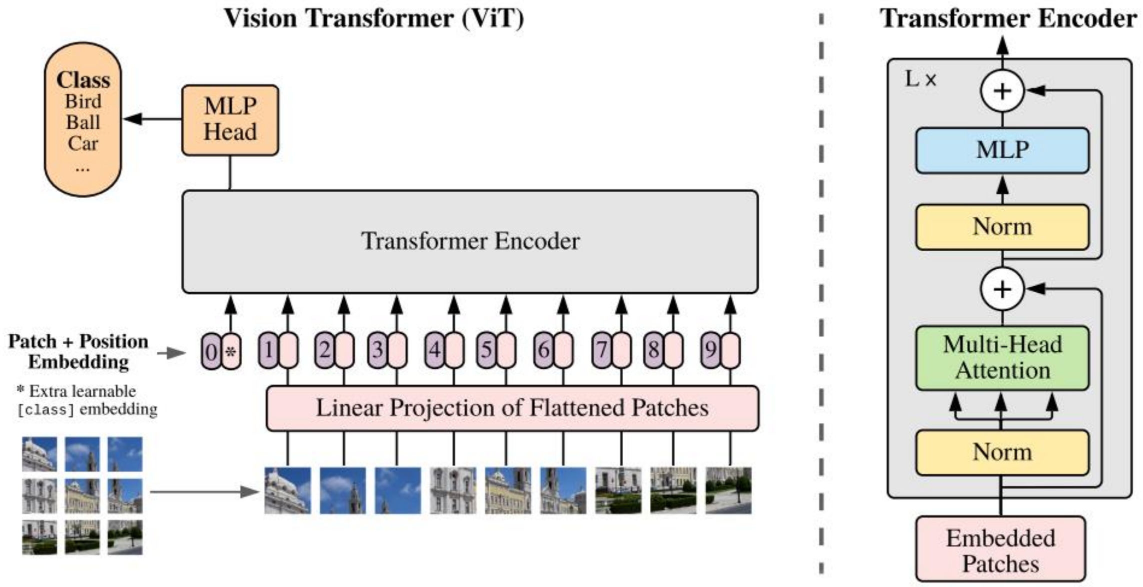
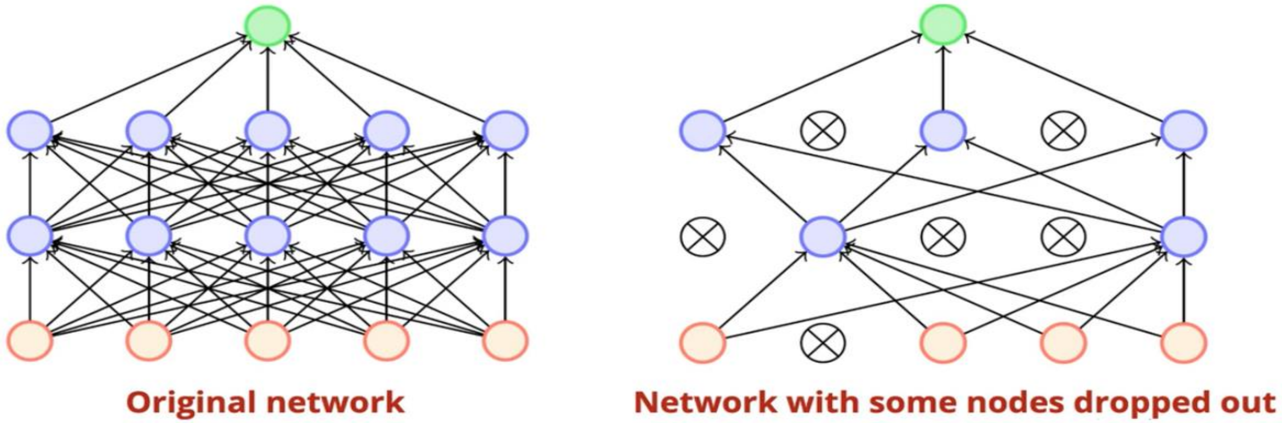


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

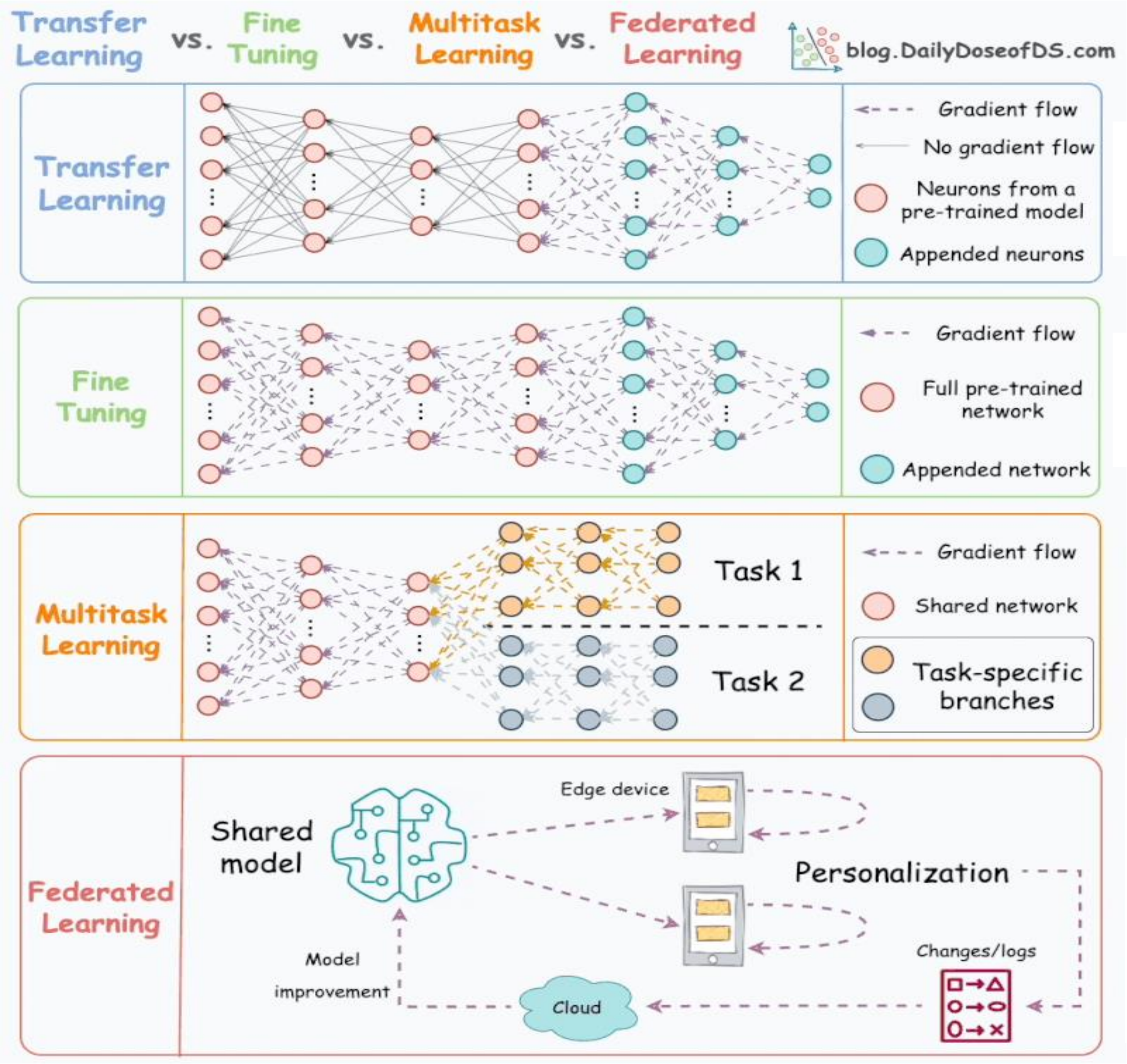
- **Vision transformer:** Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". arXiv:2010.11929 [cs.CV].

Tips on Deep Learning

- **Handling overfitting:**
  - Using regularization (e.g. L1 regularization, L2 regularization) which penalizes weights with large magnitudes. Recently introduced regularization methods in deep learning include dropout, batch normalization, etc.
- **Dropout:**
  - Randomly deactivates a few neurons in the network at each training step.
  - Helps reduce overfitting for better generalisation.



Tips on Deep Learning



Transfer learning involves taking a pre-trained model, typically trained on a large dataset for a related task, and transferring its knowledge to a new, similar task.

Note: Fine-tuning is a type of transfer learning.

Federated learning is a decentralized approach to machine learning. Instead of sending data to a central server, models are sent to devices, trained locally, and only model updates are gathered and sent back to the server.

- https://www.blog.dailydoseofds.com/p/transfer-learning-vs-fine-tuning

Tips on Deep Learning

- **Explore more about the following topics:**
  - Domain adaptation (a type of transfer learning)
    - https://medium.com/nerd-for-tech/domain-adaptation-problems-in-machine-learning-ddfdff1f227c
  - Self-supervised learning (a type of unsupervised learning)
    - https://medium.com/@zhonghong9998/exploring-self-supervised-learning-training-without-labeled-data-6e1a47dc5876
  - Meta learning (learning to learn)
    - https://medium.com/geekculture/an-introduction-to-meta-learning-b328ada3b27f
  - Model compression
    - https://xailient.com/blog/4-popular-model-compression-techniques-explained/

Conclusion

- There are **two** broad types of artificial neural network, characterized by direction of the flow of information between its layers: uni-directional feed-forward neural network (MLP) and bi-directional recurrent neural network.

- There are different deep learning types: CNNs, RNNs and its variants such as LSTM and GRU, and transformer. Regularization methods in deep learning to overcome overfitting problem include dropout, batch normalization, etc.

- Currently, transformer is very popular and is used in NLP, computer vision, audio and multi-modal processing. It is also the backbone of large language models (LLM) such as GPTs, BERT, etc.

References

- I. Goodfellow, Y. Bengio, A. Courville, ‘Deep Learning’, MIT Press, 2016. https://www.deeplearningbook.org/
- Understanding Deep Learning: https://udlbook.github.io/udlbook/
- Dive into Deep Learning: https://d2l.ai/
- The Little Book of Deep Learning: https://fleuret.org/francois/
- Deep Multi-Task and Meta Learning: https://cs330.stanford.edu/
- D. Foster and K. Friston, Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play, 2nd Edition, 2023.