Machine Learning: Deep Learning Architectures

CDM Computing Subgroup Mini-Workshop Series
20th of May, 2024

Matthew Green

ARC CENTRE OF EXCELLENCE FOR DARK MATTER PARTICLE PHYSICS

THE UNIVERSITY of ADELAIDE

Reminder: What did we learn from Lecture #1

- The basics of ML
    - Perceptron
    - Multi-Layered Perceptron (MLPs)
    - Activation Functions
    - Loss functions
    - Gradient Descent
    - Hyperparameters
    - Overfitting
    - Train/Validation/Test sets
    - Common Performance Metrics
- If you haven't watched the lecture, a recording is available on the gitlab!

Outline of this lecture

- Why different Architectures?
- Supervised Methods
  - Dense Neural Networks
  - Convolution Neural Networks
  - Graph Neural Networks
- Exercise on Particle Classification
- Dynamic Networks
  - Recurrent Neural Networks
  - Transformers

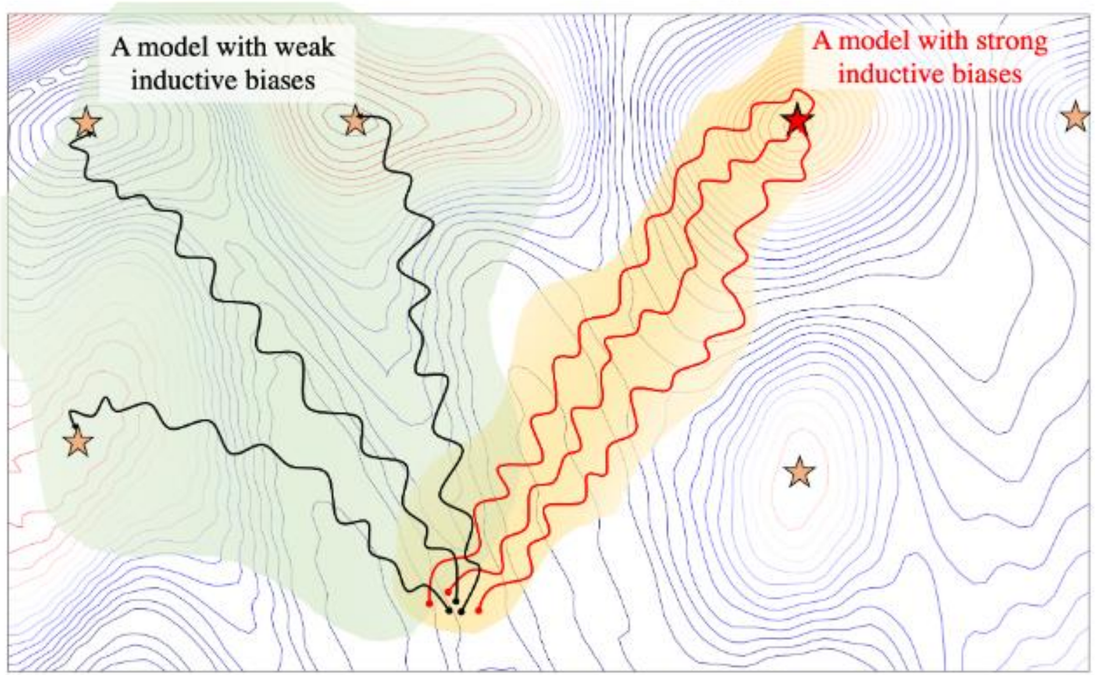
Why different architectures: Inductive Bias

- *Inductive bias* allows a learning algorithm to prioritize one solution (or interpretation) over another, independent of the observed data
- Ideally, inductive biases both improve the search for solutions without substantially diminishing performance, as well as help find solutions which generalize in a desirable way

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

https://arxiv.org/abs/1806.01261

Inductive Bias Example

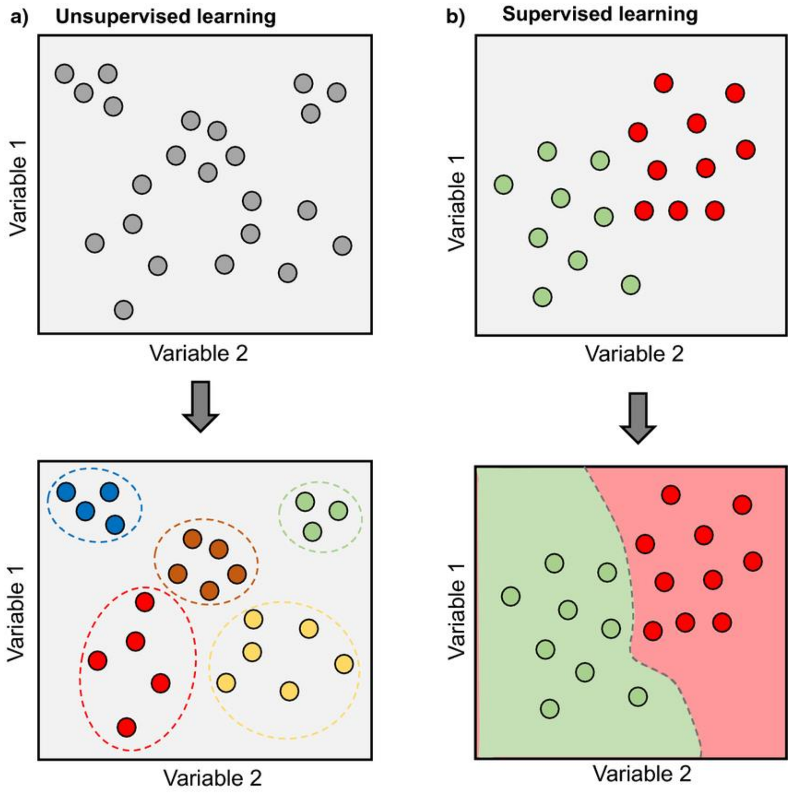


- A drawing of how inductive biases can affect models' preferences to converge to different local minima. The inductive biases are shown by coloured regions (green and yellow) which indicates regions that models prefer to explore

https://samiraabnar.github.io/articles/2020-05/indist

Supervised Methods

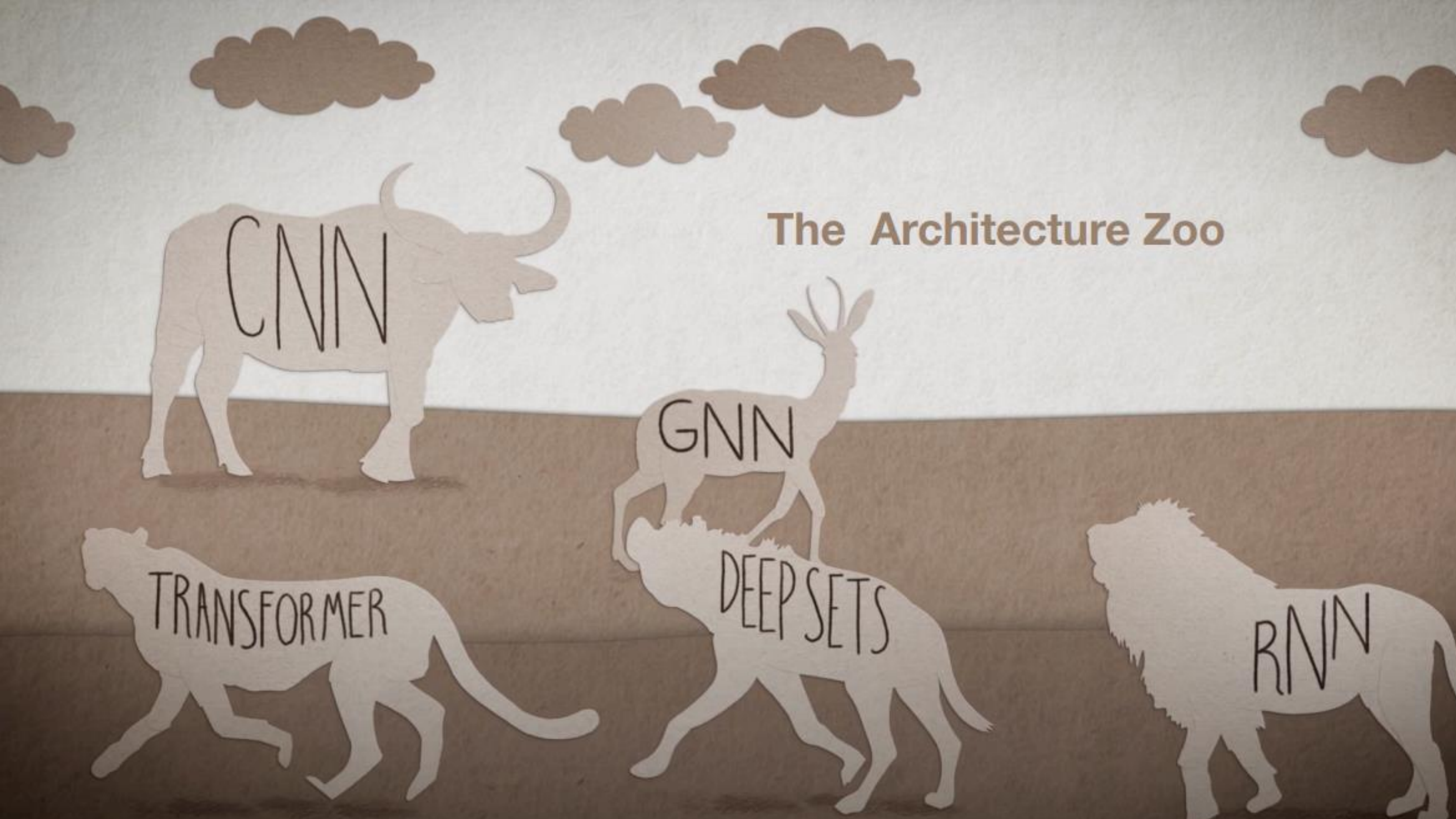
Reminder: Supervised vs Unsupervised



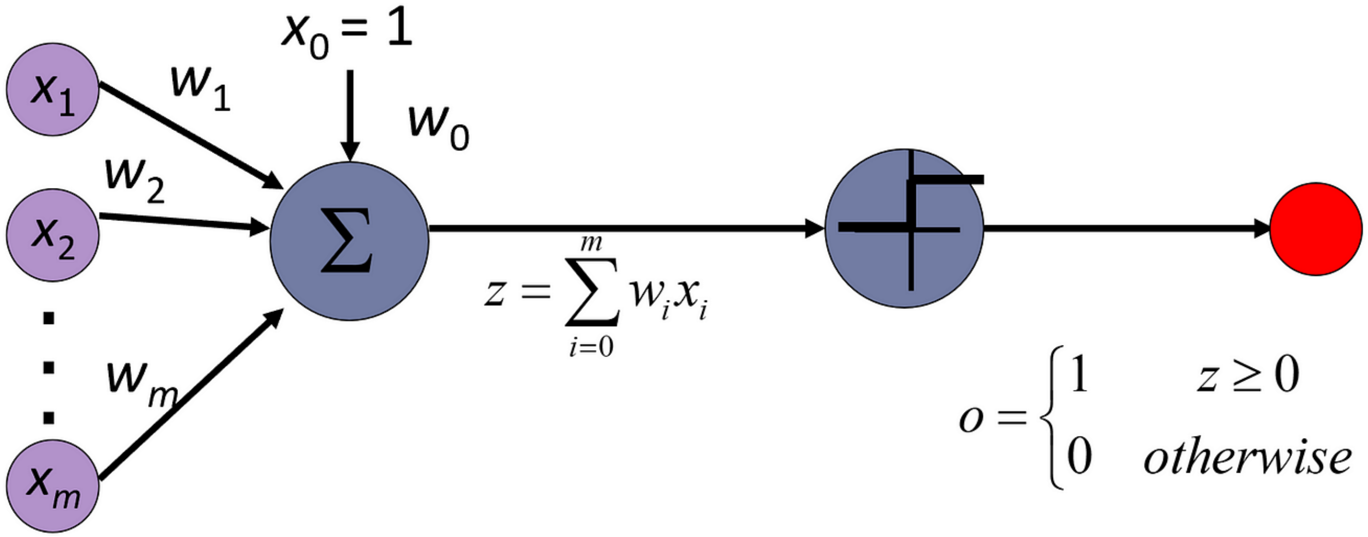
Supervised is 95%+ of ML

We know the correct answer, and we are training the model to be able to compute it

The basis of supervised models can be used for unsupervised models!


The Architecture Zoo
CNN
GNN
TRANSFORMER
DEEP SETS
RNN

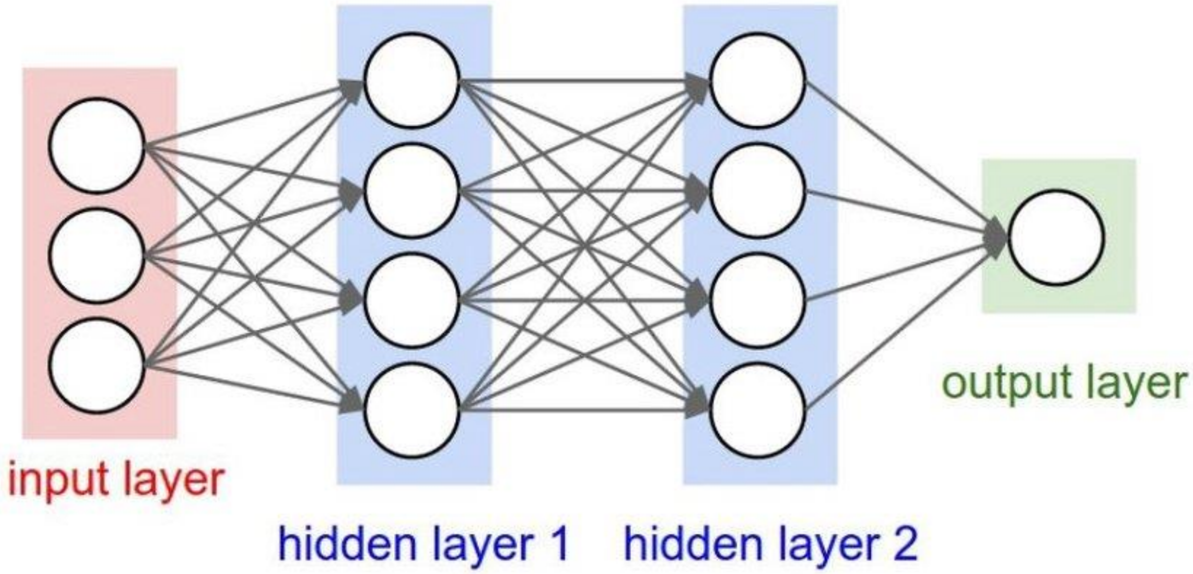
Perceptron



See the Basics of ML lecture from the kick-off week!

Multi-Layered Perceptron (Dense Neural Networks)

- Lots of perceptrons, each with independent weights and biases
- Introduces a "hidden" layer, which cannot be observed
    - Also called a "latent space"
- These are the classic "neural network"



Example: Python Code (pytorch)

```
# Define the neural network architecture
class DenseNN(nn.Module):
    def __init__(self,inputNum):
        super(DenseNN, self).__init__()
        self.inputNum=inputNum
        self.fc1 = nn.Linear(in_features=inputNum, out_features=64)  # Input layer
        self.fc2 = nn.Linear(in_features=64, out_features=64)     # Hidden layer
        self.fc3 = nn.Linear(in_features=64, out_features=64)     # Hidden layer
        self.fc4 = nn.Linear(in_features=64, out_features=4)     # Output layer

    def forward(self, x):
        x = x.view(-1, self.inputNum)   # Flatten the input
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

Problems of DNNs

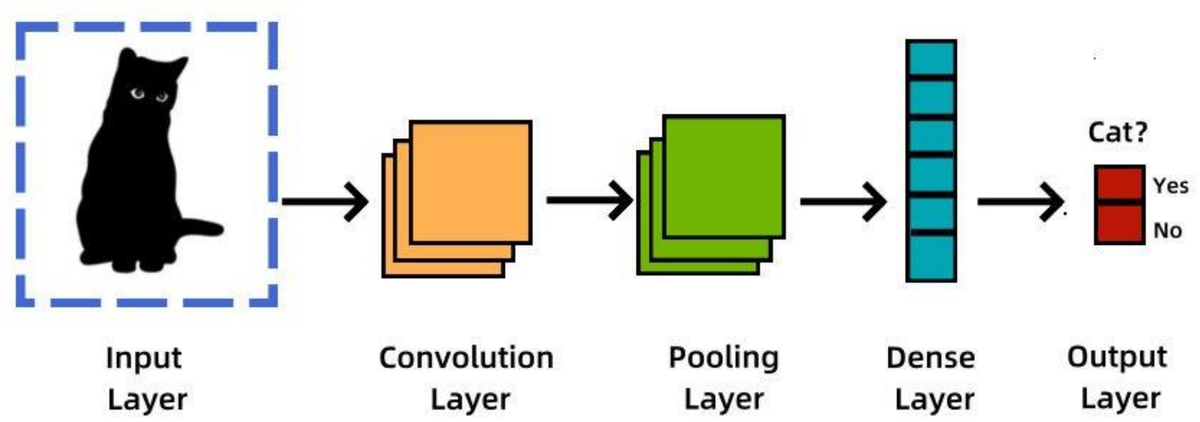
Does not encode locality
Does not encode translational invariance
Lots of parameters

Is there a cat in the picture?

How about now?

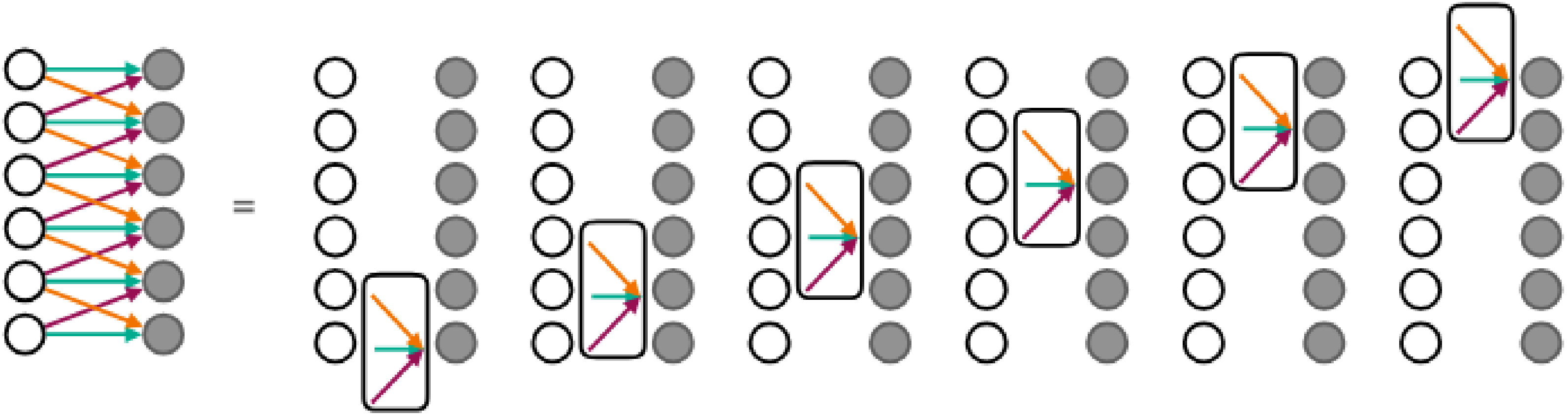
Convolution Neural Networks (CNNs)

- Sometimes we can be smarter on how we insert data into our network
  - For example, what if our data is an image?
- CNNs are a type of neural network architecture specifically designed for image recognition tasks in computer vision



Convolutions

- Element-wise multiplication and sum of the overlapping elements between the kernel and the input
- This is equivalent to a filter that slides across the input in 1-D



- What are we learning? We are learning the filter/kernel

Convolution in higher dimensions

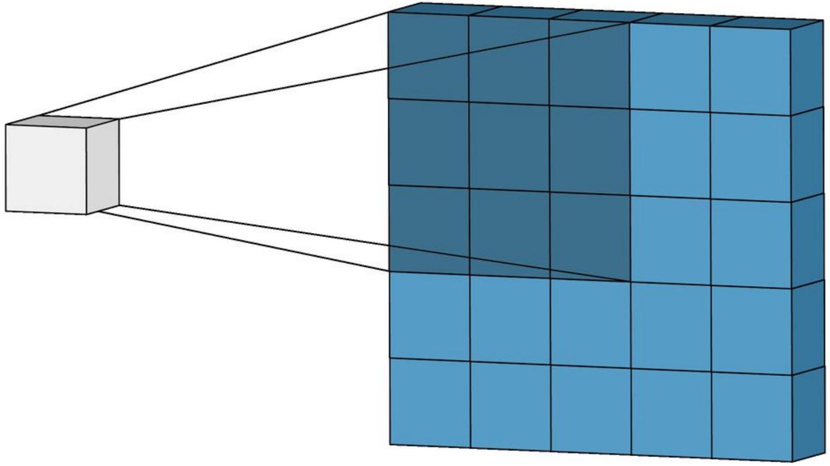
We can expand this to an arbitrary number of dimensions

Some hyperparameters:

Kernel Size: How large of an area considered in the convolution (3 in this case)

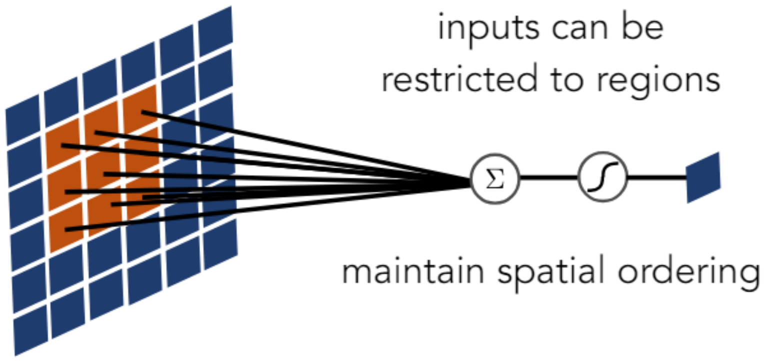
Stride: How many steps in a single interaction (1 in this case)

Both be tuned to improve computational speed vs performance



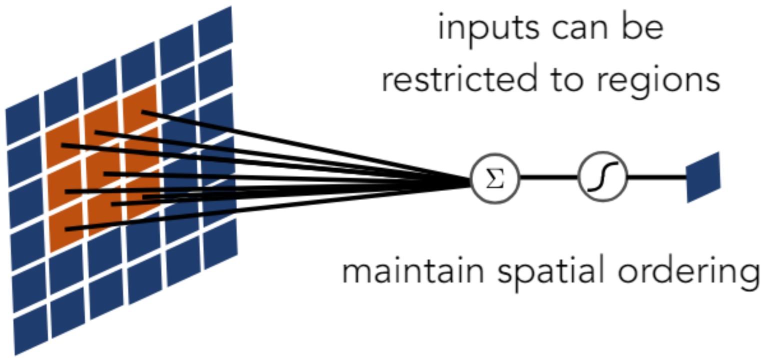
Properties of CNNs

- Locality: nearby areas tend to contain stronger patterns

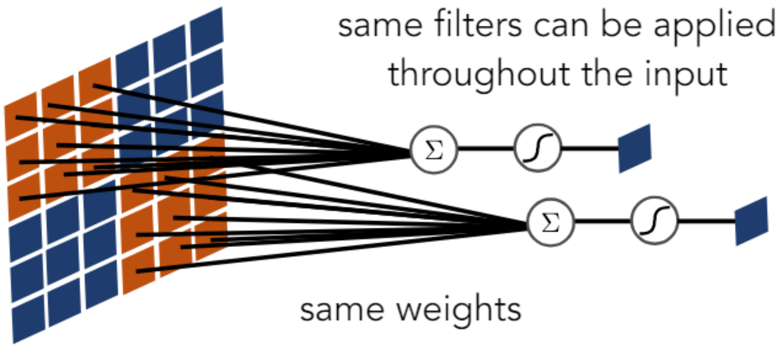


Properties of CNNs

- Locality: nearby areas tend to contain stronger patterns



- Translation Invariance: Relative positions are relevant.



Pooling

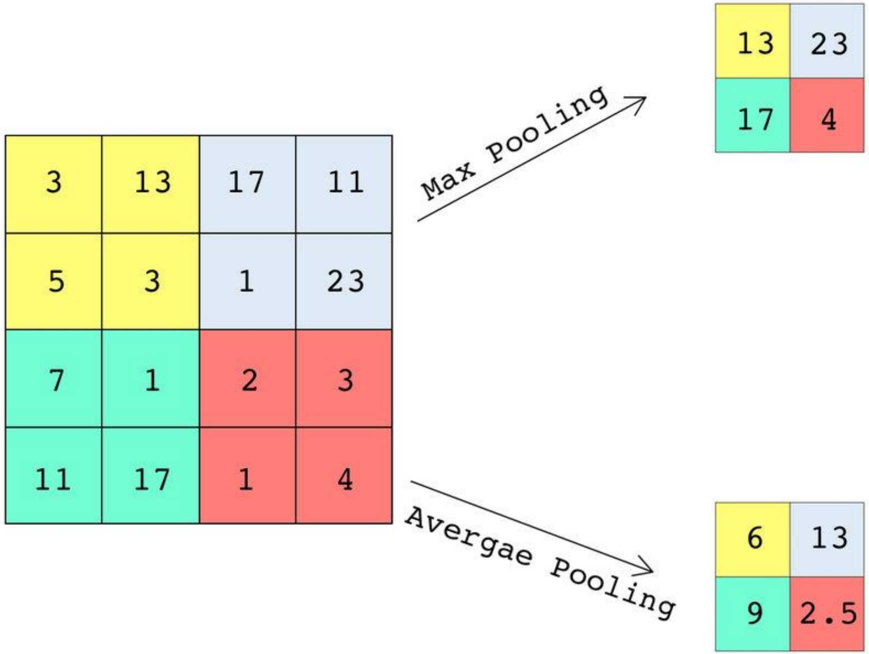
Pooling layers, which do not have learnable weights, are used to apply an even stronger down sampling to their input

Useful for reducing the dimensions of our data

Two common approaches:

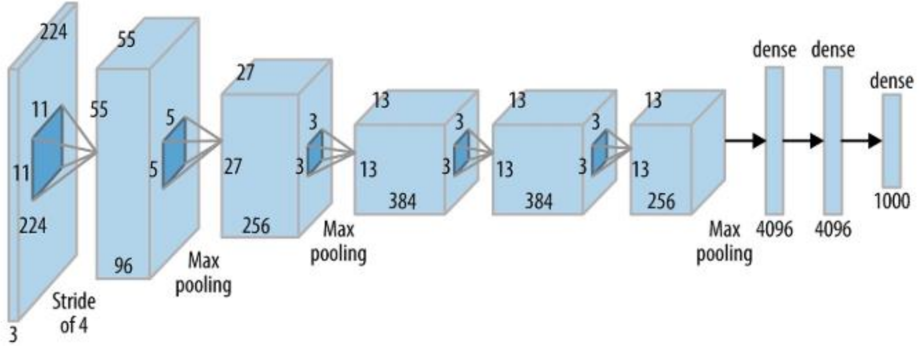
$$Average\ Pooling: y = \frac{1}{N}\sum x_v$$

$$Maximum\ Pooling: y = argmax\ (x_v)$$



CNN Example: AlexNet (2012)

- Winner of the ImageNet LSVRC-2012 challenge
  - Accuracy of 84.7% compared with runner-up of 78.4%
- ~62M parameters
  - 5 convolution layers
  - Max pooling
  - 3 fully-connected layers



AlexNet Network - Structural Details													
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
						Total							**62,378,344**

Example: ResNet (2015)

- "Residual" Neural Network
- Propagate the previous input forward throughout the training
  - Output = CNN(x) + x
- Winner of the ImageNet LSVRC-2015 challenge
  - Accuracy of 95.51% in classification tasks.
  - Outperforming humans!
- Extension of this: DenseNet
  - Perform several of these blocks throughout the network

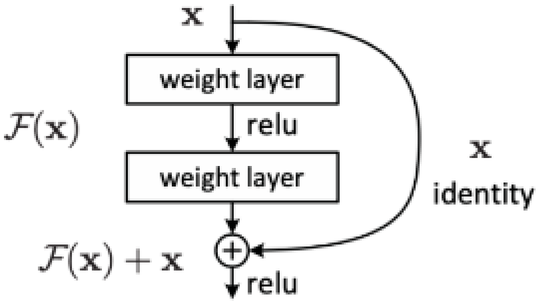
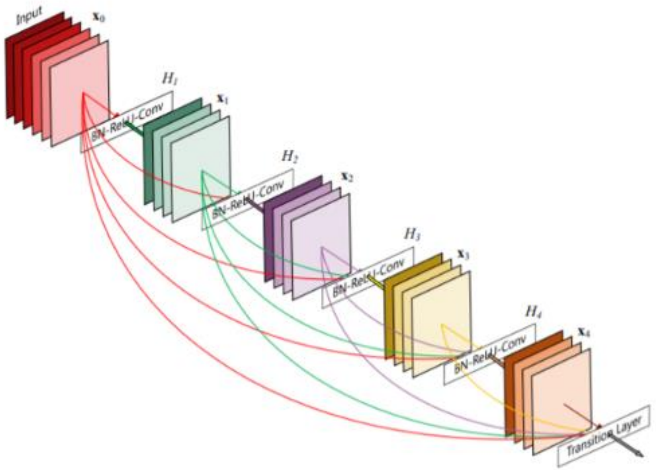


Figure 2. Residual learning: a building block.



Example: Python Code

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=2, kernel_size=2, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  # Max pooling layer
        self.conv2 = nn.Conv2d(in_channels=2, out_channels=4, kernel_size=2, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)  # Max pooling layer
        self.conv3 = nn.Conv2d(in_channels=4, out_channels=8, kernel_size=2, stride=1, padding=1)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)  # Max pooling layer
        self.fc = nn.Linear(in_features=8 * 24 * 24, out_features=4)  # Adjusted input size

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = self.pool1(x)  # Max pooling
        x = torch.relu(self.conv2(x))
        x = self.pool2(x)  # Max pooling
        x = torch.relu(self.conv3(x))
        x = self.pool3(x)  # Max pooling
        x = x.view(-1, 8 * 24 * 24)  # Adjusted input size
        x = self.fc(x)
        return x
```

In_channels specifies the number of channels in the input image

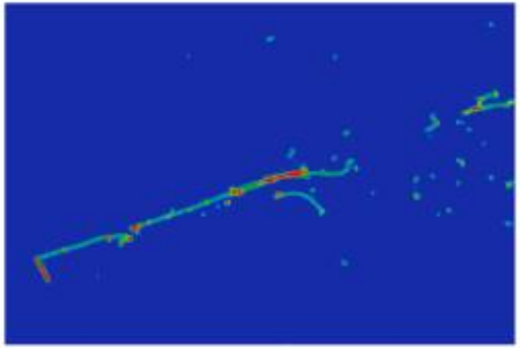
Out_channels refers to the number of filters (or kernels) that will be applied to the input image

Kernel_size defines the size of the filter or kernel used in the convolution

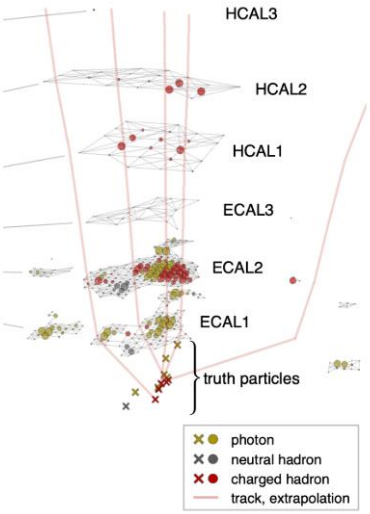
Stride controls how the filter convolves around the input image

Padding adds zeros to the periphery of the input image

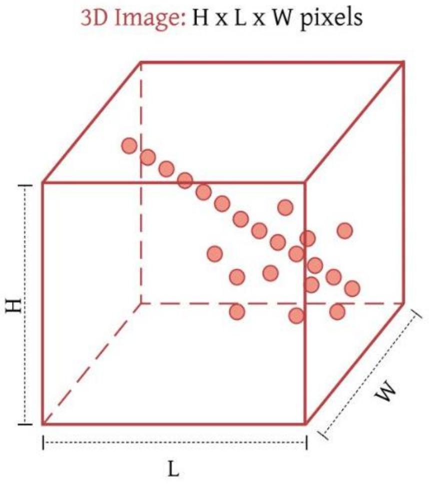
Drawbacks of CNNs

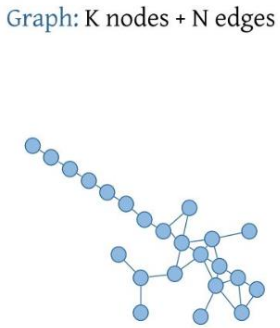


- CNNs are great for data that is dense and is on a regular “grid”
- Dense: All pixels might be helpful for the classification
- Sparse: Most pixels are background.
    - A standard CNN would perform loads of useless computations
- Sometimes our data is represented with different dimensions other than “standard”



Graph Neural Networks (GNNs)

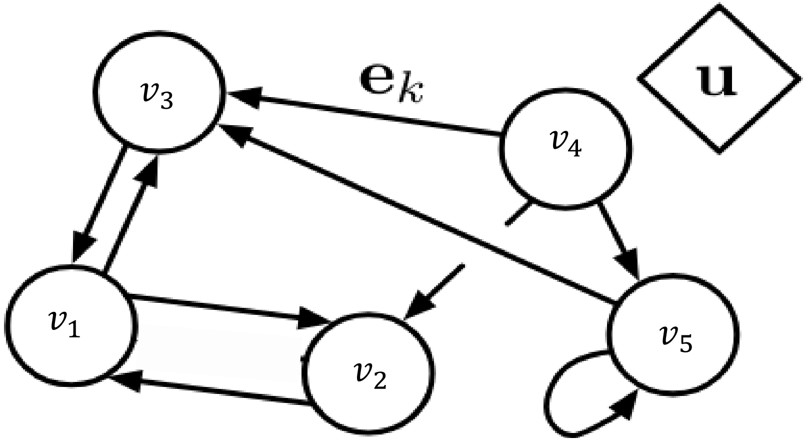




- GNNs can learn and process information from the complex structure of graphs, which makes them suitable for tasks such as node classification, link prediction, or graph classification
- Compared to CNNs, GNNs can handle graph data with variable size and structure, making them more suitable for relational data applications

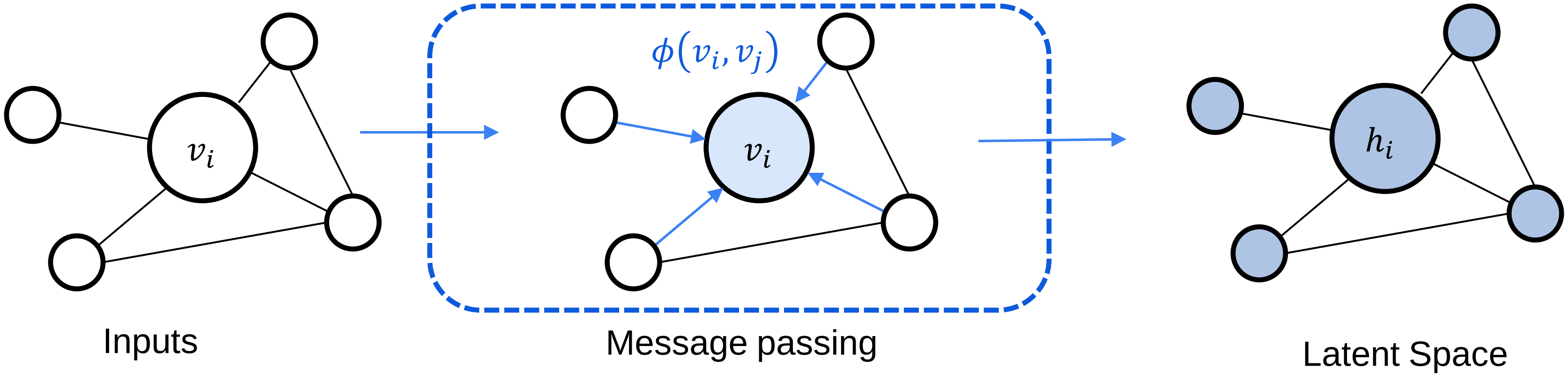
Graph Formalism

- Three main features of graphs:
  - Node ($\boldsymbol{v}_i$)
  - Edge ($\boldsymbol{e}_k$)
  - Global ($\boldsymbol{u}$)
- Graph Connectivity = adjacency matrix N x N
  - $A = \{a_{ij} = 1 \; if \; i \; is \; connected \; to \; k\}$
- Ex
- $A_{ij} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$



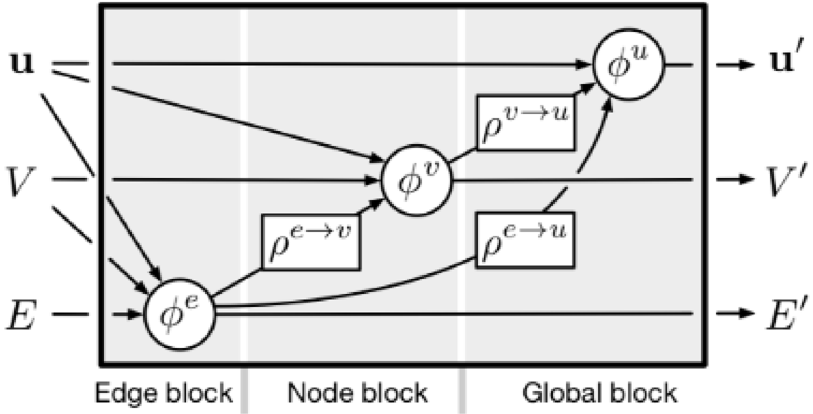
Message-passing

- For all neighbours of node compute a "message" via a NN: $\phi(v_i, v_j)$
- Update the node features by summing all the messages: $h_i = \sum_j \phi(v_i, v_j)$



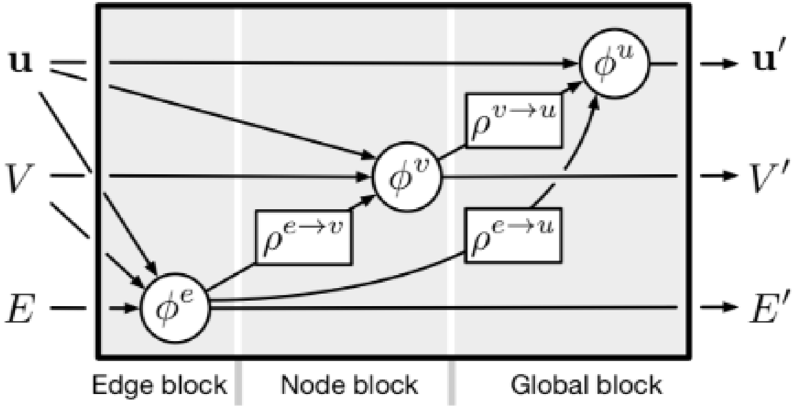
Message-passing generalised

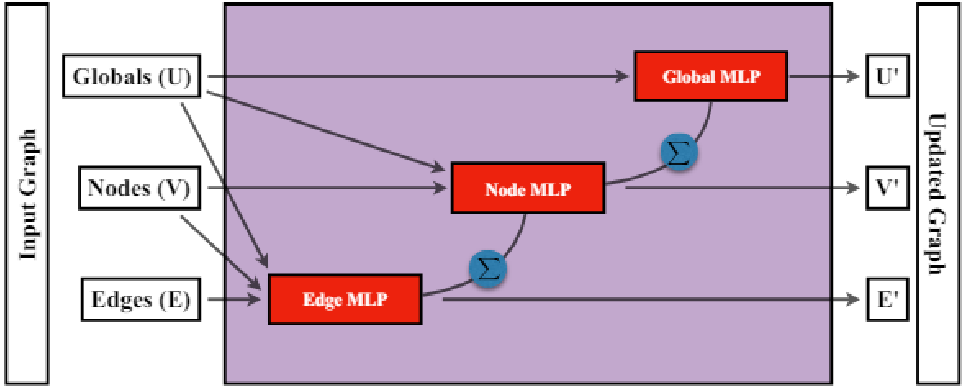
- A full GNN block predicts node, edge, and global output attributes based on incoming node, edge, and global attributes.
- ϕ^i are update functions, or our MLPs
- ρ^i are aggregation functions, which



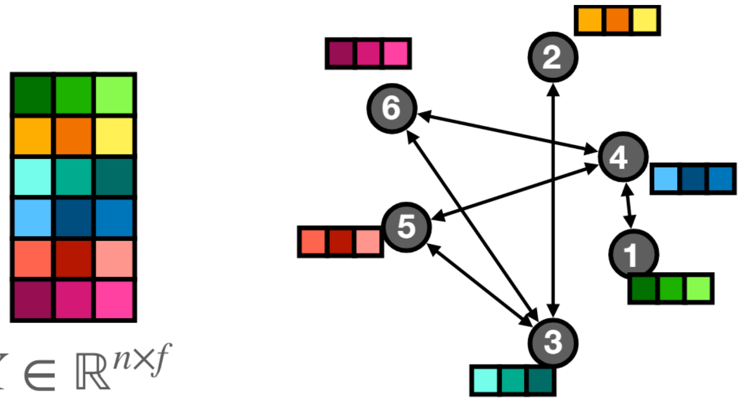
https://arxiv.org/pdf/2012.01249

Example of GN block

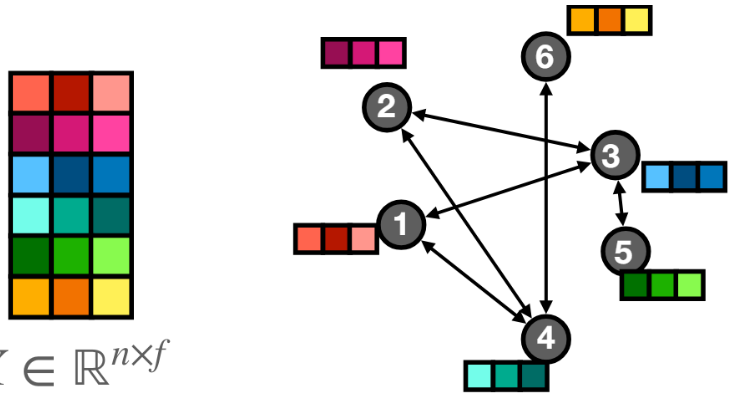




Permutation Invariance



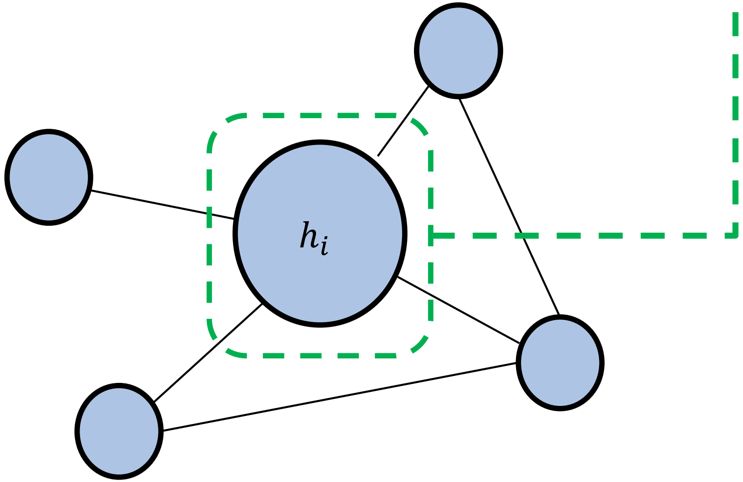
$X \in \mathbb{R}^{n \times f}$



$X \in \mathbb{R}^{n \times f}$

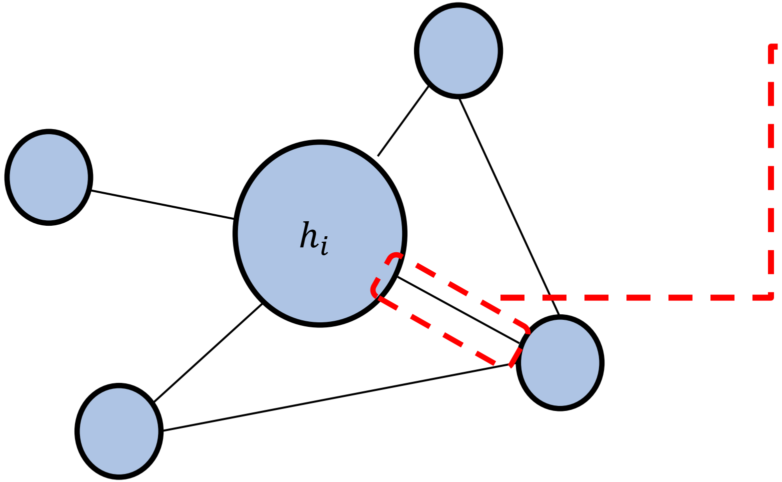
- One property of graph data is that the outputs should be invariant under permutations of nodes
  - i.e. node ordering does not matter
- Node- or edge-level outputs should be equivariant under permutations of the nodes (outputs should be permuted if inputs are permuted)

How to use GNNs



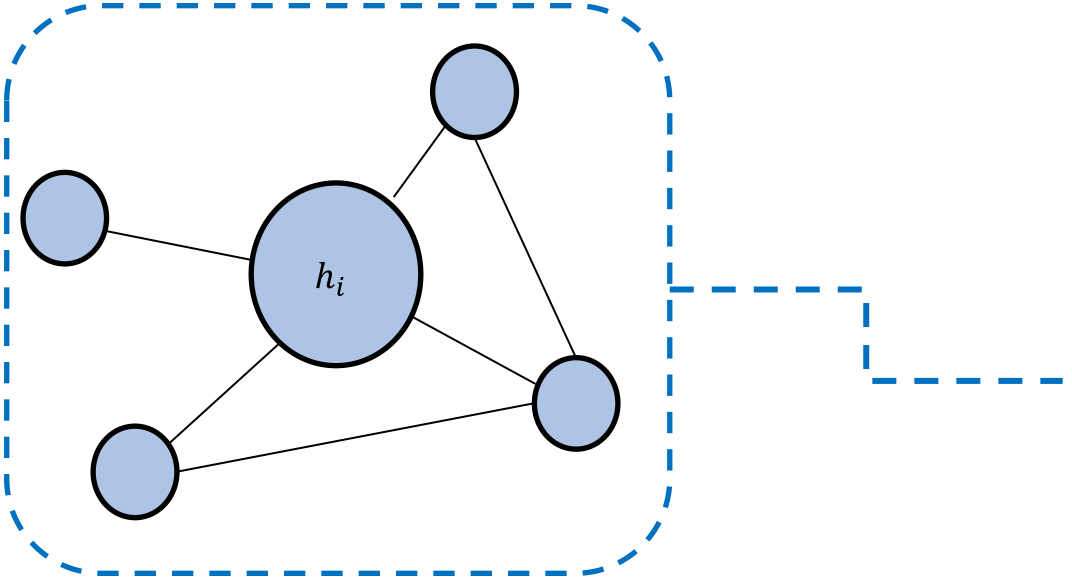
- Node-level tasks:
    - Predict a label, type, category or attribute of a node
    - E.g. detect fake accounts in a large social network with millions of users

How to use GNNs



- Node-level tasks:
  - Predict a label, type, category or attribute of a node
  - E.g. detect fake accounts in a large social network with millions of users
- Edge-level Tasks
  - Given a set of nodes and an incomplete set of edges between these nodes, infer the missing edges
  - E.g. predict the probability that a user will be interested in a product

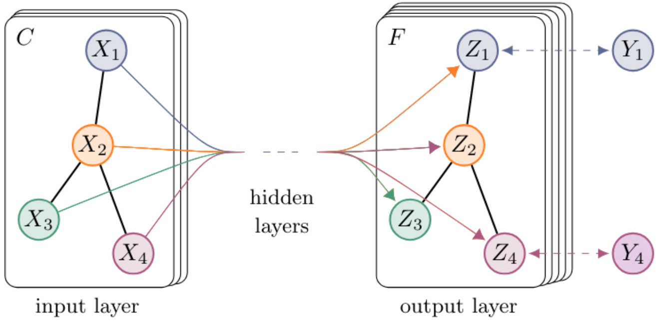
How to use GNNs



- Node-level tasks:
  - Predict a label, type, category or attribute of a node
  - E.g. detect fake accounts in a large social network with millions of users
- Edge-level Tasks
  - Given a set of nodes and an incomplete set of edges between these nodes, infer the missing edges
  - E.g. predict the probability that a user will be interested in a product
- Graph-level Tasks:
  - Carry a classification, regression or clustering task over entire graphs
  - E.g. predict IceCube event classification

Example: Graph Convolution Network (GCN) Layer

- Most cited GNN literature and is very commonly used
- In pytorch_geometric: GCNConv
- Enforce self-connections by adding the identity matrix to the adjacency matrix
- Renormalization trick to solve exploding / vanishing gradient problems.
- Con: Does not natively support edge features



(a) Graph Convolutional Network

https://arxiv.org/pdf/1609.02907

More types here: https://arxiv.org/pdf/2204.03080

Example:

```python
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, global_mean_pool

class GNN(torch.nn.Module):
    def __init__(self, num_node_features, hidden_channels, num_classes):
        super(GNN, self).__init__()
        self.conv1 = GCNConv(num_node_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.fc = torch.nn.Linear(hidden_channels, num_classes)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        x = global_mean_pool(x, batch)  # Global pooling to get graph-level representation
        x = self.fc(x)
        return F.log_softmax(x, dim=1)
```

Exercise/Challenge

Particle Image Classification

The Challenge:

- Four type of particles (electron, photon, muon, and proton) are simulated in liquid argon medium and the 2D projections of their 3D energy deposition patterns ("trajectories") are recorded
- Use one of the supervised methods to perform image classification.
- The challenge is to develop a classifier algorithm that identify which of four types is present in an image.
- Try using the CNN or GNN provided.. Does it get better performance?
- https://github.com/mjg-phys/cdm-computing-subgroup/tree/main

Dynamic Networks

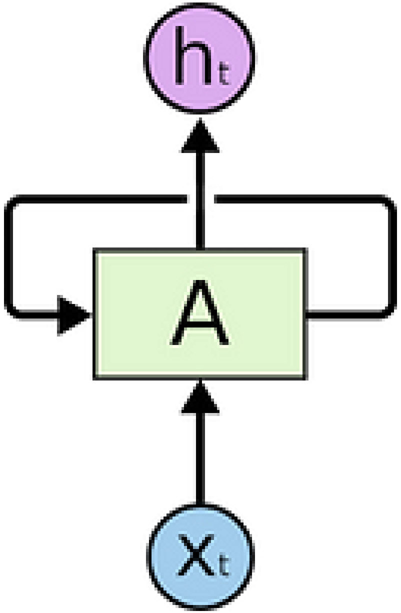
- All of the architectures we have gone through are functions of the inputs into the model and its trainable parameters

$$f(x, \phi)$$

- Some of the most powerful architectures allow for the networks parameters to become dynamics depending on the inputs

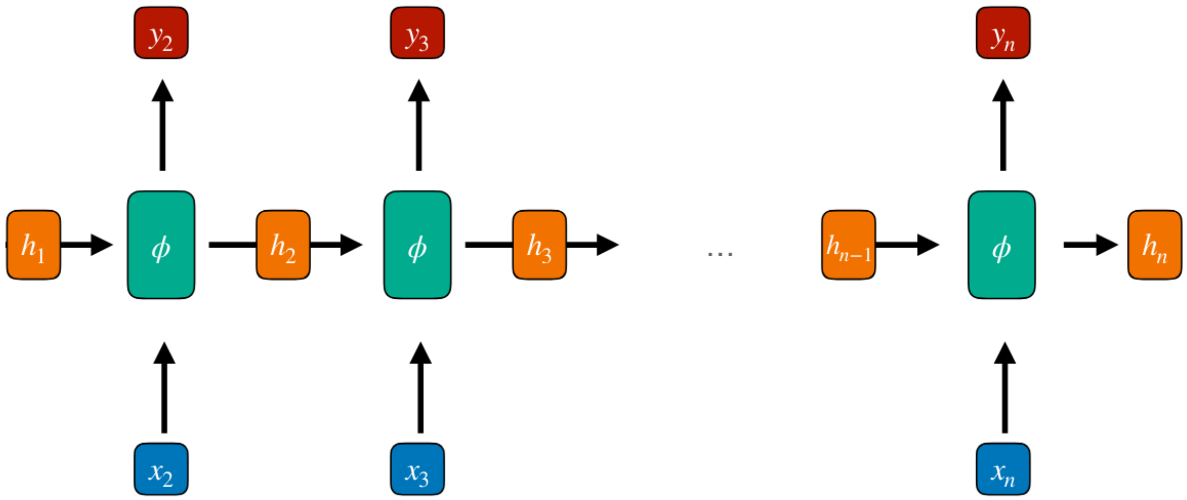
Recurrent Neural Networks (RNNs)

- RNNs have loops in them, allowing for information to persist through inputs
  - "Memory"
- For example, the output h_t will be used as in input for x_{t+1}
- Very useful for time-ordered or sequence data



Recurrent Neural Networks (RNNs)

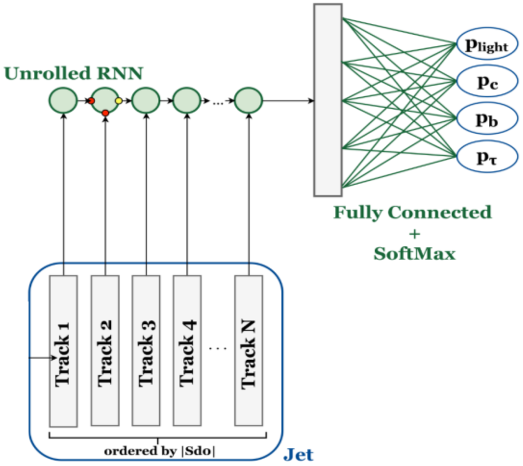
- Can be applied to arbitrary length sequences (to form a sentence)



RNNs Uses

- RNNs used to be used a lot for language models
- Example: RNN trained on all the works of Shakespeare can generate new passages
- In physics, RNNs have been used to perform flavour-tagging for jets within HEP

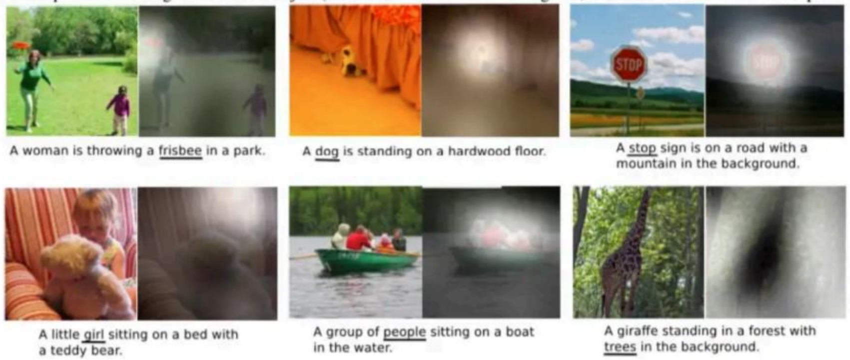
```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

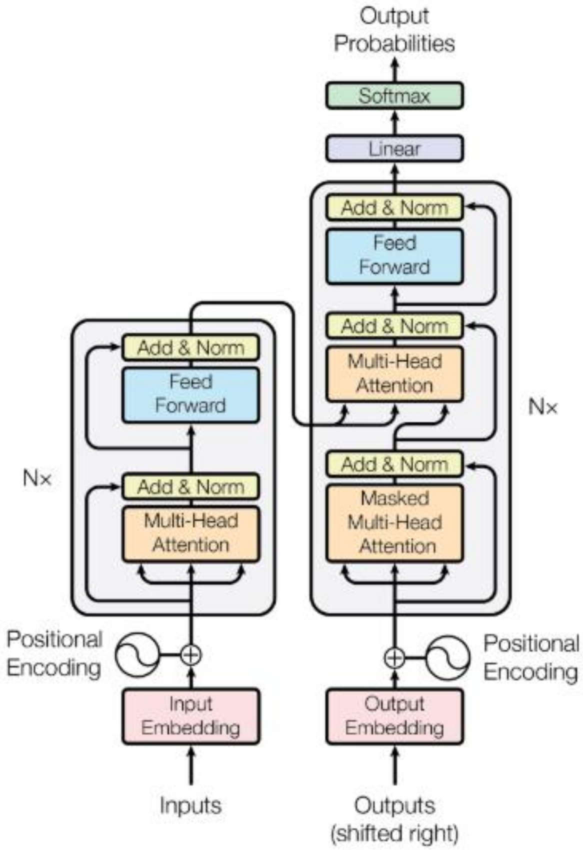


Transformers

- Combine the ideas of GNNs and RNNs
- Utilises something called "Attention"

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)





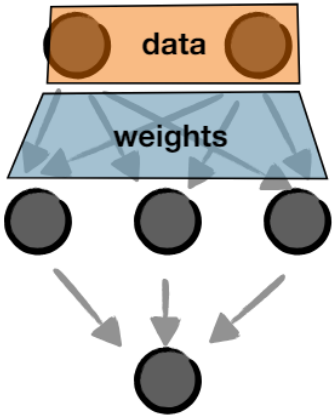
Attention

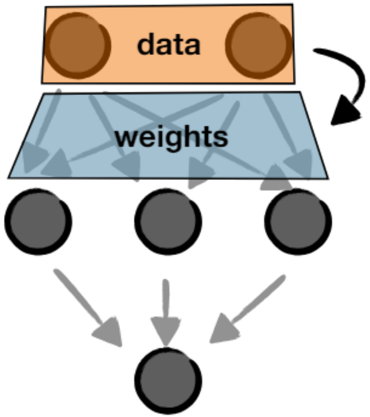
$$y = Wx \quad \longrightarrow \quad y = A(x)\,x$$

Standard Neural Net

Network with attention

weights are fixed
by the training data
input is just passed through

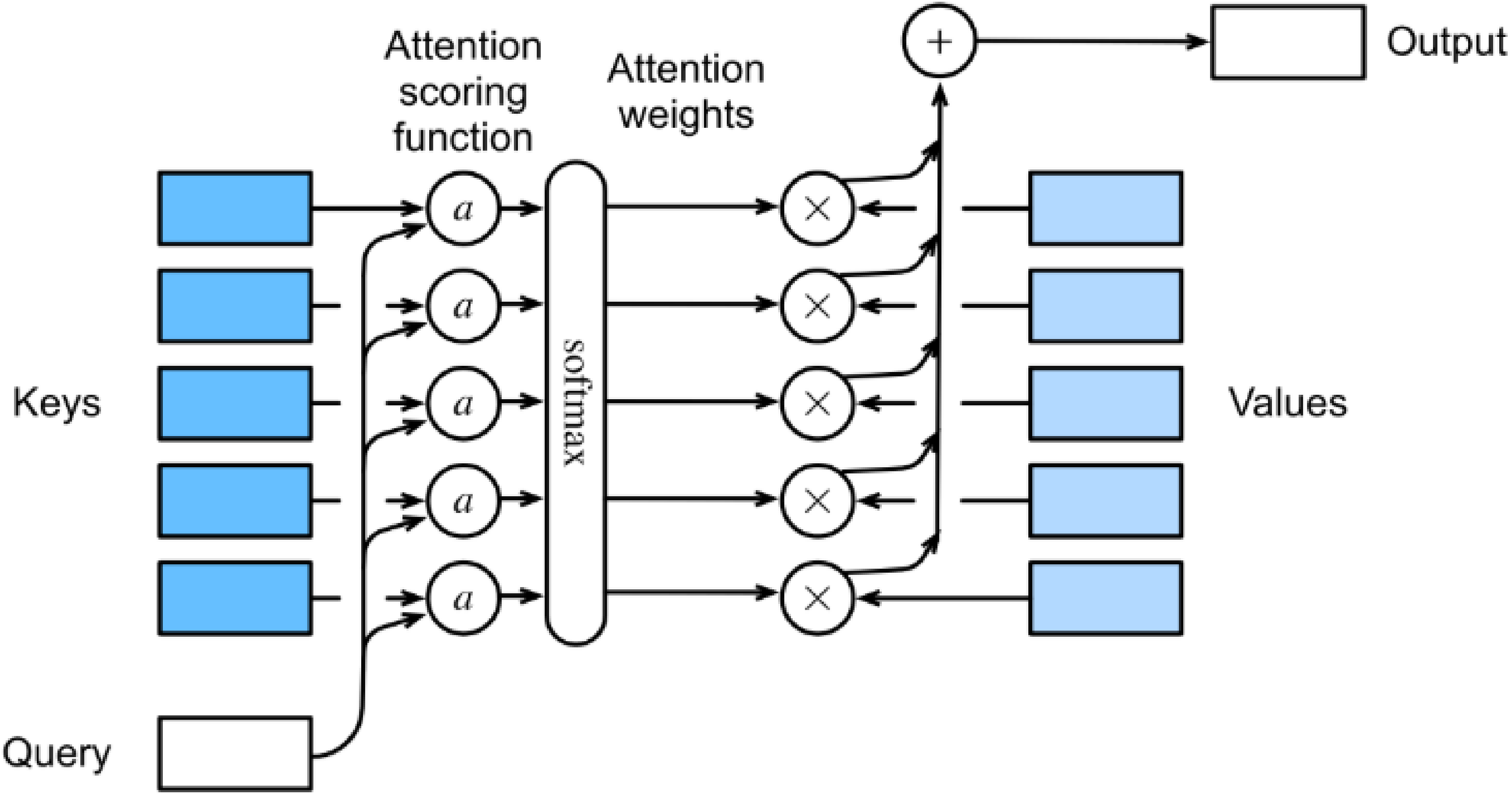




input data influences
the weights at
the time of processing

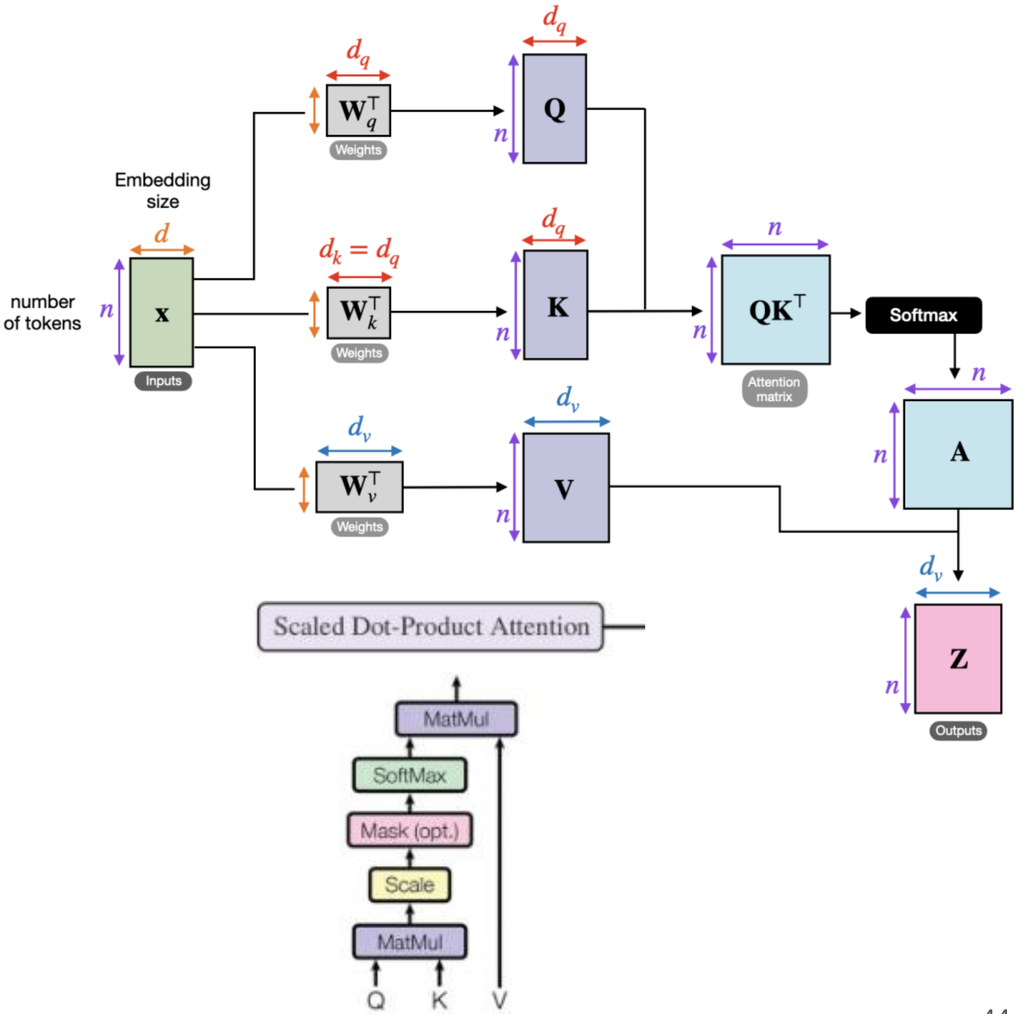
Attention

- We define
  - Queries $Q = \{q_1, q_2, \ldots, q_m\}$
  - Keys $\text{K} = \{k_1, k_2, \ldots, k_n\}$
  - Values $\text{K} = \{v_1, v_2, \ldots, v_n\}$
- We compute the similarity function $S_{ij} = SIMILARITY\left(q_i, k_j\right)$
- We normalise using the soft-max: $A_{ij} = \frac{e^{S_{ij}}}{\sum_{l=1}^{n} e^{S_{il}}}$
- Finally, we output a weighted average of all values based on similarity: $O_i = A_{ij}\, V_j$


Attention scoring function
Attention weights
Output
Keys
Values
Query
softmax
a
a
a
a
a

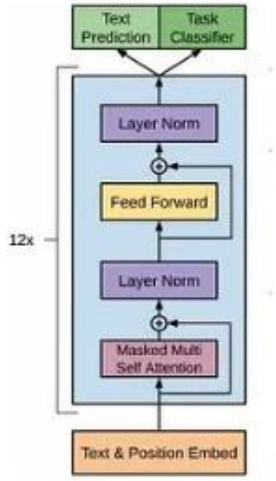
Self- Attention

- Special Case of attention where we use the same input stream, X, as the queries, keys, and values.
- Used to add context to collections of objects.
    - Make every element aware of the other elements and learn the relationship between them.
- Sometimes called Scaled Dot-Product Attention



Uses of Transformers

- Currently the "state-of-the-art" in most applications
- Are the basis of Large Language Models (LLMs)
  - ChatGPT - Generative Pre-Trained Transformer
- Often needs a lot of data to perform well



How to choose the right architecture

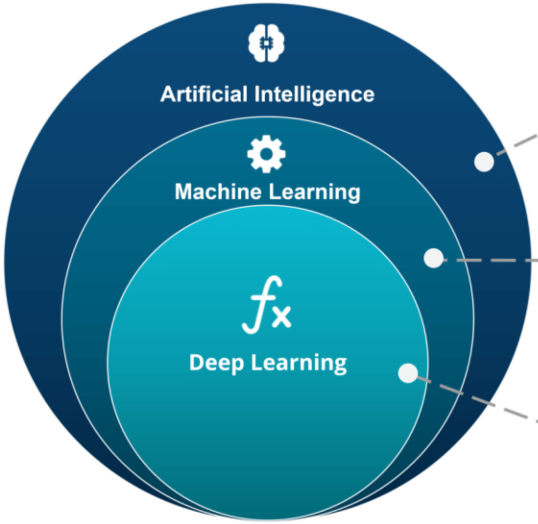
- When choosing a deep learning architecture, consider the following factors:
    - Data type and task complexity: different architectures are designed to handle different types of data and tasks.
    - Amount of training data: some architectures require large amounts of data to train effectively, while others can achieve good results with smaller amounts of data.
    - Network capacity and computing resources: having more model parameters can potentially improve a model's performance, allowing the model to learn more complex representations of the data. However:
        - Larger models require more computational resources to train and inference
        - As the number of parameters increases, so does the risk of overfitting the training data
        - Optimisation algorithms can also struggle with larger models due to increased computation time
- Overall, the best architecture for a deep learning task depends on various factors and requires experimentation and iteration to find the optimal solution.

Next Steps in a fortnight…

- Will go over some “solutions” of the google colab
  - Give it a go!
- Will go over unsupervised techniques
  - Generative Adverse Networks
  - Auto-Encoders
  - Variational Auto-Encoders
  - Normalising Flows
  - Diffusion Models

~Thanks for listening~

Addendum : AI vs ML vs DL



ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible