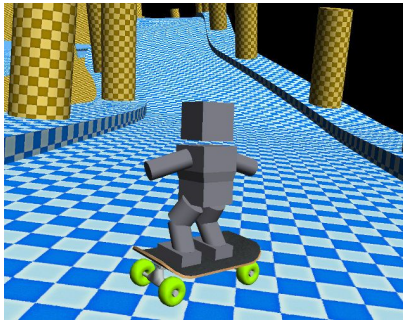
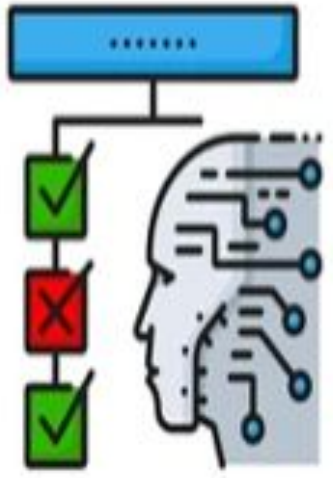


# 10-703: Recitation 1

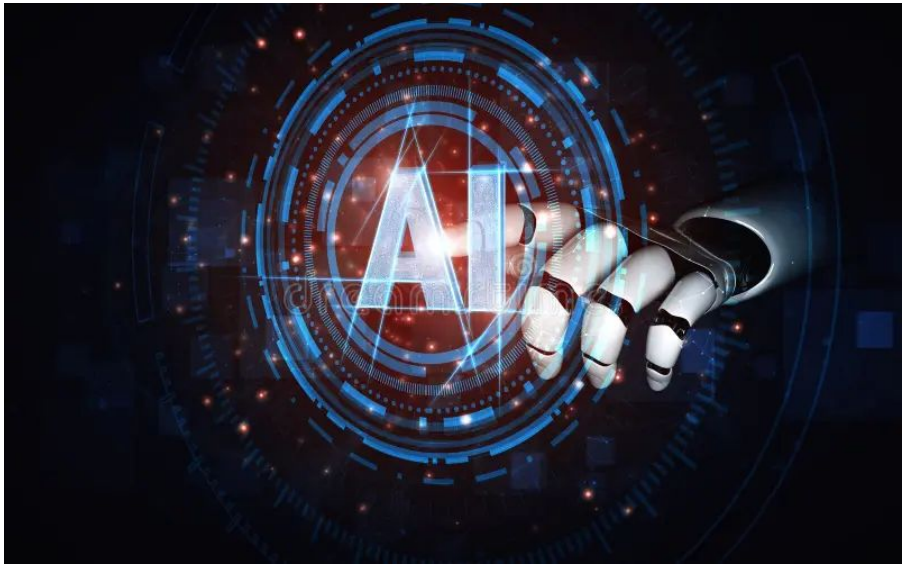
Fall 2025

Li Cao



# Topics We Will Be Covering

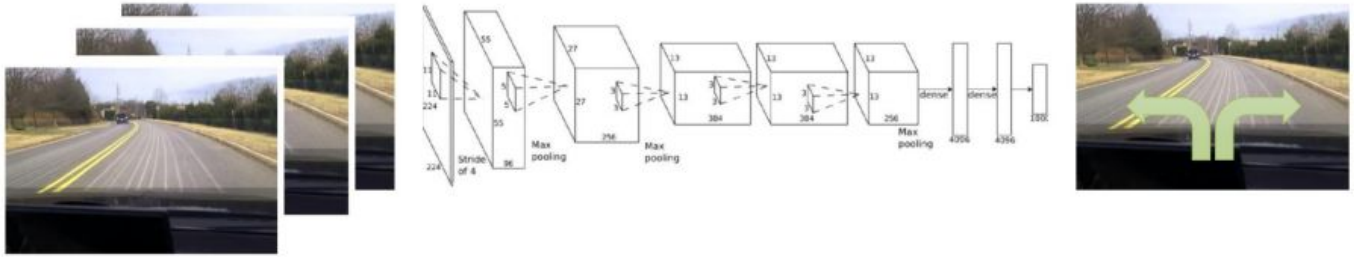
- Neural Networks
  - MLPs, CNNs, RNNs, Transformers
- Types of Learning
- Pytorch Tutorial
- Gymnasium Tutorial
- Extra Resources




Source: https://www.dreamstime.com/illustration/assistant-design.html?pg=16&view=latest-uploads

# Why deep learning?

*“Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.”* *– LeCun, Bengio, Hinton*




Scales well with large amounts of data

Less reliant on human feature engineering

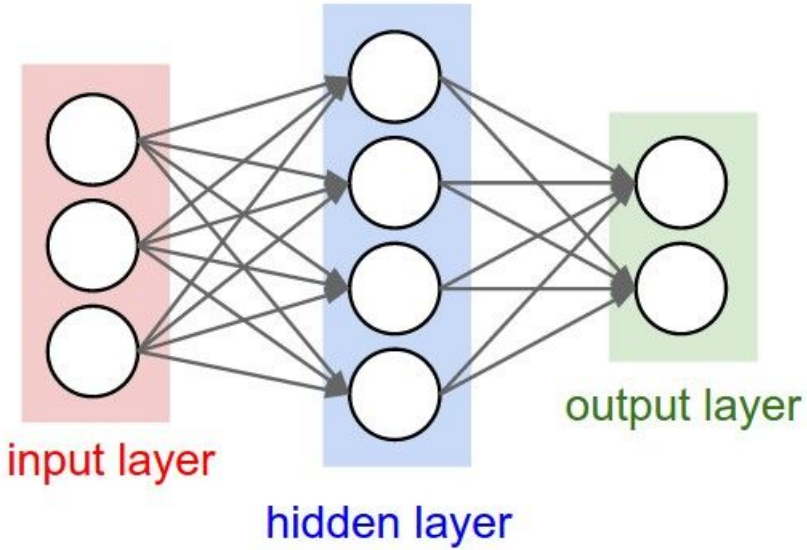
**Learn General Function Approximation**

# Why “deep” RL?

- Distinction between *classical* and *deep* RL
  - Traditionally, restricted to simpler function approximators (e.g. discrete lookup tables)
    - **Restricted to tabular methods (ex: bandits)**: problems in which the state and actions spaces are small enough for approximate value functions to be represented as arrays and tables
  - Modern deep learning methods can handle much more complex functions
- Function approximation is super useful for RL (and in many fields)
  - Represent agents, reward functions, dynamics models, etc. using deep networks
  - **Handle richer input modalities** like images and text; classical methods often rely on hand-crafted low-dimensional features
- The use of deep learning is most useful in problems with high-dimensional state space
  - This means that with deep learning, Reinforcement Learning is able to solve more complicated tasks with lower prior knowledge because of its ability to learn different levels of abstractions from data

# Multi-layer perceptrons (MLPs)

- A universal function approximator
  - Very general-purpose
  - Limited inductive bias
- Stack of *feedforward* layers
  - Input layer, hidden layer(s), output layer
  - **Nonlinear** activation function
- Hierarchical representation

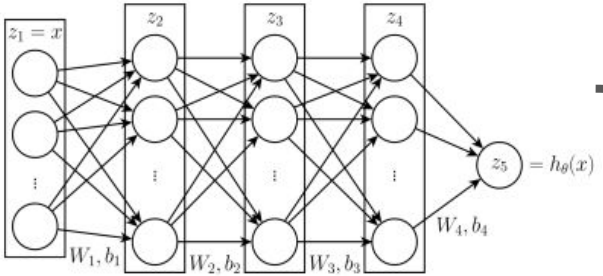



**This is most of what you need for the homeworks (architecture-wise)!**

Source: https://cs231n.github.io/neural-networks-1/

# Multi-layer perceptrons (MLPs)

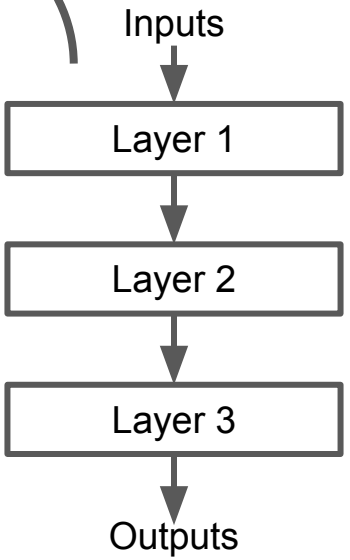
- Some notation
  - Input features: $x \in \mathbb{R}^d$
  - Outputs: $y \in \mathcal{Y}$
  - Network parameters: $\theta \in \mathbb{R}^k$
  - Network function: $h_\theta : \mathbb{R}^d \to \mathcal{Y}$
  - Activation function: $f \in \{\sigma, \tanh, \text{relu}, \dots\}$
  - Loss function: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+ \in \{\text{mse}, \text{nll}, \text{bce}, \dots\}$




- Computing the network output

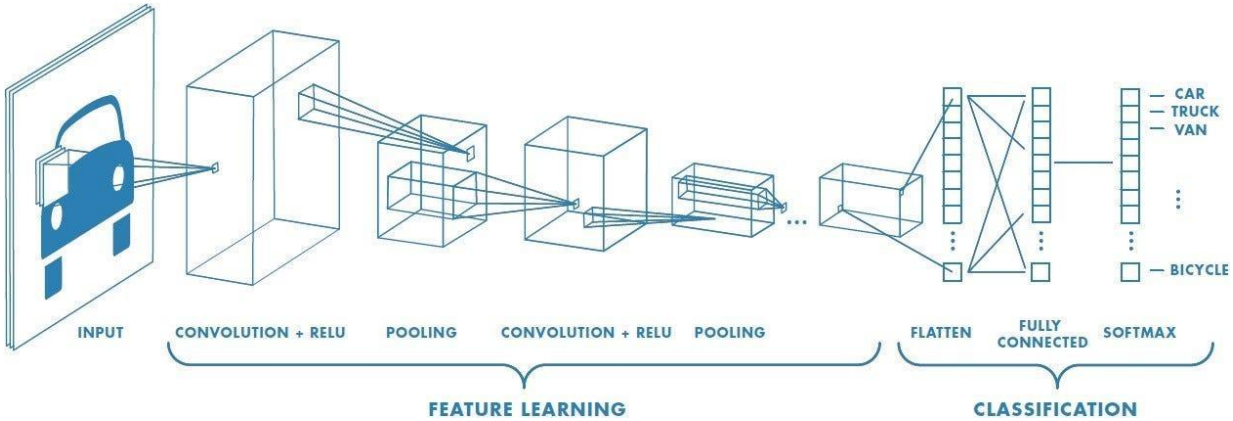
$$z_{i+1} = f_i(W_i z_i + b_i), \quad i = 1, \dots, k-1, \; z_1 = x$$
$$h_\theta(x) = z_k$$

Same network, different visualization!




Source: 10-403 recitation slides, https://cmudeeprl.github.io/403website_s22/assets/lectures/s22/recitation_1-neural_nets.pdf

# Convolutional Neural Networks

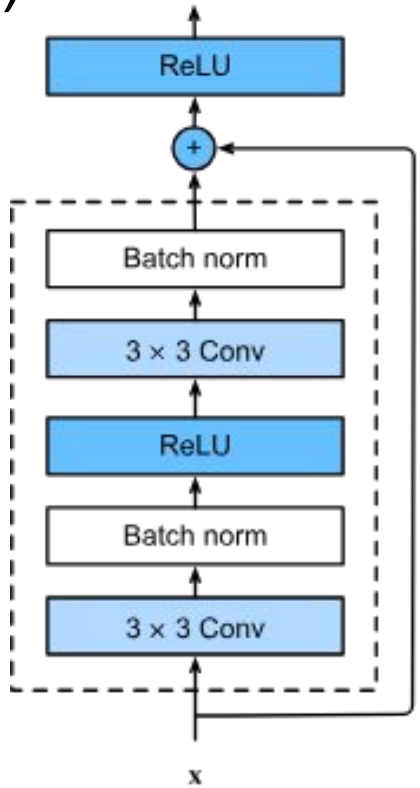



- A convolutional neural network can have several layers that **each learn to detect different features of an image**. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. The filters can start as very simple features, such as brightness and edges, and increase in complexity to features that uniquely define the object.
- Why not flatten an image and pass it through an MLP?
  - CNNs are able to successfully capture the Spatial and Temporal dependencies - **Theoretically Shift Invariant**
  - CNNs are able to fit the image dataset better due to the reduction in the number of parameters and weights reusability

Source: A Comprehensive Guide to Convolutional Neural Networks (https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/)

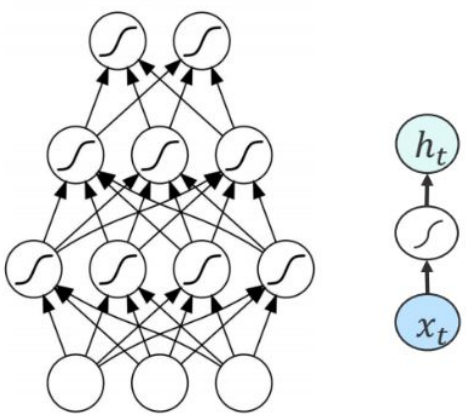
# Residual Neural Networks (ex. Resnet)

- Add a **Residual Connection** to every block
- Now the conv block just needs to output an update to the input features (helps for deep models)!
- Add **Batch Norm**
- Not super important, helps with model stability/training
- Bigger models/more data!



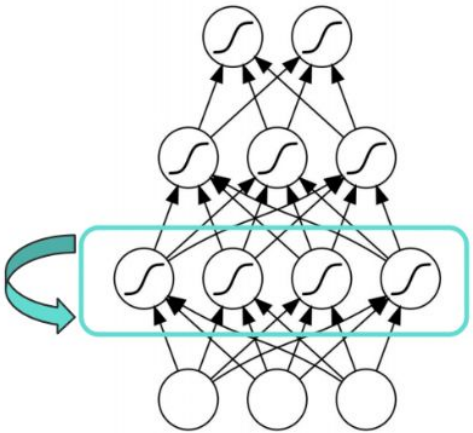

Source: Residual Networks (ResNet) and ResNEXt (https://d2l.ai/chapter_convolutional-modern/resnet.html)

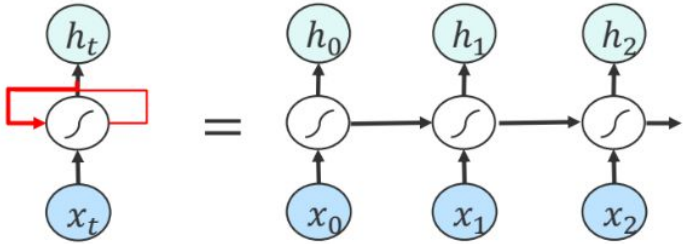
# Recurrent neural networks (RNNs) ex: next word prediction






MLP: at each timestep, the computation is independent of previous timesteps (*stateless*)

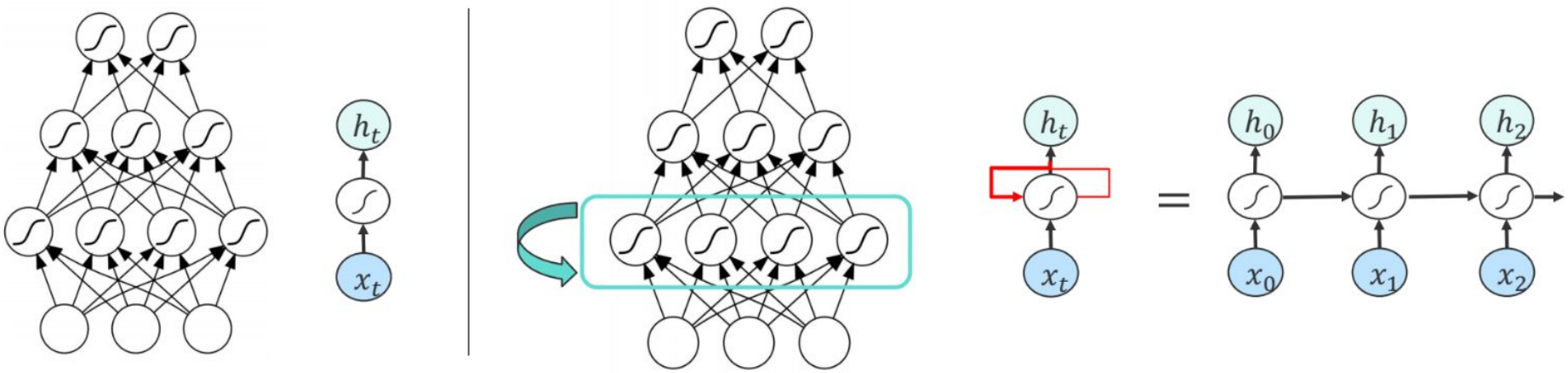





RNN: at each timestep, the computation is dependent on previous timesteps (*stateful*)

Source: Eric Xing, https://www.dropbox.com/s/uetn2j5opxjg0x8/Deep%20Sequence%20Models.pdf?dl=0

# Recurrent neural networks (RNNs) ex: next word prediction




- Natural fit for sequential decision-making problems
- **Vanishing / exploding gradient problem** (alleviated by LSTMs / GRUs, later on SSMs (shoutout Albert Gu))
- Hard to parallelize due to sequential computation

Source: Eric Xing, https://www.dropbox.com/s/uetn2j5opxjg0x8/Deep%20Sequence%20Models.pdf?dl=0

# Transformers (ex. Self attention)

- Scary word, just a different type of deep learning block!
- Notice the dotted residual connections that skip the attention block
- Feed Forward Networks or FFN (aka an MLP) still present here
- Embedding might be the initial question you ask chatGPT (i.e. “what are fun things to do in New York”)

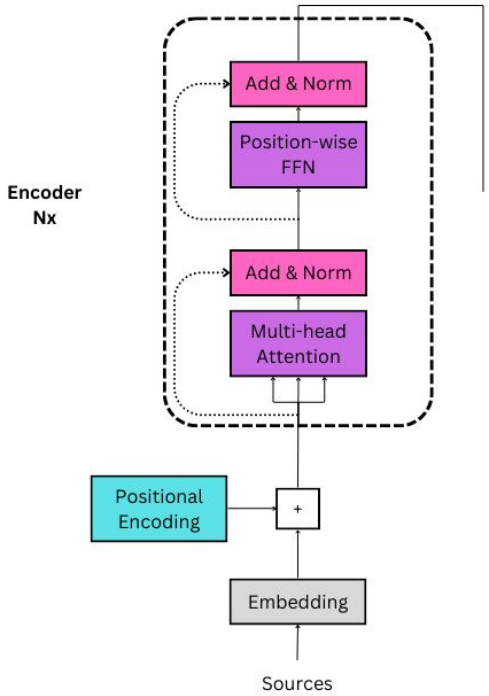
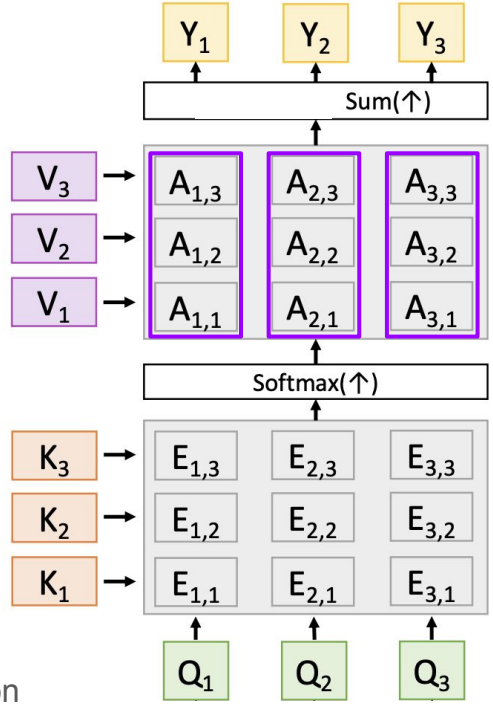



Image Source: https://medium.com/@hunter-j-phillips/position-wise-feed-forward-network-ffn-d4cc9e997b4c

# Attention Block Deep Dive

Can think of as a *differentiable dictionary lookup* that takes three arguments:

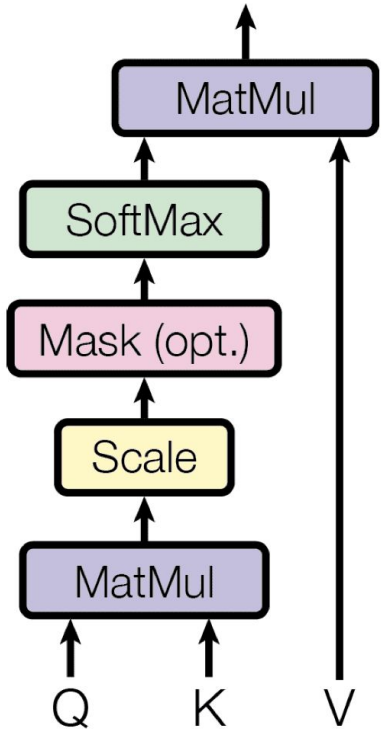
Query (**Q**), Key (**K**), Value (**V**)




- **Input:** N features size k each
- **Step 1:** Use learnable **query** matrix to create N queries size k' each
- **Step 2:** Use learnable **key** matrix to create N keys size k' each
- **Step 3:** Use learnable **value** matrix to create N values size k' each
- **Step 4:** For each query vector, compute ***dot product*** with all key vectors, ***softmax*** the resulting vector size N
- **Step 5:** For each output feature, aggregate N values using the probability distribution step 4
- **Output:** N features size k each, all of the new N features has some information from all other features

# Scaled dot-product attention

$$Q = W_q * x_{\text{input}}$$
$$K = W_k * x_{\text{input}}$$
$$V = W_v * x_{\text{input}}$$
$$A(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$




Source: Richard Socher, https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture12.pdf, UMich Deep Learning for Computer Vision https://web.eecs.umich.edu/~justincj/slides/eecs498/WI2022/598_WI2022_lecture18.pdf

# Self-attention vs Cross-attention

- Where do Q, K, V come from?
  - Recall that Q is the query, K is the key, and V is the value (all are vectors)

- **Self-attention**: Q, K, V are all computed from the same input vector

- **Cross-attention**: Q comes from input vector v1, and (K,V) comes from another input vector v2 (we say v1 *cross-attends* to v2)

$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

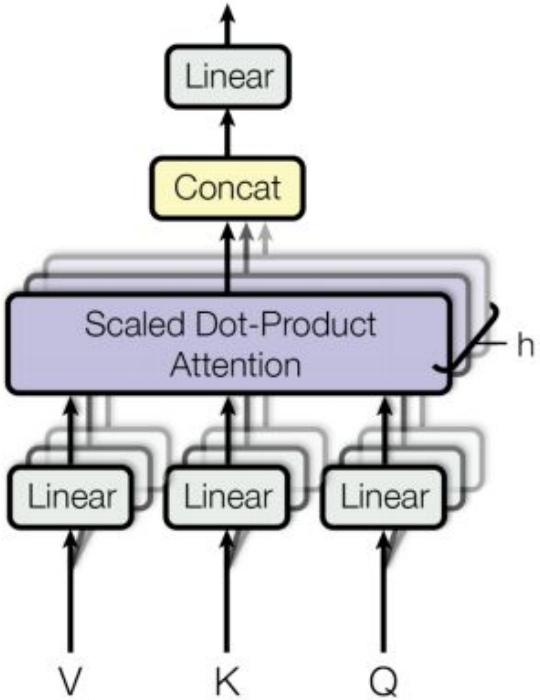
Source: Richard Socher, https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture12.pdf

# Multi-headed attention

For an input vector, we can generate an arbitrary number of Q, K, V vectors

If we generate *h* sets of Q,K,V vectors, then we have multi-headed attention with *h* heads, each with N output vectors

In the FFN step, we will turn these *h* heads back into a single set of N output vectors




Source: Richard Socher, https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture12.pdf

# Attention is all you need figure (ex machine translation)

Two main differences from standard feedforward layers:

1. Self attention first, then cross attention
2. Attention can selectively attend to / ignore specific sequence elements
   a. Standard feedforward layers can do this too, but the key difference is *sparsity*
3. Handles *arbitrary length* sequences of vectors

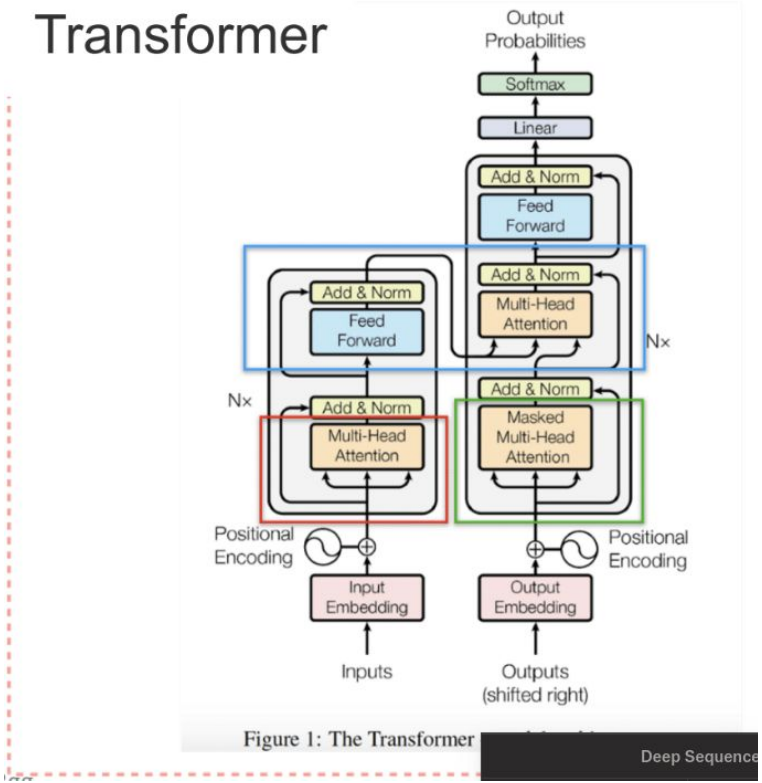



Image Source: Eric Xing, Attention is all you need: https://arxiv.org/pdf/1706.03762.pdf

# Types of Learning

1. Supervised Learning
    a. Labeled data
    b. Goal is to predict the label (i.e. classification or regression) given new data
2. Unsupervised Learning
    a. Unlabeled data
    b. Goal is to cluster data/create feature space for data/generate new data
3. Reinforcement Learning
    a. Given access to an environment
    b. Goal is to maximize the reward in that environment by taking actions
    c. You can formulate both SL and USL as a form of RL (what's the reward, state, and action?)

# PyTorch

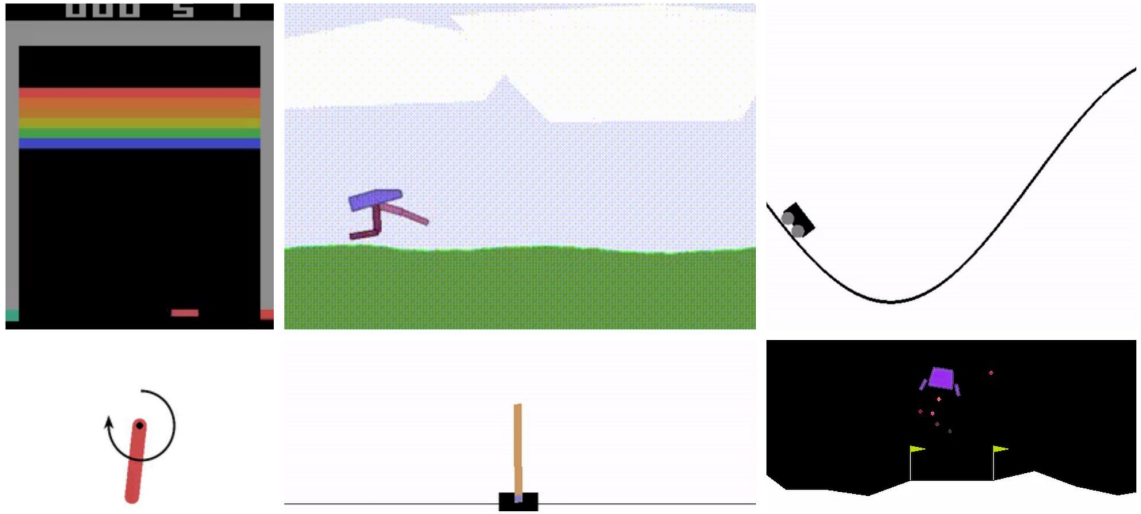
- All homeworks can be completed using PyTorch
- Read documentation and Tutorials
- PyTorch Installation: https://pytorch.org/get-started/locally/
- PyTorch Tutorials: https://pytorch.org/tutorials/
- Transformer Tutorial: https://nlp.seas.harvard.edu/annotated-transformer/

# PyTorch demo

https://colab.research.google.com/drive/1EZVQMCKYRFLVKOSejWYGjetfCBomrR9q

# Gymnasium

- Gymnasium is a Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments
- Many popular environments such as cartpole and mountain car can be used to evaluate the performance of an algorithm



# Gym Demo

We've made a Google Colab notebook as a base introduction to Gym.  Access it below:

https://colab.research.google.com/drive/1SM1MbubczWQFh9piDd1rbaxowKBUAzd6?usp=sharing

# Questions?