Day 22: Game Reinforcement Learning

This was Jack's guest lecture, it was very good.

- Deep learning is any neural network with ≥ 1 hidden layer
  - Includes backpropagation, optimization, stochastic gradient descent, hyperparameter tuning, architecture, loss functions, weight initialization, scaling, and more.
    - Focusing on architecture and scale today
  - Naturally parallel
- **Activation Function:** mapping of a neurons input to its output
  - Without an activation function, a neural network would just be a linear combination (ie. line)
  - Different activation functions are used in different contexts
- **Rectified Linear Unit (ReLU):** very popular activation function: $ReLU(x) = max(0, x)$
  - Popular because derivative is cheap to compute ($1 \text{ if } x > 0 \text{ else } 0$), converges quickly, and sparsely activated (lots of nodes of a network will be 0 and so won't activate)
  - Issues: neurons that only output negative values never fire. There are techniques to deal with this (not covered today)
- **Sigmoid:** activation function: $\frac{1}{1+e^{-x}}$
- **Universal Approximation Theorem:**

> A deep neural network of arbitrary width and a non-polynomial activation function can approximate ***any*** function

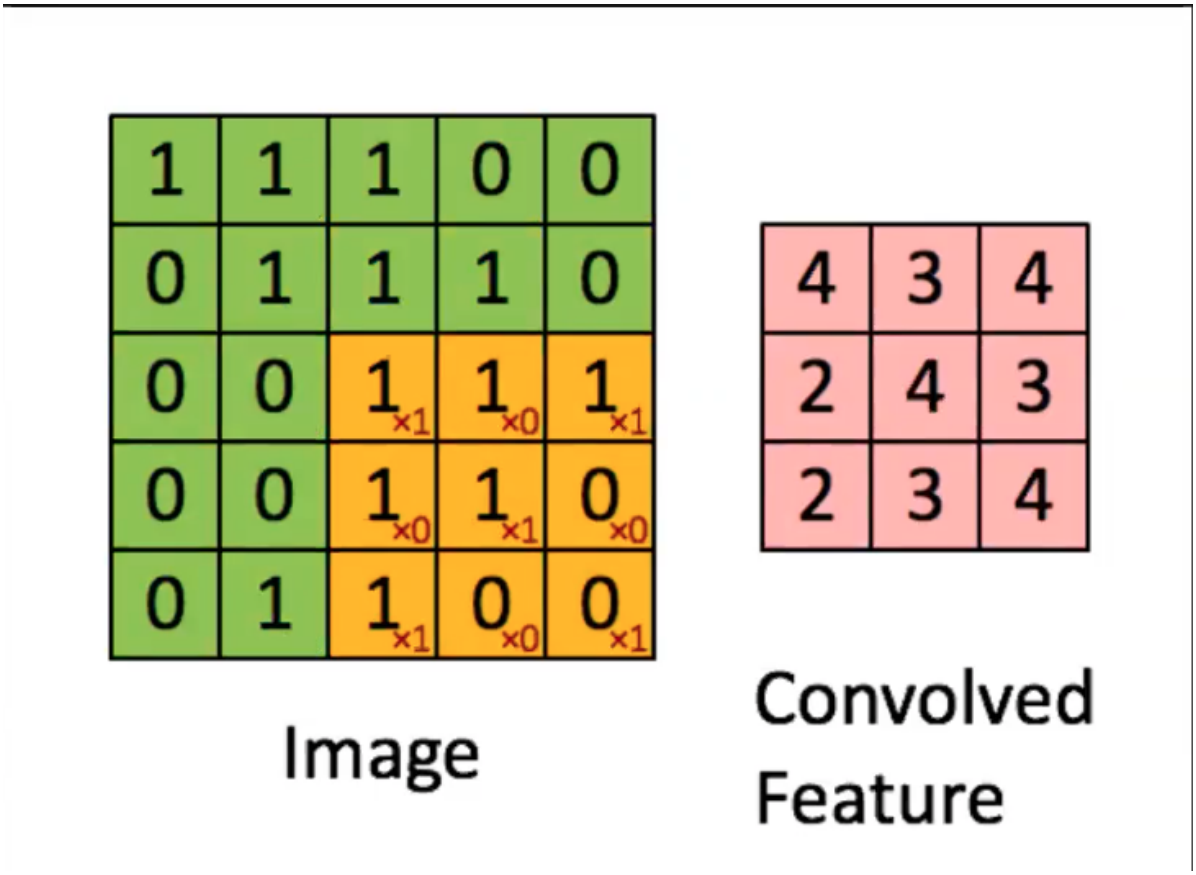
For any function f, there exists a neural network NN such that: $\forall x, f(x) \approx NN(x)$

This does not mean anything about speed or cost.

- Architecture

Convolutional Neural Network

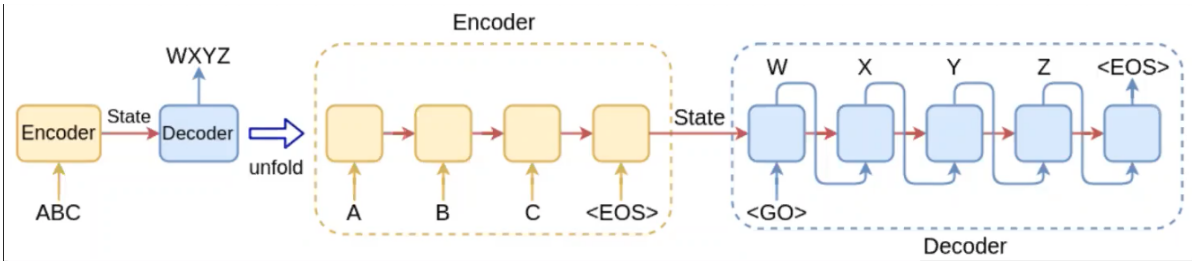
- Based off mammal eye, used for computer vision
- Example: To use perceptrons for edge detection, you'd need a neuron representing every possible edge in the image, which is impossible
- *Convolution:* A *kernel* which processes a small subs-image is applied to each subregion of the image, and the outputs of the kernel are used as the *convolved feature* to be fed into the neural network
  - The kernel is a weighted average (or dot product) of its inputs
  - If you add more dimensions, you can stack them as separate channels and run the kernel against all of them at the same time (ex. colors in an RGB image, or a series of frames)
  - One network can have multiple kernels, which produce multiple activation maps
  - You can stack convolutional layers/kernels to run against the activation map of one another (ex. one layer detects horizontal edges, another detects vertical edges, a third detects corners based on horizontal and vertical edges)
  - Kernels can be 1D too
  - CNNs are embarrassingly parallel, because each kernel region can be calculated entirely independently
    - This makes them ideal for GPU acceleration



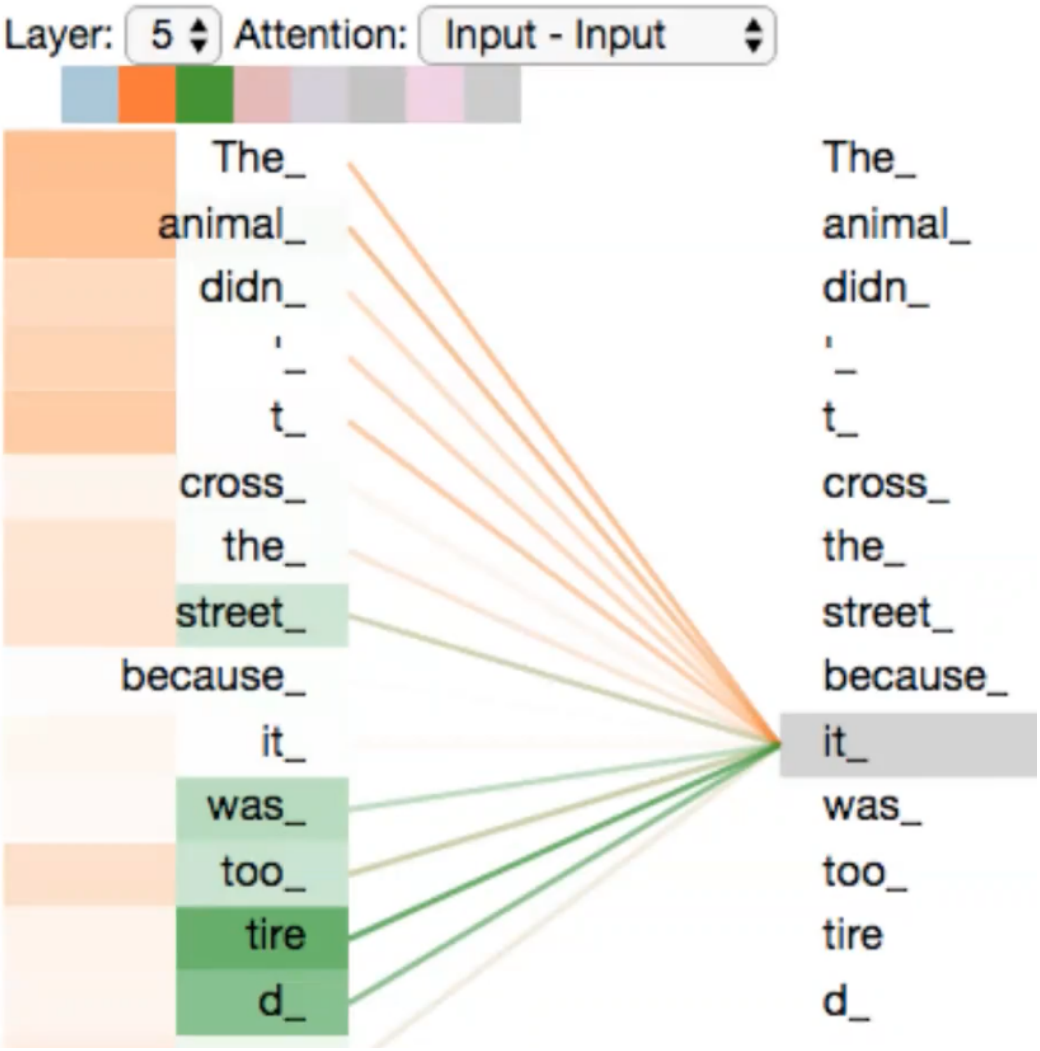
Memory and Attention

- How to keep track of things that involve time trends?
- Recurrent Neural Network: a network that has its output for the previous timestep as one of its inputs
  - **Limitation:** you can't use the output of the last input as an input for the first input
    - **One Solution:** A bidirectional RNN has a chain going in both directions
- **Vanishing Gradient Problem:** when the gradient of a neural network gets sufficiently small, then it doesn't meaningfully update it and so the network gets stuck and can't learn
  - This is specifically problematic as it relates to time-assignment: if you make a mistake early on but it doesn't affect your reward till the end, it can be difficult to meaningfully update the weights at the beginning.

- **Long/Short Term Memory (LSTM):** RNNs that can read/write to a memory vector. Not covered in detail.
- RNNs and LSTMs are commonly used for sequence-to-sequence tasks (like translating a French sentence to an English sentence):
  - State is accumulated in the hidden state vector, then unwound during decoding



- Downsides of RNNs/LSTMs:
  - Can only access memory and hidden vector, which is limited
  - Generally struggles to remember information for a long period of time
  - Only one hidden state, so can't detect different features
- Attention Networks:
  - Replacing RNNs, especially NLP
  - Basic idea: *Self-Attention*: uses the entire input instead of remembering certain information
    - Uses *way* more processing power, *but* each item and each perspective (color in the example below) can be done in parallel.
    - Very good for NLP tasks
    - Relates the input vector to itself, where each item in the input relates some amount to each other item. You can use multiple sets of weights to get multiple types of information for each item:



`It` refers to `The` and `Animal` (orange), while `tired` is an adjective that relates to `it` (green).

  - Outputs richer information about each of the inputs (put differently, outputs which things it has to pay attention to when parsing each element of the input.

- Implementation basics, given set F of features of length N:
  - Each feature $f_i \in F$ in the input has:
    - The data (x_i)
    - A value vector (v_i), where all value vectors are of length M

        - A key vector k_i of length N: a score $0 \leq k_{ij} \leq 1$ for each feature ($f_j \in F$) representing how related f_i is to f_j
    - A query vector q of length N where each value q_i is the amount you want to search for relationship to f_i (ex. all values of q are 0 except for $q_a = 1$, you're only searching for relationships to f_a).
    - To search, you get score $s_i = q_i * k_i$, which is a scalar representing the amount that f_i matched your search
    - Scores are normalized using a function called softmax, takes all the scores and normalizes them so that they add up to 1 (basically, turns them into percentages)
    - The result z is a $\sum^i \text{softmax}(s_i) * v_i$, where z is composed of each value vector v_i the percentage that f_i matched the search.
    - All of this can be parallelized
  - Ex: OpenAI's GPT-3 is an attention network trained to predict the next word in a sentence
- Generative Adversarial Networks (GANs)
  - Given a bunch of images, a GAN can produce more images that look like them
    - This is how deepfakes are made, for example
  - To train a GAN, you have two NNs: a generator (which creates new images), and a discriminator (which tries to detect which images are from the training set and which are generated).
    - You iteratively train each of them, which forces both to improve
  - The input vector to the GAN is random noise. Eventually, the network will learn to map it to things like hair length, etc.
  - Generator built using transposed convolution: start with a small vector and go to a big one (multiple layers)
    - Values of the kernels are the things being learned

- You backpropegate the discriminator's error through both the discriminator and the generator