Deep Reinforcement Learning and Control

Neural Networks – A Basic Toolbox

Recitation 1

Spring 2022, CMU 10-403

Robin Schmucker

Welcome to 10-403!

Robin Schmucker

rschmuck@andrew.cmu.edu

Office hour: Monday 3-4pm

Alex Singh

alexs1@andrew.cmu.edu

Office hour: Wednesday 5-6pm

Welcome to 10-403!

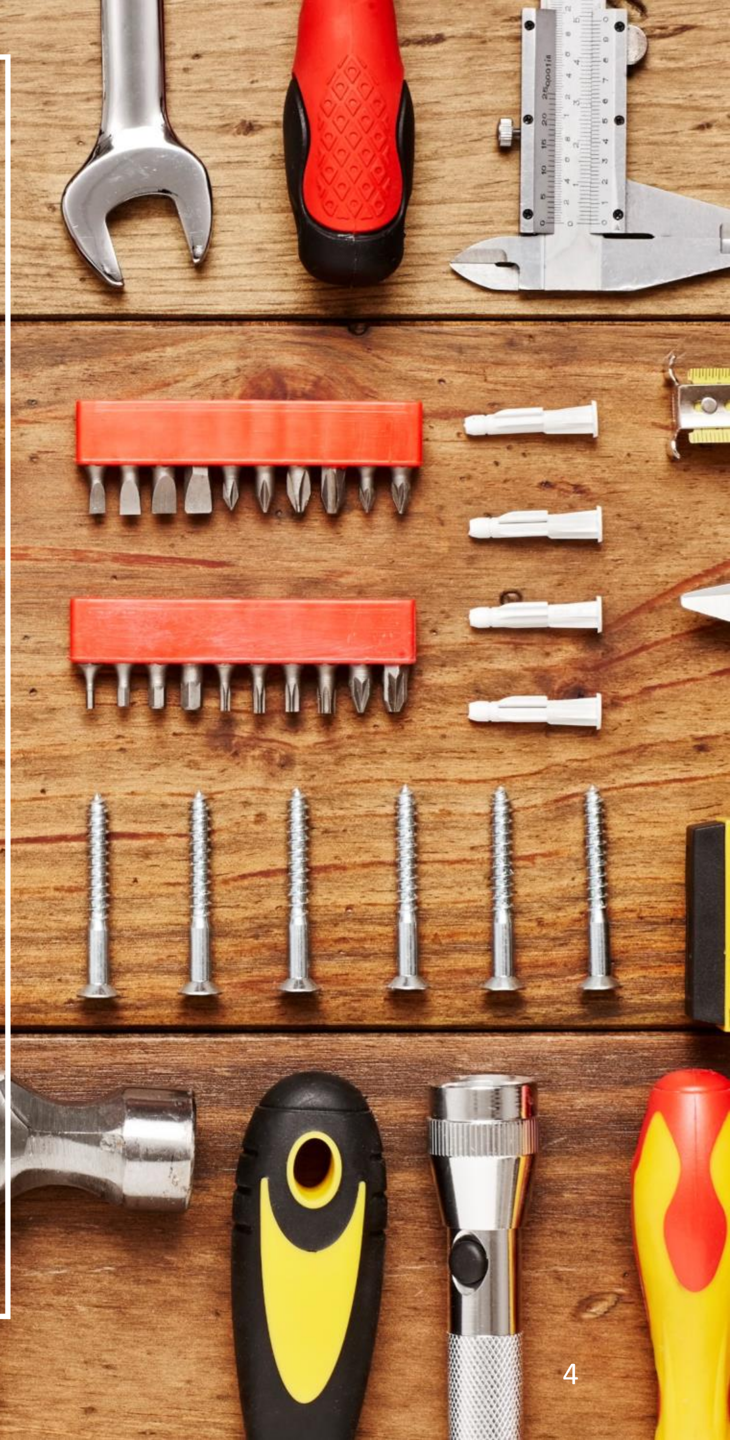
Questions?

Overview

Focus: Provide an overview of some important concepts in deep learning

-> Special classes: 10-707, 11-785, …

- Why deep learning for RL?
- Important neural architectures
- Some useful techniques for training



Some references

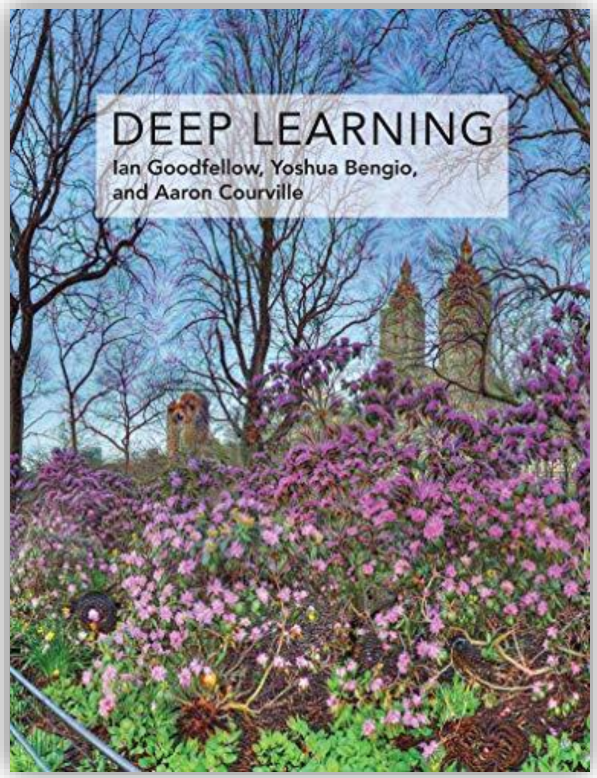
- An overview paper

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436-444.

- A standard reference

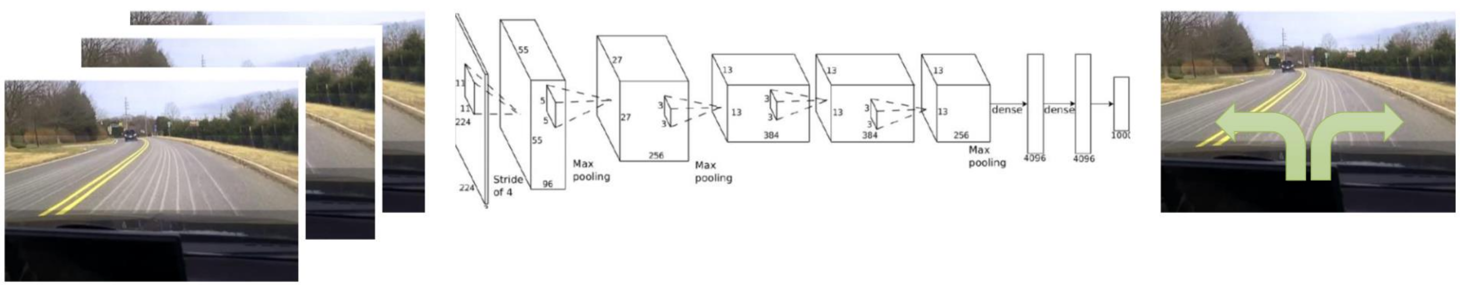
Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

- Deep learning is a fast-moving field. Often it is best to directly refer to recent research papers…



Why deep learning?

- A method for learning meaningful feature hierarchies directly from raw input data



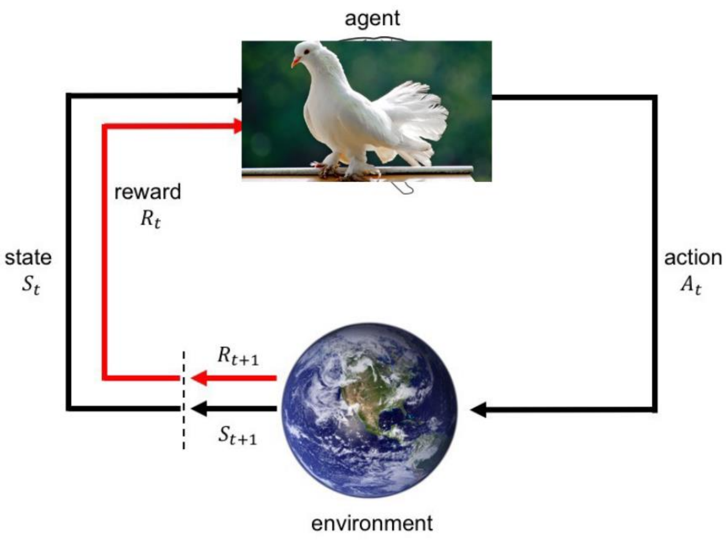
-> Frees us from human feature engineering

- Benefits from large training data
- In theory: A universal function *approximator*

Why deep learning for RL?

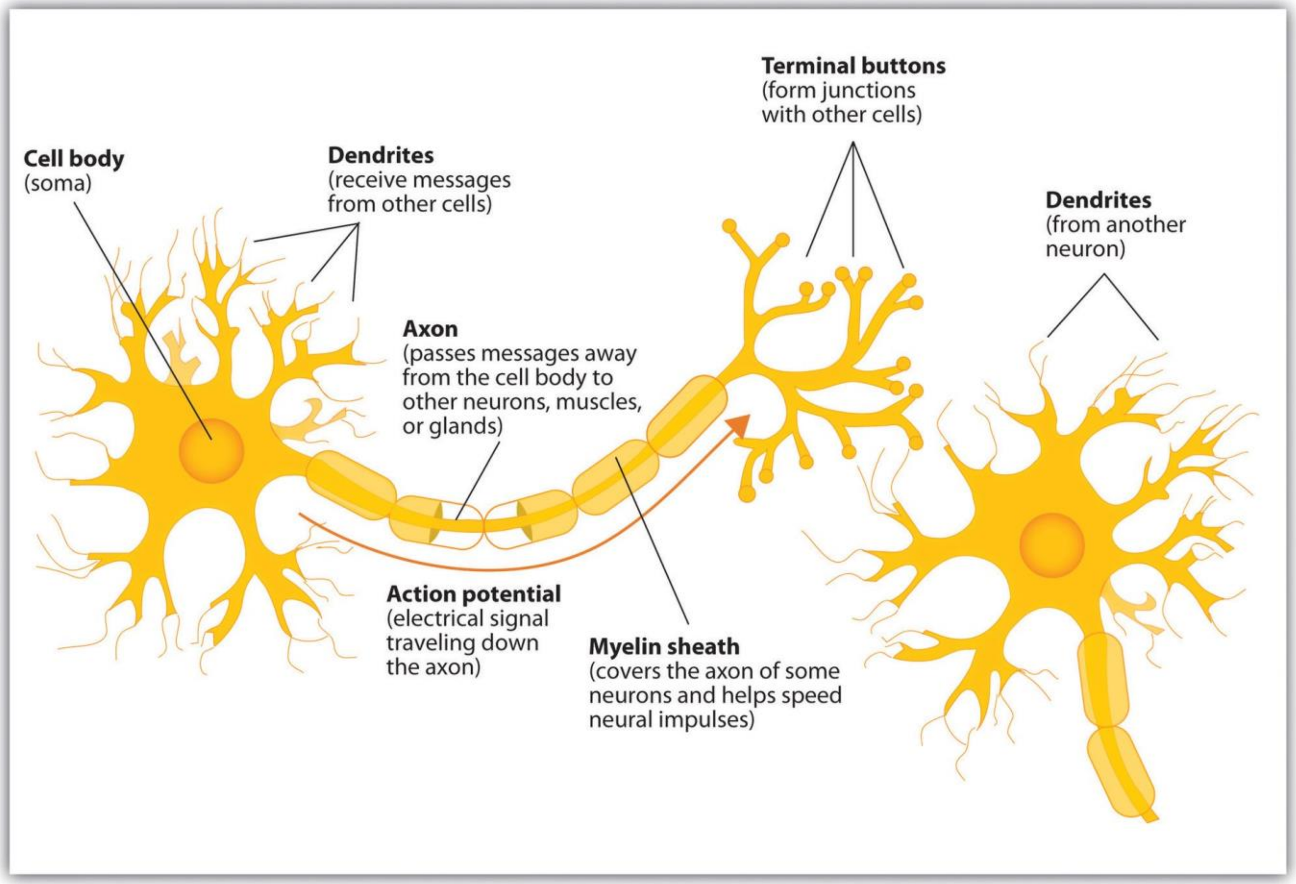
In this course we will use deep learning to approximate various types of functions. Among others…

- State representation: $f_S : X \to S$
- Policy function: $f_P : S \to A$
- Value function: $f_V : S \to \mathbb{R}$
- Model function: $f_M : A \times S \to S$



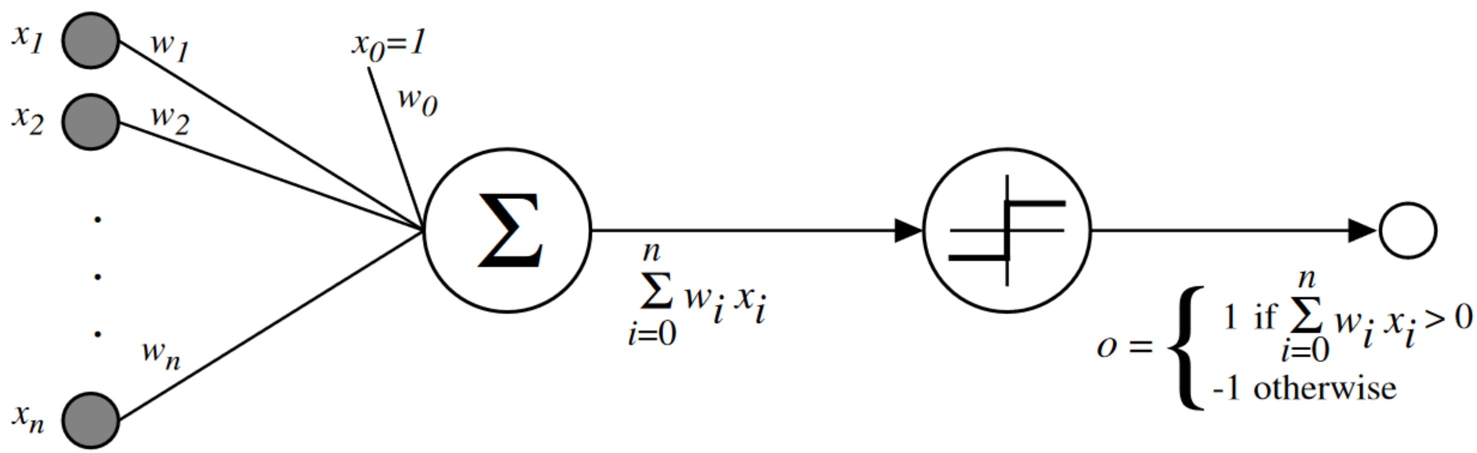
Will be formally defined in class soon

Biological Neurons



Source: [Wikipedia]

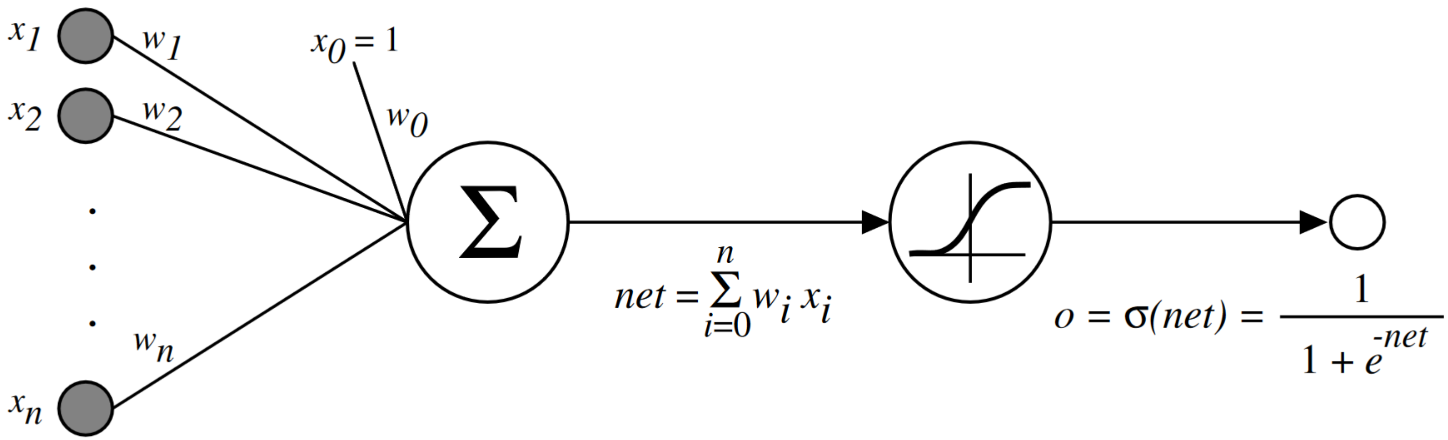
Rosenblatt Perceptron (1958)



Source: [Mitchell]

- Inspired by signaling behavior of biological neurons
- Predecessor of modern neural networks (NNs)
- Note: Non-differentiable activation function

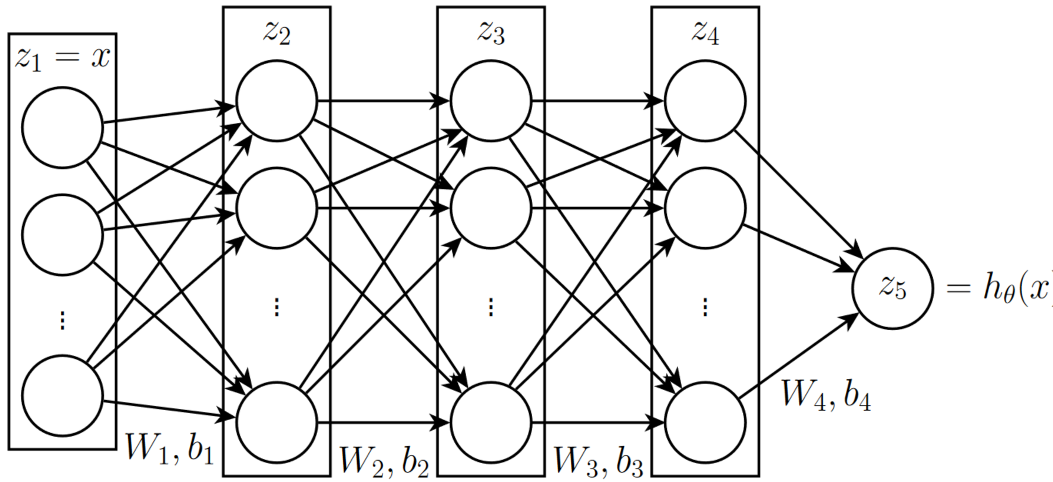
Sigmoid Unit



Source: Mitchell

- Sigmoid unit is differentiable ☺
- Weights can be optimized using gradient descent
- Problem: Cannot learn all types of functions (XOR-problem)

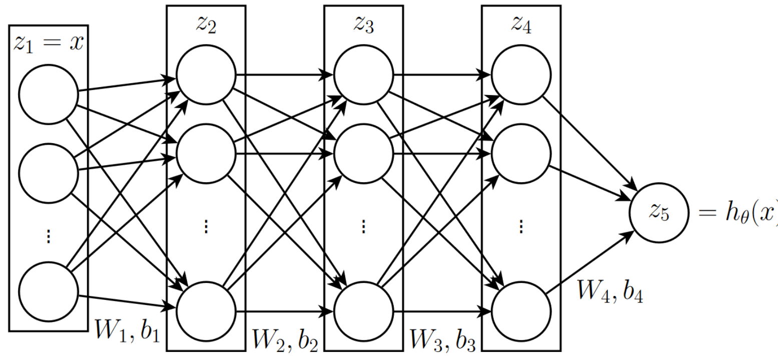
Multi-Layer Perceptron (MLP)



Source: Kolter

- Learn hierarchical feature representations
- Can be efficiently trained using GPU's, TPU's, …
- Allows us to solve XOR-Problem ☺

Multi-Layer Perceptron (MLP)



- 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: $f \in \{\sigma, \tanh, \mathrm{relu}, \dots\}$
  - Loss function: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+ \in \{\mathrm{mse}, \mathrm{nll}, \mathrm{bce}, \dots\}$
- Computing the network output

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

Multi-Layer Perceptron (MLP)

- Training data: $D = \{x_i, y_i\}_{i=1}^{n}$
- Use empirical risk minimization to identify good parameters

$$\arg\min_{\theta} \sum_{i=1}^{n} \ell(h_\theta(x_i), y_i)$$

- Idea: Compute the gradient and use optimization algorithms to find network parameters that minimize the loss ☺
- Problem: NNs are highly-nonconvex. Optimization algorithm is not guaranteed to find a global minimizer ☹
- In practice we still find good parameters…

Multi-Layer Perceptron (MLP)

- <u>Backpropagation</u>: Efficient way to compute the NN gradient

-> Split into one *forward* and one *backward* pass

function Backpropagation$(x, y, \{W_i, b_i, f_i\}_{i=1}^{k-1}, \ell)$

  Initialize: $z_1 \leftarrow x$

  For $i = 1, \ldots, k-1$

    $z_{i+1}, z'_{i+1} \leftarrow f_i(W_i z_i + b_i),\ f'_i(W_i z_i + b_i)$

  $L \leftarrow \ell(z_k, y)$

  $g_k \leftarrow \frac{\partial \ell(z_k, y)}{\partial z_k}$

  For $i = k-1, \ldots, 1$:

    $g_i = W_i^T(g_{i+1} \circ z'_{i+1})$

    $\nabla_{b_i} \leftarrow g_{i+1} \circ z'_{i+1}$

    $\nabla_{W_i} \leftarrow (g_{i+1} \circ z'_{i+1}) z_i^T$

  return $L, \{\nabla_{b_i}, \nabla_{W_i}\}_{i=1}^{k-1}$

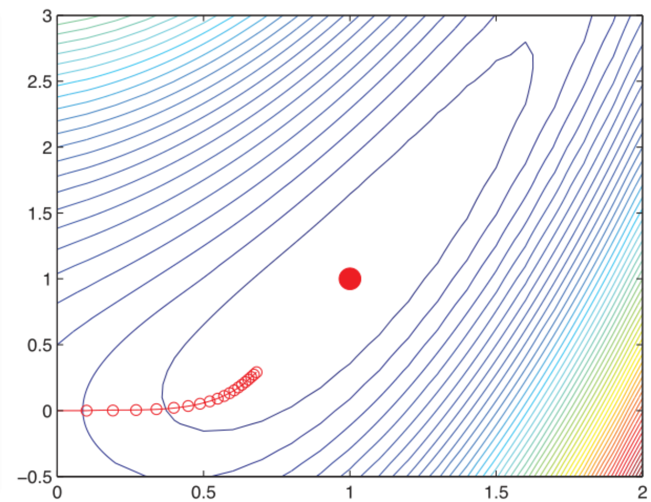
Source: Kolter

Multi-Layer Perceptron (MLP)

- <u>Stochastic gradient descent</u>: Bases optimization step on partial gradient estimate.

function SGD($\{(x^{(i)}, y^{(i)})\}, h_\theta, \ell, \alpha$)
    Initialize: $W_j, b_j \leftarrow$ Random, $j = 1, \ldots, k$
    Repeat until convergence:
        For $i = 1, \ldots, m$:
            Compute $\nabla_{W_j, b_j} \ell(h_\theta(x^{(i)}), y^{(i)}), \; j = 1, \ldots, k-1$
            Take gradient steps in all directions:
$$W_j \leftarrow W_j - \alpha \nabla_{W_j} \ell(h_\theta(x^{(i)}), y^{(i)}), \quad j = 1, \ldots, k$$
$$b_j \leftarrow b_j - \alpha \nabla_{b_j} \ell(h_\theta(x^{(i)}), y^{(i)}), \quad j = 1, \ldots, k$$
    return $\{W_j, b_j\}$

Source: Kolter

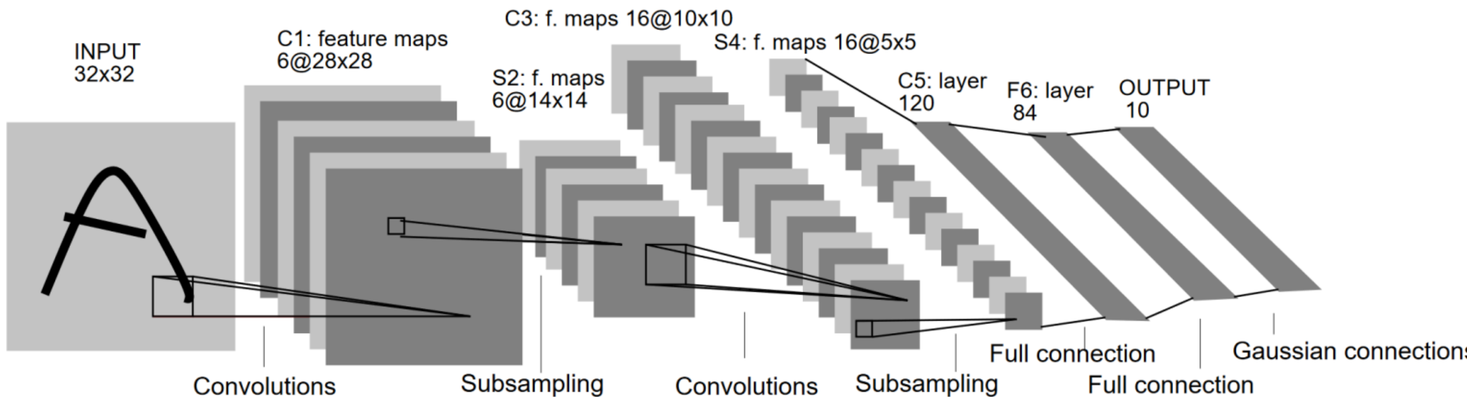


Source: Murphy

- Adam is another very popular optimization method.

Questions?

Convolutional Neural Net. (CNN)

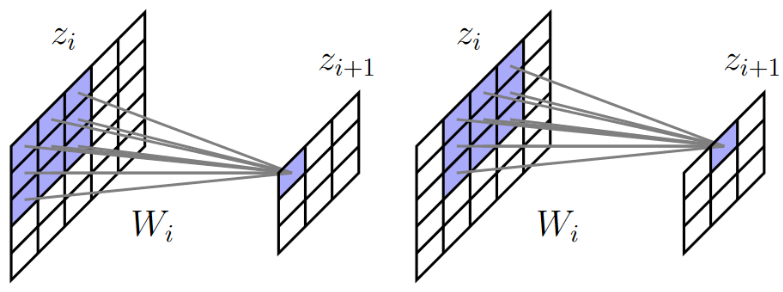


LeNet-5:[LeCun]()

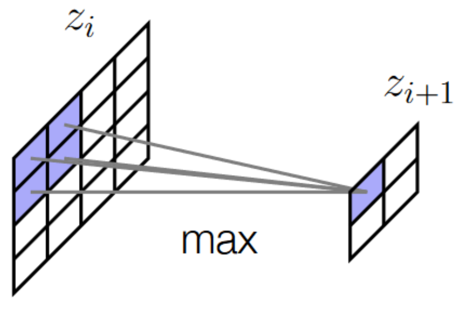
- Very popular for computer vision data
- Structural assumptions reduce number of model parameters
- Idea: Spatial relationships between pixels matter

Convolutional Neural Net. (CNN)

- <u>Convolution</u>: A matrix operator is moved across the input image and applied at every location.

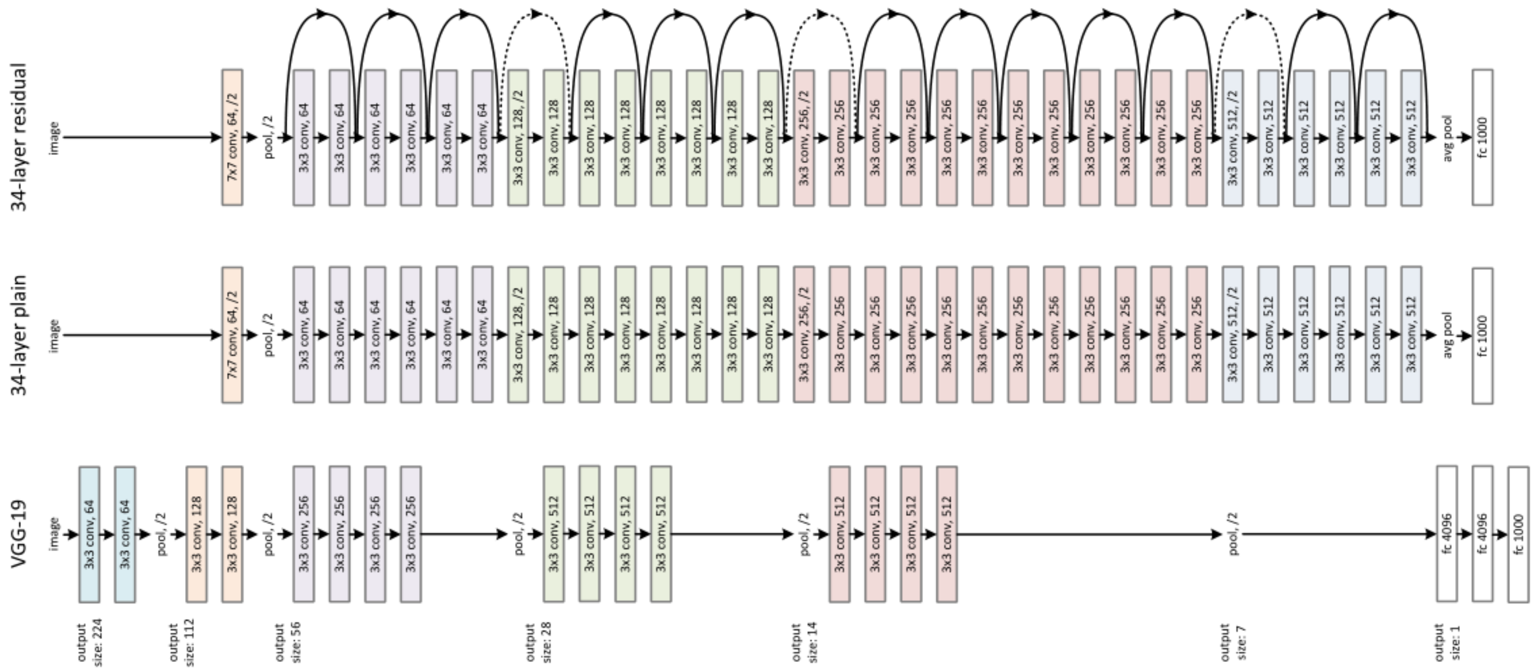


- <u>Pooling</u>: Operator that reduces output size. Popular variants are max-pooling and average.



Source: Kolter

Residual Network (ResNet)



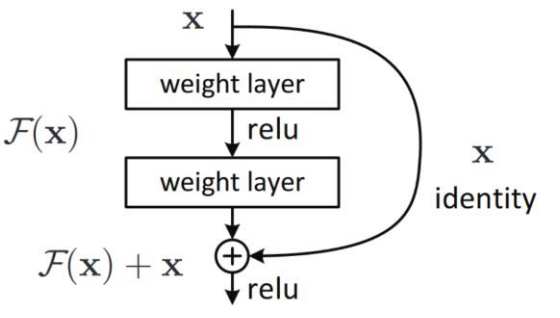
Source: He

- Very deep networks are difficult to train due to the *vanishing gradient* problem.

Residual Network (ResNet)

- Idea: Use skip connections to help the input signal to propagate in deep networks.

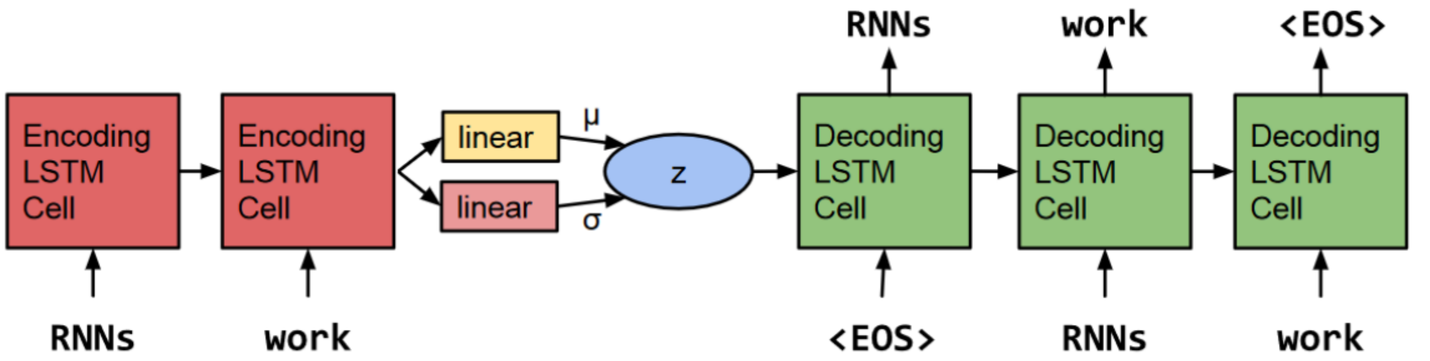
- Allows loss gradient information to pass through skipped layers



Source: He

- Deep ResNet was winner of ILSVRC 2015 competition
  - 3.57% error on ImageNet test set

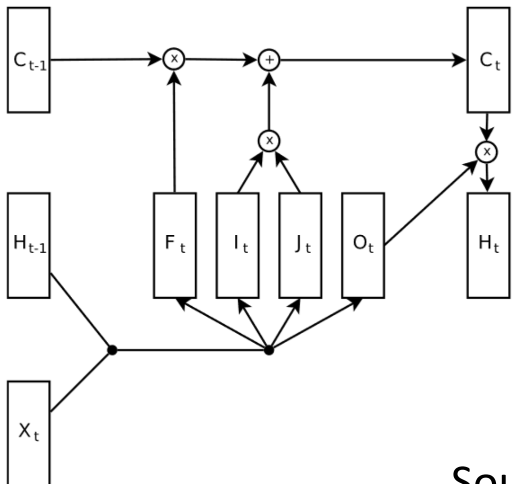
Long-Short Term Memory (LSTM)



Source: Bowman

- Popular for sequential input data (text, audio, …)
- Processes sequence one token at a time
- Maintains an internal state over time to capture long-term dependencies in the sequence

Long-Short Term Memory (LSTM)



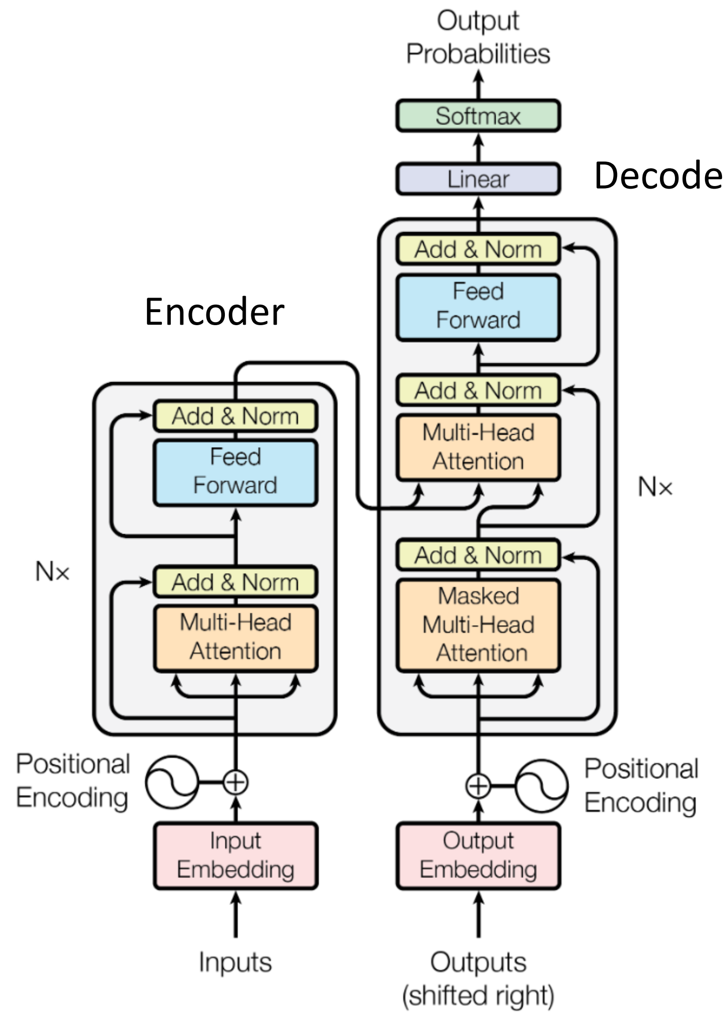
$$
\begin{aligned}
i_t &= \tanh(W_{\mathrm{xi}} x_t + W_{\mathrm{hi}} h_{t-1} + b_{\mathrm{i}}) \\
j_t &= \mathrm{sigm}(W_{\mathrm{xj}} x_t + W_{\mathrm{hj}} h_{t-1} + b_{\mathrm{j}}) \\
f_t &= \mathrm{sigm}(W_{\mathrm{xf}} x_t + W_{\mathrm{hf}} h_{t-1} + b_{\mathrm{f}}) \\
o_t &= \tanh(W_{\mathrm{xo}} x_t + W_{\mathrm{ho}} h_{t-1} + b_{\mathrm{o}}) \\
c_t &= c_{t-1} \odot f_t + i_t \odot j_t \\
h_t &= \tanh(c_t) \odot o_t
\end{aligned}
$$

Source: Jozefowicz

- Uses input-, output- and forget-gates to regulate the update of the cell- and hidden-state.
- Problem: To determine the RNN gradient one needs to "unroll" the prediction sequence -> slow to train

Transformer

- Core building block of recent language models (BERT, GPT-3, …)
- Takes in entire input sequence at once and uses attention mechanism to determine what is relevant
- Avoids vanishing gradient problem of RNN models and is also more efficient to train



Source: Vaswani

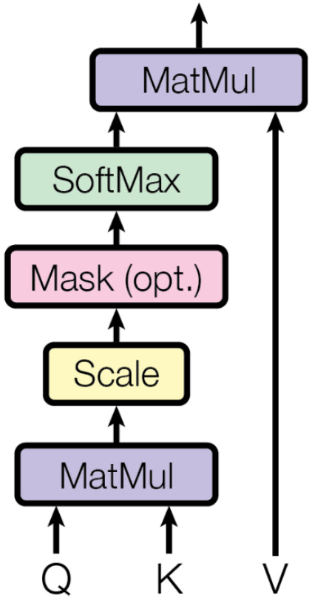
Transformer

- Attention mechanism

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

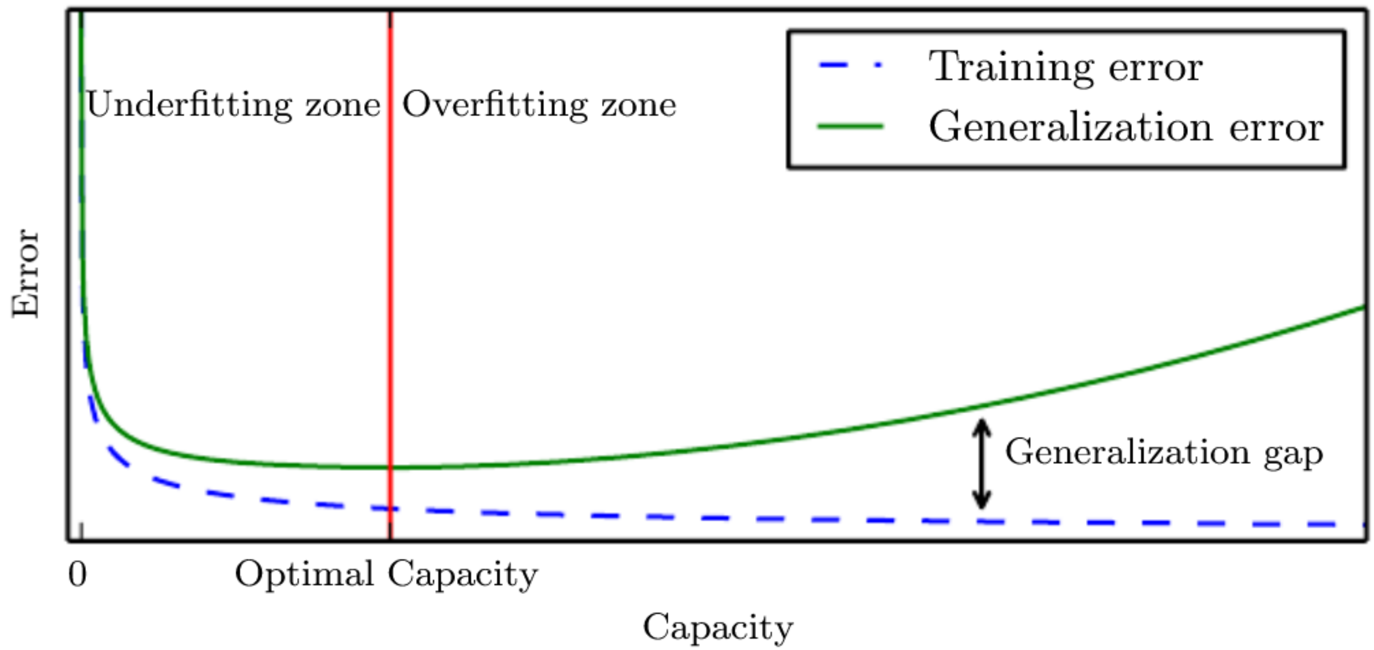
- Q : Query - $q_i = x_i W_Q$
- K : Key - $k_i = x_i W_K$
- V : Value - $v_i = x_i W_V$

Scaled Dot-Product Attention



Source: Vaswani

Overfitting



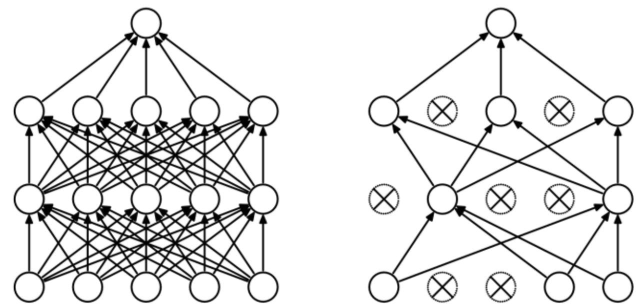
Source: Goodfellow

- Deep models are very expressive
- It is important to look out for overfitting (use validation data)

Overfitting

There are various ways to mitigate:

- Reduce capacity: Choose a smaller model which is less expressive
- Weight regularization: Introduce a penalty term that discourages weight vectors of large magnitude
- Early stopping: Stop optimization process early when validation error increases
- Dropout: Randomly remove connections during optimization



(a) Standard Neural Net (b) After applying dropout.

Source: Srivastava

Questions?