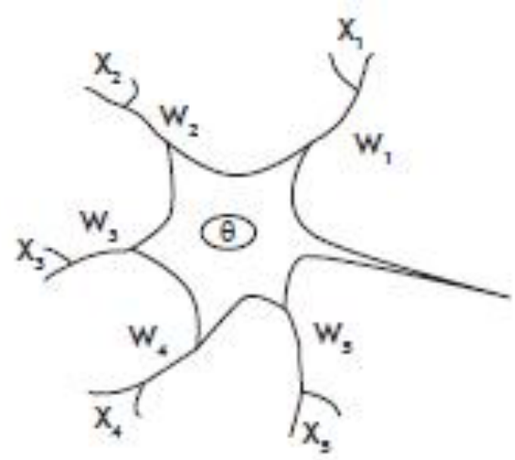
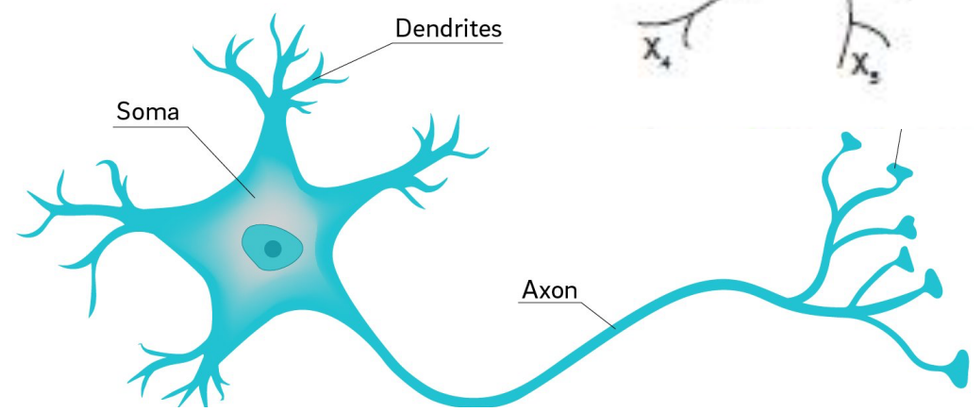
NEURAL NETWORKS

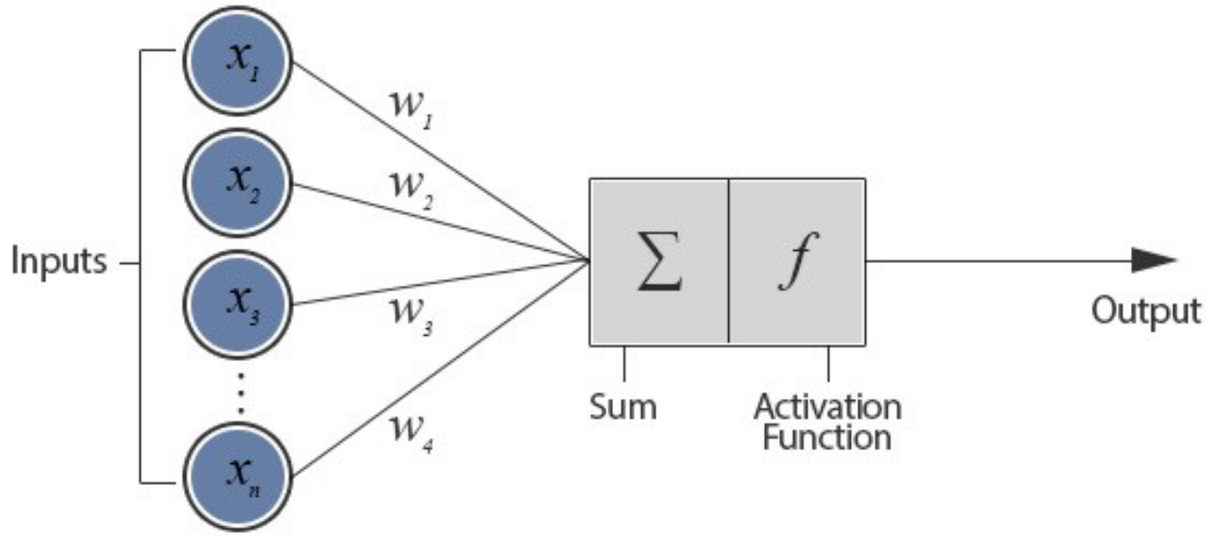
THE IDEA BEHIND ARTIFICIAL NEURONS

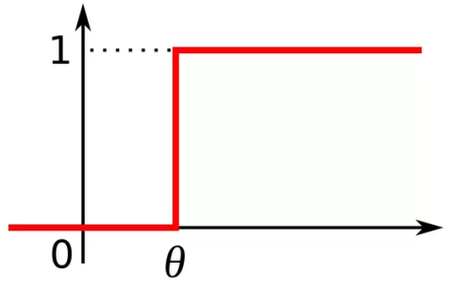
- Initially a simplified model of real neurons
- A real neuron has inputs from other neurons through synapses on its dendrites
- The inputs of a real neural are weighted! Due to the position of synapses (distance from the soma), and the properties of the dendrites
- A real neuron sums the inputs on its soma (voltages are summed)
- A real neuron has a threshold for firing: non-linear activation!



Neuron

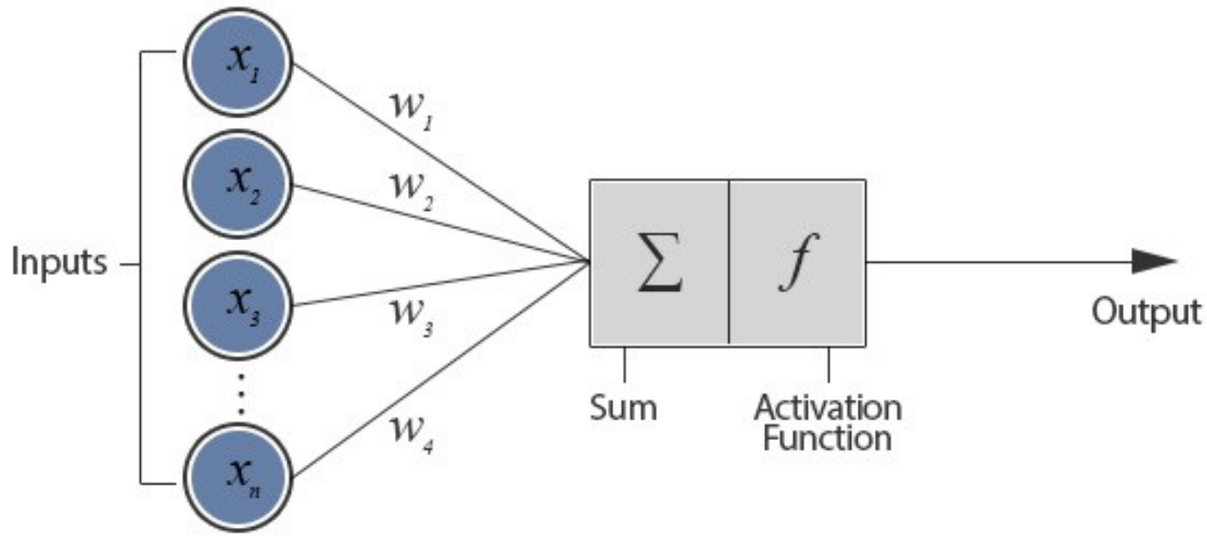






THE MATH BEHIND ARTIFICIAL NEURONS

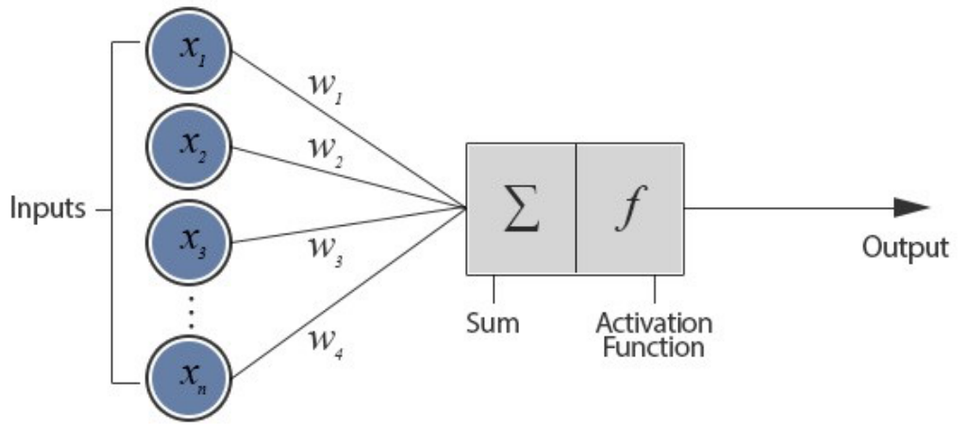
- One artificial neuron for classification is very similar to logistic regression
- One artificial neuron performs linear separation
- How does this become interesting?
  - SVM, kernel trick: project to high dimensional space where linear separation can solve the problem
  - Neurons: Follow the brain and use more neurons connected to each other: neural network!

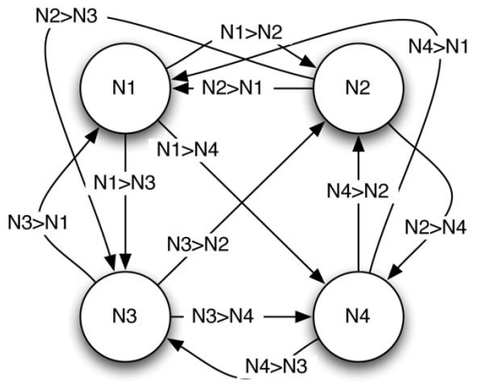


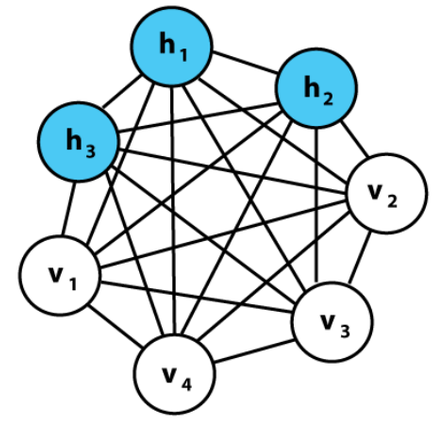
$$y = f\left(\sum_i w_i x_i + b\right)$$

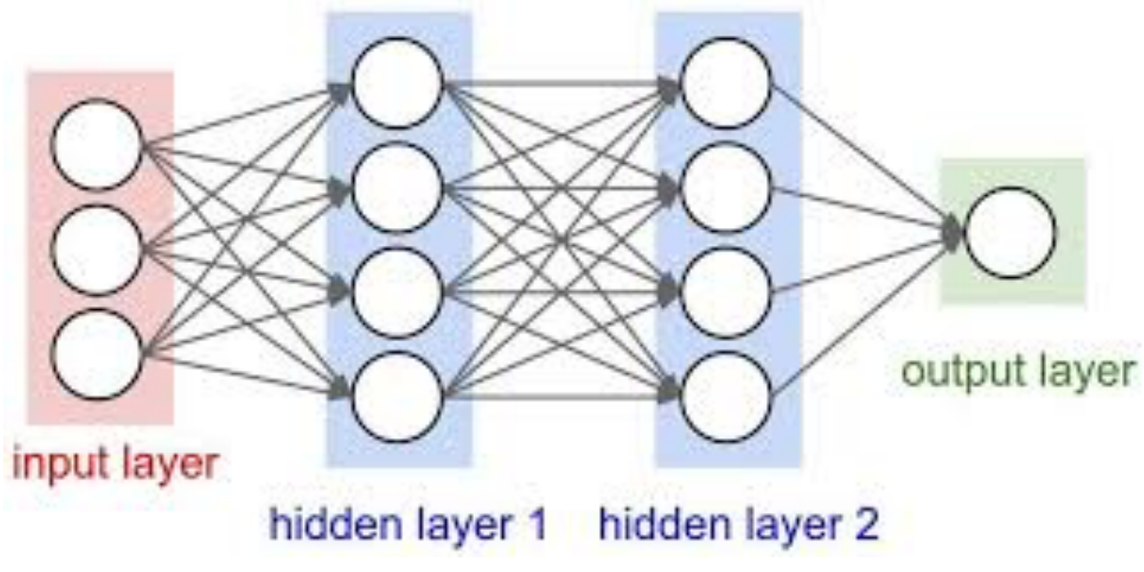
NEURAL NETWORKS

- Fully connected models, mostly of theoretical interest (Hopfield network, Boltzmann Machine)
- Supervised machine learning, function approximation: feed forward neural networks
  - Organise neurons into layers. The input of a neuron in a layer is the output of neuron from the previous layer
  - First layer is X, last is y



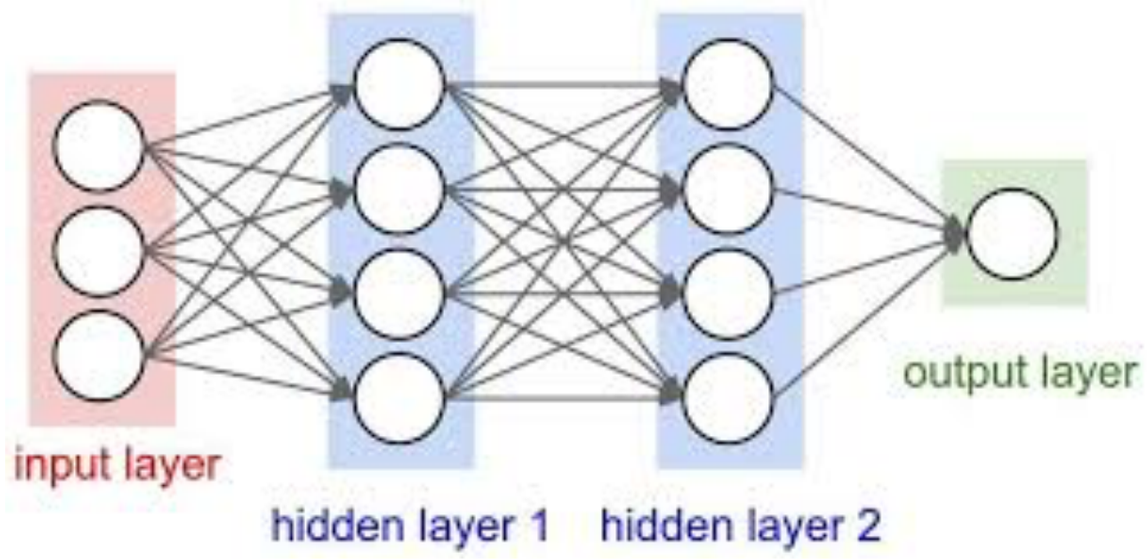


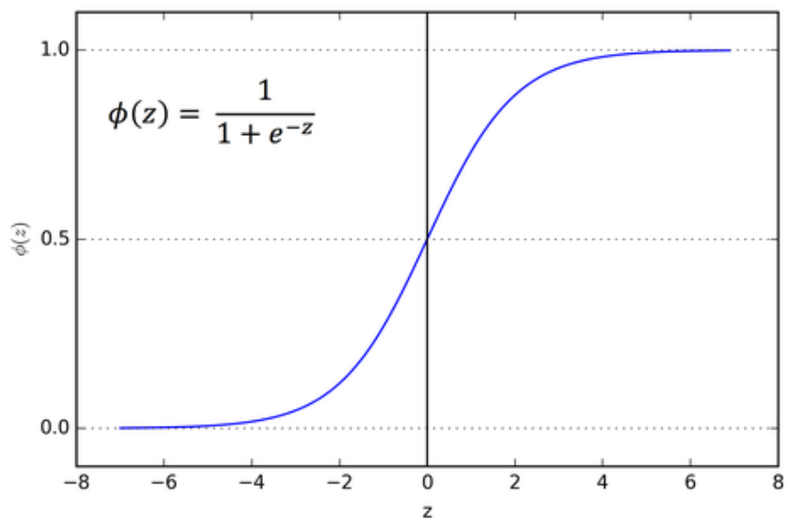




NEURAL NETWORKS

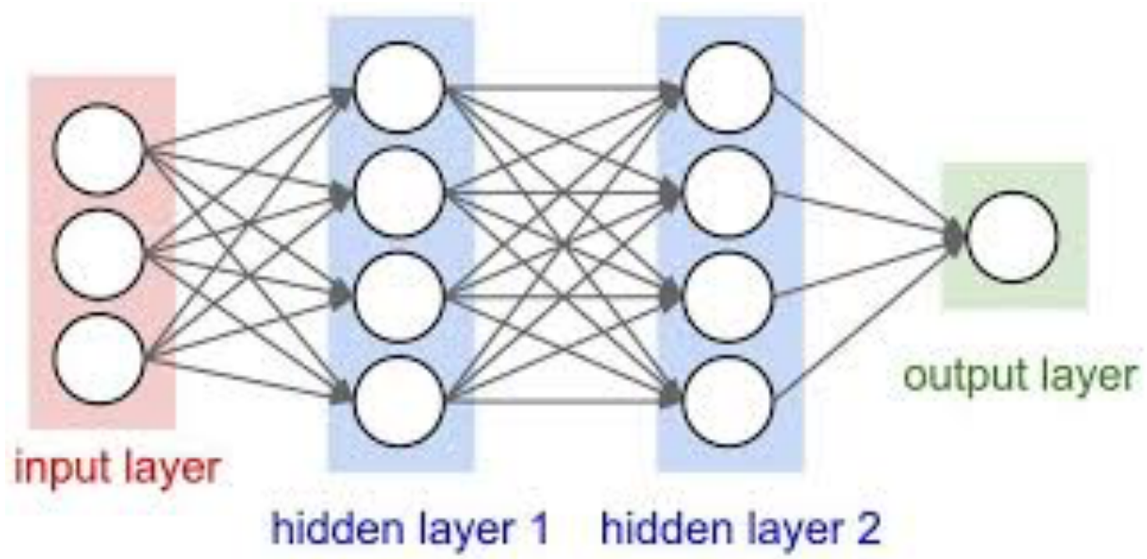
- Note: linear activations reduce the network to a linear model!
- Popular non-linear activations:
  - Sigmoid, tanh functions, ReLU
- A layer is a new representation of the data!
  - New space with #neuron dimensions
  - Iterative internal representations, in order to make the input data linearly separable by the very last layer!
- Slightly mysterious machinery!

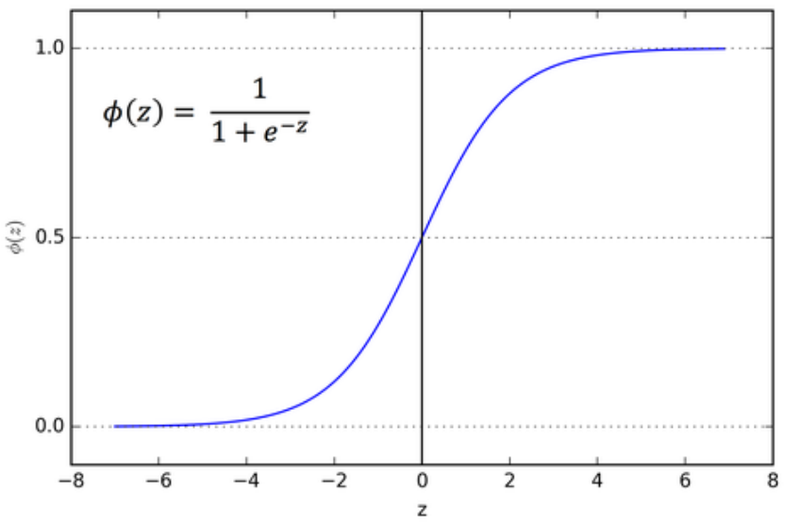




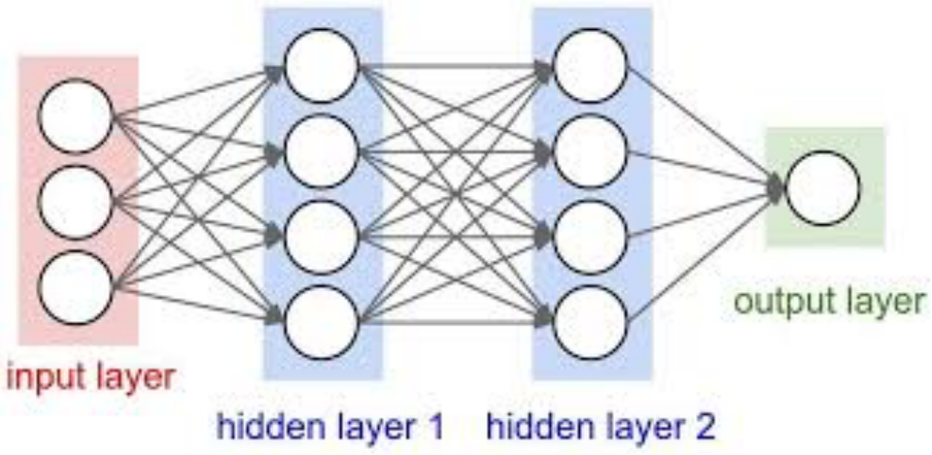
TRAINING NEURAL NETWORKS

- Loss functions just as before (MSE, Cross entropy)
  - L(y, y_pred)
- A neural network is a function composition
  - Input: x
  - Activations in first layer: f(x)
  - Activations in 2nd layer: g(f(x))
  - Etc: -> L(y, h(g(f(x))))
- NN is differentiable -> Gradient optimisation!
  - Loss function can be derived with respect to the weight parameters





TRAINING NEURAL NETWORKS



- Activations are known from a **forward pass**!
- Let's consider weights of neuron with index i in an arbitrary layer (j denotes the index of neurons in the previous layer)
- Derivation with respect to weights becomes derivation with respect to activations

$$o_i = K(s_i) = K(\sum w_{ij} o_j + b_i)$$

- For the last layer we are done, for previous ones, the loss function depends on an activation only through activations in the next layer. With the chain rule we get a recursive formula

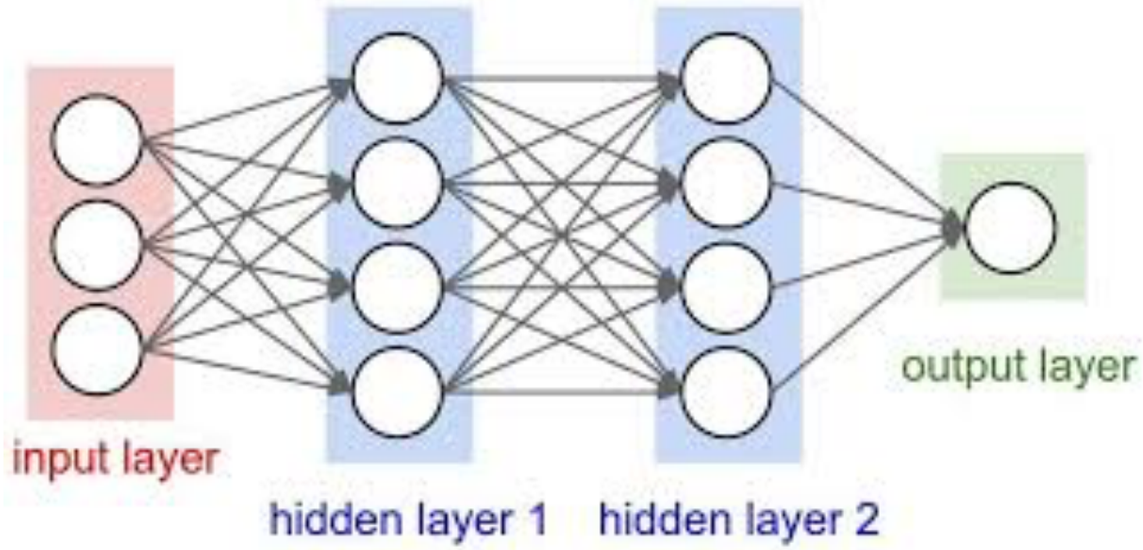
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial s_i}\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial E}{\partial o_i} K'(s_i)\, o_j$$

  - Last layer is given, previous layer can be calculated from the next layer, and so on!

$$\sum_{l \in R} \frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial o_i} = \sum_{l \in R} \frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial s_l}\frac{\partial s_l}{\partial o_i} = \sum_{l \in R} \frac{\partial E}{\partial o_l} K'(s_l) w_{li}$$

  - Local calculations: only need to keep track 2 values per neuron: activation, and a "diff"
  - **Backward pass**.

TRAINING NEURAL NETWORKS

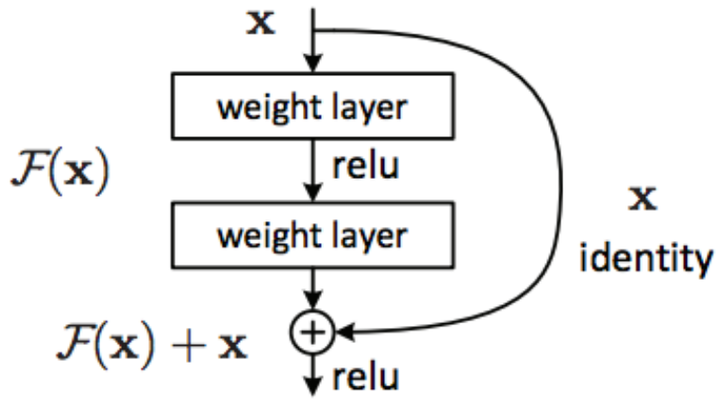


- Both forward and backwards passes are highly parallelizable
- GPU, TPU accelerators
- Backward connections do not allow the third line, no easy recursive formula
  - (Backprop through time for recurrent networks with sequence inputs)
- Skip connections are handled! E.g.: It's simply an identity neuron in a layer.

$$o_i = K(s_i) = K(\sum w_{ij} o_j + b_i)$$

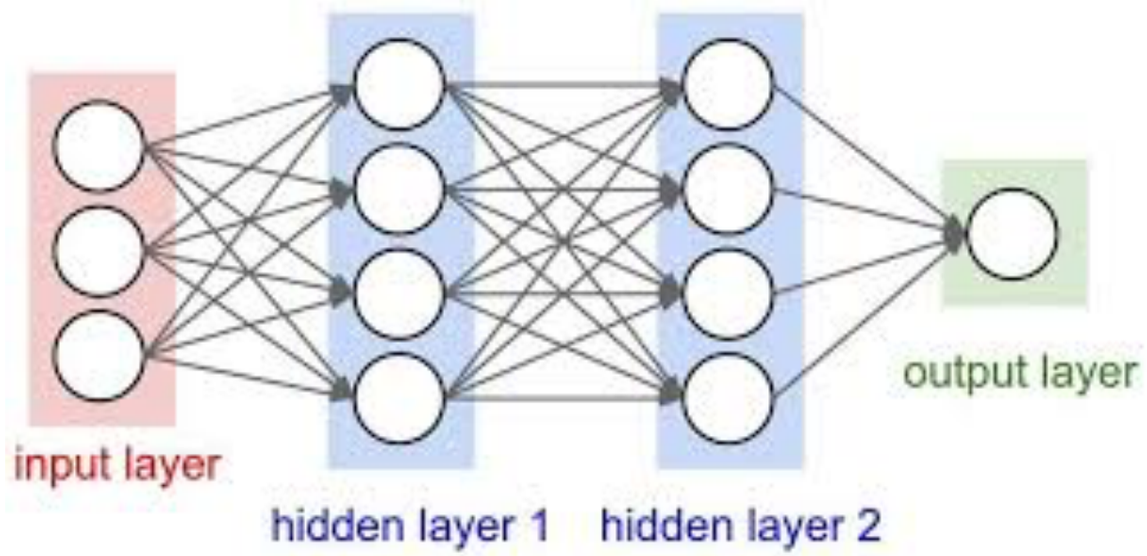
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial s_i}\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial E}{\partial o_i} K'(s_i)\, o_j$$

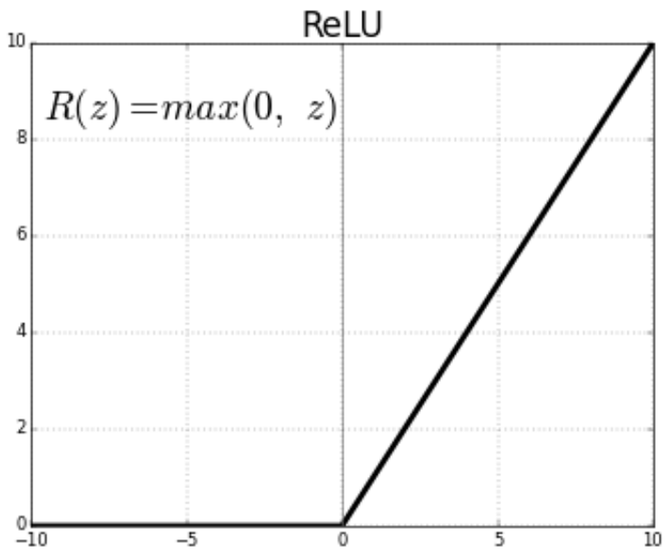
$$\sum_{l \in R} \frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial o_i} = \sum_{l \in R} \frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial s_l}\frac{\partial s_l}{\partial o_i} = \sum_{l \in R} \frac{\partial E}{\partial o_l} K'(s_l) w_{li}$$



TRAINING NEURAL NETWORKS

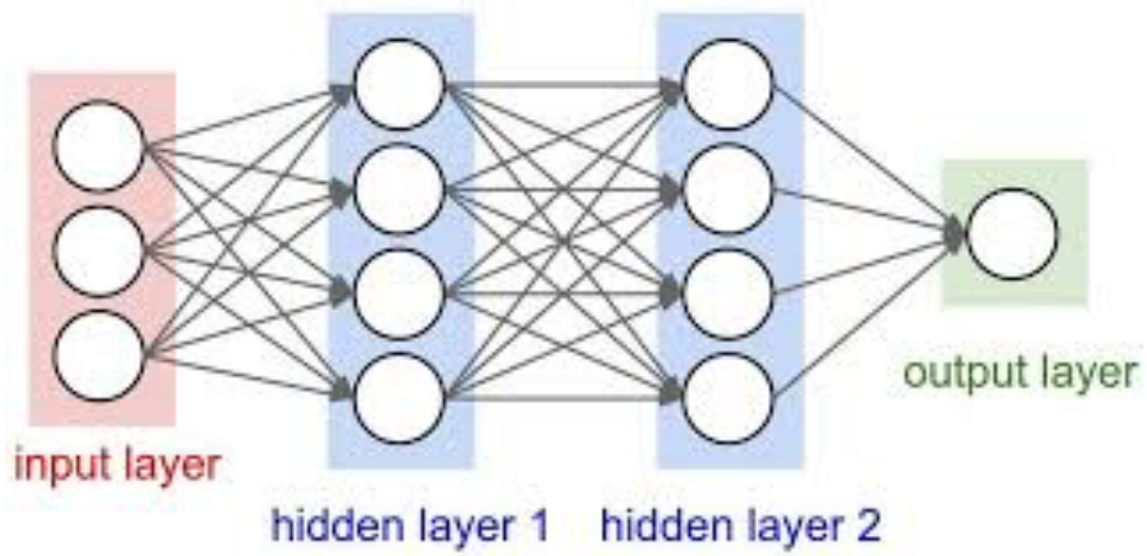
- Instead of full gradient, stochastic gradient (SGD): Gradient is only calculated from a few examples - a minibatch - at a time (1-512 samples usually)
- 1 full pass over the whole training dataset is called an epoch
- Stochasticity: order of data points. Shuffled in each epoch, to reach better solution.
- Note: use permutations of data, not random sampling, in order to use the whole dataset for learning in the best way!
- Note: online training, can easily handle **unlimited data**!





TRAINING NEURAL NETWORKS

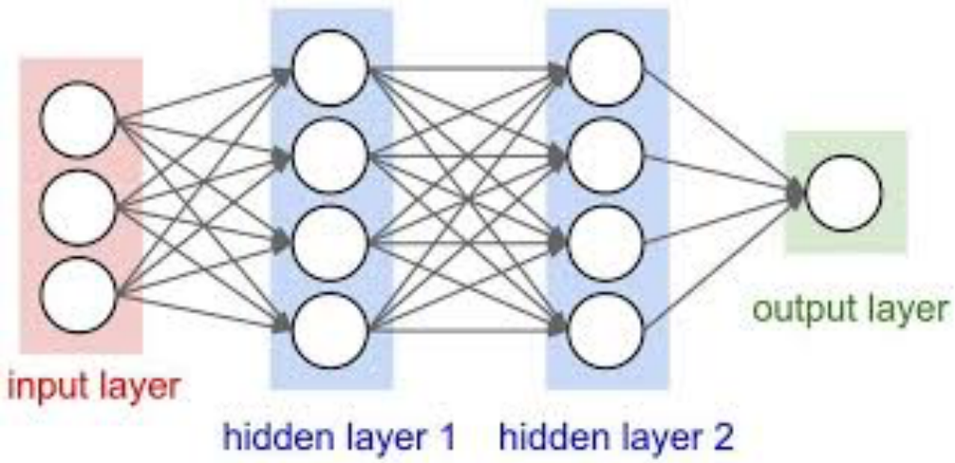
- How to chose initial parameters?
- Full 0? Each weight has the same, and not meaningful gradients. Random!
- Uniform or Gauss? Both Ok.
- Mean? 0

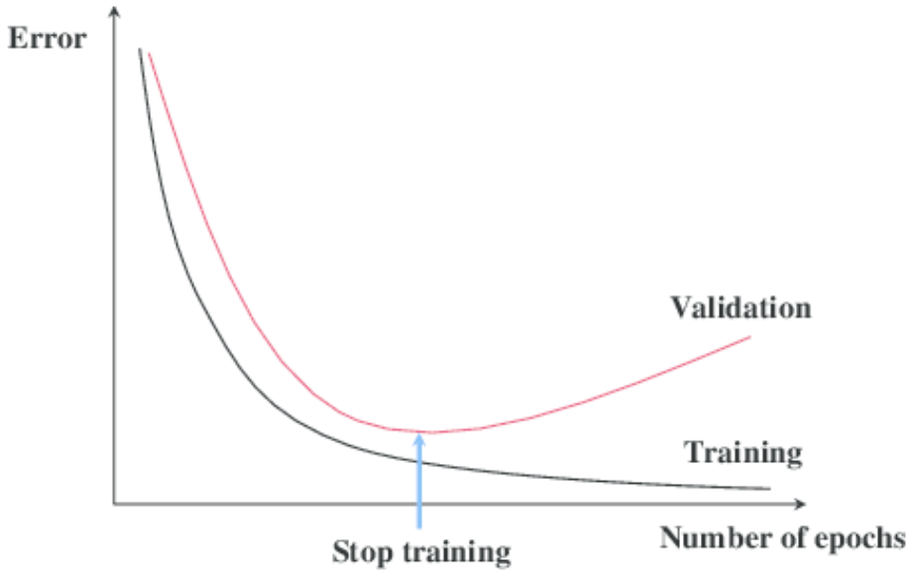


- Scale?
  - Avoid exploding passes, (forward backward too)
  - ReLU: grad(x) = x (if not 0)
  - variance: 2/(fan_in + fan_out)
- Even in 2014 they trained a 16 layer neural networks with layer-wise pre training, because of exploding gradients. Then they realised these simple schemes allow training from scratch!

REGULARISATION IN NEURAL NETWORKS, EARLY STOPPING

- Neural networks with many units and layers can easily memorise any data
- (modern image recognition networks can memorise 1.2 million, 224x224 pixel size, fully random noise images)
- L2 penalty of weights can be useful but still!
- How long should we train? “Convergence” is often 0 error on training data, fully memorised.
- Early stopping: Train-val-test splits, and stop training when error on validation does not improve. (Train-test only split will “overfit” the test data)!
- Early stopping is a regularisation! It does not improve training accuracy, but it does improve testing accuracy. It is essentially a limit, how far we can wander from the random initial parameter point.





REFERENCES

- ESL chapter 11.
- Deep learning book https://www.deeplearningbook.org