Very Brief Overview of Neural Nets

Johann Mitloehner, 2019

Perceptron: inputs x, output o, target y, weights w, and non-linear output function:

$$o(x) = \begin{cases} 1 & \text{if } x \cdot w > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

where $x \cdot w = \sum x_i w_i$ denotes the dot-product of the two vectors. Learning rule:

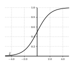
$$\Delta w_i = \eta \ (y - o) \ x_i$$

Problems with the Perceptron learning rule:

- convergence is only guaranteed if the classes can be separated by a linear hyperplane
- distance from that plane is not minimised

Single-Layer Feed-forward Net: non-linear output function, e.g., sigmoid:

$$f(z) = \frac{1}{1 + e^{-z}}$$



$$f'(z) = f(z)(1 - f(z))$$

Learning is minimisation of cost function; e.g. sum of squared errors of outputs o vs targets t:

$$o = f(x \cdot w), \quad E = \frac{1}{2} \sum_k (o_k - y_k)^2$$

Gradient Descent: weight update with learning rate η towards lower error

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta \sum_{k=1}^{N} (o_k - y_k) f'(x_k w) x_{k,i}$$

Two-Layer Feed-Forward Net:
weight matrices V from input to hidden, W from hidden to output

$$h(x) = f(Vx), \quad o(h) = f(Wh)$$

- Back-propagation of errors starting at output layer
- Given enough units in the hidden layer any function can be approximated to any error > 0

Recurrent Net:

- adds a memory state m_t which is updated in each input step t
- particularly suited for sequence processing, such as (short) sentences

$$m_t = f(Wx_t + Um_{t-1}), \quad o_t = f(Vm_t)$$

LSTM, GRU: Long-Short Term Memory, Gated Recurrent Units

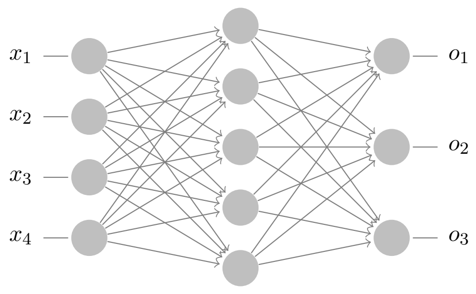
- several memory states, learn 'what to forget'
- deal with vanishing gradient problem in longer sequences

Convolutional Net: various architectures, especially successful in image processing

- *feature detectors*: apply same computation repeatedly over multiple input regions
- Pooling e.g. max pooling: reduce dimension by taking max value of regions

Two-Layer Softmax Classifier

Basic net architecture for classification tasks



Input layer $\mathbf{x}$ = application specific, e.g.

- 0/1 feature vector, such as words contained in document (bag of words)
- word or sentence embedding (word2vec, glove)

Matrices $\mathbf{V}, \mathbf{W}$ optimized by learning algorithm

Hidden layer $h_j(\mathbf{x}) = f(\sum_i x_i v_{ji})$

$f(z)$: non-linear, such as

Sigmoid: $\frac{1}{1+e^{-z}}$

Tanh: $\frac{e^z - e^{-z}}{e^z + e^{-z}}$

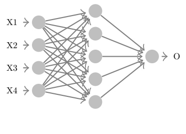
Relu: $max(0, z)$

Output layer $o_k(\mathbf{x}) = g(\sum_j h_j(\mathbf{x}) w_{kj})$

Softmax: $g_i(z) = \frac{e^{z_i}}{\sum_j e^{z_j}}$

Interpretation: probability of input $\mathbf{x}$ belonging to class i

Links:



http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html
https://github.com/Lab41/sunny-side-up/wiki/Deep-Learning-Techniques-for-Sentiment-Analysis
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/