Deep Learning basics

Deep Feedforward Networks

Learning goals

- High level understanding of feed forward networks,
- and the role and choices of activations

DEEP FEEDFORWARD NETWORKS

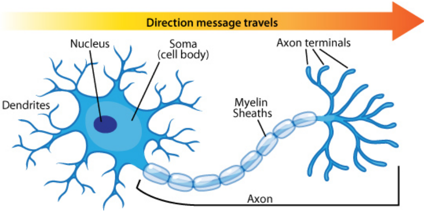
- Function approximation: find good mapping $\hat{\vec{y}} = f(\vec{x}; \vec{\theta})$ (or more exactly $f(\vec{x}; \hat{\vec{\theta}})$, but we omit the hat in future).
- *Network*: Composition of functions $f^{(1)}, f^{(2)}, f^{(3)}$ with multi-dimensional input and output
- Each $f^{(i)}$ represents one *layer* $f(\vec{x}) = f^{(1)}(f^{(2)}(f^{(3)}(\vec{x}))))$
- *Feedforward*:
  - Input $\rightarrow$ intermediate representation $\rightarrow$ output
  - No feedback connections
  - Cf. *recurrent* networks

DEEP FEEDFORWARD NETWORKS: TRAINING

- Loss function defined on output layer, e.g. $(y - f(\vec{x}; \vec{\theta}))^2$
- Quality criterion on other layers not directly defined.
- Training algorithm must decide how to use those layers most effectively (w.r.t. loss on output layer)
- Non-output layers can be viewed as providing a feature function $\phi(\vec{x})$ of the input, that is to be learned.

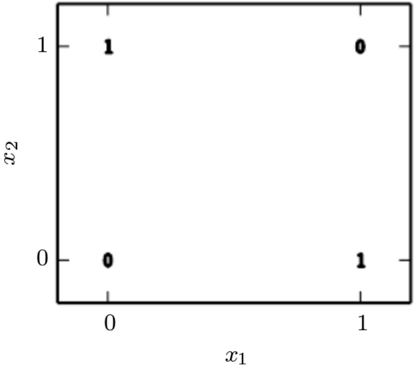
"NEURAL" NETWORKS

- Inspired by biological neurons (nerve cells)
- Neurons are connected to each other, and receive and send electrical pulses.
- *"If the [input] voltage changes by a large enough amount, an all-or-none electrochemical pulse called an action potential is generated, which travels rapidly along the cell's axon, and activates synaptic connections with other cells when it arrives."* (Wikipedia)



ACTIVATION FUNCTIONS WITH NON-LINEARITIES

- Linear Functions are limited in what they can express.
- Famous example: XOR
- Simple layered non-linear functions can represent XOR.



DESIGN CHOICES FOR OUTPUT UNITS

- Can typically be interpreted as probabilities.
    - Logistic sigmoid
    - Softmax
    - mean and variance of a Gaussian, ...
- Trained with negative log-likelihood.

SOFTMAX

- Logistic sigmoid
  - Vector $\vec{y}$ of binary outcomes, with no contraints on how many can be 1.
  - Bernoulli distribution.
- Softmax
  - Exactly one element of $\vec{y}$ is 1.
  - Multinoulli (categorical) distribution.

$$p(Y = i|\vec{\phi}(\vec{x}))$$

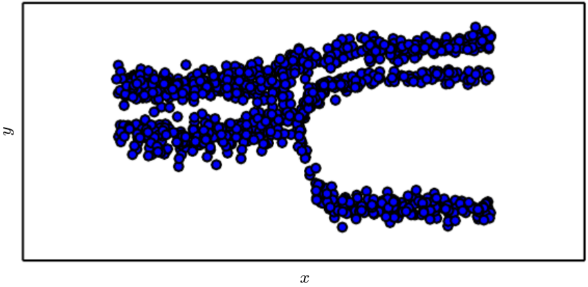
$$\sum_i p(Y = i|\vec{\phi}(\vec{x})) = 1$$

$$softmax(\vec{z})_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

PARAMETRIZING A GAUSSIAN DISTRIBUTION

- Use final layer to predict parameters of Gaussian mixture model.
- Weight of mixture component: softmax.
- Means: no non-linearity.
- Precisions ($\frac{1}{\sigma^2}$) need to be positive: softplus

$$softplus(z) = ln(1 + exp(z))$$

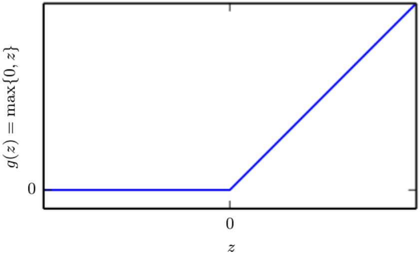


DESIGN CHOICES FOR HIDDEN UNITS

- Rectified Linear Unit:

$$relu(z) = max(0, z)$$

$$z = \vec{x}^T \vec{w} + b$$



- Consistent gradient of 1 when unit is *active* (i.e. if there is an error to propagate).
- Default choice for hidden units.

A SIMPLE RELU NETWORK TO SOLVE XOR

$$f(\vec{x}; \vec{W}, \vec{c}, \vec{w}) = \vec{w}^T max(0, \vec{W}^T \vec{x} + \vec{c})$$

$$\vec{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\vec{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\vec{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

OTHER CHOICES FOR HIDDEN UNITS

- A good activation function aids learning, and provides large gradients.
- Sigmoidal functions (logistic sigmoid)
  - have only a small region before they flatten out in either direction.
  - Practice shows that this seems to be ok in conjunction with Log-loss objective.
  - But they don't work as well as hidden units.
  - ReLU are better alternative since gradient stays constant.
- Other hidden unit functions:
  - maxout: take maximum of several values in previous layer.
  - purely linear: can serve as low-rank approximation.