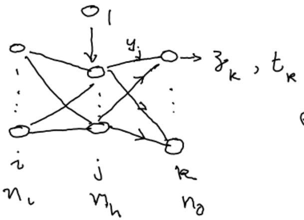
Backpropagation derivation



$$e(w) = \frac{1}{2}\sum_{k=1}^{n_o} (t_k - z_k)^2$$

Error: $e(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{n_o}(t_k - z_k)^2$ assuming square error.

General update: $\Delta\mathbf{w} = -\eta\nabla\mathbf{e}$

Specific update for w_{kj}: $\Delta w_{kj} = -\eta\frac{\partial e}{\partial w_{kj}}$

Feed forward equations:

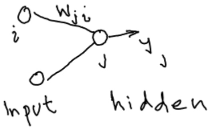
$$net_k = \sum_{j=1}^{n_h} w_{kj} y_j$$

$$z_k = f(net_k)$$ f is the activation function.

$$
\begin{aligned}
\frac{\partial e}{\partial w_{kj}} &= \frac{\partial e}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} \\
&= -(t_k - z_k) f'(net_k) \frac{\partial net_k}{\partial w_{kj}} \\
-\frac{\partial e}{\partial w_{kj}} &= \delta_k y_j, \qquad \text{where } \delta_k = (t_k - z_k) f'(net_k) \\
\Delta w_{kj} &= -\eta \frac{\partial e}{\partial w_{kj}} \\
&= \eta \delta_k y_j \\
&= \eta (t_k - z_k) f'(net_k) y_j
\end{aligned}
$$

Assumes f is differentiable.

Hidden layer backpropagation



$$\Delta w_{ji} = -\eta \frac{\partial e}{\partial w_{ji}}$$

$$\frac{\partial e}{\partial w_{ji}} = \frac{\partial e}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_i \qquad \text{since } net_j = \sum_{i=0}^{n_i} w_{ji} x_i$$

$$\frac{\partial y_j}{\partial net_j} = f'(net_j) \qquad \text{since } y_j = f(net_j)$$

$$
\begin{aligned}
\frac{\partial e}{\partial y_j} &= \frac{\partial}{\partial y_j}[\frac{1}{2}\sum_{k=1}^{n_o}(t_k - z_k)^2] \\
&= -\sum_{k=1}^{n_o}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{n_o}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{n_o}(t_k - z_k)f'(net_k)w_{kj} \\
&= -\sum_{k=1}^{n_o}\delta_k w_{kj} \\
\Delta w_{ji} &= \eta f'(net_j)x_i\sum_{k=1}^{n_o}\delta_k w_{kj} \\
&= \eta\delta_j x_i \qquad \text{where } \delta_j = f'(net_j)\sum_{k=1}^{n_o}\delta_k w_{kj}
\end{aligned}
$$

Stochastic gradient descent alg.

Algorithm 0.1: $\textsc{sgd}(nw, \eta, stopCriterion)$

Init **w** randomly
repeat

$$\begin{cases} (\mathbf{x_m}, \mathbf{t_m}) \leftarrow \text{choose randomly from } \mathcal{L} \\ \mathcal{L} \leftarrow \mathcal{L} - (\mathbf{x_m}, \mathbf{t_m}) \\ \text{Feed forward} \\ \textbf{comment: } y_j\text{s and } z_k\text{s are now available.} \\ w_{kj} \leftarrow w_{kj} + \eta\delta_k y_j \\ w_{ji} \leftarrow w_{ji} + \eta\delta_j \mathbf{x_m}_i \end{cases}$$

until *stopCriterion*
return $(\mathbf{w})$

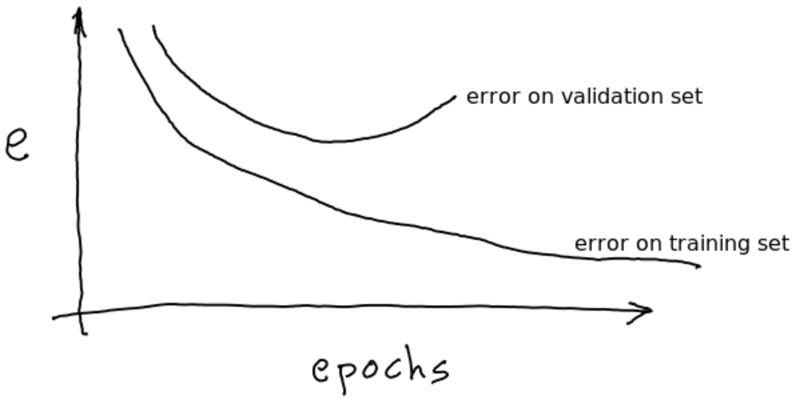
Stop criterion: a) error on a validation set b) limit θ on ∇e c) no. of epochs. An epoch is one exposure of every element in $\mathcal{L}$.

Batch alg.

Algorithm 0.2: $\text{BGD}(nw, \eta, stopCriterion)$

Init $\mathbf{w}$ randomly
repeat
$$\begin{cases} epoch, \Delta w_{ij}, \Delta w_{kj} \leftarrow 0 \\ \textbf{comment: } \text{Update being batched} \\ \textbf{repeat} \\ \begin{cases} (\mathbf{x_m}, \mathbf{t_m}) \leftarrow \text{choose randomly from } \mathcal{L} \\ \mathcal{L} \leftarrow \mathcal{L} - (\mathbf{x_m}, \mathbf{t_m}) \\ \text{Feed forward } \textbf{comment: } y_j\text{s and } z_k\text{s are now available.} \\ \Delta w_{kj} \leftarrow \Delta w_{kj} + \eta \delta_k y_j \\ \Delta w_{ji} \leftarrow \Delta w_{ji} + \eta \delta_j \mathbf{x_m}_i \end{cases} \\ \textbf{until } \mathcal{L} \text{ is empty} \\ \textbf{comment: } \text{Apply batched update} \\ w_{kj} \leftarrow w_{kj} + \Delta w_{kj} / |\mathcal{L}| \\ w_{ji} \leftarrow w_{ji} + \Delta w_{ji} / |\mathcal{L}| \end{cases}$$
until *stopCriterion*
return $(\mathbf{w})$

Behaviour of error



Stochastic vs Batch

- Stochastic:
  - Usually faster than batch.
  - Higher probability of reaching better minimum.
  - Useful in tracking changes.
- Batch:
  - Convergence is well understood.
  - Many second order techniques to speed up convergence - but computationally expensive.
  - Theoretical analysis of convergence and dynamics is simpler.

In practice one often uses mini-batches. Often, the mini-batch size increases as epochs increase.

Activation functions

- Activation function: non-linear, smooth, continuous, saturating. Typical functions are tanh, sigmoid, relu (or approximator).
- $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $\sigma(x) = \frac{1}{1+e^{-x}}$
- $relu(x) = max(0, x)$, smooth approx. $softplus(x) = ln(1 + e^x)$

tanh

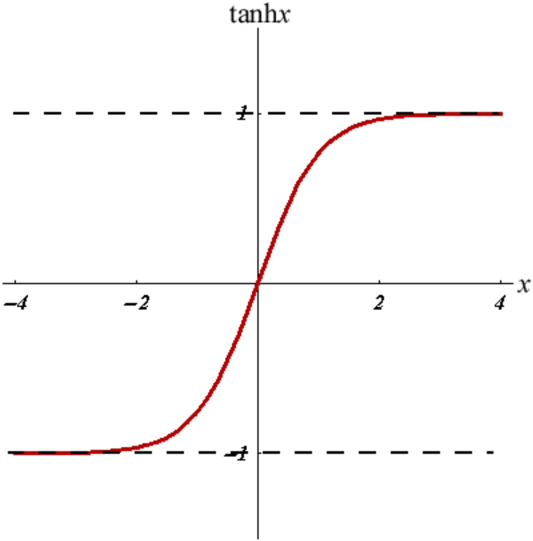


Figure: tanh - From: efunda.com

Sigmoid or logistic fn.

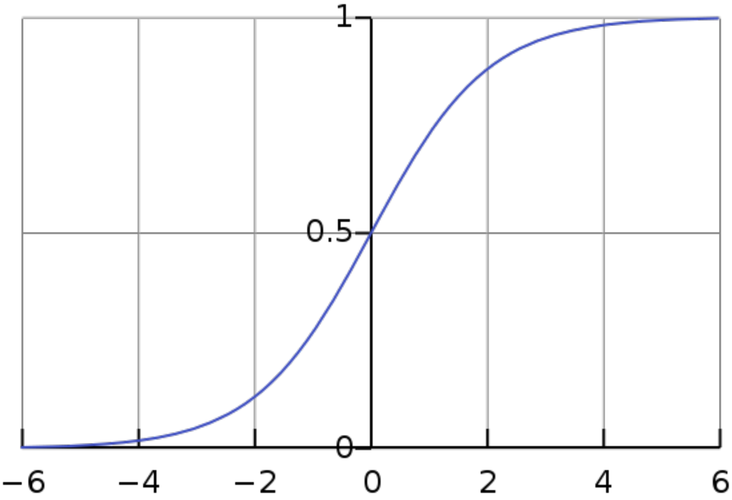


Figure: sigmoid - From: wikipedia

Relu and approx.

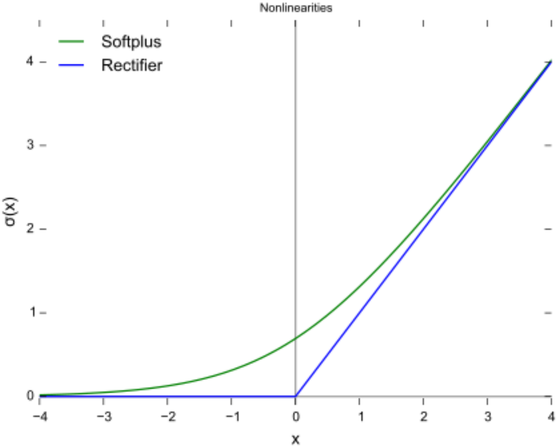


Figure: A Relu function and its smooth approx. From: wikipedia

Practical issues[1]

- Weights cannot be 0. Initialize to values such that activation function is in the linear region. One recommendation is to choose weights from a distribution with mean 0 and $\sigma_w = \frac{1}{\sqrt{fan-in-to-node}}$.
- The values of the input vector are normalized such that mean is 0 and variance is 1.
- Preprocess to remove correlated attributes (e.g. PCA).
- Presentation: order examples such that information is maximum.
  - Successive training examples from different classes.
  - Order examples such that successive examples produce large error more frequently. Danger: outliers can cause problems.

[1] Mueller, et al., Neural n/ws: tricks of the trade, Springer, 2012

Practical issues - contd.

- Learning rate η: ideally, it should be chosen such that all weights in the network converge at the same rate.
  - If possible, give each weight or group of weights its own η.
  - Learning rate should be proportional to the fan-in.
  - Weights in earlier layers should typically be larger than later ones.
  - Learning rates can also be made adaptive if the gradient is remembered.