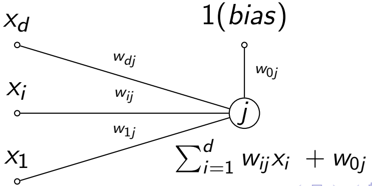
Neural networks

- Neural networks are non-linear statistical models (known in statistics as project pursuit regression).
- The basic neural network consists of 3 layers of units (usually called neurons) with interconnections between successive layers. There are no connections between units in the same layer or between the first and third layer. The three layers are called 1-input, 2-hidden, 3-output. See the figure later for details.
- Each connection has a weight w_{ij} and the input to a unit, say j, is a linear combination $\sum_{i \in (\text{units connected to j})} w_{ij} x_i \; + w_{0j}$, where x_i is the output of unit i and w_{0j} is a bias term (like a threshold).

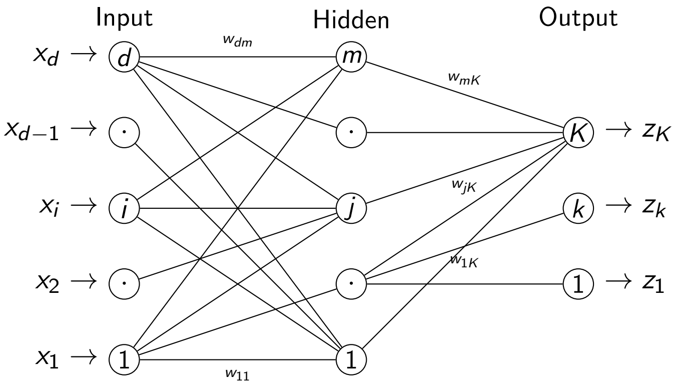


- The output from a unit j is a non-linear function f_j (called the activation function) of the input. That is output of unit j, y_j is:

$$y_j = f_j(net_j) = f_j(\sum_{i=1}^{d} w_{ij}x_i \ + w_{0j})$$

  Notationally, we represent the net input to unit j as $net_j = \sum_{i=0}^{d} w_{ij}x_i$, where $x_0 = 1$ for the bias and the output of j is $y_j = f_j(net_j)$.
- Typical activation functions are: sigmoid ($\frac{1}{1+e^{-x}}$), tanh ($\frac{e^x - e^{-x}}{e^x + e^{-x}}$), Gaussian (not as common).

Diagram of neural network with one hidden layer



Note: Each neuron in a layer is connected to all neurons in the next layer. This is not shown to avoid clutter. Above diagram will be used in the derivation later.

Feedforward operation

- In feed forward operation the output of neuron j is: $o_j = f_j(\sum_{i \in \text{(all inputs to j)}} w_{ij} o_i + w_{0j})$ where o_i is the output of neuron i in the previous layer which feeds into j, f_j is the activation function of neuron j.
- The outputs z_1 to z_K are combined/interpreted suitably to either get a class label or to get a regressed value. For regression the output layer normally has just one neuron.
- `Notation:` We use y_j for the output of neuron j in the hidden layer and z_k for the output of neuron k in the output layer. Let number of neurons in the input layer be d, in hidden layer m and output layer K. Also, subscripts i, j and k will be used for the input, hidden and output layers respectively.

Learning via backpropagation

- Learning in a 3NN (one hidden layer) happens by adjusting the weights using a squared loss function and by gradient descent as input patterns are fed one-by-one to the network.
- In online learning updates are done after each vector is exposed. In the batch version the update is done after all the input patterns have been fed to the network and individual vector updates are batched . The online version is much more commonly used.
- This happens layer by layer starting with the weights between the hidden layer and the output layer with the error being *backpropagated* to adjust the next layer of weights.
- At the start all weights are randomly initialized to small values (say between -1 to 1 avoiding 0).

- For the output layer define the loss function as:

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}\sum_{k=0}^{K}(t_k - z_k)^2$$

  where t_k is the desired output from the neuron k in the output layer and z_k is the actual output after a feedforward step. The gradient descent weight update is: $\Delta\mathbf{w} = -\eta\frac{\partial\mathcal{E}}{\partial\mathbf{w}}$, where η is the learning parameter.
- Define $net_k = \sum_{j=0}^{m} w_{jk}y_j$ where m is all the units in the previous (hidden) layer connected to unit k. Then $z_k = f(net_k)$ and $\mathcal{E}(\mathbf{w}) = \frac{1}{2}\sum_{k=0}^{K}(t_k - f_k(net_k))^2$. For a single weight w_{jk} we have $\frac{\partial\mathcal{E}}{\partial w_{jk}} = \frac{\partial\mathcal{E}}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}}$.

- $\frac{\partial \mathcal{E}}{\partial net_k} = -(t_k - z_k) f_k'(net_k)$ and $\frac{\partial net_k}{\partial w_{jk}} = y_j$. So, update for w_{jk} is:

  $$\Delta w_{jk} = \eta (t_k - z_k) f_k'(net_k) y_j$$

  Assuming f_k is differentiable we can get f' and consequently calculate Δw_{jk}.
- To find the update $\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}}{\partial w_{ij}}$ for the first layer of weights the calculation is more involved. We now want to find $\frac{\partial \mathcal{E}}{\partial w_{ij}}$. For neuron j in the hidden layer we have $y_j = f_j(net_j)$ and $net_j = \sum_{i=0}^{d} w_{ij} x_i$. Also, by chain rule:

  $$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$
- $\frac{\partial net_j}{\partial w_{ij}} = x_i$ and $\frac{\partial y_j}{\partial net_j} = f_j'(net_j)$

$$
\begin{aligned}
\frac{\partial \mathcal{E}}{\partial y_j} &= \frac{\partial[\frac{1}{2}\sum_{k=0}^{K}(t_k - z_k)^2]}{\partial y_j} \\
&= -\sum_{k=0}^{K}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=0}^{K}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=0}^{K}(t_k - z_k)f_k'(net_k)w_{jk}
\end{aligned}
$$

So,

$$\Delta w_{ij} = \eta f_j'(net_j)x_i \sum_{k=0}^{K}(t_k - z_k)f_k'(net_k)w_{jk}$$

Training algorithm for 3NN

```
Algorithm 0.10: $\textsc{Train3NN}(\mathbf{w}_i, \mathbf{w}_j, \eta, stopFn)$

comment: Online version. $\mathbf{w}_i, \mathbf{w}_j$ are first and second layer of weights;
$\mathbf{w}_i, \mathbf{w}_j \leftarrow$ initialize randomly;
while (stopFn is false);
  do {
    while ($\mathcal{L} \neq \emptyset$)
      do {
        $\mathbf{x} \leftarrow$ choose randomly from $\mathcal{L}$ without replacemnt;
        Run network in feedforward mode;
        $\mathbf{w}_j \leftarrow$ update second layer of weights using update eqn;
        $\mathbf{w}_i \leftarrow$ update first layer of weights using update eqn;
return $(\mathbf{w}_i, \mathbf{w}_j)$;
```

Stopping criteria

- Use a validation set. Typical error behaviour is shown below:

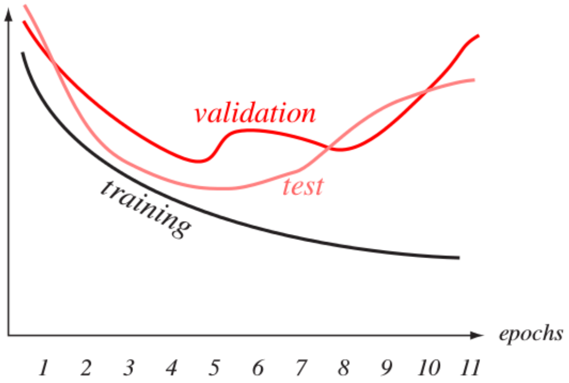


Figure: Error versus epochs

- Use a threshold for $\mathcal{E}$.