# Neural Networks

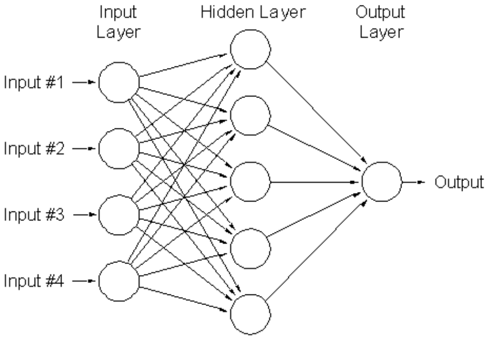
Charlotte Wickham

April 27, 2007

# Neural Networks

Simplest case

- Have p inputs $\boldsymbol{x}$
- Have one hidden layer with each unit being a function ($\phi_j$) of a linear combination of the inputs.
- Each output is a function ($\phi_0$) of a linear combination of the hidden units.



# Simplest Case continued

- For classification we have $K$ outputs predicting the probability of belonging to class $k$.
- For predicting a single continuous $Y$ we only need one output.
- $\phi_j$ is known as the activation function and is often taken to be

$$\phi_j(v) = \frac{e^v}{1 + e^v}$$

- The output functions $\phi_0$ are often taken as the identity ($\phi_0(t) = t$) for regression, and softmax for K class classification ($\phi_0(t_k) = e^{t_k} / \sum_k e^{t_k}$).

# General Case

- Allow more than one layer and connections that skip layers.
- Also have units with value 1 that feed into all hidden layer and output units to account for constant terms.
- Then model is:

$$y_k = \phi_0 \left( \sum_{i \to k} w_{ik} x_i + \sum_{j \to k} w_{jk} \phi_j \left( \sum_{i \to j} w_{ij} x_i \right) \right)$$

- Completely parametrized by weight vector $w_{ij}$.
- Can be viewed as a very flexible function of the inputs.

## Finding the weights

- Consider training points $(x_p, t_p)$ and the let the output of the neural network be $y = f(x; w)$. We want to minimize

$$E(w) = \sum_p ||t_p - f(x_p; w)||^2$$

  or for K class classification (input now $(x_p, t_{p1}, \ldots, t_{pk})$, output $y_k = f_k(x; w)$ )

$$E(w) = -\sum_p \sum_k t_{pk} log(f_k(x_p; w)).$$

- Then just a minimization problem. Many algorithms designed to do this. All iterative.
- Need starting points.
- Need stopping rule.

- Starting weights
  Generally use random points near zero. $w_{ij} \sim$ Uniform on $[-0.7, 0.7]$ is common.
- Stopping rule
  Want to avoid overfitting.
  - Early methods stopped minimization early.
  - Other option is regularization. Minimize:

    $$E(w) = \sum_p ||t_p - f(x_p; w)||^2 + \lambda \sum_{ij} w_{ij}$$

    instead.
  - Effect is to shrink weights towards zero.
  - Choose $\lambda$ using cross validation.
  - Also has the advantage that if you use too many hidden units their weights should be shrunk to zero.

# In practice

- Scale all variables to have mean 0 variance 1 - to ensure input treated equally in regularization.
- Choose decay parameter by cross validation.
- Number of hidden units generally doesn't matter as long as it is big enough.
- Can be sensitive to initial conditions. Can average a few instances.

# Supernova data

```
      size decay      fit
 [1,]    5 0.010 4.333333
 [2,]    5 0.050 4.377778
 [3,]    5 0.001 4.600000
 [4,]   10 0.010 4.133333
 [5,]   10 0.050 3.977778
 [6,]   10 0.001 4.411111
 [7,]   15 0.010 4.366667
 [8,]   15 0.050 4.322222
 [9,]   15 0.001 4.388889
[10,]   20 0.010 4.422222
[11,]   20 0.050 4.433333
[12,]   20 0.001 4.511111
```

Fit neural net with 10 hidden units and decay 0.05. Repeat five times and average result.

# Supernova data

- Disagreement between repetitions

| | | NN 1 | |
|---|---|---|---|
| | | Other | Supernova |
| NN 2 | Other | 495 | 15 |
| | Supernova | 10 | 480 |

2.5% disagreement

- Prediction error

| | | Prediction | |
|---|---|---|---|
| | | Other | Supernova |
| Actual | Other | 477 | 23 |
| | Supernova | 28 | 472 |

5.1% error