Linear models: linear regression

- Work most naturally with numeric attributes
- Standard technique for numeric prediction
  - Outcome is linear combination of attributes

$$x = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

- Weights are calculated from the training examples $\mathbf{a}^{(i)}$
- Predicted value for first training instance $\mathbf{a}^{(1)}$

$$w_0 a_0^{(1)} + w_1 a_1^{(1)} + w_2 a_2^{(1)} + \ldots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)}$$

(assuming each instance is extended with a constant attribute with value 1)

Minimizing the squared error

- Choose k +1 coefficients to minimize the squared error on the training data
- Squared error: $\sum_{i=1}^{n} \left(x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)}\right)^2$
-
- Derive coefficients using standard matrix operations by setting partial derivatives =0
- Can be done if there are more instances than attributes (roughly speaking) – involves matrix inversion
- Minimizing the *absolute error* is more difficult

Classification

- *Any* regression technique can be used for classification
  - Training: perform a regression for each class, setting the output to 1 for training instances that belong to class, and 0 for those that don't
  - Prediction: predict class corresponding to model with largest output value (*membership value*)
- For linear regression this is known as *multi-response linear regression*
- Problem: membership values are not in [0,1] range, so aren't proper probability estimates

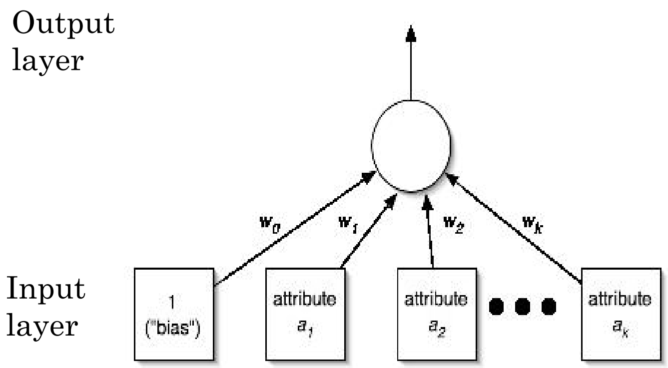
Gradient Descent

- Minimize a multivariate function f(w), where w is a k-dimensional vector.
  - Start with random values of w.
  - Apply the gradient descent rule until error is below a certain threshold:

$$w = w - \lambda \nabla f(w)$$

  - where λ is the learning rate

Perceptron as a neural network



$$out(a) = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

Learning Algorithm

1. Randomly initialize $w_1, w_2, \ldots, w_k$
2. for each instance $a^{(i)}$, do
   - Compute error $E_i = x_i - out(a^{(i)})$
3. For l=1 to k do
   - Update weight $w_l = w_l + \lambda \sum E_i a_l^{(i)}$
4. If $\sum (E_i)^2$ is small, stop; otherwise go back to Step 2.

Linear models: the perceptron

- Different approach: learn separating hyperplane
- Assumption: data is *linearly separable*
- Algorithm for learning separating hyperplane: *perceptron learning rule*
- Hyperplane: $0 = w_0 a_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$ where we again assume that there is a constant attribute with value 1 (*bias*)
- If sum is greater than zero we predict the first class, otherwise the second class

The algorithm

```
Set all weights to zero
Until all instances in the training data are classified correctly
  For each instance I in the training data
    If I is classified incorrectly by the perceptron
      If I belongs to the first class add it to the weight vector
      else subtract it from the weight vector
```

- Why does this work?
- If $(a^{(i)}, x_i)$ is correctly classified, don't change
- If wrongly classified as -1, then $w = w + a^{(i)}$
- If wrongly classified as +1, then $w = w - a^{(i)}$