Neural networks

Virgil Pavlu October 1, 2008

1 The perceptron

Lets suppose we are (as with regression regression) with $(\mathbf{x}_i, y_i); i = 1, .., m$ the data points and labels. This is a classification problem with two classes $y \in \{-1, 1\}$

Like with regression we are looking for a linear predictor (classifier)

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{w} = \sum_{d=0}^{D} x^d w^d$$

(we added the $x^0 = 1$ component so we can get the free term w^0) such that $h_{\mathbf{w}}(\mathbf{x}) \geq 0$ when $y = 1$ and $h_{\mathbf{w}}(\mathbf{x}) \leq 0$ when $y = -1$.

On $y = -1$ data points: given that all $\mathbf{x}$ and y are numerical, we will make the following transformation: when $y = -1$, we will reverse the sign of the input; that is replace $\mathbf{x}$ with **-x** and $y = -y$. Then the condition $h_{\mathbf{w}}(\mathbf{x}) \leq 0$ becomes $h_{\mathbf{w}}(\mathbf{x}) \geq 0$ for all data points.

The perceptron objective function is a combination of the number of miss-classification points and how bad the miss-classification is

$$J(\mathbf{w}) = \sum_{\mathbf{x} \in M} -h_{\mathbf{w}}(\mathbf{x}) = \sum_{\mathbf{x} \in M} -\mathbf{x}\mathbf{w}$$

where M is the set of miss-classified data points. Note that each term of the sum is positive, since miss-classified implies $\mathbf{w}\mathbf{x} < 0$. Using gradient descent, we first differentiate J

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \sum_{\mathbf{x} \in M} -\mathbf{x}^T$$

then we write down the gradient descent update rule

$$\mathbf{w} := \mathbf{w} + \lambda \sum_{\mathbf{x} \in M} \mathbf{x}^T$$

(λ is the learning rate). The batch version looks like

1. init $\mathbf{w}$
2. LOOP
3. get M = set of missclassified data points
4. $\mathbf{w} = \mathbf{w} + \lambda \sum_{\mathbf{x} \in M} \mathbf{x}^T$
5. UNTIL $|\lambda \sum_{\mathbf{x} \in M} \mathbf{x}| < \epsilon$

Assume the instances are linearly separable. Then we can modify the algorithm

1. init $\mathbf{w}$
2. LOOP
3. get M = set of missclassified data points
4. for each $\mathbf{x} \in M$ do $\mathbf{w} = \mathbf{w} + \lambda \mathbf{x}^T$
5. UNTIL M is empty

Proof of perceptron convergence Assuming data is linearly separable , or there is a solution $\bar{\mathbf{w}}$ such that $\mathbf{x}\bar{\mathbf{w}} > 0$ for all $\mathbf{x}$.
Lets call $\mathbf{w}_k$ the $\mathbf{w}$ obtained at the k-th iteration (update). Fix an $\alpha > 0$. Then

$$\mathbf{w}_{k+1} - \alpha\bar{\mathbf{w}} = (\mathbf{w}_k - \alpha\bar{\mathbf{w}}) + \mathbf{x}_k^T$$

where $\mathbf{x}_k$ is the datapoint that updated $\mathbf{w}$ at iteration k. Then

$$||\mathbf{w}_{k+1} - \alpha\bar{\mathbf{w}}||^2 = ||\mathbf{w}_k - \alpha\bar{\mathbf{w}}||^2 + 2\mathbf{x}_k(\mathbf{w}_k - \alpha\bar{\mathbf{w}}) + ||\mathbf{x}_k||^2 \leq ||\mathbf{w}_k - \alpha\bar{\mathbf{w}}||^2 - 2\mathbf{x}_k\alpha\bar{\mathbf{w}} + ||\mathbf{x}_k||^2$$

Since $\mathbf{x}_k\bar{\mathbf{w}} > 0$ all we need is an α sufficiently large to show that this update process cannot go on forever. When it stops, all datapoints must be classified correctly.

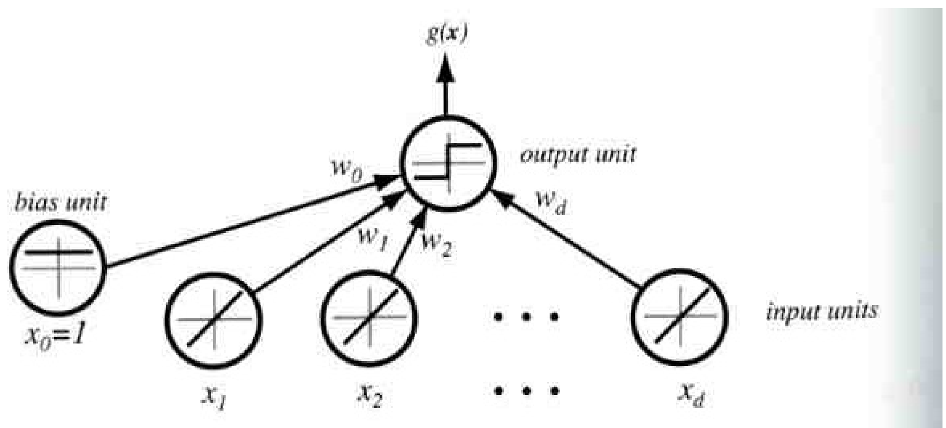

Figure 1: bias unit

2 Multilayer perceptrons

Also called *feed-forward networks*.

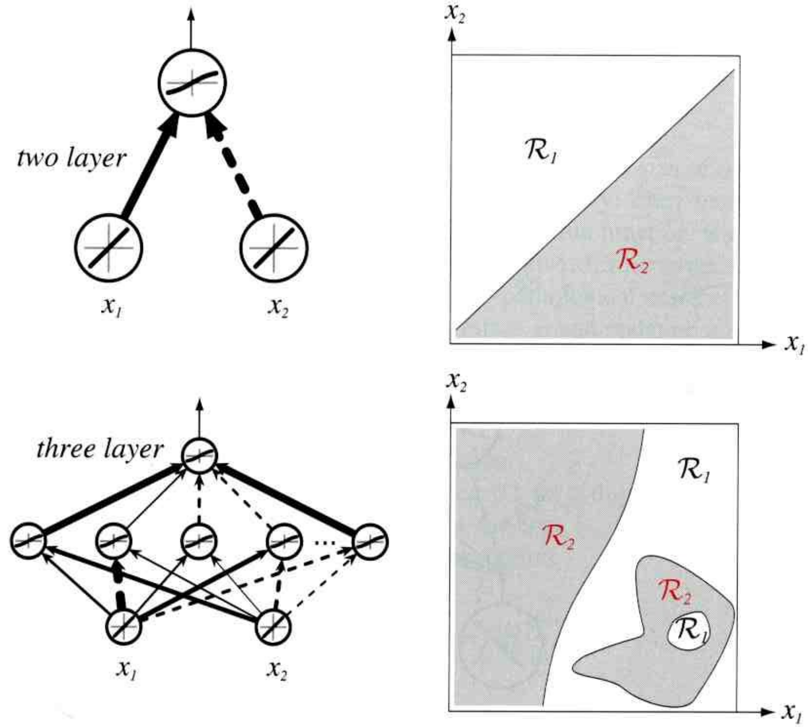

Figure 2: multilayer perceptron

2.1 More than linear functions, example: XOR

- activation functions, XOR example

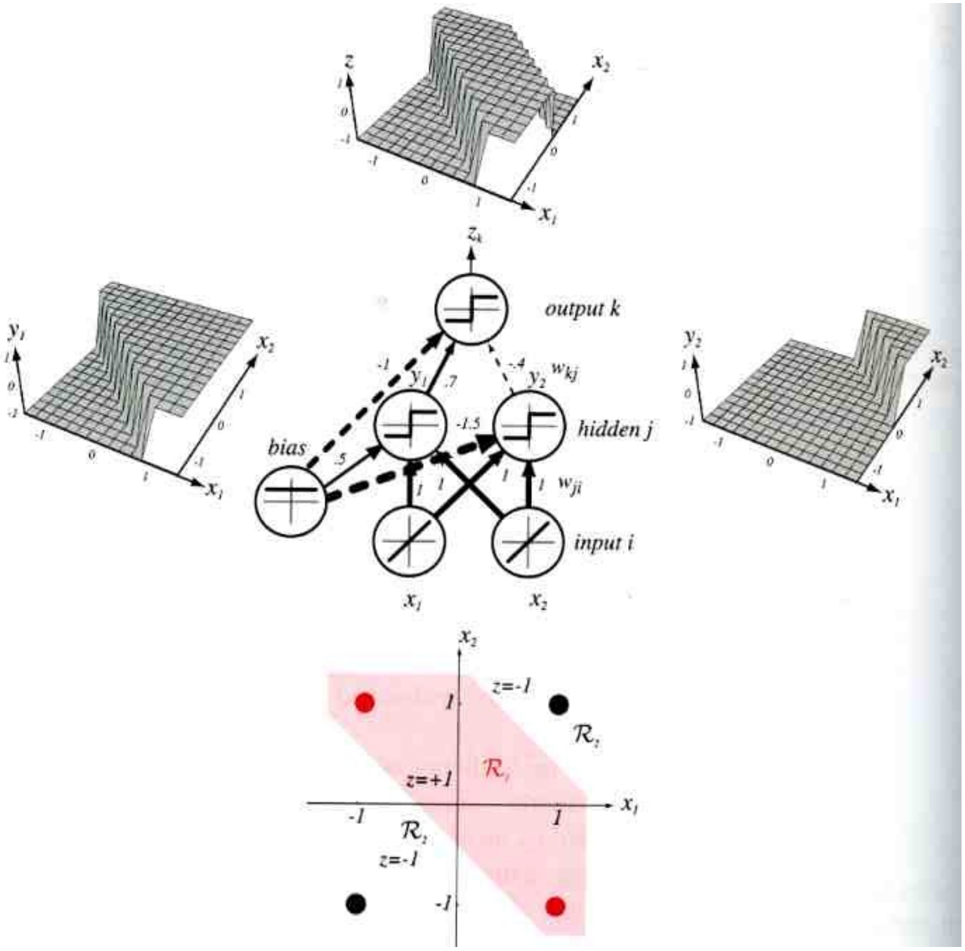

Figure 3: XOR NNet

2.2 Construction and structure of NNets

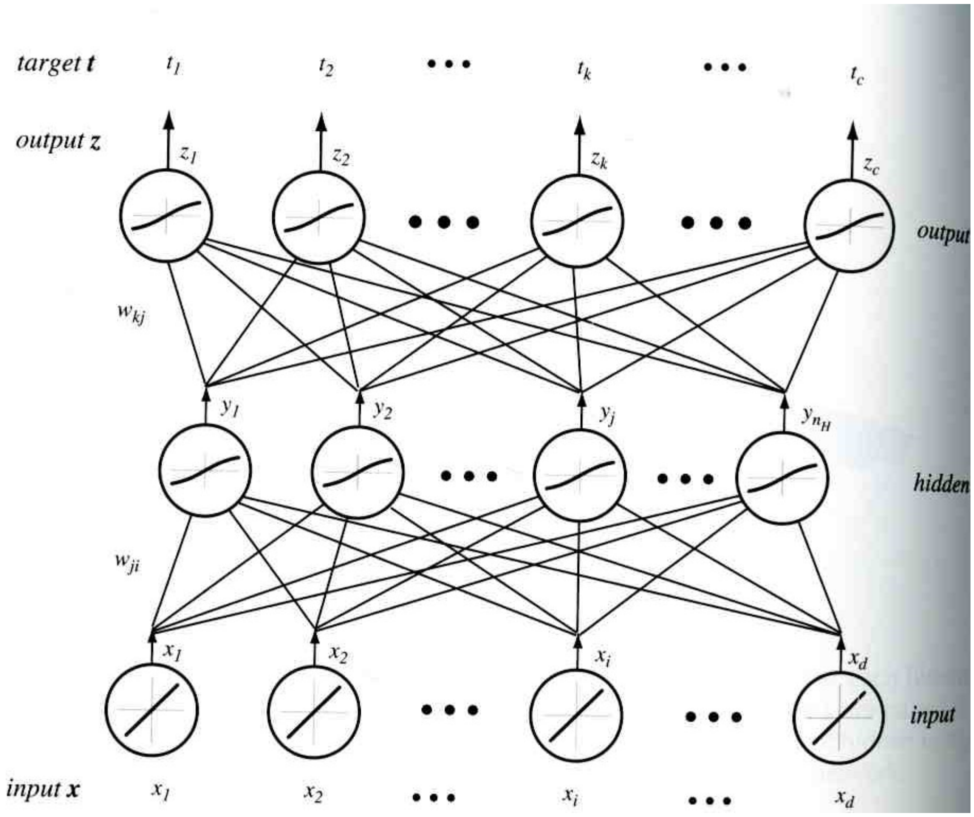

Figure 4: NNet fully connected

The oputpit units can be written as

$$g_k(\mathbf{x}) = z_k = f\left(\sum_j w_{kj} f\left(\sum_i w_{ij} x^i + w_{j0}\right) + w_{k0}\right) = F\left(F(\mathbf{x}\mathbf{w}_j)\mathbf{w}_k\right)$$

2.3 Kolmogorov theorem, expressive power of NNet

Any function g can be written

$$g(\mathbf{x}) = \sum_j \Xi_j \left(\sum_d \Psi_{dj}(x^d)\right)$$

but there is no practical way to use this theorem in practice. Usually Ξ and Ψ are very complex and not smooth.

3 Training, Error backpropagation

- error

- propagation to last set of weights (close to output)

- propagation to first set of weights (close to input)

- stochastic VS batch

4 How to improve backpropagation

4.1 Activation function

- continuous, differentiable (smoothness)
  - nonlinear
  - saturation
  - monotonicity

4.2 Scale input

4.3 Target values

4.4 Noise

4.5 Number of hidden units

4.6 Weights initialization

4.7 Learning rates

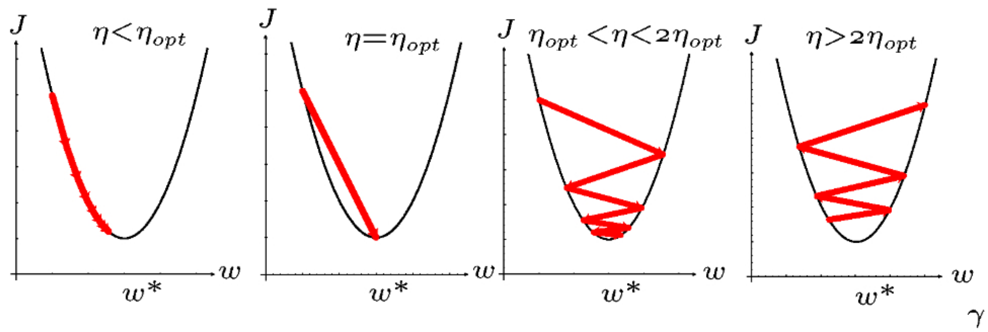

Figure 6.18: Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges.

Figure 5: learning rates

4.8 Construction and structure of NNets

5 Network size and structure. Regularization

6 Jacobian and Hessian