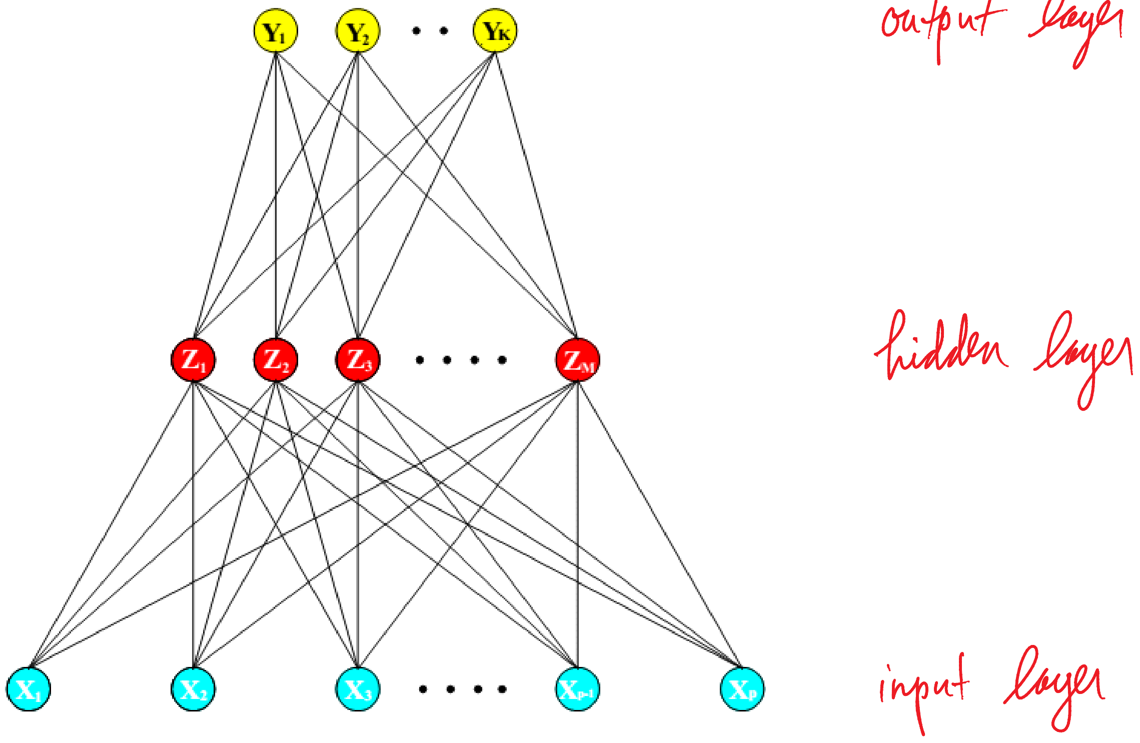
NEURAL NETWORKS

Basics

Neural networks are nonlinear models for supervised learning. They can be extended to unsupervised learning problems too, but our initial focus is supervised learning.

A neural network with a single hidden layer is depicted by the following schematic:



Input variables: $X_1, \dots, X_p$

Hidden variables: $Z_1, \dots, Z_M$

Output variables: $Y_1, \dots, Y_K$

$$X = \begin{bmatrix} 1 \\ X_1 \\ \vdots \\ X_p \end{bmatrix}, \quad Z = \begin{bmatrix} 1 \\ Z_1 \\ \vdots \\ Z_M \end{bmatrix}$$

The output predicted by a neural network given an input X is

$$(f_1(X), \dots, f_K(X))$$

where

$$f_k(X) = g_k(T) \qquad \text{where} \qquad T = \begin{bmatrix} T_1 \\ \vdots \\ T_K \end{bmatrix}$$

$$T_k = \beta_k^T Z = \beta_{k0} + \sum_{m=1}^{M} \beta_{km} Z_m ,$$

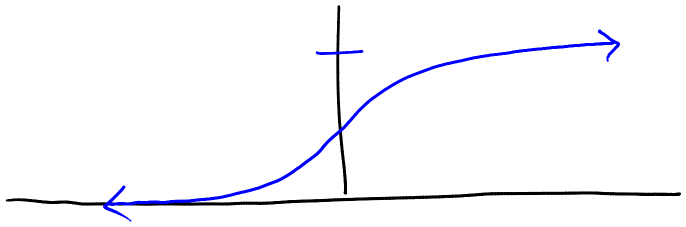
$$k = 1, \dots, K$$

$$Z_m = \sigma(\alpha_m^T X) = \sigma\left(\alpha_{m0} + \sum_{\ell=1}^{p} \alpha_{m\ell} X_\ell\right),$$

$$m = 1, \dots, M$$

The function σ is usually taken to be the sigmoid function

$$\sigma(t) = \frac{1}{1+e^{-t}}$$



The functions g_k depend on the supervised learning problem.

In regression, $K=1$ and $g(T)=T$. Then the regression model is

$$f(x) = \beta^T Z$$

$$= \beta_0 + \sum_m \beta_m \sigma(\alpha_m^T X)$$

In multiclass classification with K classes, if a pattern X belongs to class k, we associate the output

$$Y = [0 \cdots 0 \; 1 \; 0 \cdots 0]^T$$

$\nwarrow$ kth position

and set

$$g_k(T) = \frac{e^{T_k}}{\sum_{j=1}^{k} e^{T_j}}.$$

The classifier is

$$x \longmapsto \arg\max_k f_k(x)$$

In essence, $(f_1(x), \ldots, f_K(x))$ models the posterior probability distribution on the labels. In fact, the model may be viewed as multiclass logistic regression in the nonlinear features Z, except that Z is also learned.

Remarks

1. Like SVMs, NNs fit a linear model in a nonlinear feature space. Unlike SVMs, the nonlinear features are learned. Unfortunately, however, training involves nonconvex optimization.

2. Neural networks were originally conceived as models for the brain, where nodes are neurons and edges are synapses. In the early models, σ was taken to be a step function, meaning neurons "fire" when the total incoming signal exceeds a certain threshold.

3. Neural networks with one hidden layer are universal approximators. Provided M may be arbitrarily large,
   - in regression, the can approximate any continuous function arbitrarily well in the sup norm.
   - in classification, they can come arbitrarily close to the Bayes error.

4. Neural networks may have multiple hidden layer.

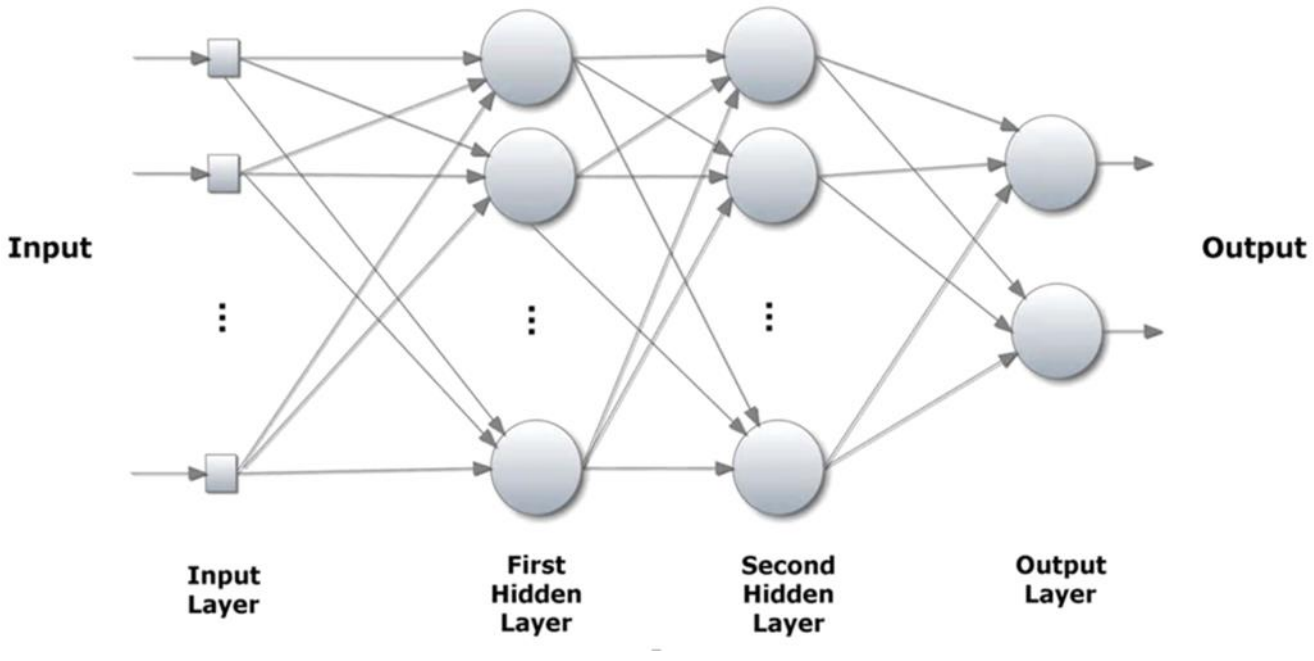
For example, consider two hidden layers:

$$f_k(X) = g_k(T)$$

$$T_k = \gamma_k^T W, \quad k=1,\dots,K \qquad W = \begin{bmatrix} 1 \\ W_1 \\ \vdots \\ W_L \end{bmatrix}$$

$$W_\ell = \sigma(\beta_\ell^T Z), \quad \ell = 1,\dots,L$$

$$Z_m = \sigma(\alpha_m^T X), \quad m = 1,\dots,M$$



Although one hidden layer already ensures universal approximation, with multiple hidden layers the same expressive power can be achieved with fewer parameters to learn (edge weights), mean less training data is necessary for good generalization.

Training Neural Networks via Backpropagation

Consider training data $(x_1, y_1), \dots, (x_N, y_N)$ where

$$x_i = \begin{bmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix}, \quad y_i = \begin{bmatrix} y_{i1} \\ \vdots \\ y_{iK} \end{bmatrix}$$

For simplicity, let's focus on NNs with a single hidden layer. Let θ denote the complete set of parameters,

$$\theta = (\underbrace{\alpha_1, \dots, \alpha_M}_{M(p+1)}, \underbrace{\beta_1, \dots, \beta_K}_{K(M+1)})$$

$M(p+1) + K(M+1)$ total parameters

The objective function depends on the setting:

- multi-output regression

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2$$

- multi-class classification "cross-entropy"

• multi-class classification

"cross-entropy"

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

= negative log-likelihood for multiclass logistic regression

We will attempt to minimize the objectives via gradient descent. For concreteness, let's focus on the squared error. Write

$$R(\theta) = \sum_{i=1}^{N} R_i(\theta)$$

where

$$R_i(\theta) = \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2$$

Then

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))\, g_k'(\beta_k^T z_i)\, z_{im}$$

$$=: \delta_{ki}\, z_{im}$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -2 \sum_{i=1}^{K} (y_{ik} - f_k(x_i))\, g_k'(\beta_k^T z_i)\, \beta_{km}\, \sigma'(\alpha_m^T x_i)\, x_{i\ell}$$

$$=: s_{mi}\, x_{il}$$

The gradient descent update at iteration $t+1$ is

$$\beta_{km}^{(t+1)} = \beta_{km}^{(r)} - \gamma_t \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}}(\theta^{(t)})$$

$$\alpha_{ml}^{(t+1)} = \alpha_{ml}^{(t)} - \gamma_t \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}}(\theta^{(t)})$$

where γ_t is the learning rate.

Notice that

$$s_{mi} = \sigma'(\alpha_m^T x_i) \cdot \sum_{k=1}^{K} \beta_{km}\, \delta_{ki}$$

This suggest an efficient two pass algorithm:

```
Initialize $\theta^{(0)}$
$t \leftarrow 0$
While termination criterion not satisfied
    • Forward pass: Using current weights $\theta^{(t)}$ compute
          $f_k(x_i)$   $\forall\, i = 1, \dots, N$,  $k = 1, \dots, K$
    • Backward pass: compute $\delta_{ki}$  $\forall\, i, k$,
          and then "back-propagate" these values
          [illegible]
```

```
            and then back-propagate these values
            to compute $s_{mi}$ $\forall m, i$
        • Compute gradient descent update
        • $t \leftarrow t+1$
End
```

The same idea applies to NNs with multiple layers, and to the multiclass classification loss.

An important advantage of backpropagation is that each hidden unit passes and receives information to and from only those units to which it is connected. Thus it can be implemented efficiently using parallel programming.

Implementation Details

There are many practical considerations when training neural networks. Here are some things to keep in mind.

1. To avoid saturating the sigmoids, data should be preprocessed to have zero mean and unit variance.

2. In general, $R(\theta)$ will have several local minima. Therefore it is a good idea to try several random starting values. Assume preprocessing as described above, $\alpha_{m\ell}, \beta_{km} \in [-0.7, 0.7]$ is reasonable.

3. Overfitting is a major concern when training neural networks. Therefore some form of regularization is needed. Perhaps the most common approach is early stopping, i.e., stopping backpropagation before reaching a local min. This is based on the following intuition: In the initial iterations, when the weights are near zero, the model is approximately linear because $\sigma(t)$ is approximately linear near $t=0$. As the weights increase, the model becomes more nonlinear. Thus, early stopping can be thought of as regularization toward a linear model.

4. Model selection: Determining the number of hidden

layers and hidden nodes per layer is very much an art.