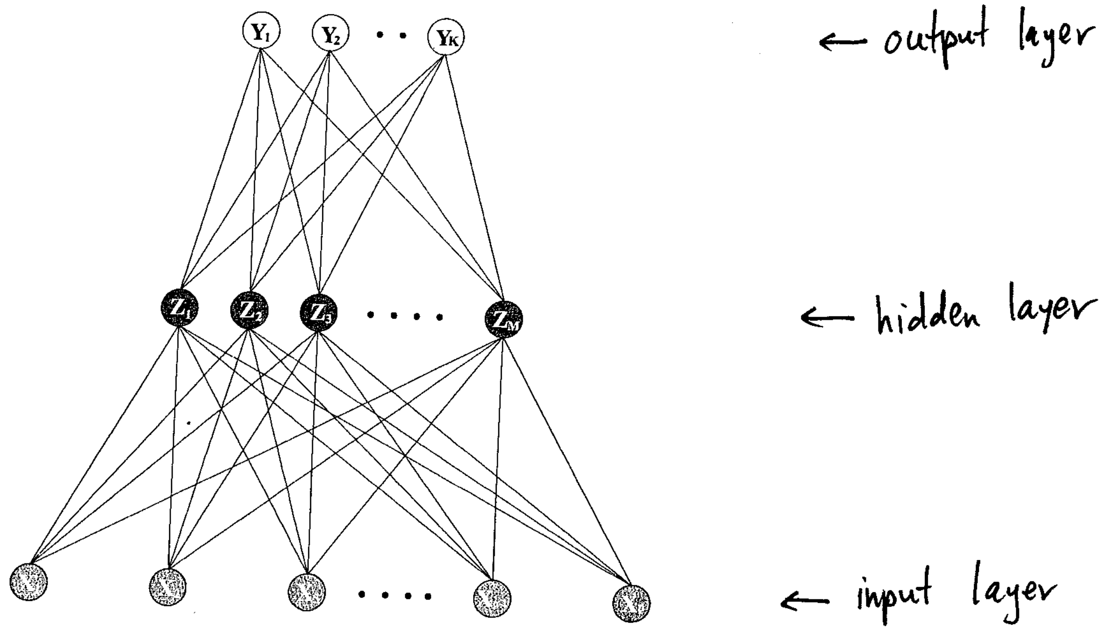
NEURAL NETWORKS

Neural networks are nonlinear models for supervised learning (classification and regression). A neural net with a single hidden layer is depicted by the following schematic:



Formally, we have

$$Z_m = \sigma(\alpha_{m0} + \alpha_m^T X), \quad m = 1, \dots, M$$

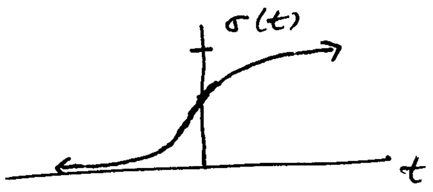
$$T_k = \beta_{k0} + \beta_k^T Z, \quad k = 1, \dots, K$$

$$f_k(X) = g_k(T), \quad k = 1, \dots, K \leftarrow \text{prediction of } Y_k$$

where $Z = (Z_1, \dots, Z_M)^T$, $T = (T_1, \dots, T_K)^T$

σ is usually taken to be the sigmoid function

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



Examples

Regression Here $K=1$ since Y is scalar, and $g(T) = T$. Then the regression model is

$$f(X) = \sum_{m=1}^{M} \beta_m \cdot \sigma(\alpha_{m0} + \alpha_m^T X) + \beta_0$$

note: this is a linear model in the nonlinear features $Z_m = \sigma(\alpha_{m0} + \alpha_m^T X)$

Binary classification Again we can take $K=1$ and $Y \in \{-1, +1\}$. Let's also take $g(t) = \frac{1-e^{-t}}{1+e^{-t}}$

Then the model predicts the sign of

$$g\left(\sum_{m=1}^{M} \beta_m \cdot \sigma(\alpha_{m0} + \alpha_m^T X) + \beta_0 \right)$$

We could also have taken $g(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ -1 & \text{if } t < 0 \end{cases} = \text{sign}(t)$,

but for training it is preferable that g be differentiable. Again, this classifier is a linear model in the nonlinear features Z_m.

<u>Multiclass Classification</u> Now let K = # of classes.

If pattern X belongs to class k, then we define the corresponding label to be

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

$\uparrow$ kth position

We also take

(A) $$g_k(T) =$$

the "softmax" function. The idea is that $f_k(X) = g_k(T)$ models the ________ probability of class k. Then the final classifier is

$$X \longmapsto$$

Even for binary classification $(K=2)$, this formulation is generally preferred.

It amounts to multiclass logistic regression in the nonlinear features Z_m.

Remarks

- Like SVMs, NNs fit a linear model in a nonlinear feature space. Unlike SVMs, those nonlinear features are <u>learned</u>. Unfortunately, however, training involves <u>nonconvex</u> optimization.

- NNs were originally conceived as models for the brain, where nodes are <u>neurons</u> and edges are <u>synapses</u>. In these early models, σ was taken to be a step function, meaning neurons "fire" when the total incoming signal exceeds a certain threshold.

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

For details, see Devroye, Györfi, and Lugosi, A Probabilistic Theory of Pattern Recognition.

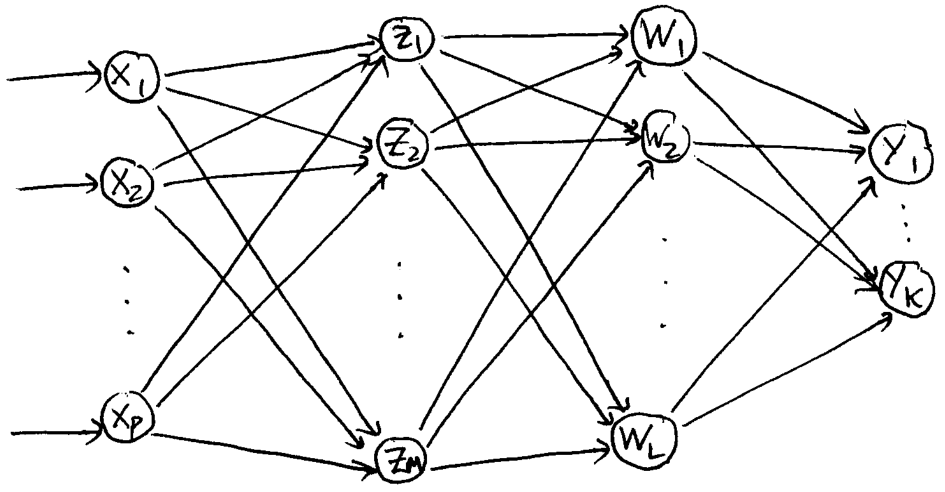
- Neural networks may have multiple hidden layers. For example, consider <u>two</u> hidden layers:

$$Z_m = \sigma(\alpha_{m0} + \alpha_m^T X), \quad m = 1, \dots, M$$

$$W_\ell = \sigma(\beta_{\ell 0} + \beta_\ell^T Z), \quad \ell = 1, \dots, L$$

$$T_k = \gamma_{k0} + \gamma_k^T W$$

$$f_k(x) = g_k(T)$$



Although one hidden layer already ensures universal approximation, there are practical advantages to multiple layers: fewer nodes / connections for the same expressive power.

Training Neural Networks: Backpropagation

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

where

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

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

$$\{\alpha_{m0}, \alpha_m \; ; \; m = 1, \dots, M\} \longrightarrow M(p+1)$$

$$\{\beta_{k0}, \beta_k \; ; \; k = 1, \dots, K\} \longrightarrow K(M+1)$$

Objective functions

- Regression or binary classification

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

- Multi-class classification

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

= negative log-likelihood for multiclass logistic regression

Gradient Descent

Let focus on the squared error loss.

Write

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

where

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

For convenience, let's add constant vectors to x_i and z so that

$$x_i = \begin{bmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix}, \quad z_i = \begin{bmatrix} 1 \\ z_{i1} \\ \vdots \\ z_{iM} \end{bmatrix}$$

and

$$\begin{aligned} f_k(x_i) &= g_k(\beta_k^T z_i) \\ &= g_k\left(\beta_{k0} + \sum_{m=1}^{M} \beta_{km} \sigma(\alpha_m^T x_i)\right) \end{aligned}$$

Then

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

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

$$=: s_{mi}\, x_{i\ell}$$

Then the gradient descent update at iteration $r+1$ is

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

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}} (\theta^{(r)})$$

where γ_r is the "learning rate."

Notice that

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

This suggests an efficient two-pass algorithm.

Backpropagation

Forward pass : using current weights $\theta^{(r)}$, compute

$\hat{f}_k(x_i)$, $i=1,\dots,N$, $k=1,\dots,K$

Backward pass : calculate δ_{mi}, and then "back-propagate" these values to the previous layer

Remarks

- The same concept applies for
  - multiple layers of hidden nodes
  - the multiclass classification loss
- The 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 processing.

Issues with Neural Network Training

Convergence speed

Backpropagation can be slow; conjugate gradient and other methods can converge faster.

Scaling inputs

To avoid saturation, data should be pre-processed to have zero mean and unit variance.

Starting values

In general $R(\theta)$ will have several local minima. Therefore it is good to try several random starting values. Assuming pre-processing as described above, $\alpha_{me}, \beta_{km} \in [-0.7, 0.7]$ is reasonable.

When the weights are near 0, the model is approximately linear (because $\sigma(t)$ is $\approx$ linear near $t=0$). Thus the model starts near a linear model and becomes more nonlinear as the weights increase.

Online learning

BP and other gradient-based algorithms can be implemented online. This is why NNs are popular in reinforcement learning.

Regularization

To avoid overfitting, regularization is necessary.

Two approaches:

- early stopping: shrinkage toward a linear model
- penalization: minimize

$$R(\theta) + \lambda J(\theta)$$

e.g.

$$J(\theta) = \sum_{k,m} \beta_{km}^2 + \sum_{m,\ell} \alpha_{m\ell}^2$$

Model Selection

Choosing the number of hidden layers and neurons is very much an art. Also, various "constrained" models have been proposed, e.g. some weight must have the same value (convolutional NNs). In general it is better to have too many than too few, and rely on regularization to avoid overfitting.

$\boxed{\text{Key}}$ A. $g_k(T) = e^{T_k} / \sum_{\ell=1}^{k} e^{T_\ell}$, posterior,

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