Notes on Statistical and Machine Learning

Neural Networks

Chapter: *20* **Prepared by:** *Chenxi Zhou*

This note is prepared based on *Chapter 11, Neural Networks* in Hastie, Tibshirani, and Friedman (2009).

I. Neural Networks

1. **Main Idea:** The central idea of the neural networks is to

   (a) extract linear combinations of the input variables as *derived features*, and

   (b) model the target as a *non-linear* function of these derived features.

2. **Basics:** In this section, we discuss the "vanilla" neural network, also called the *single hidden layer back-propagation network* or *single layer perceptron*.

   In this sense, a neural network is a *two-stage regression* or *classification model* represented by a network diagram (see Figure 1). From a statistical perspective, they are just nonlinear statistical models.

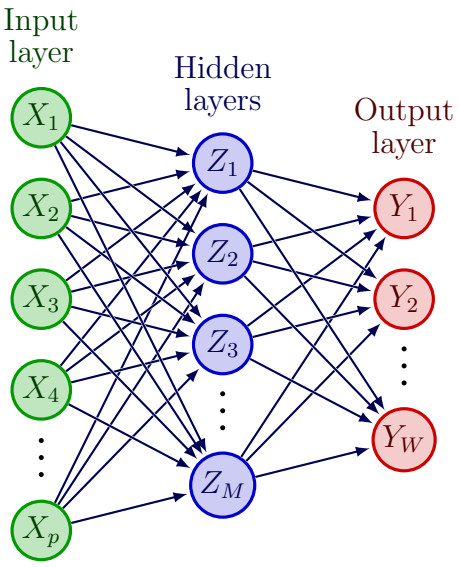

Figure 1: Schematic of a single hidden layer, feed-forward neural network.

   (a) *Output Layer:* We use W to denote the number of units in the output layer.

   - For regression, typically $W = 1$ and there is only one output unit Y_1. It can be extended to handle multiple quantitative responses.

- For W-class classification, there are W units at the output layer with the w-th unit modeling the probability of Class w. There are W target measurements Y_w, for all $w = 1, \cdots, W$, each being coded as a binary variable for the w-th class.

(b) *Derived features:* Derived features $\{Z_m\}_{m=1}^M$ are created from linear combinations of the inputs $X_1, \cdots, X_p$, and then the target Y_k is modeled as a function of linear combinations of the Z_m's, that is,

$$\begin{aligned} Z_m &= \sigma(\alpha_{0,m} + \boldsymbol{\alpha}_m^\top X), && m = 1, \cdots, M, \\ T_w &= \beta_{0,w} + \boldsymbol{\beta}_w^\top Z, && w = 1, \cdots, W, \\ Y_w &= f_w(X) = g_w(T), && w = 1, \cdots, W, \end{aligned}$$

where $Z = (Z_1, Z_2, \cdots, Z_M)$ and $T = (T_1, T_2, \cdots, T_W)$.

(c) *Hidden Layer:* The units in the middle of the network, computing the derived features Z_m, are called *hidden units* because Z_m are *not* directly observed.

Remark. In general, there can be more than one hidden layer. We can think of the Z_m as a basis expansion of the original inputs X.

(d) *Activation Function:* The activation function σ is usually chosen to be the sigmoid $\sigma(v) = 1/(1 + e^{-v})$. By using a nonlinear transformation σ, it greatly enlarges the class of linear models.

Remark. If σ is the identity function, then the entire model collapses to a linear model in the input variables. Hence, a neural network can be thought of as a nonlinear generalization of the linear model, both for regression and classification.

(e) *Output Function:* The output function $\{g_w\}_{w=1}^W$ allows a final transformation of the vector of outputs T.

- For regression, we typically choose the identity function $g_w(T) = T_w$ for all $w = 1, 2, \cdots, W$;
- For classification, one popular choice is the softmax function

$$g_w(T) = \frac{e^{T_w}}{\sum_{\ell=1}^W e^{T_\ell}}, \qquad \text{for all } w = 1, 2, \cdots, W. \tag{1}$$

3. Connection between Neural Network and Projection Pursuit Regression: Neural networks with one hidden layer has exactly the same form as the projection pursuit model. The only difference is that the PPR model uses nonparametric functions g_m, while the neural network uses σ.

Viewing the neural network model as a PPR model, we have

$$\begin{aligned} g_m(\boldsymbol{\omega}_m^\top X) &= \beta_m \sigma(\alpha_{0,m} + \boldsymbol{\alpha}_m^\top X) \\ &= \beta_m \sigma(\alpha_{0,m} + \|\boldsymbol{\alpha}_m\|_2 (\boldsymbol{\omega}_m^\top X)), \end{aligned} \tag{2}$$

where $\boldsymbol{\omega}_m := \boldsymbol{\alpha}_m / \|\boldsymbol{\alpha}_m\|_2$ is the m-th unit vector.

II. Fitting Neural Networks

1. **Weights:** The unknown parameters in neural networks are often called *weights*. In a neural network with one hidden layer, denote the complete set of weights by $\boldsymbol{\theta}$, consisting of

   (a) $\{\alpha_{0,m}, \boldsymbol{\alpha}_m \,|\, \boldsymbol{\alpha}_m \in \mathbb{R}^p\}_{m=1,\cdots,M}$, which are $M(p+1)$ parameters,

   (b) $\{\beta_{0,w}, \boldsymbol{\beta}_w \,|\, \boldsymbol{\beta}_w \in \mathbb{R}^M\}_{w=1,\cdots,W}$, which are $W(M+1)$ parameters.

2. **Loss Function:**

   (a) For regression, we use sum-of-squared errors as the loss function

   $$L(\boldsymbol{\theta}) = \sum_{w=1}^{W}\sum_{i=1}^{n}(y_{i,w} - f_w(\mathbf{x}_i))^2. \tag{3}$$

   (b) For classification, use either squared error or cross-entropy

   $$L(\boldsymbol{\theta}) = -\sum_{w=1}^{W}\sum_{i=1}^{n} y_{i,w}\log f_w(\mathbf{x}_i), \tag{4}$$

   and the corresponding classifier is $G(\mathbf{x}) = \arg\max_{w=1,\cdots,W} f_w(\mathbf{x})$.

3. **Regularization:** Typically, we add a penalty term to L or stop the algorithm early to avoid an overfit solution in fitting a neural network.

4. **Minimization Method:** We typically minimize L by gradient descent, called *back-propagation* in this setting. The gradient can be easily derived using the chain rule for differentiation.

5. **Back-Propagation Using Squared Error Loss:** Consider the squared error loss. Let $z_{m,i} := \sigma(\alpha_{0,m} + \boldsymbol{\alpha}_m^\top \mathbf{x}_i)$ and $\mathbf{z}_i := (z_{1,i}, z_{2,i}, \cdots, z_{M,i})$. Then, we have

   $$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} L_i = \sum_{i=1}^{n}\sum_{w=1}^{W}(y_{i,w} - f_w(\mathbf{x}_i))^2.$$

   And,

   $$\begin{aligned}
   \frac{\partial L_i}{\partial \beta_{w,m}} &= -2(y_{i,w} - f_w(\mathbf{x}_i))g_w'(\boldsymbol{\beta}_w^\top \mathbf{z}_i)z_{m,i},\\
   \frac{\partial L_i}{\partial \alpha_{m,\ell}} &= -\sum_{w=1}^{W} 2(y_{i,w} - f_w(\mathbf{x}_i))g_w'(\boldsymbol{\beta}_w^\top \mathbf{z}_i)\beta_{w,m}\sigma'(\boldsymbol{\alpha}_m^\top \mathbf{x}_i)x_{i,\ell}.
   \end{aligned}$$

   Then, the gradient update at the $(j+1)$-st iteration has the form

$$\begin{aligned}\beta_{w,m}^{(j+1)} &= \beta_{w,m}^{(j)} - \gamma_j \sum_{i=1}^{n} \frac{\partial L_i}{\partial \beta_{w,m}}\bigg|_{\beta_{w,m}=\beta_{w,m}^{(j)}}, \\ \alpha_{m,\ell}^{(j+1)} &= \alpha_{m\ell}^{(j)} - \gamma_j \sum_{i=1}^{n} \frac{\partial L_i}{\partial \alpha_{m,\ell}}\bigg|_{\alpha_{m,\ell}=\alpha_{m,\ell}^{(j)}},\end{aligned} \tag{5}$$

where $\gamma_j > 0$ is the learning rate.

If we define

$$\begin{aligned}\delta_{w,i} &:= -2(y_{i,w} - f_w(\mathbf{x}_i))g_w'(\boldsymbol{\beta}_w^\top \mathbf{z}_i), \\ s_{m,i} &:= -\sum_{w=1}^{W} 2(y_{i,w} - f_w(\mathbf{x}_i))g_w'(\boldsymbol{\beta}_w^\top \mathbf{z}_i)\beta_{w,m}\sigma'(\boldsymbol{\alpha}_m^\top \mathbf{x}_i),\end{aligned}$$

we can write

$$\frac{\partial L_i}{\partial \beta_{w,m}} = \delta_{w,i} z_{m,i}, \qquad \text{and} \qquad \frac{\partial L_i}{\partial \beta_{m,\ell}} = s_{m,i} x_{i,\ell}. \tag{6}$$

The quantities $\delta_{w,i}$ and $s_{m,i}$ are "errors" from the current model at the output and hidden layer units, respectively. There errors satisfy the equation

$$s_{m,i} = \sigma'(\boldsymbol{\alpha}_m^\top \mathbf{x}_i) \sum_{w=1}^{W} \beta_{w,m}\delta_{w,i}, \tag{7}$$

known as the *back-propagation equations*.

6. Two-Pass Algorithm:

(a) *Forward Pass:* current weights are fixed and the predicted values $\hat{f}_w(\mathbf{x}_i)$ are computed;

(b) *Backward Pass:* Compute the errors $\delta_{w,i}$ and then use back-propagation equation (7) to compute errors $s_{m,i}$.

7. Comments on Back-Propagation and Two-Pass Algorithm:

(a) *Advantages:* The advantages of back-propagation are its simple, local nature. It can be implemented efficiently on a parallel architecture computer.

(b) *Disadvantages:* Back-propagation can be very slow. Better approaches include conjugate gradients and variable metric methods.

Remark. Second-order optmization techniques such as Newton's method are *not* attractive in the neural network setting, because the Hessian matrix of L can be very large.

8. **Batch Learning and Online Learning:** Updates (5) are a kind of *batch learning*, with the parameter updates being a sum over all of the training cases.

   Online learning refers to processing each observation one at a time, updating the gradient after each training case, and cycling through the training cases many times. Online training allows the network to handle very large training sets, and also to update the weights as new observations come in.

   A *training epoch* refers to one sweep through the entire training set.

9. **Learning Rate:**

   (a) In *batch learning*, the learning rate γ_j for is usually taken to be a constant, and can also be optimized by a line search that minimizes the error function at each update;

   (b) In *online learning*, γ_j should decrease to zero as the iteration $j \to \infty$. This learning is a form of stochastic approximation. Algorithm is guaranteed to converge if $\gamma_j \to 0$, $\sum_{j=1}^{\infty} \gamma_j = \infty$ and $\sum_{j=1}^{\infty} \gamma_j^2 < \infty$.

III. Practical Issues in Training Neural Networks

1. **Starting Values for Weights:** Usually starting values for weights are chosen to be random values near zero. Hence, the model starts out nearly linear, and becomes nonlinear as the weights increase.

2. **Methods to Avoid Overfitting:**

   (a) *Early Stopping:* Train the model only for a while and stop well before we approach the global minimum. A validation dataset is useful for determining when to stop.

   (b) *Weight Decay:* Add a penalty to the error function and minimize $L(\boldsymbol{\theta}) + \lambda J(\boldsymbol{\theta})$, where

$$J(\boldsymbol{\theta}) = \sum_{w,m} \beta_{w,m}^2 + \sum_{m,\ell} \alpha_{m,\ell}^2, \tag{8}$$

   and $\lambda > 0$ is a tuning parameter and can be chosen using the cross validation. Larger values of λ will tend to shrink the weights toward zero.

   Other forms for the penalty have been proposed, for example,

$$J(\boldsymbol{\theta}) = \sum_{w,m} \frac{\beta_{w,m}^2}{1+\beta_{w,m}^2} + \sum_{m,\ell} \frac{\alpha_{m,\ell}^2}{1+\alpha_{m,\ell}^2}, \tag{9}$$

   known as the *weight elimination penalty*. This has the effect of shrinking smaller weights more than (8) does.

3. **Scaling of Inputs:** The scaling of input variables determines the effective scaling of the weights in the bottom layer, and has a large effects on the final output.

   Recommendation: At the outset, it is best to standardize all inputs to have mean zero and standard deviation one.

4. **Choice of the Number of Hidden Units:** It is usually better to have too many hidden units than too few.

   - With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data;
   - With too many hidden units, the extra weights can be shrunk toward zero if appropriate regularization is used.

   Typically choice of the number of hidden units is between 5 and 100. It is most common to put down a reasonably large number of units and train them with regularization.

5. **Choice of the Number of Hidden Layers:** Choice of the number of hidden layers is guided by background knowledge and experimentation. Use of multiple hidden layers allows construction of hierarchical features at different levels of resolution.

6. **Multiple Local Minima:** The loss function L is non-convex, possessing many local minima.

   (a) *Solution 1:* Try a number of random starting configurations, and choose the solution giving lowest (penalized) error.

   (b) *Solution 2:* Use the average predictions over the collection of networks as the final prediction.

References

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning*. Vol. 1. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.