Lecture Notes on Neural Networks – Draft
Advanced Machine Learning

Prof: Yannet Interian

February 6, 2017

Required Reading: Elements of Statistical learning, Hastie, Tibshirani and Friedman [1]. Chapter 11.3, 11.4, 11.5, 11.6.

Optional Reading:

- `http://www.deeplearningbook.org/` Chapter on Regularization, Chapter 8.4 on Initialization Strategies
- Bishop Chapter 5.

1 Introduction

Based on Chapter 11, Elements of Statistical learning, Hastie, Tibshirani and Friedman and Bishop Chapter 5.

In this lecture we describe a class of learning methods that was developed separately the fields of statistics and artificial intelligence. The central idea is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features. The result is a powerful learning method, with widespread applications in many fields.

Neural networks take nonlinear functions of linear combinations ("derived features") of the inputs. This is a powerful and very general approach for regression and classification, and has been shown to compete well with the best learning methods on many problems. They are especially effective in problems with a high signal-to-noise ratio and settings where prediction without interpretation is the goal.

2 Review of Logistic Regression

2.1 Two-class logistic regression

In the logistic regression model the posterior of the class "1" is written as the logistic sigmoid acting on a linear function of the feature vector x.

$$P(Y = 1|X = x) = \sigma(w_0 + w \cdot x)$$

with $P(Y = 0|X = x) = 1 - P(Y = 1|X = x)$ Here σ is the logistic sigmoid function $\sigma(a) = \frac{1}{1+e^{-a}}$. Let's denote $\hat{y}(x, w) = \sigma(w_0 + w \cdot x)$

Given a training dataset of N input vectors $x_1, \ldots, x_N$ with the corresponding target values $y_1, \ldots, y_N$ where $y_i \in \{0, 1\}$. How can we find the optimal w, w_0? We minimize the negative log likelihood. How do we get the likelihood?

$$\mathcal{L}(w, w_0) = \prod_{i=1}^{N} \hat{y}(x_i, w)^{y_i} \left(1 - \hat{y}(x_i, w)\right)^{1-y_i}$$

Therefore, the negative log likelihood has the following form

$$\mathcal{NLL}(w, w_0) = -\sum_{i=1}^{N} \{y_i \log \hat{y}(x_i, w) + (1 - y_i) \log(1 - \hat{y}(x_i, w))\}$$

2.2 Multiclass logistic regression

In multiclass logistic regression we extend the previous model for the case in which $y \in \{1, \ldots, K\}$ classes. In this case the posterior probabilities are given by a *softmax* transformation of linear functions.

$$P(Y = k|X = x) = \frac{e^{a_k}}{\sum_{k=1}^{K} e^{a_k}}$$

where $a_k = w_{k0} + w_k \cdot x$

We can encode y as a K dimensional vector $y = (y^1, \ldots, y^K)$ where if y is on class k y is a binary vector with all element zero except for element k which equals 1. The negative log likelihood function is then given by

$$\mathcal{NLL}(w) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_i^k \log \hat{y}_k(w_k, x_i)$$

where $\hat{y}_k(w_k, x_i) = P(Y = k|X = x_i)$.

3 Neural Networks

The term *neural network* has evolved to encompass a large class of models and learning methods. Here we describe the most widely used "vanilla" neural net, sometimes called the single hidden layer back-propagation network, or two-layer neural network. There has been a great deal of hype surrounding neural networks, making them seem magical and mysterious. As we make clear in this section, they are just nonlinear statistical models.

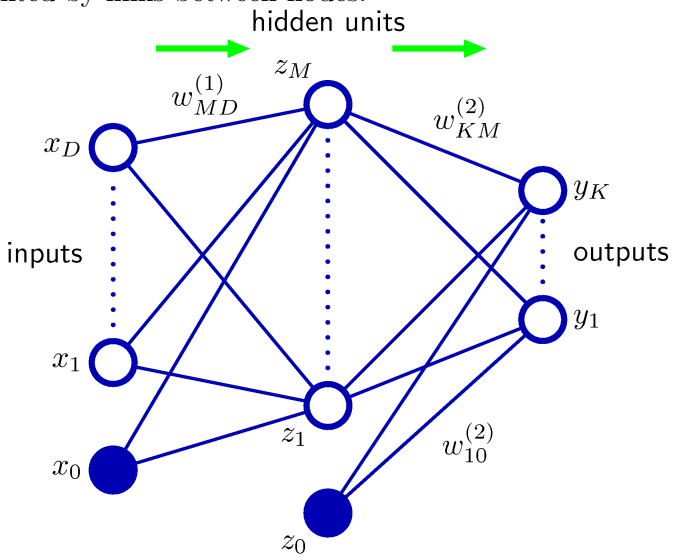
A neural network is a two-stage regression or classification model, typically represented by a network diagram as in Figure 1. This network applies both to regression or classification. For regression and binary classification, typically $K = 1$ and there is only one output unit Y_1. However, these networks can handle multiple quantitative or categorical responses in a seamless fashion. To simplify the notation we consider first the case of $K = 1$.

The following notation is based on Bishop. To describe a neural network model, we define a series of functional transformations. First we construct M linear combinations of the input variables $x = (x^1, \ldots, x^D)$ in the form

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x^i + w_{j0}^{(1)} = w_j^{(1)} \cdot x + w_{j0} \tag{1}$$

where $j = 1, \ldots, M$, and the superscript (1) indicates that the corresponding parameters are in the first "layer" of the network. The quantities $a_j^{(1)}$

Figure 1: Network diagram for a two-layer neural network. The input, hidden and output variables are represented by nodes. The weight parameters are represented by links between nodes.



are known as activations. Each of them is then transformed using a differentiable, nonlinear *activation function* $h(.)$ to give

$$z_j = h(a_j^{(1)}) \tag{2}$$

The nonlinear function $h(.)$ used to be chosen to be sigmoidal functions such as the logistic sigmoid ($\frac{1}{1+\exp(-a)}$). More recently the use of the rectified linear unit (ReLU) have been found to work better in practice [2]. The ReLu has the form $max(0, a)$.

The z_j values are linearly combined again

$$a^{(2)} = \sum_{j=1}^{M} w_j^{(2)} z_j + w_{k0}^{(2)} = w^{(2)} \cdot z + w_0^{(2)} \tag{3}$$

where $z = (z_1, \ldots, z_M)$. Finally, the $a^{(2)}$ are transformed using an appropriate activation function to give a set of network outputs $\hat{y}$. Thus for standard regression problems, the activation function is the identity so that $\hat{y} = a^{(2)}$.

For multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that

$$\hat{y} = \sigma(a^{(2)})$$

where

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

We can combine these various stages to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$\hat{y}(x, w) = \sigma\left(\sum_{j=1}^{M} w_j^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x^i + w_{j0}^{(1)}\right) + w_0^{(2)}\right)$$

3.1 Fitting Neural Networks

The neural network model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well. We denote the complete set of weights by w, which consists of

$$\{w_{ij}^{(1)}, w_j^{(2)}\}, i \in 1, \ldots D, j \in 1, \ldots M$$

For regression, we use sum-of-squared errors as our measure of fit (error function)

$$Err(w) = \sum_{n=1}^{N} (y_n - \hat{y}(x_n, w))^2$$

For classification we use either squared error or cross-entropy (deviance):

$$Err(w) = -\sum_{n=1}^{N} \{y_n \log \hat{y}(x_n, w) + (1 - y_n) \log(1 - \hat{y}(x_n, w))\}$$

Typically we don't want the global minimizer $Err(w)$, as this is likely to be an overfit solution. Instead some regularization is needed. Regularization

is achieved directly through a penalty term, or indirectly by early stopping. Details are given in the next section.

The generic approach to minimizing $Err(w)$ is by gradient descent, called back-propagation in this setting. The gradient descent equations are

$$w^{t+1} = w^t - \gamma \nabla Err(w)$$

where $\gamma \geq 0$ and γ could also be decreasing as a function of t.

Because of the compositional form of the model, the gradient can be easily derived using the chain rule for differentiation. This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit.

3.2 Back-propagation for a regression problem

Here is back-propagation in detail for a regression problem, fitting a squared error loss and one output variable. Given a training data $\{(x_n, y_n)\}_{n=1}^N$.

$$\hat{y}(x_n, w) = \sum_{j=1}^{M} w_j^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_n^i + w_{j0}^{(1)}\right) + w_0^{(2)}$$

$$\begin{aligned} Err(w) &= \sum_{n=1}^{N} Err_n \\ &= \sum_{n=1}^{N} (y_n - \hat{y}(x_n, w))^2 \end{aligned}$$

Here are the equation for the derivatives for one training point (x, y)

$$\frac{\partial Err}{\partial w_{ji}^{(1)}} = \delta w_j^{(2)} h'(a_j^{(1)}) x^i$$
$$\frac{\partial Err}{\partial w_{j0}^{(1)}} = \delta w_j^{(2)} h'(a_j^{(1)})$$
$$\frac{\partial Err}{\partial w_j^{(2)}} = \delta h(a_j^{(1)})$$
$$\frac{\partial Err}{\partial w_0^{(2)}} = \delta$$

where $\delta = -2(y - \hat{y}(x, w))$

Note that we can write these equations more compactly if we define $x = (1, x)$ and $z = (1, z)$ where $z_j = h(a_j^{(1)})$

$$\frac{\partial Err}{\partial w_{ji}^{(1)}} = \delta w_j^{(2)} h'(a_j^{(1)}) x^i$$
$$\frac{\partial Err}{\partial w_j^{(2)}} = \delta z_j$$

Also $\hat{y}(x, w)$ is simplified

$$\hat{y}(x_n, w) = \sum_{j=1}^{M} w_j^{(2)} z^j$$

where $z_j = h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x^i\right)$ and $z_0 = 1$

Can you write these equations in vectorize form?

3.3 $K > 1$

Similarly to multiclass logistic regression we can have neural networks for multiclass classification problems.

$$\hat{y_k}(w, x) = \frac{e^{a_k^2}}{\sum_{i=1}^{K} e^{a_k^2}}$$

where $a_k^2 = W^{(2)} \cdot z$, $z_0 = 1, z_j = h(a_j^{(1)})$ and $a_j^{(1)} = W^{(1)} \cdot x$ where $x^0 = 1$.

3.4 Issues in Training Neural Networks

There is quite an art in training neural networks. The model is generally overparametrized, and the optimization problem is nonconvex and unstable unless certain guidelines are followed. In this section we summarize some of the important issues.

Initial Values of w
See more on this subject in [2] (Chapter 8.4).

Training neural network algorithms are strongly affected by the choice of initialization. The initial point can determine whether the algorithm converges at all, with some initial points being so unstable that the algorithm encounters numerical difficulties and fails altogether.

Typically, biases are set to heuristically chosen constants, and weights are initialized randomly. Weights are set to values drawn randomly from a Gaussian or uniform distribution. One heuristic is to initialize the weights of a fully connected layer with m inputs and n outputs by sampling each weight from $U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$ another heuristic suggests $U(-\frac{\sqrt{6}}{\sqrt{m+n}}, \frac{\sqrt{6}}{\sqrt{m+n}})$. where U corresponds to the uniform distribution.

Overfitting: weight decay
Often neural networks have too many weights and will overfit the data at the global minimum of $Err(w)$. In early developments of neural networks, either by design or by accident, an early stopping rule was used to avoid overfitting. Here we train the model only for a while, and stop well before we approach the global minimum. Since the weights start at a highly regularized (linear)

solution, this has the effect of shrinking the final model toward a linear model. A validation dataset is useful for determining when to stop, since we expect the validation error to start increasing.

A more explicit method for regularization is **weight decay**, which is analogous to ridge regression used for linear models. We add a penalty to the error function $Err(w)$

$$Err(w) + \lambda(||w^{(1)}||^2 + ||w^{(2)}||^2)$$

and $\lambda \geq 0$ is a tuning parameter. Larger values of λ will tend to shrink the weights toward zero: typically cross-validation is used to estimate λ.

Overfitting revisited: Dropout
`http://www.deeplearningbook.org/contents/regularization.html`

Dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network. In most modern neural networks we can effectively remove a unit from a network by multiplying its output value by zero.

Srivastava et al. (2014) showed that dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, ?lter norm constraints and sparse activity regularization. Dropout may also be combined with other forms of regularization to yield a further improvement.

Overfitting revisited: Data Augmentation
The best way to make a machine learning model better is to train it on more data. In practice, the amount of data we have is limited. One way to get around this problem is to create fake data and add it to the training set. For some machine learning tasks, it is easy to create new fake data.

Dataset augmentation has been a particularly effective technique in object recognition. Images are high dimensional and include an enormous variety of factors of variation, many of which can be easily simulated. Operations like translating the training images a few pixels in each direction can often greatly improve generalization, even if the model has already been designed to be partially translation invariant by using the convolution and pooling techniques. Many other operations such as rotating the image or scaling the image have also proven quite effective.

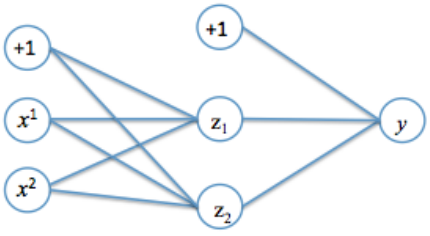
Scaling of the Inputs
Since the scaling of the inputs determines the effective scaling of the weights in the bottom layer, it can have a large effect on the quality of the final solution. At the outset it is best to standardize all inputs to have mean zero and standard deviation one. This ensures all inputs are treated equally in the regularization process, and allows one to choose a meaningful range for the random starting weights.

Number of Hidden Units and Layers
Typically the number of hidden units is somewhere in the range of 5 to 100, with the number increasing with the number of inputs and number of training cases. It is most common to put down a reasonably large number of units and train them with regularization. Some researchers use cross-validation to estimate the optimal number, but this seems unnecessary if cross-validation is used to estimate the regularization parameter.

Multiple Minima
The error function $Err(w)$ is nonconvex, possessing many local minima. As a result, the final solution obtained is quite dependent on the choice of starting weights. One must at least try a number of random starting configurations, and choose the solution giving lowest (penalized) error. Probably a better approach is to use the average predictions over the collection of networks as the final prediction. Another approach is via bagging, which averages the predictions of networks training from randomly perturbed versions of the training data.

3.5 Exercise

Consider the network below.



Assume the following network weights:

$w_{10}^{(1)}$	$w_{11}^{(1)}$	$w_{12}^{(1)}$	$w_{20}^{(1)}$	$w_{21}^{(1)}$	$w_{22}^{(1)}$	$w_{0}^{(2)}$	$w_{1}^{(2)}$	$w_{2}^{(2)}$
-30	20	20	10	-20	-20	-10	20	20

Let's say we want to calculate some binary functions. Given the below inputs, what output does the network calculate? You may use the hard threshold rule for both hidden and output layers instead of the sigmoid function:

$$\sigma(a) = h(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise.} \end{cases}$$

x^1	x^2	$a_1^{(1)}$	$a_2^{(1)}$	z_1	z_2	$a^{(2)}$	y
0	0						
0	1						
1	0						
1	1						

References

[1] Trevor J. Hastie, Robert John Tibshirani, and Jerome H. Friedman. *The elements of statistical learning : data mining, inference, and prediction.* Springer series in statistics. Springer, New York, 2009. Autres impressions : 2011 (corr.), 2013 (7e corr.).

[2] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.