Lecturer: Prof. Homrighausen Scribe: Lei Yang

In this lecture we learned the basic form and training method of neural network. We first compared the neural network with nonparametric regression and hierarchical model, then discuss Gradient descent as the usual training algorithm.

1 Nonparametric regression

Suppose $Y \in \mathbb{R}$ and we are trying to nonparametrically fit the regression function

$$\mathbb{E}Y|X = f_*(X)$$

A common approach (particularly when p is small) is to specify a fixed basis, $(\phi_j)_{j=1}^{\infty}$ and a tuning parameter J

We follow this prescription:

1. Write[1]
$$f_*(X) = \sum_{j=1}^{\infty} \beta_j \phi_j(x)$$
where $\beta_j = \langle f_*, \phi_j \rangle$
2. Truncate this expansion[2] at J
$$f_*^J(X) = \sum_{j=1}^{J} \beta_j \phi_j(x)$$
3. Estimate β_j with least squares

This approach have several weaknesses: The basis is fixed and independent of the data. If p is large, then nonparametrics doesn't work well at all because of curse of dimensionality. If the basis doesn't 'agree' with f_*, then J will have to be large to capture the structure. What if parts of f_* have substantially different structure? Then this global approach won't generalize well for local structure.

An alternative would be to have the data tell us what kind of basis to use.

2 High level overview of neural network

Neural networks are models for supervised learning. Linear combinations of features are passed through a non-linear transformation in successive layers. At the top layer, the resulting latent factors are fed into an algorithm for predictions, most commonly via least squares or logistic regression.

[1] Technically, f_* might not be in the span of the basis, in which case we have incurred an irreducible approximation error. Here, I'll just write f_* as the projection of f_* onto that span

[2] Often higher j are more rough $\Rightarrow$ this is a smoothness assumption

Letting $\mu(x) = \mathbb{E}Y|X = x$, and writing h as the link function, a neural network can be phrased as:

$$h(\mu(x)) = \beta_0 + \sum_{j=1}^{J} \beta_j \sigma(\alpha_{j0} + \alpha_j^\top x)$$

The main components are

- The derived features $Z_j = \sigma(\alpha_{j0} + \alpha_j^\top x)$ and are called the hidden units
  - The function σ is called the activation function and is very often $\sigma(u) = (1 + e^{-u})^{-1}$, known as the sigmoid function
  - The parameters α_j are estimated from the data. This is the main difference from a basis expansion approach, the basis functions are estimated from the data.
- The β are the coefficients of the regression
- The number of hidden units J is a tuning parameter

3 Neural network as a Hierarchical model

A neural network can be phrased as a hierarchical model

$$\begin{aligned} Z_j &= \sigma(\alpha_{j0} + \alpha_j^\top X) \quad j = 1, \dots J \\ W_g &= \beta_{g0} + \beta_g^\top Z \quad g = 1, \dots G \\ \mu_g(X) &= h^{-1}(W_g) \end{aligned}$$

The output depends on the application, where we map W_g to the appropriate space:

- Regression: The link function is $h(x) = x$ and we directly produce predictions of Y (here, $G = 1$)
- Classification: If there are G classes, we are modeling the probability of $Y = g$ and h is such that

  $$\hat{\mu}_g(X) = \frac{e^{W_g}}{\sum_{g'=1}^{G} e^{W_{g'}}} \quad \text{and} \quad \hat{Y}(X) = \operatorname*{argmax}_g \hat{\mu}_g(X)$$

  This is called the softmax function for historical reasons

4 Training neural networks

The neural networks have many unknown parameters.

They are usually called weights in this context.

These are

- α_{j0}, α_j for $j = 1, \dots, J$ (total of $J(p+1)$ parameters)

- β_{g0}, β_g for $g = 1, \ldots, G$ (total of $G(J+1)$ parameters)

Total parameters: $\asymp Jp + GJ$

The most common loss functions are

- Regression:
$$\hat{R} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$
- Classification: Cross-entropy
$$\hat{R} = -\sum_{i=1}^{n} \sum_{g=1}^{G} Y_{ig} \log(f_g(X_i))$$
  - Here, Y_{ig} is an indicator variable for the g^{th} class. In other words $Y_i \in \mathbb{R}^G$
  - With the softmax + cross-entropy, neural networks is a linear multinomial logistic regression model in the hidden units

5 Neural networks: Back-propagation

The usual approach to minimizing $\hat{R}$ is via gradient descent, known as back propagation.

Due to the hierarchical form, derivatives can be formed using the chain rule and then computed via a forward and backward sweep.

For squared error, let $\hat{R}_i = (Y_i - \hat{Y}_i)^2$

Then

$$\frac{\partial \hat{R}_i}{\partial \beta_j} = -2(Y_i - \hat{Y}_i) Z_{ij}$$
$$\frac{\partial \hat{R}_i}{\partial \alpha_{jk}} = -2(Y_i - \hat{Y}_i) \beta_j \sigma'(\alpha_0 + \alpha_j^\top X_i) X_{ik}$$

Given these derivatives, a gradient descent update can be found

$$\hat{\beta}_j^{t+1} = \hat{\beta}_j^t - \gamma_t \sum_{i=1}^{n} \frac{\partial R_i}{\partial \beta_j}\bigg|_{\hat{\beta}_j^t} \quad \text{This later evaluation only matters if } h(x) \neq x$$
$$\hat{\alpha}_{jk}^{t+1} = \hat{\alpha}_{jk}^t - \gamma_t \sum_{i=1}^{n} \frac{\partial R_i}{\partial \alpha_{jk}}\bigg|_{\hat{\alpha}_{jk}^t}$$

γ_t is called the learning rate, this needs to be set

Returning to

$$\frac{\partial \hat{R}_i}{\partial \beta_j} = -2(Y_i - \hat{Y}_i) Z_{ij} \qquad = a_i Z_{ij}$$
$$\frac{\partial \hat{R}_i}{\partial \alpha_{jk}} = -2(Y_i - \hat{Y}_i) \beta_j \sigma'(\alpha_0 + \alpha_j^\top X_i) X_{ik} \qquad = b_{ji} X_{ik}$$

Direct substitution of a_i into b_{ji} gives

$$b_{ji} = a_i \beta_j \sigma'(\alpha_0 + \alpha_j^\top X_i)$$

These are the back-propagation equations.

The updates given by the gradient decent can be operationalized via a two-pass algorithm:

1. Forward pass: Current weights are fixed and predictions $\hat{Y}_i$ are formed
2. Backward pass: The a_i are computed, and then converted (aka back-propagated) to get b_{ji}

And these updated quantities are used to take a gradient descent step.

6 Neural networks: Other algorithms

There are a few alternative variations on the fitting algorithm.
Many are using more general versions of non-Hessian dependent optimization algorithms like conjugate gradient.

The most popular are:

- Resilient back-propagation (with or without weight backtracking)
- Modified globally convergent version