Introduction to Neural Networks

Ryan Miller

Grinnell College
Statistics

Review of Logistic Regression

Logistic regression uses a set of features, $X_1, \ldots, X_p$, to predict a binary outcome, Y, using the following structure:

$$y_i = Bern(\pi = g(z_i)) \text{ where } g(z_i) = \frac{1}{1+exp(-z_i)}$$

Here $z_i = \hat{w}_0 + \hat{w}_1 x_{i2} + \hat{w}_2 x_{i2} + \ldots$ is the *linear predictor* for the i^{th} observation.

Review of Logistic Regression

The model's weights, $\{w_0, w_1, \ldots, w_p\}$, are found by optimizing the cross-entropy cost function:

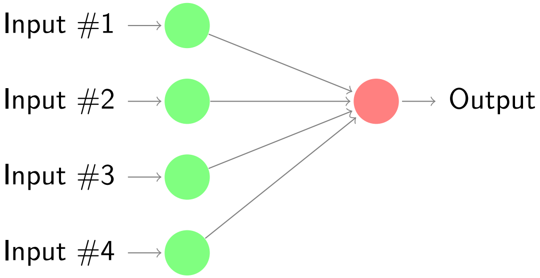
$$\text{Cost} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i log(g(z_i)) + (1 - y_i) log(1 - g(z_i))\right)$$

This optimization relies upon differentiating the cost function with respect to the unknown weights, which we can express using chain rule:

$$Gradient = \frac{\partial Cost}{\partial g} * \frac{\partial g}{\partial \mathbf{z}} * \frac{\partial \mathbf{z}}{\partial \mathbf{w}}$$

Review of Logistic Regression

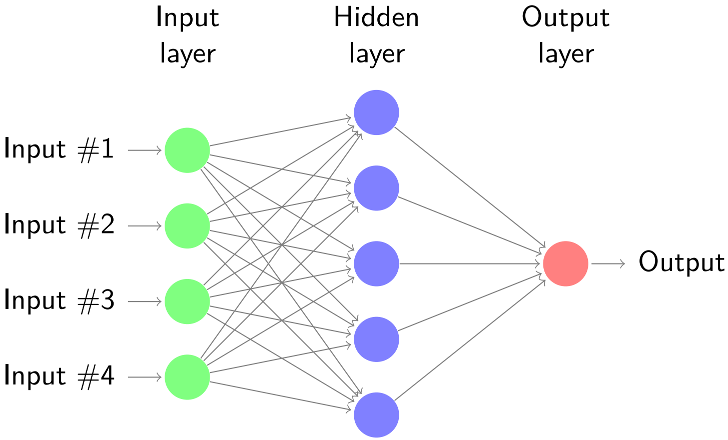
- In logistic regression, a linear combination of features is passed into the sigmoid function to be mapped to output, $\hat{Y}$
  - In this setting, we may call the sigmoid function an *activation function* (shown in red)



Neural Networks

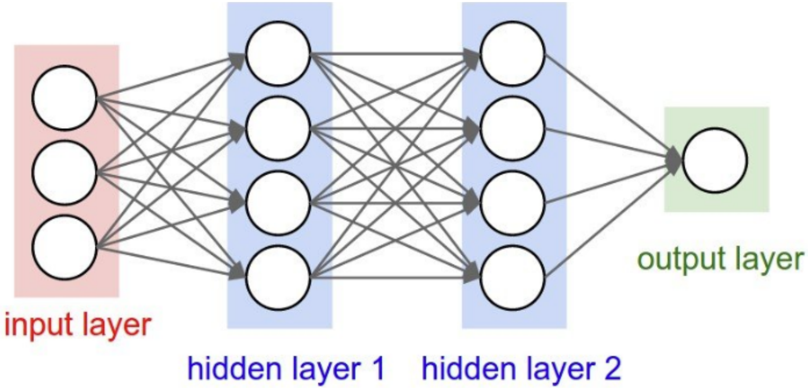
- In logistic regression, the observed features are weighted then passed into the sigmoid function and mapped to an output
- Neural networks derive new features through a similar process
  - That is, weighted combinations of observed features are passed into an activation function resulting in a *neuron* (or *hidden unit*)
- We can set up the structure of our model to contain any number of neurons
  - The model's neurons form a *hidden layer* of new features
  - A weighted combination of these neurons can then be passed into another activation function to predict the output
  - This structure is a *single layer* neural network (see next slide)

Single Layer Neural Networks



Network Depth

Our previous example used a single hidden layer, but in practice we can add more hidden layers:



Neural Nets vs. Logistic Regression

Logistic regression can be expressed as:

$$\hat{y}_i = g(\mathbf{x}_i)$$

Similarly, we could express a single layer neural network as:

$$\hat{y}_i = g(f(\mathbf{x}_i))$$

And a neural network with 2 hidden would be:

$$\hat{y}_i = g(f(h(\mathbf{x}_i)))$$

Notation

Because neural networks can contain many hidden layers, we'll introduce the following notation to keep track of the model's structure:

- $\mathbf{x}_i$ will remain the p-dimensional vector of input features (ie: the i^{th} row in our data, if it's in a tabular format)
- Superscripts, such as $\mathbf{w}^{(1)}$, will indicate the layer of object
- $\mathbf{z}^{(i)}$ will indicate the linear combination of weights and inputs in a particular layer
- $\mathbf{a}^{(i)}$ will indicate the activated output of a particular layer
- b will be used to indicate bias terms in linear combinations

Simple Example

Consider a single input feature, X_1, and a neural network with two hidden layers that each contain only a single neuron:

$$b_1^{(1)} + w_1^{(1)} X_1 = z_1^{(1)} \rightarrow g(z_1^{(1)}) = a_1^{(1)}$$

The output of the first (and only) neuron in our first hidden layer is $a_1^{(1)}$. The model then uses this output as an input to the next hidden layer:

$$b_1^{(2)} + w_1^{(2)} a_1^{(1)} = z_1^{(2)} \rightarrow g(z_1^{(2)}) = a_1^{(2)}$$

- A similar process repeats once more, yielding $\hat{Y} = a_1^{(3)}$

Grinnell College
Statistics

Learning the Parameters

Similar to logistic regression, we can use the cross-entropy cost for binary/categorical Y:

$$\text{Cost} = -\frac{1}{n}\sum_{i=1}^{n}\big(y_i log(\hat{y}_i)) + (1 - y_i)log(1 - \hat{y}_i)\big)$$

- We can use gradient descent to optimize the model's weights and biases
- This requires use to find the gradient vector, but what are the components of this vector?

Learning the Parameters

Let's first use chain rule to solve for gradient vector component $\frac{\partial Cost}{w_1^{(3)}}$:

$$\frac{\partial Cost}{w_1^{(3)}} = \frac{\partial Cost}{\hat{y}} \frac{\partial \hat{y}}{z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial w_1^{(3)}}$$

This works because $\hat{y}$ is a function of $z_1^{(3)}$ (sigmoid), and $z_1^{(3)}$ is a function of $w_1^{(2)}$

Learning the Parameters

For our simple example:

- $\frac{\partial Cost}{\hat{y}} = \frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$
- $\frac{\partial \hat{y}}{z_1^{(3)}} = g(z_1^{(3)})(1 - z_1^{(3)})$
- $\frac{\partial z_1^{(3)}}{\partial w_1^{(3)}} = a_1^{(3)}$

Notice how calculating this component of the gradient requires us to pass data, X_1, through the network to obtain the quantities $z_1^{(3)}$, $a_1^{(2)}$ and $\hat{y}$

Grinnell College
Statistics

Next, let's look at the gradient vector component $\frac{\partial Cost}{w_1^{(2)}}$:

$$\frac{\partial Cost}{w_1^{(2)}} = \frac{\partial Cost}{\hat{y}} \frac{\partial \hat{y}}{z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial w_1^{(2)}}$$

- This is similar to our previous expression after realizing $a_1^{(2)}$ is a function of $w_1^{(1)}$
- Note that gradient components for each bias term are calculated similarly

Back-propogation

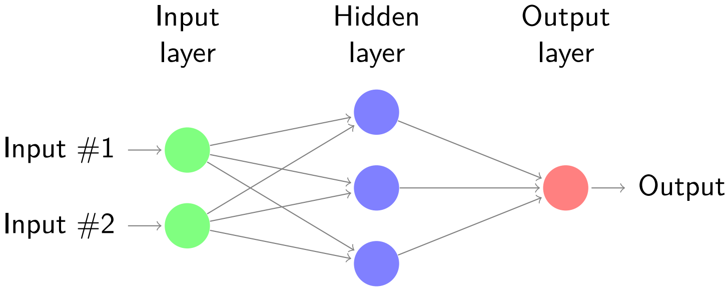
- The gradient components of parameters closer to the input layer reuse quantities that were calculated for components closer to the network's output
  - $\frac{\partial Cost}{\hat{y}}$ and $\frac{\partial \hat{y}}{z_1^{(3)}}$ in our example
- This makes it beneficial to work backwards through the model when calculating the components of the gradient vector
  - Thus, the application of chain rule to find the gradient of a neural network is often called the *back-propagation algorithm*

Forward-propogation

- You'll also hear the term *forward-propagation* (or *forward pass*) referring to the calculation of the cost function function for an observation (or batch of observations)
- As we previously mentioned, the gradient requires several intermediate quantities that are calculated during forward-propagation
  - Thus, the process for optimization begins by feeding an observation into the existing network (forward-propagation), then updating the network's parameters via back-propagation

Another Example

Now let's suppose our input layer contains two features, X_1 and X_2, or $\mathbf{x}$, and our model contains one hidden layer with three neurons:



How many weights and biases are needed as parameters in this model?

Another Example

The first neuron in the first hidden layer is given by:

$$b_1^{(1)} + w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 = z_1^{(1)} \rightarrow g(z_1^{(1)}) = a_1^{(1)}$$

The second by:

$$b_2^{(1)} + w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 = z_2^{(1)} \rightarrow g(z_2^{(1)}) = a_2^{(1)}$$

And the third is defined similarly.

Another Example

In matrix notation:

$$\mathbf{z}^{(1)} = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

and

$$\mathbf{a}^{(1)} = g(\mathbf{z}^{(1)})$$

- As you might expect, we can then find the necessary pieces of the back-propagation algorithm using chain rule and matrix calculus shortcuts
- We'll largely rely on software (`autograd`) to handle this for us, with the exception of one homework question

Activation Functions

Most modern neural networks prefer the *ReLU* (rectified linear unit) activation function to the sigmoid function because it can be computed and stored more efficiently:

$$\begin{aligned} g(z) &= 0 && \text{if } z < 0 \\ g(z) &= z && \text{if } z \geq 0 \end{aligned}$$

Grinnell College
Statistics

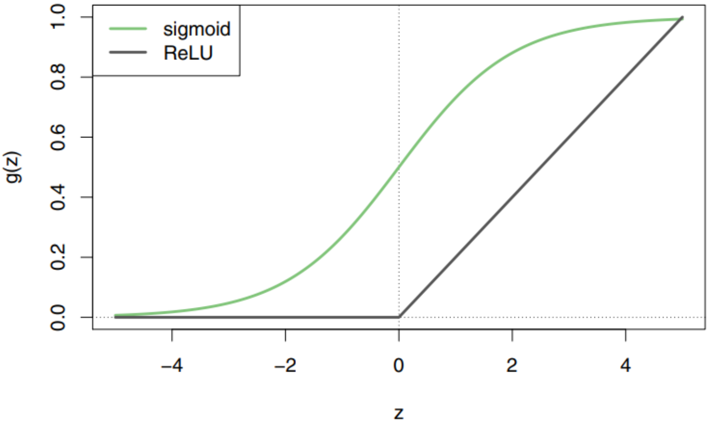
Activation Functions

Most modern neural networks prefer the *ReLU* (rectified linear unit) activation function to the sigmoid function because it can be computed and stored more efficiently:

$$\begin{aligned} \mathrm{g(z)} &= 0 && \text{if } z < 0 \\ \mathrm{g(z)} &= z && \text{if } z \geq 0 \end{aligned}$$

The derivative of ReLU function is simple (albeit discontinuous), as it's 1 if $z > 0$ and 0 otherwise. Software packages will take the derivative at $z = 0$ to be zero to promote greater sparsity.

ReLU vs. Sigmoid



Note: the ReLU function is scaled by 1/5 in this example for ease of comparison. The function is scale invariant when used as an activation function in a neural network.

Remarks on Network Depth

- Neural networks first became popular in the 1980s, but in the 1990s methods like random forests, boosting, and support vector machines received far greater attention
  - This was partly due to the computational challenges of neural networks and partly due to misunderstandings related to network depth

Remarks on Network Depth

- Neural networks first became popular in the 1980s, but in the 1990s methods like random forests, boosting, and support vector machines received far greater attention
  - This was partly due to the computational challenges of neural networks and partly due to misunderstandings related to network depth
- In the 2000s, deep neural networks (ones with many hidden layers) were found to be very success for image classification
  - In 2012, a deep neural network architecture named "AlexNet" led to a boom in *deep learning* by winning the ImageNet recognition challenge with accuracy of 84.7% (10.8% better than the nearest competitor)
  - Network depth combined with the use of GPUs for efficient training on massive datasets led to this performance

Intuition on the Role of Hidden Layers

- Why do deeper networks perform better on certain types of data, such as images?

Intuition on the Role of Hidden Layers

- Why do deeper networks perform better on certain types of data, such as images?
  - Intuitively, each hidden learning is learning features that are derived from the previous layer
- Hidden layer 1 learns patterns that are simple linear combinations of the inputs (perhaps vertical and horizontal edges of varying lengths and directions)
- Hidden layer 2 learns patterns that are linear combinations of the features identified in hidden layer 1 (perhaps simple shapes, curves, etc.)
- The next hidden layer can then learn patterns that are combinations of those shapes, curves, etc.
  - At some point, the complexity of the current features provides enough information to make accurate predictions

Intuition on the Role of Hidden Layers

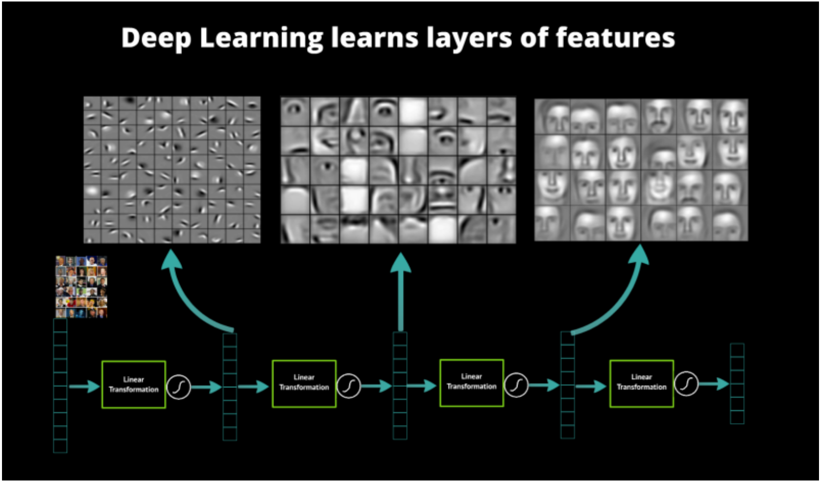


Image Credit: https://www.datarobot.com/blog/a-primer-on-deep-learning/

Grinnell College
Statistics

Closing Remarks

- Neural networks involve a lot of parameters and can learn very complex relationships, but this generally requires a lot of training data
- The simple networks we discussed today tend not to be commonly used
  - They aren't well-equipped to handle spatial structures, which make them less effective at applications involving image/textual data
  - They tend to overfit "flat" or "tabular" data to a greater extent than methods like random forests or boosted ensembles
- Next we'll learn about *convolutional neural networks*, a variation utilizes spatial relationships among features and excels in computer vision applications