STAT 452/652: Statistical Learning and Prediction

Owen G. Ward
owen_ward@sfu.ca

Fall 2023

These notes are largely based on previous iterations of this course, taught by Prof. Tom Loughin and Prof. Haolun Shi.

18 Neural Networks and Deep Learning

(Reading: ISLR 10.1 - 10.3, 10.7(optional))

Goals of this section

- Have seen several methods which can take linear combinations of variables

$$Z = \phi_1 X_1 + \ldots + \phi_p X_p$$

  - Principal Components Regression
  - Partial Least Squares
- Neural Networks allow for an extremely flexible way to do this
  - Complex structure, hard/impossible to interpret
  - Can work in very general settings
- Have massive numbers of parameters
- Can only outperform existing methods in certain scenarios
  - Often need massive amounts of data to beat much simpler methods

18.1 The simplest Neural Network

- Input variables X "fed" into a HIDDEN LAYER OF NODES, see Figure 1
- This takes a transformation of the inputs,

$$A_k = h_k(X) = g\left(w_0 + \sum_{j=1}^{p} w_{kj} X_j\right)$$

- g is some (normally) non-linear function, known as ACTIVATION FUNCTION
- These A_k are then combined using (for example) linear regression to give

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

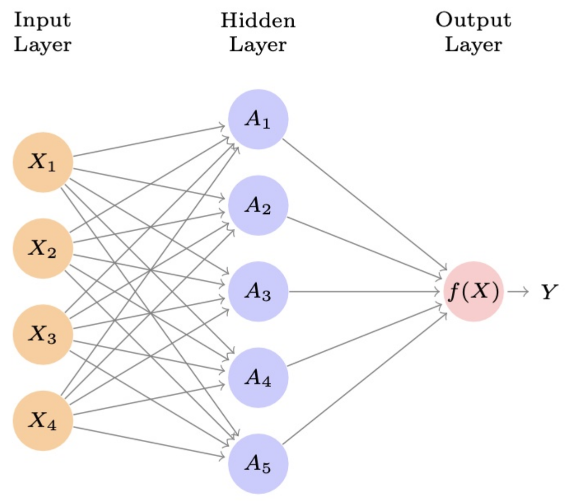

FIGURE 10.1. *Neural network with a single hidden layer. The hidden layer computes activations $A_k = h_k(X)$ that are nonlinear transformations of linear combinations of the inputs $X_1, X_2, \ldots, X_p$. Hence these A_k are not directly observed. The functions $h_k(\cdot)$ are not fixed in advance, but are learned during the training of the network. The output layer is a linear model that uses these activations A_k as inputs, resulting in a function $f(X)$.*

Figure 1: A simple Neural Network with a single hidden layer, from ISLR.

- There are a lot of moving parts here
  - Need to choose K, the number of nodes in the hidden layer
  - The choice of g. Common to use the SIGMOID, given by

$$g(z) = \frac{e^z}{1 + e^z}$$

  - ReLU a recent popular choice

$$g(z) = \begin{cases} z \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$$

  - If $g(z) = z$, this is actually just a complicated way of writing a linear regression!
  - Many parameters, even in this simple model. If we have p predictors and K nodes in the hidden layer, have $K(p+1)$ parameters in the hidden layer, and $K+1$ more for the output

- This allows very complex relationships between variables, can get interactions without including them in the original model
- How these models are actually fit is complicated, will discuss briefly later. Want to minimize squared error loss,
$$\sum_{i=1}^{n}(y_i - f(x_i))^2$$
- Important to NORMALIZE variables before using them in a neural network. More advanced software can do this as part of the model fitting

Example: Simple Neural Network (`Sec18_Neural_Nets.R`)

Fit this simple example using Keras to the data which was included in the previous version of the slides. Will also include the 'nnet' implementation. Later in the same file can include a much more complex model, see how it overfits horribly for this problem. Mention that you can try tune these simple neural network models, which is maybe harder to do for the keras versions.

18.2 Deep Learning, Convolutional Neural Networks

The simple feed-forward networks above combine every node in the previous layer into the nodes in the next layer. There are lots of ways we can make these models more complex and flexible.

- Will discuss some examples of more complex neural networks
- In theory, a single layer with very large K can approximate any function
- Normally instead add more layers to get a more flexible model
  - Deep Learning = (Basically) Multiple hidden Layers and some different transformations between these layers
  - Convolutional Neural Networks (CNN's) a form of deep neural network which has been widely successful for classification of image data
  - CNN's similar to just adding more layers to model, but consider more complex operations on the hidden layers also
  - Suppose instead of a single hidden layer, we consider 2 hidden layers, with $K = 256$ in first and $K = 128$ in second, with the goal of image classification
    - Common to use powers of 2 for the number of nodes in hidden layer
- This can lead to a model with **millions** of parameters
- Can easily overfit to the training data
- Are ways to prevent overfitting, somewhat similar to pruning regression trees
- Can also use different types of h, to capture different structure. CNNs use very specific types of h

Lets see a classic image classification problem and how CNN's can be used to solve this.

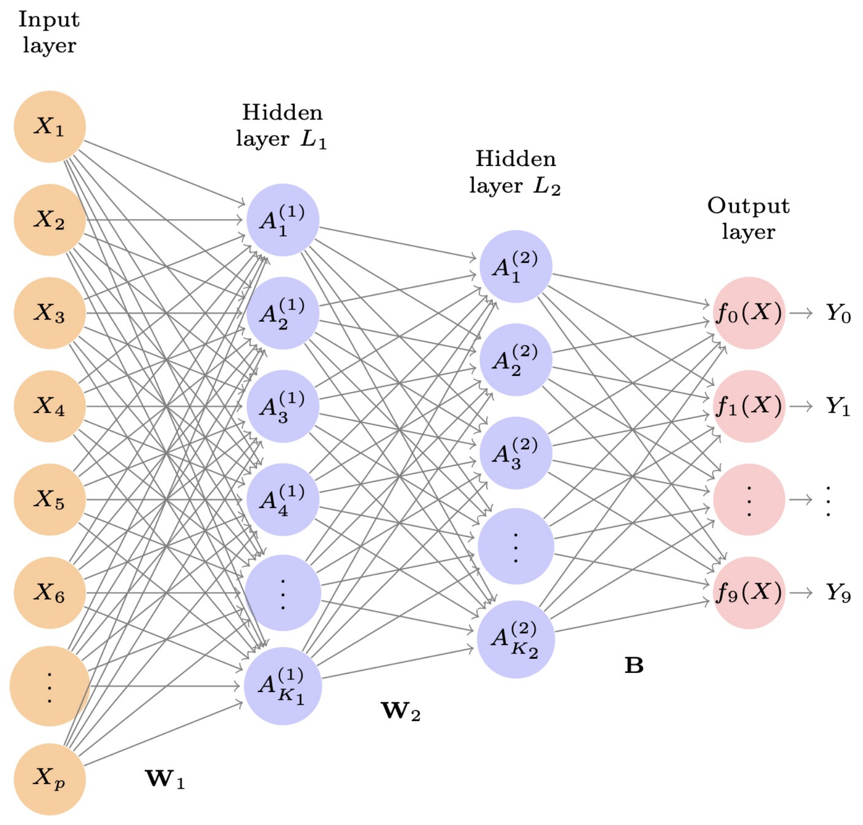

Figure 2: A Neural Network with two hidden layers, from ISLR. Normally, you decrease the number of nodes in a layer when you add new ones.

MNIST Data

The MNIST dataset is a classic in machine learning. This consists of images, each of size 784 pixels (28 by 28). These are grayscale images, so each pixel is a number between 0 and 255, corresponding to the shade of gray of that pixel. These images are of handwritten digits, from 0-9, so they take on 10 different possible values. We want to use the information in the pixels to predict which digit is represented in each image. Each image is 28 by 28 pixels, where each pixel is a number (0-255) corresponding to how dark that pixel is. This is a problem with a very clear application, such as automatically sorting mail. We have a large training set with about 60,000 images and we want to learn a model on them to predict on 10,000 test images.

A convolutional neural network works well on problems such as this. It uses some specialised operations between subsequent hidden layers to achieve this. These are widely used in deep learning architectures which give good performance. They use two specific types of layers in the neural network repeatedly: Convolutional and Max-Pooling Layers.

Convolutional Layers

- Takes the input to that layer of the network and performs specific matrix multiplications
- The specific matrix used for this is defined in a way to try **highlight** features of the image
- *Convolve* the original data.

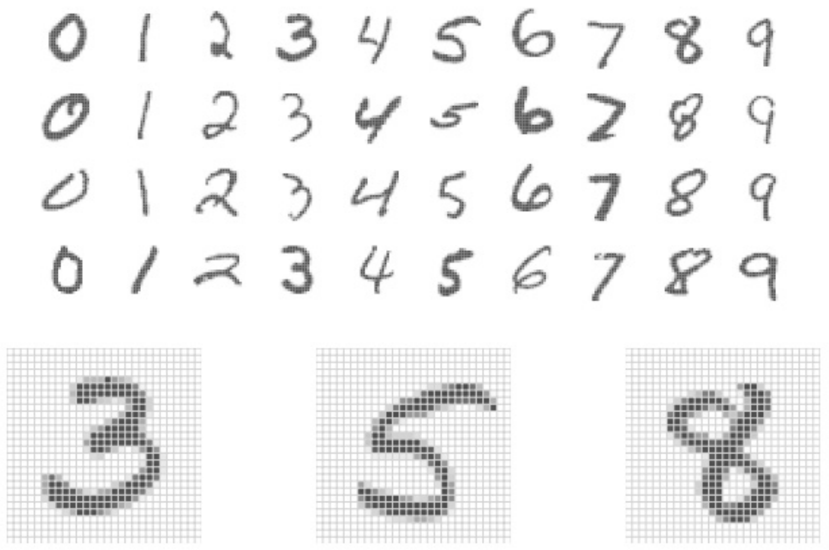
Figure 3: Some digits from the MNIST dataset. Figure taken from ISLR.

- We talked about feature engineering earlier. This is basically the computer doing feature engineering for you
- Learns the transformation from the training data
- Specific constraints on the parameters here also to make it reasonably feasible
- Used on different parts of the image, to pick out local features
- Can use several convolutional filters in a given layer. Each will give you an output of the same size as the input data! So they actually make the data larger

For each image in the MNIST example, the raw data for *each observation* is a 28×28 matrix of data. Passing this through a convolutional layer with 2 filters would give you **two** 28×28 matrices of data for each observation. We need to reduce the size of the data after we do the convolution. This is known as pooling.

Max Pooling

- After a layer with several convolutional filters, max pooling is a way to discard some of that data
- It takes the output from the convolutions and simply takes the maximum from each small $K \times K$ block, throwing away all other values
- If we use max pooling with $K = 2$, then this will shrink the dimension of the data by 2 in both height and width
- Its quite common to use a ReLU step as part of the convolutional filter, so all the entries will already be non-negative

For example,

$$\text{max-pool}\left(\begin{bmatrix}1 & 3\\7 & 4\end{bmatrix}\right) = 7,$$

the largest entry in that 2×2 block of data.

Repeating these The ability of these methods really shines when we repeatedly use them. For example, we can use a convolutional layer, followed by a max pooling layer repeatedly.

$$\begin{aligned}&\text{Raw Data} \to \text{Convolutional Filters} \to \text{Max Pool} \to\\ &\qquad\text{Convolutions} \to \text{Max Pool} \to\\ &\qquad\qquad \ldots \to \text{Final Layer} \to \text{Predictions}\end{aligned}$$

Even with only a small number of filters, this can lead to a massive amount of parameters in the neural network. These models can have incredible performance on image classification tasks.

Example: Convolutional Neural Network (`Sec18_CNN.R`)

Here we will show the code to fit a small CNN to the MNIST data. Note that running these models requires further installation on your computer that we haven't really talked about. We will use a CNN with 2 convolutional layers, each followed by a max pooling layer. The first convolutional layer will have 16 filters while the second will have 32. We also use a dropout layer, which will be defined later. A summary is available from the code.

```
model <- keras_model_sequential() %>%
        layer_conv_2d(filters = 16, kernel_size = c(3, 3),
                        padding = "same", activation = "relu",
                        input_shape = c(28, 28, 1)) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_conv_2d(filters = 32, kernel_size = c(3, 3),
                        padding = "same", activation = "relu") %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_flatten() %>%
        layer_dropout(rate = 0.5) %>%
        layer_dense(units = 256, activation = "relu") %>%
        layer_dense(units = 10, activation = "softmax")

summary(model)

Model: "sequential"
______________________________________________________________________________
 Layer (type)                        Output Shape                     Param #
==============================================================================
 conv2d_1 (Conv2D)                   (None, 28, 28, 16)               160
 max_pooling2d_1 (MaxPooling2D)      (None, 14, 14, 16)               0
 conv2d (Conv2D)                     (None, 14, 14, 32)               4640
 max_pooling2d (MaxPooling2D)        (None, 7, 7, 32)                 0
 flatten (Flatten)                   (None, 1568)                     0
 dropout (Dropout)                   (None, 1568)                     0
 dense_1 (Dense)                     (None, 256)                      401664
 dense (Dense)                       (None, 10)                       2570
```

```
================================================================================
Total params: 409,034
Trainable params: 409,034
Non-trainable params: 0
________________________________________________________________________
```

This CNN model, with more than 400,000 tuneable parameters, achieves excellent performance, with accuracy on the test data of 99%.

We can also fit a random forest to this data, with a bit of additional work. Note that this is much slower to run for this problem, so we don't try to tune it. However even a very small random forest can get 95% accuracy for this problem. Compared to the machine learning problems people work on now, this is really easy, and this CNN is actually a pretty simple neural network model!

18.3 Notes on Neural Networks

Fitting These Models

- Even the very simple implementation of a neural network we used from `keras` on the prostate data has over 50 parameters
- This is much more complex than the ensemble methods we had previously
- Still want to minimise the MSE (regression) or cross-entropy (classification) on the training data
  - Will call either of these quantities the **loss** interchangably below
- In simple models with few parameters there is often only one set of values which will minimise the loss.
- With neural networks we can lots of estimates which look like solutions
- Easy to find a good *local* solution, but might not be the best overall solution.
  - **Regularization** of the parameters one way to avoid this
  - Prevents overfitting
- Idea called **backpropagation** used to get these estimates
  - Uses the chain rule to "pass" through the network
- The term **Epoch** commonly used
  - Do not use all data to update the parameters each time, just a sample
  - One epoch means you have used samples which have the same total size as the training data
  - Note that the previous examples used 100 epochs by default, using the training data repeatedly!

Regularization To prevent a neural network overfitting, several ways to make it letter better parameter estimates.

- The `nnet` package uses `decay`, which shrinks the estimates of the parameter estimates towards 0.
  - Sort of similar to how Ridge regression does this
  - Authors recommend setting the `decay` parameter between .0001 and .01.
  - More shrinkage may be needed with larger K (the number of hidden nodes)
- The more complex neural network functions in `keras` provide other tools
- The choice of `optimizer` can have a `learning_rate` parameter, which can be used to prevent overfitting
- We used a `dropout` layer in the CNN example.
  - Randomly set some of the nodes to be 0 in each hidden layer
  - This can also be thought of as a form of regularization, actually inspired by random forests

What to take away from this

Neural networks can be extremely good models for both numeric prediction and classification.

- They need massive data to work well
- Figuring out how to fit these models to make them work is very much an art rather than a science.
  - There are many many many different ways to tinker with deep learning models, which can give you slight improvement
  - They might not be worth the extra effort
- There are few theoretical guarantees on why any of these methods work in practice.
- There is basically no interpretability to these models. If you want to use one of these models, good luck trying to explain it to someone!
- It is very easy to overfit with these complicated models!

Problem Set 20: Neural Networks and Deep Learning

Application

Refer to the Air Quality data described previously, and the analyses we have done with `Ozone` as the response variable, and the five explanatory variables (including the two engineered features).

Use Neural Nets (NN) to model the relationship between `Ozone` and all five explanatories as specified below. For the problems below use the **nnet** package.

1. Create a matrix of the five explanatory variables. Rescale each of them to lie between 0 and 1 and same them as another matrix. **Print a `summary()` of each object to confirm that you have scaled properly.**

2. Fit 4 NNs nets using only `Temp` and `Wind`, using each combination of 2 and 6 hidden nodes with 0.001 and 1 shrinkage; i,e,; (2,0.001), (2,1), (6,0.001), (6,1)

   (a) Refit each one manually 20 times or more and compute the sMSE each time.

      i. **Report the sMSE for the optimal fit for each model.**

      ii. Comment on the stability of fits for different models. In other words, **which models were most/least consistent with the sMSE values produced by different fits?**

   (b) Make a 3-D plot of each model's fit.

      i. **Report a screenshot of each fit, rotated to roughly the same angle each time to show a good comparison of the fits.** (I find it best to look down through the corner with low temp and high wind, so that the high ozone values are in the back.)

      ii. **Comment separately on how increasing number of nodes or increasing shrinkage appears to affect the fits.**

3. Now try tuning the NN *on these two variables* using 5-fold CV. Use a grid of (1, 3, 5, 7, 9) nodes and (.001, .1, .5, 1, and 2) decay. Refit each combination of parameters 10 times to find the best sMSE for that combination.

   (a) Compute the overall MSPE for each combination, and add 95% confidence intervals. **Take square roots and report the results.**

   (b) **Show relative root-MSPE boxplots of the five splits.**

   (c) Use these results to **identify (i) the best combination, and (ii) other combinations that seem to perform just as well.**

   (d) Is further tuning necessary? **That is, is the best model (i) at a boundary or (ii) quite different from neighbouring models?**

4. **BONUS (This will take a long time and some extra programming, so it is optional. But it is actually closer to how I would conduct a comparison of methods for a real data set.)** Add NN *on all variables* to the 10-fold CV comparison that has been used for LASSO, Ridge, and other methods. Use the same folds for NN that were used for the other methods. *Note that you will need to do some tuning withing each of the CV folds! That is, you will need to perform tuning* ***within*** *each 90% training set created by the 10-fold CV. Use one round of 5-fold CV for this, and refit each NN just 5 times to find the optimal fit.*

   (a) **For the each of the 10 folds, report the chosen best values of the NN tuning parameters.** (I expect that they will vary from one fold to the next)

   (b) **Compare the mean MSPE from the best-tuned NN to the other models tried so far.**

   (c) **ADD tuned NN to the relative MSPE boxplots made previously. Comment on how well it performs compared to other methods**