QBUS64840 Predictive Analytics

Forecasting with Neural Networks and Deep Learning 2

University of Sydney Business School

Table of contents

Learning objectives

- Know the methods used to train/estimate a neural network model, and the difficulties in training
- Know how to use a neural network for prediction with cross-sectional data and time series data

Neural Networks for cross-sectional data

Neural net for cross-sectional data

Suppose that the response Y is numerical.

From now on, we will use w to denote ALL the weights in the neural net. The output is $\eta(X, w)$.

The neural net model for regression is

$$Y \quad = \quad \eta(X, w) + \epsilon$$

where ϵ is an error term with mean 0 and variance σ^2. Often, we assume $\epsilon \sim \mathcal{N}(0,\sigma^2)$.

The least squares method can now be used to estimate the model parameters.

Neural net for forecasting

Least squares method for training

- Let $\{y_i, x_i = (x_{i1}, ..., x_{ip})^\top\}$, $i = 1, ..., n$ be the training dataset.
- The neural net regression model can be written as

$$y_i = \eta(x_i, w) + \epsilon_i, \quad i = 1, ..., n$$

- The parameters are $\theta = (w, \sigma^2)$.
- Define the loss function to be the sum of squared errors

$$\mathrm{Loss}(\theta) = \sum_i \ell_i(\theta), \;\; \ell_i(\theta) = \Big(y_i - \eta(x_i, w)\Big)^2$$

- LS minimizes $\mathrm{Loss}(\theta)$ to estimate θ.

Difficulties in training a neural net

We need to solve an optimization problem

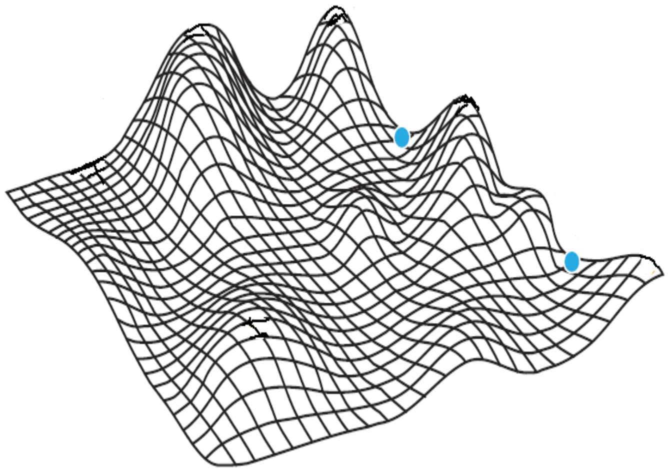
$$\text{find } \theta \text{ that minimizes } \mathrm{Loss}(\theta)$$

Difficulties

- There are a huge number of parameters
- The surface of the loss function of often highly multimodal
- We often need big data, so computational expensive

In most cases, neural net models are trained by the Stochastic Gradient Descent (SGD) method.

Multimodality issue



By Max Olson for FutureBlind

Gradient descent method for optimization

Suppose that we want to minimise a function $\mathrm{Loss}(\theta)$.

- Start from an initial $\theta^{(0)}$
- Update

$$\theta^{(t+1)} = \theta^{(t)} - a_t \nabla_\theta \mathrm{Loss}(\theta^{(t)}), \quad t = 0, 1, 2, \ldots$$

  until some convergence condition is met.

Here $\nabla_\theta \mathrm{Loss}(\theta)$ is the gradient vector of $\mathrm{Loss}(\theta)$.

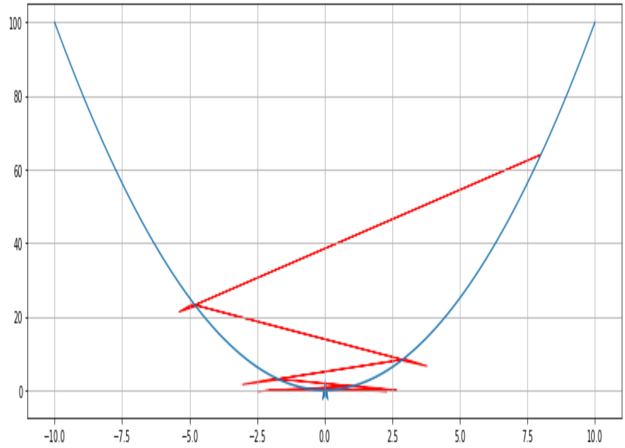
$a_t > 0$ is called learning rate or step size, such that

$$a_t \to 0, \quad \text{as} \quad t \to \infty$$

$\theta^{(t)}$ is guaranteed to converge to a local minima of $\mathrm{Loss}(\theta)$.

Gradient descent method for optimization

```
Successfully converged with a step size of 0.006022129282448782 after 16 iterations!
```



Stochastic gradient descent

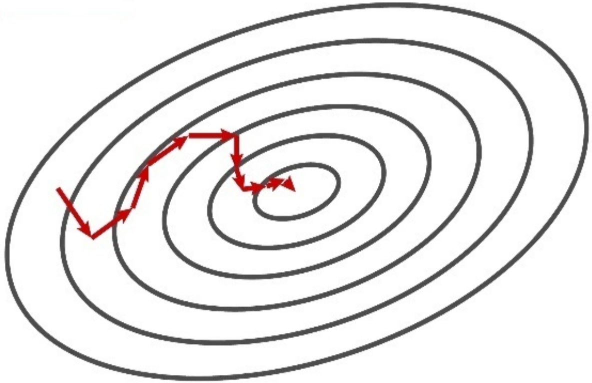
- For training a NN model, $\mathrm{Loss}(\theta) = \sum_{i=1}^{n} \ell_i(\theta)$,

$$\nabla_\theta \mathrm{Loss}(\theta) = \sum_{i=1}^{n} \nabla_\theta \ell_i(\theta)$$

- In deep learning, we often need BIG DATA, n is HUGE. So the gradient $\nabla_\theta \mathrm{Loss}(\theta)$ is a BIG sum and computationally expensive to compute. We often use stochastic gradient descent (SGD).
- In SGD, instead of using the exact/full-data gradient $\nabla_\theta \mathrm{Loss}(\theta)$, we use an estimate $\widehat{\nabla}_\theta \mathrm{Loss}(\theta)$ based on a subset of the data (of size $m \ll n$)

$$\widehat{\nabla}_\theta \mathrm{Loss}(\theta) = \frac{n}{m} \sum_{i \in \text{subset}} \nabla_\theta \ell_i(\theta)$$

Stochastic gradient descent



How to select learning rate a_t?

- In SGD, and stochastic optimization in general, the convergence depends enormously on the step size a_t.
- In DL in particular, a naive choice of a_t, such as $a_t = 1/t$, doesn't work at all
- a_t must be designed adaptively in a smarter way
  - select a_t based on the behavior of $\theta^{(0)}, \theta^{(1)}, \ldots, \theta^{(t-1)}$ so far
  - popular methods: ADAM, AdaGrad, etc.

Back-propagation algorithm

- All we need now is to compute the gradient of the neural net output $\nabla_\theta \ell(\theta)$. This is often a very long vector
- At the first glance, it seems difficult to compute this gradient vector, but luckily...
- An efficient algorithm for computing this is called back-propagation algorithm. This algorthim has been implelemted in Python.
- The back-propagation algorithm is crucial for the success of DL.

Practice recommendations

Practice recommendations

- It's not unusual that people have to spend a few days or even weeks and months to train a deep learning model
- If you can train a deep learning model successfully, then in most cases you beat conventional models (linear regression, logistic, LDA, etc) in terms of prediction accuracy
- But how to train a deep learning model successfully requires some effort. There are some implementation tricks that you might find useful in practice

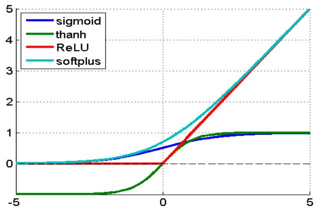
Practice recommendations

Data standardization

- The input/covariate data should be standardized before training your model
- Each numerical column in the training dataset should be standardized so that it has a zero sample mean and std. 1. Note: part of this set, called validation set, is used for tuning hyperparameters; the rest is still called training set.
- Often, no need to standardize binary columns

Practice recommendations

Activation function



- Sigmoid activation has long been used in neural nets, but it has a major drawback: the gradient vector becomes almost zeros when the neural net is deep.
- Rectified activation function : $h(S) = \max(0, S)$ stands out to be the option in many cases
- Designing activation functions is still a "hot" research topic.

Practice recommendations

Fix random seed

- All the computer can do when generating random variables is to generate a set of pseudo-uniform random numbers using a deterministic sequential rule
  - Start u_0
  - $u_n = \text{rule}(u_{n-1}),\ n = 1, 2, ...$
- The starting number u_0 is referred to as the random seed (this is an integer number).
- If you fix the random seed, then you fix randomness (yes, this sentence doesn't make sense philosophically!).
- It's advisable to fix the random seed in training deep learning models. Then all the results are reproducible.
- Typically in DL, the results will change then you change the random seed

Practice recommendations

Early stopping

- Because of their flexibility, DL models often overfit the data, even regularization priors are used.
- It's often observed that, if the model has too many layers/units, the training loss decreases steadily over the updates, but the validation loss starts increasing at some point.
- Therefore the updating procedure should be stopped if the validation loss is not decreased after a certain iterations (called patience)
- Then the set of model parameters with respect to the lowest validation loss is returned/used.

Practice recommendations

How to select the structure of the neural net?

How many hidden layers should we use? How many units in each layer?

- This is a challenging model selection problem. There is no definite answer.
- But there are basically three shapes to choose from: left-base pyramid (large to small layers), right-base pyramid (small to large layers) and retangular (roughly equal size layers)
- In many cases, a rectangular-shaped neural nets work well
- For the number of hidden layers, simply start from 1 layer, adding more until the validation loss doesn't get smaller.

Neural Networks for Time Series

Neural Networks for Time Series: Non-linear Autoregression

- For time series, we shall use lagged values of time series as inputs to a neural network and the output as the prediction
- This means, in general, the number of input neurons of a neural network is the number of the time lag and there is only one output neuron
- We will first consider feed-forward networks with one hidden layer.

Neural Networks for Time Series: Non-linear Autoregression

- We denote by NNAR(p, k) the NNs with p lagged inputs and k neurons in the hidden layer and with one output as the forecast
- For example NNAR$(12, 10)$ model is a neural network with the last 12 observations $(y_{t-1}, y_{t-2}, ..., y_{t-12})$ to fit y_t at any time step t, with 10 neurons in the hidden layer
- A NNAR$(p, 0)$ model is equivalent to ARIMA$(p, 0, 0)$ without the restrictions on the parameters to ensure stationary.

Neural Networks for Time Series: Training

- Consider the neural network NNAR(p, k). For a given section of time series $y_{t-p+1}, y_{t-p+2}, \ldots, y_t$ denote the output from NNAR(p, k) by $\widehat{y}_{t+1} = \eta(y_{t-p+1}, y_{t-p+2}, \ldots, y_t; w)$ where w collects all the weights in the network.
- Given training data $\{y_1, \ldots, y_n\}$, we form the training patterns as follow

$$\begin{aligned}
&y_1, y_2, \ldots, y_p \to y_{p+1} : \ \epsilon_{p+1} = (y_{p+1} - \eta(y_1, y_2, \ldots, y_p; w)) \\
&y_2, y_3, \ldots, y_{p+1} \to y_{p+2} : \ \epsilon_{p+2} = (y_{p+2} - \eta(y_2, y_3, \ldots, y_{p+1}; w)) \\
&y_3, y_4, \ldots, y_{p+2} \to y_{p+3} : \ \epsilon_{p+3} = (y_{p+3} - \eta(y_3, y_4, \ldots, y_{p+2}; w)) \\
&\qquad \vdots \\
&y_{n-p}, y_{n-p+1}, \ldots, y_{n-1} \to y_n : \ \epsilon_n = (y_n - \eta(y_{n-p}, y_{n-p+1}, \ldots, y_{n-1}; w))
\end{aligned}$$

- To estimate the weights w, we minimise

$$\min_w \left\{ \text{Loss}(w) = \sum_{i=p+1}^{n} \epsilon_i^2 \right\}$$

Neural Networks for Seasonal Time Series

- Except for the lagged data as inputs, it is useful to add the last observed values from the same seasons as inputs, for seasonal time series
- The notation NNAR$(p, P, k)_m$ means a model with inputs

$$(y_{t-1}, y_{t-2}, \dots, y_{t-p}, y_{t-m}, y_{t-2m}, \dots, y_{t-Pm})$$

  and k neurons in the hidden layer
- What is the input for NNAR$(5, 4, 10)_6$?

A simple Recipe

- Exploratory data analysis: Apply some of the traditional time series analysis methods to identify the lag values p and P (e.g. ACF).
- Split your data into two main sections: training section $\{Y_1, Y_2, ..., Y_n\}$ and validation section $\{Y_{n+1}, Y_{n+2}, ..., Y_T\}$. For example, you may use data in the first two years for training, the data in the third year as validation for a time series of three years data
- Define the neural network architecture

A simple Recipe

- Create the training patterns: E.g., for NNAR$(4, k)$, each training pattern will be five values, with the first four corresponding to the input neurons and the last one defining the prediction as the output unit. Here is all the data pattern for the training data $\{Y_1, Y_2, ..., Y_n\}$ for a NNAR$(4, k)$.

$$\begin{array}{rr}
\text{Training Pattern 1:} & Y_1, Y_2, Y_3, Y_4 \rightarrow Y_5 \\
\text{Training Pattern 2:} & Y_2, Y_3, Y_4, Y_5 \rightarrow Y_6 \\
\text{Training Pattern 3:} & Y_3, Y_4, Y_5, Y_6 \rightarrow Y_7 \\
\vdots & \vdots \\
\text{Training Pattern } n-4: & Y_{n-4}, Y_{n-3}, Y_{n-2}, Y_{n-1} \rightarrow Y_n
\end{array}$$

- Train the neural network on these patterns

A simple Recipe

- Forecast using the network on the validation set $\{Y_{n+1}, Y_{n+2}, ..., Y_T\}$: Here you will pass in four values as the input layer and see what the output node gets.

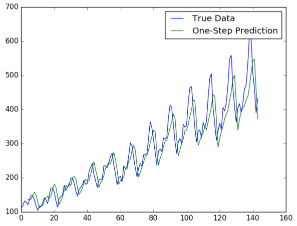
$$\begin{array}{ll}
\text{Validation Pattern 1:} & Y_{n-3}, Y_{n-2}, Y_{n-3}, Y_n \rightarrow \widehat{Y}_{n+1} \\
\text{Validation Pattern 2:} & Y_{n-2}, Y_{n-1}, Y_n, Y_{n+1} \rightarrow \widehat{Y}_{n+2} \\
\text{Validation Pattern 3:} & Y_{n-1}, Y_n, Y_{n+1}, Y_{n+2} \rightarrow \widehat{Y}_{n+3} \\
\vdots & \vdots
\end{array}$$

- Then, form the validation/test error.

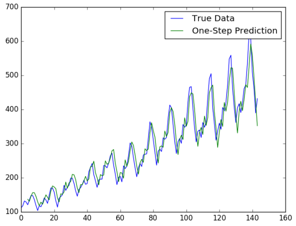
The Example

- Dataset: International Airport Arriving Passengers: in csv format
- Prepare Data: `Xtrain`, `Ytrain`, `Xtest` and `Ytest` with time lag 4
- Define the NN architecture:
  - `model = Sequential()`
  - `model.add(Dense(30, input_dim=time_lag, activation='relu'))`
  - `model.add(Dense(1))`
- This defines a network with 30 neurons on the hidden layer
- Test on `Lecture10_Example01.py`

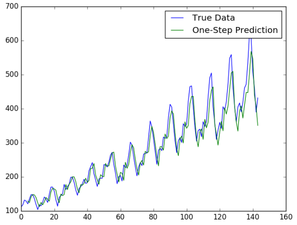
The example: Forecasting



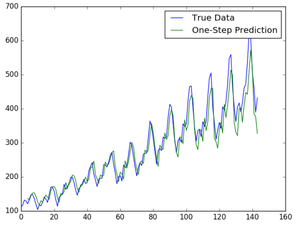
Hidden = 3



Hidden = 30



Hidden = 100



Hidden = 500