

Machine Deep Learning in a Nutshell

Modern machine learning method help people in making decision and prediction from data. These covered areas from physical, management science to finance.

Similar to CAD, Computer Aided Design, human is making use of computer for CADM, Computer Aided Decision Making. The development of modern machine learning methods is important to human endeavor by advancing human's knowledge to broader and deeper extends. Similar methods can be applied to engineering problem, product pricing setting problem or stock trading strategy decision making problem.

Here we will go through the basics and some advance topics in following sequences:

- 1) Predict from component factors (linear regression) as a simple ranking method.
- 2) Make classification decisions by extending linear regression to logistical regression.
- 3) Patterns classification and decision makings.
- 4) Common pitfalls. From social or physical phenomenon, digitized information to data analysis.

For session with the 'challenge' icon below, you can skip it without loss of reading continually but it will help you to understand the topic deeper for future's real-life applications.



Predict from component factors, linear regression


Human predicts, from knowledge of known factors to unknown future and make decision. We will look at sky and decide whether we should bring an umbrella. If the sky is dark and cloudy, most people will bring an umbrella. A bank base on a person's income, job nature, age and other component factors to make a decision of approving a credit card for him or not.

In simplest mathematical term, it can be described by a simple equation as following.

$$z = w_0 + w_1x_1 + w_2x_2 + w_3x_3 \quad \text{Equ 1.0}$$

For example, x_1 is income, x_2 is job nature and x_3 is age. We treat each factor as a predictor for the outcome z . For each factor, we weight it with some weights w_1, w_2, w_3 . Furthermore, when all the weighted stuffs add together, it must be greater than a threshold value for approving a credit card. Therefore if we select w_0 as a big enough negative number (say -100), then all the weighted sum $w_1x_1 + w_2x_2 + w_3x_3$ must greater than 100 to make the outcome z positive. If z is positive, the bank will approve the credit card. Equ 1.0 is called linear regression [3].

The challenge here is how to select the weights (w_i) such that above simple equation can be used for the bank's manager as a guiding rule for approving the credit card. Similarly, if we can use same approach to determine insurance premium instead of a approve or disapprove discrete decision, but the weights to achieve this will be difference for the credit card approval process. For more detail treatment on the

credit card example, please refer to [1, 2] . Sometimes we can find the weights by analytical matrix algebra methods, but I will show only numerical methods here, as it can be used for a broader spectrum of problems by modern software tools.

Most, if not all, of the machine deep learning methods use this as the basic unit to build highly complicated system from online store recommendation system to Google's Alpha Go and Alpha Zero which outperform human in several competitions. Today, similar methods are using from financial trading system, genes classification to

covid-19 virus classification system. Human's capabilities in understand the universes is extending to another limit never happened in our history before.



The system describe by Equ 1.0 can be build easy by Tensorflow Keras as following:

```
Dense(units=1, name="z")
```

Google Tensorflow called it Dense layer and it is one unit in a layer. Pictorial, it can be represented by a circle with weights around it.

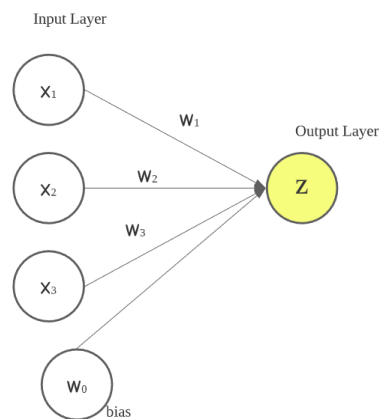


Figure 1.1

If we have 10 units in parallel, it can be described as following:

```
Dense(10, name="Layer with 10 units")
```

By making use of this Dense layer, very complicate deep learning network can be built easily. If you prefer PyTorch's syntax, you can refer to [4] which is called `torch.nn.Linear()` instead.

The layers can be further cascaded in serial to form a deeper network.

Tutorial 1.1 – [Colab Exercise for Dense\(\)](#)



You can skip the following section except you want to know how a deep learning network can be precisely described mathematically.

The parallel cascaded units will dichotomize (divide into two parts) input space differently. The serial cascaded layers will pick-up previous layer's outputs and repeat the dichotomy division process again. When the breadth of parallel cascade and depth of serial cascade are big enough to match the complexity of the dataset (input space), it can classify and represent complicated problem very effectively. Figure 1.2 show a section of a deep network.

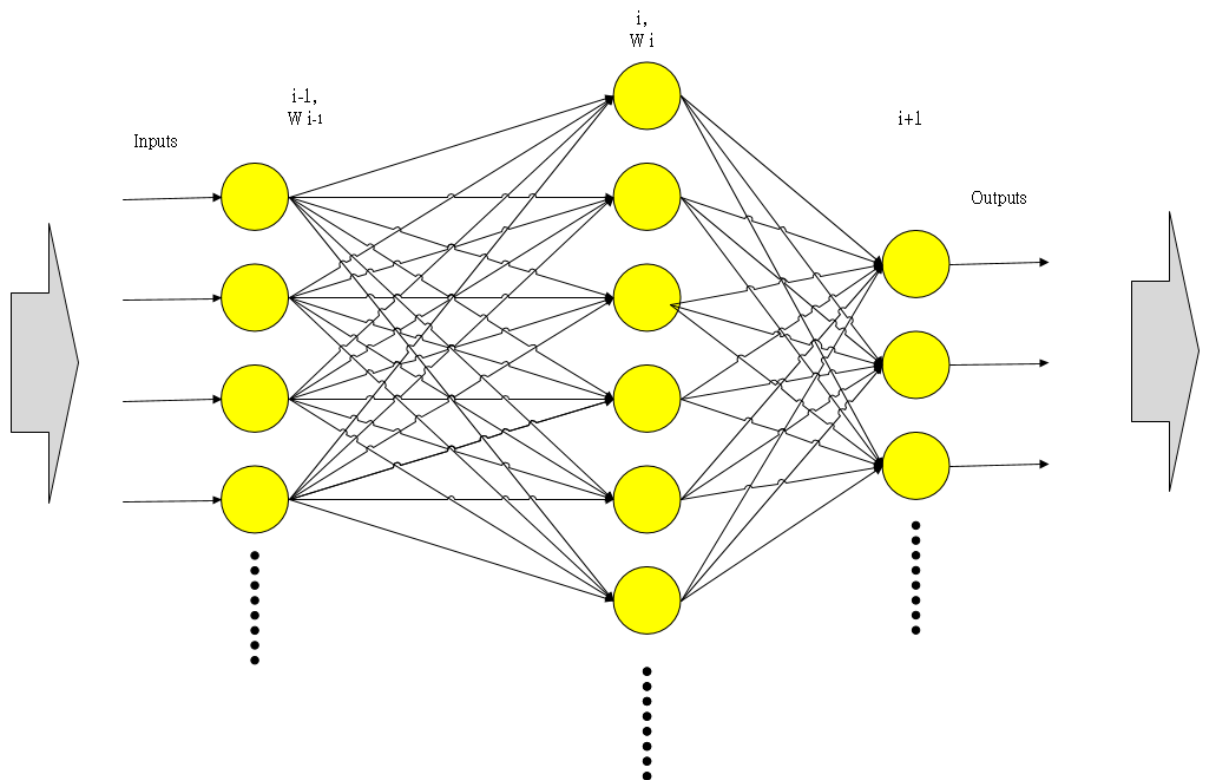


Figure 1.2

A deep network is become a hierarchical composite model where each layer (figure 3.1) applies a linear transformation followed with a nonlinear function [24] to the preceding layer.

Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be the input data, where each row of \mathbf{X} is a D-dimensional data point and N is the number of training sample in the image space and $W^i \in \mathbb{R}^{d_{i-1} \times d_i}$ be a matrix representing a linear transformation applied to the output of a layer i-1, $X_{i-1} \in \mathbb{R}^{N \times d_{i-1}}$, to obtain a d_i dimensional representation $X_{i-1}W^i \in \mathbb{R}^{N \times d_i}$ at layer i. For example, each column of W^i could represent a convolution with some filter in convolutional neural networks or the application of a linear classifier in fully connected networks.

Let $\psi_i: \mathbb{R} \rightarrow \mathbb{R}$ be a nonlinear activation function, e.g. a sigmoid $\psi_i(x) = 1/(1 + e^{-x})$ or a rectified linear unit $\psi_i(x) = \max\{0, x\}$. This non linearity is applied to each entry of $X_{i-1}W^i$ to generate the i-th layer of a neural network as $X_i = \psi_i(X_{i-1}W^i)$. The output X_I of the network is thus given by:

$$\mathcal{K}(\mathbf{X}, \mathbf{W}^1 \dots \mathbf{W}^I) = \psi_I(\psi_{I-1}(\dots \dots \psi_2(\psi_1(\mathbf{X}\mathbf{W}^1) \mathbf{W}^2) \dots \dots \mathbf{W}^{I-1})\mathbf{W}^I)$$

— Equ 1.2

When output dimensions of the network is C that equal to d_I , \mathcal{K} will be an $N \times C$ matrix and C is the number of classes for a pattern recognition problem. Notice also that the mapping can be seen as a function of all the network weights $\mathbf{W} = \{W^i\}_{i=1}^I$ with a fixed input \mathbf{X} and where the I layer is the last layer, output layer, with the indexed number I. We can view the composite mapping \mathcal{K} as a function of the input data \mathbf{X} with weights \mathbf{W} , $\mathcal{K}(\mathbf{X}, \mathbf{W})$. Furthermore, we can say that the weights \mathbf{W} in $\mathcal{K}(\mathbf{X}, \mathbf{W})$ of a network trained by the input \mathbf{X} characterized the network. This characterized network \mathcal{K} is defined and say learned some conceptual knowledges from the input dataset \mathbf{X} . It can be easily built as a class object by modern object orientated programming language together with a defined training method.

We can form and vary a deep-learning network by varying I (number of layers), d_i (width of the layer i) and ψ_i (activation function in layer i) to form different types of architectures. When we use different kinds of ψ_i for different layers, we can take them into account and see all the activation functions as $\boldsymbol{\psi} = \{\psi_i\}_{i=1}^I$. Then

$\mathcal{K}(\mathbf{X}, \mathbf{W}, \boldsymbol{\psi})$ is an extended description to cover varying activation functions.

Reference:

[1] Y.S. Abu-Mostafa, M. M. Ismail, H.T. Lin, Learning From Data, a short course.

[2] T.Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.

[3] https://en.wikipedia.org/wiki/Linear_regression#:~:text=In%20statistics%2C%20linear%20regression%20is,as%20dependent%20and%20independent%20variables

[4] https://pytorch.org/tutorials/beginner/examples_nn/two_layer_net_nn.html

Coding Exercise:

Dense() :

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Flatten, Softmax
```

```
model=Sequential()
```

```
model.add([Dense(64), input_shape=(64,64)])
```