Introduction to Deep Learning

IRS ML Section[⋆]

Hwa Chong Institution

1 Deep Learning

1.1 Definition

Deep Learning is machine learning methods based on *artificial neural networks* with *representation learning*.

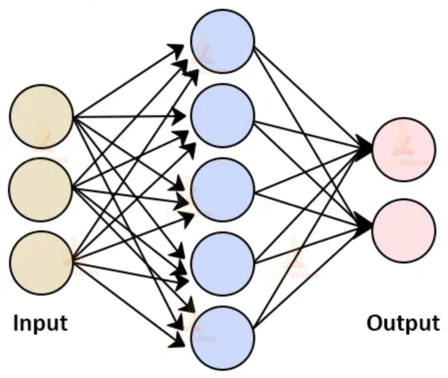


Fig. 1. A schematic diagram of an Artificial Neural Network

2 Feedforward Neural Network

Feedforward Neural Network is the most basic form of ANN where the information traverses in one direction

2.1 Neuron

Neurons in deep learning models are nodes through which data and computations flow. They are responsible for learning the "*representations*". It is about applying a matrix transformation on the input like a function f(x).

[⋆] Written by Chenghao and Wing Yip

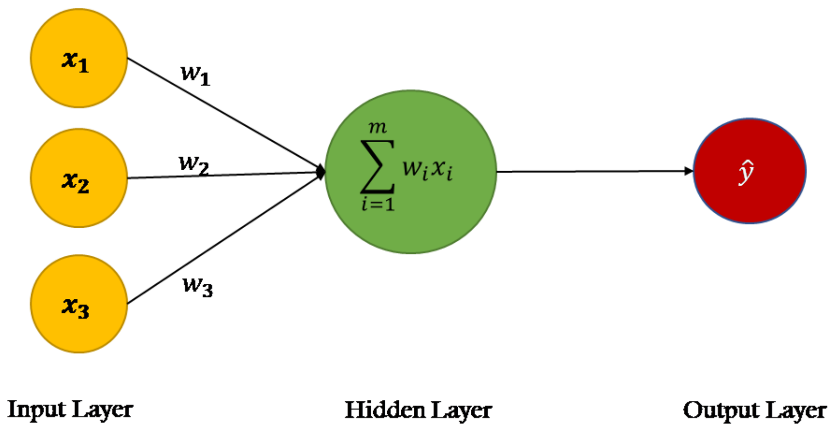


Fig. 2. A neuron is responsible for learning a representation.

3 Layers

Refer back to the figure, we can divide layers in a feedforward deep neural network into 3 parts, namely *Input Layers, Hidden Layers, and Output Layers.*

3.1 Input & Output Layers

As suggested by their names, input and output layers are used to take in input data and produce results.

As computer only knows how to process numbers, for an image input, we will describe it in terms of its RGB values at each pixel.

The number of node in the output layer is usually the same as the number of classes you want a model to classify.

3.2 Hidden Layers

Hidden layers are very important as a model's ability to learn a task is mostly dictated by how the model's hidden Layers are like.

Sigmoid

$$sigmoid(z) = \frac{1}{1+e^{-z}} \tag{1}$$

tanh

$$tanh(z) = \frac{e^{2z}-1}{e^{2z}+1} \tag{2}$$

ReLu

$$ReLU(z) = max(0, z) \tag{3}$$

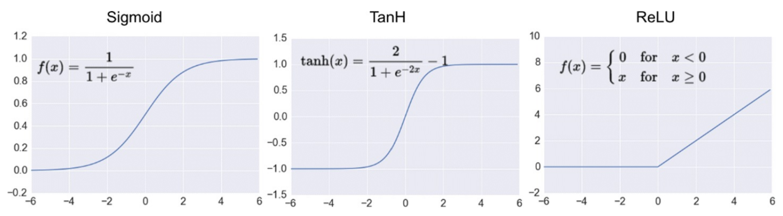


Fig. 3. Activation functions. Sigmoid (left), tanh (centre) and ReLU (right) are three commonly used activation functions.

As you can see, all these introduce non-linearity, giving the network potential to learn a non-linear relationship between the input and the output.

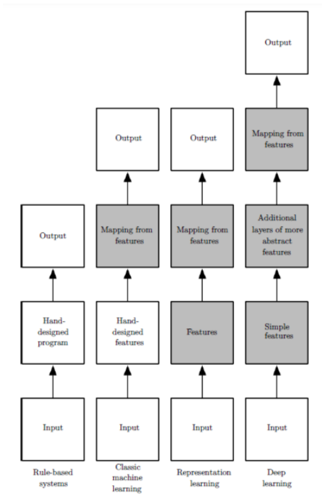


Fig. 4. Comparison between various methods and DL method

Thinking Question Why is Deep Learning performing better than traditional Machine Learning methods? (SSEF 2022)

4 Training a Neural Network

Now that we've covered the basic structure of a neural network, the next question to ask is: how do we train it? Training a neural network simply means to find values for the weights and biases that cause the network to output the correct predictions for the training data points. As it turns out, neural networks are trained using the same basic principles that you used to train your linear regression model - only on a far larger scale.

Consider an N-layer neural network $F(x)$ where x is the input vector to the network. In the case of a deep neural network, the output of each layer is the input to the next. So if we denote the function computed by the i-th layer as L_i:

$$F(x) = L_N(L_{N-1}(\ldots L_2(L_1(x))\ldots))$$

Assuming all layers are regular fully connected layers, each with a weight matrix W_i and activation function g_i, we may formulate each layer as $L_i(x) = g_i(W_i x)$, where biases are omitted for simplicity. Thus we obtain:

$$F(x) = g_N(W_N g_{N-1}(W_{N-1} \ldots g_2(W_2 g_1(W_1 x))\ldots))$$

Just like for other models, we quantify the network's prediction error using some loss function J, which we seek to minimise. We thus adjust each layer's weights using the gradient descent algorithm:

```
for $i \leftarrow 1, N$ do
    $W_i \leftarrow W_i - \nabla_{W_i} J$
end for
```

where $\nabla_{W_i} J$ is the gradient of the loss with respect to W_i.

The purpose of the backpropagation algorithm is to compute $\nabla_{W_i} J$. As its name suggests, it propagates the error signal backwards through the network to the i-th layer. For convenience, we define

$$\begin{aligned} a_0 &:= x \\ z_i &:= W_i a_{i-1} \\ a_i &:= g_i(z_i) \end{aligned}$$

for $i \geq 1$. Thus, z_i and a_i represent the unactivated and activated output vectors of the i-th layer, respectively. The activated output of the last layer a_N represents the final output of the network.

Differentiating J with respect to W_i using the chain rule, we have that

$$\begin{aligned}\frac{\partial J}{\partial z_i} &= \frac{\partial J}{\partial a_N} \odot \frac{\partial a_N}{z_N} \cdot \frac{\partial z_N}{\partial a_{N-1}} \odot \frac{\partial a_{N-1}}{z_{N-1}} \cdot \frac{\partial z_{N-1}}{\partial a_{N-2}} \odot \ldots \cdot \frac{\partial z_{i+1}}{\partial a_i} \odot \frac{\partial a_i}{\partial z_i} \\ &= \frac{\partial J}{\partial a_N} \odot g'_N \cdot W_N \odot g'_{N-1} \cdot W_{N-1} \odot \ldots \cdot W_{i+1} \odot g'_i\end{aligned}$$

where $\odot$ denotes an element-wise product, and $v \cdot M$ denotes a multiplication of a row vector[1] v by a matrix M.[2] Note that in the above equation, we differentiate scalars and vectors with respect to vectors. This may seem odd, but is provably rigorous.

We compute $\nabla_{W_i} J$ as follows:

$$\nabla_{W_i} J = (\frac{\partial J}{\partial z_i})^T a_{i-1}^T$$

$\frac{\partial J}{\partial z_i}$ can be computed recursively by

$$\begin{aligned}\frac{\partial J}{\partial z_{i-1}} &= \frac{\partial J}{\partial z_i} \cdot \frac{\partial z_i}{\partial a_{i-1}} \odot \frac{\partial a_{i-1}}{\partial z_{i-1}} \\ &= \frac{\partial J}{\partial z_i} \cdot W_i \odot g'_{i-1}\end{aligned}$$

In backpropagation, we first calculate $\frac{\partial J}{\partial z_N}$, using it to update the weights in the N-th layer. We then use it to calculate $\frac{\partial J}{\partial z_{N-1}}$, updating the weights in the $(N-1)$-th layer, and so on.

5 Loss Functions

So far, we have mainly focused on the mean squared error loss function. However, there exist other kinds of loss that serve to quantify a model's error in different ways, mainly depending on whether the model is used for classification or regression.

All loss functions receive the model's predictions $\hat{y}$ as input, so that the prediction error may be calculated.

[1] The derivative of a scalar-valued function with respect to a vector is a *covector*, or, roughly speaking, a row vector. This is such that the derivative can act on a column vector through matrix multiplication to give a scalar, which is the change in the best linear approximation of the function due to a displacement in the input of the function by that column vector.

[2] See https://en.wikipedia.org/wiki/Backpropagation for more information.

5.1 Classification Losses

Cross-Entropy When the number of possible classes is more than two, $\hat{y}_i$ is a vector $[p_1, ..., p_N]^T$, where p_i is the model's estimate of the probability that the input belongs to class i. In this case, each sample's label y_i is a vector of length N, the elements of which are all zero except for that corresponding to the actual class of sample i. For instance, a sample belonging to class 2 would have the label $[0, 1, 0, 0, ..., 0]^T$.

In calculating classification loss, we wish to calculate the degree to which the model's prediction vector $\hat{y}_i$ disagrees with the label y_i, or how "surprised" we are by $\hat{y}_i$ after seeing y_i. In information theory, *entropy* is used to quantify this.

Intuitively, the cross-entropy $H(P, Q)$ of a probability distribution Q relative to another distribution P is the number of bits required to represent an event using Q instead of P. One may consider Q as an approximation of P, and $H(P, Q)$ as encoding the extent to which Q is different from P.[3] More formally,

$$H(P, Q) = -\sum_{x \in X} P(x) \log Q(x)$$

where X is the set of events whose probabilities P and Q can both measure.

When applying this to classification loss, we take the label y_i as the "true" distribution P, and the prediction $\hat{y}_i$ as the "approximation" Q of the true distribution. Hence:

$$H(P, Q) = -\sum_{j=1}^{N} y_{i,j} \log \hat{y}_{i,j}$$

where $y_{i,j} and \hat{y}_{i,j}$ are the true and predicted probabilities of sample i belonging to class j, respectively. To reiterate, $y_{i,j}$ is 1 if j is the true class, and 0 otherwise.

5.2 Regression Losses

Typically, using mean absolute error or mean squared error suffices for regression tasks. Both these loss functions receive a *vector* $\hat{y}$ of the model's predictions as input, where each $\hat{y}_i$ is the model's prediction on the i-th training example.

Mean Absolute Error

$$MAE(\hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i|$$

$$MAE'(\hat{y}) = \frac{1}{N} \sum_{i=1}^{N} sgn(\hat{y}_i - y_i)$$

[3] This idea of "difference" between probability distributions is more accurately represented by the *Kullback-Leibler divergence*, which you may read more about at https://en.wikipedia.org/wiki/Kullback–Leibler_divergence.

Mean Squared Error

$$MSE(\hat{y}) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$$MSE'(\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)$$

Note: It is useful to cache $\hat{y} - y_i$ in a variable to avoid evaluating it multiple times when calculating MAE or MSE and their derivatives.

6 optimizer

An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improve the accuracy. Examples of optimizers include:

- Gradient Descent
- Stochastic Gradient Descent
- RMSprop
- Adam

We will explain these different optimizers in the next session. For now, go and play around with these optimizers.

7 Convolutional Neural Network

7.1 Convolutional Layer

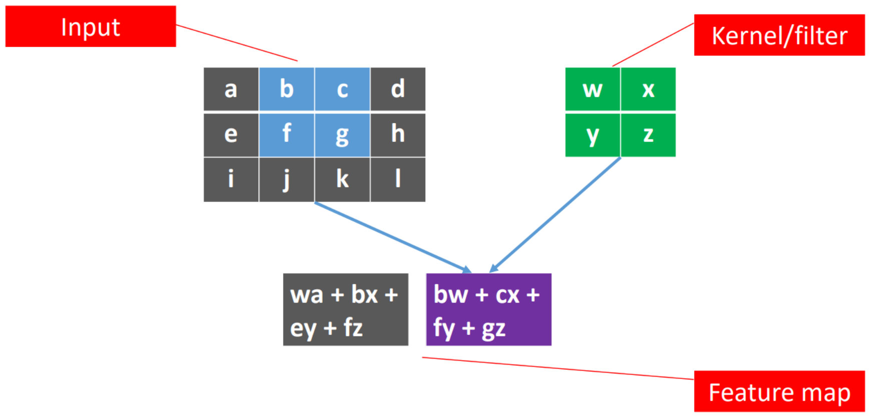
A convolution converts all the pixels in its receptive field into a single value. We will allow us to decrease the image size and bring important the information into a smaller field.

7.2 Pooling Layer

Usually we have max pooling and average pooling, its purpose is similar to that of a convolutional layer. But this is not trainabled.

7.3 Fully-connected Layer

Fully-connected Layers are similar to a feedforward Neural Network that will be responsible for learning certain features and doing the final prediction.



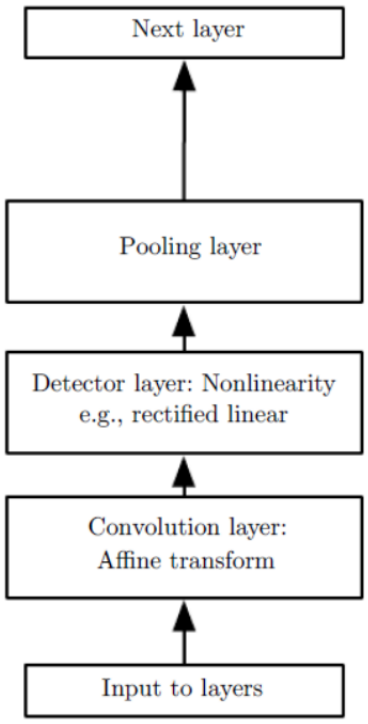

Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Fig. 5. Schematic Diagram for a Convolutional layer