Assignement -12

Dhanush V Nayak
ee23btech11015@iith.ac.in

1 DEEP LEARNING

Deep Learning is a type of machine learning where we use complex mathematical structures called circuits to guess outcomes.These circuits have multiple layers in them that's why its called "deep". Object recognition, language translation, speech recognition, and many more applications use deep learning. Deep learning originated from the working of neurons.Thats why the networks trained are called neural networks. Methods like decision trees also handle complex relationships but only for a small portion of inputs. Deep learning allows all input variables to work together in complex ways.

1.1 Simple Feedforward Networks

A feedforward network is a neural network where information flows in only one direction, from input to output, without any loops. The connections between the nodes form a *directed acyclic graph*, where each node computes a function of its inputs and passes the result forward to the next layer. In contrast, a recurrent network allows the outputs to be fed back into its inputs, creating a *dynamical system* with memory. Recurrent networks will be discussed in later sections.

An example of a feedforward network is a *Boolean circuit*, where each node computes Boolean functions of its inputs. In neural networks, however, the input values are usually continuous, and each node computes a weighted sum of its inputs, followed by a non-linear activation function.

1.1.1 Networks as Complex Functions

Each node in a network is called a unit. Following the McCulloch-Pitts model, a unit calculates the weighted sum of its inputs from predecessor nodes, applies a non-linear activation function, and produces its output. Mathematically, the output a_j of unit j is given by:

$$a_j = g_j\left(\sum_i w_{i,j}a_i\right) = g_j(\text{in}_j)$$

where g_j is the activation function, and $w_{i,j}$ is the weight between units i and j. To ensure that the input is non-zero, even if all inputs are zero, each unit also has an extra input from a dummy unit fixed at $+1$, with a corresponding weight $w_{0,j}$. Thus, we can rewrite the equation in vector form:

$$a_j = g_j(\mathbf{w}^T\mathbf{x})$$

where $\mathbf{w}$ is the vector of weights, and $\mathbf{x}$ is the input vector.

The nonlinearity of the activation function is crucial. Without it, the network would only represent a linear function, regardless of its complexity. The universal approximation theorem states that a network with two layers—one with a non-linear activation and one with a linear activation—can approximate any continuous function to any level of accuracy.

Several common activation functions are used in neural networks:

- Sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- ReLU (Rectified Linear Unit):

$$\text{ReLU}(x) = \max(0, x)$$

- Softplus function:

$$\text{softplus}(x) = \log(1 + e^{x})$$

- Tanh function:

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

The tanh function is a scaled version of the sigmoid, with a range between -1 and $+1$.

By combining multiple units in a network, we create a computation graph or dataflow graph. For example, the output $\hat{y}$ of a simple network can be expressed as:

$$\hat{y} = g_5(w_{0,5} + w_{3,5}g_3(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g_4(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2))$$

This shows how the output depends on the inputs x_1, x_2 and the weights w.

We can generalize this using matrix notation. Let $W^{(1)}$ and $W^{(2)}$ denote the weight matrices for the first and second layers, respectively. Then, the network function $h_w(x)$ is given by:

$$h_w(x) = g^{(2)}(W^{(2)}g^{(1)}(W^{(1)}x))$$

Here, $g^{(1)}$ and $g^{(2)}$ are the activation functions in the first and second layers, respectively. This represents a general feedforward network as a sequence of matrix multiplications followed by activation functions. The computation graph is small and shallow. We construct graphs and adjust their weights to fit the data. In fully connected networks, every node in one layer is connected to every node in the next layer.

1.1.2 Gradients and learning

In supervised learning, we minimize the loss function using gradient descent. The loss function measures the difference between the predicted output, $\hat{y} = h_w(x)$, and the true output y. For a neural network, the weights are adjusted by computing the gradient of the loss function and updating the weights to reduce this loss.

For a simple case where a weight $w_{3,5}$ is connected to the output unit, the gradient of the loss function is computed using the chain rule:

$$\frac{\partial}{\partial w_{3,5}}\text{Loss}(h_w) = -2(y - \hat{y})g_5'(in_5)a_3$$

This is straightforward because $w_{3,5}$ directly affects the output. For weights like $w_{1,3}$, which are not directly connected to the output unit, the chain rule is applied again:

$$\frac{\partial}{\partial w_{1,3}}\text{Loss}(h_w) = -2(y - \hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)x_1$$

This gives the general method of back-propagation, where the error is propagated backward through the network, adjusting weights based on the perceived error at each unit. With modern deep learning , the process of calculating gradients is automated via automatic differentiation, making experimentation with different network structures and applies calculus ina systematic way to calculate gradients.

2 Computation Graphs for Deep Learning

2.1 Input encoding

In computational graphs, input and output nodes represent connections from data x and output y. For Boolean attributes, n input nodes are created with Bernoulli mapping . Numeric attributes (integer or real) are typically used directly but may be scaled to fit a range or a log scale if needed.For images, each pixel in an RGB image can be treated as a group of attributes, but handling pixel adjacency is important. Specialized array-like internal structures are typically used in networks handling images to preserve this relationship. Attributes with more than two values are encoded using one-hot encoding.This is better than using integers because, in neural networks, adjacent numbers imply relationships, which can be misleading.

2.2 Output layers and loss functions

In neural networks, encoding output data y is similar to encoding input data. For a task like predicting the Weather variable with values {sun, rain, cloud, snow}, we use one-hot encoding, where each output node represents a possible outcome.

The network tries to predict $\hat{y}$, which ideally matches the true y. A common loss is the squared-error, but for most deep learning tasks, we use negative log likelihood or cross-entropy, especially when interpreting the output as a probability. The goal is to minimize the following:

$$w^* = \arg\min_w -\sum_{j=1}^{N} \log P_w(y_j|x_j)$$

Cross-entropy measures the difference between two distributions P and Q (the predicted output):

$$H(P,Q) = \int P(z) \log Q(z)\, dz$$

For binary classification, a sigmoid function is used, outputting a value between 0 and 1, interpreted as the probability of the positive class. In multiclass classification, with d possible categories, we use a softmax layer, which ensures the output values sum to 1:

$$\text{softmax}(in)_k = \frac{e^{in_k}}{\sum_{k'=1}^{d} e^{in_{k'}}}$$

Softmax highlights the largest values, assigning higher probabilities to them.

In regression, where y is continuous, we use a linear output layer (no activation function). This is equivalent to doing classical linear regression, minimizing squared error. For more complex cases, like multimodal data, we can use a mixture density layer, which predicts a mixture of Gaussian distributions. The network learns the mixture by adjusting the weights during training.

2.3 Hidden layers

Neural networks are trained by exposing them to many pairs of input x and output y. During this process, the model performs several analysis across multiple layers, interpreting x in new ways.One reason deep learning works well is because the network can break down complex transformations into simpler ones. Just like the models that do face recognition started their journey back by identifying basic shapes. For many years, the most common activation functions in hidden layers were sigmoid and tanh, but since around 2010, ReLU and softplus have become more popular because they help avoid the "vanishing gradient" problem, which can prevent networks from learning effectively. There are many ways to design neural networks, but still we have a dilemma of why certain structures work better for a specific problems.