Neural Networks

Satyanarayana Gajjarapu
AI24BTECH11009
Department of Artificial Intelligence
ai24btech11009@iith.ac.in

Deep learning is a broad family of techniques for machine learning in which hypotheses take the form of complex algebraic circuits with tunable connection strengths. It is most widely used approach for speech recognition, image synthesis, etc. and also plays a significant role in reinforcement learning applications. It has origins in the work that tried to model networks of neurons in the brain using computational circuits. The networks trained by deep learning methods are often called **neural networks**. Linear and logistic regression can handle a large number of input variables but the computation path from each input to the output is very short. Decision lists and decision trees allow for long computation paths but only for a small fraction of the possible input vectors. The aim of deep learning is to train circuits whose computation paths are long and are able to handle a large number of input variables. In neural networks, usually the input values are continuous, the nodes take continuous inputs and generate continuous outputs.

1 Feedforward Network

A **feedforward network** has connections only in one direction, it forms a **directed acyclic graph** involving designated input and output nodes without any loops. Boolean circuits are an example of feedforward networks. A **recurrent network** feeds its intermediate or final outputs back into its own inputs. Each node within a network is called a **unit**. A unit calculates the weighted sum of the inputs from predecessor nodes and produces an output using nonlinear function. Let a_j denote the output of unit j and $w_{i,j}$ be the weight attached to the link from unit i to j.

$$a_j = g_j(\sum_i w_{i,j}a_i) \equiv g_j(in_j)$$

where g_j is a **nonlinear activation function** associated with unit j and in_j is the weighted sum of the inputs to unit j. To make the total weighted input in_j to unit j nonzero even when the outputs of the previous layer are zero. The equation is modified using **w**, the vector of weights leading into unit j and **x**, the vector of inputs to unit j.

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

The nonlinearity of the activation function is important because without it, any composition of units would still represent a linear function. Examples of activation functions are **sigmoid**, **rectified linear unit** (ReLU), **softplus** and **tanh** functions.

$$\sigma(x) = 1/(1+e^{-x}), \text{ReLU}(x) = max(0,x), \text{softplus}(x) = \log 1 + e^x, \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

All these are monotonically nondecreasing whose derivatives are nonnegative. The output $\hat{y}$ can be expressed as a function $h_{\mathbf{w}}(\mathbf{x})$ of the inputs and the weights. The network can be assumed as a **computation graph** or **dataflow graph**, essentially a circuit in which each node represents an elementary computation. Let $\mathbf{W}$ be the weight matrix for the network, $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ be the weights

Preprint. Under review.

in first and second layer respectively. $\mathbf{g}^{(1)}$ and $\mathbf{g}^{(2)}$ be the activation functions in the first and second layers. Then the entire network can be expressed as

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{g}^{(2)}(\mathbf{W}^{(2)}\mathbf{g}^{(1)}(\mathbf{W}^{(1)}\mathbf{x}))$$

The approach of gradient descent can be used to learning the weights in computation graphs. The squared loss function L_2 is used to calculate the gradient descent for a single training example.

$$\begin{aligned} Loss(h_{\mathbf{w}}) &= L_2(y, h_{\mathbf{w}}(\mathbf{x})) = \|y - h_{\mathbf{w}}(\mathbf{x})\|^2 = (y - \hat{y})^2 \\ \frac{\partial}{\partial w_{3,5}} Loss(h_{\mathbf{w}}) &= -2(y - \hat{y})g_5'(in_5)a_3 \\ \frac{\partial}{\partial w_{1,3}} Loss(h_{\mathbf{w}}) &= -2(y - \hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)x_1 \end{aligned}$$

The above two equations are two different cases to show the gradient of loss function. The phenomenon of passing the error at the output back into the network is called **backpropagation**. Many deep learning algorithms involve **automatic differentiation** which uses reverse mode differentiation to efficiently compute gradients for backpropagation.

2 Computational Graphs

The input and output nodes of a computational graph are the ones that connect directly to the input data **x** and the output data **y**. Most deep learning applications interpret the output values $\hat{y}$ as probabilities and use **negative log likelihood** as the loss function. Maximizing the log likelihood of the data is equivalent to minimizing a loss function defined as the negative log likelihood. In deep learning, it is common to refer about minimizing the **cross entropy** loss $H(P, Q)$.

$$H(P, Q) = \int P(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z}$$

During the training process, a neural network is shown many input values **x** and many corresponding output values **y**. While processing an input vector **x**, the neural network performs several intermediate computations before producing the output **y**. Each layer transforms the representation produced by the preceding layer to produce a new representation.