Introduction to Neural Networks

Tirtharaj Dash

August 11, 2019

This note is primarily prepared as an introduction to the Neural Network course (BITS F312) at BITS Pilani, Goa Campus. The resources that are followed during the preparation of this note are provided as footnotes.

1 Some Terminologies

Deduction Deduction or deductive reasoning works from the "general" to the "specific". This is also called a "top-down" approach. The deductive reasoning works as follows: think of a theory about a topic and then narrow it down to a specific hypothesis (a hypothesis that we test or can test). Narrow down further if we would like to collect observations for hypothesis (note that we collect observations to accept or reject the hypothesis and the reason we do that is to confirm or refute our original theory). In a conclusion, when we use deduction we reason from general principles to specific cases, as in applying a mathematical theorem to a particular problem or in citing a law of physics to predict the outcome of an experiment.

Theory $\rightarrow$ Hypothesis $\rightarrow$ Observation $\rightarrow$ Confirmation

Here is an example:

Theory: All men are mortal.

Hypothesis: Tirtharaj may be mortal.

Observation: Tirtharaj dies.

Confirmation Tirtharaj is mortal.

(In the above example, Tirtharaj is not your instructor!)

Induction Induction or inductive reasoning works the other way around, it works from observation (or observations) works toward generalizations and theories. This is also called a "bottom-up" approach. Inductive reason starts from specific observations, look for patterns (or no patterns), regularities (or irregularities), formulate a hypothesis that we could work with and finally ended up developing general theories or drawing a conclusion. Note that that is how Newton reached to "Law of Gravitation" from "apple and his head? observation"). In conclusion, when we use Induction we observe several specific instances and from them infer a general principle or law.

$$\mathsf{Observation(s)} \rightarrow \mathsf{Pattern} \rightarrow \mathsf{Hypothesis} \rightarrow \mathsf{Theory}$$

Here is an example:

Observation: A few fell from the tree on the ground.

Pattern: All the apple fell on the ground.

Hypothesis: All objects fall on the ground.

Theory All objects fall on the ground due to some force.

ML, in essence, is inductive reasoning.

Well-posed learning problem Data should be available or should be producible. Three things are needed to define a well-posed learning problem:

- a task, T
- exprience, E with which it will achieve T
- performance measure, P

2 Loose similarity between Learning in Biological Brain and Neural Machine Learning

The (complex) neural circuits in biological organisms learn and function based on external stimuli. Similarly, machine learning (ML) models are

learned from the external training data containing examples of input–output pairs. This learning assumes that there exists a relationship between the inputs (independent variables) and output (dependent variable). This relationship could be considered to be a "function". An instance of training data might contain twitter messages (input) and its label as positive or negative (output).

The task (T) in the above example is to label twitter message as positive or negative. The goal is to construct a machine by continuously learning from many such message–label pairs (experience, E: also constitutes of model parameters) given at various time intervals. The quality of learning is decided based on a performance measure (P).

Tom Mitchell defines ML as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

In this course, we focus on this computer program as a neural network (NN), which is essentially a function (mostly, nonlinear). These training data pairs are fed into the neural network by using the input (feature) representations to make predictions about the output labels. The training data provides feedback to the correctness of the parameters (synaptic weights) in the NN (see an example in Figure 1). The goal of changing the weights is to modify the computed function to make the predictions more correct in future iterations. Therefore, the weights are changed carefully in a mathematically justified way to reduce the error in computation on that instance of input–output pair.

By successively adjusting the weights between neurons over many input-output pairs, the function ($\hat{y} = f(\mathbf{x})$) computed by the neural network is refined over time so that it provides more accurate predictions. Therefore, if the neural network is trained with many different positive and negative tweets, it will eventually be able to properly recognize future tweets it has not seen before. This ability to accurately compute functions of unseen inputs by training over a finite set of input-output pairs is referred to as **model generalization**. The primary usefulness of all machine learning models is gained from their ability to generalize their learning from seen training data to unseen examples.

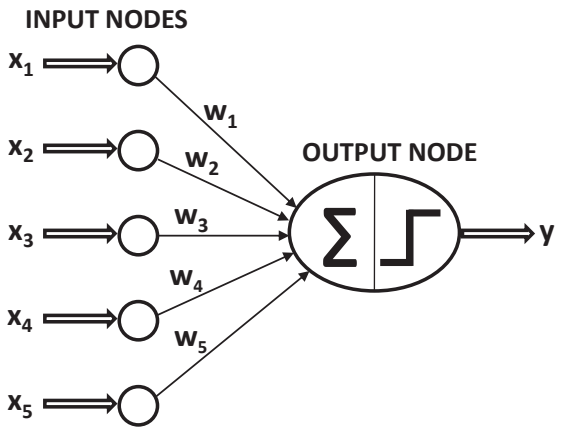

Figure 1: A simple single layered neural network taking input defined by 5 attributes $\mathbf{x} = (x_1, x_2, \ldots, x_5)$, and producing 1 ouput y. Here, we will refer the computed output of a neural network as $\hat{y}$, and true/desired output as y.

The most basic units of computation in the neural network are inspired by traditional ML algorithms like *least-squares regression* and *logistic regression*. NNs gain their power by putting (stacking: chaining) together many such basic units and learning the weights of these units jointly in order to minimize the prediction error (also called **loss**). One can view an NN as a **computational graph** of basic units. Large and complex NN architectures have more learning and generalizing power over the simple ones — these complex networks are now regarded as **Deep Neural Networks** and the idea of the concept learned as a complex nonlinear function of inputs can be referred to as **Deep Learning**. If an NN is built without hooking together multiple units, the learning algorithm often reduces to the classical ML model: an instance of such an NN is **Perceptron**. Figure 2[1] shows that Deep learners become more attractive than conventional methods primarily when sufficient data/computational power is available.

Further readings: Understanding a dataset, what is a well-posed problem, Bias-variance tradeoff, the complexity of ML models, optimiza-

[1]Book: Aggarwal, C. C. (2018). Neural networks and deep learning; Springer

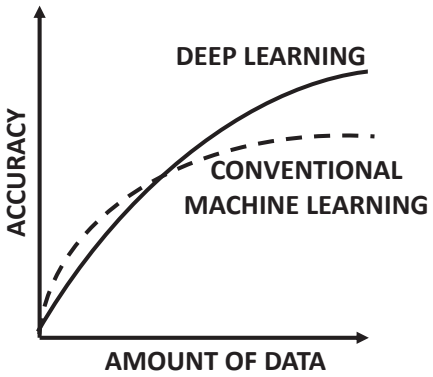

Figure 2: An illustrative figure capturing the notion of power of standard non-neural ML methods and Deep Networks

tion and gradient descent procedures, the curse of dimensionality in ML, No Free Lunch (NFL) theorem in ML

3 Perceptron

It is a single-layer network: a set of inputs is directly mapped to an output by using a generalized variation of a linear function. This neural network contains a single input layer and an output node. The basic architecture of the perceptron is shown in Figure 1 in which the inputs are marked x_1 through to x_5 for 5 input features in data, and the perceptron computed output is $\hat{y}$. To make it generic, we will write inputs as d-dimensional vector $\mathbf{x} = (x_1, \ldots, x_d)$. Furthermore, we will assume that there is an additional **bias neuron** which takes on the input always +1, and the corresponding synaptic weight for this neuron is w_0. We will include this bias neuron in out input vector $\mathbf{x} = (x_0 = +1, x_1, \ldots, x_d)$ and the corresponding synaptic weights are $\mathbf{w} = (w_0, w_1, \ldots, w_d)$.

The network computes a linear function $\mathbf{w}^\top \mathbf{x} = \sum_{i=0}^{d} w_i x_i$. The prediction of the network is computed as:

$$\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x})$$

The sign function maps a real value to either +1 or –1, which is appropriate for binary classification. Note the hat on top of the variable y to indicate

that it is a predicted value rather than an observed value[2]. The error of the prediction is therefore $\mathcal{E}(\mathbf{x}) = y - \hat{y}$, which is one of the values drawn from the set $\{-2, 0, +2\}$. In cases where the error value $\mathcal{E}(\mathbf{x})$ is nonzero, the weights in the neural network need to be updated in the (negative) direction of the error gradient[3].

The sign function described above serves the role of an **activation function**. Different choices of activation functions can be used to simulate different types of models used in machine learning, like *least-squares regression with numeric targets*, the *support vector machine*, or a *logistic regression classifier*.

Based on our discussion so far, we can define a perceptron as a hyperplane:

$$\mathbf{w}^\top \mathbf{x} = 0 \tag{1}$$

We can visualise Equation (1) as:

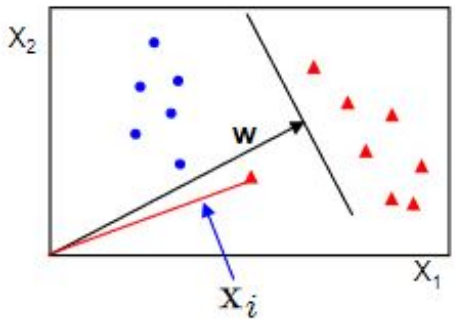

Figure 3: Illustration of perceptron equation in feature space of two dimension x_1 and x_2; the weight vector $\mathbf{w}$ and an instance $\mathbf{x}_i$

During the learning stage, described previously, the perceptron will try to accommodate the misclassified example by updating the weights:

3.1 Learning in Perceptron

Initially, learning in perceptron was heuristically designed to minimize the number of misclassifications. We can write the perceptron algorithm in the

[2]Observed value is also called true value, denoted as y

[3]Gradient descent procedure

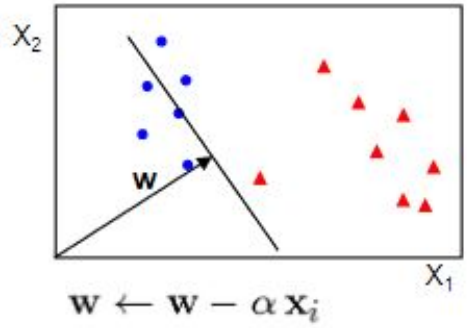

Figure 4: Weight update in perceptron: α is a constant (> 1) and is called learning step factor or learning rate

least-squares form with respect to all training instances in a data set $\mathcal{D}$ containing feature-label pairs:

$$\min_{\mathbf{w}} \mathcal{L} = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} (y_i - \hat{y}_i)^2 \tag{2}$$

This type of minimization objective function is also referred to as a **loss function**[4]. But, for this objective function, the sign function is not differentiable, with step-like jumps at specific points. Furthermore, the sign function takes on constant values over large portions of the domain, and therefore the exact gradient takes on zero values at differentiable points. This results in a staircase-like loss surface, which is not suitable for gradient-descent. The perceptron algorithm (implicitly) uses a *smooth approximation* of the gradient of this objective function with respect to each example:

$$\begin{aligned} \nabla_{\mathbf{w}}^{\text{smooth}} \mathcal{L} &= \frac{\partial}{\partial \mathbf{w}} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} (y_i - \text{sign}(\mathbf{w}^\top \mathbf{x}_i))^2 \\ &= \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} (y_i - \hat{y}_i)(-\mathbf{x}_i) \end{aligned}$$

By doing this, the staircase is smoothed out into a sloping surface defined by the smoothing function.

Although the above objective function is defined over the entire training data, the training algorithm of neural networks works by feeding each input data instance $\mathbf{x}$ into the network one by one (or in small batches) to create

[4]For mathematical convenience, we will multiply this by $\frac{1}{2}$.

the prediction $\hat{y}$. The weights are then updated, based on the error value $\mathcal{E}(\mathbf{x}) = (y - \hat{y})$. Specifically, when the data point $\mathbf{x}$ is fed into the network, the weight vector $\mathbf{w}$ is updated as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - \hat{y})\mathbf{x} \tag{3}$$

The parameter α regulates the learning rate of the neural network. The perceptron algorithm repeatedly cycles through all the training examples in random order and iteratively adjusts the weights until convergence is reached. A single training data point may be cycled through many times. Each such cycle is referred to as a *epoch*.

In *mini-batch stochastic gradient descent*, the aforementioned updates of Equation (3) are implemented over a randomly chosen subset of training points $\mathcal{S}$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i,\ \mathbf{x}_i \in \mathcal{S}} (y_i - \hat{y}_i)\mathbf{x}_i \tag{4}$$

Homework

Consider a data set in which the two points $\{(-1, -1), (1, 1)\}$ belong to one class, and the other two points $\{(1, -1), (-1, 1)\}$ belong to the other class. Start with perceptron parameter values at (0, 0), and work out a few stochastic gradient-descent updates with $\alpha = 1$. While performing the stochastic gradient-descent updates, cycle through the training points in any order.

(a) Does the algorithm converge in the sense that the change in objective function becomes extremely small over time?

(b) Explain why the situation in (a) occurs.