Intro to Machine Learning Final Paper

Sang Jun Yum
New York University

This paper presents the fundamental ideas of Neural Network and its learning rule. The ideas, information, and knowledge are retrieved from four different papers written by David H. Ackley, Geoffrey E. Hinton, Terrence J. Sejnowski, David E. Rumelhart, Ronald J. Williams, Paul J. Werbos, and F. Rosenblatt.

Introduction

In ***Learning representation by back-propagating errors***, basic formations of Neural Network and its learning rule by back-propagation are explained. The idea of back-propagation is more deeply and numerically explained in ***Backpropagation through time: What it does and how to do it***. Basically, neural network's 'Feed-Forward' corresponds to repetition of the following tasks: computing the dot product between the weight vectors and input data vectors, and then to pass the results into the activation function, then, letting the outputs of this activation function behaves as new inputs. This process is repeated until the feed-forward computation traverses through the entire network, and outputs the result. Then, the error is computed with the output of the machine and the reference output, and the gradient of the error is achieved via back-propagation using chain rule.

In ***The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain***, and ***A Learning Algorithm for Boltzmann Machines***, the concept of Neural Network is explained in relation to other fields of science. Ackley, Hinton and Sejinowski, on the other hands, explain the neural network in thermophysical contents; they present the idea of Boltzmann Machine, a neural network of which neuron's states depend on the energy gap and temperature. F. Rosenblatt explains the idea of Perceptron, an early stage neural network of which design is strongly influenced by the biological and anatomical mechanisms of perceptual recognition. For better modeling and visualization, Rosenblatt divides the perception system into four (or three) compartments; Retina, Projection Area, Association Area, and Response.

Neural Network, How it works

Feed-Forward

In conventional Machine Learning, dot product between input and weight vector is computed, then the result of this computation is fed into the hypothesis function to compute the output. However, in Deep Learning, layer(s) of hidden neurons is introduced between the input layer and the output layer. The same computation mentioned above is done to the hidden layers; dot products between input and weight vectors are computed, then the results from this computation are fed into the non-linear hypothesis (activation) function, but this time, the results of this non-linear hypothesis are not the output of the machine, but activation values of a new hidden layer. This activation values act as new inputs. The dot products between this new inputs and weights are computed and then passed again into the non-linear hypothesis function. Such task is repeated until the output of the whole network is computed. As this repeated computation traverses through a (hidden) layer by another, the level of abstraction increases, and as briefly mentioned in the introduction, this traversing is called 'Feed-Forward'. In ***Learning representation by back-propagating errors***, feed-forward is explained using simple mathematical techniques. Let j^{th} layer be a layer that is located 'forward' (or 'upward') to the i^{th} layer. Let y_i be the activation value (or vector) of the i^{th} layer, and let w_{ji} be the 'connection' or 'weight' between neurons of i^{th} layer and j^{th} layer. Then, the dot product between the activation vector and the connection between i^{th} layer and j^{th} layer is,

$$x_j = \sum_i y_i w_{ji} \tag{1}$$

- **Voltage:** For simplicity, value or vector that is passed into a hypothesis or activation function is referred as 'voltage' (Note that not only the result of equation (1) but also equation (11) can also be denoted as voltage)

Then, the activation value of j^{th} layer is computed as following.

$$y_j = \frac{1}{1 + e^{-x_j}} \tag{2}$$

The computations shown in (1) and (2) are repeated until the output layer. For simplicity, it is assumed that j^{th} layer is the output layer. The ultimate goal of the network is to find sets of weights that make the output vector produced by the network resembles the reference output vector the most. Thus, assuming that the j^{th} layer is the output layer, the error can be constructed as following,

$$E = \frac{1}{2} \sum_c \sum_j (y_{j,c} - d_{j,c})^2 \tag{3}$$

where c is an index over cases, j is an index over the output layer, y is the actual (reference) output of the network, and d is the given label.

Back Propagation

In short, feed-forward can simply be seen as passing in the voltage into the forward layers repeatedly. Now, given the output of the machine and error function, the primary concern is to adjust the weight vectors using gradient descent. However, computing the gradient of the error with respect to weights is a little more complicated in neural networks than in machine learning algorithms that do not involve any hidden layers because neural networks normally involve multiple levels of weight vectors. The goal is to compute the gradient of error with respect to each level of weights. Given j^{th}layer is the output layer and i^{th}layer is the last hidden layer, if $\frac{\partial E}{\partial w_{ji}}$ can be computed gradient of error with respect to every level of weights can be computed. The computation of gradient starts with computing $\frac{\partial E}{\partial y_j}$.

$$\frac{\partial E}{\partial y_j} = y_j - d_j \tag{4}$$

Then compute $\frac{\partial E}{\partial x_j}$.

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j}\frac{dy_j}{dx_j} = \frac{\partial E}{\partial y_j} y_j(1 - y_j) \tag{5}$$

The final step is to compute $\frac{\partial E}{\partial w_{ji}}$ using (4) and (5).

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} y_j \tag{6}$$

Simply put, computing the gradient of error with respect to the weight vector can be expressed as a chain rule.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j}\frac{\partial x_j}{\partial w_{ji}} \tag{7}$$

Process (4), (5) and (6) (or simply process (7)) are repeated until $w_{2,1}$, the connection between the first layer (the input layer) and the second layer (the first hidden layer) is computed. Then, the gradient descent is performed iteratively.

$$w \longleftarrow w - \epsilon \frac{\partial E}{\partial w} \tag{8}$$

Rumelhart, Hinton, and Williams denote above as following.

$$\triangle w = -\epsilon \frac{\partial E}{\partial w} \tag{9}$$

Improved version of update rule of equation (9) is briefly explained; acceleration method is a learning technique that considers the state of the previous ‘epoch’. Instead of using equation (9) the acceleration method uses the following equation.

$$\triangle w(t) = -\epsilon \frac{\partial E}{\partial w(t)} + \alpha \triangle w(t-1) \tag{10}$$

In equation (10), epoch t is incremented by 1 when all (input) data sweep through all layers of the network. α is an exponential decaying factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients to the weight change.

Batch Learning and Pattern Learning

In previous subsections, basic ideas of Feed-Forward and Back Propagation are explained. Nonetheless, the timing of weight update is still questionable. In ***Learning representation by back-propagating errors***, it is mentioned that $\frac{\partial E}{\partial w}$ is accumulated over all the input-output before changing the weight. Thus, weights are adjusted only after all inputs complete feed-forward. In ***Backpropagation through time: What it does and how to do it***, Paul J. Werbos defines such method as 'batch learning'. In this paper another learning method, pattern learning where weights are continually updated after observation of each input is introduced. Pseudo-code implementation of pattern learning is following.

```
procedure PATTERN LEARNING
    for $pass \leftarrow 0$ to $1000$ do
        for $t \leftarrow 0$ to $T$ do                    ▷ $T$ is the total number of dataset
            feed forward to compute output $y$
            compute error $E$
            get gradient of $E$ with respect to $w$ using chain rule
            perform update: $\triangle w = -\epsilon \frac{\partial E}{\partial w}$
        end for
    end for
end procedure
```

Note that in pattern learning, weight is adjusted in response only to the current pattern or t^{th} input vector. The technique that takes the state of the weight at previous moment (or previous input) into the consideration is explained in the following subsection.

Back Propagation Through Time and Recurrent Network

The technique of taking previous time steps into account is more profoundly explained when it comes to the Back Propagation Through Time (BPTT) Let t be both 'time step' and index of the input matrix X. Note that in such case, pattern learning is likely to be used. $W_{ji,t}$ is the t^{th} row index of the weight matrix between i^{th} and j^{th} layer. Of course, $W_{ji,t}$ can also be denoted as $w_{ji,t}$, a weight vector between i^{th} and j^{th} layer at time step t assuming the weight vector is shared. In such a system where previous time steps are important, feed-forward algorithm considers previous time steps. Instead of being fed with equation (1), the activation function is fed with the following value.

$$\sum W_{ji,t}X(t) + \sum W_{ji,t-1}X(t-1) + \sum W_{ji,t-2}X(t-2) \tag{11}$$

- **Epoch and Time Step:** Note that t in equation (11) is not t of equation (10) from the previous subsection; t in (11) is 'time step' and t in (10) is 'epoch'.

The back propagation, on the contrary, considers the the state (activation value) of the future time steps. Let $net_j(t)$ denote the output of equation (11) on j^{th} layer. Let E denote the error, and let $sig(\alpha)$ denote sigmoid function of α.
Then,

$$\frac{\partial E}{\partial sig(net_j(t))} = \frac{\partial E}{\partial sig(net_j(t))} + \sum W_{ji,t+1}\frac{\partial E}{\partial net_j(t+1)} + \sum W_{ji,t+2}\frac{\partial E}{\partial net_j(t+2)} \tag{12}$$

After computing equation (13), $\frac{\partial E}{\partial W_{ij,t+1}}$ and $\frac{\partial E}{\partial W_{ij,t+2}}$ are computed as below.

$$\frac{\partial E}{\partial W_{ij,t+1}} = \sum_{t=1}^{T} \frac{\partial E}{\partial net_i(t+1)} sig(net_j(t)) \tag{13}$$

$$\frac{\partial E}{\partial W_{ij,t+2}} = \sum_{t=1}^{T} \frac{\partial E}{\partial net_i(t+2)} sig(net_j(t)) \tag{14}$$

Notice in equation (14) and (15), $\frac{\partial E}{\partial W_{ij,t+1}}$ and $\frac{\partial E}{\partial W_{ij,t+2}}$ are denoted, not $\frac{\partial E}{\partial W_{ji,t+1}}$ and $\frac{\partial E}{\partial W_{ji,t+2}}$. The back propagation through time operation is impossible until $\frac{\partial E}{\partial W_{ij,t+1}}$ and $\frac{\partial E}{\partial W_{ij,t+2}}$ are known. The timeline is reversed for the back propagation step; back propagation can only be used by proceeding backwards in time, where the computation starts from $\frac{\partial E}{\partial W_{ji,T}}$ to $\frac{\partial E}{\partial W_{ji,1}}$. How the change in the 'future time step' influences the activation value of the 'current time step' becomes the major question in back propagation step.

Neural Network, Expansion

In previous section, the fundamental mechanisms of Neural Network are introduced. Feed-Forward and Back Propagation enable neurons to 'interact' and 'communicate' each other in a way that they adjust connections among them for better model. Furthermore, feed forward/back propagation method that are used when neurons receive 'time series' data is introduced. In this section, different neural networks that are inspired by physics (and statistical mechanic) and biology are explained: The Boltzmann Machine and Perceptron.

The Boltzmann Machine

Boltzmann Machine is a Neural Network that is named after Boltzmann distribution, a probability distribution of particles in a physical system over various (Energy) states. By definition, a Boltzmann Machine is a network of symmetrically connected neurons that make decision of being 'on' or 'off' based on the energy configuration. Just like a general neural network, Boltzmann Machine is comprised of 'units' (neurons) that are categorized into hidden and visible and connections (weights) among the units. Note that in conventional neural network (Feed-Forward Network), the connections exist only between neurons of different layers. Boltzmann Machine is free from such structural restraint; the notion of 'layer' is vague in Boltzmann Machine. It is possible that every neuron is connected to each other in Boltzmann Machine.

As mentioned earlier, the state of each neuron is determined by energy configuration. More precisely, it is determined by the energy gap between the on and off states of the neuron. Let S_i denotes the ‘state’ or ‘activation value’ of the i^{th} neuron. Then,

$$S_i = \begin{cases} 1 & \text{if } i^{th} \text{ unit is on} \\ 0 & \text{if } i^{th} \text{ unit is off} \end{cases} \tag{15}$$

Then, the global energy can be written as following.

$$E = -\sum_{i<j} w_{ij} S_i S_j + \sum_i \theta_i S_i \tag{16}$$

where w_{ij} is the weight between i^{th} and j^{th}neuron, and θ_i is a threshold. Then, the energy gap between on and off state of k^{th}neuron can be written as

$$\triangle E_k = \sum_i w_{ki} S_i - \theta_k \tag{17}$$

Equation (15) is determined by some probability distribution function P_k.

$$P_k = sigmoid(\triangle E_k / T) \tag{18}$$

where T is the temperature. Note that equation (18) resembles the Boltzmann distribution; equation (18) is a probability distribution for a particle with two (0 and 1) energy states. As in Boltzmann distribution, the probability distribution over physical system with particles that are exposed to heat at a given temperature converges as the temperature eventually reaches thermal equilibrium (the point where $T = 0$) where by definition the global energy is minimum. Note that equation (18) converges (to zero or one) as T approaches 0. As the learning of Boltzmann Machine happens, the weights in equation (18) (and (17)) will be adjusted so that equation (18) resembles the environment’s probability distribution.

The knowledge of equations above does not directly connect to the entire learning algorithm of the Boltzmann Machine. In the previous section, finding the error was relatively simple and straightforward; the error was, as noted in equation (3), simply an aggregation of square of vector distance between the given output and output of the network. In order to construct the error in Boltzmann Machine, it is imperative to understand the ultimate goal of the Boltzmann Machine; the goal is to modify the weights among neurons to construct an internal model that produces the probability distribution that is the most similar to the environment the machine is exposed. To achieve such goal, the neurons are categorized into 'hidden' and 'visible'. The 'visible' neurons function as interface between the network and the environment that the network is exposed to; simply put, the visible neurons gather information about the environment to which they are exposed. Visible neurons are 'clamped' into specific states by the given environment.

- **Clamping:** When a neuron has its value forcibly set and fixed to some externally provided value, it is said to be clamped. Usually, clamped neurons function as an input layer for the network.

Hidden neurons, on the other hands are secluded from the environment. They are not clamped by the environment, and can merely show the relation of the neuron and its neighbors. It is assumed that the environmental input vectors persist long enough to meet its thermal equilibrium (the point where T is 0) so that the entire probability distribution is shown to the visible neurons. Then, the discrepancy between the network's internal model and the environment is

$$G = \sum P(V_\alpha) ln \frac{P(V_\alpha)}{P'(V_\alpha)} \tag{19}$$

where $P(V_\alpha)$ is the probability of α^{th} state of the visible neurons when their states are determined by the environment, and $P^{'}(V_\alpha)$ is the corresponding probability of hidden units that are not exposed to the environment. Equation (19) acts as the empirical cost function of the Boltzmann Machine. Then, the gradient can be written as

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}(p_{ij} - p_{ij}^{'}) \tag{20}$$

Because T is a parameter shared universally, for simplicity, $\frac{1}{T}$ is omitted. Then, the learning rule can be written as

$$\triangle w_{ij} = \epsilon(p_{ij} - p_{ij}^{'}) \tag{21}$$

or

$$w_{ij} \longleftarrow w_{ij} + \epsilon(p_{ij} - p_{ij}^{'}) \tag{22}$$

Thus, as mentioned earlier, the result of the learning is the probability distribution that most resembles the Boltzmann distribution of the environment to which visible neurons are exposed.

Perceptron

Rosenblatt explains the learning algorithm of Perceptron in terms of probability theory. Even though the wordings, terms and mathematical expressions used by Rosenblatt are distinct from those used in papers written by Ackley, Hinton, Sejnowski, Rumelhart, Williams and Werbos, it is noticeable that Rosenblatt presents theoretical methodology of Feed-Forward, Back Propagation and even advanced method such as 'time step as variable' from which later idea such as BPTT is influenced.

Perceptron is one of the earliest Neural Network that is strongly inspired by a biological nervous system for perceptual recognition. As any neural network or machine learning algorithm does, perceptron is based on the idea of "connectionist", rather than "coded memory theorists" (Rosenblatt, 1958); voltage that flow among the neurons are not represented as coded memory at all, and neurons simply act as on-off switches.

- **Impulse and Voltage:** In this section, impulse and voltage are used interchangeably. Plus, input and stimulus are also used interchangeably.

The set of neurons that corresponds to input layers are called retina, or S-point. S-points receives stimuli from outside of the system, and this impulse is transmitted to the layers of neurons called association cells or A-units. The area that contain these A-units is called ‘projection area’. The set of S-points transmitting impulses to a particular A-unit is called ‘origin points’ of that A-unit. Then again, the impulse is transmitted from A-units to area called association area. For simplicity it is assumed that retina directly connects to the association area, and in further explanation projection area, association area and A-units are used interchangeably. A-unit then transmits the impulse to Response cells $R_1, R_2, ..., R_n$ which correspond to the output layers. The set of A-units transmitting impulses to particular response cell is called source set. Compared to Boltzmann Machine, more severe structural restraints exist in Perceptron; from S-points to A-units, the connections are forward-only (only feed-forward occurs), and there is no feedback. From A-units to response cells, connections are bi-directional, thus have feedback ability (both feed-forward and back propagation occur)

It is assumed that the responses from response cells to A-units are mutually exclusive: Occurrence of R_i would inhibit other R s and their source sets. If impulse from a source set is stronger than others, the R_i that corresponds to the source set is more likely to happen. In a system with such rule, if such impulse-response can be trained, inductively, it is possible to train every A-unit and its connection to response cells. More precisely, it will be possible to modify A-units and their connections such that a class of stimuli evokes stronger impulse in a response cell R_i then in other Rs. By exposing the retina layer to a large sample of stimuli, similar stimuli will tend to form pathways to the same sets of responding cells; the learning occurs.

Training Perceptron is consisted of mainly two different phases: predominant phase where some of A-units respond to the stimuli but R-units are still inactive, and postdominant phase where one R-unit is active and the actions from other R-units and their corresponding source sets are inhibited.

During the predominant phase, response cells are out of question since the voltage has not yet reached them. To observe the learning, two variables are of primary importance: P_α: the expected proportion of A-units activated by a certain stimulus, and P_c: the conditional probability that an A-unit which responds to a given stimulus (S_1) will also respond to another given stimulus (S_2). P_α can be referred as, in neural scientific wording, 'retinal area illuminated by a certain stimuli'. It is observable that P_α reduces when the threshold (θ) increases or the proportion of inhibitory connection increases. In addition, as the gap between excitatory and inhibitory proportion decreases, P_α has less variation for stimuli of different size. P_c shows even sharper decrease when the threshold increases, and it also decreases as the inhibitory connections increases. Note that P_c is practically zero unless stimuli are perfectly identical.

- **Excitatory and inhibitory:** Each response cell has excitatory feedback to its own source set, and inhibitatory feedback to the all A-units that are not in its source set.

The postdominant phase succeeds predominant phase. In postdominant phase, one of the response cells activates and inhibits activity from other response cells and their source sets. Thus, the activity is limited to a single source set and the problem of "which response's source set responds first" should be resolved. There are two solutions to this problem: the μ-system where the response of which input has the biggest mean value responds first, and Σ-system where the response of which input sum is the greatest responds first. Then, the learning experience should be chosen from two different types of hypothetical experiences: first, the experience where Perceptron is forced to give desired response to some series of stimuli thus output the probability of correct choice of response (P_γ), and second, the experience where Perceptron is fed with the stimuli of the same class but not identical to measure the probability of correct generalization (P_g) Then, the choice of value-gain system should also be made among three systems: α system where an active A-unit simply gains the incremented call value V (the probability of completing transmission) for every voltage, β system where each active A-unit gains apportioned V over time, and γ system where an active A-unit increments its V by decreasing V of inactive A-units. However, regardless the choice among these scenarios, the postdominant phase concerns with the probability that the correct response will be preferred over all alternative.

Bivalent System During the learning, it is observed that the active A-unit only increases its cell value V. Even in γ value-gain system where actual decrease of cell value among inactive A-units occur, there is no negative gain in the active A-unit. In Bivalent System, both positive and negative gain in V of the active A-unit can occur, enabling perceptron to perform learning scenario that is similar to reinforcement learning.

Taking 'Time' into the Consideration in Perceptron The idea that the current state of neurons being influenced by the previous time step(s), which is explained in the previous section can also be applied to Perceptron. Theoretically, Perceptron can be trained with sequential data that involves 'time step' such as velocity, given that the inputs leave traces such as altered threshold that causes the activation of A-unit at at time t to be influenced by the activity at time $t-1$.

Conclusion

This paper explains the fundamental mechanism of Neural Network learning; through Feed-Forward and Back Propagation, neurons 'communicate' each other, and adjust the connection among them in a way that the output of the network is trained to resemble the real world (reference) output. An advanced method of such communication, the Back Propagation through time in Recurrent Network is also explained with the information retrieved from ***Backpropagation Through Time: What It Does and How do Do It***. Such method, which is also briefly mentioned in F. Rosenblatt's ***The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*** enables the network to perform temporal pattern recognition.
Neural Networks that are influenced by different fields of sciences are also introduced: David

H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski's Boltzmann Machine, a machine that is designed to train to resemble the Boltzmann Distribution, and F. Rosenblatt's Perceptron, an early stage Neural Network that is inspired by nervous system. Since this paper is aimed to explain these topics in relation to one another, a few exhaustive details of each topics, such as encoder problem, annealing, and mathematical representation of value-gain systems are omitted.

References

Rosenblatt, Frank. “The perceptron: A probabilistic model for information storage and organization in the brain.” Psychological review 65.6 (1958): 386.

Ackley, David H., Geoffrey E. Hinton, and Terrence J. Sejnowski. “A learning algorithm for Boltzmann machines.” Cognitive science 9.1 (1985): 147-169.

Williams, D. R. G. H. R., and Geoffrey Hinton. “Learning representations by back-propagating errors.” Nature 323.6088 (1986): 533-538.

Werbos, Paul J. “Backpropagation through time: what it does and how to do it.” Proceedings of the IEEE 78.10 (1990): 1550-1560.