Homework 3: Back Propagation

Prof. Davi Geiger & TA. Deshana Desai

Due March 27th, 2018

Introduction

Using Backpropagation algorithm to train a two layer XOR problem and a two layer multiplication problem.

Let us define a one hidden layer network with two input units, N hidden layer units and one output unit, with training set of D data samples, using the following notation

1. the input vector for two units as $x^d = (x_1^d, x_2^d)$ or $x_i^d; i = 1, 2, d = 1, \dots, D$. (For the XOR problem introduced below, 4 data points exist in the dataset - $x^1 = (1, 0), x^2 = (1, 1), x^3 = (0, 1), x^4 = (0, 0)$.)

2. the hidden layer with N units as $h_j; j = 1, \dots N$. Each of these units (or neurons) is connected to all the units of the previous layer with a weight w_{ji}^1. Here j refers to the unit index of the hidden layer and the i refers to the unit index of the previous layer.

3. the non-linear function $ReLu(t) = \max(0, t) = \begin{cases} t & t \geq 0 \\ 0 & t < 0 \end{cases}$.

N is a hyper-parameter we will specify for this homework.

Forward Calculations

The output is then computed as follows

$$y(x^d, w^1, w^2, b^2, b^3) = ReLu\left(\sum_{j=1}^{N}\left[w_j^2\, Relu\left(b_j^2 + \sum_{i=1}^{2} w_{ji}^1 x_i^d\right) + b^3\right]\right) \quad (1)$$

Define the variables

1. $o_j^1 = b_j^2 + \sum_{i=1}^{2} w_{ji}^1 x_i^d; \quad j = 1, \dots, N.$

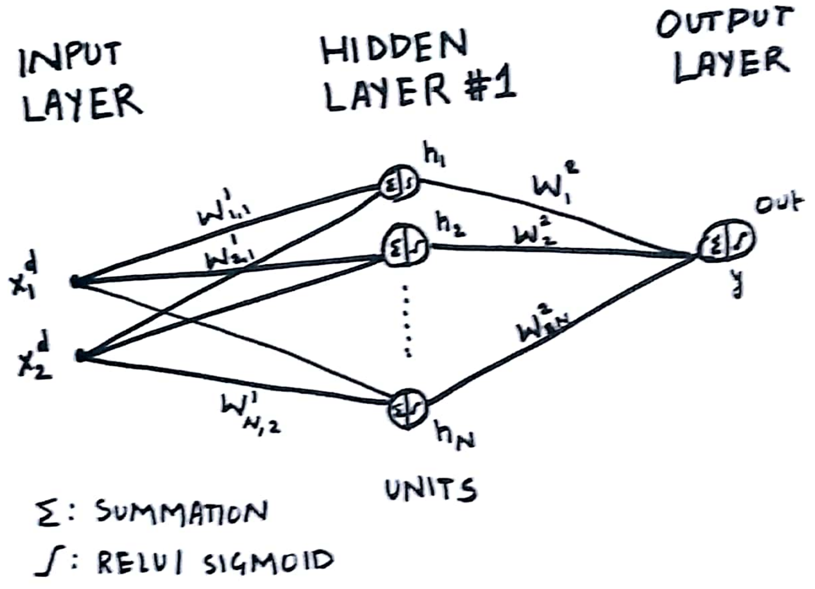

Figure 1: Network architecture.

2. $h_j = Relu(o_j^1) \quad \rightarrow \quad y = ReLu\left(b^3 + \sum_{j=1}^{N} w_j^2 h_j\right).$
3. $o^2 = b^3 + \sum_{j=1}^{N} w_j^2 h_j \quad \rightarrow \quad y = ReLu(o^2)$

In matrix notation we can write equation 1 as

$$y(x^d, w^1, w^2, b^2, b^3) = ReLu(b^3 + w^2 \cdot h) = Relu(b^3 + w^2 \cdot Relu(b^2 + w^1 x^d)) \qquad (2)$$

where b^3 is a scalar, x^d is a vector of length 2, w^2, b^2, h are vectors of length N, w^1 is a matrix of size $N \times 2$, and $v = Relu(u)$ applied to a vector u, is meant to output a vector v where each entry is the result of applying the $Relu$ function to each entry of the vector u.

Loss Functions

Say we have $d = 1, \ldots, D$ labeled examples to be used for training with input and output pairs (x^d, t^d). The loss function we use for training in this homework is

$$E_\lambda(w^1,w^2,b^2,b^3) = \frac{1}{D}\sum_{d=1}^{D} E_\lambda^d(w^1,w^2,b^2,b^3) \tag{3}$$

where

$$E_\lambda^d(w^1,w^2,b^2,b^3) = (y(x^d,w^1,w^2,b^2,b^3) - t^d)^2 + \lambda\left(|w^1|^2 + |w^2|^2 + |b^2|^2 + |b^3|^2\right) \tag{4}$$

where $|w^1|^2 = \sum_{i=1}^{2}\sum_{j=1}^{N}(w_{ji}^1)^2$, $|w^2|^2 = \sum_{j=1}^{N}(w_j^2)^2$, $|b^2|^2 = \sum_{j=1}^{N}(b_j^2)^2$ and λ is a hyper-parameter you estimate in this homework.

Say we have $v = 1,\ldots,V$ data examples for validation, the validation set is then $V_x = \{(x^v,t^v); v = 1,\ldots,V\}$.

In this homework, we also create a set of labeled data for generalization. The generalization set (also said to be the test data) is then $G_x = \{(x^g,t^g); g = 1,\ldots,G\}$

In both cases, the validation loss and generalization loss is simply

$$E_{V_x}(w^1,w^2,b^2,b^3) = \frac{1}{V}\sum_{v=1}^{V}(y(x^v,w^1,w^2,b^2,b^3) - t^v)^2 \tag{5}$$

$$E_{G_x}(w^1,w^2,b^2,b^3) = \frac{1}{G}\sum_{g=1}^{V,G}(y(x^g,w^1,w^2,b^2,b^3) - t^g)^2 \tag{6}$$

Gradient Descent

The general formula for updating the weights and bias is via gradient descent, where the steps are indexed by $\tau = 1,\ldots,T$, where T is the stopping criteria, another hyper-parameter to be estimated in this homework.

$$w^{1,2}(\tau+1) = w^{1,2}(\tau) - \eta\sum_{d=1}^{D}\frac{\partial E^d}{\partial w^{1,2}}(\tau) \qquad b^{2,3}(\tau+1) = b^{2,3}(\tau) - \eta\sum_{d=1}^{D}\frac{\partial E^d}{\partial b^{2,3}}(\tau) \tag{7}$$

where η, the speed of descent/learning is a hyper-parameter to be estimated.

Computing these gradients

In order to compute these formulae more precisely, we will need the derivative of the Relu(t) function, which can be readily derived $\frac{\partial Relu(t)}{\partial t} = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$

Top Layer: We have the N weights w_j^2 and N bias b_j^2 to update as follows

$$\begin{aligned}
\frac{\partial E^d}{\partial w_j^2} &= 2(y-t^d)\frac{\partial y}{\partial w_j^2} + 2\lambda w_j^2 = 2(y-t^d)\frac{\partial Relu(o^2)}{\partial o^2}\frac{\partial o^2}{\partial w_j^2} + 2\lambda w_j^2 \\
&= 2(y-t^d)\frac{\partial Relu(o^2)}{\partial o^2} h_j + 2\lambda w_j^2 \\
&= 2\lambda w_j^2 + \begin{cases} 2(y-t^d)h_j & o^2 \geq 0 \\ 0 & o^2 < 0 \end{cases}
\end{aligned} \tag{8}$$

where we used $y = Relu(o^2)$ and $o^2 = b^3 + \sum_{j=1}^N w_j^2 h_j$. Similarly for the bias b^3

$$\begin{aligned}
\frac{\partial E^d}{\partial b^3} &= 2(y-t^d)\frac{\partial y^d}{\partial b^3} + 2\lambda b^3 = 2(y-t^d)\frac{\partial Relu(o^2)}{\partial o^2}\frac{\partial o^2}{\partial b^3} + 2\lambda b^3 \\
&= 2\lambda b^3 + \begin{cases} 2(y-t^d) & o^2 \geq 0 \\ 0 & o^2 < 0 \end{cases}
\end{aligned} \tag{9}$$

Bottom Layer: We have the $N \times 2$ weights w_{ji}^1 to update as follows

$$\frac{\partial E^d}{\partial w_{ji}^1} = 2(y-t^d)\frac{\partial y^d}{\partial w_{ji}^1} + 2\lambda w_{ji}^1 \tag{10}$$

$$\begin{aligned}
&= 2(y-t^d)\frac{\partial Relu(o^2)}{\partial o^2}\frac{\partial o^2}{\partial h_j}\frac{\partial h_j}{\partial o_j^1}\frac{\partial o_j^1}{\partial w_{ji}^1} + 2\lambda w_{ji}^1 \\
&= 2(y-t^d)\frac{\partial Relu(o^2)}{\partial o^2} w_j^2 \frac{\partial Relu(o_j^1)}{\partial o_j^1} x_i^d + 2\lambda w_{ji}^1 \\
&= 2\lambda w_{ji}^1 + \begin{cases} 2(y-t^d) w_j^2 x_i^d & o_j^1 \geq 0 \,\&\, o^2 \geq 0 \\ 0 & else \end{cases}
\end{aligned} \tag{11}$$

where we used $y = Relu(o^2)$ and $o^2 = b^3 + \sum_{j=1}^N w_j^2 h_j$, $h_j = Relu(o_j^1)$, and $o_j^1 = b_j^2 + \sum_{i=1}^2 w_{ji}^1 x_i^d$. Similarly, for the bias at each hidden layer unit b_j^2 we have

$$
\begin{aligned}
\frac{\partial E^d}{\partial b_j^2} &= 2(y - t^d)\frac{\partial y^d}{\partial b_j^2} + 2\lambda b_j^2 = 2(y - t^d)\frac{\partial Relu(o^2)}{\partial o^2}\frac{\partial o^2}{\partial h_j}\frac{\partial h_j}{\partial o_j^1}\frac{\partial o_j^1}{\partial b_j^2} + 2\lambda b_j^2 \\
&= 2\lambda b_j^2 + 2(y - t^d)\frac{\partial y^d}{\partial b_j^2} + 2\lambda b_j^2 = 2(y - t^d)\frac{\partial Relu(o^2)}{\partial o^2} w_j^2 \frac{\partial Relu(o_j^1)}{\partial o_j^1} \\
&= 2\lambda b_j^2 + \begin{cases} 2(y - t^d)w_j^2 & o_j^1 \geq 0 \,\&\, o^2 \geq 0 \\ 0 & else \end{cases}
\end{aligned} \tag{12}
$$

Algorithm

We have four hyper-parameters to estimate λ (to bias the solution to small weights and not fit the data exactly. Remember, the goal is not training perfectly, but generalizing well), η (speed of learning), T (when to stop learning, so validation/generalization is good even if training error is not perfect). Clearly λ and T work on the same problem at different aspects of it. N is the last one, the number of hidden layer units. This one we will specify on the homework.

The back propagation algorithm works by first (at step $\tau = 0$) initializing all the weights and bias at small random values (with zero mean).

Then we loop for $\tau = 1, \ldots T$ and each step we loop for the training data $d = 1, \ldots, D$ and for each training pair (x^d, t^d) we compute the gradient equation 7 for each of the parameters (weights and bias) using (8), (12), (11) accordingly.

Visualization of the Algorithm

Plot a graph of the average loss $E_\lambda(w^1, w^2, b^2)$ from (3) versus τ. On the same graph, superimpose in a different color the plot of the average validation loss (5) versus τ (here using data from the validation set).

These plots will help you choose the hyper-parameter T, the stoppage step where the validation loss does not decrease.

Finally, after the algorithm is finished (no more iterations), we also want to know the single value of the average generalization (6), using the generalization training set.

Homework: Two Cases

The first case to study is the well known XOR example. For this example, the hidden layer maybe simply made of two units, $N = 2$, and the input and target output are binary values.

There are only four different possible inputs/outputs scenarios, $\left(x^1 = (0,0), t^1 = 0\right)$, $\left(x^2 = (0,1), t^2 = 1\right)$, $\left(x^3 = (1,0), t^3 = 1\right)$, $\left(x^4 = (1,1), t^4 = 0\right)$. Thus, for the training set, validation set and generalization set, for each step τ we only have these four examples.

Show the plots we described in visualization, show the final generalization loss, show all the weights of the network (list them). How to show the final generalization output ? we should be able to run your forward code, with the final weights, and your generalization set, and produce your final outputs.

The second example we ask you to run the multiplication function. The inputs are x_1, x_2 and they are real values. Well, let us restrict to multiplications of integers ? The target output should be $t = x_1 \times x_2$ also an integer number. Note that it should work on negative numbers as well, i.e., $x_1, x_2, t \in \mathbb{Z}$. We may restrict further to $x_1, x_2, t \in (-M, M)$, say M=100. It is easy to generate a large test set, not so large validation set, and an even larger generalization set. Say $D = 1,000$, $V = 200$ and $G = 4,000$. Of course, make sure to include cases of multiplication by zero and by 1.

In this case, the hidden layer should be "large" to have a chance to work. Consider $N = 20$ (I am not sure it will work!). Display the same plots and values as the XOR problem and we should be able to run your forward code, with the final weights, and produce your final outputs.