Parallel Stochastic Gradient Descent for Stacked Autoencoders

Jason Liang
jasonzliang@utexas.edu

Keith Kelly
keith@ices.utexas.edu

***Abstract*—The abstract goes here.**

I. Introduction

Autoencoders are a method for performing representation learning, a process during which a more useful representation of the input data is automatically determined. Representation learning is important in machine learning since “the performance of machine learning methods is heavily dependent on the choice of data representation (or features) in which they are applied” [1]. An autoencoder is a neural network in which the output layer and the input layer have the same size, so that the original input can be used to determine how well or how poorly the autoencoder encodes the data. Consider a neural network with one hidden layer and one output layer which has the same size as the input layer. Suppose that the input $x \in \mathbb{R}^m$ (and the output as well) and suppose that the hidden layer has n nodes. Then we have a weight matrix $W \in \mathbb{R}^{m \times n}$ and bias vectors b and $b^{'}$ in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively. Let $s(x) = 1/(1+e^{-x})$ be the sigmoid (logistic) transfer function. Then we have a neural network as shown in 1. We want that $n < m$ so that the learned representation of the input exists in a lower dimensional space than the input. The output can be considered a prediction of the input given the code in the hidden layer. We can then determine how well the autoencoder encodes (and then decodes) the data by comparing input to the output. If they are similar, then the code in the hidden layer represents a good, lower dimensional representation of the input data.

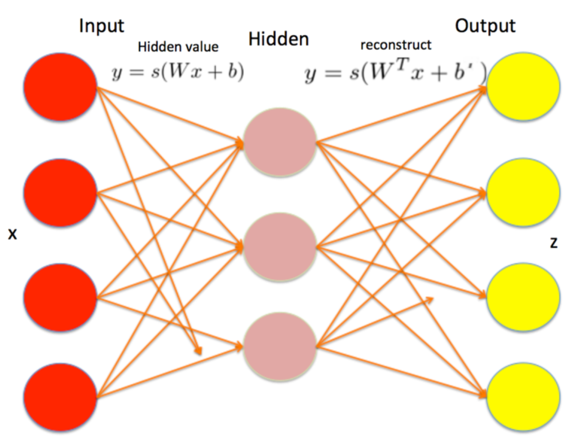

Fig. 1. autoencoder

II. Related Work

related work here, cite like this [2]

III. Algorithm Description

We start with a random weight matrix W and random biases b and $b^{'}$. We take a given input x, and feed it forward through the network and compute the error between the target output t and the actual output z. Often we use the squared loss error $E(t,z) = \frac{1}{2}\|t-z\|_2^2$ to determine the difference between the two. In the case of an autoencoder, the target output is the same as the input. If the error is not satisfactory, we can adjust the weight matrix and that biases in order to attempt to learn a better representation of the data. A common method of updating the weight and biases is via backpropogation. We can use gradient descent to update the weights and biases. We will first consider the update for the weights and biases from the last hidden layer to the output layer with a squared loss error function and derive the updates. We use as an example a simple three layer neural network (one input, one hidden and one output node). Some notation:

Symbol	Meaning
E	Error as computed at the output layer
x_j	Node j in the input layer
y_j	Node j in the hidden layer
z_j	Node j in the output layer
n_j	$\sum_{i=1}^n W_{ij}x_i + b_j$
t_j	Target output at node j
W_{ij}^H	Weight i,j from input to hidden layer
W_{ij}^O	Weight i,j from hidden to output layer
$s(x_j)$	$1/(1+e^{-x_j})$

The derivative of the output error E with respect to an output matrix weight W_{ij}^O is as follows.

$$\begin{aligned}\frac{\partial E}{\partial W_{ij}^O} &= \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial W_{ij}^O} \\ &= (z_j - t_j)\frac{\partial s(n_j)}{\partial x_j}\frac{\partial x_j}{\partial W_{ij}^O} \\ &= (z_j - t_j)s(n_j)(1-s(n_j))x_i \\ &= (z_j - t_j)z_j(1-z_j)x_i\end{aligned} \tag{1}$$

Now that we have the gradient for the error associated to a single training example, we can compute the updates.

$$\begin{aligned}\delta_j^O &= (z_j - t_j)z_j(1-z_j) \\ W_{ij}^O &\leftarrow W_{ij} - \eta\delta_j^O x_i \\ b_j &\leftarrow b_j - \eta\delta_j\end{aligned} \tag{2}$$

The computation of the gradient for the weight matrix between hidden layers is simiilarly easy to compute.

$$\begin{aligned}\frac{\partial E}{\partial W_{ij}^H} &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial W_{ij}^H} \\ &= \left(\sum_{k=1}^{m}\frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial n_k}\frac{\partial n_k}{\partial y_j}\right)\frac{\partial y_j}{\partial n_j}\frac{\partial n_j}{\partial}W_{ij}^H \\ &= \left(\sum_{k=1}^{m}(z_k - t_k)(1-z_k)z_k W_{jk}^O\right)y_j(1-y_j)x_i\end{aligned} \tag{3}$$

And then using the computed gradient we can define the updates to be used for the hidden layers

$$\begin{aligned}\delta_j^H &= \left(\sum_{k=1}^{m}(z_k - t_k)(1-z_k)z_k W_{jk}^O\right)y_j(1-y_j) \\ W_{ij}^H &\leftarrow W_{ij}^H - \eta\delta_j^H x_i \\ b_j &\leftarrow b_j - \eta\delta_j^H\end{aligned} \tag{4}$$

In general, for a neural network we may have different output error function and these will result in different update rules. We will also give the updates for the cross-entropy error function with softmax activation in the final layer. The cross entropy error function is given by $E(x,t) = -\sum_{i=1}^{n}\left(t_i \ln z_i + (1-t_i)\ln(1-z_i)\right)$ and the softmax function is given by $\sigma(x_j) = e^{x_j}/(\sum_k e^{x_k})$. Following the same procedure as above for computing the gradient and the updates, we find that for hidden/output layer

$$\begin{aligned}\frac{\partial E}{\partial W_{ij}^O} &= (z_j - t_j)y_i \\ \delta_j^O &= (z_j - t_j) \\ W_{ij}^O &\leftarrow W_{ij} - \eta\delta_j^O x_i \\ b_j &\leftarrow b_j - \eta\delta_j.\end{aligned} \tag{5}$$

Note also that we find that the updates for the input/hidden layer is the same as in the squared error loss function with sigmoid activation.

The algorithm and derivations for the autoencoder (a special type of neural network) are a slight variation on the above derivations for a more general neural network.

We give the whole algorithm in algorithm 1.

IV. Experimental Results

experimental results here

V. Future Work

future work goes here

VI. Conclusion

conclusion goes here

Algorithm 1 Backpropogation

```
Initialize the weights and biases randomly
for iter = 1, 2, 3... do
    for all Examples x in training set (randomize) do
        z ← Feedforward x
        Compute output layer δ_j^O
        W_ij ← W_ij − η δ_j^O x_i
        b_j ← b_j − η δ_j^O
        for all Layers in reverse order do
            Compute hidden layer delta δ_k^H
            W_ij^H ← W_ij^H − η δ_j^H x_i
            b_j ← b_j − η δ_j^H
        end for
    end for
end for
```

References

[1] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[2] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.