Neural Network Notes

Kevin J. Marshall

July 6, 2019

1 Feedforward Neural Networks

The basic computational units of a neural network (NN) are called neurons. Neurons are grouped together in finite sets called layers. The structure of a neural network consists of an input layer followed by a finite set of hidden computational layers and a final output layer. In feedforward NNs, information flows in a single direction from the inputs to the outputs and the computation graph may always be interpreted as a directed acyclic graph (DAG). Since the computational graph is a DAG, the neural network may always be arranged such that each layer of the feedforward NN only depends on the layer to its left (the input side). This can be accomplished by performing a topological sort on the neurons (nodes) of the DAG and grouping them into layers l_k with depth k where $l_k \equiv \{n_0, \ldots, n_{k-1}\}$ is the set of all neurons in layer k. Upon arranging the K layers from $l_0, l_1, \ldots, l_K$ each layer's neuron set l_k will only depend on neurons from the set of layers $\{l_{j<k}\}$.

1.1 Matrix Representation

During training, a set of N vector valued input-output pairs $\{(\boldsymbol{x}_0, \boldsymbol{y}_0), \ldots, (\boldsymbol{x}_{N-1}, \boldsymbol{y}_{N-1})\}$ is fed into the network. For each layer, k, the following computation is carried out,

$$o_j^k = w_{ij}^k a_i^{k-1} \tag{1}$$

$$a_j^k = f(o_j^k) \tag{2}$$

where we have used Einstein summation notation in Eqns. 1 and 2 and where the associated vector expressions are

$$\boldsymbol{o}^k = (\boldsymbol{W}^k)^T \boldsymbol{a}^{k-1} \tag{3}$$

$$\boldsymbol{a}^k = f(\boldsymbol{o}^k) \tag{4}$$

In these equations

- o_j^k is the output computation of neuron j in layer k and $\boldsymbol{o}^k$ is a vector with row dimension $|n_k + 1|$ including a bias term.
- a_j^k is the nonlinear output or activation of neuron j in layer k after passing through the nonlinear function $f(\cdot)$
- $\boldsymbol{W}^k$ is the weight matrix of layer k such that elements W_{ij}^k describe the weight associated with the i'th neuron input in layer $k-1$ to the j'th neuron in layer k. The dimensions of the weight matrix are $(d_{k-1}+1) \times d_k$ where $d_{k-1} = |l_{k-1}|$ and $d_k = |l_k|$ are the number of neurons in layers $k-1$ and k and $+1$ accounts for weights associated with the bias neuron.

- $f(\cdot; j, k)$ is a nonlinear activation function $f : \mathbb{R}^{n_k} \to \mathbb{R}^{n_k}$ which may be different for each layer and each neuron. Historically this function is taken to be the same for all neurons in all hidden layers, $f(\cdot; j, k) = f(\cdot)$, and often takes the form of the logistic function

$$f(x) = \frac{A}{1 + \exp(-r(x - x_0))} \tag{5}$$

  where A is the curve's maximum value, x_0 is the x-value of the functions midpoint, and r is the logistic growth rate (steepness of the curve). Standard practice uses $A = 1, r = 1, x_0 = 0$ which centers the function at $x = 0$ and scales the input between to $(0, 1)$ however the shape of this curve may adjusted in order to transform and/or scale layer outputs.

Once the information flow reaches the output layer, l_K, an error may be computed for each input based on the nodal output values $\hat{y}_n$ and the target values y_n. Gradient decent techniques attempt to minimize the error by adjusting the weights. Weight update rules involve computing

$$\begin{aligned}
\frac{\partial E}{\partial w_{ij}^k} &= \frac{\partial E}{\partial o_l^k} \frac{\partial o_l^k}{\partial w_{ij}^k} \\
&= \frac{\partial E}{\partial o_l^k} \frac{\partial}{\partial w_{ij}^k} \left(w_{ml}^k o_m^{k-1} \right) \\
&= \frac{\partial E}{\partial o_l^k} \delta_{mi} \delta_{jl} o_m^{k-1} \\
&= \delta_j^k o_i^{k-1}
\end{aligned} \tag{6}$$

1.2 Bias

A bias term has been hidden in Eqn. 3 such that $\boldsymbol{W}^T$ is given by

$$(\boldsymbol{W}^k)^T \boldsymbol{a}^{k-1} = \begin{bmatrix} w_{00} & w_{10} & \dots & w_{n_{k-1}0} & w_{b_{k-1}0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{0n_k} & w_{1n_k} & \dots & w_{n_{k-1}n_k} & w_{b_{k-1}n_k} \end{bmatrix} \begin{bmatrix} a_0^{k-1} \\ \vdots \\ a_{n_{k-1}}^{k-1} \\ 1 \end{bmatrix} \tag{7}$$

for $|l_k| = n_k$ and $|l_{k-1}| = n_{k-1}$ neurons in layers k and $k-1$ and a bias weight vector of $[w_{b_{k-1}0}, \dots, w_{b_{k-1}n_k}]^T$. In general, layers contain a decreasing number of nodes as the depth increases, i.e. $|l_k| \leq |n_{k-1}|$.

1.3 Activation Functions

Activation functions act on neuron layer outputs and may be interpreted as individual layers themselves. These functions introduce non-linearities into the prediction model. Several common activation functions and their derivatives are given below. In what follows we assume $\boldsymbol{o}^k$ to be the input to the activation layer and $\boldsymbol{a}^k$ to be the associated output.

- Identity:

$$a_i^k = f(o_i^k) = o_i^k \tag{8}$$

$$\frac{\partial a_i^k}{\partial o_i^k} = 1 \tag{9}$$

- Logistic:

$$a_i^k = f(o_i^k) = \frac{1}{1+\exp(-o_i^k)} \tag{10}$$
$$\frac{\partial a_i^k}{\partial o_i^k} = \frac{\exp(-o_i^k)}{(1+\exp(-o_i^k))^2} = f(o_i^k)(1-f(o_i^k)) \tag{11}$$

- TanH:

$$a_i^k = f(o_i^k) = \tanh(o_i^k) \tag{12}$$
$$\frac{\partial a_i^k}{\partial o_i^k} = \operatorname{sech}^2(o_i^k) = 1 - f^2(o_i^k) \tag{13}$$

- arcTan:

$$a_i^k = f(o_i^k) = \arctan(o_i^k) \tag{14}$$
$$\frac{\partial a_i^k}{\partial o_i^k} = \frac{1}{(o_i^k)^2+1} \tag{15}$$

- ReLU (rectified linear unit)

$$a_i^k = f(o_i^k) = \begin{cases} o_i^k, & o_i^k \geq 0 \\ 0, & o_i^k < 0 \end{cases} \tag{16}$$
$$\frac{\partial a_i^k}{\partial o_i^k} = \begin{cases} 1, & f(o_i^k) \geq 0 \\ 0, & f(o_i^k) < 0 \end{cases} \tag{17}$$

- PLU (parametric linear unit):

$$a_i^k = f(o_i^k) = \begin{cases} \alpha o_i^k, & o_i^k \geq 0 \\ 0, & o_i^k < 0 \end{cases} \tag{18}$$
$$\frac{\partial a_i^k}{\partial o_i^k} = \begin{cases} \alpha, & f(o_i^k) \geq 0 \\ 0, & f(o_i^k) < 0 \end{cases} \tag{19}$$

- ELU (exponential linear unit):

$$a_i^k = f(o_i^k) = \begin{cases} \alpha(\exp(o_i^k)-1), & o_i^k \geq 0 \\ 0, & o_i^k < 0 \end{cases} \tag{20}$$
$$\frac{\partial a_i^k}{\partial o_i^k} = \begin{cases} f(o_i^k)+\alpha, & f(o_i^k) \geq 0 \\ 0, & f(o_i^k) < 0 \end{cases} \tag{21}$$

- SoftMax:

$$a_i^k = f(o_i^k) = \frac{\exp(o_i^k)}{\sum_j \exp(o_j^k)} \tag{22}$$
$$\frac{\partial a_i^k}{\partial o_j^k} = f(o_i^k)\delta_{ij} - f(o_i^k)f(o_j^k) \tag{23}$$

- LogSoftMax:

$$a_i^k = f(o_i^k) = o_i^k - \log\left(\sum_j \exp(o_j^k)\right) \tag{24}$$

$$\frac{\partial a_i^k}{\partial o_j^k} = \delta_{ij} - \exp_i(f(o_j^k)) \tag{25}$$

1.4 Loss Functions

1.4.1 Mean Squared Error

The mean squared error is a good classifier for the numeric output in which case weight updates are computed by minimizing the error

$$\begin{aligned} E &= \frac{1}{2N}\sum_n (\hat{\boldsymbol{y}}^n - \boldsymbol{y}^n)^2 \\ &= \frac{1}{2N}\sum_n (f(\boldsymbol{a}^K; K)^n - \boldsymbol{y}^n)^2 \\ &= \frac{1}{N}\sum_n E_n \end{aligned} \tag{26}$$

where $\hat{\boldsymbol{y}}^n$ and $\boldsymbol{y}^n$ are respectively the output (prediction) vector and target vector of the n'th input $(\boldsymbol{x}^n, \boldsymbol{y}^n)$. For the output layer we find that Eqn. 6 becomes

$$\begin{aligned} \frac{\partial E_n}{\partial w_{ij}^K} &= \frac{\partial E_n}{\partial \hat{y}_l^K}\frac{\partial \hat{y}_l^K}{\partial o_m^K}\frac{\partial \hat{o}_m^K}{\partial w_{ij}^K} \\ &= (\hat{y}_l^K - y_l^p) f'(o_l^K)\delta_{ml}\delta_{in}\delta_{jm}a_n^{K-1} \\ &= (\hat{y}_j^K - y_j^p) f'(o_j^K) a_i^{K-1} \end{aligned} \tag{27}$$

We note that equation 27 actually describes three layer operations which take place sequentially. These operations may be described from right to left in accordance with back propagation as

1. Error propagation over the loss function layer,

$$\frac{\partial E^p}{\partial \hat{y}_l^p} = (\hat{y}_l^K - y_l^p) \tag{28}$$

2. Error propagation over the activation layer

$$\frac{\partial E^p}{\partial a_m^K} = \frac{\partial E^p}{\partial \hat{y}_l^p} f'(o_l^K)\delta_{ml} = \frac{\partial E^p}{\partial \hat{y}_m^p} f'(o_m^K) \tag{29}$$

3. Error propagation over the last layer K to find gradient updates

$$\frac{\partial E^p}{\partial w_{ij}^K} = \frac{\partial E^p}{\partial o_m^K}\delta_{in}\delta_{jm}a_n^{K-1} = \frac{\partial E^p}{\partial o_j^K} a_i^{K-1} \tag{30}$$

The key here is to notice that back propagation inputs are error gradients on the forward pass inputs. Gradient update rules for weight matrices are obtained

1.5 Hidden Layers

For hidden layers l_k, $0 \leq k < K$, the calculation becomes,

$$\begin{aligned}
\frac{\partial E_n}{\partial w_{ij}^k} &= \frac{\partial E_n}{\partial o_l^k}\frac{\partial o_l^k}{\partial w_{ij}^k} \\
&= \frac{\partial E_n}{\partial o_m^{k+1}}\frac{\partial o_m^{k+1}}{\partial o_l^k}\frac{\partial o_l^k}{\partial w_{ij}^k} \\
&= \frac{\partial E_n}{\partial o_m^{k+1}}\frac{\partial o_m^{k+1}}{\partial o_l^k}\delta_{jl}a_i^{k-1} \\
&= \frac{\partial E_n}{\partial o_m^{k+1}}\frac{\partial o_m^{k+1}}{\partial o_j^k}a_i^{k-1} \\
&= \frac{\partial E_n}{\partial o_m^{k+1}}\frac{\partial o_m^{k+1}}{\partial a_p^k}\frac{\partial a_p^k}{\partial o_j^k}o_i^{k-1} \\
&= \frac{\partial E_n}{\partial o_m^{k+1}}w_{pm}^{k+1}f'(o_p^k)\delta_{pj}o_i^{k-1} \\
&= \delta_m^{k+1}w_{jm}^{k+1}f'(a_j^k)o_i^{k-1}
\end{aligned} \tag{31}$$

2 Layer Based Architecture

The neural network architecture described above has grouped activation functions in with network processing layers. Insight into the back propagation algorithm may be obtained by separating these layer computations.

The output layer takes y

3 Things to do...

- Need some way to get data in. If using MINST we need to research how to import $28\times 28 = 756$ pixels.
- Design API to connect layers. If we assume layers are fully connected we have to initialize layers from left to right or right to left. For example, 756 inputs may funnel into hidden layer l_0 which could have 100 neurons. The weight matrix would then be 756×100 which is the same as $\#inputs \times n_0$. The second hidden layer could then have 50 neurons leading to a weight matrix $\boldsymbol{W}^1$ of dimensions $(n_0 + 1) \times n_1$. Each weight matrix $\boldsymbol{W}^k$ would have dimensions $(n_{k-1} + 1) \times n_k$ with an output vector, $\boldsymbol{o}^k$ of dimension n_k.
- Need to understand learning algorithm. Run feed forward on a batch set of training examples then back propagate error.
- Test on MINST, then get threading/CUDA working...