Notes on MSE Gradients for Neural Networks

David Meyer

dmm@{1-4-5.net,uoregon.edu,...}

April 21, 2017

1 Introduction

2 Mean Squared Error (MSE)

First, notation: scalars are represented in regular math font, e.g., y_i, where vectors are in bold, e.g., $\mathbf{x}_i$. Given these definitions we can define our *labelled data* or *training examples* as a set of n tuples where the i^{th} tuple has the form $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathbb{R}^n$ is a vector of inputs and $y_i \in \mathbb{R}$ is the observed output.

Ideally our neural network should output y_i when given $\mathbf{x}_i$ as an input. Of course, during training this doesn't always happen so we need to define an *error or cost* function that quantifies the difference between the actual observed output and the prediction of the neural network. A simple measure of the error is the Mean Squared Error, or MSE. We define the MSE as follows:

$$E := \frac{1}{m} \sum_{i=1}^{m} (h(\mathbf{x}_i) - y_i)^2 \tag{1}$$

where $h(\mathbf{x}_i)$ is the output of the neural network.

3 Basic Building Blocks: Perceptrons

The simplest classifiers out of which we will build our neural network are perceptrons [1]. In reality, a perceptron is a linear classifier. A perceptron takes an input vector $\mathbf{x}$ which is multiplied pairwise by a weight vector $\mathbf{w}$, then sums the products up together with a bias term b. This sum (the *dot product*, Equation 3) is then fed through an activation function

$\sigma : \mathbb{R}, \mathbb{R}$. This is depicted in Figure 1. Note here that $w_0 = b$ and $a_0 = 1$; I like Figure 1 but I will use the more conventional $\mathbf{x}$ for the input vector rather than $\mathbf{a}$ as is used in the figure. The behavior of the perceptron can then be described as $\sigma(\mathbf{w} \cdot \mathbf{x})$, where $\mathbf{w}$ and $\mathbf{x}$ have the following form[1]:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$\mathbf{w}^{\mathrm{T}}$ ($\mathbf{w}$ transpose) is defined to be

$$\boldsymbol{w}^T = \begin{bmatrix} w_1, & w_2, & \ldots, & w_n \end{bmatrix}$$

The *dot product* between $\mathbf{w}$ and $\mathbf{x}$, $\mathbf{w} \cdot \mathbf{x}$, is defined as

$$\mathbf{w} \cdot \mathbf{x} = \boldsymbol{w}^T \boldsymbol{x} = \begin{bmatrix} w_1, & w_2, & \cdots, & w_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Note that the weight vector, $\mathbf{w}$ will be a $M \times N$ matrix if there is more than one layer of artificial neurons.

The last piece of the puzzle are the kinds of activation functions[2] that σ might be:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent: $\sigma(x) = \tanh(x)$

[1] Again noting that $\mathbf{a}$ in Figure 1 is frequently called $\mathbf{x}$, the input vector; I'll use $\mathbf{x}$ here.
[2] Activation functions are sometimes called *link* functions in a Generalized Linear Model setting.

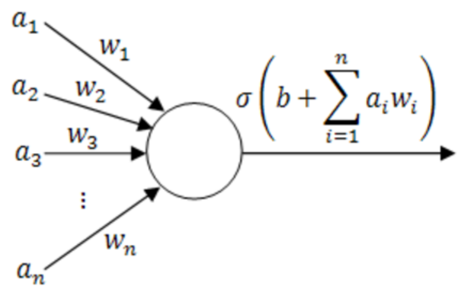

Figure 1: Basic Perceptron/Linear Classifier

- Linear: $\sigma(x) = x$
- Rectified Linear Unit: $\sigma(x) = \max(0, x)$
- Exponential Linear Unit: $\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{otherwise} \end{cases}$
- ...

4 Building a Single Layer Neural Network

So far we've defined the error E as the MSE, namely, $E := \frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}_i) - y_i)^2$. Here both the error and the output of the network ($h_{\mathbf{w}}(\mathbf{x}_i) = \sigma(\mathbf{w} \cdot \mathbf{x}_i)$) depend on the weight vector $\mathbf{w}$. We write the error function, parameterized by $\mathbf{w}$, as

$$E(\mathbf{w}) := \frac{1}{m}\sum_{i=1}^{m}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 \tag{2}$$

Now, our goal is to find a weight vector $\mathbf{w}$ such that $E(\mathbf{w})$ is minimized. In effect this means that the perceptron will correctly predict the output for the inputs in the training set. Of course, we want the perceptron to *generalize*, so that it makes correct predictions on the test set and on new examples. But how to do this minimization?

We do the minimization by applying the *gradient descent* algorithm. In effect we will treat the error as a surface in n-dimensional space and search for the greatest downwards slope at the current point $\mathbf{w}_t$ and will go in that direction to obtain $\mathbf{w}_{t+1}$. Following this process we will hopefully find a minimum point on the error surface and we will use the coordinates

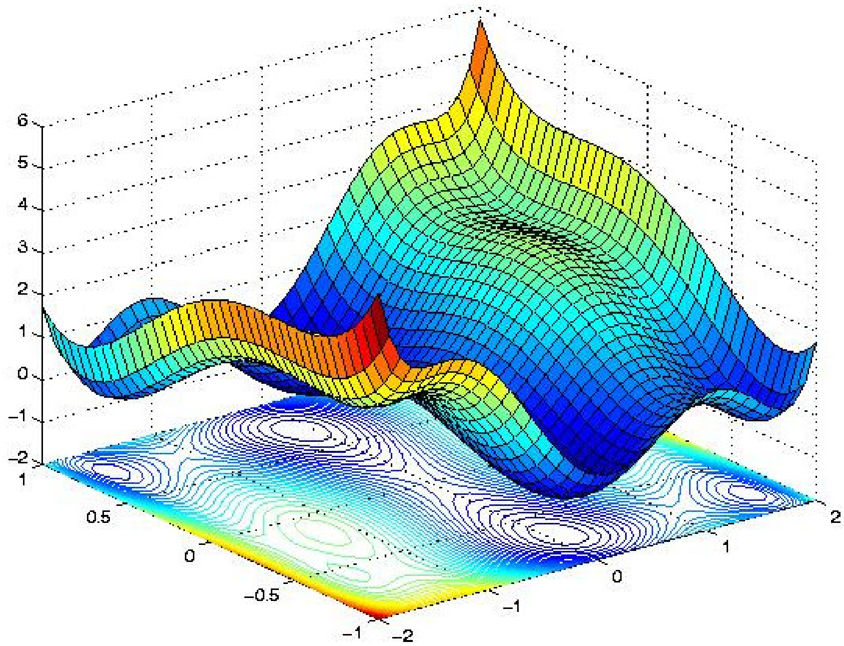

Figure 2: Non-Convex Error Surface

of that point as the final weight vector[3]. In any event, the update rule can be stated as follows (in both *partial derivative* and *gradient* notations):

$$
\begin{aligned}
\mathbf{w}_{t+1} &:= \mathbf{w}_t - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} && \text{\# partial derivative notation} \quad (3)\\
\mathbf{w}_{t+1} &:= \mathbf{w}_t - \eta \nabla_{\mathbf{w}} E(\mathbf{w}) && \text{\# gradient (nabla) notation} \quad (4)
\end{aligned}
$$

where η is the *learning rate*. Now, notice that the *gradient* of E on w is

$$
\nabla_w E(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{\partial E(\mathbf{w})}{\partial \mathbf{w_0}}, \frac{\partial E(\mathbf{w})}{\partial \mathbf{w_1}}, \cdots, \frac{\partial E(\mathbf{w})}{\partial \mathbf{w_n}} \right] \tag{5}
$$

Now we can calculate the gradient, $\nabla_{\mathbf{w}} E(\mathbf{w})$. We start by calculating $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w_j}}$ for each j. So first.... Note that the *chain rule* states that if $h(x) = f(g(x))$ then the derivative $\frac{dh(x)}{dx} = h'(x) = f'(g(x))g'(x)$. We will also use the *power rule*: If $y = u^n$, then $\frac{dy}{dx} = nu^{n-1}\frac{du}{dx}$. So

[3]Consider, however, the situation in which the error surface is non-convex, such as is in Figure 2.

the partial derivative $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w_j}}$ can be computed as follows: So for example element of the gradient $0 \leq j \leq n$

$$
\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w_j}} &= \frac{\partial}{\partial w_j} \frac{1}{m} \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 && \text{\# definition of } E && (6) \\
&= \frac{1}{m} \sum_{i=1}^{m} 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \frac{\partial}{\partial w_j} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) && \text{\# power rule} && (7) \\
&= \frac{1}{m} \sum_{i=1}^{m} 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \frac{\partial}{\partial w_j} \sigma(\mathbf{w} \cdot \mathbf{x}_i) && \text{\# } h_{\mathbf{w}}(\mathbf{x}_i) = \sigma(\mathbf{w} \cdot \mathbf{x}_i) && (8) \\
&= \frac{1}{m} \sum_{i=1}^{m} 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \sigma'(\mathbf{w} \cdot \mathbf{x}_i) \frac{\partial}{\partial w_j} \mathbf{w} \cdot \mathbf{x}_i && \text{\# chain rule} && (9) \\
&= \frac{1}{m} \sum_{i=1}^{m} 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \sigma'(\mathbf{w} \cdot \mathbf{x}_i) \frac{\partial}{\partial w_j} \sum_{k=1}^{n} w_k x_{i,k} && \text{\# defn dot product} && (10) \\
&= \frac{1}{m} \sum_{i=1}^{m} 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \sigma'(\mathbf{w} \cdot \mathbf{x}_i) x_{i,j} && \text{\# } \frac{\partial w_k x_{i,k}}{\partial w_j} \neq 0 \text{ when } k = j && (11)
\end{aligned}
$$

Note that going from Equation 7 to Equation 8 uses the *sum rule*

$$\frac{d}{dx}(f(x) + g(x)) = \frac{d}{dx} f(x) + \frac{d}{dx} g(x) \tag{12}$$

Here $f(x) = h_{\mathbf{w}}(\mathbf{x}_i)$ and $g(x) = -y_i$. $\frac{\partial}{\partial w_j} h_{\mathbf{w}}(\mathbf{x}_i) = \frac{\partial}{\partial w_j} \sigma(\mathbf{w} \cdot \mathbf{x}_i)$ and $\frac{\partial y_i}{\partial w_j} = 0$, so we're left with term $\frac{\partial}{\partial w_j} \sigma(\mathbf{w} \cdot \mathbf{x}_i)$ as we see in Equation 8.

Now, using the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$, who's derivative $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, gives us

$$
\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w_j}} &= \frac{2}{m} \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \sigma'(\mathbf{w} \cdot \mathbf{x}_i) x_{i,j} && (13) \\
&= \frac{2}{m} \sum_{i=1}^{m} (\sigma(\mathbf{w} \cdot \mathbf{x}_i) - y_i) \sigma(\mathbf{w} \cdot \mathbf{x})(1 - \sigma(\mathbf{w} \cdot \mathbf{x})) x_{i,j} && (14) \\
& && (15)
\end{aligned}
$$

Now, we can compute the gradient $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ as follows:

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \frac{2}{m}\sum_{i=1}^{m}(\sigma(\mathbf{w}\cdot\mathbf{x}_i) - y_i)\sigma(\mathbf{w}\cdot\mathbf{x}_i)(1-\sigma(\mathbf{w}\cdot\mathbf{x}_i))\mathbf{x}_i \tag{16}$$

$$\tag{17}$$

Finally, let the update rate $\eta = 0.1$. Then the update to $\mathbf{w}$ is computed as

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \frac{0.2}{m}\sum_{i=1}^{m}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)h_{\mathbf{w}}(\mathbf{x}_i)(1-h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i \tag{18}$$

where $h_{\mathbf{w}}(\mathbf{x}_i) = \sigma(\mathbf{w}_t \cdot \mathbf{x}_i)$.

5 What About Multilayer Networks?

Consider a more general multilayer neural network, such as shown in Figure 3. Here we are using the notation $w_{i,j}$ to denote the weights on the connection between perceptrons (nodes) i and j. Note that the notation $w_{i\to j} \equiv w_{i,j}$ in Figure 3. Now, armed with this notation we can write the sum of the inputs to perceptron (node) j as

$$s_j := \sum_k z_k w_{k,j} \tag{19}$$

Here k iterates over all the perceptrons connected to j. The output of j is written as $z_j = \sigma(s_j)$, where σ is j's activation (link) function.

Now, we can use the same error (cost) function for the multlayer network, $E(\mathbf{w})$

$$E(\mathbf{w}) := \frac{1}{m}\sum_{i=1}^{m}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 \tag{20}$$

except that now $\mathbf{w}$ is a matrix that contains all the weights for the network:
$\mathbf{w} = [w_{i,j}]\ \forall i, j$.

The goal is again to find the $\mathbf{w}$ that minimizes $E(\mathbf{w})$ using gradient descent. So we need to calculate $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$. The first step is to separate the contributions of each of the m training examples using the following observation:

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial E_i(\mathbf{w})}{\partial \mathbf{w}} \tag{21}$$

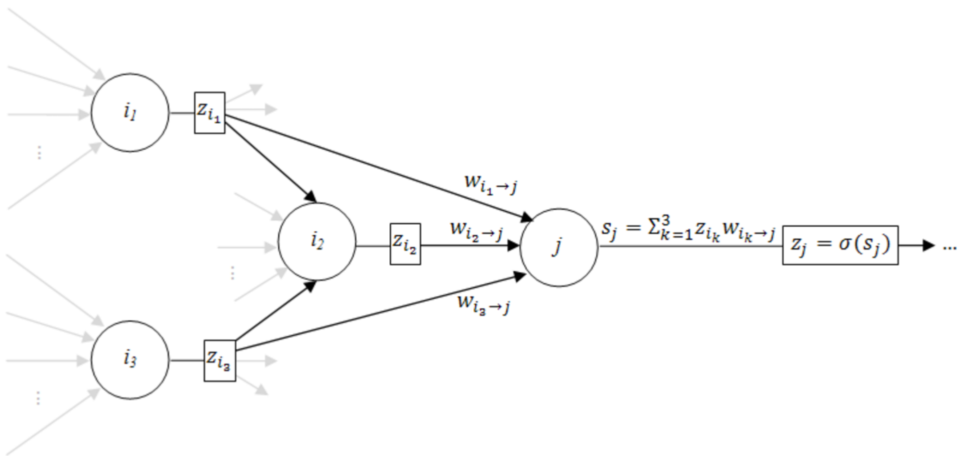

Figure 3: Multi-Layer Perceptron

where $E_i(\mathbf{w}) = (h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$. Then

$$\begin{aligned}
\frac{\partial E_i(\mathbf{w})}{\partial w_{j,k}} &= \frac{\partial}{\partial w_{j,k}}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 && \text{\# definition of } E && (22)\\
&= 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)\frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial w_{j,k}} && && (23)\\
&= 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)\frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_k}\frac{\partial s_k}{\partial w_{j,k}} && \text{\# chain rule} && (24)\\
&= 2(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)\frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_k} z_j && && (25)
\end{aligned}$$

Note that in going from Equation 24 to Equation 25, $s_k = \sum_i z_i w_{i,k}$, so $\frac{\partial s_k}{\partial w_{j,k}} \neq 0$ where $i = j$ and 0 otherwise.

Now, if the k^{th} node is an output node, then

$$\frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_k} = \frac{\partial \sigma(s_k)}{\partial s_k} = \sigma'(s_k) \tag{26}$$

so that

$$\frac{\partial E_i(\mathbf{w})}{\partial w_{j,k}} = 2(h\mathbf{w}(\mathbf{x}_i) - y_i)\sigma'(s_k)z_j \tag{27}$$

On the other hand, if k is not an output node, then changes to s_k can affect all the nodes which are connected to k's output, as follows

$$\begin{aligned}
\frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_k} &= \frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial z_k}\frac{\partial z_k}{\partial s_k} && \text{\# chain rule again} && (28)\\
&= \frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial z_k}\sigma'(s_k) && \# \; z_k = \sigma(s_k) && (29)\\
&= \sum_{o\in\{v|v\to k\}} \frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_o}\frac{\partial s_o}{\partial z_k}\sigma'(s_k) && \# \; v \text{ is connected to } k && (30)\\
&= \sum_{o\in\{v|v\to k\}} \frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_o} w_{k,o}\sigma'(s_k) && \# \; s_o = \sum_i z_i w_{i,o} && (31)
\end{aligned}$$

Note that in going from Equation 30 to Equation 31 we see that

$$\frac{\partial s_o}{\partial z_k} = \frac{\partial}{\partial z_k}\sum_i z_i w_{i,o} \tag{32}$$

which is only non-zero when $i = k$, so that $\frac{\partial s_o}{\partial z_k} = w_{k,o}$ (Equation 31).

So what is left is to calculate s_k and z_k (feeding forward) and then work backwards from the output calculating $\frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_k}$ and back propagate the error down the network ("backprop"). The summary looks like:

- k is an output node: $\frac{\partial E_i(\mathbf{w})}{\partial w_{j,k}} = 2(h\mathbf{w}(\mathbf{x}_i) - y_i)\sigma'(s_k)z_j$
- otherwise: $\frac{\partial E_i(\mathbf{w})}{\partial w_{j,k}} = 2(h\mathbf{w}(\mathbf{x}_i) - y_i)\sigma'(s_k)z_j \sum_{o\in\{v|v,k\}} \frac{\partial h_{\mathbf{w}}(\mathbf{x}_i)}{\partial s_o} w_{k,o}$

Using these results we see that

$$\frac{\partial E_i(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{\partial E_i(\mathbf{w})}{\partial w_{j,k}}\right] \; \forall j,k \tag{33}$$

Finally, the weights can be updated in batch mode, in which case the update rule for batch size of m is

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \tag{34}$$

$$:= \mathbf{w}_t - \eta \sum_{i=1}^{m} \frac{\partial E_i(\mathbf{w})}{\partial \mathbf{w}} \tag{35}$$

Or if we take a Stochastic Gradient Descent (SGD) approach (one training example at a time):

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \tag{36}$$

6 Acknowledgements

Thanks to Armin Wasicek for his careful reading of earlier versions of this document.

References

[1] F. Rosenblatt. http://psycnet.apa.org/psycinfo/1959-09865-001, Nov 1958.