Backpropagation

Sejin Nam
University of Hawaii at Manoa

Preface

An **artificial neural network**, in short **ANN**, is a machine learning model inspired by biological neurons found in our brains. As with any machine learning models, ANN model parameters must be optimized by training on large datasets in order to make good predictions. For ANN, **Gradient Descent** (a generic optimization that tweaks parameters iteratively in order to minimize a cost function) is used to find those optimized parameters. In the context of ANN, **backpropagation** is simply a Gradient Descent that computes the gradient of the network's errors with regard to every single model parameter efficiently. In this article, I will show my own derivation of backprogation of sequential ANN model. Not every variable is represented by conventional letters, and some variables and terms might have not even been defined before in machine learning community. It is implied that all materials that I did not come up with in this article cites the book by Aurelien Geron, Hands-on Machine Learning with Scikit-learn, Keras, and Tensorflow [1].

Contents

1 Artificial Neural Network

An ANN, just like other machine learning models, takes a **model input** x (a vector or column matrix containing scalars called **features**), which might have a **target** (or label) value y, and returns a **model output** $h(x)$ (h is called hypothesis or prediction function, and is often called interchangeably with a term model). The model output is either a scalar or a vector. We must first discuss what ANN is before we talk about backpropagation. And before we discuss what ANN is in detail, we should talk about the most fundamental component of ANN: Artificial neuron or neuron in short.

1.1 Neuron

A **neuron**, in the context of ANN, is a mathematical function which takes an input vector and outputs a scalar (the input vector is a vector containing scalar outputs from other neurons). A neuron is graphically represented as a circle and scalars are represented as arrow sticks in the figure 1.1.

There are two special neurons called **bias neuron** and **input neuron** which take no input vector. As their outputs, a bias neuron always returns 1 and an input neuron returns a feature value. Apart from those two neurons, a regular neuron is composed of connection weights (or weights for short), bias, matrix multiplication, and activation (transfer) function (how those components transform the input into the neural output is described below).

1.2 Neural Variables

Now let us define variables for a neuron that takes an input vector of M dimensions (M is always greater than 1).

- a = input vector or input for short
- a_i = i-th component of a
- w = weight vector
- w_j = j-th component of w (weight for short) where $1 \leq j \leq M - 1$
- $b = w_M$ = bias
- o = output scalar
- g = activation function

The indexing for vector and matrix is one-based in this article, i.e., indices of a and w start from 1 and all the indices are natural numbers. The input always contains 1 from a bias neuron, and in my definition it will always be the last element of the input, i.e., $a_M = 1$. The number of weights is one less than the number of elements of the input.

1.3 Neuron Output

Let $\mathbf{a}$ be a column matrix of a and $\mathbf{w}$ be a row matrix of w and b for a neuron:

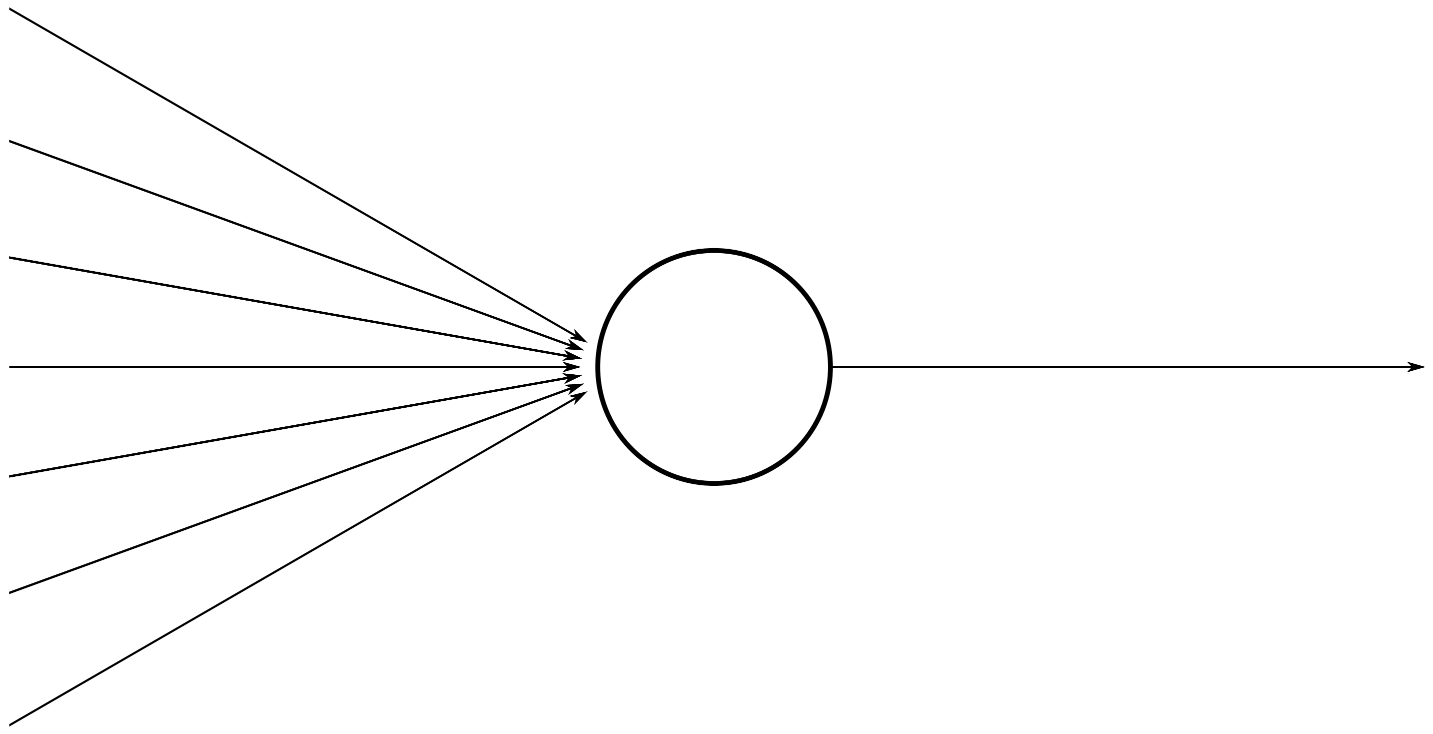

Figure 1.1: Graphical representation of an artificial neuron. Each arrow stick from right side represents a scalar (outputs from other neurons), and they collectively point to the left side of the neuron to represent an input vector. The right side arrow stick represents a neural output (scalar)

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{N-1} \\ a_N \end{bmatrix} \tag{1.1}$$

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_{N-1} & w_N \end{bmatrix} \tag{1.2}$$

where N is a number of elements of the input, $a_N = 1$, and $w_N = b$. In this article, I call those matrices input and weight vector as well (in this case the number of elements of both is the same because of the bias term). The output value for the neuron is then:

$$o = g(\mathbf{wa}) \tag{1.3}$$

The most typical activation function is ReLU:

$$g(x) = \max(0, x) \tag{1.4}$$

1.4 Layer

In ANN, a **layer** is an exclusive set of neurons. There are three types of layers: input, hidden, and output. The **input layer** contains multiple input neurons and one bias neuron, and is the very first layer that an

ANN starts with. The number of input neurons is exactly equal to the size of the model input x, and each input neuron outputs distinctive feature values, i.e., no input neurons output the same feature value. The **hidden layer** contains multiple regular neurons and one bias neuron. The **output layer** contains multiple regular neurons and returns the final ouput of the ANN $h(x)$, and this layer is the last layer of the neural network. A neuron in an output layer is called an **output neuron**. Typically, regular neurons in the same layer have the same activation function and it is usually safe to assume that is always the case. In this article, activation functions for neurons in the same layer are the same.

A layer, just like a neuron, returns an output, called **layer output** as a collection of output scalars from neurons in that layer. And such layer output can also be an input of a subsequent layer, called **layer input**. Such layer input provides its scalar values to form input vectors for neurons in that layer.

Now consider a hidden layer that contains N regular neurons which take the same input vector of M dimensions, $\mathbf{a}$, from the preceding layer output (N is presumed to be at least 1). If the neurons have an activation function g, then we can refine some of the neural variables above in the the following way:

- w_i = weight vector for the i-th neuron where $1 \leq i \leq N$
- w_{ij} = j-th component of w_i where $1 \leq j \leq M-1$
- $b_i = w_{iM}$ = bias for the i-th neuron
- $\mathbf{W} = N \times M$ weight matrix (or **kernel**) where $\mathbf{W}_{ij} = w_{ij}$
- $o_i = g([\mathbf{Wa}]_i)$ = output scalar from the i-th neuron $1 \leq i \leq N$
- $o_{N+1} = 1$ = output scalar from the bias neuron

1.5 Sequential Model

In a **sequential model** of ANN, each hidden layer is stacked next to each other without branching (an ANN model that is not sequential is a **functional model**), starting from an input layer and ending with an output layer. And if an ANN contains a deep stack of hidden layers, it is called a **deep neural network** (**DNN**). In the 1990s, an ANN with more than two hidden layers was considered deep. Nowadays, it is common to see ANNs with hundreds of layers, so the definition of "deep" is quite blurred.

In a sequential model, the signal flows from one layer to another sequentially, meaning a layer, except for the input layer, takes a layer output from the preceding layer as its input. Also, clearly there should be at least two layers in any ANN, the first layer being an input layer and the last layer being an output layer. The figure 1.2 is an example of a sequential neural network. In this article, only a sequential model case will be considered.

1.6 Dense Layer

We say that two neurons are connected if one neuron takes an input vector that contains an output scalar from the other neuron as an element of the input. If every neuron except for the bias neuron in a layer is connected to every neuron in the previous layer, then the layer is called a **fully connected layer** or a **dense layer**(an input layer can never be fully connected as it takes no input). If every layer except for the input layer is fully connected in an ANN, then the ANN is called a **fully connected neural network**. The sequential neural network in the figure 1.2 is a fully connected neural network. In this article, we will only consider a fully connected neural network model.

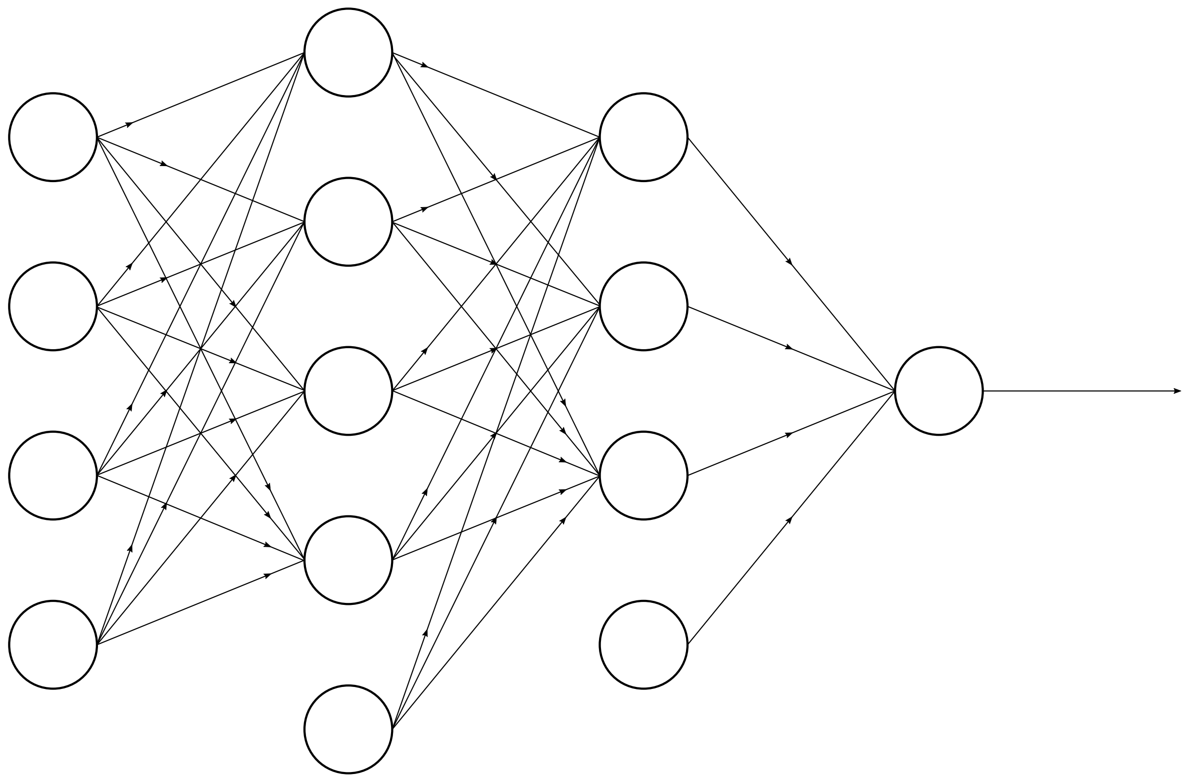

Figure 1.2: Graphical representation of a simple sequential model. The leftmost layer is the input layer and it has four neurons, the first three being input neurons and the last being a bias neuron. There are two hidden layers in this picture. The first hidden layer has five neurons and the second hidden layer has four neurons. The last layer is the output layer (only one neuron) and returns the final output scalar of the model. A line with an arrow on it between two neurons represents connection. Notice the very last neurons of each hidden layer: they do not form any connections with neurons in the previous layers as those are bias neurons.

1.7 Neural Network Variables

Consider a fully connected sequeltial ANN which contains L hidden layers. And suppose $1 \leq L$ for at least one hidden layer to be present. The model input x is fed into the network which then returns the model output $h(x)$. Building upon the previously defined variables, let us define the following variables:

- N_0 = number of input neurons in the input layer = model input size
- x_i = i-th component of x where $1 \leq i < N_0$
- $o_i^0 = x_i$ = output scalar from the i-th neuron in the input layer where $1 \leq i \leq N_0$
- $o_{N_0+1}^0 = 1$ = output scalar from the bias neuron in the input layer
- a^k = input vector for the k-th hidden layer where $1 \leq k \leq L$
- $\mathbf{a}_k$ = column matrix representation of a^k
- M_k = size or dimension of a^k
- a_i^k = i-th component of a^k where $1 \leq i \leq M_k$

- N_k = number of regular neurons in the k-th hidden layer
- w_i^k = weight vector for the i-th neuron in the k-th hidden layer where $1 \leq i \leq N_k$
- w_{ij}^k = j-th component of w_i^k where $1 \leq j \leq M_k - 1$
- $b_i^k = w_{iM_k}^k$ = bias for the i-th neuron in the k-th layer
- $\mathbf{W}_k = N_k \times M_k$ weight matrix of the k-th hidden layer where $[\mathbf{W}_k]_{ij} = w_{ij}^k$
- $\mathbf{c}_k = \mathbf{W}_k \mathbf{a}_k$ = column matrix representation of weighted sums of a_i^k elements with w_i^k for indices i where $[\mathbf{c}_k]_i = c_i^k$
- g^k = activation function for the k-th hidden layer
- $o_i^k = g^k(c_i^k)$ = output scalar from the i-th neuron in the k-th hidden layer where $1 \leq i \leq N_k$
- $o_{N_k+1}^k = 1$ = output scalar from the bias neuron in the k-th hidden layer
- a^{L+1} = input vector for the output layer
- $\mathbf{a}_{L+1}$ = column matrix representation of a^{L+1}
- M_{L+1} = size or dimension of a^{L+1}
- a_i^{L+1} = i-th component of a^{L+1} where $1 \leq i \leq M_{L+1}$
- N_{L+1} = number of neurons in the output layer = size of model output $h(x)$
- w_i^{L+1} = weight vector for the i-th neuron in the output layer where $1 \leq i \leq N_{L+1}$
- w_{ij}^{L+1} = j-th component of w_i^{L+1} where $1 \leq j \leq M_{L+1} - 1$
- $b_i^{L+1} = w_{iM_{L+1}}^{L+1}$ = bias for the i-th neuron in the output layer
- $\mathbf{W}_{L+1} = N_{L+1} \times M_{L+1}$ weight matrix of the output layer where $[\mathbf{W}_{L+1}]_{ij} = w_{ij}^{L+1}$
- $\mathbf{c}_{L+1} = \mathbf{W}_{L+1} \mathbf{a}_{L+1}$ = column matrix representation of weighted sums of a_i^{L+1} elements with w_i^{L+1} for indices i where $[\mathbf{c}_{L+1}]_i = c_i^{L+1}$
- $\mathbf{c}^{L+1} = (\mathbf{c}_{L+1})^T$ = transpose of $\mathbf{c}_{L+1}$
- σ_i = activation function for the i-th output neuron
- $h_i = \sigma_i(\mathbf{c}_{L+1})$ = i-th element of model output $h(x)$ where $1 \leq i \leq N_{L+1}$

As there should be at least one feature value in x and at least one regular neuron in a non-input layer, we know that

$$1 \leq N_k \tag{1.5}$$

for $0 \leq k \leq L + 1$. Since the considered model is sequential and fully connected, the entire collection of output scalars from a non-output layer forms an input vector for the subsequent layer. In other words,

$$o_i^{k-1} = a_i^k \tag{1.6}$$

for $1 \leq k \leq L+1$. Also it should be apparent that

$$N_{k-1} + 1 = M_k \tag{1.7}$$

since N_{k-1} regular neurons (or input neurons) and one bias neuron all return output scalars that form the input vector a^k.

The model h has model parameters in the output layer as its function parameters because its element is returned from an activation function that depends on $\mathbf{c}_{L+1}$ which is obtained by $\mathbf{W}_{L+1}\mathbf{a}_{L+1}$. h also has model parameters in the last hidden layer as its function parameters because it depends on $\mathbf{a}_{L+1}$ that is obtained from an activation function that depends on $\mathbf{W}_L\mathbf{a}_L$. You can use this induction to conclude that h has model parameters in every layer as its function parameters, and $\mathbf{a}^k$ depends on model parameters in layers that come before the k-th hidden layer (or the output layer if $k = L+1$) if $1 < k$.

If you inspect the definitions of activation function for a hidden and the output layer, you will notice that for the hidden layer, g depends on only one element of $\mathbf{c}$ whereas for the output layer, σ can depend on $\mathbf{c}$. One example of σ that depends on $\mathbf{c}$ is softmax function:

$$\sigma_i(\mathbf{c}) = \frac{e^{c_i}}{\sum\limits_{j=1}^{N} e^{c_j}} \tag{1.8}$$

Even though nothing prevents activation function to depend on any variables, we will stick with the above case.

2 Backpropagation

With a fully connected sequential neural network to consider, we can now discuss what backpropagation is in detail. For our model h, weights w^k_{ij} are model parameters we want to optimize by training. For validation, L, N_k, g_k (with $1 \leq k \leq L$), and σ are hyperparameters to be tuned.

2.1 Cost Function

Let X be a collection of m input data, i.e., $X = \{x^1, x^2, \ldots, x^{m-1}, x^m\}$, and Y be a collection of targets for X, i.e., $Y = \{y^1, y^2, \ldots, y^{m-1}, y^m\}$ (y^i is the target for x^i). The sizes of x and y are N_0 and N_{L+1} respectively, and clearly X and Y have the same cardinality m. Suppose the model is trained on those datasets. Then a cost function J of the model h is a function of X and Y ($J(X, Y; h)$). One example of J is cross entropy:

$$J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{N_{L+1}} y^i_j \ln(h^i_j) \tag{2.1}$$

Where $h^i = h(x^i)$. Now let W, G, N be collections of w^k_{ij}, g_k, N_k of the model respectively. Since those are function parameters of h, including L, they are function parameters of J as well, i.e., ($J(X, Y; h) = J(X, Y; W, G, N, L)$. Since backpropagation is a gradient descent that optimizes the model parameters, we can let W be a variable of J and let X and Y be the function parameters instead, i.e., $J(W; X, Y, G, N, L) = J(W)$. The training is simply finding W that minimizes J (you can add an additional term to regularize the cost function, but in this article no regularization is considered as it is not relevant to the discussion of backpropagation).

2.2 Gradient Descent

With the cost function J, we can update W iteratively by the following assignment operation until J is miminized (gradient descent method):

$$w^k_{ij} \leftarrow w^k_{ij} - \eta\frac{\partial J}{\partial w^k_{ij}} \tag{2.2}$$

Where η is called **learning rate**. All of w^k_{ij} must be iterated before the next iteration. For each iteration, you can also update the learning rate, but in this article the learning rate is kept constant.

2.3 Partial Derivative of J

Let us evaluate the partial derivative term in 2.2. First we know that J contains $m \times N_{L+1}$ many h terms that depend on weights. Thus:

$$\frac{\partial J}{\partial w^k_{ij}} = \sum_{p=1}^{m}\sum_{q=1}^{N_{L+1}} \frac{\partial J}{\partial h^p_q}\frac{\partial h^p_q}{\partial w^k_{ij}} \tag{2.3}$$

$\partial J/\partial h^p_q$ term expression is specific to the choice of the cost function, so we will leave it as it is. For $\partial h^p_q/\partial w^k_{ij}$, we can further evaluate. Since 2.3 deals with m many x, h term has a superscript p for x^p. For simplicity, we will evaluate the second derivative term with a general x so we do not have to be concerned with the superscript p, i.e., $\partial h_q/\partial w_{ij}^k$ notation will be used. Now let us evaluate $\partial h_q/\partial w_{ij}^k$:

$$\begin{aligned}\frac{\partial h_q}{\partial w_{ij}^k} &= \frac{\partial \sigma_q}{\partial w_{ij}^k}\\ &= \sum_{r=1}^{N_{L+1}} \frac{\partial \sigma_q}{\partial c_r^{L+1}} \frac{\partial c_r^{L+1}}{\partial w_{ij}^k}\\ &= \frac{\partial \sigma_q}{\partial \mathbf{c}^{L+1}} \frac{\partial \mathbf{c}_{L+1}}{\partial w_{ij}^k} \end{aligned} \tag{2.4}$$

where

$$\frac{\partial \sigma_q}{\partial \mathbf{c}^{L+1}} = \begin{bmatrix} \frac{\partial \sigma_q}{\partial c_1^{L+1}} & \frac{\partial \sigma_q}{\partial c_2^{L+1}} & \cdots & \frac{\partial \sigma_q}{\partial c_{N^{L+1}-1}^{L+1}} & \frac{\partial \sigma_q}{\partial c_{N^{L+1}}^{L+1}} \end{bmatrix} \tag{2.5}$$

And we know that

$$c_r^l = \sum_{s=1}^{N_{l-1}+1} w_{rs}^l a_s^l \tag{2.6}$$

where $1 \leq l \leq L+1$ and the number of summands is determined by 1.7. The partial derivative of 2.6 with respect to w_{ij}^k is then:

$$\frac{\partial c_r^l}{\partial w_{ij}^k} = \sum_{s=1}^{N_{l-1}+1} (\frac{\partial w_{rs}^l}{\partial w_{ij}^k} a_s^l + w_{rs}^l \frac{\partial a_s^l}{\partial w_{ij}^k}) \tag{2.7}$$

where we know a^l depends on weights only in layers that come before the l-th hidden layer (or output layer if $l = L+1$) if $1 < l$. Thus, 2.7 is simply 0 if $l < k$.

If $l = k$, then:

$$\begin{aligned}\frac{\partial c_r^l}{\partial w_{ij}^k} &= \sum_{s=1}^{N_{l-1}+1} \delta_{ri}\delta_{sj} a_s^k\\ &= \delta_{ri} a_j^k \end{aligned} \tag{2.8}$$

where Kronecker delta relation is used:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

In other words,

$$\frac{\partial \mathbf{c}_k}{\partial w_{ij}^k} = a_j^k \begin{bmatrix} \delta_{1i} \\ \delta_{2i} \\ \vdots \\ \delta_{N_k-1,i} \\ \delta_{N_k i} \end{bmatrix} \tag{2.10}$$

If $k < l$, then:

$$\begin{aligned}
\frac{\partial c_r^l}{\partial w_{ij}^k} &= \sum_{s=1}^{N_{l-1}+1} w_{rs}^l \frac{\partial a_s^l}{\partial w_{ij}^k} && \left(\text{from 2.7 with } \frac{\partial w_{rs}^l}{\partial w_{ij}^k} = 0\right) \\
&= \sum_{s=1}^{N_{l-1}} w_{rs}^l \frac{\partial o_s^{l-1}}{\partial w_{ij}^k} && (a_s^l = o_s^{l-1} \text{ and } o_{N_{l-1}+1}^{l-1} = 1) \\
&= \sum_{s=1}^{N_{l-1}} w_{rs}^l \frac{dg^{l-1}}{dc_s^{l-1}} \frac{\partial c_s^{l-1}}{\partial w_{ij}^k} && (o_s^{l-1} = g^{l-1}(c_s^{l-1}) \text{ with chain rule}) \qquad (2.11)
\end{aligned}$$

If we define N_l by N_{l-1} matrix $\mathbf{S}_{l-1}$ such that:

$$[\mathbf{S}_{l-1}]_{rs} = w_{rs}^l \frac{dg^{l-1}}{dc_s^{l-1}} \tag{2.12}$$

then 2.11 in matrix form is

$$\frac{\partial \mathbf{c}_l}{\partial w_{ij}^k} = \mathbf{S}_{l-1} \frac{\partial \mathbf{c}_{l-1}}{\partial w_{ij}^k} \tag{2.13}$$

You can see that the second term on the right side of 2.13 is the same as the left side term but the subscript of **c** is one less. We can use this recursive definition to expand 2.13 until the subscript of **c** on the right side reaches k:

$$\frac{\partial \mathbf{c}_l}{\partial w_{ij}^k} = \mathbf{S}_{l-1}\mathbf{S}_{l-2}\ldots\mathbf{S}_{k+1}\mathbf{S}_k \frac{\partial \mathbf{c}_k}{\partial w_{ij}^k} \tag{2.14}$$

More succinctly,

$$\frac{\partial \mathbf{c}_l}{\partial w_{ij}^k} = \mathbf{T}_k^{l-1} \frac{\partial \mathbf{c}_k}{\partial w_{ij}^k} \tag{2.15}$$

where

$$\mathbf{T}_j^i = \begin{cases} 1, & \text{if } i < j \\ \mathbf{S}_i\mathbf{S}_{i-1}\ldots\mathbf{S}_{j+1}\mathbf{S}_j, & \text{if } j \le i \end{cases} \tag{2.16}$$

Then 2.4 can be re-written as:

$$\frac{\partial h_q}{\partial w_{ij}^k} = \frac{\partial \sigma_q}{\partial \mathbf{c}^{L+1}} \mathbf{T}_k^L \frac{\partial \mathbf{c}_k}{\partial w_{ij}^k} \tag{2.17}$$

2.17 is evaluated with x. 2.17 can simply be evaluated with x^p for 2.3. This will complete the partial derivative of the cost function.

References

[1] Aurelien Geron. *Hands-on Machine learning with Scikit-learn, Keras, and Tensorflow.* 2nd ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reily Media, 2019. ISBN: 9781492032649.

Index