Formalize Forward and Backpropagation

As with any supervised machine learning algorithm, we need to train it from data. We will do so using (stochastic) gradient descent, which requires the computation of the gradients of the loss with respect to the model parameters.

We can compute the loss with the forward propagation and the gradient of the loss with respect to the weights with backward propagation. The bulk of the actual training, where we modify the model to perform better on training data, happens in the backpropagation. Listed below are the steps to training the neural net.

☆ Key Points

We train the model using gradient descent, so we update $\mathbf{W}_l$ with $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$.

To calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$, we use the chain rule: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_l} \odot \frac{\partial \mathbf{a}_l}{\partial \mathbf{W}_l}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_{l-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_l} \odot \frac{\partial \mathbf{a}_l}{\partial \mathbf{a}_{l-1}}$.

We use back propagation to update the weights during (stochastic) gradient descent, starting from the last layer's to the first's.

Step 0: Propagate Input Forward

Forward propagation is actually really simple to implement. We pass a data point $\mathbf{x}$ forward through the neural network and get an output $h(\mathbf{x})$. Recall that we have defined a neural network model to be:

$$h(\mathbf{x}) = \sigma_L\left(\mathbf{W}_L \sigma_{L-1}\left(\mathbf{W}_{L-1} \cdots \sigma_2\left(\mathbf{W}_2 \sigma_1\left(\mathbf{W}_1 \mathbf{x}\right)\right) \cdots\right)\right)$$

Let's unpack the above a bit on a sample example with 3 hidden layers (thus a total of 4 tunable weight matrices), defining helper vector $\mathbf{z}$. Assume $\mathbf{b}$ has been absorbed by $\mathbf{W}$.

$$\begin{aligned}
\mathbf{z}_0 &= \mathbf{x} \\
\mathbf{z}_1 &= \sigma_1(\mathbf{W}_1 \mathbf{z}_0) \\
\mathbf{z}_2 &= \sigma_2(\mathbf{W}_2 \mathbf{z}_1) \\
\mathbf{z}_3 &= \sigma_3(\mathbf{W}_3 \mathbf{z}_2) \\
\mathbf{z}_4 &= \sigma_4(\mathbf{W}_4 \mathbf{z}_3)
\end{aligned}$$

We will also label the intermediate product, $\mathbf{a}$, that will become useful in the next step:

$$\begin{aligned}
\mathbf{a}_1 &= \mathbf{W}_1 \mathbf{z}_0 \\
\mathbf{a}_2 &= \mathbf{W}_2 \mathbf{z}_1 \\
\mathbf{a}_3 &= \mathbf{W}_3 \mathbf{z}_2 \\
\mathbf{a}_4 &= \mathbf{W}_4 \mathbf{z}_3
\end{aligned}$$

Thus, our neural net is now:

$$\begin{aligned}
\mathbf{z}_1 &= \sigma_1(\mathbf{a}_1) \\
\mathbf{z}_2 &= \sigma_2(\mathbf{a}_2) \\
\mathbf{z}_3 &= \sigma_3(\mathbf{a}_3) \\
\mathbf{z}_4 &= \sigma_4(\mathbf{a}_4)
\end{aligned}$$

Ultimately, each layer $\mathbf{a}_l$ is passed through the transition function σ_l to get the corresponding $\mathbf{z}_l$, which is in turn multiplied by the weights to get the next layer $\mathbf{a}_{l+1}$. Typically a forward pass is done on a batch of input data points, $\mathbf{X} \in \mathbb{R}^{n \times d}$. To write the algorithm for a batch of points, we use the notation $\mathbf{A}_l$ and $\mathbf{Z}_l$ and assume that the weight matrices $\mathbf{W}_l$ are of shapes: $\mathbf{W}_1 \in \mathbb{R}^{d \times d_1}, \mathbf{W}_2 \in \mathbb{R}^{d_1 \times d_2}, \ldots, \mathbf{W}_L \in \mathbb{R}^{d_{L-1} \times d_L}$. Consequently, the operation $\mathbf{a}_l = \mathbf{W}_l \mathbf{z}_{l-1}$ looks like $\mathbf{A}_l = \mathbf{Z}_{l-1} \mathbf{W}_l$. The formal algorithm for forward propagation is:

procedure FORWARD-PASS($\mathbf{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}, \mathbf{X} \in \mathbb{R}^{n \times d}$)
  $\mathbf{Z}_0 \leftarrow \mathbf{X}$
  for $l = 1 : L$ **do**
    $\mathbf{A}_l \leftarrow \mathbf{Z}_{l-1} \mathbf{W}_l$
    $\mathbf{Z}_l \leftarrow \sigma_l(\mathbf{A}_l)$
  end for
  return $\mathbf{Z}_L$
end procedure

Often, you can use the same transition function for each layer — **except the last one σ_L**. The last layer determines the type of output of the neural network and is chosen to fit the use case. For example, we can use the softmax function for a classification problem or the identity function (i.e., just outputting the input) for a regression problem.

Step 1: Evaluate Loss

We propagate an input $\mathbf{x}$ forward to evaluate the output of the neural network $h(\mathbf{x}) = \mathbf{z}_4$ and calculate the loss $\mathcal{L}$. We've seen several different loss functions before. Here we will use Mean Squared Error to demonstrate:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \left(h(\mathbf{x}_i) - y_i\right)^2$$

Step 2: Update Weights

The ultimate goal is to reduce the loss function $\mathcal{L}$ using gradient descent, which updates all weights of the network. The backpropagation algorithm updates weights, starting from the last layer and proceeding backward. Intuitively, the algorithm "propagates the loss backwards". Recall that to update the weights using gradient descent, you would do $\mathbf{W}_l \leftarrow \mathbf{W}_l - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$ for each tunable weight matrix $\mathbf{W}_l$.

Step 2a: Update last layer's weights using chain rule

We want to calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$ for all the weights in all the layers. $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_L}$ is the easiest to calculate of all the gradients, which get tougher as l decreases because of multiple nested transition functions. To combat this problem, we will use the chain rule to decompose the gradient. Consider the 3-hidden-layer neural network example from above.

First of all, since $\mathcal{L}$ depends on $\mathbf{z}_4 = \sigma_4(\mathbf{a}_4)$ and $\mathbf{a}_4 = \mathbf{W}_4 \mathbf{z}_3$. $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_4} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_4} \frac{\partial \mathbf{a}_4}{\partial \mathbf{W}_4}$. Each factor is easier to calculate now.

- $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_4} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_4} \odot \frac{\partial \mathbf{z}_4}{\partial \mathbf{a}_4} = 2(\mathbf{z}_4 - y) \odot \sigma_4'(\mathbf{a}_4)$ where σ_4' is the derivative of σ_4 and $\odot$ is element-wise multiplication
- $\frac{\partial \mathbf{a}_4}{\partial \mathbf{W}_4} = \mathbf{z}_3^\top$

Therefore, $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_4} = \left[2(\mathbf{z}_4 - y) \odot \sigma_4'(\mathbf{a}_4)\right] \mathbf{z}_3^\top$. We can refer to $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_4}$ as the "link" for ultimately calculating the gradient with respect to $\mathbf{W}_4$.

Step 2b: Prepare the "link" for the preceding layer and calculate its weights' gradient using chain rule

Before we calculate the gradient to update the next layer, $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_3}$, we need the crucial "link" $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_3}$. Recall that, like before, we can break the update down:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_3} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_3} \frac{\partial \mathbf{a}_3}{\partial \mathbf{W}_3}$$

This may seem daunting, but in fact, this calculation takes almost no time. $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_3}$ is easily computable from the gradient of the previous layer.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_3} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_4} \odot \frac{\partial \mathbf{a}_4}{\partial \mathbf{a}_3}$$

We reuse $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_4}$ calculated in step 2a. Further notice that $\frac{\partial \mathbf{a}_4}{\partial \mathbf{a}_3} = \frac{\partial \mathbf{a}_4}{\partial \mathbf{z}_3} \frac{\partial \mathbf{z}_3}{\partial \mathbf{a}_3}$. Since $\mathbf{a}_4 = \mathbf{W}_4 \mathbf{z}_3$ and $\mathbf{z}_3 = \sigma_3(\mathbf{a}_3)$, $\frac{\partial \mathbf{a}_4}{\partial \mathbf{a}_3} = \mathbf{W}_4 \sigma_3'(\mathbf{a}_3)$, where σ_3' is the derivative of σ_3. Substituting all the values in, we get:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{a}_3} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}_4} \odot \mathbf{W}_4 \sigma_3'(\mathbf{a}_3) \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}_3} &= \left[\frac{\partial \mathcal{L}}{\partial \mathbf{a}_4} \odot \mathbf{W}_4 \sigma_3'(\mathbf{a}_3)\right] \mathbf{z}_2^\top
\end{aligned}$$

A nifty feature of many popular transition functions is that their derivatives are very simple. Additionally, if you use the same transition function between each layer, then you could also just store what the derivative of the transition function is.

Step 2c: Repeat step 2b for remaining layers of the network

As we have seen in step 2b, the "links" $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_l}$ are useful for calculating the gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_j}$ for $j \leq l$. For any layer, $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_l} \frac{\partial \mathbf{a}_l}{\partial \mathbf{W}_l}$, and for all but the last layer, $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_{l-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_l} \odot \frac{\partial \mathbf{a}_l}{\partial \mathbf{a}_{l-1}}$. The links can thus be sequentially calculated. These two equations thus define the backpropagation algorithm. In our specific case, we also have $\frac{\partial \mathbf{a}_l}{\partial \mathbf{W}_l} = \mathbf{z}_{l-1}^\top$ and $\frac{\partial \mathbf{a}_l}{\partial \mathbf{a}_{l-1}} = \mathbf{W}_l \sigma_{l-1}'(\mathbf{a}_{l-1})$. Finally, we update $\mathbf{W}_l = \mathbf{W}_l - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$.

Pseudocode

We can wrap the steps for backward propagation in the pseudocode below. Note that we define the link as $\delta_l = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_l}$.

procedure BACKWARDPROP($\mathbf{W}, \{\mathbf{A}_1, \ldots, \mathbf{A}_L\}, \{\mathbf{Z}_0, \ldots, \mathbf{Z}_L\}, \mathbf{y} \in \mathbb{R}^n$)
  $\delta_L \leftarrow \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_L} \odot \sigma_L'(\mathbf{A}_L)$
  for $l = (L-1) : 1$ **do**
    $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{l+1}} \leftarrow \mathbf{Z}_l^\top \delta_{l+1}$
    $\delta_l \leftarrow \left(\delta_{l+1} \mathbf{W}_l^\top\right) \odot \sigma_l'(\mathbf{A}_l)$
  end for
  $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} \leftarrow \mathbf{Z}_0^\top \delta_1$
  return $\left\{\frac{\partial L}{\partial \mathbf{W}_1}, \ldots, \frac{\partial L}{\partial \mathbf{W}_L}\right\}$
end procedure