CS 189 Introduction to Machine Learning
Spring 2022 Marvin Zhang DIS10

1 Forward and Backward Pass Walkthrough

Let's perform the forward and backward passes for a simple neural network with one hidden layer:

$$\mathbf{h} = s(\mathbf{Vx}) \qquad \mathbf{z} = s(\mathbf{Wh}) \qquad \ell = \frac{1}{2}\|\mathbf{y}-\mathbf{z}\|_2^2$$
$$\mathbf{a} = \mathbf{Vx} \qquad \mathbf{b} = \mathbf{Wh}$$

Here, $\mathbf{x} \in \mathbb{R}^{d^{(0)}}$ is the input vector, $\mathbf{h} \in \mathbb{R}^{d^{(1)}}$ contains the hidden layer activations after being passed through the sigmoid $s(\cdot)$ nonlinearity, $\mathbf{z} \in \mathbb{R}^{d^{(2)}}$ is the output vector, $\mathbf{y} \in \mathbb{R}^{d^{(2)}}$ is the label associated with $\mathbf{x}$, and ℓ is the squared error loss function. $\mathbf{a} \in \mathbb{R}^{d^{(1)}}$ and $\mathbf{b} \in \mathbb{R}^{d^{(2)}}$ are the pre-activation values that may be useful notation for some parts.

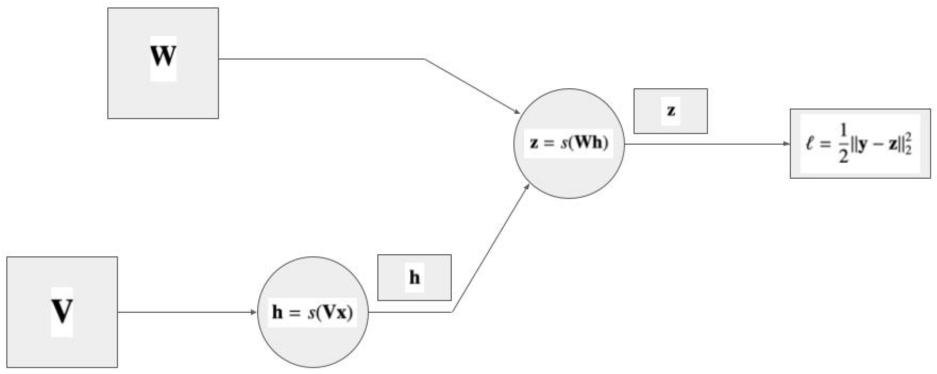
The multivariate chain rule for a composed function such as $f(g(h(\mathbf{x})))$ in one of the following forms may be useful in some parts:

$$\nabla_{\mathbf{x}} f(g(h(\mathbf{x}))) = (\frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial \mathbf{x}})^\top = (\frac{\partial h}{\partial \mathbf{x}})^\top \cdot (\frac{\partial g}{\partial h})^\top \cdot (\frac{\partial f}{\partial g})^\top = \nabla_{\mathbf{x}} h \cdot \nabla_h g \cdot \nabla_g f \tag{1}$$

$$\nabla_{\mathbf{x}} f(g(\mathbf{x})) = \sum_{j=1}^{k} (\frac{\partial f}{\partial g_j} \cdot \frac{\partial g_j}{\partial \mathbf{x}})^\top = \sum_{j=1}^{k} \nabla_{\mathbf{x}} g_j \cdot \nabla_{g_j} f \qquad \text{where } g(\mathbf{x}) \in \mathbb{R}^k \tag{2}$$

(a) **Fill in the diagram below to represent the forward pass graph of the neural network, and write out the forward pass output and loss given a point x.**

Solution:



(b) **What are the parameters we need to learn? What gradients do we need to perform a gradient descent update, and what are their shapes?**

Solution: $\mathbf{V}$ and $\mathbf{W}$ are the weight matrices we need to learn. Therefore, we need $\nabla_{\mathbf{V}}\ell, \nabla_{\mathbf{W}}\ell$. $\nabla_{\mathbf{V}}\ell$ is the same shape as $\mathbf{V}$, which is $d^{(1)} \times d^{(0)}$. $\nabla_{\mathbf{W}}\ell$ is the same shape as $\mathbf{W}$, which is $d^{(2)} \times d^{(1)}$.

(c) Let's start computing gradients from the end of the computation graph. **Compute $\nabla_{\mathbf{z}}\ell$ and write down its shape.**

Solution:

$$\nabla_{\mathbf{z}}\ell = \mathbf{z} - \mathbf{y} \in \mathbb{R}^{d^{(2)}}$$

(d) **Now, compute $\nabla_{\mathbf{W}}\ell$ and write down its shape.** This is most easily approached by computing gradients with respect to each *row* $\mathbf{w}_j$ of $\mathbf{W}$ and then stacking them to form the full $\nabla_{\mathbf{W}}\ell$ gradient:

$$\nabla_{\mathbf{W}}\ell = \begin{bmatrix} (\nabla_{\mathbf{w}_1}\ell)^\top \\ \vdots \\ (\nabla_{\mathbf{w}_{d^{(2)}}}\ell)^\top \end{bmatrix}$$

Consider the chain rule provided above to break the gradient down into more manageable pieces.

Solution: It's often easier to work with derivatives, and then take the transpose at the end to convert to a gradient. Let's expand out the derivative we want to calculate:

$$\frac{\partial \ell}{\partial \mathbf{w}_j} = \frac{\partial \ell}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_j} \cdot \frac{\partial b_j}{\partial \mathbf{w}_j}$$

This chain rule decomposition breaks the gradient down into three parts. The first term is a derivative that we just computed in the previous part. The second term is the derivative of the activation function. The third term is the derivative of the linear transformation $\mathbf{b} = \mathbf{W}\mathbf{h} = \begin{bmatrix} \mathbf{w}_1^\top \mathbf{h} & \dots & \mathbf{w}_{d^{(2)}}^\top \mathbf{h} \end{bmatrix}^\top$. Since we have the first term, we just need to calculate the last two.

Observe that we have isolated the derivative to only the j-th component of $\mathbf{z}$ and $\mathbf{b}$, since the j-th row of $\mathbf{W}$ only contributes to those entries! Draw out the matrix multiplication in terms of rows of $\mathbf{W}$ to convince yourself of this fact.

For $\frac{\partial z_j}{\partial b_j}$, first notice that $z_j = s(b_j)$, so we can plug in the activation function derivative, which is for sigmoid in this case: $\frac{\partial z_j}{\partial b_j} = s'(b_j) = s(b_j)(1 - s(b_j)) = z_j(1 - z_j)$.

For $\frac{\partial b_j}{\partial \mathbf{w}_j}$, notice that $b_j = w_j^\top \mathbf{h}$. This is the derivative of a dot product which we have seen, so $\frac{\partial b_j}{\partial \mathbf{w}_j} = \mathbf{h}^\top$.

Writing this out as a gradient:

$$\nabla_{\mathbf{w}_j}\ell = (\frac{\partial \ell}{\partial z_j} \cdot s(b_j)(1 - s(b_j)) \cdot \mathbf{h}^\top)^\top = \mathbf{h} \cdot s(b_j)(1 - s(b_j)) \cdot \nabla_{z_j}\ell$$

Stacking as rows in a matrix, we can construct the full gradient with respect to $\mathbf{W}$:

$$\nabla_{\mathbf{W}}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial z_1} \cdot z_1(1-z_1) \cdot \mathbf{h}^\top \\ \vdots \\ \frac{\partial \ell}{\partial z_{d^{(2)}}} \cdot z_{d^{(2)}}(1-z_{d^{(2)}}) \cdot \mathbf{h}^\top \end{bmatrix} = \Omega \cdot \nabla_{\mathbf{z}}\ell \cdot \mathbf{h}^\top$$

Ω is a diagonal matrix with $z_j(1-z_j)$ as its diagonal entries. The shape of this gradient is $d^{(2)} \times d^{(1)}$, which is the same shape as $\mathbf{W}$, as expected.

(e) To compute the next gradient with respect to $\mathbf{V}$, we'll first need an intermediate gradient $\nabla_{\mathbf{h}}\ell$. **Compute $\nabla_{\mathbf{h}}\ell$ and write down its shape.** Again, consider the chain rule provided above to break the gradient down into more manageable pieces.

Solution:

$$\underbrace{\frac{\partial \ell}{\partial \mathbf{h}}}_{1\times d^{(1)}} = \underbrace{\frac{\partial \ell}{\partial \mathbf{z}}}_{1\times d^{(2)}} \cdot \underbrace{\frac{\partial \mathbf{z}}{\partial \mathbf{b}}}_{d^{(2)}\times d^{(2)}} \cdot \underbrace{\frac{\partial \mathbf{b}}{\partial \mathbf{h}}}_{d^{(2)}\times d^{(1)}}$$

Again, the first term can be filled in from a previous part, so we're left to calculate the last two terms.

$\frac{\partial \mathbf{z}}{\partial \mathbf{b}}$ is a Jacobian, which contains partial derivatives of each component of $\mathbf{z}$ with respect to each component of $\mathbf{b}$. Since each component of $\mathbf{z}$ is just the activation applied to the same component of $\mathbf{b}$, this matrix simplifies to a diagonal matrix with sigmoid derivative down the diagonal. We already defined such a matrix called Ω in the previous part.

For the last term, $\frac{\partial \mathbf{b}}{\partial \mathbf{h}} = \frac{\partial (\mathbf{Wh})}{\partial \mathbf{h}} = \mathbf{W}$ can be shown by taking the derivative one row at a time.

Putting it together and taking the transpose, we get the gradient:

$$\nabla_{\mathbf{h}}\ell = (\frac{\partial \ell}{\partial \mathbf{h}})^\top = \mathbf{W}^\top \cdot \Omega \cdot \nabla_{\mathbf{z}}\ell$$

The shape of this gradient is $d^{(1)} \times 1$, the same shape as $\mathbf{h}$, as expected.

(f) Now, we're ready to compute the gradient with respect to $\mathbf{V}$. **Compute $\nabla_{\mathbf{V}}\ell$ and write down its shape.** You'll want to do this in a similar way to calculating $\nabla_{\mathbf{W}}\ell$: **compute gradients with respect to each *row* $\mathbf{v}_j$ of $\mathbf{V}$ and then stacking them to form the full $\nabla_{\mathbf{V}}\ell$ gradient.**

Remember that you have access to every gradient you've computed already in previous parts.

Solution:

$$\frac{\partial \ell}{\partial \mathbf{v}_j} = \frac{\partial \ell}{\partial h_j} \cdot \frac{\partial h_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial \mathbf{v}_j}$$

We have the first term from the gradient $\nabla_{\mathbf{h}}\ell$ that was propagated backward from the next layer.

Recall that $\mathbf{a} = \mathbf{V}\mathbf{x}$ and $h_j = s(a_j)$, and notice that the remaining two terms are almost identical to the ones we calculated when considering rows of $\mathbf{W}$, so we can just plug in the result we found from that part.

$$\nabla_{\mathbf{v}_j}\ell = (\frac{\partial \ell}{\partial h_j} \cdot h_j(1-h_j) \cdot \mathbf{x}^\top)^\top = \mathbf{x} \cdot h_j(1-h_j) \cdot \nabla_{h_j}\ell$$
$$\nabla_{\mathbf{V}}\ell = \Lambda \cdot \nabla_{\mathbf{h}}\ell \cdot \mathbf{x}^\top$$

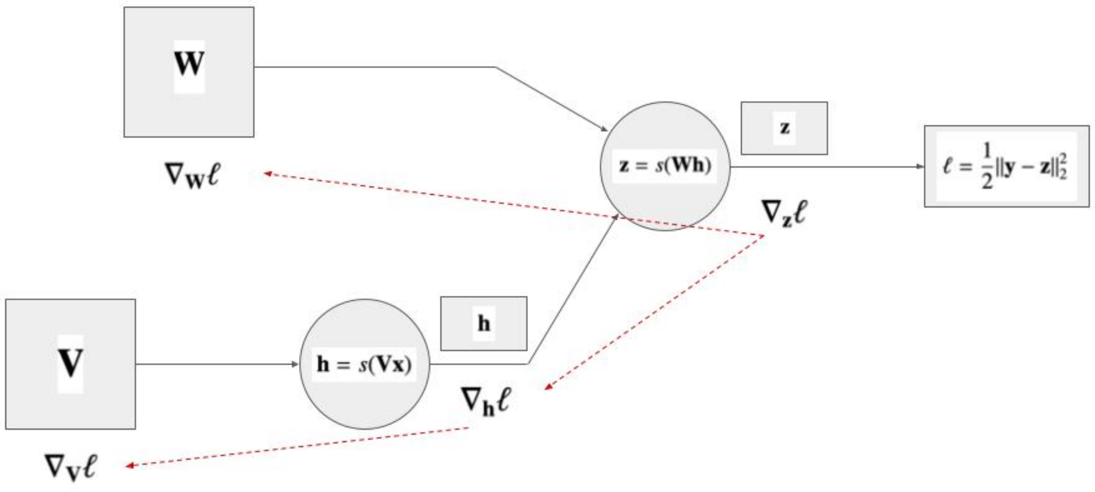
Λ is a diagonal matrix similar to Ω, except it has $h_j(1-h_j)$ along its diagonal instead. The shape of this gradient is $d^{(1)} \times d^{(0)}$ as expected.

(g) **Go back through your gradient computations and indicate which gradients can be calculated in the forward pass and which gradients need to be filled in by backpropagation.** What is the main benefit of backpropagation?

Solution: Solutions above explain this already.

Backpropagation allows us to split up the gradients we need (i.e. the activation function can implement its own gradient, and the linear transformation can keep track of its own gradient, etc.) and reuse work from later layers to use in computing gradients for earlier layers. As seen in the similarities between the $\mathbf{V}$ and $\mathbf{W}$ gradients, the gradient calculations can be modularized and be extended to any number of layers.

The following figure shows the gradients that are passed back:


$\mathbf{W}$
$\nabla_{\mathbf{W}}\ell$
$\mathbf{z} = s(\mathbf{W}\mathbf{h})$
$\mathbf{z}$
$\nabla_{\mathbf{z}}\ell$
$\ell = \frac{1}{2}\|\mathbf{y}-\mathbf{z}\|_2^2$
$\mathbf{V}$
$\mathbf{h} = s(\mathbf{V}\mathbf{x})$
$\mathbf{h}$
$\nabla_{\mathbf{h}}\ell$
$\nabla_{\mathbf{V}}\ell$

(h) Finally, **write down the gradient descent update given a single point x in terms of $\nabla_{\mathbf{V}}\ell, \nabla_{\mathbf{W}}\ell$ using a learning rate α.**

Solution:

$$\mathbf{V} \leftarrow \mathbf{V} - \alpha \nabla_{\mathbf{V}} \ell$$
$$\mathbf{W} \leftarrow \mathbf{W} - \alpha \nabla_{\mathbf{W}} \ell$$

2 Initialization of Weights for Backpropagation (Optional)

Assume a fully-connected 1-hidden-layer network. Denote the dimensionalities of the input, hidden, and output layers as $d^{(0)}$, $d^{(1)}$, and $d^{(2)}$. That is, the input (which we will denote with a superscript (0)) has dimensions $x_1^{(0)}, \dots, x_{d^{(0)}}^{(0)}$. Let g denote the activation function applied at each layer. As defined in lecture, let $S_j^{(l)} = \sum_{i=1}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$ be the weighted input to node j in layer l, and let $\delta_j^{(l)} = \frac{\partial \ell}{\partial S_j^{(l)}}$ be the partial derivative of the final loss ℓ with respect to $S_j^{(l)}$.

Recall that backpropagation is simply an efficient method to compute the gradient of the loss function so we can use it with gradient descent. These methods require the parameters to be initialized to some value. In logistic regression we were able to initialize all weights as 0.

(a) Imagine that we initialize the values of our weights to be some constant w. After performing the forward pass, what is the value of $x_j^{(1)}$ in terms of the elements of $\{x_i^{(0)} : i = 1, \dots, d^{(0)}\}$? What is the relationship between each $x_j^{(1)}$?

Solution: Since all of the weights are equal, we have:

$$x_j^{(1)} = g\left(\sum_{i=1}^{d^{(0)}} w_{ij}^{(1)} x_i^{(0)}\right) = g\left(w \sum_{i=1}^{d^{(0)}} x_i^{(0)}\right)$$

Note that this equation does not depend on j so all $x_j^{(1)}$ are equal.

(b) After the backward pass of backpropagation, what is the relation between the members of the set $\{\delta_i^{(1)} : i = 1, \dots, d^{(1)}\}$, assuming we have calculated $\{\delta_j^{(2)} : j = 1, \dots, d^{(2)}\}$?

Solution: Since all of the weights are equal:

$$\begin{aligned}\delta_i^{(1)} &= \sum_{j=1}^{d^{(2)}} \delta_j^{(2)} w_{ij}^{(2)} g'(S_i^{(1)}) \\ &= g'(S_i^{(1)}) w \sum_{j=1}^{d^{(2)}} \delta_j^{(2)} \\ &= g'\left(w \sum_{k=1}^{d^{(0)}} x_k^{(0)}\right) w \sum_{j=1}^{d^{(2)}} \delta_j^{(2)}\end{aligned}$$

The sum term is the same for all values of i in layer 1. The members of the set are equal.

(c) For a reasonable loss function, can we say the same about each $\delta_i^{(2)}$?

Solution: No. $\delta_i^{(2)}$ depends on y_i, which is different for each i. Note that *any* reasonable output loss/activation should result in $\delta_i^{(2)}$ depending on a target y_i (otherwise the output is not attempting to approach a particular target value).

(d) After the weights are updated and one iteration of gradient descent has been completed, what can we say about the weights?

Solution: Our gradient descent update looks like this:

$$\begin{aligned} w_{ij}^{(l)} &= w_{ij}^{(l)} - \eta \delta_j^{(l)} x_i^{(l-1)} = w - \eta \delta_j^{(l)} x_i^{(l-1)} \\ \implies w_{ij}^{(1)} &= w - \eta \delta_j^{(1)} x_i^{(0)} \\ \implies w_{ij}^{(2)} &= w - \eta \delta_j^{(2)} x_i^{(1)} \end{aligned}$$

For a given i, $w_{ij}^{(1)}$ will be the same for all j because $\delta_j^{(1)}$ is equal for all j.

For a given j, $w_{ij}^{(2)}$ will be the same for all i because $x_i^{(1)}$ is equal for all i.

(e) *Even though* $w_{ij}^{(2)}$ is different for each j (but for a fixed j, it is the same for each i), this pattern continues. Why?

Solution: Let $w_{ij}^{(1)} = w_i^{(1)}$ for all i. Let $w_{ij}^{(2)} = w_j^{(2)}$ for all j. Then:

$$x_j^{(1)} = g\left(\sum_{i=1}^{d^{(0)}} w_{ij}^{(1)} x_i^{(0)}\right) = g\left(\sum_{i=1}^{d^{(0)}} w_i^{(1)} x_i^{(0)}\right)$$

Which does not depend on j. Also,

$$\delta_i^{(1)} = \sum_{j=1}^{d^{(2)}} \delta_j^{(2)} w_{ij}^{(2)} g'\left(\sum_{k=1}^{d^{(0)}} w_{ki}^{(1)} x_k^{(0)}\right) = \sum_{j=1}^{d^{(2)}} \delta_j^{(2)} w_j^{(2)} g'\left(\sum_{k=1}^{d^{(0)}} w_k^{(1)} x_k^{(0)}\right)$$

Which does not depend on j.

All $x_j^{(1)}$ being the same and all $\delta_i^{(1)}$ being the same will continue no matter how many iterations you perform. Intuitively, $x_j^{(1)}$ will always be the same for all j, due to the "outgoing" symmetry in the weights in layer 1, and the $\delta_i^{(1)}$ will always be the same due to the "incoming" symmetry in the weights in layer 2.

(f) To solve this problem, we randomly initialize our weights. This is called symmetry breaking. Why are we able to set our weights to 0 for logistic regression?

Solution: Logistic regression is a fully-connected neural network with one output node and zero hidden layers (remember that the $\delta_j^{(2)}$'s are different).

Another reason: in logistic regression, the loss function is convex. Any starting point should lead us to the global optimum.