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:

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

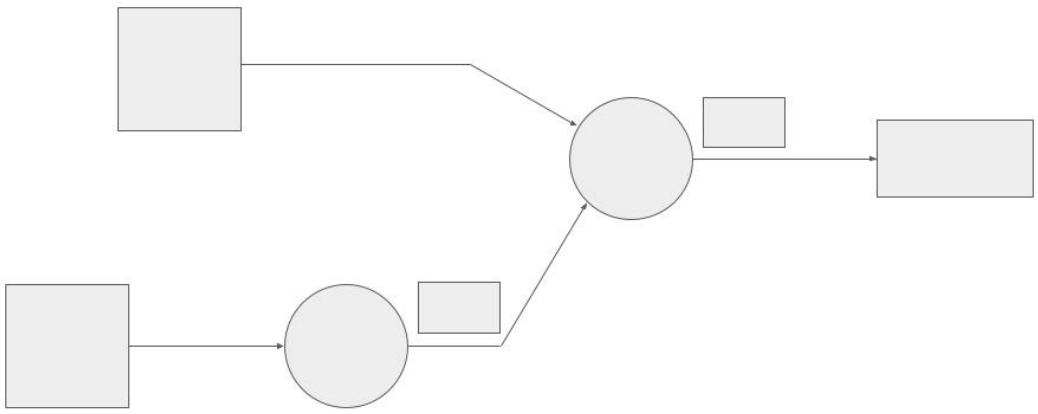
Here, $\mathbf{x} \in \mathbb{R}^{d^{(0)}}$ is the input vector, $\mathbf{h} \in \mathbb{R}^{d^{(1)}}$ contains the hidden layer activations after being passed through the sigmoid $s(\cdot)$ nonlinearity, $\mathbf{z} \in \mathbb{R}^{d^{(2)}}$ is the output vector, $\mathbf{y} \in \mathbb{R}^{d^{(2)}}$ 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:

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

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

(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.**



(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?**

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

(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.

(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.

(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.

(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?

(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.**

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 $x_1^{(0)}, \ldots, x_{d^{(0)}}^{(0)}$. 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 $\{x_i^{(0)} : i = 1, \ldots, d^{(0)}\}$? What is the relationship between each $x_j^{(1)}$?

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

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

(d) *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?

(e) 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?