NOTES ON BACKPROPOGATION

IORDAN GANEV

Contents

1. Introduction

These notes are an exploration of the backpropogation algorithm for computing the gradient of a loss function on the parameter space of a neural network.

1.1. **What is a neural network?** We view a neural network as a sequence of vector spaces, known as 'layers', together with (1) a sequence of affine maps between the successive layers, and (2) a non-linear activation applied to each layer. The composition of these maps is known as the feedforward function of the network. We give a mathematical formulation of neural networks and their parameter spaces in Section 2; for now, we note that the parameters of a neural network consist of a weight matrix and bias vector for each affine map; changing these parameters alters the feedforward function.

1.2. **Gradient descent.** Fitting a neural network model to labeled sample data amounts to finding parameters for which the feedforward function reproduces the labels from the inputs as closely as possible. More precisely, one defines a loss function on the parameter space based on the sample data, and seeks parameters that minimize this loss function. Gradient descent is a powerful technique to locate such a minimum; it consists of iteratively applying the following steps: take the current set of values of parameters, compute the gradient of the loss at those parameter values, and update the parameters by subtracting a (small) multiple of the gradient from the current one. This process is known as 'training' the neural network.

1.3. **Backpropogation.** Gradient descent is most effective when the gradient of the loss function has a closed from expression, or is otherwise straightforward to compute. The backpropogation algorithm contributes to the gradient descent process by providing an efficient way of computing the gradient at a set of parameter values. We now give a brief summary of the algorithm, which consists of a forward pass and a backward pass.

The forward pass takes the input data and proceeds iteratively through the layers; for each layer, one computes and caches the intermediate feature vector resulting from the input data as well as the Jacobian matrix of the layer's activation function.

The backward pass uses the cached objects from the forward pass to iteratively compute gradients of the loss with respect to the intermediate feature vectors starting with the last feature vector and moving inwards. The backward pass also computes the gradient of the loss function with respect to each weight matrix and bias vector; these are given directly in terms of the gradients of loss with respect to the feature vectors, and also require the feature vectors cached in the forward pass. Due to the combination of a forward caching pass and a backward pass, the backpropogation algorithm can be regarded as an example of dynamic programming.

1.4. **Outline of these notes.** In these notes, we first set the notation for neural networks (Section 2) before providing the theoretical justification for backpropogation (Proposition 3.1 of Section 3). The key tool behind the proof is the multivariate version of the chain rule.We also give an explicit pseudo-code implementation of backpropogation (Algorithm 1). We then turn our attention to backpropogation in batches (Section 4) and extend the main result to batches (Proposition 4.1). We include a list of exercises (Section 5), and close with an appendix on alternative implementations of the backpropogation algorithm (Appendix A).

1.5. **Notation.** We denote the space of m by n matrices by $\mathbb{R}^{m\times n}$. We use the symbol @ for matrix multiplication, including multiplying a matrix with a vector. We use the symbol $\circ$ for function composition. Unless specified otherwise, we identify $\mathbb{R}^n$ with $\mathbb{R}^{n\times 1}$ so that a vector $v \in \mathbb{R}^n$ will be assumed to be a column vector. Its transpose v^T is a row vector, that is, an element of $\mathbb{R}^{1\times n}$. Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a function which is differentiable at $x = (x_1, \dots, x_n) \in \mathbb{R}^n$. Recall that the Jacobian matrix dF_x of F at x is the m by n matrix of partial derivatives $\frac{\partial F_j}{\partial x_i}$, where $j = 1, \dots, m$ and $i = 1, \dots, n$. If $m = 1$, the gradient $\nabla_x F$ of F at x is the column n-vector with i-th entry equal to the partial derivative $\frac{\partial F}{\partial x_i}$. Hence, the gradient is the transpose of the Jacobian: $\nabla_x F = (dF_x)^T$.

2. Neural networks

2.1. **The parameter space.** Consider a neural network with $L \geq 1$ layers, input dimension n_0, output dimension n_L, and hidden dimensions given by $n_1, \dots, n_{L-1}$. For convenience, we group the dimensions into a tuple $\mathbf{n} = (n_0, n_1, \dots, n_L)$. The parameters of such a network consist of a $n_\ell \times n_{\ell-1}$ weight matrix W_ℓ and an n_ℓ-dimensional bias vector b_ℓ for every layer $\ell = 1, \ldots, L$. The space of all possible parameters for a network with dimension vector $\mathbf{n}$ is given as the following vector space:

$$\text{Param}(\mathbf{n}) = \mathbb{R}^{n_1 \times n_0} \times \mathbb{R}^{n_2 \times n_1} \times \cdots \times \times \mathbb{R}^{n_L \times n_{L-1}} \times \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \cdots \times \mathbb{R}^{n_L}.$$

We write an element therein as a pair $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ of tuples $\mathbf{W} = (W_1, \ldots, W_L)$ and $\mathbf{b} = (b_1, \ldots, b_L)$, so that $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b_\ell \in \mathbb{R}^{n_\ell}$ for $\ell = 1, \ldots, L$. When $\mathbf{n}$ is clear from context, we write simply Param for the parameter space.

2.2. **Activation functions.** Fix a piecewise differentiable activation function $\sigma_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell}$ for each $\ell = 1, \ldots, L$. When necessary, we set σ_0 to be the identity function on $\mathbb{R}^{n_0}$. The activations are conventionally be pointwise, which means that a single function $\mathbb{R} \to \mathbb{R}$ is applied to every coordinate of $\mathbb{R}^{n_\ell}$. However, we do not require this assumption. In what follows, we will be interested in differentiating each activation functions to form the associated Jacobian matrix $d(\sigma_\ell)_z \in \mathbb{R}^{n_\ell \times n_\ell}$ at a point $z \in \mathbb{R}^{n_\ell}$.

2.3. **Feedforward functions.** Given parameters $\boldsymbol{\theta} \in$ Param and activations σ_ℓ, we define the partial feedforward functions $F_{\boldsymbol{\theta},\ell} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_\ell}$ recursively:

$$F_{\boldsymbol{\theta},\ell}(x) = W_\ell \ @ \ (\sigma_{\ell-1} \circ F_{\boldsymbol{\theta},\ell-1}(x)) + b_\ell$$

for $\ell = 1, \ldots, L$, with $F_{\boldsymbol{\theta},0}(x) = x$. The full feedforward function $F_{\boldsymbol{\theta}} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ is given by applying the last activation the final partial feedforwrad function, i.e, $F_{\boldsymbol{\theta}} = \sigma_L \circ F_{\boldsymbol{\theta},L}$

2.4. **Loss functions.** We fix a differentiable cost function $C : \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \to \mathbb{R}$, such as mean square error or cross-entropy. The loss function $\mathcal{L}$ of our model is defined as:

$$\mathcal{L} : \text{Param} \times \mathbb{R}^{n_0} \times \mathbb{R}^{n_L} \to \mathbb{R}, \qquad \mathcal{L}(\boldsymbol{\theta}, (x, y)) = C(F_{\boldsymbol{\theta}}(x), y). \tag{2.1}$$

where $\mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$ is the space of all possible training data pairs. In other words, given parameters $\boldsymbol{\theta}$ and a sample (x, y), the associated loss is the value of the cost function comparing $y \in \mathbb{R}^{n_L}$ to the output $F_{\boldsymbol{\theta}}(x)$ of the neural network with parameters $\boldsymbol{\theta}$ evaluated at $x \in \mathbb{R}^{n_0}$. We also have the ℓ-th partial loss function $\mathcal{L}_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}$ defined using reverse recursion as

$$\mathcal{L}_\ell(z) = \begin{cases} C(\sigma_L(z), y) & \text{for } \ell = L \\ \mathcal{L}_{\ell+1}\left(W_{\ell+1} \circ \sigma_\ell(z) + b_{\ell+1}\right) & \text{for } \ell = L-1, \ldots, 0 \end{cases}$$

This recursive definition is similar to that of the partial feedforward functions $F_{\boldsymbol{\theta},\ell}$; the distinction is that we use reverse recursion starting with the last layer to define the partial loss functions. By abuse of notation, we write $\nabla_z \mathcal{L}$ for the gradient $\nabla_z \mathcal{L}_\ell$ of $\mathcal{L}_\ell$ at $z \in \mathbb{R}^{n_\ell}$.

3. Main result

3.1. **Statement.** To state the main result, we fix $x \in \mathbb{R}^{n_0}$, $y \in \mathbb{R}^{n_L}$, and $\boldsymbol{\theta} \in \text{Param}$. We also fix an activation function $\sigma_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell}$ for each layer. For $\ell = 1, \dots, L$, set:

$\mathbf{z}_\ell := F_{\boldsymbol{\theta},\ell}(x) \in \mathbb{R}^{n_\ell}$	ℓ-th pre-activation feature vector
$\mathbf{a}_\ell := \sigma_\ell(\mathbf{z}_\ell) \in \mathbb{R}^{n_\ell}$	ℓ-th post-activation feature vector
$\hat{y} := F_{\boldsymbol{\theta}}(x) = \sigma(\mathbf{z}_L) \in \mathbb{R}^{n_L}$	the output of the network
$\texttt{Jac}_\ell := d(\sigma_\ell)_{\mathbf{z}_\ell} \in \mathbb{R}^{n_\ell \times n_\ell}$	Jacobian matrix of the activation

To be clear, $\texttt{Jac}_\ell$ is the Jacobian matrix of the activation σ_ℓ differentiated at the pre-activation feature vector $\mathbf{z}_\ell$. In the statement of the following proposition, $\nabla_{\hat{y}} C(-, y)$ denotes the gradient of the cost function as a function of the predicted value $\hat{y}$, with the true value y fixed.

Proposition 3.1. *Fix a sample point* $(x, y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$ *and parameters* $\boldsymbol{\theta} \in \text{Param}$. *Using notation defined above, we have, for* $\ell = 1, \dots L$*:*

$$\nabla_{\mathbf{z}_L} \mathcal{L} = \texttt{Jac}_L^T \mathbin{@} \nabla_{\hat{y}} C(-, y) \tag{3.1}$$

$$\nabla_{\mathbf{z}_{\ell-1}} \mathcal{L} = \texttt{Jac}_{\ell-1}^T \mathbin{@} W_\ell^T \mathbin{@} \nabla_{\mathbf{z}_\ell} \mathcal{L} \tag{3.2}$$

$$\nabla_{W_\ell} \mathcal{L} = \nabla_{\mathbf{z}_\ell} \mathcal{L} \mathbin{@} \mathbf{a}_{\ell-1}^T \tag{3.3}$$

$$\nabla_{b_\ell} \mathcal{L} = \nabla_{\mathbf{z}_\ell} \mathcal{L} \tag{3.4}$$

3.2. **Proofs.** We prove each of the claims in turn.

Proof of Equation 3.1. In the case $\ell = L$, the partial loss function is $\mathcal{L}_L = C(-, y) \circ \sigma_L$, so that the differential of $\mathcal{L}_L$ at $\mathbf{z}_L$ is given by:

$$d\left(\mathcal{L}_L\right)_{\mathbf{z}_L} = d\left(C(-, y) \circ \sigma_L\right)_{\mathbf{z}_L} = d\left(C(-, y)\right)_{\hat{y}} \mathbin{@} d(\sigma_L)_{\mathbf{z}_L}$$

where we use chain rule and the fact that $\hat{y} = \sigma_L(\mathbf{z}_L)$. Since the gradient is the transpose of the differential, taking transposes yields Equation 3.1. $\square$

Proof of Equation 3.2. Let $\ell \in \{L, \dots, 1\}$. Then the differential of $\mathcal{L}_{\ell-1}$ as $\mathbf{z}_{\ell-1}$ is given by:

$$\begin{aligned} d\left(\mathcal{L}_{\ell-1}\right)_{\mathbf{z}_{\ell-1}} &= d\left[\mathcal{L}_\ell\left(W_\ell \circ \sigma_{\ell-1}(\mathbf{z}_{\ell-1}) + b_\ell\right)\right]_{\mathbf{z}_\ell} \\ &= d\left[\mathcal{L}_\ell \circ \left[a \mapsto (W_\ell a + b_\ell)\right] \circ \sigma_{\ell-1}\right]_{\mathbf{z}_{\ell-1}} \\ &= d\left(\mathcal{L}_\ell\right)_{\mathbf{z}_\ell} \mathbin{@} W_\ell \mathbin{@} d(\sigma_{\ell-1})_{\mathbf{z}_{\ell-1}} \end{aligned}$$

where we use chain rule, the definition of $\mathbf{z}_{\ell+1}$ and the fact that the differential of an affine linear map at any point is given by the linear part[1]. Taking transposes yields Equation 3.2. $\square$

[1]The affine linear map in this case is $\mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$ taking a to $W_\ell \mathbin{@} a + b_\ell$, and the linear part is W_ℓ.

Proof of Equations 3.3 and 3.4. First note that the following diagram commutes:

$$\begin{tikzcd} \mathbb{R}^{n_\ell \times n_{\ell-1}} \arrow[r, "\phi"] \arrow[d, hook] & \mathbb{R}^{n_\ell} \arrow[d, "\mathcal{L}_\ell"] \\ \text{Param} \arrow[r, "\mathcal{L}"] & \mathbb{R} \end{tikzcd} \tag{3.5}$$

where the left vertical map is the natural inclusion, and $\phi(M) = M \,@\, \mathbf{a}_{\ell-1} + b_\ell$. The latter map is affine linear and its differential at any point is given by right multiplication by $\mathbf{a}_{\ell-1}$ (See Exercise 3). Hence, the transpose of the differential is given by right multiplication by the transpose $\mathbf{a}_{\ell-1}^T$ of $\mathbf{a}_{\ell-1}$, that is:

$$d\phi^T(v) = v \,@\, \mathbf{a}_{\ell-1}^T$$

for any $v \in \mathbb{R}^{n_\ell}$. Observe that the left-hand-side of Equation 3.3 is the gradient at W_ℓ of the 'down then across' composition in Diagram 3.5. Hence, it is also equal to the gradient at W_ℓ of the 'across then down' composition, and we have:

$$\nabla_{W_\ell}\mathcal{L} = \nabla_{W_\ell}\left[\mathcal{L}_\ell \circ \phi\right] = (d\phi)^T\left(\nabla_{\mathbf{z}_\ell}\mathcal{L}_\ell\right) = \nabla_{\mathbf{z}_\ell}\mathcal{L}_\ell \,@\, \mathbf{a}_{\ell-1}^T.$$

The argument for Equation 3.4 is similar; in fact simpler. □

Remark 3.2. We note that if σ_ℓ is pointwise, then the Jacobian matrix $\texttt{Jac}_\ell$ is a diagonal matrix. We write $\sigma'_\ell(\mathbf{z}_\ell) \in \mathbb{R}^{n_\ell}$ for the vector formed by applying the derivative σ'_ℓ to each coordinate in $\mathbf{z}_\ell$. In this pointwise case, Equation 3.2 can be written as the Hadamard product of the $n_{\ell-1}$ vectors $\sigma'(\mathbf{z}_{\ell-1})$ and $W_\ell^T \,@\, \nabla_{\mathbf{z}_\ell}\mathcal{L}$.

3.3. Algorithm.

In Algorithm 1, we use the equations of Proposition 3.1 to formulate an explicit algorithm that computes the gradients of the loss function using backpropogation. Note that the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(-,(x,y))$ is an element of Param, so has a coordinate for each W_ℓ and each b_ℓ.

Corollary 3.3. *Fix a sample point $(x,y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$ and parameters $\boldsymbol{\theta} \in$ Param. With these inputs, Algorithm 1 returns the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(-,(x,y))$ of the loss function $\mathcal{L}(-,(x,y))$ at $\boldsymbol{\theta}$.*

Proof. Fix $x \in \mathbb{R}^{n_0}$, $y \in \mathbb{R}^{n_L}$, and $\boldsymbol{\theta} \in$ Param. The definitions of $\mathbf{z}_\ell$ and $\texttt{Jac}_\ell$ given above match the definitions of the corresponding variables in Algorithm 1 (Lines 3 and 5). Also, the gradient $\nabla_{\mathbf{z}_\ell}\mathcal{L} = \nabla_{\mathbf{z}_\ell}\mathcal{L}_\ell$ of $\mathcal{L}_\ell$ at $\mathbf{z}_\ell$ corresponds to the variable `grad_z` in Algorithm 1. Now, Equation 3.1 matches Line 6 in Algorithm 1, Equation 3.2 matches Line 8, Equation 3.3 matches Line 9, and Equation 3.4 matches Line 10. The claim follows by inspection. □

In the pseudo-code of Algorithms 3 and 4 (appearing in Appendix A), we provide alternative implementations of backpropogation. Specifically, Algorithm 3 uses only the post-activation feature vectors, while Algorithm 4 uses both the pre- and post-activation feature vectors.

Algorithm 1: Computing Gradients Using Back Propagation

```
   inputs: sample point $(x, y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$,
           weights $\mathbf{W} = (W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}})_{\ell=1}^{L}$,
           activations $(\sigma_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell})_{\ell=1}^{L}$
1  $\mathbf{z}_0 \leftarrow x$
   // Forward propagation
2  for $\ell \leftarrow 1$ to $L$ do
3      $\mathbf{z}_\ell \leftarrow W_\ell$ @ $\sigma_{\ell-1}(\mathbf{z}_{\ell-1}) + b_\ell$                // feature vector
4      Jac$_\ell$ $\leftarrow$ Jacobian $(\sigma_L, \mathbf{z}_\ell)$                        // Jacobian matrix
5  end
   // Back propagation
6  grad_z$_L$ $\leftarrow$ Jac$_L^T$ @ Gradient $(C(-, y), \sigma_L(\mathrm{x}_L))$       // gradient w.r.t. $\mathbf{z}_L$
7  for $\ell \leftarrow L$ to $1$ do
8      grad_z$_{\ell-1}$ $\leftarrow$ Jac$_{\ell-1}^T$ @ $(W_\ell)^T$ @ grad_z$_\ell$          // gradient w.r.t. $\mathbf{z}_\ell$
9      grad_W$_\ell$ $\leftarrow$ grad_z$_\ell$ @ $(\sigma_{\ell-1}(\mathbf{z}_{\ell-1}))^T$           // gradient w.r.t. $W_\ell$
10     grad_b$_\ell$ $\leftarrow$ grad_z$_\ell$                                   // gradient w.r.t. $b_\ell$
11 end
12 return: gradient (grad_W$_\ell$ , grad_b$_\ell$)$_{\ell=1}^{L}$
```

4. Batches

We now adapt backpropogation to the setting of working with a batch of N data points.

4.1. **Batch size first.** Suppose we have samples $(x_1, y_1), \ldots, (x_N, y_N)$ where each (x_i, y_i) belongs to $\mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$. It is conventional to package these samples into matrices using the 'batch size first' convention:

$$X \in \mathbb{R}^{N \times n_0} \qquad \text{and} \qquad Y \in \mathbb{R}^{N \times n_L}$$

where the i-th row of the matrix X is $X[i] = x_i^T$ and similarly for Y. This convention may be less intuitive than the 'batch size last' convention where X would have size $n_0 \times N$; indeed the 'batch size first' convention requires more adjustments to our previous discussion. However, it is common enough that it merits discussion.

4.2. **Notation and result.** For fixed parameters $\boldsymbol{\theta}$ and input batch $X \in \mathbb{R}^{N \times n_0}$, we set $Z_0 = A_0 = X$ and recursively define the ℓ-th pre- and post-activation feature matrix Z_ℓ and A_ℓ, respectively, as:

$$\begin{aligned} Z_\ell &= A_{\ell-1} \,@\, W_\ell^T + \mathbb{1}_N \,@\, b_\ell^T && \in \mathbb{R}^{N \times n_\ell} \\ A_\ell &= \sigma_\ell\left(Z_\ell\right) \in \mathbb{R}^{N \times n_\ell} && \in \mathbb{R}^{N \times n_\ell} \end{aligned}$$

for $\ell = 1, \ldots, L$, where each activation σ_ℓ is applied to the rows, and $\mathbb{1}_N$ is the column vector of all ones in $\mathbb{R}^N$. Note that $\mathbb{1}_N$ @ b_ℓ^T is an $N \times n_\ell$ matrix, each of whose rows is a copy of (the transpose of) the vector $b_\ell \in \mathbb{R}^{n_\ell}$. Hence, we have a feedforward function[2]

$$\widetilde{F}_{\boldsymbol{\theta}} : \mathbb{R}^{N \times n_0} \to \mathbb{R}^{N \times n_L}$$

taking X to the matrix of predictions $\hat{Y} := A_L \in \mathbb{R}^{N \times n_L}$. For each layer $\ell = 1, \ldots, L$ and each sample $i = 1, \ldots, N$, set $\mathtt{Jac}_{i,\ell} \in \mathbb{R}^{n_\ell \times n_\ell}$ to be the Jacobian of the activation $\sigma_\ell : \mathbb{R}^{n_L} \to \mathbb{R}^{n_L}$ at the i-th row $Z_\ell[i]$ of $Z_\ell \in \mathbb{R}^{N \times n_\ell}$.

Next, given a cost function $C : \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \to \mathbb{R}$, define the extended cost function by summing C over the pairs of input rows:

$$\widetilde{C} : \mathbb{R}^{N \times n_L} \times \mathbb{R}^{N \times n_L} \to \mathbb{R}, \qquad (U, V) \mapsto \sum_i^N C(U[i], V[i])$$

where $U[i]$ and $V[i]$ are the i-th rows of the $N \times n_L$ matrices U and V, respectively. As in the single sample case, our sample data (X, Y) gives rise to a loss function

$$\mathcal{L} = \mathcal{L}_{(X,Y)} : \mathrm{Param} \to \mathbb{R}, \qquad \boldsymbol{\theta} \mapsto \widetilde{C}(\widetilde{F}_{\boldsymbol{\theta}}(X), Y).$$

For every layer ℓ, is a partial loss function $\widetilde{\mathcal{L}}_\ell : \mathbb{R}^{N \times n_\ell} \to \mathbb{R}$ which first applies the remainder of the feedforward function with parameters $\boldsymbol{\theta}$ (sending Z_ℓ to $\widetilde{F}_{\boldsymbol{\theta}}(X) = \hat{Y}$ in our notation), and then applies the cost function $\widetilde{C}(-, Y)$. We abbreviate the gardient $\nabla_{Z_\ell} \widetilde{\mathcal{L}}_\ell$ of the partial loss function $\widetilde{\mathcal{L}}_\ell$ at Z_ℓ by simply $\nabla_{Z_\ell} \mathcal{L}$; this is an N by n_ℓ matrix.

Adapting the proof of Proposition 3.1 one can prove the following:

Proposition 4.1. *Fix a batch of sample data $X \in \mathbb{R}^{N \times n_0}$ and $Y \in \mathbb{R}^{N \times n_L}$, and parameters $\boldsymbol{\theta} \in \mathrm{Param}$. Using the notation $[i]$ to denote the i-th row of a matrix, we have:*

$$(\nabla_{Z_L} \mathcal{L})[i] = \nabla_{\hat{Y}[i]} C(-, Y[i]) \text{ @ } \mathtt{Jac}_{i,L} \tag{4.1}$$

$$(\nabla_{Z_{\ell-1}} \mathcal{L})[i] = (\nabla_{Z_\ell} \mathcal{L})[i] \text{ @ } W_\ell \text{ @ } \mathtt{Jac}_{i,\ell-1} \tag{4.2}$$

$$\nabla_{W_\ell} \mathcal{L} = \left(\nabla_{Z_\ell} \mathcal{L}\right)^T \text{ @ } A_{\ell-1} \tag{4.3}$$

$$\nabla_{b_\ell} \mathcal{L} = \left(\nabla_{Z_\ell} \mathcal{L}\right)^T \text{ @ } \mathbb{1}_N. \tag{4.4}$$

for $i = 1, \ldots, N$, and $\ell = 1, \ldots, L$.

4.3. **Tensors and endomorphisms.** Recall that there is an isomorphism $\mathbb{R}^{N \times n} \simeq \mathbb{R}^N \otimes \mathbb{R}^n$. Explicitly, a matrix $M \in \mathbb{R}^{N \times n}$ can be identified with $\sum_{i=1}^N e_i \otimes M[i]^T \in \mathbb{R}^N \otimes \mathbb{R}^n$ where $M[i]$ is the i-th row of M, and e_i is the i-th basis element of $\mathbb{R}^N$. Also recall that $\mathbb{R}^{N \times N}$ can be identified with the algebra of linear endomorphisms of the vector space $\mathbb{R}^N$, that is, $\mathbb{R}^{N \times N} \simeq \mathrm{End}(\mathbb{R}^N)$. Futhermore, we have an isomorphism:

$$\mathrm{End}(\mathbb{R}^N) \otimes \mathrm{End}(\mathbb{R}^n) \simeq \mathrm{End}(\mathbb{R}^N \otimes \mathbb{R}^n)$$

[2]If desired, one can define the partial feedforward functions $\widetilde{F}_{\boldsymbol{\theta},\ell} : \mathbb{R}^{N \times n_0} \to \mathbb{R}^{N \times n_\ell}$. These will be similar to the single sample case, but with transposes where appropriate.

Returning to backpropogation, for $\ell = 1, \dots, L$, set:

$$\Phi_\ell := \sum_i E_{ii} \otimes \mathtt{Jac}_{i,\ell}^T \in \mathbb{R}^{N\times N} \otimes \mathbb{R}^{n_\ell \times n_\ell} \simeq \mathrm{End}(\mathbb{R}^N \otimes \mathbb{R}^{n_\ell})$$

where $E_{ii} \in \mathbb{R}^{N\times N}$ is the elementary matrix with 1 in the diagonal (i,i) entry and zeros elsewhere. Identifying Φ_ℓ as an endomorphism of $\mathbb{R}^N \otimes \mathbb{R}^{n_\ell}$, equations 4.1 and 4.2 can be written as:

$$\nabla_{Z_L}\mathcal{L} = \Phi_L \left(\nabla_{\hat{Y}} C(-, Y)\right)$$
$$\nabla_{Z_{\ell-1}}\mathcal{L} = \Phi_{\ell-1} \left(\nabla_{Z_\ell}\mathcal{L} \mathbin{@} W_\ell\right)$$

noting that $\nabla_{\hat{Y}} C(-, Y) \in \mathbb{R}^{N\times n_L}$ and $\nabla_{Z_\ell}\mathcal{L} \mathbin{@} W_\ell \in \mathbb{R}^{N\times n_{\ell-1}}$. Justifying this reformulation uses the facts that (1) $E_{ii}(e_i) = e_i$ for any $i = 1, \dots, N$, and (2) $J^T @ v^T = (v @ J)^T$ for any matrix $J \in \mathbb{R}^{n\times n}$ and row vector $v \in R^{1\times n}$.

4.4. **Algorithm.** We use Proposition 4.1 to implement an algorithm for computing the gradients of the parameters using backpropogation in batches (Algorithm 2). The justification of this algorithm proceeds along similar lines to that of Corollary 3.3.

Algorithm 2: Computing Gradients Using Back Propagation with Batches

```
inputs: batch $X \in \mathbb{R}^{N\times n_0}$ , $Y \in \mathbb{R}^{N\times n_L}$,
        weights $\mathbf{W} = (W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}})_{\ell=1}^L$ ,
        activations $(\sigma_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell})_{\ell=1}^L$

$Z_0 \leftarrow X$
for $\ell \leftarrow 1$ to $L$ do                                  // Forward propagation
    $Z_\ell \leftarrow \sigma_{\ell-1}(Z_{\ell-1}) @ W_\ell^T + \mathbb{1}_N @ b_\ell^T$      // Apply activation to rows
end

$G = \mathtt{Gradient}(C(-,y), \sigma(z_L))$                      // Set-up for Back prop.

for $i \leftarrow 1$ to $N$ do
    $\mathtt{Jac}_{i,\ell} \leftarrow \mathtt{Jacobian}(\sigma_L, Z_\ell[i])$                 // Jacobian matrix
    $\mathtt{grad\_Z}_L[i] \leftarrow G[i] @ \mathtt{Jac}_{i,L}$                 // gradient w.r.t. $Z_L$
end

for $\ell \leftarrow L$ to 1 do                                    // Back propagation
    for $i \leftarrow 1$ to $N$ do
        $\mathtt{grad\_Z}_{\ell-1}[i] \leftarrow \mathtt{grad\_Z}_\ell[i] @ W_\ell @ \mathtt{Jac}_{i,\ell-1}$   // gradient w.r.t. $Z_{\ell-1}$
    end

    $\mathtt{grad\_W}_\ell \leftarrow \mathtt{grad\_Z}_\ell^T @ \sigma(Z_{\ell-1})$             // gradient w.r.t. $W_\ell$
    $\mathtt{grad\_b}_\ell \leftarrow \mathtt{grad\_Z}_\ell^T @ \mathbb{1}_N$                  // gradient w.r.t. $b_\ell$
end

return: gradients $(\mathtt{grad\_W}_\ell , \mathtt{grad\_b}_\ell)_{\ell=1}^L$
```

5. Exercises

(1) Compute the Jacobian matrix (when it exists) for the following activation functions $\sigma : \mathbb{R}^n \to \mathbb{R}^n$. Indicate where the function is not differentiable.
  (a) Pointwise sigmoid $z \mapsto \frac{1}{1+e^{-z}}$.
  (b) Pointwise ReLU $z \mapsto \max(0, z)$.
  (c) Pointwise arctan $z \mapsto \arctan(z)$.
  (d) Softmax $z = (z_1, \dots, z_n) \mapsto \left(\frac{\exp(z_1)}{\sum_j \exp(z_j)}, \dots, \frac{\exp(z_n)}{\sum_j \exp(z_j)}\right)$.
  (e) Radial Sigmoid $z \mapsto \frac{1}{1+e^{-|z|}} \cdot \frac{z}{|z|}$.

(2) Compute the Jacobian matrix (when it exists) and its transpose for the following 'pooling' functions[3] $\sigma : \mathbb{R}^n \to \mathbb{R}$. Indicate where the function is not differentiable.
  (a) Max pooling $z = (z_1, \dots, z_n) \mapsto \max(z_i)$.
  (b) Average pooling $z = (z_1, \dots, z_n) \mapsto \frac{1}{n} \sum_i z_i$.

(3) Consider $\phi : \mathbb{R}^{n \times m} \to \mathbb{R}^n$ defined by $M \mapsto M \mathbin{@} a + b$ for fixed $a \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$. Prove that the transpose of the differential at any $M \in \mathbb{R}^{n \times m}$ is given by: $d\phi_M^T(v) = v \mathbin{@} a^T$ for any $v \in \mathbb{R}^n$. (Note the lack of dependency on M.)

(4) Verify that the run time of Algorithm 1 is $O(Ln_{\max}^2)$, where $n_{\max} = \max_{0 \le \ell \le L}(n_\ell)$. Similarly, verify that the run time of Algorithm 2 is $O(NLn_{\max}^2)$.

(5) Recover Algorithm 1 from Algorithm 2 by setting $N = 1$ and taking transposes where appropriate.

(6) Verify that Algorithms 2, 3, 4, and 5 accurately compute the gradients $\nabla_{W_\ell}\mathcal{L}$ and $\nabla_{b_\ell}\mathcal{L}$. Interpret each of $\texttt{grad\_a}_\ell$ and $\texttt{grad\_A}_\ell$ as gradients of partial loss functions $\mathbb{R}^{n_\ell} \to \mathbb{R}$ and $\mathbb{R}^{N \times n_\ell} \to \mathbb{R}$, with respect to the post-activation feature vectors a_ℓ and A_ℓ, respectively.

[3]Hint: The Jacobian of max pooling is a column vector with a single one at the index of the maximum entry, while the Jacobian of average pooling is the column vector with $1/n$ in every coordinate.

Appendix A. Variations of the algorithm

In this appendix, we provide alternative implementations of the backpropogation algorithm. Algorithm 3 uses only the post-activation feature vectors, while Algorithm 4 uses both the pre- and post-activation feature vectors. Algorithm 5 uses both pre- and post-activation feature vectors in a batch setting.

Algorithm 3: Computing Gradients Using Back Propagation (alternative version)

```
inputs: sample point $(x, y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$,
        weights $\mathbf{W} = (W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}})_{\ell=1}^{L}$,
        activations $(\sigma_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell})_{\ell=1}^{L}$
$\mathtt{a}_0 \leftarrow x$
for $\ell \leftarrow 1$ to $L$ do                                        // Forward propagation
    $\mathtt{a}_\ell \leftarrow \sigma_\ell\,(W_\ell$ @ $\mathtt{a}_{\ell-1} + b_\ell)$                         // feature vector
    $\mathtt{Jac}_\ell \leftarrow \mathtt{Jacobian}\,(\sigma_L, W_\ell$ @ $\mathtt{a}_{\ell-1} + b_\ell)$            // Jacobian matrix
end
$\mathtt{grad\_a}_L = \mathtt{Gradient}\,(C(-, y), \mathtt{a}_L)$
for $\ell \leftarrow L$ to $1$ do                                        // Back propagation
    $\mathtt{grad\_W}_\ell \leftarrow \mathtt{Jac}_\ell^T$ @ $\mathtt{grad\_a}_\ell$ @ $\mathtt{a}_{\ell-1}^T$               // gradient w.r.t. $W_\ell$
    $\mathtt{grad\_b}_\ell \leftarrow \mathtt{Jac}_\ell^T$ @ $\mathtt{grad\_a}_\ell$                          // gradient w.r.t. $b_\ell$
    $\mathtt{grad\_a}_{\ell-1} \leftarrow (W_{\ell-1})^T$ @ $\mathtt{Jac}_\ell^T$ @ $\mathtt{grad\_a}_\ell$           // gradient w.r.t. $\mathtt{a}_\ell$
end
return: gradients $(\mathtt{grad\_W}_\ell\ ,\ \mathtt{grad\_b}_\ell)_{\ell=1}^{L}$
```

Algorithm 4: Computing Gradients Using Back Propagation (combined version)

```
inputs: sample point $(x, y) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$,
        weights $\mathbf{W} = (W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}})_{\ell=1}^{L}$,
        activations $(\sigma_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell})_{\ell=1}^{L}$
$\mathtt{a}_0 \leftarrow x$
for $\ell \leftarrow 1$ to $L$ do                                  // Forward propagation
    $\mathtt{z}_\ell \leftarrow W_\ell$ @ $\mathtt{a}_{\ell-1} + b_\ell$
    $\mathtt{a}_\ell \leftarrow \sigma_\ell(\mathtt{z}_\ell)$
    $\mathtt{Jac}_\ell \leftarrow \mathtt{Jacobian}(\sigma_L, \mathtt{z}_\ell)$                        // Jacobian matrix
end
$\mathtt{grad\_a}_L = \mathtt{Gradient}(C(-, y), \mathtt{a}_L)$
for $\ell \leftarrow L$ to $1$ do                                     // Back propagation
    $\mathtt{grad\_z}_\ell \leftarrow \mathtt{Jac}_\ell^T$ @ $\mathtt{grad\_a}_\ell$                      // gradient w.r.t. $\mathtt{z}_\ell$
    $\mathtt{grad\_a}_{\ell-1} \leftarrow (W_{\ell-1})^T$ @ $\mathtt{grad\_z}_\ell$                  // gradient w.r.t. $\mathtt{a}_\ell$
    $\mathtt{grad\_W}_\ell \leftarrow \mathtt{grad\_z}_\ell$ @ $\mathtt{a}_{\ell-1}^T$                   // gradient w.r.t. $W_\ell$
    $\mathtt{grad\_b}_\ell \leftarrow \mathtt{grad\_z}_\ell$                          // gradient w.r.t. $b_\ell$
end
return: gradients $(\mathtt{grad\_W}_\ell, \mathtt{grad\_b}_\ell)_{\ell=1}^{L}$
```

Algorithm 5: Computing Gradients Using Back Propagation with Batches (combined version)

```
inputs: batch $X \in \mathbb{R}^{N \times n_0}$ , $Y \in \mathbb{R}^{N \times n_L}$,
        weights $\mathbf{W} = (W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}})_{\ell=1}^{L}$ ,
        activations $(\sigma_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell})_{\ell=1}^{L}$

$\mathtt{A}_0 \leftarrow X$
for $\ell \leftarrow 1$ to $L$ do                                        // Forward propagation
    $\mathtt{Z}_\ell \leftarrow \mathtt{A}_{\ell-1}$ @ $W_\ell^T + \mathbb{1}_N$ @ $b_\ell^T$             // Apply activation to rows
    $\mathtt{A}_\ell \leftarrow \sigma_\ell(\mathtt{Z}_\ell)$
    for $i \leftarrow 1$ to $N$ do
        $\mathtt{Jac}_{i,\ell} \leftarrow \mathtt{Jacobian}(\sigma_L, \mathtt{Z}_\ell[i])$                      // Jacobian matrix
    end
end

$\mathtt{grad\_A}_L = \mathtt{Gradient}(C(-, y), \mathtt{A}_L)$
for $\ell \leftarrow L$ to $1$ do                                         // Back propagation
    for $i \leftarrow 1$ to $N$ do
        $\mathtt{grad\_Z}_\ell[i] \leftarrow \mathtt{grad\_A}_\ell[i]$ @ $\mathtt{Jac}_{i,\ell}$                  // gradient w.r.t. $\mathtt{Z}_\ell$
    end
    $\mathtt{grad\_A}_{\ell-1} \leftarrow \mathtt{grad\_Z}_\ell$ @ $W_{\ell-1}$                          // gradient w.r.t. $\mathtt{A}_\ell$
    $\mathtt{grad\_W}_\ell \leftarrow \mathtt{grad\_Z}_\ell^T$ @ $\mathtt{A}_{\ell-1}$                          // gradient w.r.t. $W_\ell$
    $\mathtt{grad\_b}_\ell \leftarrow \mathtt{grad\_Z}_\ell^T$ @ $\mathbb{1}_N$                              // gradient w.r.t. $b_\ell$
end
return: gradients $(\mathtt{grad\_W}_\ell , \mathtt{grad\_b}_\ell)_{\ell=1}^{L}$
```