Computing Neural Network Gradients

Kevin Clark

1 Introduction

The purpose of these notes is to demonstrate how to quickly compute neural network gradients in a completely vectorized way. It is complementary to the last part of lecture 3 in CS224n 2019, which goes over the same material.

2 Vectorized Gradients

While it is a good exercise to compute the gradient of a neural network with respect to a single parameter (e.g., a single element in a weight matrix), in practice this tends to be quite slow. Instead, it is more efficient to keep everything in matrix/vector form. The basic building block of vectorized gradients is the *Jacobian Matrix*. Suppose we have a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ that maps a vector of length n to a vector of length m: $\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, ..., x_n), f_2(x_1, ..., x_n), ..., f_m(x_1, ..., x_n)]$. Then its Jacobian is the following $m \times n$ matrix:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

That is, $(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial f_i}{\partial x_j}$ (which is just a standard non-vector derivative). The Jacobian matrix will be useful for us because we can apply the chain rule to a vector-valued function just by multiplying Jacobians.

As a little illustration of this, suppose we have a function $\boldsymbol{f}(x) = [f_1(x), f_2(x)]$ taking a scalar to a vector of size 2 and a function $\boldsymbol{g}(\boldsymbol{y}) = [g_1(y_1, y_2), g_2(y_1, y_2)]$ taking a vector of size two to a vector of size two. Now let's compose them to get $\boldsymbol{g}(x) = [g_1(f_1(x), f_2(x)), g_2(f_1(x), f_2(x))]$. Using the regular chain rule, we can compute the derivative of $\boldsymbol{g}$ as the Jacobian

$$\frac{\partial \boldsymbol{g}}{\partial x} = \begin{bmatrix} \frac{\partial}{\partial x} g_1(f_1(x), f_2(x)) \\ \frac{\partial}{\partial x} g_2(f_1(x), f_2(x)) \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_1}{\partial f_2}\frac{\partial f_2}{\partial x} \\ \frac{\partial g_2}{\partial f_1}\frac{\partial f_1}{\partial x} + \frac{\partial g_2}{\partial f_2}\frac{\partial f_2}{\partial x} \end{bmatrix}$$

And we see this is the same as multiplying the two Jacobians:

$$\frac{\partial \boldsymbol{g}}{\partial x} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial x} = \begin{bmatrix} \frac{\partial g_1}{\partial f_1} & \frac{\partial g_1}{\partial f_2} \\ \frac{\partial g_2}{\partial f_1} & \frac{\partial g_2}{\partial f_2} \end{bmatrix} \begin{bmatrix} \frac{\partial f_1}{\partial x} \\ \frac{\partial f_2}{\partial x} \end{bmatrix}$$

3 Useful Identities

This section will now go over how to compute the Jacobian for several simple functions. It will provide some useful identities you can apply when taking neural network gradients.

(1) **Matrix times column vector with respect to the column vector** ($\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$, what is $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$?)

Suppose $\boldsymbol{W} \in \mathbb{R}^{n \times m}$. Then we can think of $\boldsymbol{z}$ as a function of $\boldsymbol{x}$ taking an m-dimensional vector to an n-dimensional vector. So its Jacobian will be $n \times m$. Note that

$$z_i = \sum_{k=1}^{m} W_{ik} x_k$$

So an entry $(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij}$ of the Jacobian will be

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^{m} W_{ik} x_k = \sum_{k=1}^{m} W_{ik} \frac{\partial}{\partial x_j} x_k = W_{ij}$$

because $\frac{\partial}{\partial x_j} x_k = 1$ if $k = j$ and 0 if otherwise. So we see that $\boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}}$

(2) **Row vector times matrix with respect to the row vector** ($\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W}$, what is $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$?)

A computation similar to (1) shows that $\boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}^T}$.

(3) **A vector with itself**
($\boldsymbol{z} = \boldsymbol{x}$, what is $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$?)
We have $z_i = x_i$. So

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} x_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}$$

So we see that the Jacobian $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$ is a diagonal matrix where the entry at (i, i) is 1. This is just the identity matrix: $\boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{I}}$. When applying the chain

rule, this term will disappear because a matrix or vector multiplied by the identity matrix does not change.

(4) **An elementwise function applied a vector**
($\boldsymbol{z} = f(\boldsymbol{x})$, what is $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$?)
Since f is being applied elementwise, we have $z_i = f(x_i)$. So

$$(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})_{ij} = \frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} f(x_i) = \begin{cases} f'(x_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases}$$

So we see that the Jacobian $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$ is a diagonal matrix where the entry at (i, i) is the derivative of f applied to x_i. We can write this as $\boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \text{diag}(f'(\boldsymbol{x}))}$.
Since multiplication by a diagonal matrix is the same as doing elementwise multiplication by the diagonal, we could also write $\boxed{\circ f'(\boldsymbol{x})}$ when applying the chain rule.

(5) **Matrix times column vector with respect to the matrix**
($\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$, $\boldsymbol{\delta} = \frac{\partial J}{\partial \boldsymbol{z}}$ what is $\frac{\partial J}{\partial \boldsymbol{W}} = \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$?)

This is a bit more complicated than the other identities. The reason for including $\frac{\partial J}{\partial \boldsymbol{z}}$in the above problem formulation will become clear in a moment.

First suppose we have a loss function J (a scalar) and are computing its gradient with respect to a matrix $\boldsymbol{W} \in \mathbb{R}^{n \times m}$. Then we could think of J as a function of $\boldsymbol{W}$ taking nm inputs (the entries of $\boldsymbol{W}$) to a single output (J). This means the Jacobian $\frac{\partial J}{\partial \boldsymbol{W}}$ would be a $1 \times nm$ vector. But in practice this is not a very useful way of arranging the gradient. It would be much nicer if the derivatives were in a $n \times m$ matrix like this:

$$\frac{\partial J}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial J}{\partial W_{11}} & \cdots & \frac{\partial J}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial W_{n1}} & \cdots & \frac{\partial J}{\partial W_{nm}} \end{bmatrix}$$

Since this matrix has the same shape as $\boldsymbol{W}$, we could just subtract it (times the learning rate) from $\boldsymbol{W}$ when doing gradient descent. So (in a slight abuse of notation) let's find this matrix as $\frac{\partial J}{\partial \boldsymbol{W}}$ instead.

This way of arranging the gradients becomes complicated when computing $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$. Unlike J, $\boldsymbol{z}$ is a vector. So if we are trying to rearrange the gradients like with $\frac{\partial J}{\partial \boldsymbol{W}}$, $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$ would be an $n \times m \times n$ tensor! Luckily, we can avoid the issue by taking the gradient with respect to a single weight W_{ij} instead.

$\frac{\partial \boldsymbol{z}}{\partial W_{ij}}$ is just a vector, which is much easier to deal with. We have

$$z_k = \sum_{l=1}^{m} W_{kl} x_l$$
$$\frac{\partial z_k}{\partial W_{ij}} = \sum_{l=1}^{m} x_l \frac{\partial}{\partial W_{ij}} W_{kl}$$

Note that $\frac{\partial}{\partial W_{ij}} W_{kl} = 1$ if $i = k$ and $j = l$ and 0 if otherwise. So if $k \neq i$ everything in the sum is zero and the gradient is zero. Otherwise, the only nonzero element of the sum is when $l = j$, so we just get x_j. Thus we find $\frac{\partial z_k}{\partial W_{ij}} = x_j$ if $k = i$ and 0 if otherwise. Another way of writing this is

$$\frac{\partial \boldsymbol{z}}{\partial W_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{th element}$$

Now let's compute $\frac{\partial J}{\partial W_{ij}}$

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial W_{ij}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial W_{ij}} = \sum_{k=1}^{m} \delta_k \frac{\partial z_k}{\partial W_{ij}} = \delta_i x_j$$

(the only nonzero term in the sum is $\delta_i \frac{\partial z_i}{\partial W_{ij}}$). To get $\frac{\partial J}{\partial \boldsymbol{W}}$ we want a matrix where entry (i, j) is $\delta_i x_j$. This matrix is equal to the outer product $\boxed{\frac{\partial J}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T}$

(6) **Row vector time matrix with respect to the matrix** ($\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W}$, $\boldsymbol{\delta} = \frac{\partial J}{\partial \boldsymbol{z}}$ what is $\frac{\partial J}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$?)

A similar computation to (5) shows that $\boxed{\frac{\partial J}{\partial \boldsymbol{W}} = \boldsymbol{x}^T \boldsymbol{\delta}}$.

(7) **Cross-entropy loss with respect to logits** ($\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{\theta})$, $J = CE(\boldsymbol{y}, \hat{\boldsymbol{y}})$, what is $\frac{\partial J}{\partial \boldsymbol{\theta}}$?)

The gradient is $\boxed{\frac{\partial J}{\partial \boldsymbol{\theta}} = \hat{\boldsymbol{y}} - \boldsymbol{y}}$

(or $(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$ if $\boldsymbol{y}$ is a column vector).

These identities will be enough to let you quickly compute the gradients for many neural networks. However, it's important to know how to compute Jacobians for other functions as well in case they show up. Some examples if you want practice: dot product of two vectors, elementwise product of two vectors, 2-norm of a vector. Feel free to use these identities in the assignments. One option is just to memorize them. Another option is to figure them out by looking at the dimensions. For example, only one ordering/orientation of $\boldsymbol{\delta}$ and $\boldsymbol{x}$ will produce the correct shape for $\frac{\partial J}{\partial \boldsymbol{W}}$ (assuming $\boldsymbol{W}$ is not square).

4 Gradient Layout

Jacobean formulation is great for applying the chain rule: you just have to multiply the Jacobians. However, when doing SGD it's more convenient to follow the convention "the shape of the gradient equals the shape of the parameter" (as we did when computing $\frac{\partial J}{\partial \boldsymbol{W}}$). That way subtracting the gradient times the learning rate from the parameters is easy. **We expect answers to homework questions to follow this convention.** Therefore if you compute the gradient of a column vector using Jacobian formulation, you should take the transpose when reporting your final answer so the gradient is a column vector. Another option is to always follow the convention. In this case the identities may not work, but you can still figure out the answer by making sure the dimensions of your derivatives match up. Up to you which of these options you choose!

5 Example: 1-Layer Neural Network

This section provides an example of computing the gradients of a full neural network. In particular we are going to compute the gradients of a one-layer neural network trained with cross-entropy loss. The forward pass of the model is as follows:

$$
\begin{aligned}
\boldsymbol{x} &= \text{input} \\
\boldsymbol{z} &= \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}_1 \\
\boldsymbol{h} &= \text{ReLU}(\boldsymbol{z}) \\
\boldsymbol{\theta} &= \boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2 \\
\hat{\boldsymbol{y}} &= \text{softmax}(\boldsymbol{\theta}) \\
J &= CE(\boldsymbol{y}, \hat{\boldsymbol{y}})
\end{aligned}
$$

It helps to break up the model into the simplest parts possible, so note that we defined $\boldsymbol{z}$ and $\boldsymbol{\theta}$ to split up the activation functions from the linear transformations in the network's layers. The dimensions of the model's parameters are

$$
\boldsymbol{x} \in \mathbb{R}^{D_x \times 1} \qquad \boldsymbol{b}_1 \in \mathbb{R}^{D_h \times 1} \qquad \boldsymbol{W} \in \mathbb{R}^{D_h \times D_x} \qquad \boldsymbol{b}_2 \in \mathbb{R}^{N_c \times 1} \qquad \boldsymbol{U} \in \mathbb{R}^{N_c \times D_h}
$$

where D_x is the size of our input, D_h is the size of our hidden layer, and N_c is the number of classes.

In this example, we will compute all of the network's gradients:

$$\frac{\partial J}{\partial \boldsymbol{U}} \qquad \frac{\partial J}{\partial \boldsymbol{b}_2} \qquad \frac{\partial J}{\partial \boldsymbol{W}} \qquad \frac{\partial J}{\partial \boldsymbol{b}_1} \qquad \frac{\partial J}{\partial \boldsymbol{x}}$$

To start with, recall that $\text{ReLU}(x) = \max(x, 0)$. This means

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if otherwise} \end{cases} = \text{sgn}(\text{ReLU}(x))$$

where sgn is the signum function. Note that we are able to write the derivative of the activation in terms of the activation itself.

Now let's write out the chain rule for $\frac{\partial J}{\partial \boldsymbol{U}}$ and $\frac{\partial J}{\partial \boldsymbol{b}_2}$:

$$\frac{\partial J}{\partial \boldsymbol{U}} = \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{U}}$$
$$\frac{\partial J}{\partial \boldsymbol{b}_2} = \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{b}_2}$$

Notice that $\frac{\partial J}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{\theta}} = \frac{\partial J}{\partial \boldsymbol{\theta}}$ is present in both gradients. This makes the math a bit cumbersome. Even worse, if we're implementing the model without automatic differentiation, computing $\frac{\partial J}{\partial \boldsymbol{\theta}}$ twice will be inefficient. So it will help us to define some variables to represent the intermediate derivatives:

$$\boldsymbol{\delta}_1 = \frac{\partial J}{\partial \boldsymbol{\theta}} \qquad \boldsymbol{\delta}_2 = \frac{\partial J}{\partial \boldsymbol{z}}$$

These can be thought as the error signals passed down to $\boldsymbol{\theta}$ and $\boldsymbol{z}$ when doing backpropagation. We can compute them as follows:

$$\begin{aligned}
\boldsymbol{\delta}_1 &= \frac{\partial J}{\partial \boldsymbol{\theta}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T && \text{this is just identity (7)} \\
\boldsymbol{\delta}_2 &= \frac{\partial J}{\partial \boldsymbol{z}} = \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} && \text{using the chain rule} \\
&= \boldsymbol{\delta}_1 \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} && \text{substituting in } \boldsymbol{\delta}_1 \\
&= \boldsymbol{\delta}_1 \, \boldsymbol{U} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} && \text{using identity (1)} \\
&= \boldsymbol{\delta}_1 \, \boldsymbol{U} \circ \text{ReLU}'(\boldsymbol{z}) && \text{using identity (4)} \\
&= \boldsymbol{\delta}_1 \, \boldsymbol{U} \circ \text{sgn}(\boldsymbol{h}) && \text{we computed this earlier}
\end{aligned}$$

A good way of checking our work is by looking at the dimensions of the Jacobians:

$$\begin{array}{ccccccc}
\dfrac{\partial J}{\partial \boldsymbol{z}} & = & \boldsymbol{\delta}_1 & & \boldsymbol{U} & \circ & \text{sgn}(\boldsymbol{h}) \\
(1 \times D_h) & & (1 \times N_c) & & (N_c \times D_h) & & (D_h)
\end{array}$$

We see that the dimensions of all the terms in the gradient match up (i.e., the number of columns in a term equals the number of rows in the next term). This will always be the case if we computed our gradients correctly.

Now we can use the error terms to compute our gradients. Note that we transpose out answers when computing the gradients for column vectors terms to follow the shape convention.

$$\frac{\partial J}{\partial \boldsymbol{U}} = \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{U}} = \boldsymbol{\delta}_1 \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{U}} = \boldsymbol{\delta}_1^T \boldsymbol{h}^T \qquad \text{using identity (5)}$$

$$\frac{\partial J}{\partial \boldsymbol{b}_2} = \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{b}_2} = \boldsymbol{\delta}_1 \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{b}_2} = \boldsymbol{\delta}_1^T \qquad \text{using identity (3) and transposing}$$

$$\frac{\partial J}{\partial \boldsymbol{W}} = \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta}_2 \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta}_2^T \boldsymbol{x}^T \qquad \text{using identity (5)}$$

$$\frac{\partial J}{\partial \boldsymbol{b}_1} = \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}_1} = \boldsymbol{\delta}_2 \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}_1} = \boldsymbol{\delta}_2^T \qquad \text{using identity (3) and transposing}$$

$$\frac{\partial J}{\partial \boldsymbol{x}} = \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = (\boldsymbol{\delta}_2 \boldsymbol{W})^T \qquad \text{using identity (1) and transposing}$$