Softmax activation derivative

The next calculation that we need to perform is the partial derivative of the Softmax function, which is a bit more complicated task than the derivative of the Categorical Cross-Entropy loss. Let's remind ourselves of the equation of the Softmax activation function and define the derivative:

$$S_{i,j} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \quad \rightarrow \quad \frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}}$$

Where $S_{i,j}$ denotes *j*-th Softmax's output of *i*-th sample, *z* — input array which is a list of input vectors (output vectors from the previous layer), $z_{i,j}$ — *j*-th Softmax's input of *i*-th sample, *L* — number of inputs, $z_{i,k}$ — *k*-th Softmax's input of *i*-th sample.

As we described in chapter 4, the Softmax function equals the exponentiated input divided by the sum of all exponentiated inputs. In other words, we need to exponentiate all of the values first, then divide each of them by the sum of all of them to perform the normalization. Each input to the Softmax impacts each of the outputs, and we need to calculate the partial derivative of each output with respect to each input. From the programming side of things, if we calculate the impact of one list on the other list, we'll receive a matrix of values as a result. That's exactly what we'll calculate here — we'll calculate the **Jacobian matrix** (which we'll explain later) of the vectors, which we'll dive deeper into soon.

To calculate this derivative, we need to first define the derivative of the division operation:

$$f(x) = \frac{g(x)}{h(x)} \quad \rightarrow \quad f'(x) = \frac{g'(x) \cdot h(x) - g(x) \cdot h'(x)}{[h(x)]^2}$$

In order to calculate the derivative of the division operation, we need to take the derivative of the numerator multiplied by the denominator, subtract the numerator multiplied by the derivative of the denominator from it, and then divide the result by the squared denominator.

We can now start solving the derivative:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}} =$$

Let's apply the derivative of the division operation:

$$= \frac{\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot \frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}}{\left[\sum_{l=1}^{L} e^{z_{i,l}}\right]^2} =$$

At this step, we have two partial derivatives present in the equation. For the one on the right side of the numerator (right side of the subtraction operator):

$$\frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}$$

We need to calculate the derivative of the sum of the constant,*e* (Euler's number), raised to power $z_{i,l}$ (where *l* denotes consecutive indices from *1* to the number of the Softmax outputs — *L*) with respect to the $z_{i,k}$. The derivative of the sum operation is the sum of derivatives, and the derivative of the constant *e* raised to power *n* (e^n) with respect to *n* equals e^n:

$$\frac{d}{dn} e^n = e^n \cdot \frac{d}{dn} n = e^n \cdot 1 = e^n$$

It is a special case when the derivative of an exponential function equals this exponential function itself, as its exponent is exactly what we are deriving with respect to, thus its derivative equals *1*. We also know that the range *1...L* contains *k* (*k* is one of the indices from this range) exactly once and then, in this case, the derivative is going to equal *e* to the power of the $z_{i,k}$ (as *j* equals *k*) and *0* otherwise (when *j* does not equal *k* as $z_{i,l}$ won't contain $z_{i,k}$ and will be treated as a constant — The derivative of the constant equals *0*):

$$\frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}} = \frac{\partial}{\partial z_{i,k}} e^{z_{i,1}} + \frac{\partial}{\partial z_{i,k}} e^{z_{i,2}} + \ldots + \frac{\partial}{\partial z_{i,k}} e^{z_{i,k}} + \ldots + \frac{\partial}{\partial z_{i,k}} e^{z_{i,L-1}} + \frac{\partial}{\partial z_{i,k}} e^{z_{i,L}}$$

$$= 0 + 0 + \ldots + e^{z_{i,k}} + \ldots + 0 + 0 = e^{z_{i,k}}$$

The derivative on the left side of the subtraction operator in the denominator is a slightly different case:

$$\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}}$$

It does not contain the sum over all of the elements like the derivative we solved moments ago, so it can become either *0* if $j \neq k$ or *e* to the power of the $z_{i,j}$ if $j=k$. That means, starting from this step, we need to calculate the derivatives separately for both cases. Let's start with $j=k$.

In the case of $j=k$, the derivative on the left side is going to equal *e* to the power of the $z_{i,j}$ and the derivative on the right solves to the same value in both cases. Let's substitute them:

$$= \frac{e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

The numerator contains the constant *e* to the power of $z_{i,j}$ in both the minuend (the value we are subtracting from) and subtrahend (the value we are subtracting from the minuend) of the subtraction operation. Because of this, we can regroup the numerator to contain this value multiplied by the subtraction of their current multipliers. We can also write the denominator as a multiplication of the value instead of using the power of *2*:

$$= \frac{e^{z_{i,j}} \cdot (\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}})}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

Then let's split the whole equation into 2 parts:

$$= \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} =$$

We moved *e* from the numerator and the sum from the denominator to its own fraction, and the content of the parentheses in the numerator, and the other sum from the denominator as another fraction, both joined by the multiplication operation. Now we can further split the "right" fraction into two separate fractions:

$$= \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot (\frac{\sum_{l=1}^{L} e^{z_{i,l}}}{\sum_{l=1}^{L} e^{z_{i,l}}} - \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}}) =$$

In this case, as it's a subtraction operation, we separated both values from the numerator, dividing them both by the denominator and applying the subtraction operation between new fractions. If we

look closely, the "left" fraction turns into the Softmax function's equation, as well as the "right" one, with the middle fraction solving to *1* as the numerator and the denominator are the same values:

$$= S_{i,j} \cdot (1 - S_{i,k})$$

Note that the "left" Softmax function carries the $_j$ parameter, and the "right" one $_k$ — both came from their numerators, respectively.

Full solution:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}} = \frac{\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot \frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

$$\frac{e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} = \frac{e^{z_{i,j}} \cdot (\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}})}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

$$\frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{\sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot (\frac{\sum_{l=1}^{L} e^{z_{i,l}}}{\sum_{l=1}^{L} e^{z_{i,l}}} - \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}}) =$$

$$S_{i,j} \cdot (1 - S_{i,k})$$

Now we have to go back and solve the derivative in the case of *j≠k*. In this case, the "left" derivative of the original equation solves to *0* as the whole expression is treated as a constant:

$$= \frac{0 \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

The difference is that now the whole subtrahend solves to *0*, leaving us with just the minuend in the numerator:

$$= \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

Now, exactly like before, we can write the denominator as the multiplication of the values instead of using the power of *2*:

$$= \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

That lets us to split this fraction into 2 fractions, using the multiplication operation:

$$= -\frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} =$$

Now both fractions represent the Softmax function:

$$= -S_{i,j} \cdot S_{i,k}$$

Note that the left Softmax function carries the $_j$ parameter, and the "right" one has $_k$ — both came from their numerators, respectively.

Full solution:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}} = \frac{\frac{\partial}{\partial z_{i,k}} e^{z_{i,j}} \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot \frac{\partial}{\partial z_{i,k}} \sum_{l=1}^{L} e^{z_{i,l}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} =$$

$$\frac{0 \cdot \sum_{l=1}^{L} e^{z_{i,l}} - e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} = \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{[\sum_{l=1}^{L} e^{z_{i,l}}]^2} = \frac{-e^{z_{i,j}} \cdot e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}} \cdot \sum_{l=1}^{L} e^{z_{i,l}}} =$$

$$-\frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \cdot \frac{e^{z_{i,k}}}{\sum_{l=1}^{L} e^{z_{i,l}}} = -S_{i,j} \cdot S_{i,k}$$

As a summary, the solution of the derivative of the Softmax function with respect to its inputs is:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \begin{cases} S_{i,j} \cdot (1 - S_{i,k}) & j = k \\ -S_{i,j} \cdot S_{i,k} & j \neq k \end{cases}$$

That's not the end of the calculation that we can perform here. When left in this form, we'll have 2 separate equations to code and use in different cases, which isn't very convenient for the speed of calculations. We can, however, further morph the result of the second case of the derivative:

$$-S_{i,j} \cdot S_{i,k} = S_{i,j} \cdot (-S_{i,k}) = S_{i,j} \cdot (0 - S_{i,k})$$

In the first step, we moved the second Softmax along the minus sign into the brackets so we can add a zero inside of them and right before this value. That does not change the solution, but now:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \begin{cases} S_{i,j} \cdot (1 - S_{i,k}) & j = k \\ S_{i,j} \cdot (0 - S_{i,k}) & j \neq k \end{cases}$$

Both solutions look very similar, they differ only in a single value. Conveniently, there exists **Kronecker delta** function (which we'll explain soon) whose equation is:

$$\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

We can apply it here, simplifying our equation further to:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = S_{i,j} \cdot (\delta_{j,k} - S_{i,k})$$

That's the final math solution to the derivative of the Softmax function's outputs with respect to each of its inputs. To make it a little bit easier to implement in Python using NumPy, let's transform the equation for the last time:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = S_{i,j} \cdot (\delta_{j,k} - S_{i,k}) = S_{i,j}\delta_{j,k} - S_{i,j}S_{i,k}$$

We basically multiplied S_{ij} by both sides of the subtraction operation from the parentheses.

Softmax activation derivative code implementation

This lets us code the solution using just two NumPy functions, which we'll explain now step by step:

Let's make up a single sample:

```
softmax_output = [0.7, 0.1, 0.2]
```

And shape it as a list of samples:

```
import numpy as np

softmax_output = [0.7, 0.1, 0.2]

softmax_output = np.array(softmax_output).reshape(-1, 1)
print(softmax_output)

>>>
array([[0.7],
       [0.1],
       [0.2]])
```

The left side of the equation is Softmax's output multiplied by the Kronecker delta. The Kronecker delta equals *1* when both inputs are equal, and *0* otherwise. If we visualize this as an array, we'll have an array of zeros with ones on the diagonal — you might remember that we already have implemented such a solution using the `np.eye` method:

```
print(np.eye(softmax_output.shape[0]))

>>>
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Now we'll do the multiplication of both of the values from the equation part:

```
print(softmax_output * np.eye(softmax_output.shape[0]))

>>>
array([[0.7, 0. , 0. ],
       [0. , 0.1, 0. ],
       [0. , 0. , 0.2]])
```

It turns out that we can gain some speed by replacing this by the `np.diagflat` method call, which computes the same solution — the diagflat method creates an array using an input vector as the diagonal:

```
print(np.diagflat(softmax_output))

>>>
array([[0.7, 0. , 0. ],
       [0. , 0.1, 0. ],
       [0. , 0. , 0.2]])
```

The other part of the equation is $S_{i,j}S_{i,k}$ — the multiplication of the Softmax outputs, iterating over the $_j$ and $_k$ indices respectively. Since, for each sample (the $_i$ index), we'll have to multiply the values from the Softmax function's output (in all of the combinations), we can use the dot product operation. For this, we'll just have to transpose the second argument to get its row vector form (as described in chapter 2):

```
print(np.dot(softmax_output, softmax_output.T))

>>>
array([[0.49, 0.07, 0.14],
       [0.07, 0.01, 0.02],
       [0.14, 0.02, 0.04]])
```

Finally, we can perform the subtraction of both arrays (following the equation):

```
print(np.diagflat(softmax_output) -
      np.dot(softmax_output, softmax_output.T))

>>>
array([[ 0.21, -0.07, -0.14],
       [-0.07,  0.09, -0.02],
       [-0.14, -0.02,  0.16]])
```

The matrix result of the equation and the array solution provided by the code is called the **Jacobian matrix**. In our case, the Jacobian matrix is an array of partial derivatives in all of the combinations of both input vectors. Remember, we are calculating the partial derivatives of every output of the Softmax function with respect to each input separately. We do this because each input influences each output due to the normalization process, which takes the sum of all the exponentiated inputs. The result of this operation, performed on a batch of samples, is a list of the Jacobian matrices, which effectively forms a 3D matrix — you can visualize it as a column whose levels are Jacobian matrices being the sample-wise gradient of the Softmax function.

This raises a question — if sample-wise gradients are the Jacobian matrices, how do we perform the chain rule with the gradient back-propagated from the loss function, since it's a vector for each sample? Also, what do we do with the fact that the previous layer, which is the Dense layer, will expect the gradients to be a 2D array? Currently, we have a 3D array of the partial derivatives — a list of the Jacobian matrices. The derivative of the Softmax function with respect to any of its inputs returns a vector of partial derivatives (a row from the Jacobian matrix), as this input influences all the outputs, thus also influencing the partial derivative for each of them. We need to sum the values from these vectors so that each of the inputs for each of the samples will return a single partial derivative value instead. Because each input influences all of the outputs, the returned vector of the partial derivatives has to be summed up for the final partial derivative with respect to this input. We can perform this operation on each of the Jacobian matrices directly, applying the chain rule at the same time (applying the gradient from the loss function) using `np.dot()` — For each sample, it'll take the row from the Jacobian matrix and multiply it by the corresponding value from the loss function's gradient. As a result, the dot product of each of these vectors and values will return a singular value, forming a vector of the partial derivatives sample-wise and a 2D array (a list of the resulting vectors) batch-wise.

Let's code the solution:

```
# Softmax activation
class Activation_Softmax:
    ...
    # Backward pass
    def backward(self, dvalues):

        # Create uninitialized array
        self.dinputs = np.empty_like(dvalues)

        # Enumerate outputs and gradients
        for index, (single_output, single_dvalues) in \
                enumerate(zip(self.output, dvalues)):
            # Flatten output array
            single_output = single_output.reshape(-1, 1)
            # Calculate Jacobian matrix of the output
            jacobian_matrix = np.diagflat(single_output) - \
                              np.dot(single_output, single_output.T)
            # Calculate sample-wise gradient
            # and add it to the array of sample gradients
            self.dinputs[index] = np.dot(jacobian_matrix,
                                         single_dvalues)
```

First, we created an empty array (which will become the resulting gradient array) with the same shape as the gradients that we're receiving to apply the chain rule. The `np.empty_like` method creates an empty and uninitialized array. Uninitialized means that we can expect it to contain meaningless values, but we'll set all of them shortly anyway, so there's no need for initialization (for example, with zeros using `np.zeros()` instead). In the next step, we're going to iterate sample-wise over pairs of the outputs and gradients, calculating the partial derivatives as described earlier and calculating the final product (applying the chain rule) of the Jacobian matrix and gradient vector (from the passed-in gradient array), storing the resulting vector as a row in the dinput array. We're going to store each vector in each row while iterating, forming the output array.

Common Categorical Cross-Entropy loss and Softmax activation derivative

At the moment, we have calculated the partial derivatives of the Categorical Cross-Entropy loss and Softmax activation functions, and we can finally use them, but there is still one more step that we can perform to speed the calculations up. Different books and tutorials usually mention the derivative of the loss function with respect to the Softmax inputs, or even weight and biases of the output layer directly and don't go into the details of the partial derivatives of these functions separately. This is partially because the derivatives of both functions combine to solve a simple equation — the whole code implementation is simpler and faster to execute. When we look at our current code, we perform multiple operations to calculate the gradients and even include a loop in the backward step of the activation function.

Let's apply the chain rule to calculate the partial derivative of the Categorical Cross-Entropy loss function with respect to the Softmax function inputs. First, let's define this derivative by applying the chain rule:

$$\frac{\partial L_i}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial S_{i,j}}{\partial z_{i,k}} =$$

This partial derivative equals the partial derivative of the loss function with respect to its inputs, multiplied (using the chain rule) by the partial derivative of the activation function with respect to its inputs. Now we need to systematize semantics — we know that the inputs to the loss function, *y-hat*$_{i,j}$, are the outputs of the activation function, $S_{i,j}$:

$$\hat{y}_{i,j} = S_{i,j}$$

That means that we can update the equation to the form of:

$$= \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

Now we can substitute the equation for the partial derivative of the Categorical Cross-Entropy function, but, since we are calculating the partial derivative with respect to the Softmax inputs, we'll use the one containing the sum operator over all of the outputs — it will soon become clear why. The derivative:

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}}$$

After substitution to the combined derivative's equation:

$$= -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

Now, as we calculated before, the partial derivative of the Softmax activation, before applying Kronecker delta to it:

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \begin{cases} S_{i,j} \cdot (1 - S_{i,k}) & j = k \\ -S_{i,j} \cdot S_{i,k} & j \neq k \end{cases}$$

Let's actually do the substitution of the S_{ij} with *y-hat*$_{ij}$ here as well:

$$\frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = \begin{cases} \hat{y}_{i,j} \cdot (1 - \hat{y}_{i,k}) & j = k \\ -\hat{y}_{i,j} \cdot \hat{y}_{i,k} & j \neq k \end{cases}$$

The solution is different depending on if $j=k$ or $j\neq k$. To handle for this situation, we have to split the current partial derivative following these cases — when they both match and when they do not:

$$-\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}} - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}}$$

For the $j\neq k$ case, we just updated the sum operator to exclude k and that's the only change:

$$-\sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}}$$

For the $j=k$ case, we do not need the sum operator as it will sum only one element, of index k. For

the same reason, we also replace *j*indices with *k*:

$$-\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}}$$

Back to the main equation:

$$= -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}} - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

Now we can substitute the partial derivatives of the activation function for both cases with the newly-defined solutions:

$$= -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \hat{y}_{i,k} \cdot (1 - \hat{y}_{i,k}) - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} (-\hat{y}_{i,j} \hat{y}_{i,k}) =$$

We can cancel out the *y-hat*$_{i,k}$ from both sides of the subtraction in the equation — both contain it as part of the multiplication operations and in their denominators. Then on the "right" side of the equation, we can replace 2 minus signs with the plus one and remove the parentheses:

$$= -y_{i,k} \cdot (1 - \hat{y}_{i,k}) + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} =$$

Now let's multiply the -$y_{i,k}$ with the content of the parentheses on the "left" side of the equation:

$$= -y_{i,k} + y_{i,k} \hat{y}_{i,k} + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} =$$

Now let's look at the sum operation — it adds up $y_{i,j}$*y-hat*$_{i,k}$ over all possible values of index *j* except for when it equals *k*. Then, on the left of this part of the equation, we have $y_{i,k}$*y-hat*$_{i,k}$, which contains $y_{i,k}$ — the exact element that is excluded from the sum. We can then join both expressions:

$$= -y_{i,k} + \sum_{j} y_{i,j} \hat{y}_{i,k} =$$

Now the sum operator iterates over all of the possible values of *j* and, since we know that $y_{i,j}$ for each *i* is the one-hot encoded vector of ground-truth values, the sum of all of its elements equals *1*. In other words, following the earlier explanation in this chapter — this sum will multiply *0* by the *y-hat*$_{i,k}$ except for a single situation, the true label, where it'll multiply *1* by this value. We can then simplify it further to:

$$= -y_{i,k} + \hat{y}_{i,k} = \hat{y}_{i,k} - y_{i,k}$$

Full solution:

$$\frac{\partial L_i}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial L_i}{\partial \hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} =$$

$$= -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \frac{\partial \hat{y}_{i,k}}{\partial z_{i,k}} - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} \cdot \frac{\partial \hat{y}_{i,j}}{\partial z_{i,k}} = -\frac{y_{i,k}}{\hat{y}_{i,k}} \cdot \hat{y}_{i,k} \cdot (1 - \hat{y}_{i,k}) - \sum_{j \neq k} \frac{y_{i,j}}{\hat{y}_{i,j}} (-\hat{y}_{i,j} \hat{y}_{i,k}) =$$

$$= -y_{i,k} \cdot (1 - \hat{y}_{i,k}) + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} = -y_{i,k} + y_{i,k} \hat{y}_{i,k} + \sum_{j \neq k} y_{i,j} \hat{y}_{i,k} =$$

$$= -y_{i,k} + \sum_j y_{i,j} \hat{y}_{i,k} = -y_{i,k} + \hat{y}_{i,k} = \hat{y}_{i,k} - y_{i,k}$$

As we can see, when we apply the chain rule to both partial derivatives, the whole equation simplifies significantly to the subtraction of the predicted and ground truth values. It is also multiple times faster to compute.