1

1.a Geometric interpretation of gradient descent

Let C be the cost function we are aiming to minimize. Then the update rule for gradient descent takes the form

$$v \to v' = v - \eta \nabla C = v - \varepsilon \frac{\nabla C}{||\nabla C||}.$$

For the one-dimensional case this simplifies to

$$v \to v' = v - \varepsilon \cdot \mathrm{sgn}(C'(v)).$$

Geomtrically speaking, this implies we are always going ε to the left or the right. The direction depends on whether the cost function is increasing or decreasing at our current location.

1.b b

The use of mini-batches is justified by the following heuristic:

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C, \tag{1}$$

where n is the full size of the training data set and m is the size of the mini-batch. Let's compare $m = 1$ to $m = 20$. The validity of approximation (3) depends on m, with a higher m implying a better approximation (there are established concentration-inequalities to support this claim). For $m = 1$ the estimate of the gradient will frequently be worse than for $m = 20$. Consequently, each step with $m = 1$ will generally not decrease the cost as much as $m = 20$. In worst cases, we might have a terrible approximation and might increase the cost.

On the other hand, evaluating $m = 1$ is faster than $m = 20$, so we will be able to do more steps with the same time and resources.

2 Problem

2.a

Following the notation in the Nelson Book consider (BP2):

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$$

In component form, this is

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'\left(z_j^l\right).$$

If we replace the activation function at a single neuron by f, this becomes for a fixed l, j

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} f' \left(z_j^l \right).$$

Accordingly, $\delta^1, \ldots, \delta^{l-1}$, which depend on δ_j^l, will change. By (BP3) and (BP4), the step estimate for b_j^l, the biases and weights in all the layers before the change will have to be adjusted.

2.b

Let f the softmax activation function i.e.

$$a_j^L = f_j(z_1^L, \ldots) = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

and C be the log-likelihood cost

$$C \equiv -\ln a_j^L$$

Then we have for the cost in the outputlayer:

$$\begin{aligned} \delta_j^L &= \frac{\partial C}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \frac{\partial f_j(z_1^L, \ldots)}{\partial z_j^L} \\ &= -\frac{1}{a_j^L} \frac{\partial f_j(z_1^L, \ldots)}{\partial z_j^L}. \end{aligned}$$

https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1

do

$$\delta_j^L = a_j^L - y_j$$

2.c

If we replace the sigmoid layer with a linear identity layer, the neural network simplifies to a single big matrix multiplication from the input layer to the output layer. We will always have $\sigma' \left(z^L \right) = 1$. Accordingly, we have:

$$
\begin{aligned}
\delta^L &= \nabla_a C \\
\delta^l &= \left(w^{l+1}\right)^T \delta^{l+1} = \prod_{i=l+1}^{L-1} \left(w^i\right)^T \nabla_a C \\
\frac{\partial C}{\partial b_j^l} &= \delta_j^l = \left(w_{j\cdot}^{l+1}\right)^T \prod_{i=l+2}^{L-1} \left(w^i\right)^T \nabla_a C \\
\frac{\partial C}{\partial w_{jk}^l} &= a_k^{l-1} \delta_j^l = a_k^{l-1} \left(w_{j\cdot}^{l+1}\right)^T \prod_{i=l+2}^{L-1} \left(w^i\right)^T \nabla_a C
\end{aligned}
\tag{2}
$$

The upgrade step of the gradient descent will be governed by the last two lines.

3 Problem

3.a Problem

Consider

$$- \left[y \ln a + (1 - y) \ln(1 - a)\right], \tag{3}$$

and

$$- \left[a \ln y + (1 - a) \ln(1 - y)\right]. \tag{4}$$

We know that $a = \sigma(z) \in (0, 1)$, because σ only takes values in $(0, 1)$. On the other hand, $y \in [0, 1]$. We will use $a \in (0, 1)$ in the following. For $y = 0$ we have in equation (3):

$$-[y \ln a + (1 - y) \ln(1 - a)] = \ln(1 - a).$$

This makes sense. For $y = 1$ we have in equation (3):

$$-[y \ln a + (1 - y) \ln(1 - a)] = \ln a.$$

This makes sense. For $y = 0$ we have in equation (4):

$$-[a \ln y + (1 - a) \ln(1 - y)] = -[a \cdot (-\infty) + (1 - a) \cdot 0] = \infty.$$

This makes no sense. For $y = 1$ we have in equation (4):

$$-[a \ln y + (1 - a) \ln(1 - y)] = -[a \cdot (0) + (1 - a) \cdot (-\infty)] = \infty.$$

This makes no sense. Those problems do not arise in equation (3), because $a \in (0, 1)$.

3.b Problem

Let σ be the sigmoid function and C the cross-entropy cost. By equations (3.7) and (3.8) in the Nielson book we have:

$$\begin{aligned}
\frac{\partial C}{\partial w_j} &= \frac{1}{n}\sum_x x_j(\sigma(z) - y), \\
\frac{\partial C}{\partial b} &= \frac{1}{n}\sum_x (\sigma(z) - y).
\end{aligned}$$

Note, that the derivation for those equations in the Nielson book is valid for general $y, a \in [0,1]$. Consequently, if we have $\sigma(z) = y$ for all training inputs, we have $\frac{\partial C}{\partial w_j} = \frac{\partial C}{\partial b} = 0$. This shows that $\sigma(z) = y$ is a critical point of the cost function. It remains to verify that it is a minimum. Note, that

$$\begin{aligned}
\frac{\partial C}{\partial w_j \partial w_k} &= \frac{\partial C}{\partial w_j \partial z}\frac{\partial z}{\partial w_k} \\
&= \left(\frac{1}{n}\sum_x x_j \sigma'(z)\right)(a_k) > 0.
\end{aligned}$$

$$\begin{aligned}
\frac{\partial C}{\partial b \partial z} &= \frac{\partial C}{\partial b \partial z}\frac{\partial z}{\partial b} \\
&= \left(\frac{1}{n}\sum_x \sigma'(z)\right) > 0.
\end{aligned}$$

This concludes the proof. Furthermore, plugging in $\sigma(z) = y$ yields

$$C = -\frac{1}{n}\sum_x [y \ln y + (1-y)\ln(1-y)].$$

3.c Problem

Let

$$W^1 = \begin{bmatrix} 0.15 & 0.25 \\ 0.2 & 0.3 \end{bmatrix} = \begin{bmatrix} w^1_{1,1} & w^1_{2,1} \\ w^1_{1,2} & w^1_{2,2} \end{bmatrix},$$

looks wrong

$$b^1 = \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix} = \begin{bmatrix} b^1_1 \\ b^1_2 \end{bmatrix},$$

$$W^2 = \begin{bmatrix} 0.4 & \\ & 0.55 \end{bmatrix} = \begin{bmatrix} w^2_{1,1} & w^2_{2,1} \\ w^2_{1,2} & w^2_{2,2} \end{bmatrix},$$

$$b^2 = \begin{bmatrix} 0.6 \\ 0.6 \end{bmatrix} = \begin{bmatrix} b^2_1 \\ b^2_2 \end{bmatrix}.$$

Then the output of the neural network for an input x is

$$\sigma(W^2(\sigma(W^1 x + b^1)) + b^2).$$

We can calculate the $\delta^1, \delta^2, \delta^3$ with the central equations of the back-propagation algorithm:

$$\begin{aligned} \delta^L &= \nabla_a C \odot \sigma'\left(z^L\right) \\ \delta^l &= \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \end{aligned}$$

For the cross entropy function C we have