Assignment 1
MLPs, CNNs and Backpropagation

Francesco Dal Canton
f.dal.canton@student.uva.nl
12404225

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

1.1 a)

i. The gradient of the cross entropy loss w.r.t. the output of the softmax is:

$$\begin{aligned}\frac{\partial L}{\partial x^{(N)}} &= \frac{\partial\left[-\sum_i t_i \log x_i^{(N)}\right]}{\partial x^{(N)}}\\ &= -\frac{t_i}{x_i^{(N)}}\end{aligned}$$

So that the resulting gradient $\in \mathbb{R}^{1\times d_N}$.

ii. The gradient of the softmax function w.r.t. its input is:

$$\begin{aligned}\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} &= \frac{\partial\left[\frac{\exp \tilde{x}^{(N)}}{\sum_{i=1}^{d_N}\exp \tilde{x}_i^{(N)}}\right]}{\partial \tilde{x}^{(N)}}\\ &= \frac{\partial \sigma(\tilde{x}^{(N)})}{\partial \tilde{x}^{(N)}}\end{aligned}$$

We can express the result by specifying the gradient for each combination of i and j and disambiguating depending on whether $i = j$ or not:

$$\frac{\partial \sigma(\tilde{x}_i^{(N)})}{\partial \tilde{x}_j^{(N)}} = \begin{cases}\sigma(\tilde{x}_i^{(N)})\left[1-\sigma(\tilde{x}_j^{(N)})\right] & \text{if } i = j\\ -\sigma(\tilde{x}_i^{(N)})\sigma(\tilde{x}_j^{(N)}) & \text{if } i \neq j\end{cases}$$

So that the resulting gradient $\in \mathbb{R}^{d_N\times d_N}$.

iii. The gradient of the ReLU function w.r.t. its input is:

$$\frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} = \frac{\partial\left[\max(0, \tilde{x}^{(l)})\right]}{\partial \tilde{x}^{(l)}}$$

Again, we can better express the result by specifying the gradient for each i. Moreover, since the ReLU function acts on each element independently, we have a gradient of 0 for all cases where we're deriving $x_i^{(l)}$ w.r.t. $\tilde{x}_j^{(l)}$ with $i \neq j$.

$$\frac{\partial \left[\max(0, \tilde{x}_i^{(l)})\right]}{\partial \tilde{x}_i^{(l)}} = \begin{cases} 0 & \text{if } \tilde{x}_i^{(l)} < 0 \\ 1 & \text{if } \tilde{x}_i^{(l)} > 0 \end{cases}$$

Note that the gradient is undefined for $\tilde{x}_i^{(l)} = 0$, but in the NumPy implementation we'll consider it to be 0.

The resulting gradient is $\in \mathbb{R}^{d_l \times d_l}$.

iv. The gradient of the feedforward layer w.r.t. its input is given by:

$$\begin{aligned} \frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} &= \frac{\partial \left[W^{(l)} x^{(l-1)} + b^{(l)}\right]}{\partial x^{(l-1)}} \\ &= W^{(l)} \end{aligned}$$

So that the resulting gradient is $\in \mathbb{R}^{d_l \times d_{l-1}}$.

v. The gradient of the feedforward layer w.r.t. its weight matrix is given by:

$$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \frac{\partial \left[W^{(l)} x^{(l-1)} + b^{(l)}\right]}{\partial W^{(l)}}$$

We know that the resulting gradient is $\in \mathbb{R}^{d_l \times (d_l \times d_{l-1})}$, but it's easier to express it for each element, disambiguating the various cases:

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}} = \frac{\partial \left[\sum_m^{d_{l-1}} W_{jm}^{(l)} x_m^{(l-1)} + b_j^{(l)}\right]}{\partial W_{jk}^{(l)}}$$

Since the formula doesn't involve multiplying W_{jk} and x_i unless $i = k$, we can express the gradient as follows:

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}} = \begin{cases} x_i^{(l-1)} & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}$$

vi. The gradient of the feedforward layer w.r.t. its bias is given by:

$$\begin{aligned} \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} &= \frac{\partial \left[W^{(l)} x^{(l-1)} + b^{(l)}\right]}{\partial b^{(l)}} \\ &= I \end{aligned}$$

Where I is the identity matrix. Similarly to the ReLU, the bias is applied independently to each element of the vector $W^{(l)} x^{(l-1)}$, meaning that the gradient is 0 for each value that is not along the diagonal. This means the resulting gradient is $\in \mathbb{R}^{d_l \times d_l}$.

1.1 b)

i. The gradient of the cross entropy function w.r.t. the softmax function's input is given by:

$$\frac{\partial L}{\partial \tilde{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}$$

Since the first gradient in the product is $\in \mathbb{R}^{1 \times d_N}$ and the second is $\in \mathbb{R}^{d_n \times d_N}$, their product must be $\in \mathbb{R}^{1 \times d_N}$. Given the two values the gradient of the softmax can take, we can express the gradient for each i as:

$$\frac{\partial L}{\partial \tilde{x}_i^{(N)}} = -\frac{t_i}{x_i^{(N)}} \sigma(\tilde{x}_i^{(N)}) \left[1 - \sigma(\tilde{x}_i^{(N)})\right] + \sum_{i \neq j}^{d_N} \frac{t_i}{x_i^{(N)}} \sigma(\tilde{x}_i^{(N)}) \sigma(\tilde{x}_j^{(N)})$$

ii. The gradient of the cross entropy function w.r.t. the output of a feedforward layer is:

$$\begin{aligned}\frac{\partial L}{\partial \tilde{x}^{(l)}} &= \frac{\partial L}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial \tilde{x}^{(l)}} \\ &= \frac{\partial L}{\partial x^{(l)}} \text{diag}(y)\end{aligned}$$

Where $\text{diag}(y)$ indicates a diagonal matrix where the values along the diagonal are given by vector y. Each value y_i in vector $y \in \mathbb{R}^{d_l}$ is given by:

$$y_i = \begin{cases} 1 & \text{if } \tilde{x}^{(l)} > 0 \\ 0 & \text{if } \tilde{x}^{(l)} < 0 \end{cases}$$

Although in practice, $y_i = 0$ if $\tilde{x}^{(l)} = 0$.
Given the resulting gradient is a product of a vector $\in \mathbb{R}^{1 \times d_l}$ and a matrix $\in \mathbb{R}^{d_l \times d_l}$, the product is $\in \mathbb{R}^{1 \times d_l}$.

iii. The gradient of the cross entropy function w.r.t. the input to a feedforward layer is:

$$\begin{aligned}\frac{\partial L}{\partial x^{(l)}} &= \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{x}^{(l+1)}} W^{(l+1)}\end{aligned}$$

Since the first term of the product is $\in \mathbb{R}^{1 \times d_{l+1}}$ and the second is $\in \mathbb{R}^{d_{l+1} \times d_l}$, the resulting gradient is $\in \mathbb{R}^{1 \times d_l}$.

iv. The gradient of the cross entropy function w.r.t. the weight matrix of a feedforward layer is:

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}$$

Where the gradient on the right hand side has been defined above in terms of its elements as:

$$\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}} = \begin{cases} x_i^{(l-1)} & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}$$

Since the first element in the product is $\in \mathbb{R}^{1 \times d_l}$ and the second is $\in \mathbb{R}^{d_l \times (d_l \times d_{l-1})}$, the resulting gradient is $\in \mathbb{R}^{1 \times (d_l \times d_{l-1})}$. In practice, the first dimension is ignored when performing the update to the weight matrix.

v. The gradient of the cross entropy function w.r.t. the bias of a feedforward layer is:

$$\begin{aligned}\frac{\partial L}{\partial b^{(l)}} &= \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} \\ &= \frac{\partial L}{\partial \tilde{x}^{(l)}} I\end{aligned}$$

Since the first element in the product is $\in \mathbb{R}^{1 \times d_l}$ and the second is $\in \mathbb{R}^{d_l \times d_l}$, the resulting gradient is $\in \mathbb{R}^{1 \times d_l}$.

1.1 c)

Using a batch size $B \neq 1$ means that instead of using inputs and outputs to modules $\in \mathbb{R}^{d \times 1}$, we're using matrices $\in \mathbb{R}^{d \times B}$. This, in turn, adds the dimension B to each gradient calculation used in the backpropagation.

In practice, since the final update for a batch is computed as the average of the updates for each of the inputs in the batch, the extra dimension B disappears when performing the actual update.

1.2 NumPy implementation

2 PyTorch MLP

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

3.2 Manual implementation of backward pass

3.2 a)

3.2 b)

3.2 c)

4 PyTorch CNN