Technical Details

Architecture Overview

Everything is implemented in an object-oriented manner. The architecture of the project is the following:

NeuralNetwork is the backbone of the project. It is assigned an optimizer and a loss function and stores the layers of the network. It contains functions like `forward()`, `backward()`, `infer()` and `train()`.

FullyConnectedLayer is a subclass of **Layer**, and as its name suggests, it implements a fully connected layer, with the functions `forward()`, `backward()` and `infer()`.

ActivationFunction is the supertype of **Relu**, **Sigmoid** and **Identity**. Its interface simply consists of the two functions `apply()` and `derivative()`.

LossFunction is the supertype of **MeanSquaredError** and **CrossEntropy**. Its interface consists of the two functions `compute()` and `derivative()`.

Optimizer is the supertype of **AdamOptimizer** and **VanillaSGDOptimizer**. The main function they implement is `update_gradients()`.

More details can be found in the source code comments.

Forwarding

Let $a^{(L)} \in \mathbb{R}^n$ be the input of layer L. It first gets normalized as follows:

$$\hat{a}^{(L)} = \frac{a^{(L)} - \mu^{(L)}}{\sqrt{{\sigma^{(L)}}^2 + \varepsilon}}$$

where $\mu^{(L)}, {\sigma^{(L)}}^2 \in \mathbb{R}^n$ are the mean resp. the variance of all inputs that this layer receives, and $\varepsilon \in \mathbb{R}^n$ is a vector of small values ensuring numerical stability (we use an element-wise division operator here).

We then scale and shift this normalized vector by the parameters $\gamma^{(L)} \in \mathbb{R}^n$ and $\beta^{(L)} \in \mathbb{R}^n$ of layer L:

$$\bar{a}^{(L)} = \hat{a}^{(L)} \odot \gamma^{(L)} + \beta^{(L)}$$

where $\odot$ denotes the element-wise multiplication.

Next, we multiply $\bar{a}^{(L)}$ by the weights and add the bias:

$$z^{(L)} = W^{(L)}\bar{a}^{(L)} + b^{(L)}$$

where $W^{(L)} \in \mathbb{R}^{m \times n}$ and $b^{(L)} \in \mathbb{R}^m$ are the weights respectively the bias of layer L.

Finally, we apply the activation function $\sigma : \mathbb{R}^m \to \mathbb{R}^m$:

$$a^{(L)} = \sigma(z^{(L)})$$

Note: Each layer keeps a "running mean" and a "running variance", that get updated as follows every time data is forwarded and that are used for normalizing new data when doing inference.

$$\begin{aligned}\mu^{(L)}_{\text{running}} &= \text{momentum} \cdot \mu^{(L)}_{\text{running}} + (1 - \text{momentum})\mu^{(L)} \\ \sigma^{(L)}_{\text{running}} &= \text{momentum} \cdot \sigma^{(L)}_{\text{running}} + (1 - \text{momentum})\sigma^{(L)}\end{aligned}$$

where $\text{momentum} \in (0, 1)$.

Backpropagation

Let $\mathcal{L}$ be the loss function. We use the notation dy instead of $\frac{\partial \mathcal{L}}{\partial y}$. Provided we receive the gradient from the next layer, $da^{(L+1)}$, we propagate the gradients through layer L as follows:

$$\begin{aligned}
dz^{(L)} &= da^{(L+1)} \odot \sigma'(z^{(L)}) \\
dW^{(L)} &= dz^{(L)} \bar{a}^{(L)^T} \\
db^{(L)} &= dz^{(L)} \\
d\bar{a}^{(L)} &= dW^{(L)^T} dz^{(L)} \\
d\gamma^{(L)} &= d\bar{a}^{(L)} \odot \hat{a}^{(L)} \\
d\beta^{(L)} &= d\bar{a}^{(L)} \\
d\hat{a}^{(L)} &= d\bar{a}^{(L)} \odot \gamma^{(L)} \\
da^{(L)} &= d\hat{a}^{(L)} \odot \frac{1}{\sqrt{\sigma^{(L)^2} + \varepsilon}}
\end{aligned}$$

Derivative of the MSE loss

The derivative of the MSE loss is straightforward. The mean squared error (MSE) loss for a batch of m samples is defined as:

$$\mathrm{MSE}(\mathbf{Y}_{\text{true}}, \mathbf{Y}_{\text{pred}}) = \frac{1}{m} \sum_{i=1}^{m} \| y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)} \|^2$$

where $y_{\text{true}}^{(i)}, y_{\text{pred}}^{(i)} \in \mathbb{R}^n$ are the true and predicted vectors for the i-th sample in the batch.

Taking the derivative with respect to one sample gives:

$$\frac{\partial \mathrm{MSE}}{\partial y_{\text{pred}}^{(i)}} = \frac{2}{m} \left(y_{\text{pred}}^{(i)} - y_{\text{true}}^{(i)} \right)$$

Therefore, the gradient for the full batch is the matrix:

$$\frac{\partial \mathrm{MSE}}{\partial \mathbf{Y}_{\text{pred}}} = \frac{2}{m} \left(\mathbf{Y}_{\text{pred}} - \mathbf{Y}_{\text{true}} \right)$$

Derivative of the cross-entropy loss

The cross-entropy loss for a batch of m samples is defined as:

$$\mathrm{CELoss}(\mathbf{Z}, \mathbf{Y}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} y_j^{(i)} \log \left(\frac{e^{z_j^{(i)}}}{\sum_{k=1}^{n} e^{z_k^{(i)}}} \right) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} y_j^{(i)} \left(-z_j^{(i)} + \log \left(\sum_{k=1}^{n} e^{z_k^{(i)}} \right) \right)$$

where $z^{(i)}, y^{(i)} \in \mathbb{R}^n$ are the logits and one-hot encoded labels for the i-th sample in the batch. The first derivation is here for numerical stability, since it avoids dividing by very small numbers or taking the logarithm of values close to 0. To further enhance the computational stability, we leverage the fact that:

$$\log \left(\sum_{k=1}^{n} e^{z_k^{(i)}} \right) = \log \left(e^{z_{\max}^{(i)}} \sum_{k=1}^{n} e^{z_k^{(i)} - z_{\max}^{(i)}} \right) = z_{\max}^{(i)} + \log \left(\sum_{k=1}^{n} e^{z_k^{(i)} - z_{\max}^{(i)}} \right)$$

where $z_{\max}^{(i)}$ is the largest logit value for sample i, because now, we reduce the risk of computing the exponential of a large number.

We now have:

$$\mathrm{CELoss}(\mathbf{Z},\mathbf{Y}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} y_j^{(i)}\left(-z_j^{(i)} + z_{\max}^{(i)} + \log\left(\sum_{k=1}^{n} e^{z_k^{(i)} - z_{\max}^{(i)}}\right)\right)$$

This formula is the one that we use in the code when computing the cross-entropy loss of our network on some dataset.

For the i-th sample, when taking the derivative with respect to the j-th logit, $z_j^{(i)}$, we have the following two cases.

Case $y_j^{(i)} = 1$:

$$\begin{aligned}
&\frac{\partial}{\partial z_j^{(i)}}\sum_{j'=1}^{n} y_{j'}^{(i)}\left(-z_{j'}^{(i)} + \log\left(\sum_{k=1}^{n} e^{z_k^{(i)}}\right)\right)\\
&= \frac{\partial}{\partial z_j^{(i)}}\left(-z_j^{(i)} + \log\left(\sum_{k=1}^{n} e^{z_k^{(i)}}\right)\right)\\
&= -1 + \frac{e^{z_j^{(i)}}}{\sum_{k=1}^{n} e^{z_k^{(i)}}}
\end{aligned}$$

Case $y_l^{(i)} = 1, l \neq j$:

$$\begin{aligned}
&\frac{\partial}{\partial z_j^{(i)}}\sum_{j'=1}^{n} y_{j'}^{(i)}\left(-z_{j'}^{(i)} + \log\left(\sum_{k=1}^{n} e^{z_k^{(i)}}\right)\right)\\
&= \frac{\partial}{\partial z_j^{(i)}}\left(-z_l^{(i)} + \log\left(\sum_{k=1}^{n} e^{z_k^{(i)}}\right)\right)\\
&= \frac{e^{z_j^{(i)}}}{\sum_{k=1}^{n} e^{z_k^{(i)}}}
\end{aligned}$$

Therefore, for the final derivative of the cross-entropy loss function of the i-th sample with respect to $z_j^{(i)}$, we get:

$$\frac{\partial \mathrm{CELoss}(\mathbf{Z},\mathbf{Y})}{\partial z_j^{(i)}} = \frac{1}{m}\left(\frac{e^{z_j^{(i)}}}{\sum_{k=1}^{n} e^{z_k^{(i)}}} - y_j^{(i)}\right)$$

For better numerical stability, we use the same trick as before once again and we get the following formula, that is actually implemented in the code.

$$\frac{\partial \mathrm{CELoss}(\mathbf{Z},\mathbf{Y})}{\partial z_j^{(i)}} = \frac{1}{m}\left(\frac{e^{z_j^{(i)} - z_{\max}^{(i)}}}{\sum_{k=1}^{n} e^{z_k^{(i)} - z_{\max}^{(i)}}} - y_j^{(i)}\right)$$