Deep Learning 1 - Homework 1

Pedro M.P. Curvo
MSc Artificial Intelligence
University of Amsterdam
pedro.pombeiro.curvo@student.uva.nl

Question 1

a) b) c)

The linear model described previously can be written as:

$$Y = XW^T + B \tag{1}$$

Where Y is the output, X is the input, W is the weight matrix and B is the bias, with $Y \in \mathbb{R}^{S \times N}$, $X \in \mathbb{R}^{S \times M}$, $W \in \mathbb{R}^{N \times M}$ and $B \in \mathbb{R}^{1 \times N}$.

In order to compute $\frac{\partial L}{\partial W}$, we can use the chain rule:

$$\begin{aligned}
\frac{\partial L}{\partial W} &= \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial W} \\
&= \frac{\partial L}{\partial Y}\frac{\partial (XW^T + B)}{\partial W} \quad \text{(from equation 1)} \\
&= \frac{\partial L}{\partial Y}\frac{\partial (XW^T)}{\partial W} \quad \text{(since B does not depend on W)} \\
&= \frac{\partial L}{\partial Y}^T X
\end{aligned}$$

Similarly, to compute $\frac{\partial L}{\partial B}$, we can use the chain rule:

$$\begin{aligned}
\frac{\partial L}{\partial B} &= \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial B} \\
&= \frac{\partial L}{\partial Y}\frac{\partial (XW^T + B)}{\partial B} \quad \text{(from equation 1)} \\
&= \frac{\partial L}{\partial Y}\frac{\partial B}{\partial B} \quad \text{(since B does not depend on W)} \\
&= \sum_{i=1}^{S} \frac{\partial L}{\partial Y_i}
\end{aligned}$$

Finally, to compute $\frac{\partial L}{\partial X}$, we can use the chain rule:

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

$$
\begin{aligned}
\frac{\partial L}{\partial X} &= \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial X} \\
&= \frac{\partial L}{\partial Y}\frac{\partial (XW^T + B)}{\partial X} \quad \text{(from equation 1)} \\
&= \frac{\partial L}{\partial Y}\frac{\partial (XW^T)}{\partial X} \quad \text{(since B does not depend on W)} \\
&= \frac{\partial L}{\partial Y}W
\end{aligned}
$$

d)

Considering the element-wise activation h, given by:

$$Y = h(X) \Rightarrow Y_{ij} = h(X_{ij})$$

By applying the chain rule, we can compute $\frac{\partial L}{\partial X}$ as follows:

$$
\begin{aligned}
\frac{\partial L}{\partial X_{ij}} &= \frac{\partial L}{\partial Y_{ij}}\frac{\partial Y_{ij}}{\partial X_{ij}} \\
&= \frac{\partial L}{\partial Y_{ij}}\frac{\partial h(X_{ij})}{\partial X_{ij}}
\end{aligned}
$$

Which can have a simple notation of:

$$\frac{\partial L}{\partial X_{ij}} = \frac{\partial L}{\partial Y_{ij}} h'(X_{ij})$$

where $h'(X)$ is the derivative of the activation function h with respect to their input.

Now, this rule is applied to all elements of the matrix X, resulting in the following:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \odot h'(X)$$

where $\odot$ is the Hadamard product and h' is applied element-wise to the matrix X.

As we can see, since we are applying the Hadamard product on a derivative that is also applied element-wise, the result will have the same dimensions as the input matrix X. Hence, $\frac{\partial L}{\partial X} \in \mathbb{R}^{S \times M}$.

e)

As presented, the gradients can be given by:

$$
\begin{aligned}
\frac{\partial L}{\partial Z} = Y \odot \left(\frac{\partial L}{\partial Y} - \left(\frac{\partial L}{\partial Y} \odot Y\right) 11^T\right) \\
\frac{\partial L}{\partial Y} = -\frac{1}{S}\left(\frac{T}{Y}\right)
\end{aligned}
$$

First, we began by replacing the expression for $\frac{\partial L}{\partial Y}$ in the expression for $\frac{\partial L}{\partial Z}$:

$$\frac{\partial L}{\partial Z} = Y \odot \left(-\frac{1}{S} \left(\frac{T}{Y} \right) - \left(-\frac{1}{S} \left(\frac{T}{Y} \right) \odot Y \right) 11^T \right)$$

Now, since $-\frac{1}{S}$ is a scalar, we can take it out of the Hadamard product:

$$\frac{\partial L}{\partial Z} = -\frac{1}{S} Y \odot \left(\frac{T}{Y} - \left(\frac{T}{Y} \odot Y \right) 11^T \right)$$

Now, we can use the distributive property of the Hadamard product to simplify the expression:

$$\frac{\partial L}{\partial Z} = -\frac{1}{S} \left(Y \odot \frac{T}{Y} - Y \odot \left(\left(\frac{T}{Y} \odot Y \right) 11^T \right) \right)$$

Now, since the division is element-wise, we can cancel out the Hadamard product with the division:

$$\frac{\partial L}{\partial Z} = -\frac{1}{S} \left(T - Y \odot \left((T) \, 11^T \right) \right)$$

Now, we need to look at the matrix product of $T11^T$. 11^T gives us a matrix of ones, with dimensions $C \times C$. Naming the result as H, we have that:

$$H_{ij} = \sum_{k=1}^{C} T_{ik} 1 = \sum_{k=1}^{C} T_{ik}$$

But since T is a one-hot encoded matrix, we have that $\sum_j T_{ij} = 1$. Hence, $H_{ij} = 1$ for all i and j.

Now, we can simplify the expression for $\frac{\partial L}{\partial Z}$:

$$\begin{aligned} \frac{\partial L}{\partial Z} &= -\frac{1}{S} (T - Y \odot H) \\ &= -\frac{1}{S} (T - Y) \\ &= \frac{1}{S} (Y - T) \end{aligned}$$

With that, we can infer that:

$$\begin{aligned} \alpha &= \frac{1}{S} \\ M &= Y - T \end{aligned}$$

Since S is the number of samples, than $\alpha \in \mathbb{R}^+$ as we wanted to show.

Question 2

TODO: Insert Image

Question 3

TODO: Insert Image

Question 4

a)

To show that the eingevalues for the Hessian matrix in a strictly local minimum are all possitive, we need to began by showing that the Hessian matrix is positive-definite at that point.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function that is twice differentiable at a point x^*, hence $f \in C^2$. Assuming that x^* is a local minimum, we have that $\nabla f(x^*) = 0$, since if x^* is a local extremum, the gradient must be zero.

If we know expand the Taylor series of f around x^*, we have that:

$$\begin{aligned} f(x + \lambda d) &= f(x) + \lambda \nabla f(x)^T d + \frac{1}{2}\lambda^2 d^T H_f(x) d + o(\lambda^2 d^T d) \\ &= f(x) + \frac{1}{2}\lambda^2 d^T H_f(x) d + o(\lambda^2 d^T d) \end{aligned}$$

Because x^* is a stricly local minimum, we have that, by definition:

$$\begin{aligned} 0 &< \lim_{\lambda \to 0} \frac{f(x^* + \lambda d) - f(x^*)}{\lambda^2} \\ &= \lim_{\lambda \to 0} \frac{f(x^*) + \frac{1}{2}\lambda^2 d^T H_f(x^*) d + o(\lambda^2 d^T d) - f(x^*)}{\lambda^2} \\ &= \frac{1}{2} d^T H_f(x^*) d \end{aligned}$$

This is true for all $d \neq 0$. By the definiton of positive-definite matrix:

$$d^T H_f(x^*) d > 0$$

Hence, the Hessian matrix at a strictly local minimum is positive-definite, by the properties above.

Now, to proof the statement, we need to show that all the eigenvalues of a positive-definite matrix are positive.

If we start by assuming that there is one eigenvalue λ that is negative, then for its corresponding eigenvector v:

$$\begin{aligned} v^T H v &= \lambda v^T v \\ &= \lambda ||v||^2 < 0 \quad \text{(since } \lambda < 0\text{) and } ||v||^2 > 0) \end{aligned}$$

This contradicts the definition of positive-definite matrix.

Now, if we assume that there is one eigenvalue λ that is zero, then for its corresponding eigenvector v:

$$\begin{aligned} v^T H v &= \lambda v^T v \\ &= 0 \quad \text{(since } \lambda = 0) \end{aligned}$$

Which also contradicts the definition of positive-definite matrix.

Hence, all the eigenvalues of a positive-definite matrix are positive. We can than conclude, that for a strictly local minimum, the Hessian matrix is positive-definite and all its eigenvalues are positive.

b)

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function that is twice differentiable at a point p. If p is a stricly local minimum, we have that $H_f(p)$ is positive-definite with all n eigenvalues positive.

Now, if think that the eigenvalues can assume two possible states, we have that the number of possible states is 2^n. With this, only one configuration is possible, which is the one where all the eigenvalues are positive, hence the probability of this configuration is $\frac{1}{2^n}$.

Now, for the saddle points, we have that the Hessian matrix is indefinite, with both positive and negative eigenvalues, Hence, we have $2^n - 2$ possible configurations (all the states minus the one with all positive eigenvalues and the one with all negative eigenvalues). With probability $\frac{2^n-2}{2^n}$.

This shows that the number of saddle points is exponentially larger than the number of local minima as we wanted to show.

c)

A saddle points p has the property of having its gradient equal to zero, $\nabla f(p) = 0$. Now, by taking the formula of gradient descent, we have that:

$$x^{\tau+1} = x^{\tau} - \eta \nabla f(x_t)$$

Since, $\nabla f(p) = 0$, we have that $x^{\tau+1} = x^{\tau}$. Hence, the algorithm will not move from the saddle point.

Question 5

a)

Taking the derivative of L with respect to γ_i:

$$\begin{aligned}
\frac{\partial L}{\partial \gamma_i} &= \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial \gamma_i} \\
&= \frac{\partial L}{\partial y_i} \frac{\partial}{\partial \gamma_i} (\gamma_i x_i + \beta_i) \\
&= \frac{\partial L}{\partial y_i} x_i
\end{aligned}$$

Since the batch norm is applied in a batch, we have that:

$$\frac{\partial L}{\partial \gamma_i} = \sum_{j=1}^{S} \frac{\partial L}{\partial y_i^{(j)}} x_i^{(j)}$$

Where S is the number of samples in the batch and $y_i^{(j)}$ is the output of the i-th neuron for the j-th sample and $x_i^{(j)}$ is the input of the i-th neuron for the j-th sample.

Taking the derivative of L with respect to β_i:

$$\begin{aligned}
\frac{\partial L}{\partial \beta_i} &= \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial \beta_i} \\
&= \frac{\partial L}{\partial y_i} \frac{\partial}{\partial \beta_i} (\gamma_i x_i + \beta_i) \\
&= \frac{\partial L}{\partial y_i}
\end{aligned}$$

Since the batch norm is applied in a batch, we have that:

$$\frac{\partial L}{\partial \beta_i} = \sum_{j=1}^{S} \frac{\partial L}{\partial y_i^{(j)}}$$

b)

Since we have a linear connected layer with input since 20 and output size 40, we have that $x_i \in \mathbb{R}^{20}$ and $y_i \in \mathbb{R}^{40}$. Hence, the weight matrix W has dimensions 40×20 and the bias B has dimensions 1×40. This makes the number of parameters in the linear layer $40 \times 20 + 40 = 840$.

Now, for the batch norm, we have that $\gamma_i \in \mathbb{R}^{40}$ and $\beta_i \in \mathbb{R}^{40}$, since they have the same dimensions as the output of the linear layer. Hence, the number of parameters in the batch norm layer is $40 + 40 = 80$.

This makes the total number of parameters in the network $840 + 80 = 920$.

c)

Batch Normalization, as the name indicates, normalizes the activations of the neurons in a batch to have zero mean and unit variance during training, which helps stabilize the training process. However, during training we usually are passing batches of data through the network with the same dimensions, but during inference we might be passing single samples through the network or smaller batches. This poses a problem, since the the variance and mean of a smaller batch might not be representative of the ones during training. Besides that, for small batch sizes or single samples, normalization fails due to the lack of statistics. To solve this problem, Batch Normalization layers usually have two modes, one for training and one for inference. During training, the layer computes the mean and variance of the batch and normalizes the activations and keeps a running average of the mean and variance. During inference, the layer uses the running average of the mean and variance from the training to normalize the activations. This solution is presented in PyTorch as indicated by [PyTorch] and originally proposed by [Ioffe and Szegedy, 2015].

d)

A dead neuron has the property that outputs zero for all inputs after a certain time during training becoming inactive and statics, since the weights are not updated anymore. This neuron then does not contribute to the learning process and can be seen as a waste of resources. This dead neurons are common in ReLU activation. This can be explained by the fact that the ReLU activation function is defined as $h(x) = max(0, x)$, which means that during training, if the input of the neuron is negative, which can be caused by negative weights or negative updates on the weights, the neuron will output zero and the weights will not be updated anymore because the gradient will be zero, by the chain rule. This means that this neuron does not contribute either to the output or either to the update of the weights. Dead neurons are harmful to the training process, since they become a waste of computational resources and decrease the capacity of the network, by reducing the number of neurons that are actually learning and by reducing the capacity of updating the weights. It can also lead that during training, the network will dispose features in layers that might be important for the learning process.

e)

As presented in the previous question, in the case of ReLU activation, dead neurons can be caused by negative inputs to the layer. Well, if we have all negative inputs to the layer, the output of the layer will be zero, leading to the dead neuron problem. However, by using a Batch Normalization layer, we can avoid this problem. This is because the Batch Normalization layer normalizes the activations of the neurons to have zero mean and unit variance, meaning that, even with all negative inputs, the Batch Normalization layer will garantee that half of the neurons will have positive outputs. With this, we prevent a zero output on the neuron and avoid the dead neuron problem.

References

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. (arXiv:1502.03167), Mar. 2015. URL `http://arxiv.org/abs/1502.03167`. arXiv:1502.03167.

PyTorch. URL `https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html`.