

Deep Learning Practical Work 1-a and 1-b

Introduction to neural networks

Aymeric DELEFOSSE & Charles VIN

2023 – 2024

1 Introduction to neural networks

1.1 Theoretical Foundation

1.1.1 Supervised dataset

1. ★ What are the train, val and test sets used for? The training dataset is utilized to train the model, while the test dataset is employed to assess the model's performance on previously unseen data. Lastly, the validation set constitutes a distinct subset of the dataset employed for the purpose of refining and optimizing the model's hyperparameters.

2. What is the influence of the number of exemples N ? A larger number of examples can enhance the model's capacity to generalize and improve its robustness against noise or outliers. Conversely, a smaller number of examples can make the model susceptible to overfitting. It is important to note that increasing the dataset size can also lead to an escalation in the computational complexity of the model training process.

1.1.2 Network architecture

3. Why is it important to add activation functions between linear transformations? Otherwise, we would simply be aggregating linear functions, resulting in a linear output. Activation functions introduce non-linearity to the network, enabling the model to capture and learn more intricate patterns than those achievable through linear transformations alone.

4. ★ What are the sizes n_x , n_h , n_y in the figure 1? In practice, how are these sizes chosen?

- $n_x = 2$ represents the input size (data dimension).
- $n_h = 4$ represents the hidden layer size, selected based on the desired complexity of features to be learned in the hidden layer. An excessively large size can result in overfitting.
- $n_y = 2$ represents the output size, determined according to the number of classes in y .

5. What do the vectors \hat{y} and y represent? What is the difference between these two quantities? $y \in \{0,1\}$ represents the ground truth, where the values are binary (0 or 1). $\hat{y} \in [0,1]$ represents a probability-like score assigned to each class by the model. \hat{y} reflects the model's level of confidence in its predictions for each class.

6. Why use a SoftMax function as the output activation function? The reason for employing the SoftMax function is to transform $\tilde{y} \in \mathbb{R}$ into a probability distribution. While there are several methods to achieve this transformation, SoftMax is a commonly utilized choice, especially in multi-class classification problems.

7. Write the mathematical equations allowing to perform the *forward* pass of the neural network, i.e. allowing to successively produce \tilde{h} , h , \tilde{y} , \hat{y} , starting at x . Let W_i and b_i denote the parameters for layer i , $f_i(x) = xW_i^T + b_i$ represent the linear transformation, and $g_i(x)$ be the activation function for layer i .

Calculate the weighted sum and activation for the first hidden layer:

$$\begin{aligned}\tilde{h} &= f_0(x) \\ h &= g_0(\tilde{h})\end{aligned}$$

Proceed to the output layer by computing the weighted sum and activation for the output layer:

$$\begin{aligned}\tilde{y} &= f_1(h) \\ \hat{y} &= g_1(\tilde{y})\end{aligned}$$

These equations describe the sequential steps involved in the forward pass of the neural network, ultimately producing the output \hat{y} based on the input x .

1.2 Loss function

8. During training, we try to minimize the loss function. For cross entropy and squared error, how must the \hat{y}_i vary to decrease the global loss function \mathcal{L} ? Our aim is to minimize the loss function \mathcal{L} to train a model effectively. To decrease cross-entropy loss, make \hat{y}_i closer to 1 when y_i is 1 and closer to 0 when y_i is 0.

For the cross-entropy loss, \hat{y}_i should vary in a way that makes it closer to the true target value y_i for each data point. Specifically, when $y_i = 1$, \hat{y}_i should be pushed towards 1. The closer \hat{y}_i is to 1, the lower the loss. Conversely, when $y_i = 0$, \hat{y}_i should be pushed towards 0. The closer \hat{y}_i is to 0, the lower the loss.

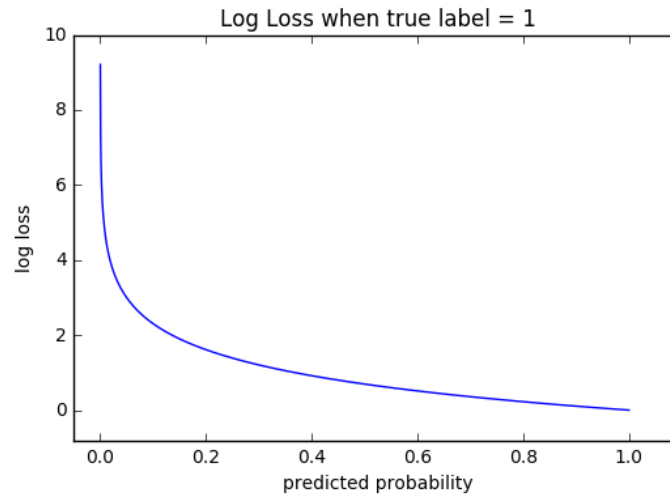
For squared error loss, \hat{y}_i should vary in a way that makes it closer to the true target value y_i for each data point. The goal is to minimize the squared difference between \hat{y}_i and y_i . This means that if \hat{y}_i is greater than y_i , it should decrease, and if \hat{y}_i is smaller than y_i , it should increase.

9. How are these functions better suited to classification or regression tasks? Cross-entropy loss is better suited for classification tasks for several reasons:

- Cross-entropy loss is based on the negative logarithm of the predicted probability of the true class. This logarithmic nature amplifies errors when the predicted probability deviates from 1 (for the correct class) and from 0 (for incorrect classes), depicted in Figure 1.2. Consequently, it effectively penalizes misclassifications, making it particularly suitable for classification tasks. However, it is imperative that \hat{y} remains within the range $[0, 1]$ for this loss to be effective.
- Cross-entropy loss is often used in conjunction with the SoftMax activation function in the output layer of neural networks for multi-class classification. The SoftMax function ensures that the predicted probabilities sum to 1, aligning perfectly with the requirements of the cross-entropy loss.

On the other hand, Mean Squared Error serves a different role and is better suited for regression tasks:

- MSE is ideal when dealing with $(y, \hat{y}) \in \mathbb{R}^2$ (as opposed to the bounded interval $[0, 1]$).
- MSE is advantageous due to its convexity, which simplifies optimization. However, it's important to note that it may not always be the best choice for every regression task, especially when dealing with outliers, as it can be sensitive to extreme values.



1.3 Optimization algorithm

10. What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case? Gradient computation becomes computationally expensive as it scales with the number of examples included. However, including more examples enhances gradient exploration with precision and stability, analogous to the exploration-exploitation trade-off in reinforcement learning.

Classic gradient descent offers stability, particularly with convex loss functions, featuring deterministic and reproducible updates. Nonetheless, its stability may lead to getting trapped in local minima when dealing with non-convex loss functions. Additionally, it is computationally intensive, especially for large datasets, as it demands evaluating the gradient with the entire dataset in each iteration.

Mini-Batch Stochastic Gradient Descent (SGD) converges faster compared to classic gradient descent by updating model parameters with small, random data subsets (mini-batches). This stochasticity aids in escaping local minima and exploring the loss landscape more efficiently. However, it may introduce noise and oscillations.

On the other hand, Online Stochastic Gradient Descent offers extremely rapid updates, processing one training example at a time, making it suitable for streaming or large-scale online learning scenarios. However, its highly noisy updates can result in erratic convergence or divergence, necessitating careful learning rate tuning.

For training deep neural networks in the general case, mini-batch stochastic gradient descent is the most reasonable choice. It strikes a balance between the computational efficiency of classic gradient descent and the noise resilience and convergence speed of online SGD.

11. ★ What is the influence of the *learning rate* η on learning? The learning rate plays a crucial role in influencing various aspects of the learning process, including convergence speed, stability, and the quality of the final model.

A higher learning rate typically results in faster convergence because it leads to larger updates in model parameters during each iteration. However, an excessively high learning rate can cause issues such as overshooting, where the optimization process diverges or oscillates around the minimum, preventing successful convergence.

Conversely, a smaller learning rate allows the optimization algorithm to take smaller steps, which can be advantageous for exploring local minima more thoroughly and escaping shallow local minima. Nonetheless, if the learning rate is too small, it may lead to slow convergence or getting stuck in local minima.

To address these challenges, various techniques like learning rate schedules (gradually reducing the learning rate over time) or adaptive learning rate methods (e.g., Adam) have been developed to automate learning rate adjustments during training. The choice of learning rate may also be influenced by the batch size used in mini-batch stochastic gradient descent, as smaller batch sizes may require smaller learning rates to maintain stability.

In practice, selecting an appropriate learning rate often involves experimentation and can be problem-specific. Techniques such as grid search or random search, in combination with cross-validation, can aid in determining an optimal learning rate for a particular task.

12. ★ Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the *loss* with respect to the parameters, using the naive approach and the *backprop* algorithm. GPT est vraiment pas clair ici, il faut creuser [ce post medium](#) à partir de "Why backpropagation?".

13. What criteria must the network architecture meet to allow such an optimization procedure? For the backpropagation algorithm to be applicable, several criteria must be met. First of all, each layer function, including the activation functions and the loss function, must be differentiable with respect to their parameters. This is crucial for calculating gradients during the training process. Furthermore, as backpropagation relies on the sequential flow of information from the input layer to the output layer, the network architecture should possess a feedforward, sequential structure without loops or recurrent connections. Loops or recurrent connections can introduce complications in gradient calculations.

14. The function SoftMax and the *loss* of cross-entropy are often used together and their gradient is very simple. Show that the *loss* can be simplified by:

$$l = - \sum_i y_i \tilde{y}_i + \log \left(\sum_i e^{\tilde{y}_i} \right).$$

Let us define the cross-entropy loss as $l(y, \hat{y}) = - \sum_{i=1}^N y_i \log \hat{y}_i$, where N represents the total number of examples. The SoftMax function for a vector $\tilde{y} \in \mathbb{R}^D$ is defined as $\text{SoftMax}(\tilde{y}_i) = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^D e^{\tilde{y}_j}}$. Notably, the output of the SoftMax function serves as the input for the cross-entropy loss, denoted as $\hat{y}_i = \text{SoftMax}(\tilde{y}_i)$. Let us substitute this value into the expression.

$$\begin{aligned} l(y, \hat{y}) &= - \sum_{i=1}^N y_i \log \hat{y}_i \\ &= - \sum_{i=1}^N y_i \log \text{SoftMax}(\tilde{y}_i) \\ &= - \sum_{i=1}^N y_i \log \frac{e^{\tilde{y}_i}}{\sum_{j=1}^D e^{\tilde{y}_j}} \\ &= - \sum_{i=1}^N y_i \left[\log e^{\tilde{y}_i} - \log \sum_{j=1}^D e^{\tilde{y}_j} \right] \\ &= - \sum_{i=1}^N y_i \tilde{y}_i - y_i \log \sum_{j=1}^D e^{\tilde{y}_j} \\ &= - \sum_{i=1}^N y_i \tilde{y}_i + \sum_{i=1}^N y_i \log \left(\sum_j e^{\tilde{y}_j} \right) \\ &= - \sum_{i=1}^N y_i \tilde{y}_i + \log \left(\sum_j e^{\tilde{y}_j} \right) \sum_{i=1}^N y_i \end{aligned}$$

Since y_i represents a probability distribution, we know that $\sum_i y_i = 1$. Thus, we have the final expression for the cross-entropy loss:

$$l(y, \hat{y}) = - \sum_i y_i \tilde{y}_i + \log \left(\sum_i e^{\tilde{y}_i} \right)$$

15. Write the gradient of the *loss (cross-entropy)* relative to the intermediate output \tilde{y}

$$\begin{aligned}\frac{\partial l}{\partial \tilde{y}_i} &= -y_i + \frac{\frac{\partial}{\partial \tilde{y}_i} (\sum_{j=1}^N e^{\tilde{y}_j})}{\sum_{j=1}^N e^{\tilde{y}_j}} \\ &= -y_i + \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}} \\ &= -y_i + \text{SoftMax}(\tilde{y})_i \\ \nabla_{\tilde{y}} l &= \begin{pmatrix} \frac{\partial l}{\partial \tilde{y}_1} \\ \vdots \\ \frac{\partial l}{\partial \tilde{y}_{n_y}} \end{pmatrix} = \begin{pmatrix} \text{SoftMax}(\tilde{y})_1 - y_1 \\ \vdots \\ \text{SoftMax}(\tilde{y})_{n_y} - y_{n_y} \end{pmatrix} = \hat{y} - y.\end{aligned}$$

16. Using the *backpropagation*, write the gradient of the *loss* with respect to the weights of the output layer $\nabla_{W_y} l$. Note that writing this gradient uses $\nabla_{\tilde{y}} l$. Do the same for $\nabla_{b_y} l$. Starting with $\nabla_{W_y} l$, we have:

$$\frac{\partial l}{\partial W_{y,ij}} = \sum_k \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial W_{y,ij}}$$

This can be expressed as a matrix:

$$\nabla_{W_y} l = \begin{pmatrix} \frac{\partial l}{\partial W_{y,1,1}} & \cdots & \frac{\partial l}{\partial W_{y,1,n_h}} \\ \vdots & \ddots & \vdots \\ \frac{\partial l}{\partial W_{y,n_y,1}} & \cdots & \frac{\partial l}{\partial W_{y,n_y,n_h}} \end{pmatrix}$$

First, let's compute \tilde{y}_k

$$\begin{aligned}\tilde{y} &= hW_y^T + b^y \\ &= (h_1 \quad \cdots \quad h_{n_h}) * \begin{pmatrix} W_{1,1} & \cdots & W_{1,n_y} \\ \vdots & \ddots & \vdots \\ W_{n_h,1} & \cdots & W_{n_h,n_y} \end{pmatrix} + (b_1^y \quad \cdots \quad b_{n_y}^y) \\ &= (\sum_{j=1}^{n_h} W_{1,j}^y h_j + b_1^y \quad \sum_{j=1}^{n_h} W_{2,j}^y h_j + b_2^y \quad \cdots \quad \sum_{j=1}^{n_h} W_{n_y,j}^y h_{k,j} + b_{n_y}^y) \in \mathbb{R}^{1 \times n_h} \\ \tilde{y}_k &= \sum_{j=1}^{n_h} W_{k,j}^y h_j + b_k^y, k \in [1, n_h]\end{aligned}$$

With this expression, we can now proceed with the calculation of the partial derivative of \tilde{y}_k with respect to W_{ij}^y :

$$\frac{\partial \tilde{y}_k}{\partial W_{ij}^y} = \begin{cases} h_j & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

Now, we need to find $\frac{\partial l}{\partial \tilde{y}_k}$. From the previous question, we have:

$$\frac{\partial l}{\partial \tilde{y}_k} = -y_k + \text{SoftMax}(\tilde{y})_k = \hat{y}_k - y_k$$

Now, combining these results:

$$\frac{\partial l}{\partial W_{i,j}^y} = \sum_{k=1}^{n_h} \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial W_{i,j}^y} = \frac{\partial l}{\partial \tilde{y}_i} \frac{\partial \tilde{y}_i}{\partial W_{i,j}^y} = (\hat{y}_i - y_i) h_j = (\nabla_{W_y} l)_{i,j}$$

So, the gradient of the loss with respect to the weights of the output layer $\nabla_{W_y} l$ is given by $\nabla_{\tilde{y}}^T l$.

$$\begin{aligned}
\nabla_{W_y} l &= \begin{pmatrix} \frac{\partial l}{\partial W_{1,1}^y} & \cdots & \frac{\partial l}{\partial W_{1,n_h}^y} \\ \vdots & \ddots & \vdots \\ \frac{\partial l}{\partial W_{n_y,1}^y} & \cdots & \frac{\partial l}{\partial W_{n_y,n_h}^y} \end{pmatrix} \\
&= \begin{pmatrix} (\hat{y}_1 - y_1)h_1 & \cdots & (\hat{y}_1 - y_1)h_{n_h} \\ \vdots & \ddots & \vdots \\ (\hat{y}_{n_y} - y_{n_y})h_1 & \cdots & (\hat{y}_{n_y} - y_{n_y})h_{n_h} \end{pmatrix} \\
&= \begin{pmatrix} \hat{y}_1 - y_1 \\ \vdots \\ \hat{y}_{n_y} - y_{n_y} \end{pmatrix} (h_1 \quad h_2 \quad \cdots \quad h_{n_h}) \\
&= \nabla_{\tilde{y}}^T h
\end{aligned}$$

17. ★ Compute other gradients : $\nabla_{\tilde{h}} l, \nabla_{W_h} l, \nabla_{b_h} l$

1. The gradient of the loss with respect to \tilde{h} can be computed using the chain rule:

$$\frac{\partial l}{\partial \tilde{h}_i} = \sum_k \frac{\partial l}{\partial h_k} \frac{\partial h_k}{\partial \tilde{h}_i}.$$

Let's compute those two terms

•

$$\frac{\partial h_k}{\partial \tilde{h}_i} = \frac{\partial \tanh(\tilde{h}_i)}{\partial \tilde{h}_i} = \begin{cases} 1 - \tanh^2(\tilde{h}_i) = 1 - h_i^2 & \text{if } k = i \\ 0 & \text{otherwise} \end{cases}.$$

- Having $\tilde{y}_i = \sum_{j=1}^{n_h} W_{i,j}^y h_j + b_i^y$ in mind and recovering $\frac{\partial l}{\partial \tilde{y}_i} = \hat{y}_i - y_i = \delta_i^y$ from past question.

$$\frac{\partial l}{\partial h_k} = \sum_{j=1} \frac{\partial l}{\partial \tilde{y}_j} \frac{\partial \tilde{y}_j}{\partial h_k} = \sum_{j=1} \delta_j^y W_{j,k}^y.$$

Finally we have

$$\begin{aligned}
\frac{\partial l}{\partial \tilde{h}_i} &= \sum_k \frac{\partial l}{\partial h_k} \frac{\partial h_k}{\partial \tilde{h}_i} \\
&= \sum_k \left(\sum_j \delta_j^y W_{j,k}^y \right) \times \begin{cases} 1 - h_i^2 & \text{if } k = i \\ 0 & \text{otherwise} \end{cases} \\
&= (1 - h_i^2) \left(\sum_j \delta_j^y W_{j,i}^y \right) \\
&= \delta_i^h
\end{aligned}$$

So, the gradient $\nabla_{\tilde{h}} l$ is a vector with elements δ_i^h

$$\nabla_{\tilde{h}} l = (1 - h^2) \odot (\nabla_{\tilde{y}} l * W^y).$$

2. $\nabla_{W_h} l$ is a matrix composed of elements

$$\frac{\partial l}{\partial W_{i,j}^h} = \sum_k \frac{\partial l}{\partial \tilde{h}_k} \frac{\partial \tilde{h}_k}{\partial W_{i,j}^h}.$$

From the previous question we already have $\frac{\partial l}{\partial h_k} = (1 - h_k^2) \left(\sum_j \delta_j^y W_{j,k}^y \right) = \delta_k^h$. So let's compute the other term $\frac{\partial \tilde{h}_k}{\partial W_{i,j}^h}$ from $\tilde{h}_k = \sum_{j=1}^{n_x} W_{k,j}^h x_j + b_k^h$

$$\frac{\partial \tilde{h}_k}{\partial W_{i,j}^h} = \begin{cases} x_j & \text{if } k = i \\ 0 & \text{otherwise} \end{cases}.$$

So

$$\begin{aligned} \frac{\partial l}{\partial W_{i,j}^h} &= \sum_k \frac{\partial l}{\partial \tilde{h}_k} \frac{\partial \tilde{h}_k}{\partial W_{i,j}^h} \\ &= \sum_k \delta_k^h \times \begin{cases} x_j & \text{if } k = i \\ 0 & \text{otherwise} \end{cases} \\ &= \delta_i^h x_j \\ \nabla_W^h &= \nabla_h^T l * x \end{aligned}$$

3. Last but not least

$$\frac{\partial l}{\partial b_i^j} = \sum_k \frac{\partial l}{\partial \tilde{h}_k} \frac{\partial \tilde{h}_k}{\partial b_i^j} = \sum_k \delta_k^h * \begin{cases} 1 & \text{if } k = i \\ 0 & \text{else} \end{cases} = \delta_i^h.$$

$$\nabla_{b^h} = \nabla_h^T l.$$