Basics on deep learning for vision

Charles Vin, Aymeric Delefosse

S1-2023

1 Introduction to neural networks

1.1 Theorical Foundation

1.1.1 Supervised dataset

1. What are the train, val and test sets used for ? The train dataset is used to train the model. The test dataset is used to test the model on data it has nether seen before. Finaly the validation set is a separate portion of the dataset used to fine-tune and optimize the model's hyperparameters.

2. What is the influence of the number of exemples N ? A large the number of example can help the model to generalize more and be more robust to noise or outlier. A small number of example can prone to overfitting. Increasing N can also increase the computational complexity of training the model.

1.1.2 Network architecture

3. Why is it important to add activation functions between linear transformations? Otherwise we just sum linear functions so it stays linear. So activation functions introduce non-linearity to the network which permit the model to capture and learn more complex patern than linear.

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

- $n_x = 2$ is the size of the input, the dimension of our data.
- $n_h = 4$ is the size of the hidden layer. It chosen proportionaly to the conplexity of the feature we want to develop in the hiden layer. A large size can lead to overfitting
- $n_y = 2$ is the size of the output, it's choosen in function of the number of class of y

5. What do the vectors $\hat{y}$ and y represent? What is the difference between these two quantities? $y \in \{0, 1\}$ is the ground truth while $\hat{y} \in [0, 1]$ is like a probabilty for each class. $\hat{y}$ express the model's confidence in each class prediction.

6. Why use a $SoftMax$ function as the output activation function? $\tilde{y} \in \mathbb{R}$ so we have to transform it into a probability distribution. There is many way to do that but the $SoftMax$ is commonly used 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 $\hat{h}$, h, $\tilde{y}$, $\hat{y}$, starting at x. Let note W_i, b_i the parameter for the i layer, $f_i(x) = W_i x + b_i$ and $g_i(x)$ the activation function of the layer i.

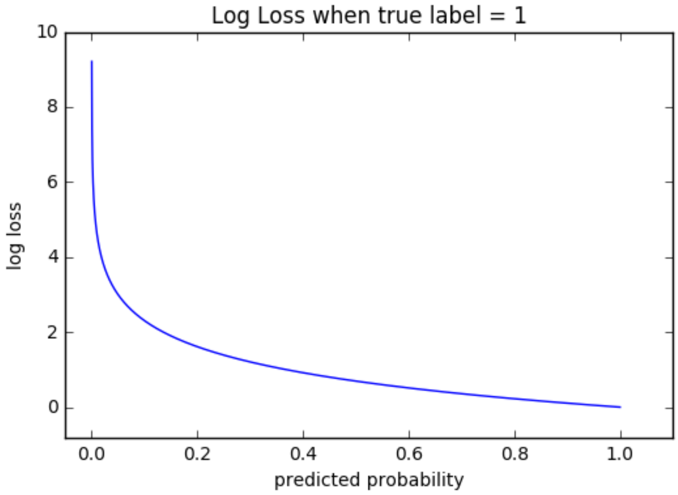
$$\begin{aligned}
\tilde{h} &= f_0(x) \\
h &= g_0(\tilde{h}) \\
\tilde{y} &= f_1(h) \\
\hat{y} &= g_1(\tilde{y})
\end{aligned}$$

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 $\mathcal{L}$? ?

9. How are these functions better suited to classification or regression tasks? Cross-entropy is better suited classification task for numerus reason :

- Cross-entropy is based on the negative logarithm of the predicted probability of the true class. Due to this logarithmic nature, it magnifies the loss when the predicted probability deviates from 1 (for the correct class) and from 0 (for incorrect classes) as you can see in figure 1.2. It would absolutly not work for $\hat{y}$ outside $[0, 1]$. This amplification of errors is beneficial for training the model effectively.
- 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, a perfect output for the CE-Loss



In the other hand, MSE play the opposite role. It would not work at all with $\hat{y} \in [0, 1]$ but is really good with $y, \hat{y} \in \mathbb{R}$. It also cool because of it convexity. However, it's essential to note that MSE is not always the best choice for every regression task. For example, ot can be pretty sensitive to outliers.

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? I one hand, computing a gradient is computationaly expensive proportionately to the number of example we choose to include in it. In the other hand, the more we include examples in it, the more we exploring the gradient with precision and stability. It's a bit like the exploration-action compromise in RL.

The classic gradient descent is pretty stable exspecialy for convex loss function. Her update are deterministic and reproductible as there are nothing stocastic in the algorythm. But when the loss function is not convex, her stability can prone it to get stuck in local minima. Futhermore this method is computationally expensive, especially for large datasets, as it requires evaluating the gradient using the entire dataset in each iteration.

The Mini-Batch Stochastic Gradient Descent (SGD) has the advantages to have a faster convergence compared to classic gradient descent, as it updates the model parameters using small random subsets of the data (mini-batches). This randomness helps to escape local minima and explore the loss surface more effectively than classic gradient descen but can lead to noises and oscillations.

Finaly the Online Stochastic Gradient Descent have extremely fast updates, as it processes one training example at a time and is well-suited for streaming or large-scale online learning scenarios where data arrives sequentially. But her highly noisy update can lead to erratic convergence or divergence and require careful tuning of learning rates.

In the general case for training deep neural networks, mini-batch stochastic gradient descent is the most reasonable choice. It strikes a good 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 choice of learning rate can significantly influence the learning process, including convergence speed, stability, and the quality of the final model.

A higher learning rate generally leads to faster convergence, as it results in larger updates to the model parameters in each iteration. However, if the learning rate is too high, it can cause overshooting, where the optimization process diverges or oscillates around the minimum, preventing convergence.

A smaller learning rate can help the optimization algorithm explore local minima more thoroughly, as it takes smaller steps, allowing it to escape shallow local minima. However, too small of a learning rate can cause the algorithm to get stuck in local minima or take an excessively long time to converge.

Techniques like learning rate schedules (e.g., reducing the learning rate over time) or adaptive learning rate methods (Adam) can help mitigate the need for manual tuning. The choice of learning rate may also depend on the batch size used in mini-batch SGD. 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-dependent. Techniques like grid search or random search, along with cross-validation, can help identify a suitable learning rate for a given 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 ? To use backpropagation, each layer function, activation function and loss must be differentiable with respect to their parameters. The network must have kind of a linear structure, without loops or recurrent connections.

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:

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

Let denote the cross entropy loss $l(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log \hat{y}_i$ with N the total number of example. The Soft Max function $x \in R^D, SoftMax(\tilde{y})_i = \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{D} e^{\tilde{y}_j}}$. The output of the SoftMax function is the input of the cross entropy loss (i.e. $\hat{y}_i = SoftMax(\tilde{y})_i$), let's replace it.

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

AHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH

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

$$\begin{aligned}
\frac{\partial l}{\partial \tilde{y}_i} &= -y_i + \frac{\frac{\partial}{\partial \tilde{y}_i}(\sum_{j=1}^{N} e^{\tilde{y}_j})}{\sum_{j=1}^{N} e^{\tilde{y}_j}} \\
&= -y_i + \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{N} e^{\tilde{y}_j}} \\
&= -y_i + SoftMax(\tilde{y})_i
\end{aligned}$$

$$\nabla_{\tilde{y}} l = \begin{pmatrix} \frac{\partial l}{\partial \tilde{y}_1} \\ \vdots \\ \frac{\partial l}{\partial \tilde{y}_N} \end{pmatrix} = \begin{pmatrix} SoftMax(\tilde{y})_1 - y_1 \\ \vdots \\ SoftMax(\tilde{y})_N - y_N \end{pmatrix} = \hat{y} - y.$$

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 $\nabla_{b_y} l$.

$$\begin{aligned}
\frac{\partial l}{\partial W_{y,ij}} &= \sum_k \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial W_{y,ij}} \\
\Rightarrow \nabla_{W_y} l &= \begin{pmatrix} \frac{\partial l}{\partial W_{y,1,1}} & \cdots & \frac{\partial l}{\partial W_{y,1,n_h}} \\ \vdots & \ddots & \vdots \\ \frac{\partial l}{\partial W_{y,n_y,1}} & \cdots & \frac{\partial l}{\partial W_{y,n_y,n_h}} \end{pmatrix}
\end{aligned}$$

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

$$\begin{aligned}
\tilde{y}_k &= h_k W_y \\
&= \begin{pmatrix} h_{k,1} & \dots & h_{k,n_h} \end{pmatrix} * \begin{pmatrix} W_{1,1} & \dots & W_{1,b} \\ \vdots & \ddots & \vdots \\ W_{a,1} & \dots & W_{a,b} \end{pmatrix} \\
&= \begin{pmatrix} h_{k,1} W_{y,1,1} + \cdots + h_{k,n_h} W_{a,1} & \dots & h_{k,1} W_{1,b} + \cdots + h_{k,n_k} W_{a,b} \end{pmatrix}
\end{aligned}$$

Now its partial derivative

$$\frac{\partial \tilde{y}_k}{\partial W_{y,ij}} = h_{k,j}.$$

Then from the previous question

$$\frac{\partial l}{\partial \tilde{y}_k} = -y_i + SoftMax(\tilde{y})_i.$$

Combine both :

$$\begin{aligned}
\frac{\partial l}{\partial W_{y,ij}} &= \sum_k \frac{\partial l}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial W_{y,ij}} \\
&= \sum_k -y_i h_{k,i} + h_{k,i} SoftMax(\tilde{y})_i \\
&= (\nabla_{W_y} l)_{i,j}
\end{aligned}$$

Finaly :

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