Artificial neural networks

Blaž Erzar

Machine Learning For Data Science 1 2023/24
Homework 4 report

Abstract

This report presents the implementation and evaluation of the artificial neural network model. We define the forward pass using matrix multiplications and compute the gradient using backpropagation. To verify the gradient is valid, we compare it to a numerical approximation. We use the housing datasets to compare the implemented model to linear regression and multinomial logistic regression. In both cases, the neural network achieves lower loss. In the end, we evaluate multiple neural network architectures on the final dataset to select the best one for test predictions.

1 Introduction

In this homework, we implemented a simple feed-forward neural network using backpropagation. We support multiple activation functions, such as sigmoid and ReLU, as well as both regression and classification. Models for both tasks share all the code, the only difference is the last activation layer. In regression, we simply use an identity layer, passing input through, while we use a softmax layer for classification to convert logits to probabilities.

To verify that the gradient calculated using backpropagation is valid, we compare it to a numerical approximation. In the end, we evaluate the model on two housing datasets and compare the results to linear models. Finally, we optimise the model architecture for the final dataset and generate predictions for the test set.

2 Methods

2.1 Forward pass

Let $\mathbf{Z}^{(l)}$ be the output of the l-th layer, $\mathbf{A}^{(l)}$ the output of the following activation layer and $\mathbf{W}^{(l)}$ parameters of the layer. For the l-th linear layer, the forward pass is defined as $\mathbf{Z}^{(l)} = \mathbf{A}^{(l-1)}\mathbf{W}^{(l)}$. Similarly, we define the forward pass of the activation layer: $\mathbf{A}^{(l)} = g(\mathbf{Z}^{(l)})$, where g is the activation function, e.g., sigmoid $g_\sigma(z) = (1 + e^{-z})^{-1}$ or ReLU $g_{ReLU}(z) = \max(0, z)$. For both of these activation functions, we also define the piecewise derivatives, $g'_\sigma(z) = g_\sigma(z)(1 - g_\sigma(z))$ and $g'_{ReLU}(z) = \mathbb{I}[g_{ReLU}(z) > 0]$.

We use one matrix for both weights and biases of the linear layer. Because of this, we need to add a column of ones to the input every time and remove one row from the gradient. Biases are initialised to 0, and for weights, we use the He initialisation [1].

2.2 Backpropagation

In regression, we are optimising the mean squared error loss function

$$J(\mathbf{Y}, \mathbf{A}^{(L)}) = \frac{1}{2n}||\mathbf{A}^{(L)} - \mathbf{Y}||^2,$$

while in classification, we use the cross-entropy loss:

$$J(\mathbf{Y}, \mathbf{A}^{(L)}) = -\sum_{i,j} y_{i,j} \log(a_{i,j}).$$

It turns out, that in both cases, the first gradient equals

$$\frac{\partial J}{\partial \mathbf{Z}^{(L)}} = \mathbf{A}^{(L)} - \mathbf{Y}.$$

We use it to compute the gradient of the last linear layer:

$$\frac{\partial J}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathbf{Z}^{(L)}}{\partial \mathbf{W}^{(L)}} \frac{\partial J}{\partial \mathbf{Z}^{(L)}} = \mathbf{A}^{(L-1)^\intercal} \frac{\partial J}{\partial \mathbf{Z}^{(L)}}. \quad (1)$$

For all the other layers, we first update the gradient wrt. the output:

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{Z}^{(L-1)}} &= \frac{\partial \mathbf{A}^{(L-1)}}{\partial \mathbf{Z}^{(L-1)}} \frac{\partial \mathbf{Z}^{(L)}}{\partial \mathbf{A}^{(L-1)}} \frac{\partial J}{\partial \mathbf{Z}^{(L)}} \\ &= g'\left(\mathbf{Z}^{(L-1)}\right) \odot \left(\frac{\partial J}{\partial \mathbf{Z}^{(L)}} \mathbf{W}^{(L)^\intercal}\right),\end{aligned}$$

and then use the already-used formula (1).

Using this process, we can calculate all the gradients. For each layer, the steps are the same: update the gradient wrt. to the output and compute the gradient of the linear layer.

For the regularisation, we add $\frac{\lambda}{2}||\mathbf{W}||^2$ to the loss function (ignoring the biases) and $\lambda\mathbf{W}$ to the gradient. The final loss function is optimised using L-BFGS-B.

2.3 Gradient verification

To verify that the calculated gradient matches, we approximate the partial derivatives with the definition

of the derivative:

$$\nabla J_i(\theta) = \lim_{h \to 0} \frac{J(\theta_1, \ldots, \theta_i + h, \ldots, \theta_k) - J(\theta)}{h}.$$

The computed mean absolute differences between components are shown in Figure 1. We can see that the difference decreases with better numerical approximations for smaller h, meaning that the gradient and the loss function do match.

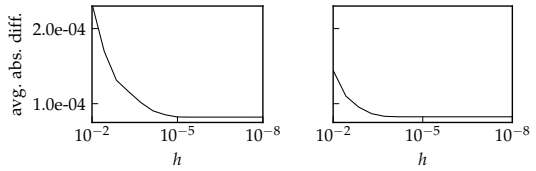

Figure 1: *The average absolute differences between components of the numerical and backpropagated gradient for regression (left) and classification (right).*

2.4 Model selection and preprocessing

To select the best model in the following sections, we use cross-validation. For the housing datasets, we use 10 folds and 5 folds for the final dataset.

For the housing datasets, we use the learning rate $\lambda = 0.01$, while using $\lambda = 0.001$ for the final dataset. These learning rates were empirically selected because they worked best with most of the architectures. For the activation function, we use ReLU because the results were similar using sigmoid, but ReLU runs faster.

When selecting the best model for the housing datasets, we standardise the dataset to $\mu = 0$ and $\sigma = 1$, but for the final dataset, we use the original values because of better results. For classification, we convert discrete target values to numbers between 0 and $c - 1$, where c is the number of classes.

3 Results

On both datasets, we evaluate multiple network architectures, from 0 to 2 or 3 hidden layers whose sizes range from 8 to 64.

3.1 Housing datasets

For both datasets, the architecture with 2 hidden layers of size 16 works the best. For regression, we achieve the MSE loss of 13.03, which is lower than linear regression at 18.89, see Figure 1.

In classification, the neural network is again better than the linear model - multinomial logistic regression in this case. The linear model only achieved a loss of 0.308 and an accuracy of 86.1%, which is around 2% worse than the neural network, see Figure 2.

Units	MSE loss
[16, 16]	13.03
[16]	13.25
[8]	13.89
[32, 32]	14.00
[32, 16]	15.68

Table 1: *Top* 5 *cross-validation losses on the housing2r dataset for different neural network architectures.*

Units	Log loss	Accuracy [%]
[16, 16]	0.247	89.8
[64]	0.250	88.4
[32]	0.252	88.8
[16]	0.256	88.6
[8]	0.258	88.4

Table 2: *Top* 5 *cross-validation losses with corresponding accuracies on the housing3 dataset for different neural network architectures.*

3.2 Final dataset

We evaluate models on the final train dataset to select the optimal one for prediction on the test set. As we can see from Figure 3, the optimal architecture contains 2 hidden layers with size 64.

Units	Log loss	Accuracy [%]	Time [s]
[64, 64]	0.529	79.8%	299
[64]	0.534	80.0%	200
[32, 32]	0.542	79.2%	410
[32, 16]	0.546	79.2%	354
[32, 32, 32]	0.547	79.3%	371

Table 3: *Top* 5 *cross-validation losses, accuracies and running times on the final dataset for different neural network architectures.*

4 Discussion

We have described the process of implementing a neural network and have shown that the gradient calculation is correct. The gradient allows us to optimise the loss function and the optimised model performs better on the housing datasets than the linear models. To select the best model on the final dataset, we again use cross-validation and achieve an accuracy of around 80%.

References

[1] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852.