Deep learning homework 1

Matej Miočić (63180206)

1 Introduction

In this homework we implemented a basic neural network and various feature implementations such as using different optimizers (SGD and ADAM), L2 regularization and learning rate scheduler. We examine the impact of each feature and its parameters by comparing it to the default network.

2 Experiments

2.1 Implementation

We used our neural network on the CIFAR-10 dataset. This is a 10 class prediction problem. This means we use cross entropy as our loss function (1):

$$J = \frac{1}{m}((-\sum_{i}^{m}\sum_{j}^{C}\hat{y}_{ij}\log(a_{ij}^{(L)})) + \frac{\lambda}{2}\|w\|^2), \tag{1}$$

where J is the cost function, m is the number of instances, C is the number of classes, $a_{ij}^{(L)}$ is the predicted probability for i-th instance, $\hat{y}_{ij}$ is the true binary value of i-th instance and λ as regularization parameter.

To obtain probabilities of each class we used a softmax activation function (2) on the last layer:

$$f(x_i) = \frac{e^{x_i}}{\sum_j^C e^{x_{ij}}}. \tag{2}$$

We use sigmoid (3) as our activation function for all other layers:

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{3}$$

In the following we show derivations of partial derivatives for our setup without regularization. We show equations for the last layer (4):

$$\frac{\partial J}{\partial W^{(L)}} = \frac{\partial Z^{(L)}}{\partial W^{(L)}} \times \frac{\partial J^{(L)}}{\partial Z^{(L)}}, \tag{4}$$

and for each component (5):

$$\frac{\partial Z^{(L)}}{\partial W^{(L)}} = A^{(L-1)}, \quad \frac{\partial J^{(L)}}{\partial Z^{(L)}} = A^{(L)} - Y. \tag{5}$$

And every other layer (6):

$$\frac{\partial J}{\partial W^{(l)}} = \frac{\partial Z^{(l)}}{\partial W^{(l)}} \times \frac{\partial A^{(l)}}{\partial Z^{(l)}} \times \frac{\partial Z^{(l)}}{\partial A^{(l-1)}} \times \frac{\partial A^{(l+1)}}{\partial Z^{(l+1)}} \times \frac{\partial J}{\partial A^{(l+1)}}, \tag{6}$$

where we denote any layer but last with l. We show derivation of new components (7):

$$\frac{\partial Z^{(l)}}{\partial A^{(l-1)}} = W^{(l)}, \quad \frac{\partial A^{(l)}}{\partial Z^{(l)}} = A^{(l)}(1 - A^{(l)}). \tag{7}$$

For all of the following experiments (except where noted) we used:

- num_of_hidden_layers = 2,
- num_of_hidden_nodes = 100,
- batch_size = 64,
- epoch = 20,
- optimizer = SGD,
- γ = 0.3,
- decay_rate = 0,
- λ = 0

2.2 Regularization

We implemented L2 regularization (8) which is a technique to prevent overfitting by adding a penalty term to the loss function. Since we change the loss function, the gradient update changes as well:

$$w_{t+1} = w_t - \gamma\left(\nabla J + \lambda w_t\right). \tag{8}$$

During training, the weight update for L2 regularization involves adding a scaled version of the weights to the gradient of the loss function. The scale factor is proportional to the value of the regularization parameter: λ, which determines the strength of the regularization.

As we can see, the weight update includes an additional term that penalizes the weights proportional to their size. This encourages the weights to be small, which in turn encourages a simpler model that is less prone to overfitting.

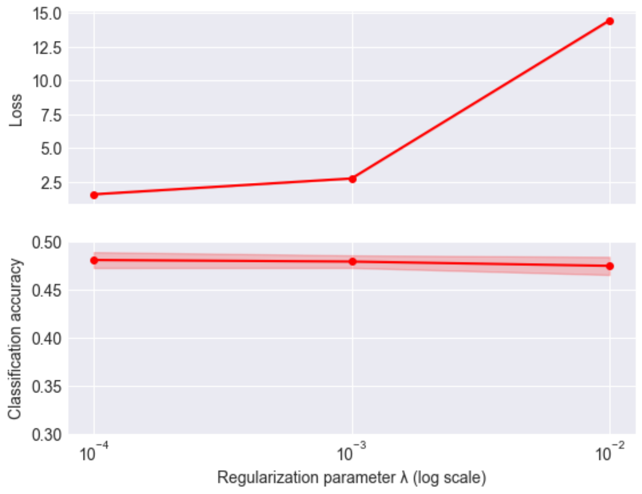

Figure 1: Loss and classification accuracy with standard deviation for 10 times repeated run on test set for different regularization parameter values: λ = {0.01, 0.001, 0.0001}

In Figure 1 we show obtained loss and classification accuracy for different regularization parameter values. We observe that setting λ too big, may cause the model to not train properly anymore resulting in bigger loss.

2.3 Optimizers

We implemented 2 optimization algorithms for our neural network: Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (ADAM). SGD (9) updates the model's parameters by computing the gradients of the loss function then adjusts the parameters in the opposite direction of the gradient with a learning rate:

$$w_{t+1} = w_t - \gamma \nabla J(w_t; x_i, y_i). \tag{9}$$

Meanwhile ADAM (10) is a variant of SGD that dynamically adjusts the learning rate for each parameter based on the estimated first and second moments of the gradients:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla J(w_{t-1}; x_i, y_i), \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla J(w_{t-1}; x_i, y_i))^2, \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}, w_{t+1} = w_t - \frac{\gamma}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t, \end{aligned} \tag{10}$$

where we used β_1 = 0.9, β_2 = 0.999 and ϵ = 10^{-8}, and t is the iteration.

In Figure 2 we show how results change depending on the choice of learning rate. We show that ADAM optimizer fails with a too big learning rate, meanwhile SGD performs best when learning rate is around 0.1.

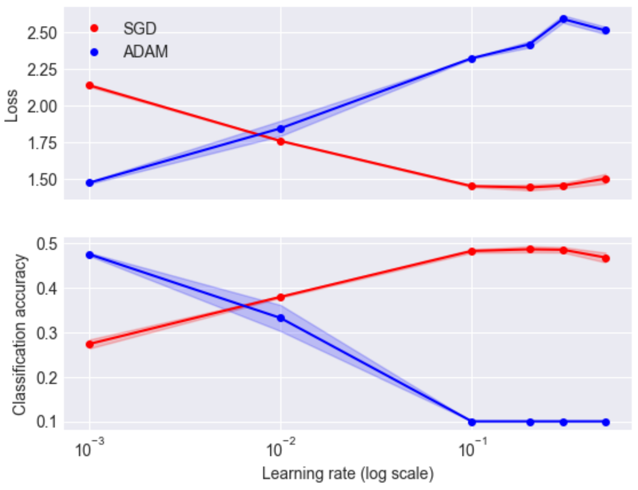

Figure 2: Loss and classification accuracy with standard deviation for 10 times repeated run on test set of 2 optimization alghorithms: SGD and ADAM for different learning rates: γ = {0.001, 0.01, 0.1, 0.2, 0.3, 0.5}

2.4 Learning rate decay

We also implemented exponentaial learning rate decay where the learning rate decreases exponentially with each iteration, which means that it starts off high and gradually decreases over time. The decay rate k controls how quickly the learning rate decreases.

$$\gamma_t = \gamma_0 \cdot e^{(-kt)}, \tag{11}$$

where γ_t is the learning rate at iteration t, γ_0 is the initial learning rate, k is the decay rate, and t is the current iteration.

Since we did not want our learning rate to decrease too rapidly, we set t parameter to the epoch number.

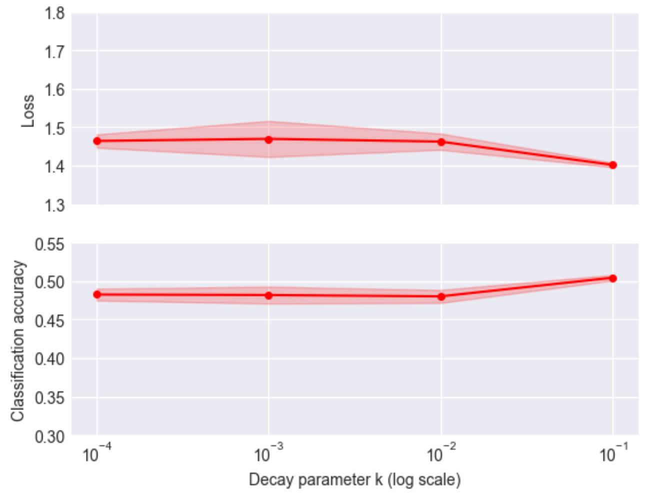

Figure 3: Loss and classification accuracy with standard deviation for 10 times repeated run on test set for different regularization parameter values: k = {0.1, 0.01, 0.001, 0.0001}

In Figure 3 we observe that in our case having a bigger decay rate, enabled the model to obtain a lower loss, which means that the lower learning rate enabled the model to make smaller steps towards minima.

3 Conclusion

We implemented and studied various feature implementations and their parameters. First we implemented L2 regularization where we showed that a too big regularization parameter λ may prevent model to train properly. Then we have implemented 2 optimization algorithms where we have shown that different values of learning parameter satisfy different optimization algorithms. And lastly we implemented exponential learning rate, where we tested different values of decay parameter. Although we did not explore different structures of network and different number of epoch, we have achieved around 50% classification accuracy which beats the majority baseline which is around 10%.