DD2424 Deep Learning in Data Science

Assignment 3

Clara Tump *tump@kth.se*

June 25, 2019

1 Introduction

This report describes the implementation of and experiments with a k-layer neural network with a cyclical learning rate and batch normalization.

2 Implementation

A k-layer neural network was implemented from scratch using Python 3.6. The neural network is trained using backpropagation and mini-batch gradient descent. As a data set CIFAR-10 is used, which consists of 32x32 images corresponding to 10 classes of objects. The model makes use of cyclical learning rate and batch normalization. The hyperparameter λ is optimized using course and fine search.

3 Experiments

3.1 Gradients

The gradients were checked against numerically computed gradients using the centered difference method. Table 1 shows the relative errors for the gradient for a 4-layer neural network with hidden layer sizes [4,5,6]. The input dimension is 3 and the check was made using one mini-batch containing 6 input examples. These relative errors are all below 10^{-6} which indicates that the gradient computations with batch normalization are correct. It can be noted that the errors for b0, b1 and b2 are irrelevant, since they are redundant when using batch normalization.

Parameter	Relative error in gradient
w0	$3.112 * 10^{-6}$
b0	$2.512 * 10^{-8}$
w1	$2.221 * 10^{-8}$
b1	$2.552 * 10^{-9}$
w2	$3.756 * 10^{-7}$
b2	$1.676 * 10^{-8}$
w3	$4.374 * 10^{-7}$
b3	$1.399 * 10^{-9}$

Table 1: Results of checking the gradient computations. The relative error with respect to the centered difference method are given for each parameter of the model.

3.2 Evolution of the loss

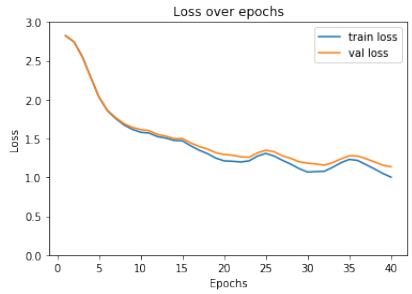
Figure 1 shows the evolution of the loss for a 3-layer neural net with 50 nodes in both hidden layers, with and without batch normalization. The parameters were initialized using He initialization and the hyperparameters were set as follows:

$$eta\_min = 1e-5 \qquad eta\_max = 1e-1 \qquad \lambda = 0.005 \qquad n\_s = 5 * 45,000/n\_batch. \tag{1}$$

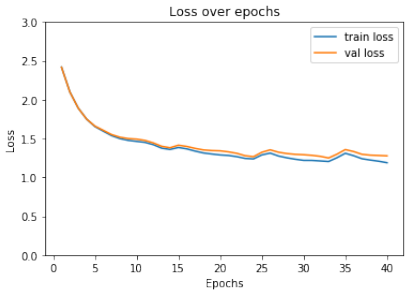
The model was trained for 40 epochs. Figure 2 shows the evolution of the loss for a 9 layer neural net with hidden layer sizes [50, 50, 30, 20, 20, 10, 10, 10, 10] and the same initialization and hyperparameter settings.

These results show that batch normalization has the effect that the loss keeps on decreasing in later epochs, also ending at a lower final loss. We can conclude that batch normalization even has a positive

effect on a relatively shallow model with only 2 hidden layers. For a deeper network batch normalization has an even greater positive effect.

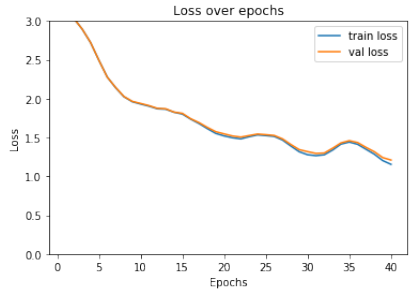


(a) Loss with batch normalization.
Final loss: **1.003**

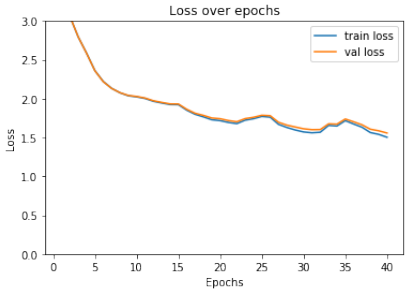


(b) Loss without batch normalization.
Final loss: **1.188**

Figure 1: Evolution of the loss for a 2-layer neural net with and without batch normalization



(a) Loss with batch normalization.
Final loss: **1.155**



(b) Loss without batch normalization.
Final loss: **1.503**

Figure 2: Evolution of the loss for a 9-layer neural net with and without batch normalization

3.3 Optimization of regularization hyperparameter λ

Course and fine search were used for the optimization of the hyperparameter λ. In the course search, a broad range of values were tested, and their performance on the validation set were recorded. 20 epochs were used to train the models. Five values were tested: $1e^k$ with $k \in \{-5, -4, -3, -2, -1\}$. The results are given in Table 4

k	val acc
-5	0.5264
-4	0.5002
-3	0.5128
-2	0.5401
-1	0.3834

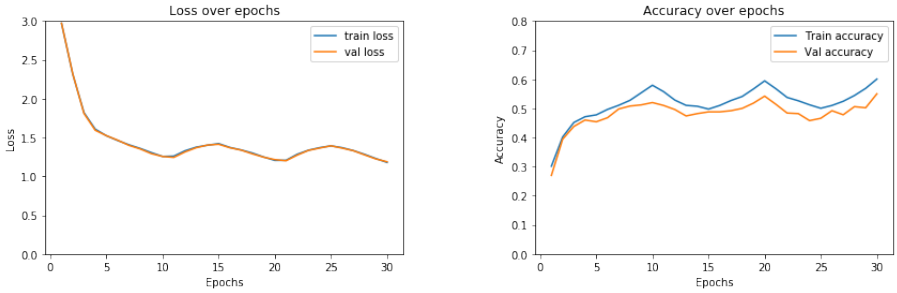
Table 2: Results of course search for hyperparameter λ

As can be seen in Table 4, the performance is the best when $\lambda = 1^{-2}$. Subsequently, the fine search explored the performance on 6 values of λ around these high performance values in the course search: $[4 * 1e-3, 8 * 1e-3, 1e-2, 2 * 1e-2, 4e-2, 8e-2]$.

The results of the fine search are given in Table 3. These results show the best performance with $\lambda = 8e-3$. This value is thus chosen for the final model. This model is run for 32 epochs, with α set to 0.7. This results in a final test accuracy if **0.5308**. The loss and accuracy of this model is plotted in Figure 3.

λ	val acc
$4 * 1e-3$	0.5150
$8 * 1e-3$	0.5413
$1 * 1e-2$	0.5381
$2 * 1e-2$	0.5239
$4 * 1e-2$	0.4135
$8 * 1e-2$	0.3653

Table 3: Results of fine search for hyperparameter λ



(a) Loss (b) Accuracy

Figure 3: Final model with optimized λ. Final test accuracy: **0.5308**

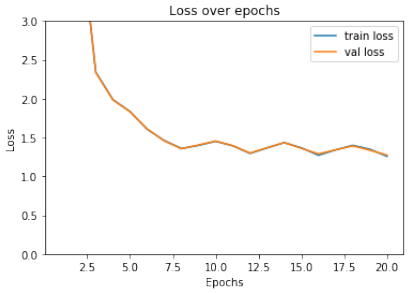
3.4 Sensitivity to Initialization

Lastly, an experiment was conducted to test the sensitivity to initialization with and without batch normalization. Six models were compared, each with a different initialization of the weights (mean 0, variance sig), and each was tested with and without batch normalization. Each model was run for 20 epochs using the standard parameter settings as mentioned in (1).

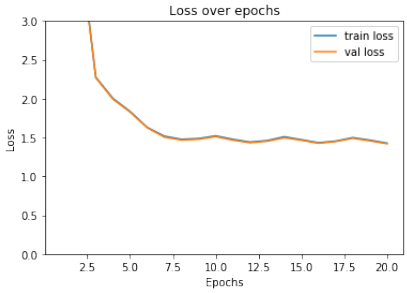
The results are given in Table 4 and the evolution of the loss for each model is shown in Figure 4. It can be seen that the models with batch normalization are significantly less sensitive to initialization: each model with batch normalization performs well with a test accuracy higher than 0.52. However, without batch normalization, the model does not even learn anymore if the variance of the initialized *sig* becomes too small. These results indicate that batch normalization provides robustness to initialization of the parameters.

sig / batch norm	**yes**	**no**
1e-1	0.5239	0.1966
1e-3	0.5234	0.1
1e-4	0.5282	0.1

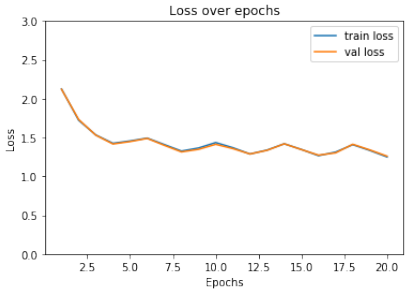
Table 4: Results of the experiment on sensitivity of initialization. With different initialization of the variances of the weights (sig) while using batch normalization or not (batch norm).



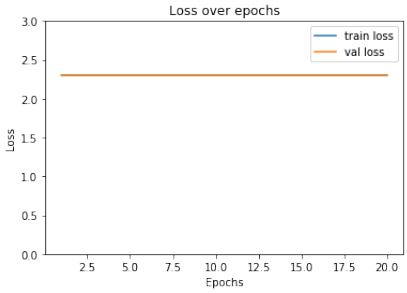
(a) sig=1e-1, with bn



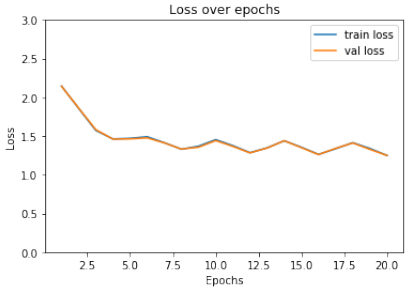
(b) sig=1e-1, without bn



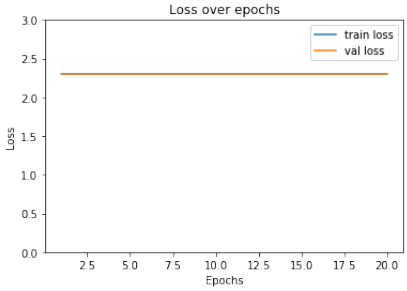
(c) sig=1e-3, with bn



(d) sig=1e-3, without bn



(e) sig=1e-4, with bn



(f) sig=1e-4, without bn

Figure 4: Loss plots for the sensitivity of initialization experiment