Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

I

Курсова робота
за спеціальністю „Інженерія програмного забезпечення”

Optimal initialization of neural networks

Керівник курсової роботи
доц. к ф. -м н, Крюкова Г. В.

_____
(підпис)

“____” _____ 2020 р.

Виконав
студент 3 курсу
факультету інформатики
Кузьменко Д. О.

Table of contents

Contents

Introduction

Artificial neural networks have seen a big surge in popularity. The reason for that is the fact that a lot of different areas, such as photo/video – oriented tasks (object detection, object recognition, semantic segmentation, bounding boxes etc.), neural machine translation, optical character recognition, automated driving and many others, are at the moment the best application of such an approach. Moreover, the use of so called “deep” neural networks is getting widely known now. While having both the computational power increase (modern GPUs are helping with the matter) and the access to enormous amounts of data pushes machine learning engineers and researchers to give up the idea of shallow networks (the ones that have few layers).

Training the aforementioned deep networks is a rather complicated task and requires quite a tedious approach. The most vital concept of training is the **parameter(weight) initialization.** Depending on the correctness of the initial parameter set, the training may result in completely different outcomes.

There are two main problems that are dealt with provided the correct weight initialization – **vanishing** and **exploding gradient** problems. This is of course dependent on the activation function used during training and is especially true for ReLU activation function. There are different approaches to parameter initialization – from quite old ones, such as random initialization, Xavier initialization and He initialization to most recent ones, for instance FuseInit, fixed-update initialization (Fixup) and many others.

Challenges

1. Exploding & Vanishing gradient problems

The backpropagation algorithm works by going from the output layer to the input layer, while doing the error propagation on the way. The gradients of cost function w.r.t to each parameter in the network are used to updates the weights with a Gradient Step. However, quite common situation is that gradients become smaller and smaller as the algorithms gets down to lower layers. Lower connection weights remain unchanged and the convergence to a good solution is almost impossible. This is a so-called vanishing gradient problem.

As a matter of fact, the opposite can happen transpire – the gradients can get so big, that the algorithm diverges due to layers getting large weight updates (exploding gradient problem).

2. Normalization and how to train without it.

Basic initialization techniques

- Zero initialization

If initialized with zeros, weights provide the same derivative with respect to loss function for every weight in the weight matrix. This, in turn, makes for a typical classifier (for classification tasks) or regressor (for regression tasks) which is not particularly better than a linear model.

- Random initialization

Initializing weights at random is considered to be better than just zero assignment. However, exploding and vanishing gradient problems come in play here.

Going a bit further. He and Xavier initialization

The signal has to flow properly in both directions. To achieve that, we need the variance of the outputs of the layer to be equal to the variance of its inputs, as well as gradients ought to have equal variance pre- and post-layer flowing. Both requirements, unfortunately, cannot be guaranteed, unless a layer possesses an equal number of input and output connections. In order to deal with exploding and vanishing gradient problems, a new approach has been developed by Xavier Glorot and Yoshua Bengio. They came up with an idea, quite impressive for the time, to initialize weights randomly using a normal distribution with a standard deviation of 1 and a mean of 0.

The connection weights should be initialized as described in Equation 1, where n_{inputs} and $n_{outputs}$ are the number of input and output connections for the layer. It has to be

noted that this initialization technique also known as Xavier Initialization is only limited to logistic sigmoid/tanh activation functions.

Table 1. Initialization parameters for each type of activation function.

Activation function	Uniform distribution [–r, r]	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Equation 1. Xavier initialization (when using the logistic activation function)

Normal distribution with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Or a uniform distribution between -r and +r, with $r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$

$$Var(w_i) = \frac{1}{fan\_in}$$

Xavier Initialization (with a factor of 1)

He Normal also known as He Initialization. In this method, the weights ‘look’ at the size of the previous layer which leads to successful attainment of a global minimum of the cost function. Although weights are random, they differ in range (depending on the size of the previous layer). Mostly used for ReLU and Leaky ReLU activation functions.

$$Var(w_i) = \frac{2}{fan\_in}$$

He Initialization (with a factor multiplied by 2)

Recent approaches to parameter initialization.

FuseInit

A new initialization technique by Ramina Ghods, Andrew Lan et al. suggests training a deeper network with any kind of random initialization, which would then be used to initialize a shallower network by fusing neighboring layers.

Training with and without normalization. Batch normalization. Fixup Init.

Despite the fact that He initialization combined with leaky ReLU or ELU activation function addresses exploding and vanishing gradient problem to certain extent, it is still not perfect as it doesn't guarantee that alleviated at first exploding/vanishing gradients will not come back later during training. Sergey Ioffe and Christian Szegedy in ***2015 paper*** proposed a technique called *Batch Normalization,* which addresses the problem of Internal Covariate shift (the problem of distribution of each layer's inputs changing during training, as the weights of the previous layers are changed. This technique involves adding zero-centering and input normalization, then scaling and shifting the result using two new parameters (one for scaling and one for shifting). This is all done just before the activation function. To apply these operations, the algorithm needs to estimate the inputs' mean and standard deviation. It is being done via the evaluation of the mean and standard deviation of the inputs over the current mini-batch. The equation 2 summarizes the algorithm.

Equation 2. Batch Normalization algorithm

1. $\mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B} \mathbf{x}^{(i)}$
2. $\sigma_B{}^2 = \frac{1}{m_B}\sum_{i=1}^{m_B}\left(\mathbf{x}^{(i)} - \mu_B\right)^2$
3. $\hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu_B}{\sqrt{\sigma_B{}^2 + \epsilon}}$
4. $\mathbf{z}^{(i)} = \gamma \hat{\mathbf{x}}^{(i)} + \beta$

- *μ_B is the empirical mean, evaluated over the whole mini-batch B*
- *σ_B is the empirical standard deviation, also evaluated over the whole mini-batch.*
- *m_B is the number of instances in the mini-batch*
- *$x^{(i)}$ is the zero-centered and normalized input.*
- *γ is the scaling parameter for the layer.*
- *β is the shifting parameter (offset) for the layer.*
- *ϵ is a tiny number to avoid division by zero (typically 10^{-3}). This is called a smoothing term.*
- *$z^{(i)}$ is the output of the BN operation: it is a scaled and shifted version of the inputs.*

With the help of Batch Normalization, the vanishing gradient problem was strongly reduced, and the network itself is less sensitive to weight initialization.

Fixup Initialization (training residual networks without normalization)

Fixup Init, a method proposed recently by Hongyi Zhang, Yann Dauphin and Tengyu Ma, rescales the standard initialization of residual branches by adjusting for the network architecture. This in turn allows training at top learning rate without the actual normalization.

Algorithm rules:

1. Initialize the classification layer and the last layer of each residual block to 0.
2. Initialize every other layer using He Initialization and scale only the weight layers inside residual blocks by $L^{-1/(2*m-2)}$.
3. Add a scalar multiplier initialized at 1 in every block and a scalar bias initialized at 0 before each convolution, linear and element-wise activation layer.

Equation 3 suggests new methods to initialize a residual block through rescaling the standard initialization of i-th layer in a residual branch by its corresponding scalar a_i., a_i could be set to equal ***$L^{-1/(2*m-2)}$.*** *Rule 2 is the most vital and necessary rule for training deep residual networks.*

Equation 3.

$$\left(\prod_{i\in[m]\setminus\{j\}} a_i\right) x = \Theta\left(\frac{1}{\sqrt{L}}\right), \quad \text{where} \quad j \in \arg\min_k a_k$$

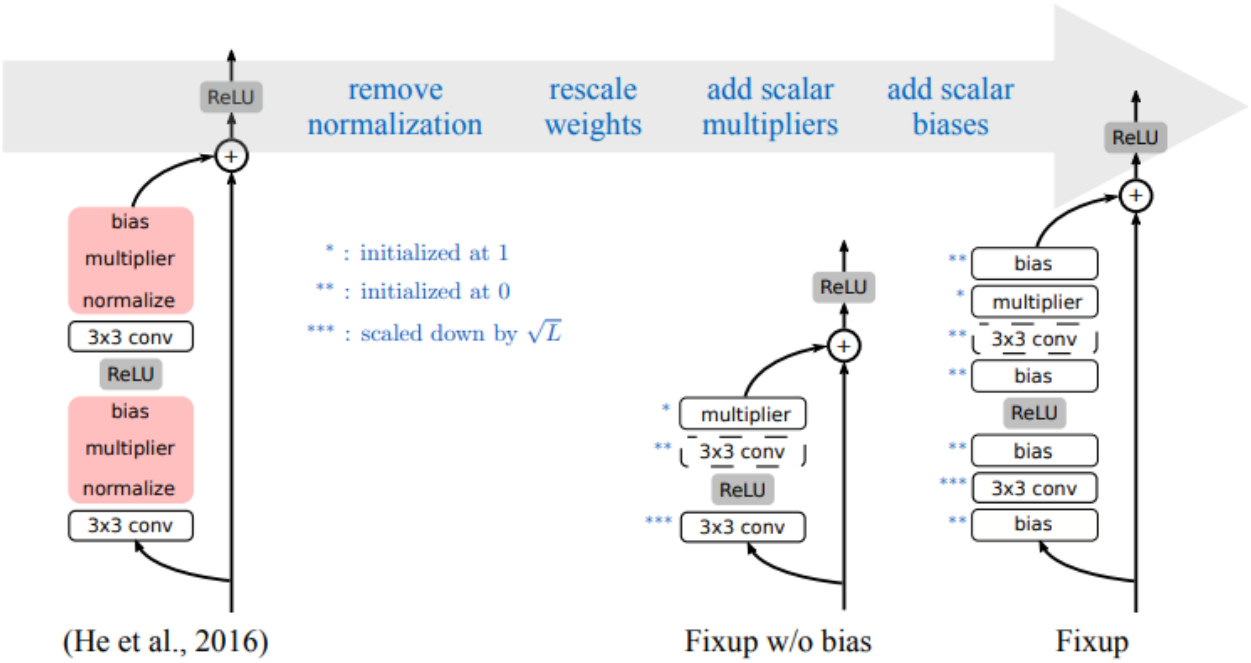


Figure 1. To the left is a residual block of ResNet. BatchNorm layers are marked in red. In the middle – a simple network block that trains stably when stacked together. To the right Fixup Init improves by adding bias parameters.

Benchmarks will be calculated on generated via sklearn.datasets.make_blob dataset with 100k samples with 5 features and 4 centroids. Data has been one-hot-encoded before being fed into neural network. Algorithm used for conversion – Gradient Descend. Following scenarios are tested: sigmoid with zeros init, random, He, Xavier and Fixup; ReLU with zeros init, random, He, Xavier and Fixup.

Dependence on the activation function.

$$v^2 = \frac{1}{N(g'(0))^2(1+g(0)^2)} \quad (4)$$

Equation 4. g(x) is an activation function, v^2 – variance squared.

The equation shows a general weight initialization strategy for any differentiable activation function.

Activation function differentiable at 0. Hyperbolic Tangent and Sigmoid.

If we substitute g with tanh, we get g(0) = 0, and g'(0) = 1. And receive that v is approximately 1/sqrt(N). The results resemble Xavier Initialization:

$$Var(w_i) = \frac{1}{fan\_in}$$

For the sigmoid activation defined as

$$g(x) = \frac{1}{1 + \mathrm{e}^{-x}} \quad (5)$$

We have g(0) = 0.5, g'(0) = ¼. We get v ≈ 3.6/sqrt(N) **[1]**. The benchmarks listed below prove the fact of the stalled convergence over time with Xavier initialization, while showing the better results with initialization suggested in [1].

Activation functions not differentiable at zero. ReLU, Leaky ReLU.
Let M be the number of layers in a deep neural network. The relationship between the inputs m-th layer(x_m) and m+1th layer are described by the following recursions:

$$y_m(i) = \sum_{j=1}^{j=N} \mathbf{W}_m(i,j) x_m(j) = \sum_{j=1}^{j=N} p_{ij}$$

$$x_{m+1}(i) = \mathrm{g}(y_m(i)). \quad (6)$$

g(x) is once again an activation function, W_m is a matrix of weights of the mth layer, N is the number of nodes in the hidden layer. The weights are

identically distributed normal random variables with the mean of 0 and variance of v^2. For the first layer however, same rules apply for the weights, with the exception of variance being equal to 1.

For the demonstration purposes, r_m denotes the mean, u^2_m denotes variance of $y_m(i)$.

Since ReLU, F(x) = max(0, x), is not differentiable at 0, the optimal value of v^2 cannot be computed optimally.

During the first iteration, Wm(i, j) is independent of $x_m(k)$ for all values of i, j and k. (7).

Taking into consideration (6) and (7) and the fact that $E(W_m(i,j)) = 0$ yields the following:

$$r_m = E(y_m(i)) = \sum_{j=1}^{j=N} E(\mathbf{W}_m(i,j))E(x_m(j)) = 0. \tag{8}$$

Referring to (8), for the first iteration the next rule is applied - $y_m(i) \sim N(0, u^2_m)$.

We need both the mean and the variance of $x_{m+1}(i) = \max(0, y_m(i))$.

The mean is received via

$$\mu_{m+1} = E(y_m(i)\mathbf{I}(y_m(i) > 0)) = \frac{1}{u_m\sqrt{2\pi}} \int_0^\infty x\mathrm{e}^{-\frac{x^2}{2u_m^2}} dx = \frac{u_m}{\sqrt{2\pi}} \int_0^\infty \mathrm{e}^{-t} dt = \frac{u_m}{\sqrt{2\pi}}. \tag{9}$$

The variance,

$$E(x_{m+1}(i)^2) = \frac{1}{u_m\sqrt{2\pi}} \int_0^\infty x^2\mathrm{e}^{-\frac{x^2}{2u_m^2}} dx = \left(\frac{1}{2}\right) \frac{1}{u\sqrt{2\pi}} \int_{-\infty}^\infty x^2\mathrm{e}^{-\frac{x^2}{2u^2}} dx = \frac{u_m^2}{2}.$$

(10)

Using (9) and (10), we receive:

$$s^2_{m+1} = E(x_{m+1}(i)^2) - \mu^2_{m+1} = \frac{u^2_m}{2} - \frac{u^2_m}{2\pi} \approx 0.34u^2_m. \quad (11)$$

In order to keep variance at the same level at each iteration, we require s^2_{m+1} to be approximately equal to 1, which gets us $u^2_m = 1 / 0.34 \approx 3$. Inserting (11) into (9) we receive $\mu_{m+1} \approx 0.7$. (12)

$u^2_m = N*v^2(s^2_m + \mu^2 m)$ (13).

Conclusively, putting (12) into (13) gets us $N*v^2(1+0.49) = 3 \Rightarrow v^2 \approx 2/N$.(14)

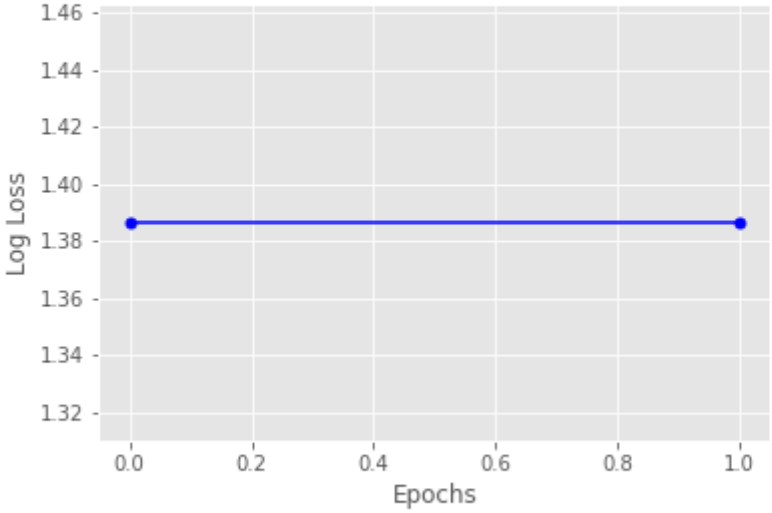
The result obtained in (14) resembles the He Initialization. Therefore, He Init is suitable for ReLU and Leaky ReLU activation functions.

Benchmarks (sigmoid activation)

Initializing with zeros is not a good idea.

```
zeros sigmoid
```

```
100% 100/100 [00:03<00:00, 28.84epoch/s]
```

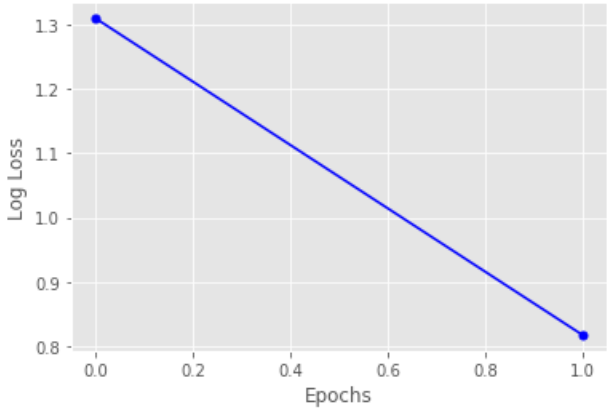



```
Training accuracy 0.25
Validation accuracy 0.25
```

Random init leads to a much better results in comparison with being initialized with zeros.

```
random sigmoid
```

```
100% 100/100 [00:03<00:00, 29.69epoch/s]
```

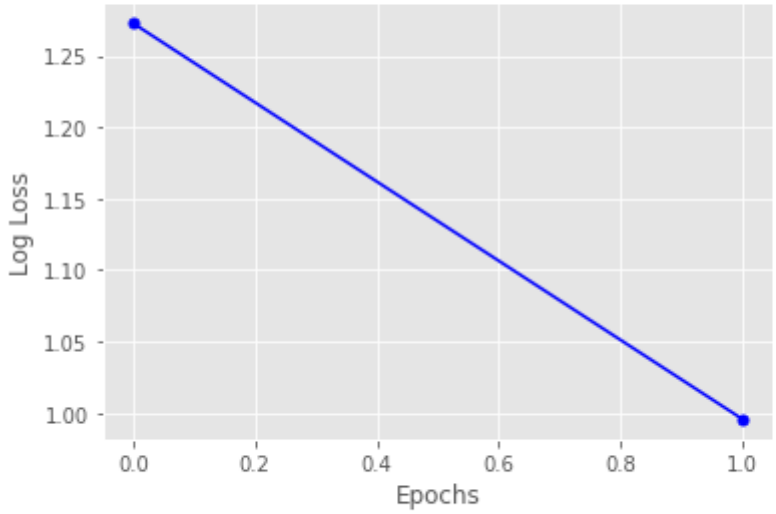



```
Training accuracy 0.62871
Validation accuracy 0.63052
```

Xavier init is performing really nice with sigmoid activation function.

```
xavier sigmoid
```

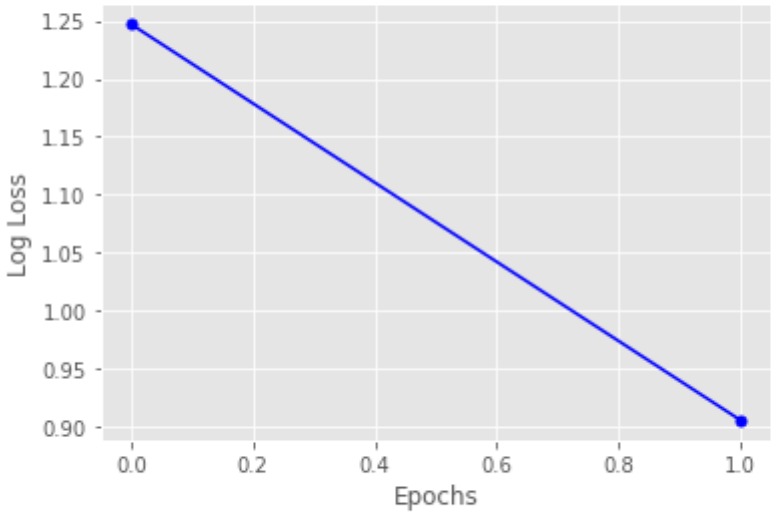
100% 100/100 [00:03<00:00, 29.01epoch/s]




```
Training accuracy 0.91324
Validation accuracy 0.9128
```

```
he sigmoid
```

100% 100/100 [00:03<00:00, 29.10epoch/s]

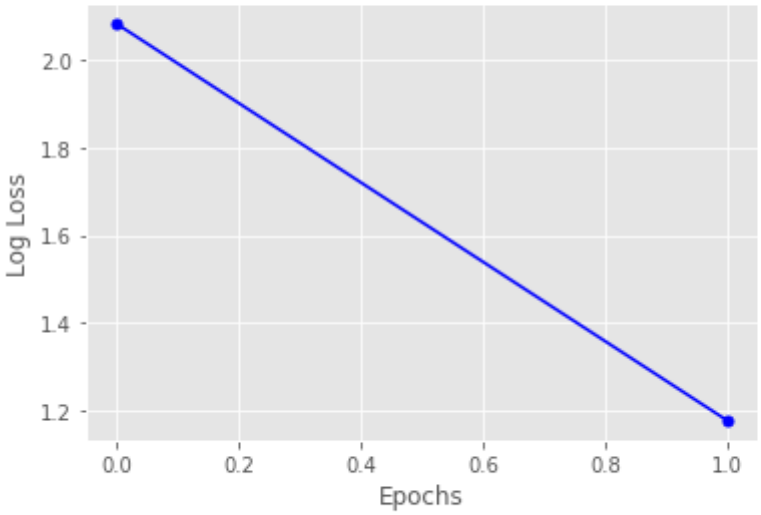



```
Training accuracy 0.82605
Validation accuracy 0.82768
```

He with sigmoid is clearly not performing as well as Xavier

```
fixup sigmoid
```

```
100% |████████████████████████████████| 100/100 [00:03<00:00, 29.95epoch/s]
```



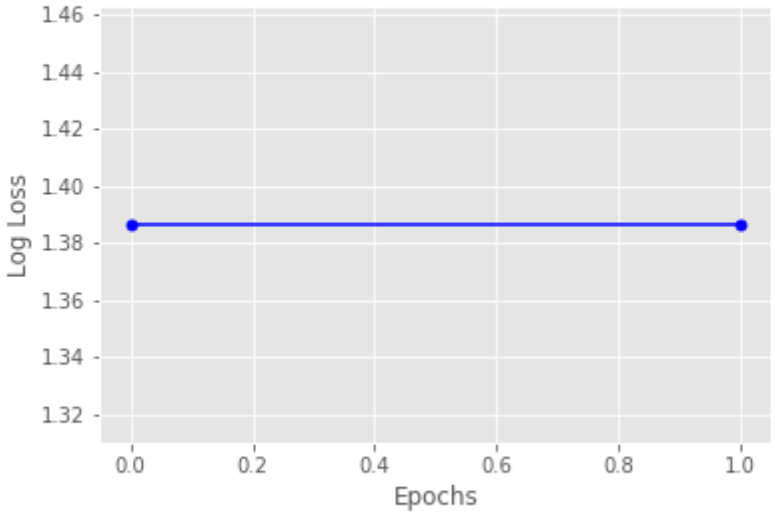

```
Training accuracy 0.56749
Validation accuracy 0.56792
```

Fixup is definitely not too great on this kind of task

Benchmarks (ReLU activation)

```
zeros relu
```

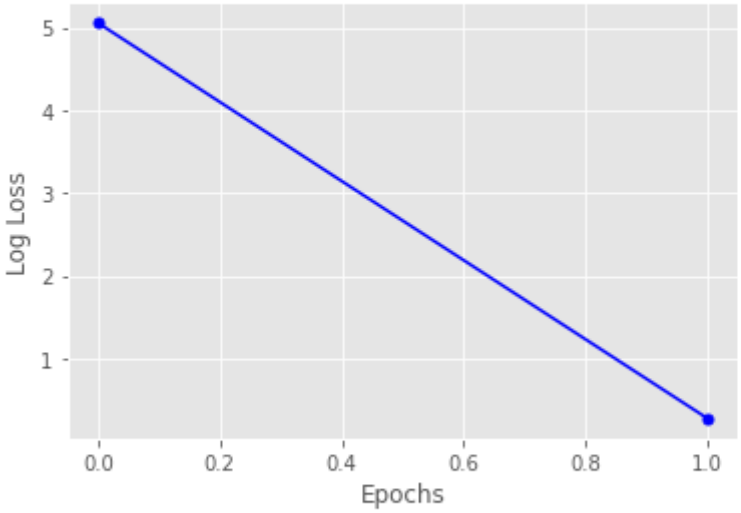
```
100% 100/100 [00:02<00:00, 34.94epoch/s]
```




```
Training accuracy 0.25
Validation accuracy 0.25
```

```
random relu
```

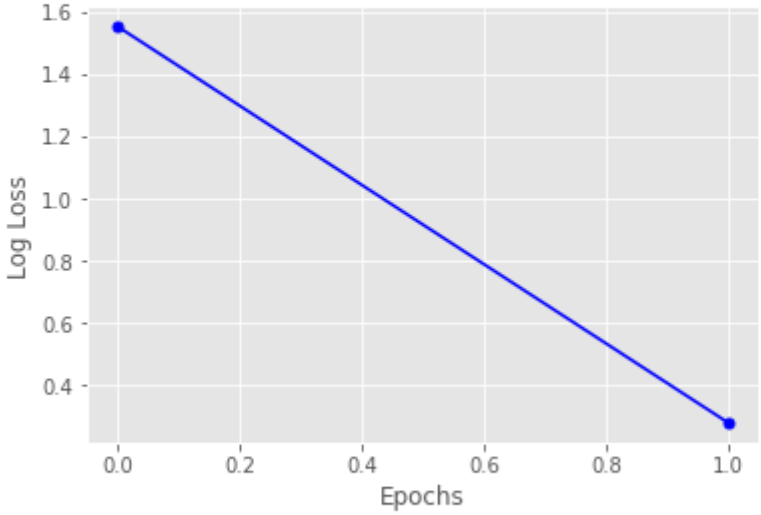
```
100% 100/100 [00:02<00:00, 35.35epoch/s]
```




```
Training accuracy 0.96129
Validation accuracy 0.96228
```

```
xavier relu
```

```
100% 100/100 [00:02<00:00, 35.01epoch/s]
```

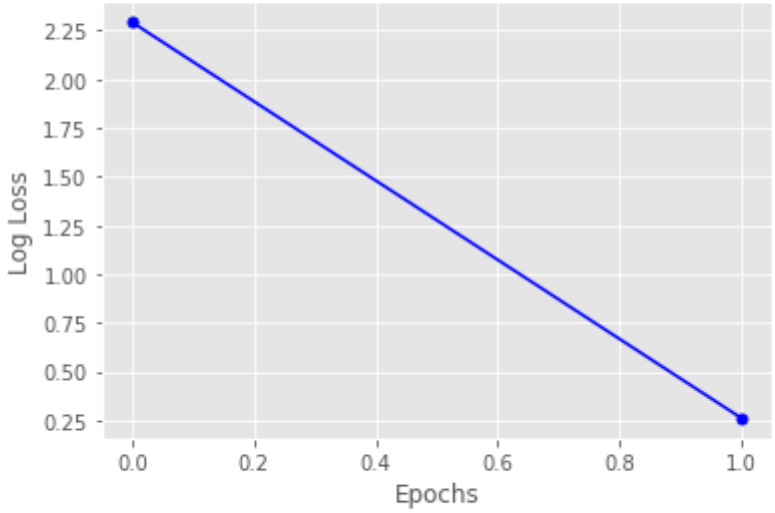



```
Training accuracy 0.9892
Validation accuracy 0.98936
```

Quite obviously fails to work as good as He Init with ReLU

```
he relu
```

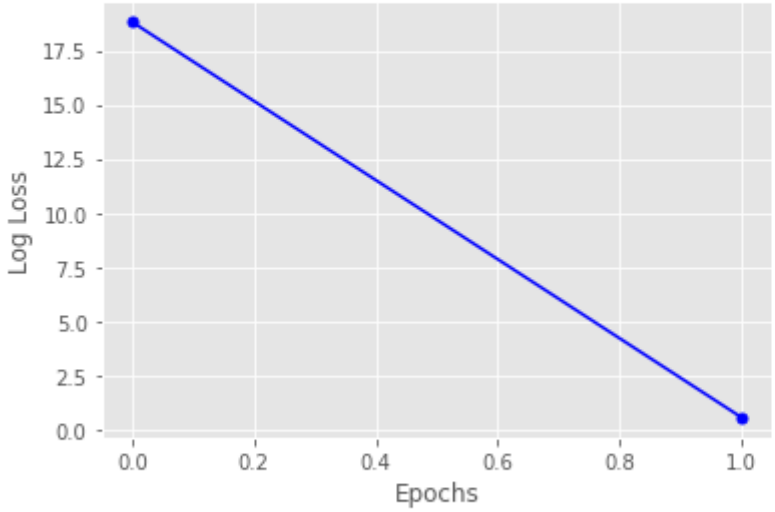
```
100% 100/100 [00:02<00:00, 34.53epoch/s]
```




```
Training accuracy 0.97971
Validation accuracy 0.97912
```

```
fixup relu
```

```
100% ████████████████████ 100/100 [00:02<00:00, 34.15epoch/s]
```




```
Training accuracy 0.92289
Validation accuracy 0.92404
```

Fixup shows a bit worse performance than He with ReLU, but still pretty solid.

In spite of the not as impressive performance on our test data (as I am not using the deep residual networks in my research due to limited compute availability), Fixup Init is still outperforming most of the traditional initialization methods in machine translation task, according to Fixup Paper.

Conclusions

The problem of weight initialization in neural networks was investigated in the given work. Different initialization techniques were listed and tested on generated dataset. Frankly, Xavier and He Initialization methods are outperforming random and zeros initialization. Other methods such as FuseInit and Fixup Init are performing well on specific tasks and with suitable architectures of neural networks. The problems of vanishing and exploding gradient has been addressed as well. The data preparation can without a problem be implemented via Scikit-Learn tools and Pytorch framework.

Reference literature:

https://arxiv.org/pdf/1903.11482.pdf

https://arxiv.org/pdf/1912.05137.pdf

https://arxiv.org/pdf/2001.10509.pdf

https://arxiv.org/pdf/1704.08863.pdf

https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

https://towardsdatascience.com/implementing-different-activation-functions-and-weight-initialization-methods-using-python-c78643b9f20f

Hands on Machine Learning with Scikit-Learn & Tensorflow. Concepts, Tools, and Techniques to Build Intelligent Systems. Aurelien Geron. 2017 Edition.