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

Курсова робота

за спеціальністю „Інженерія програмного забезпечення”

Optimal initialization of neural networks

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

Виконав
студент 3 курсу
факультету інформатики
Кузьменко Д. О.

Aim of the course work

- Research the existing methods of weight initialization in neural networks, partially deep neural networks

This is achieved through the following tasks:

1) Begin with basic initialization techniques, i.e. zero/random initialization

2) Research the He and Xavier initialization

3) Consider recent approaches to the problem and test them out

4) Test the multiple hypotheses on generated data, taking into consideration different activation functions

Terminology

- Gradient of a function '*f*' – a vector field, whose value at certain point '*m*' is the vector of partial derivatives of *f* at *m*.
- Exploding/vanishing gradient problem – the problem of gradients becoming too small (dying out) or too big(exploding) in the process of training the neural network(namely backpropagation).
- Activation function – the function that defines the output of the node, given the input/set of inputs.

Initialization methods used in this course work

- Random initialization
- Zeros initialization
- He Normal (He initialization)
- Xavier initialization
- FixUp Init

```python
if initialize == "random":
    for i in range(1,self.l_cnt + 1):
        self.parameters["Weight" + str(i)] = np.random.randn(self.l_size[i-1],self.l_size[i])
        self.parameters["Bias" + str(i)] = np.random.randn(1,self.l_size[i])

elif initialize == "zeros":
    for i in range(1,self.l_cnt + 1):
        self.parameters["Weight" + str(i)]=np.zeros((self.l_size[i-1],self.l_size[i]))
        self.parameters["Bias" + str(i)]=np.zeros((1,self.l_size[i]))

elif initialize == "he":
    for i in range(1,self.l_cnt + 1):
        self.parameters["Weight" + str(i)] = np.random.randn(self.l_size[i-1],self.l_size[i])*np.sqrt(2/self.l_size[i-1])
        self.parameters["Bias" + str(i)] = np.random.randn(1,self.l_size[i])

elif initialize == "xavier":
    for i in range(1,self.l_cnt + 1):
        self.parameters["Weight" + str(i)]=np.random.randn(self.l_size[i-1],self.l_size[i])*np.sqrt(1/self.l_size[i-1])
        self.parameters["Bias" + str(i)]=np.random.randn(1,self.l_size[i])

elif initialize == "fixup":
    for i in range(1,self.l_cnt + 1):
        self.parameters["Weight" + str(i)]=np.random.randn(self.l_size[i-1],self.l_size[i])*np.sqrt(2/(self.l_size[i-1] ** (-0.5)))
        self.parameters["Bias" + str(i)]=np.random.randn(1,self.l_size[i])
```

Activation functions used in the research

Differentiable at 0:

- Hyperbolic tangent
- Sigmoid

Not differentiable at 0:

- ReLU
- Leaky ReLU

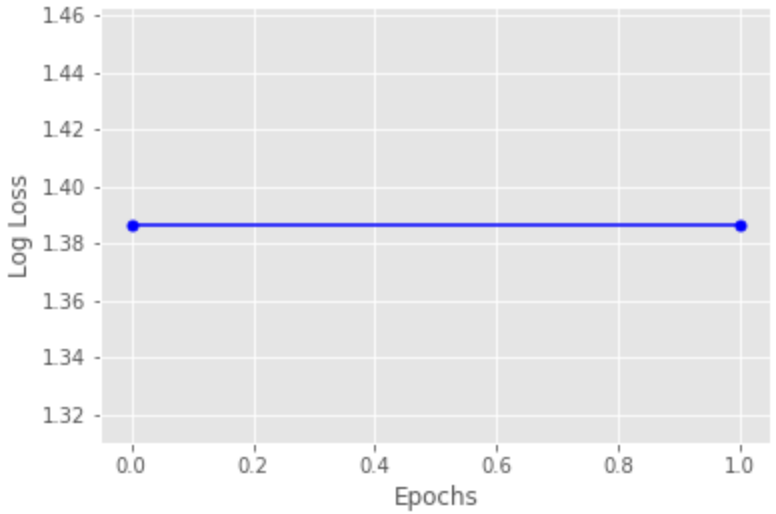
```
def forward_activation(self, X):
    if self.function == "sigmoid":
        return 1 / (1 + np.exp(-X))
    elif self.function == "tanh":
        return np.tanh(X)
    elif self.function == "relu":
        return np.maximum(0, X)
    elif self.function == "leaky_relu":
        return np.maximum(self.slope * X, X)

def grad_activation(self, X):
    if self.function == "sigmoid":
        return X * (1 - X)
    elif self.function == "tanh":
        return (1 - np.square(X))
    elif self.function == "relu":
        return 1 * (X > 0)
    elif self.function == "leaky_relu":
        d = np.zeros_like(X)
        d[X <= 0] = self.slope
        d[X > 0] = 1
        return d
```

Performance

zeros sigmoid

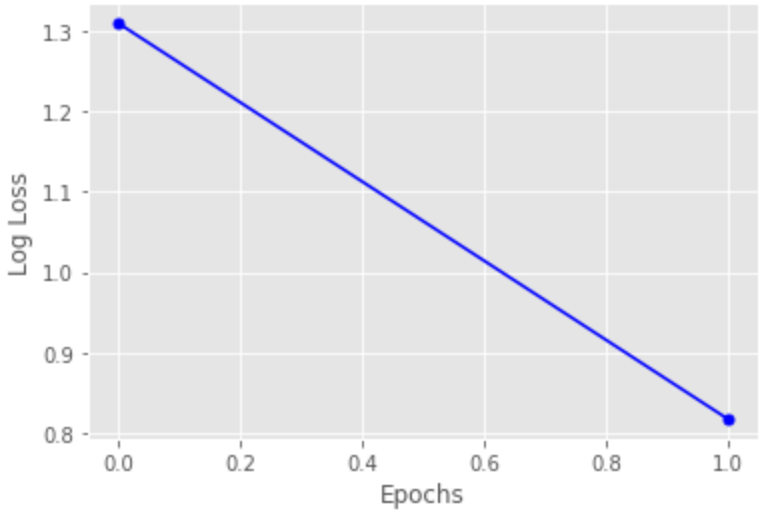
100% 100/100 [00:03<00:00, 28.84epoch/s]



Training accuracy 0.25
Validation accuracy 0.25

random sigmoid

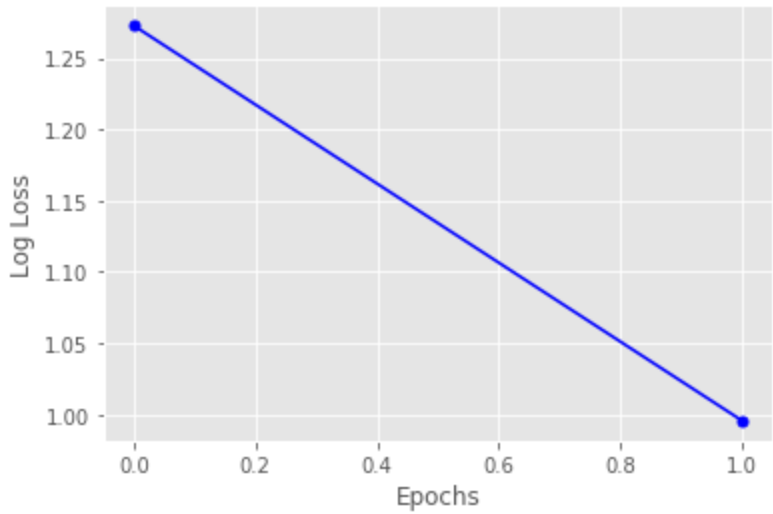
100% 100/100 [00:03<00:00, 29.69epoch/s]



Training accuracy 0.62871
Validation accuracy 0.63052

xavier sigmoid

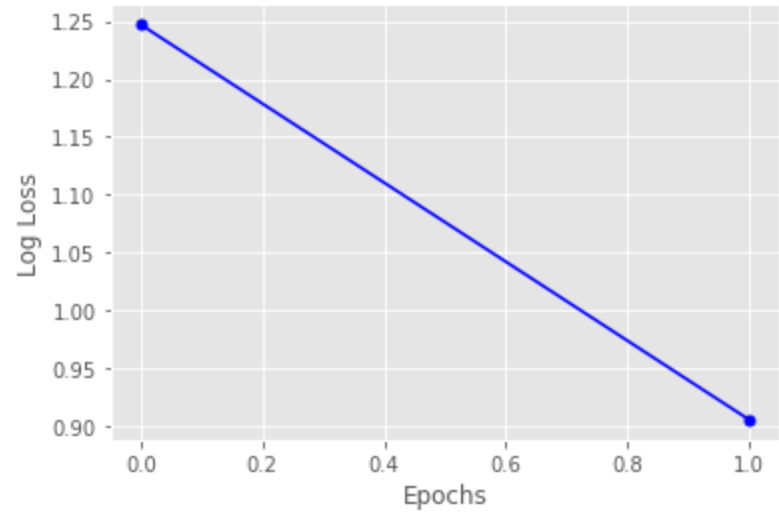
100% 100/100 [00:03<00:00, 29.01epoch/s]



Training accuracy 0.91324
Validation accuracy 0.9128

he sigmoid

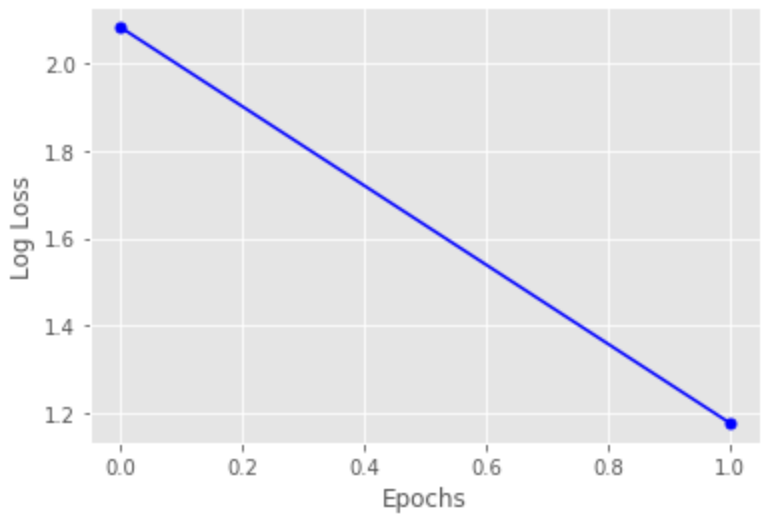
100% 100/100 [00:03<00:00, 29.10epoch/s]



Training accuracy 0.82605
Validation accuracy 0.82768

fixup sigmoid

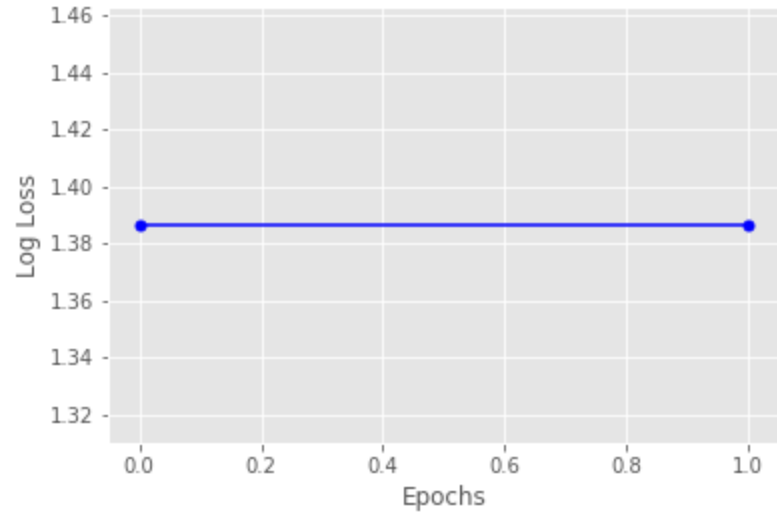
100% 100/100 [00:03<00:00, 29.95epoch/s]



Training accuracy 0.56749
Validation accuracy 0.56792

zeros relu

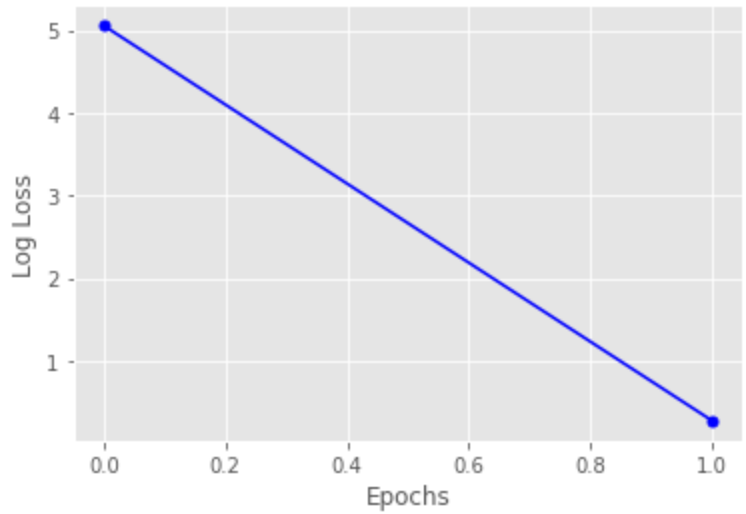
100% 100/100 [00:02<00:00, 34.94epoch/s]



Training accuracy 0.25
Validation accuracy 0.25

random relu

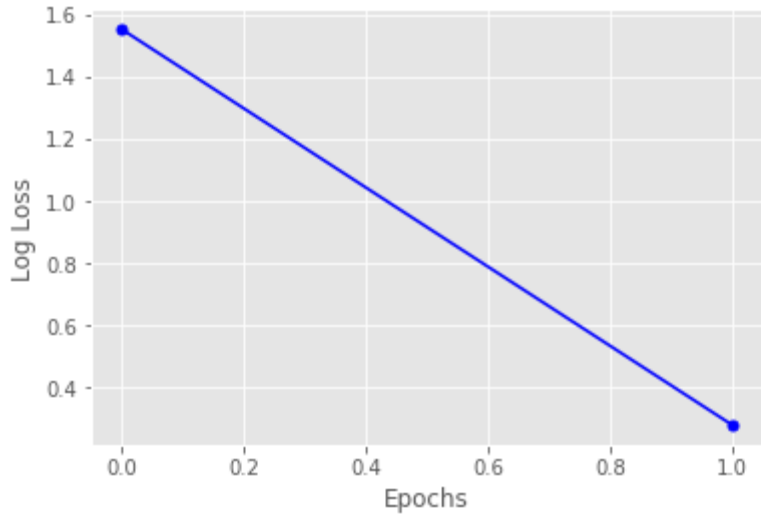
100% 100/100 [00:02<00:00, 35.35epoch/s]



Training accuracy 0.96129
Validation accuracy 0.96228

xavier relu

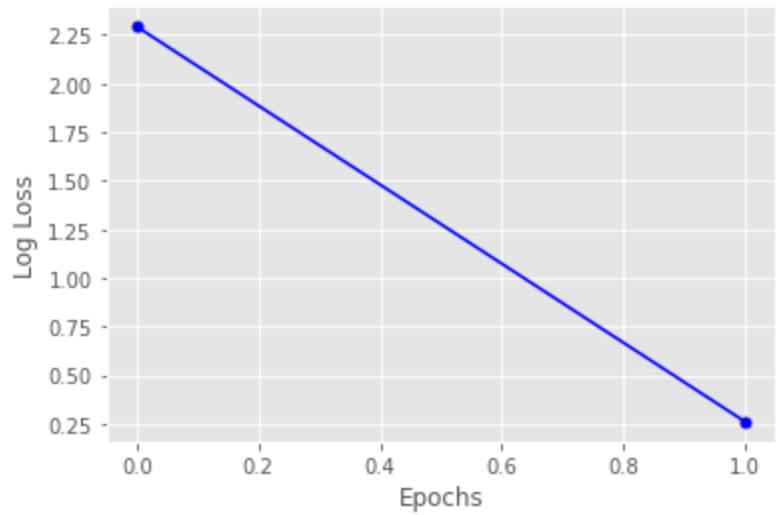
100% 100/100 [00:02<00:00, 35.01epoch/s]



Training accuracy 0.9892
Validation accuracy 0.98936

he relu

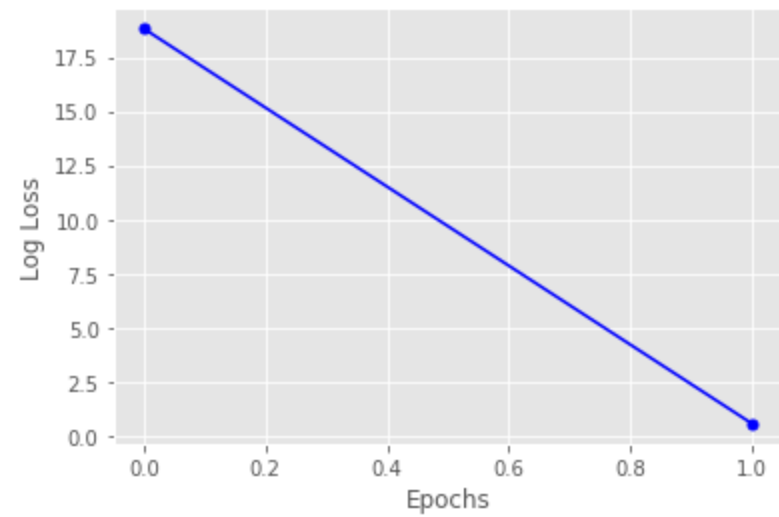
100% 100/100 [00:02<00:00, 34.53epoch/s]



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

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.

Thank you for your attention