Homework 4 - Benchmark ReLU SELU

Kevin Pratama, *20175046*

***Abstract*—The Convolutional Neural Network (CNN) has shown an excellent performance in computer vision task. Applications of CNN include image classification, image semantic segmentation, object detection in images, etc. This project will observe 4 different choice of hyperparameter on their effect to CNN with ReLU and newly propose SELU as activation function.**

I. Normalization

In machine learning and deep learning, we can handle various types of datasets such as image, sound and text. These datasets can have multiple dimensions or scales. A feature that has smaller scale might play a tiny role compare to the feature that has bigger scale. However, both feature may contain important information that will be useful for the task we want to perform. To deal with this situation, we may want to normalize the features independently to the same scale, so they contribute equally while performing the task.

One of the methods to do normalization is Z-score normalization. This method is widely used for normalization in many machine learning algorithms (e.g., support vector machines, logistic regression, and neural networks). The result of normalization is that the features will be rescaled so that they will have the properties of a standard normal distribution with zero-mean and unit-variance. Standardizing is not only important if we are comparing measurements that have different units, but it is also a general requirement for many machine learning algorithms.

$$z = \frac{x - \mu}{\sigma}$$

The general method of Z-score calculation (z) is to determine the distribution mean (μ) and standard deviation (σ) for each feature. Next subtract the mean from each feature then divide the values (mean is already subtracted) of each feature by its standard deviation.

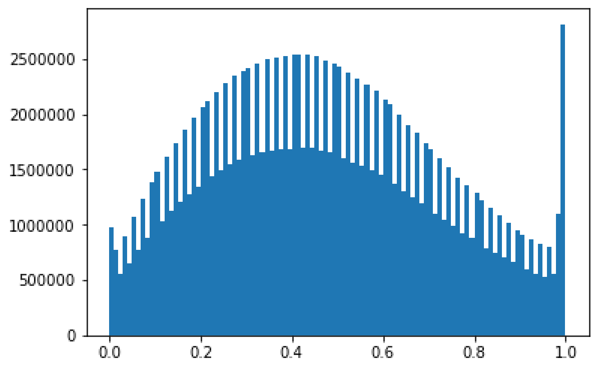

Fig. 1: Cifar10 Distribution before performing normalization

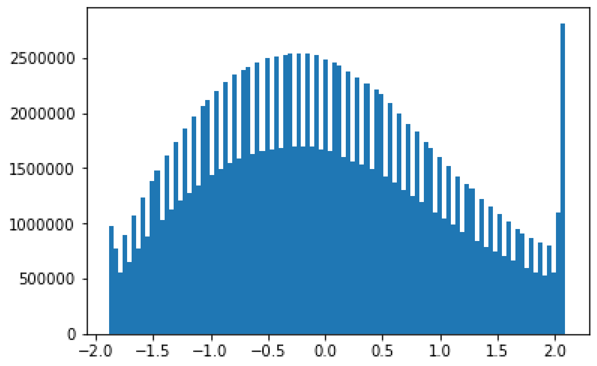

Fig. 2: Cifar10 distribution after performing normalization

II. Initialization

Training deep neural network is known to be difficult. One of the reasons for this difficulty is due to lots of hyperparameter (learning rate, number of hidden layer, loss function, etc.) to tune in deep network. Since 2006, several research have successfully trained neural network with result that showing the superiority of deep network compare to shallow network.

When training deep neural network, initializing the weights of the network can be the determine factor of successful convergence. Initialize the right weight can also lead to a faster convergence. Initialize weight with small value will make the variance of the input signal shrink as it passes through layer in the network. Eventually, the input will drops to a very small value it become useless for learning. Similarly, if the weight is initialize with a large value the variance of input data tends to increase rapidly with every passing layer and become too

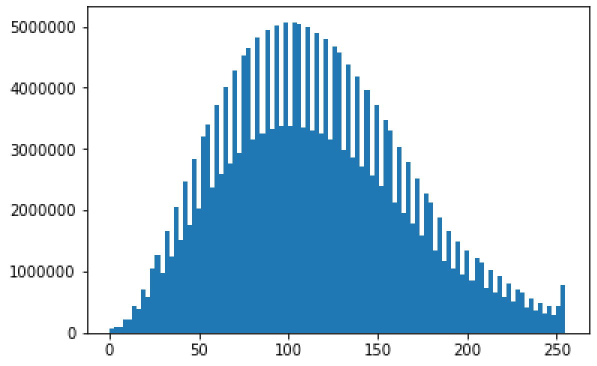

Fig. 3: SVHN Distribution before performing normalization

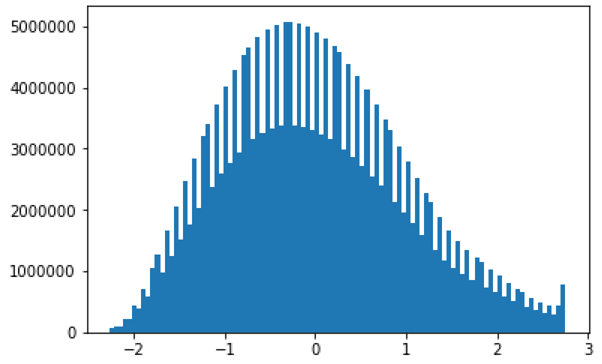

Fig. 4: SVHN distribution after performing normalization

massive to be useful.

Initializing the network with the right weights (not too small or too big) is very important. Unfortunately it seems to be a random process as we don't know the weight that will work for our data. A group of researcher (Xavier et al., 2010) propose a new method of initialization that make sure the weights are initialize properly and have a reasonable range. The main idea of Xavier initialization is to maintain the variance of the weight to remain the same as it passes through layers. This help the network to keep the input signal from exploding to a high value or vanishing to zero.

$$Var(w_i) = \frac{2}{n_{in} + n_{out}}$$

Eq1: Xavier Initialization Method

Xavier initialization is derive based on the assumption that the neural network are using symmetric activation function. It also assumed that the the weights are initialized independently and both input and weights have zero mean.

Building on Xavier initialization, He et al. come up with different initialization method. Contrary to Xavier initialization, He initialization derive based on a non linear activation function and assume that both input and weight do not have zero mean. This lead to a different conclusion to Xavier initialization.

$$Var(w_i) = \frac{2}{n_{in}}$$

Eq2: He Initialization Method

This project evaluate the effect of Xavier and He initialization on Convolution Neural network with ReLU or SELU as activation function. The result can be seen in TABLE I and TABLE II. Additionally, this project also evaluate the initialization method that is used in Self-Normalizing Neural Networks (SNN) paper where the variance calculation of weight is given in the formula below.

$$Var(w_i) = \frac{1}{n_{in}}$$

Eq3: SNN Initialization Method

	Xavier	He	SNN init
ReLU	73.61%	73.97%	75.13%
SELU	74.64%	73.98%	75.77%

TABLE I: Test Accuracy on Cifar-10

	Xavier	He	SNN init
ReLU	91.56%	91.33%	92.65%
SELU	90.68%	91.80%	91.85%

TABLE II: Test Accuracy on SVHN

Without surprise, the initialization method from SNN paper works best for SELU network. According to the author, the initialization for SNN will help the output of SNN model to stay at normalized point at zero mean and unit variance which could be the reason it is best suited for SeLU network. The interesting point is that SNN initialization also work for ReLU network with test accuracy outperform SELU network on SVHN dataset.

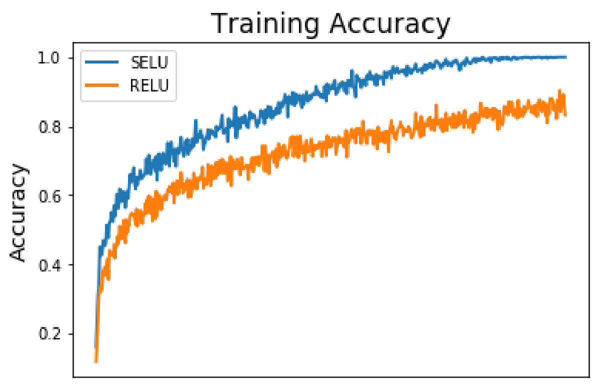

Fig. 5: Training Accuracy on Cifar-10 using Xavier Initialization

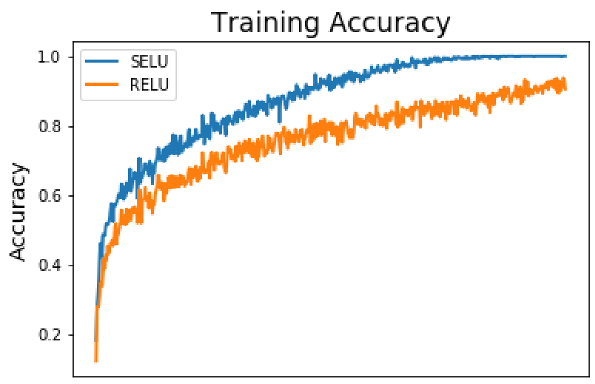

Fig. 6: Training Accuracy on Cifar-10 using He Initialization

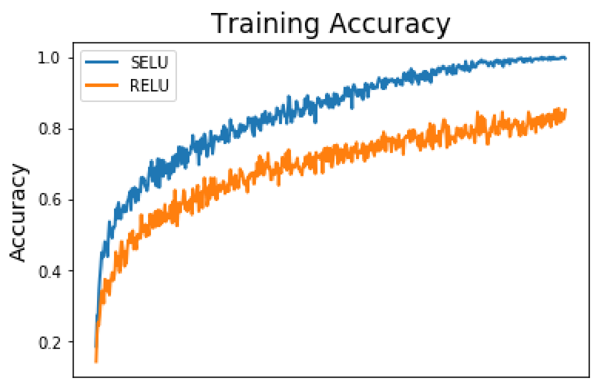

Fig. 7: Training Accuracy on Cifar-10 using SNN Initialization

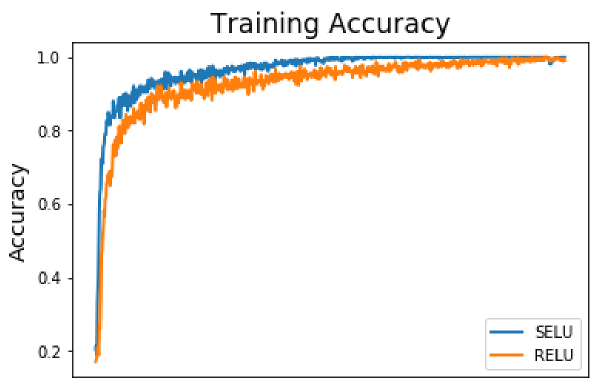

Fig. 8: Training Accuracy on SVHN using Xavier Initialization

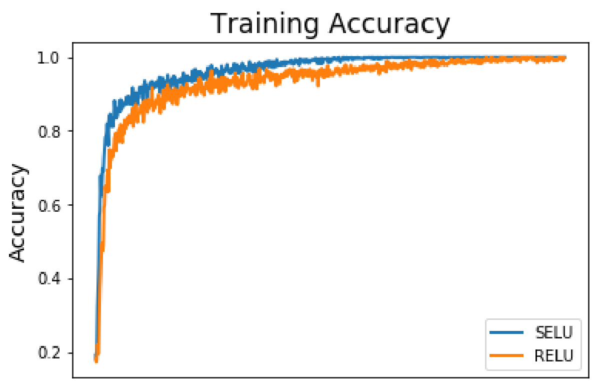

Fig. 9: Training Accuracy on SVHN using He Initialization

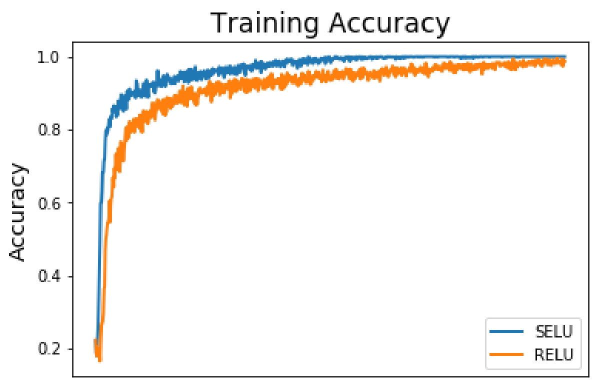

Fig. 10: Training Accuracy on SVHN using SNN Initialization

III. Network Configuration

This project compare 4 different network configuration. The network configuration is given in TABLE III

	Model 1	Model 2	Model 3	Model 4
Conv	64, 5x5, 2	64, 5x5, 2	64, 5x5, 2	64, 5x5, 2
			64, 5x5, 2	64, 5x5, 2
			64, 5x5, 2	64, 5x5, 2
			64, 5x5, 2	64, 5x5, 2
Pool	3x3, 2	3x3, 2	3x3, 2	3x3, 2
Conv	64, 5x5, 2	64, 5x5, 2	64, 5x5, 2	128, 5x5, 2
		64, 5x5, 2	64, 5x5, 2	128, 5x5, 2
		64, 5x5, 2	64, 5x5, 2	128, 5x5, 2
		64, 5x5, 2	64, 5x5, 2	128, 5x5, 2
			64, 5x5, 2	128, 5x5, 2
			64, 5x5, 2	128, 5x5, 2
Pool	3x3, 2	3x3, 2	3x3, 2	3x3, 2
Conv	128, 3x3, 2	128, 3x3, 2	128, 3x3, 2	256, 3x3, 2
	128, 3x3, 2	128, 3x3, 2	128, 3x3, 2	256, 3x3, 2
	128, 3x3, 2	128, 3x3, 2	128, 3x3, 2	256, 3x3, 2
		128, 3x3, 2	128, 3x3, 2	256, 3x3, 2
		128, 3x3, 2	128, 3x3, 2	256, 3x3, 2
		128, 3x3, 2	128, 3x3, 2	256, 3x3, 2
			128, 3x3, 2	256, 3x3, 2
			128, 3x3, 2	256, 3x3, 2
Pool	3x3, 2	3x3, 2	3x3, 2	3x3, 2

TABLE III: Network Configuration

	Model 1	Model 2	Model 3	Model 4
ReLU	73.97%	69.88%	68.70%	70.12%
SELU	75.77%	76.23%	77.28%	80.19%

TABLE IV: Test Accuracy on Cifar-10

	Model 1	Model 2	Model 3	Model 4
ReLU	91.33%	92.51%	92.99%	93.47%
SELU	91.85%	92.77%	93.46%	94.17%

TABLE V: Test Accuracy on SVHN

From TABLE IV reader can observe that SELU network gain a better test accuracy as the network goes deeper and bigger on both dataset. This behavior is not seen in ReLU network. A deeper ReLU network doesn't get a significant improvement in term of accuracy with accuracy goes down in model 2 and 3 on Cifar-10 dataset. Similar behavior can be observe in TABLE V where deeper SELU network get a bigger improvement in term of test accuracy compare to ReLU network.

One hypothesis regarding SELU performance is its ability to normalize the layer output. One week link of deep neural network is its vulnerability of gradient issues as the network goes deeper and bigger. SELU ability to normalize its output make a deep network less prone to gradient issue, hence the performance improvement.

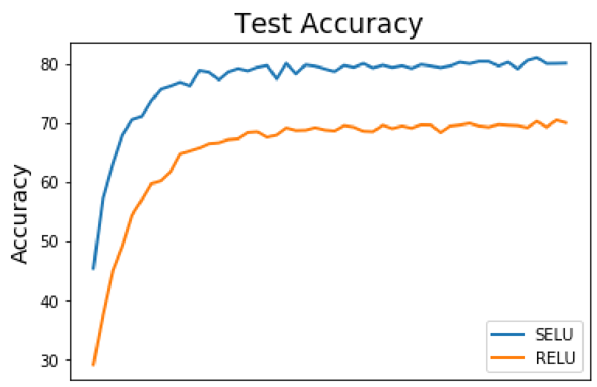

Fig. 11: Test Accuracy on Cifar-10 using model 4

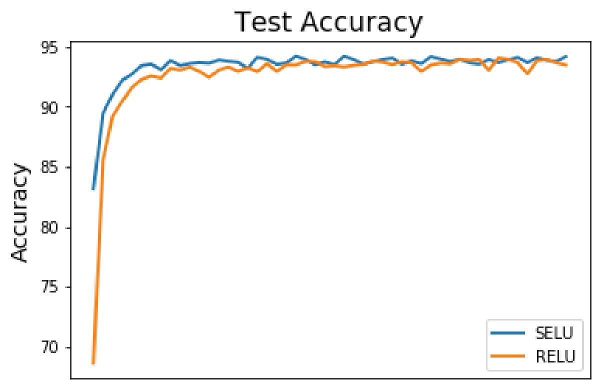

Fig. 12: Test Accuracy on SVHN using model 4

IV. Gradient Optimization Techniques

This section will compare two Gradient Optimization Techniques. The first technique is Momentum. Momentum is a method to help accelerate gradient descent to reach convergence. It does that by navigating gradient descent along the relevant direction and dampens oscillations. Momentum adds a fraction (γ) from the update from the past step to the current update.

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta)$$

$$\theta = \theta - v_t$$

Eq3: Momentum Update Rule

The second optimization techniques is Adam. Adam stands for Adaptive Moment Estimation. It is a method that computes adaptive learning rate for each parameter. Adaptive learning rate means it adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. Adam also keep both exponentially decaying average of past squared gradients and past gradient with parameter β_1 and β_2 control the decay rates of these averages.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla J(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla J(\theta_t))^2$$

Eq4: Exponentially decaying average of past gradients and past squared gradient

m_t and v_t are estimates of the first moment and the second moment of the gradients. Both are biased towards zero as it initialized with 0. The author of Adam counteract this bias by computing bias-correction.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Eq5: Bias Correction formula

Then the bias correction is used to update parameter.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

Eq6: Adam Update Rule

This project set the momentum term (γ) with 0.9. For Adam optimization, this project will use the default value proposed by the Adam author which is 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. The result can be seen in TABLE VI and VII.

	Momentum	Adam
ReLU	53.61%	73%
SELU	52.16%	75.89%

TABLE VI: Test Accuracy on Cifar-10

	Momentum	Adam
ReLU	79.75%	91.12%
SELU	82.27%	91.91%

TABLE VII: Test Accuracy on SVHN

From this experiment result, it is obvious that Adam optimizer produce a better result compare to the Momentum optimizer. The author of Adam describe this method as combining the advantages of two other optimization method, AdaGrad and RMSProp. Not only adapting the parameter learning rate on the average first moment as in RMSProp, Adam also use the average of the second moments of the gradient. This combination has led to a notable improvement from Momentum optimizer.

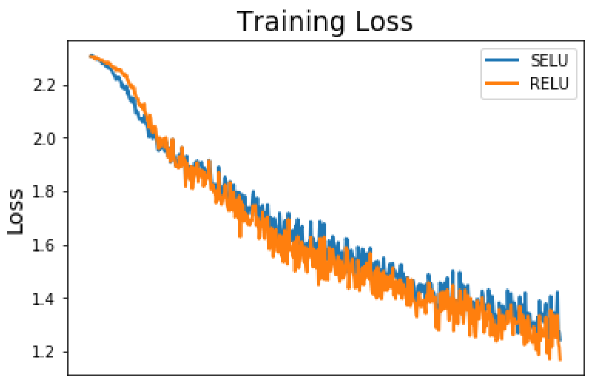

Fig. 13: Training Loss on Cifar-10 with Momentum

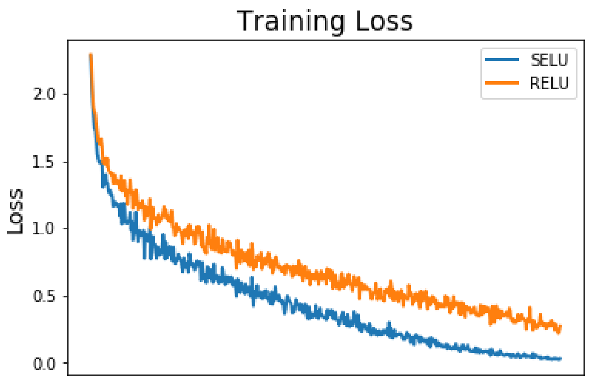

Fig. 14: Training Loss on Cifar-10 with Adam

V. Regularization

One of the cause for a poor performance in deep learning is overfitting. Overfitting happens when a model learn the training data too well it negatively impact the model ability to generalize data and make model perform poorly on new unseen data. This project will observe the impact of two regularization method on CNN model.

The first regularization method is L2 regularization. L2 regularization add a penalty equal to the sum of the squared of value of the weights to the error term. L2 regularization will force the parameters to be small since the bigger the L2 regularization penalty, the smaller the weight.

$$\varepsilon_{L2} = \varepsilon + \lambda\Sigma_{i=1}^{k} w_i^2$$

Eq6: L2 Regularization

The second regularization method is dropout. As a neural network learns, neuron settle to their context and tuned for specific specialization. The network become relying on this specialization which will lead to overfitting the training data. Dropout randomly ignore neurons during training which means

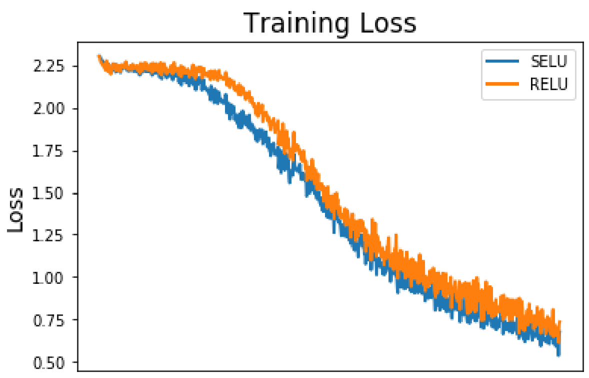

Fig. 15: Training Loss on SVHN with Momentum

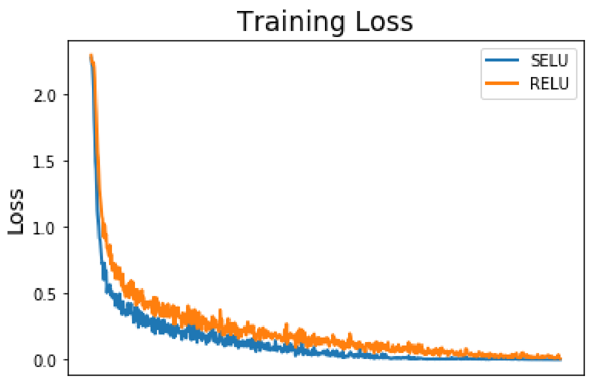

Fig. 16: Training Loss on SVHN with Adam

the neurons contribution is temporally removed. When neurons are randomly ignore during training, the other neurons will step in to handle the representation required to make prediction. This make the network became less dependent to the specific neurons, result in a network with better generalization and less likely to overfit the training data.

	L2	Dropout
ReLU	73.48%	75.78%
SELU	75.95%	76.63%

TABLE VIII: Test Accuracy on Cifar-10

	L2	Dropout
ReLU	91.68%	93.04%
SELU	91.43%	92.16%

TABLE IX: Test Accuracy on SVHN

The reader will notice a superior performance of dropout compare to L2 regularization. Randomly shut-off neurons during training prevent co-adaptation among them, therefore forcing them to be less dependent to other neuron to correct

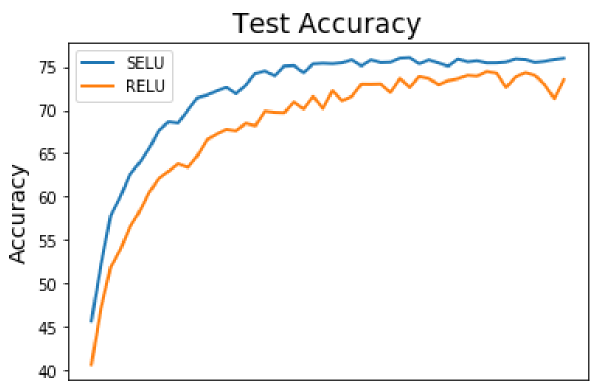

Fig. 17: Test Accuracy on Cifar-10 using L2 Regularization

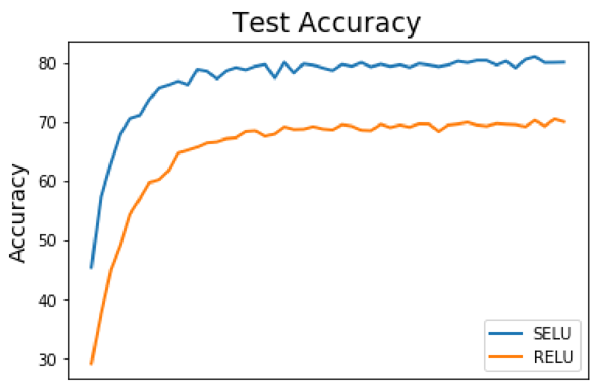

Fig. 18: Test Accuracy on Cifar-10 using Dropout

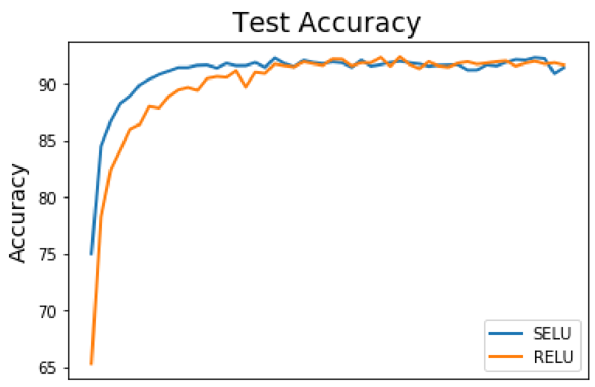

Fig. 19: Test Accuracy on SVHN using L2 Regularization

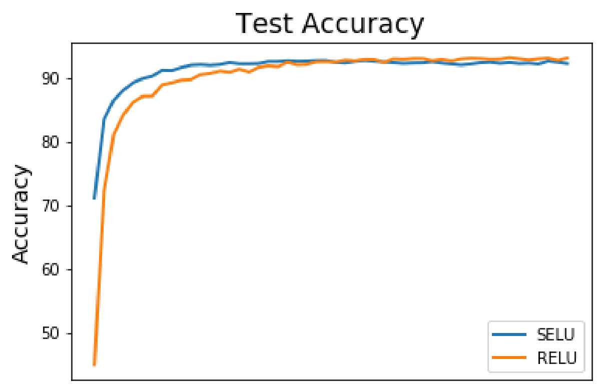

Fig. 20: Test Accuracy on SVHN using Dropout

its mistake. Dropout allow a single network to model a large number of sub-networks in inexpensive way, making it a more robust regularization than L2 regularization.

VI. Activation Function

This project has compared ReLU and SELU network with four hyperparameter (initialization method, network configuration, optimization technique and regularization). All four hyperparameter give various effect (negatively and positively) on both network in term of test accuracy and speed of convergence.

One clear advantage of SELU network over ReLU network that is observed from this project is its speed of convergence regardless the choice of hyperparameter. The only time that ReLU network beat SELU network is when Momentum is selected as optimization technique. But since Adam optimization has been proven to be a more robust optimization technique, this result is ignorable.

One more advantage of SELU network is its ability to use a big and deep network and gain a better test accuracy. This is clearly observed as the test accuracy become better with deeper network with the deepest SELU network in this project gain the best test accuracy. Meanwhile there is no obvious sign of this behavior with ReLU network.

References

[1] Glorot, X.; & Bengio, Y. (2010), *Understanding the difficulty of training deep feedforward neural networks*, In Proc. AISTATS, volume 9, pp. 249256 .

[2] K, He.; X, Zhang.; S, Ren.; & J, Sun. (2015), *Delving deep into rectifiers: surpassing human-level performance on imagenet classification*, In Proceedings of the IEEE International Conference on Computer Vision (2015), 10261034 .

[3] Klambauer, G.; Unterthiner, T.; Mayr, A. & Hochreiter, S. (2017), *Self-Normalizing Neural Networks*, CoRR abs/1706.02515 .

[4] Qian, N. (1999), *On the momentum term in gradient descent learning algorithms*, Neural networks : the official journal of the International Neural Network Society, 12(1):145151 .

[5] Kingma, D.; & Ba, J. (2015), *Adam: a Method for Stochastic Optimization*, International Conference on Learning Representations, pages 113 .

[6] Hinton, G.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; & Salakhutdinov, R. (2012), *Improving neural networks by preventing coadaptation of feature detectors*, http://arxiv.org/abs/1207.0580