Comparing Different Training Methods for Multiple Deep Convolutional Architectures

Giorgos Tziafas [a] Xabi Krant [b]

[a] *University of Groningen, student number: 3913171*
[b] *University of Groningen, student number: 2955156*

Abstract

In this report we perform a comparative study of different training methods in state-of-the-art deep achitectures for the multiclass image recognition task in the CIFAR10 dataset. We compare different combinations of activation units and optimization algorithms. We do not wish to find the optimal settings in order to replicate state-of-the-art results, but rather provide a practical intuition about what methods work better for each architecture. In our results we discover that the unpopular Gaussian Error Linear Unit (GeLU) non-linearity exceeds the performance of other activation functions across the more complex architectures.

1 Introduction

Deep Convolutional Neural Networks (CNNs) have been the standard image classification tool since the groundbreaking results reported with the publication of the AlexNet architecture [4] in the 2012 ILSVRC competition. In our study, we focus on *AlexNet* as well as two other very popular vision models, *VGG* and *ResNet*, architectures which achieve both high accuracy and computational efficiency. Our dataset of choice is the *CIFAR10*, one of the benchmark datasets for multi-class classification tasks, comprised of 60000 labeled $32x32$ color images, representing 10 different target classes. The raw data are preprocessed into a trainable format. For our experiments, we establish a baseline setup for each model. Then, we experiment using combinations of different activation units and optimization algorithms and report the results. For the training we used the *Google Collaboratory* provided GPUs. All software is implemented in the *Pytorch* framework and can be found `here`.

2 Architectures

2.1 AlexNet

As mentioned above, this is the model that established deep CNNs for image classification, being the first historically to achieve such good results on the *ImageNet* dataset. Our network consists of 5 convolutional layers, 3 max-pooling, 2 dropout layers and 3 fully connected layers. For our experiments, we modify the last layer from the standard architecture to apply softmax classification for simply 10 classes.

2.2 ResNet

To improve network architectures, many researchers have relied on creating deeper networks. The problem with this is that the back-propagated gradients become infinitely small, as a result of repeated multiplications. The performance of a network degrades rapidly when the network goes deeper. To solve this, ResNet [1] used "identity shortcut connections" or skip connections implemented in basic residual blocks, an idea that has been used before in Highway Networks [6]. In our architecture's instance, the

network consists of a total of 19 convolution layers, always followed by batch normalization layers, implemented in a total of 4 residual blocks before the softmax classification.

2.3 VGG

The deepest VGG architecture [5] discussed in the paper consists of 16 convolutional layers followed by 3 fully connected layers, but they also tested ones with only 8 conv. layers. One of the appealing aspects of VGG is the uniform architecture. The architecture uses a lot of filters with a receptive field of 3x3, which is opposed to previous architectures that mostly relied on 7x7 receptive fields. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers. The width of the network is 64 in the first layer, which is doubled after the pooling layers, to 512 at the end.

3 Experiments

We first perform the following preprocessing steps in the raw dataset in order to prepare out data for the networks: (1). Split the dataset $90\% - 10\%$ into a *training* and a *validation* set. (2). Reshape all image data to the form of RGB $64x64$ images (3). Convert image data to floating points and normalize them over a standardized normal distribution with mean $\sigma = 0.485$ and standard deviation of $std = 0.224$. (4). Shuffle and batch all data into batches of 256 examples and push them into tensor objects. As in all multi-class classification tasks we are using the *cross-entropy* loss function to compute gradients for backpropagation. The baseline model for all architectures consists of *Rectified Linear Unit (ReLU)* activations, the *Stochastic Gradient Descent* with momentum optimizer and the default `torchvision` regularization techniques for each network. *AlexNet* is trained and validated for 25 and the other models for 12 epochs, values that were chosen as illustrative of each model's training capacity, after several experimentation. For each epoch, the corresponding errors are reported. Then, each model is tested once in the validation set. The *accuracy* for testing is recorded against the top-1 most probable label.

3.1 Activation Units

The most popular activation function, *ReLU*, is compared with three different variations of it, regarding the properties of the function in the negative input space. The formulas for all Linear Units are shown below and their corresponding graphs are demonstrated in Figure 1, along with the training error for all architectures.

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}, \quad ELU(x) = \begin{cases} x, & x > 0 \\ a \cdot (e^x - 1), & x \leq 0 \end{cases}, Leaky\_ReLU(x) = \begin{cases} x, & x > 0 \\ ax, & x \leq 0 \end{cases}$$

,

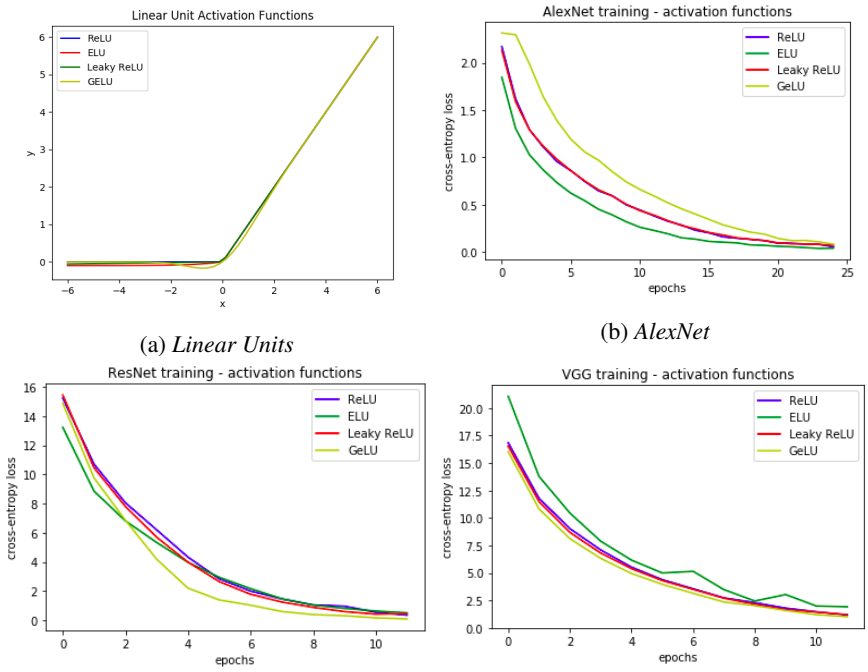
$$GeLU(x) = \sigma x \cdot (1 + tanh(\sqrt{\frac{2}{\pi}}(x + vx^3)))$$

All the variations aim at giving a slope to the curve, driving the gradients on the negative space instead of keeping them unaffected. For the *Leaky ReLU* we choose a factor of $\alpha = 0.01$, for the exponential *LU* a factor of $\alpha = 1$ and for the *GeLU* [2], we choose the proposed pair of mean $\sigma = 0.5$ and standard variation $v = 0.0044715$. The results are reported in Table 1.

3.2 Optimization Algorithms

The standard choice for our baseline model is the *Stochastic Gradient Descent (SGD)* optimizer, enhanced with *momentum=0.93* so that derivatives are computed over exponentially weighted averages, thus minimizing the effect of noisy gradients [7]. We use this optimizer with a learning rate of 0.1 and we add a weight decay of 0.01, in order to explore it's regularization potential. Furthermore, we experiment across all architectures with two other standard optimizers for our baseline setup:

Root Mean Square Error (RMS) prop A variation of the Gradient Descent paradigm that attempts to combines the robustness of Rprop by using signed gradients with averaging over mini-batches. The main idea is to keep the moving average of squared gradients and use it to normalize gradients by the median.



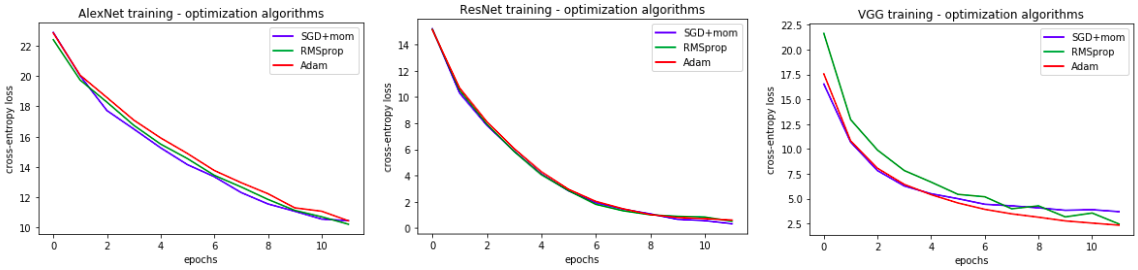
(a) *Linear Units* (b) *AlexNet*

(c) *ResNet* (d) *VGG*

Figure 1: *(a)*: Variations of Linear Unit activations, *(b),(c),(d)*: Training error for all activation variations across the baseline setup architectures.

Adam Another adaptive method [3], aiming at the combination of maintaining per-parameter weights (as previously in *Adagrad*) with the benefits of *RMSprop*. Due to it's computational efficiency, memory requirements, robustness in noisy/sparse gradients and easy hyper-parameter interpretation, *Adam* is considered the go-to algorithm for many computer vision applications.

The adaptive algorithms are able to train the model in fewer epochs, with a learning rate of 0.0001 and the same weight decay. The training errors for all three different optimizers across all models are demonstrated in Figure 2. We train the models for a total of 12 epochs (which results in underfitting in the case of the first model) in order to demonstrate their relative performance while simultaneously saving computing time. Results are reported in table 2



(a) *AlexNet* (b) *ResNet* (c) *VGG*

Figure 2: Training error for all optimization algorithm variations across the baseline setup architectures.

4 Results and Discussion

Reflecting on our experiments and the results of the two tables, we highlight the following:

	AlexNet		ResNet		VGG	
ReLU	96	74	100	74	95	80
Leaky ReLU	96	74	99	74	95	80
ELU	98	75	98	75	95	80
GELU	**98**	**75**	**99**	**75**	**95**	**80**

Table 1: Final train and test percentage accuracies reported for all baseline setup models with different activation functions for the optimal number of epochs in each model.

	AlexNet		ResNet		VGG	
SGD+mom	64	62	99	79	95	81
RMSprop	66	61	99	79	97	81
Adam	**65**	**64**	**99**	**80**	**97**	**81**

Table 2: Final train and test percentage accuracies reported for all baseline setup models with different optimization algorithms after a common total of 12 epochs.

Architectures As expected, the more sophisticated architectures of *ResNet* and *VGG* outperform the basic model of *AlexNet*, as the last one needs more than double the training time to achieve the same performance. We notice that the *VGG* architecture, which strongly utilizes batch normalization layers after every convolution layer, is the one that produces higher results, also immune to hyperparameter tuning.

Regularization We notice the effect of weight decay by observing the first rows of the two tables, where in the second case there is an increase in test accuracy due to the weight decay that was employed inside the optimizers.

Activations In all cases, given enough training all activations are able to produce the same results. However, they differ strongly on the training itself, as seen in Figure 1. Apart from the case of *AlexNet*, where the *ELU* activation is the faster and smoother, in the other cases the *GELU* function is the one that converges faster and without any oscillations to the optimized loss minimum. This is due to batch normalization that is applied in the last two models.

Optimizers We observe that all optimizers, when tuned at the correct setting, are able to achieve the same results, however with *Adam* being slightly faster and smoother in optimizing the *VGG* network.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[5] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[6] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[7] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.