FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS (ELUS)

Djork-Arné Clevert, Thomas Unterthiner & Sepp Hochreiter
Institute of Bioinformatics
Johannes Kepler University, Linz, Austria
{okko,unterthiner,hochreit}@bioinf.jku.at

ABSTRACT

We introduce the "exponential linear unit" (ELU) which speeds up learning in deep neural networks and leads to higher classification accuracies. Like rectified linear units (ReLUs), leaky ReLUs (LReLUs) and parametrized ReLUs (PReLUs), ELUs alleviate the vanishing gradient problem via the identity for positive values. However ELUs have improved learning characteristics compared to the units with other activation functions. In contrast to ReLUs, ELUs have negative values which allows them to push mean unit activations closer to zero like batch normalization but with lower computational complexity. Mean shifts toward zero speed up learning by bringing the normal gradient closer to the unit natural gradient because of a reduced bias shift effect. While LReLUs and PReLUs have negative values, too, they do not ensure a noise-robust deactivation state. ELUs saturate to a negative value with smaller inputs and thereby decrease the forward propagated variation and information. Therefore ELUs code the degree of presence of particular phenomena in the input, while they do not quantitatively model the degree of their absence.

In experiments, ELUs lead not only to faster learning, but also to significantly better generalization performance than ReLUs and LReLUs on networks with more than 5 layers. On CIFAR-100 ELUs networks significantly outperform ReLU networks with batch normalization while batch normalization does not improve ELU networks. ELU networks are among the top 10 reported CIFAR-10 results and yield the best published result on CIFAR-100, without resorting to multi-view evaluation or model averaging. On ImageNet, ELU networks considerably speed up learning compared to a ReLU network with the same architecture, obtaining less than 10% classification error for a single crop, single model network.

1 INTRODUCTION

Currently the most popular activation function for neural networks is the rectified linear unit (ReLU), which was first proposed for restricted Boltzmann machines (Nair & Hinton, 2010) and then successfully used for neural networks (Glorot et al., 2011). The ReLU activation function is the identity for positive arguments and zero otherwise. Besides producing sparse codes, the main advantage of ReLUs is that they alleviate the vanishing gradient problem (Hochreiter, 1998; Hochreiter et al., 2001) since the derivative of 1 for positive values is not contractive (Glorot et al., 2011). However ReLUs are non-negative and, therefore, have a mean activation larger than zero.

Units that have a non-zero mean activation act as bias for the next layer. If such units do not cancel each other out, learning causes a *bias shift* for units in next layer. The more the units are correlated, the higher their bias shift. We will see that Fisher optimal learning, i.e., the natural gradient (Amari, 1998), would correct for the bias shift by adjusting the weight updates. Thus, less bias shift brings the standard gradient closer to the natural gradient and speeds up learning. We aim at activation functions that push activation means closer to zero to decrease the bias shift effect.

Centering the activations at zero has been proposed in order to keep the off-diagonal entries of the Fisher information matrix small (Raiko et al., 2012). For neural network it is known that centering

the activations speeds up learning (LeCun et al., 1991; 1998; Schraudolph, 1998). "Batch normalization" also centers activations with the goal to counter the internal covariate shift (Ioffe & Szegedy, 2015). Also the Projected Natural Gradient Descent algorithm (PRONG) centers the activations by implicitly whitening them (Desjardins et al., 2015).

An alternative to centering is to push the mean activation toward zero by an appropriate activation function. Therefore $tanh$ has been preferred over logistic functions (LeCun et al., 1991; 1998). Recently "Leaky ReLUs" (LReLUs) that replace the negative part of the ReLU with a linear function have been shown to be superior to ReLUs (Maas et al., 2013). Parametric Rectified Linear Units (PReLUs) generalize LReLUs by learning the slope of the negative part which yielded improved learning behavior on large image benchmark data sets (He et al., 2015). Another variant are Randomized Leaky Rectified Linear Units (RReLUs) which randomly sample the slope of the negative part which raised the performance on image benchmark datasets and convolutional networks (Xu et al., 2015).

In contrast to ReLUs, activation functions like LReLUs, PReLUs, and RReLUs do not ensure a noise-robust deactivation state. We propose an activation function that has negative values to allow for mean activations close to zero, but which saturates to a negative value with smaller arguments. The saturation decreases the variation of the units if deactivated, so the precise deactivation argument is less relevant. Such an activation function can code the degree of presence of particular phenomena in the input, but does not quantitatively model the degree of their absence. Therefore, such an activation function is more robust to noise. Consequently, dependencies between coding units are much easier to model and much easier to interpret since only activated code units carry much information. Furthermore, distinct concepts are much less likely to interfere with such activation functions since the deactivation state is non-informative, i.e. variance decreasing.

2 Bias Shift Correction Speeds Up Learning

To derive and analyze the bias shift effect mentioned in the introduction, we utilize the natural gradient. The natural gradient corrects the gradient direction with the inverse Fisher information matrix and, thereby, enables Fisher optimal learning, which ensures the steepest descent in the Riemannian parameter manifold and Fisher efficiency for online learning (Amari, 1998). The recently introduced Hessian-Free Optimization technique (Martens, 2010) and the Krylov Subspace Descent methods (Vinyals & Povey, 2012) use an extended Gauss-Newton approximation of the Hessian, therefore they can be interpreted as versions of natural gradient descent (Pascanu & Bengio, 2014).

Since for neural networks the Fisher information matrix is typically too expensive to compute, different approximations of the natural gradient have been proposed. Topmoumoute Online natural Gradient Algorithm (TONGA) (LeRoux et al., 2008) uses a low-rank approximation of natural gradient descent. FActorized Natural Gradient (FANG) (Grosse & Salakhudinov, 2015) estimates the natural gradient via an approximation of the Fisher information matrix by a Gaussian graphical model. The Fisher information matrix can be approximated by a block-diagonal matrix, where unit or quasi-diagonal natural gradients are used (Olivier, 2013). Unit natural gradients or "Unitwise Fisher's scoring" (Kurita, 1993) are based on natural gradients for perceptrons (Amari, 1998; Yang & Amari, 1998). We will base our analysis on the unit natural gradient.

We assume a parameterized probabilistic model $p(\boldsymbol{x}; \boldsymbol{w})$ with parameter vector $\boldsymbol{w}$ and data $\boldsymbol{x}$. The training data are $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) \in \mathbb{R}^{(d+1)\times N}$ with $\boldsymbol{x}_n = (\boldsymbol{z}_n^T, y_n)^T \in \mathbb{R}^{d+1}$, where $\boldsymbol{z}_n$ is the input for example n and y_n is its label. $L(p(.; \boldsymbol{w}), \boldsymbol{x})$ is the loss of example $\boldsymbol{x} = (\boldsymbol{z}^T, y)^T$ using model $p(.; \boldsymbol{w})$. The average loss on the training data $\boldsymbol{X}$ is the empirical risk $R_{\text{emp}}(p(.; \boldsymbol{w}), \boldsymbol{X})$. Gradient descent updates the weight vector $\boldsymbol{w}$ by $\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta \nabla_{\boldsymbol{w}} R_{\text{emp}}$ where η is the learning rate. The *natural gradient* is the inverse Fisher information matrix $\tilde{\boldsymbol{F}}^{-1}$ multiplied by the gradient of the empirical risk: $\nabla_{\boldsymbol{w}}^{\text{nat}} R_{\text{emp}} = \tilde{\boldsymbol{F}}^{-1} \nabla_{\boldsymbol{w}} R_{\text{emp}}$. For a multi-layer perceptron $\boldsymbol{a}$ is the unit activation vector and $a_0 = 1$ is the bias unit activation. We consider the ingoing weights to unit i, therefore we drop the index i: $w_j = w_{ij}$ for the weight from unit j to unit i, $a = a_i$ for the activation, and w_0 for the bias weight of unit i. The activation function f maps the net input $\text{net} = \sum_j w_j a_j$ of unit i to its activation $a = f(\text{net})$. For computing the Fisher information matrix, the derivative of the log-output probability $\frac{\partial}{\partial w_j} \ln p(\boldsymbol{z}; \boldsymbol{w})$ is required. Therefore we define the δ at unit i as $\delta = \frac{\partial}{\partial \text{net}} \ln p(\boldsymbol{z}; \boldsymbol{w})$,

which can be computed via backpropagation, but using the log-output probability instead of the conventional loss function. The derivative is $\frac{\partial}{\partial w_j} \ln p(\boldsymbol{z}; \boldsymbol{w}) = \delta a_j$.

We restrict the Fisher information matrix to weights leading to unit i which is the *unit Fisher information matrix* $\boldsymbol{F}$. $\boldsymbol{F}$ captures only the interactions of weights to unit i. Consequently, the unit natural gradient only corrects the interactions of weights to unit i, i.e. considers the Riemannian parameter manifold only in a subspace. The unit Fisher information matrix is

$$[\boldsymbol{F}(\boldsymbol{w})]_{kj} \;=\; \mathrm{E}_{p(\boldsymbol{z};\boldsymbol{w})}\left(\frac{\partial \ln p(\boldsymbol{z};\boldsymbol{w})}{\partial w_k}\;\frac{\partial \ln p(\boldsymbol{z};\boldsymbol{w})}{\partial w_j}\right) \;=\; \mathrm{E}_{p(\boldsymbol{z};\boldsymbol{w})}(\delta^2\; a_k\; a_j)\;. \tag{1}$$

Weighting the activations by δ^2 is equivalent to adjusting the probability of drawing inputs $\boldsymbol{z}$. Inputs $\boldsymbol{z}$ with large δ^2 are drawn with higher probability. Since $0 \leq \delta^2 = \delta^2(\boldsymbol{z})$, we can define a distribution $q(\boldsymbol{z})$:

$$q(\boldsymbol{z}) \;=\; \delta^2(\boldsymbol{z})\; p(\boldsymbol{z})\;\left(\int \delta^2(\boldsymbol{z})\; p(\boldsymbol{z})\; d\boldsymbol{z}\right)^{-1} \;=\; \delta^2(\boldsymbol{z})\; p(\boldsymbol{z})\; \mathrm{E}^{-1}_{p(\boldsymbol{z})}(\delta^2)\;. \tag{2}$$

Using $q(\boldsymbol{z})$, the entries of $\boldsymbol{F}$ can be expressed as second moments:

$$[\boldsymbol{F}(\boldsymbol{w})]_{kj} \;=\; \mathrm{E}_{p(\boldsymbol{z})}(\delta^2\; a_k\; a_j) \;=\; \int \delta^2\; a_k\; a_j\; p(\boldsymbol{z})\; d\boldsymbol{z} \;=\; \mathrm{E}_{p(\boldsymbol{z})}(\delta^2)\; \mathrm{E}_{q(\boldsymbol{z})}(a_k\; a_j)\;. \tag{3}$$

If the bias unit is $a_0 = 1$ with weight w_0 then the weight vector can be divided into a bias part w_0 and the rest $\boldsymbol{w}$: $(\boldsymbol{w}^T, w_0)^T$. For the row $\boldsymbol{b} = [\boldsymbol{F}(\boldsymbol{w})]_0$ that corresponds to the bias weight, we have:

$$\boldsymbol{b} \;=\; \mathrm{E}_{p(\boldsymbol{z})}(\delta^2 \boldsymbol{a}) \;=\; \mathrm{E}_{p(\boldsymbol{z})}(\delta^2)\; \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}) \;=\; \mathrm{Cov}_{p(\boldsymbol{z})}(\delta^2, \boldsymbol{a}) \;+\; \mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a})\; \mathrm{E}_{p(\boldsymbol{z})}(\delta^2)\;. \tag{4}$$

The next Theorem 1 gives the correction of the standard gradient by the unit natural gradient where the bias weight is treated separately (see also Yang & Amari (1998)).

Theorem 1. *The unit natural gradient corrects the weight update $(\Delta \boldsymbol{w}^T, \Delta w_0)^T$ to a unit i by following affine transformation of the gradient $\nabla_{(\boldsymbol{w}^T, w_0)^T} R_{\mathrm{emp}} = (\boldsymbol{g}^T, g_0)^T$:*

$$\begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta w_0 \end{pmatrix} \;=\; \begin{pmatrix} \boldsymbol{A}^{-1}\; (\boldsymbol{g} \;-\; \Delta w_0\; \boldsymbol{b}) \\ s\; \left(g_0 \;-\; \boldsymbol{b}^T\; \boldsymbol{A}^{-1} \boldsymbol{g}\right) \end{pmatrix}\;, \tag{5}$$

where $\boldsymbol{A} = [\boldsymbol{F}(\boldsymbol{w})]_{\neg 0, \neg 0} = \mathrm{E}_{p(\boldsymbol{z})}(\delta^2) \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}\boldsymbol{a}^T)$ is the unit Fisher information matrix without row 0 and column 0 corresponding to the bias weight. The vector $\boldsymbol{b} = [\boldsymbol{F}(\boldsymbol{w})]_0$ is the zeroth column of $\boldsymbol{F}$ corresponding to the bias weight, and the positive scalar s is

$$s \;=\; \mathrm{E}^{-1}_{p(\boldsymbol{z})}(\delta^2)\; \left(1 \;+\; \mathrm{E}^{T}_{q(\boldsymbol{z})}(\boldsymbol{a})\; \mathrm{Var}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a})\; \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})\right)\;, \tag{6}$$

where $\boldsymbol{a}$ is the vector of activations of units with weights to unit i and $q(\boldsymbol{z}) = \delta^2(\boldsymbol{z}) p(\boldsymbol{z}) \mathrm{E}^{-1}_{p(\boldsymbol{z})}(\delta^2)$.

Proof. Multiplying the inverse Fisher matrix $\boldsymbol{F}^{-1}$ with the separated gradient $\nabla_{(\boldsymbol{w}^T, w_0)^T} R_{\mathrm{emp}}((\boldsymbol{w}^T, w_0)^T, \boldsymbol{X}) = (\boldsymbol{g}^T, g_0)^T$ gives the weight update $(\Delta \boldsymbol{w}^T, \Delta w_0)^T$:

$$\begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta w_0 \end{pmatrix} \;=\; \begin{pmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{g} \\ g_0 \end{pmatrix} \;=\; \begin{pmatrix} \boldsymbol{A}^{-1}\, \boldsymbol{g} \;+\; \boldsymbol{u}\; s^{-1} \boldsymbol{u}^T \boldsymbol{g} \;+\; g_o\; \boldsymbol{u} \\ \boldsymbol{u}^T \boldsymbol{g} \;+\; s\; g_0 \end{pmatrix}\;. \tag{7}$$

where

$$\boldsymbol{b} \;=\; [\boldsymbol{F}(\boldsymbol{w})]_0 \quad , \quad c \;=\; [\boldsymbol{F}(\boldsymbol{w})]_{00} \quad , \quad \boldsymbol{u} \;=\; -\; s\; \boldsymbol{A}^{-1}\; \boldsymbol{b} \quad , \quad s \;=\; \left(c \;-\; \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b}\right)^{-1}\;. \tag{8}$$

The previous formula is derived in Lemma 1 in the appendix. Using Δw_0 in the update gives

$$\begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta w_0 \end{pmatrix} \;=\; \begin{pmatrix} \boldsymbol{A}^{-1}\, \boldsymbol{g} \;+\; s^{-1} \boldsymbol{u}\; \Delta w_0 \\ \boldsymbol{u}^T \boldsymbol{g} \;+\; s\; g_0 \end{pmatrix} \quad , \quad \begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta w_0 \end{pmatrix} \;=\; \begin{pmatrix} \boldsymbol{A}^{-1}\; (\boldsymbol{g} \;-\; \Delta w_0\; \boldsymbol{b}) \\ s\; \left(g_0 \;-\; \boldsymbol{b}^T\; \boldsymbol{A}^{-1} \boldsymbol{g}\right) \end{pmatrix}\;. \tag{9}$$

The right hand side is obtained by inserting $\boldsymbol{u} \;=\; -s \boldsymbol{A}^{-1} \boldsymbol{b}$ in the left hand side update. Since $c = F_{00} = \mathrm{E}_{p(\boldsymbol{z})}(\delta^2)$, $\boldsymbol{b} = \mathrm{E}_{p(\boldsymbol{z})}(\delta^2) \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$, and $\boldsymbol{A} = \mathrm{E}_{p(\boldsymbol{z})}(\delta^2) \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}\boldsymbol{a}^T)$, we obtain

$$s \;=\; \left(c \;-\; \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b}\right)^{-1} \;=\; \mathrm{E}^{-1}_{p(\boldsymbol{z})}(\delta^2)\; \left(1 \;-\; \mathrm{E}^{T}_{q(\boldsymbol{z})}(\boldsymbol{a})\; \mathrm{E}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a}\boldsymbol{a}^T)\; \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})\right)^{-1}\;. \tag{10}$$

Applying Lemma 2 in the appendix gives the formula for s. □

The *bias shift* (mean shift) of unit i is the change of unit i's mean value due to the weight update. Bias shifts of unit i lead to oscillations and impede learning. See Section 4.4 in LeCun et al. (1998) for demonstrating this effect at the inputs and in LeCun et al. (1991) for explaining this effect using the input covariance matrix. Such bias shifts are mitigated or even prevented by the unit natural gradient. The *bias shift correction* of the unit natural gradient is the effect on the bias shift due to $\boldsymbol{b}$ which captures the interaction between the bias unit and the incoming units. Without bias shift correction, i.e., $\boldsymbol{b} = \boldsymbol{0}$ and $s = c^{-1}$, the weight updates are $\Delta \boldsymbol{w} = \boldsymbol{A}^{-1}\boldsymbol{g}$ and $\Delta w_0 = c^{-1} g_0$. As only the activations depend on the input, the bias shift can be computed by multiplying the weight update by the mean of the activation vector $\boldsymbol{a}$. Thus we obtain the bias shift $(\mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a})^T, 1)(\Delta \boldsymbol{w}^T, \Delta w_0)^T = \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a}) \boldsymbol{A}^{-1}\boldsymbol{g} + c^{-1} g_0$. The bias shift strongly depends on the correlation of the incoming units which is captured by $\boldsymbol{A}^{-1}$.

Next, Theorem 2 states that the bias shift correction by the unit natural gradient can be considered to correct the incoming mean $\mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a})$ proportional to $\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$ toward zero.

Theorem 2. *The bias shift correction by the unit natural gradient is equivalent to an additive correction of the incoming mean by $-k\, \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$ and a multiplicative correction of the bias unit by k, where*

$$k \;=\; 1 \;+\; \left(\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}) \;-\; \mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a})\right)^T \mathrm{Var}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a})\, \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})\;. \tag{11}$$

Proof. Using $\Delta w_0 = -s \boldsymbol{b}^T \boldsymbol{A}^{-1}\boldsymbol{g} + s g_0$, the bias shift is:

$$\begin{aligned}
&\begin{pmatrix} \mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a}) \\ 1 \end{pmatrix}^T \begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta w_0 \end{pmatrix} \;=\; \begin{pmatrix} \mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a}) \\ 1 \end{pmatrix}^T \begin{pmatrix} \boldsymbol{A}^{-1}\, \boldsymbol{g} \;-\; \boldsymbol{A}^{-1}\boldsymbol{b}\, \Delta w_0 \\ \Delta w_0 \end{pmatrix} \\
&= \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a})\, \boldsymbol{A}^{-1}\, \boldsymbol{g} \;+\; \left(1 \;-\; \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a})\, \boldsymbol{A}^{-1}\boldsymbol{b}\right)\, \Delta w_0 \\
&= \left(\mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a}) \;-\; \underbrace{\left(1 \;-\; \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a})\, \boldsymbol{A}^{-1}\boldsymbol{b}\right)\, s\, \boldsymbol{b}^T}\right) \boldsymbol{A}^{-1}\, \boldsymbol{g} \;+\; s\, \left(1 \;-\; \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a})\, \boldsymbol{A}^{-1}\boldsymbol{b}\right)\, g_0\;.
\end{aligned} \tag{12}$$

The mean correction term, indicated by an underbrace in previous formula, is

$$\begin{aligned}
&s\, \left(1 \;-\; \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a})\, \boldsymbol{A}^{-1}\boldsymbol{b}\right)\, \boldsymbol{b} \;=\; \mathrm{E}^{-1}_{p(\boldsymbol{z})}(\delta^2)\, \left(1 \;-\; \mathrm{E}^T_{q(\boldsymbol{z})}(\boldsymbol{a})\, \mathrm{E}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a}\boldsymbol{a}^T)\, \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})\right)^{-1} \\
&\quad \left(1 \;-\; \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a})\, \mathrm{E}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a}\boldsymbol{a}^T)\, \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})\right)\, \mathrm{E}_{p(\boldsymbol{z})}(\delta^2)\, \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}) \\
&= \underbrace{\left(1 - \mathrm{E}^T_{q(\boldsymbol{z})}(\boldsymbol{a})\, \mathrm{E}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a}\boldsymbol{a}^T)\, \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})\right)^{-1} \left(1 - \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a})\, \mathrm{E}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a}\boldsymbol{a}^T)\, \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})\right)}_{k} \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}).
\end{aligned} \tag{13}$$

The expression Eq. (11) for k follows from Lemma 2 in the appendix. The bias unit correction term is $s\left(1 - \mathrm{E}^T_{p(\boldsymbol{z})}(\boldsymbol{a}) \boldsymbol{A}^{-1}\boldsymbol{b}\right) g_0 = k c^{-1} g_0$. □

In Theorem 2 we can reformulate $k = 1 + \mathrm{E}^{-1}_{p(\boldsymbol{z})}(\delta^2) \mathrm{Cov}^T_{p(\boldsymbol{z})}(\delta^2, \boldsymbol{a}) \mathrm{Var}^{-1}_{q(\boldsymbol{z})}(\boldsymbol{a}) \mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$. Therefore k increases with the length of $\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$ for given variances and covariances. Consequently the bias shift correction through the unit natural gradient is governed by the length of $\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$. The bias shift correction is zero for $\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}) = \boldsymbol{0}$ since $k = 1$ does not correct the bias unit multiplicatively. Using Eq. (4), $\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$ is split into an offset and an information containing term:

$$\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a}) \;=\; \mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a}) \;+\; \mathrm{E}^{-1}_{p(\boldsymbol{z})}(\delta^2)\, \mathrm{Cov}_{p(\boldsymbol{z})}(\delta^2, \boldsymbol{a})\;. \tag{14}$$

In general, *smaller positive* $\mathrm{E}_{p(\boldsymbol{z})}(\boldsymbol{a})$ *lead to smaller positive* $\mathrm{E}_{q(\boldsymbol{z})}(\boldsymbol{a})$*, therefore to smaller corrections.* The reason is that in general the largest absolute components of $\mathrm{Cov}_{p(\boldsymbol{z})}(\delta^2, \boldsymbol{a})$ are positive, since activated inputs will activate the unit i which in turn will have large impact on the output.

To summarize, the unit natural gradient corrects the bias shift of unit i via the interactions of incoming units with the bias unit to ensure efficient learning. This correction is equivalent to shifting the mean activations of the incoming units toward zero and scaling up the bias unit. To reduce the

undesired bias shift effect without the natural gradient, either the (i) activation of incoming units can be centered at zero or (ii) activation functions with negative values can be used. We introduce a new activation function with negative values while keeping the identity for positive arguments where it is not contradicting.

3 EXPONENTIAL LINEAR UNITS (ELUS)

The *exponential linear unit* (ELU) with $0 < \alpha$ is

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}. \tag{15}$$

The ELU hyperparameter α controls the value to which an ELU saturates for negative net inputs (see Fig. 1). ELUs diminish the vanishing gradient effect as rectified linear units (ReLUs) and leaky ReLUs (LReLUs) do. The vanishing gradient problem is alleviated because the positive part of these functions is the identity, therefore their derivative is one and not contractive. In contrast, $\tanh$ and sigmoid activation functions are contractive almost everywhere.

In contrast to ReLUs, ELUs have negative values which pushes the mean of the activations closer to zero. Mean activations that are closer to zero enable faster learning as they bring the gradient closer to the natural gradient (see Theorem 2 and text thereafter). ELUs saturate to a negative value when the argument gets smaller. Saturation means a small derivative which decreases the variation and the information that is propagated to the next layer. Therefore the representation is both noise-robust and low-complex (Hochreiter & Schmidhuber, 1999). ELUs code the degree of presence of input concepts, while they neither quantify the degree of their absence nor distinguish the causes of their absence. This property of non-informative deactivation states is also present at ReLUs and allowed to detect biclusters corresponding to biological modules in gene expression datasets (Clevert et al., 2015) and to identify toxicophores in toxicity prediction (Unterthiner et al., 2015; Mayr et al., 2015). The enabling features for these interpretations is that activation can be clearly distinguished from deactivation and that only active units carry relevant information and can crosstalk.

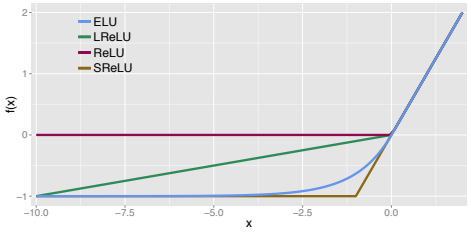


Figure 1: The rectified linear unit (ReLU), the leaky ReLU (LReLU, $\alpha = 0.1$), the shifted ReLUs (SReLUs), and the exponential linear unit (ELU, $\alpha = 1.0$).

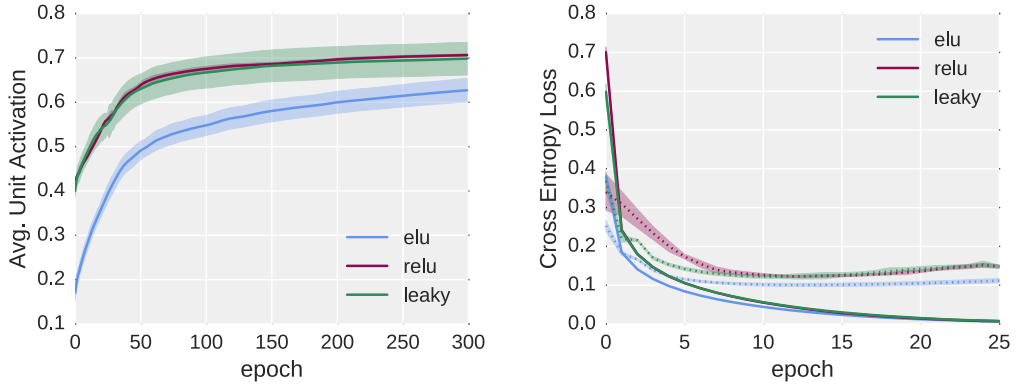
4 EXPERIMENTS USING ELUS

In this section, we assess the performance of exponential linear units (ELUs) if used for unsupervised and supervised learning of deep autoencoders and deep convolutional networks. ELUs with $\alpha = 1.0$ are compared to (i) Rectified Linear Units (ReLUs) with activation $f(x) = \max(0, x)$, (ii) Leaky ReLUs (LReLUs) with activation $f(x) = \max(\alpha x, x)$ ($0 < \alpha < 1$), and (iii) Shifted ReLUs (SReLUs) with activation $f(x) = \max(-1, x)$. Comparisons are done with and without batch normalization. The following benchmark datasets are used: (i) *MNIST* (gray images in 10 classes, 60k train and 10k test), (ii) *CIFAR-10* (color images in 10 classes, 50k train and 10k test), (iii) *CIFAR-100* (color images in 100 classes, 50k train and 10k test), and (iv) *ImageNet* (color images in 1,000 classes, 1.3M train and 100k tests).

4.1 MNIST

4.1.1 LEARNING BEHAVIOR

We first want to verify that ELUs keep the mean activations closer to zero than other units. Fully connected deep neural networks with ELUs ($\alpha = 1.0$), ReLUs, and LReLUs ($\alpha = 0.1$) were trained on the MNIST digit classification dataset while each hidden unit's activation was tracked. Each



(a) Average unit activation

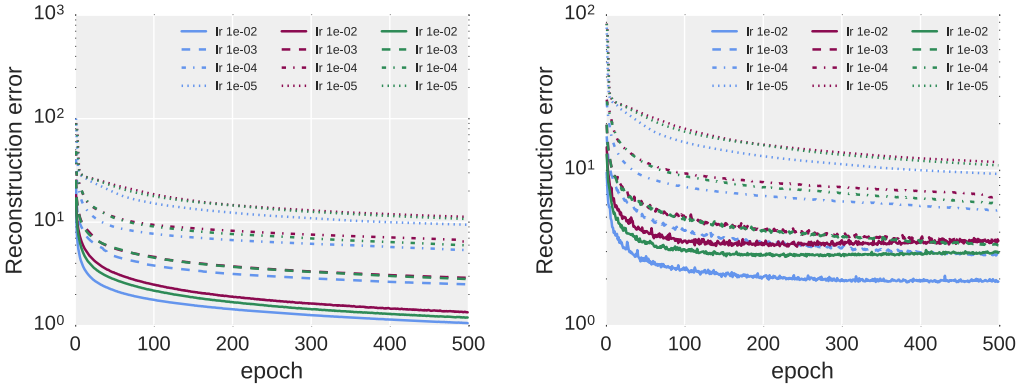
(b) Cross entropy loss

Figure 2: ELU networks evaluated at MNIST. Lines are the average over five runs with different random initializations, error bars show standard deviation. Panel (a): median of the average unit activation for different activation functions. Panel (b): Training set (straight line) and validation set (dotted line) cross entropy loss. All lines stay flat after epoch 25.

network had eight hidden layers of 128 units each, and was trained for 300 epochs by stochastic gradient descent with learning rate 0.01 and mini-batches of size 64. The weights have been initialized according to (He et al., 2015). After each epoch we calculated the units' average activations on a fixed subset of the training data. Fig. 2 shows the median over all units along learning. ELUs stay have smaller median throughout the training process. The training error of ELU networks decreases much more rapidly than for the other networks.

Section C in the appendix compares the variance of median activation in ReLU and ELU networks. The median varies much more in ReLU networks. This indicates that ReLU networks continuously try to correct the bias shift introduced by previous weight updates while this effect is much less prominent in ELU networks.

4.1.2 AUTOENCODER LEARNING



(a) Training set

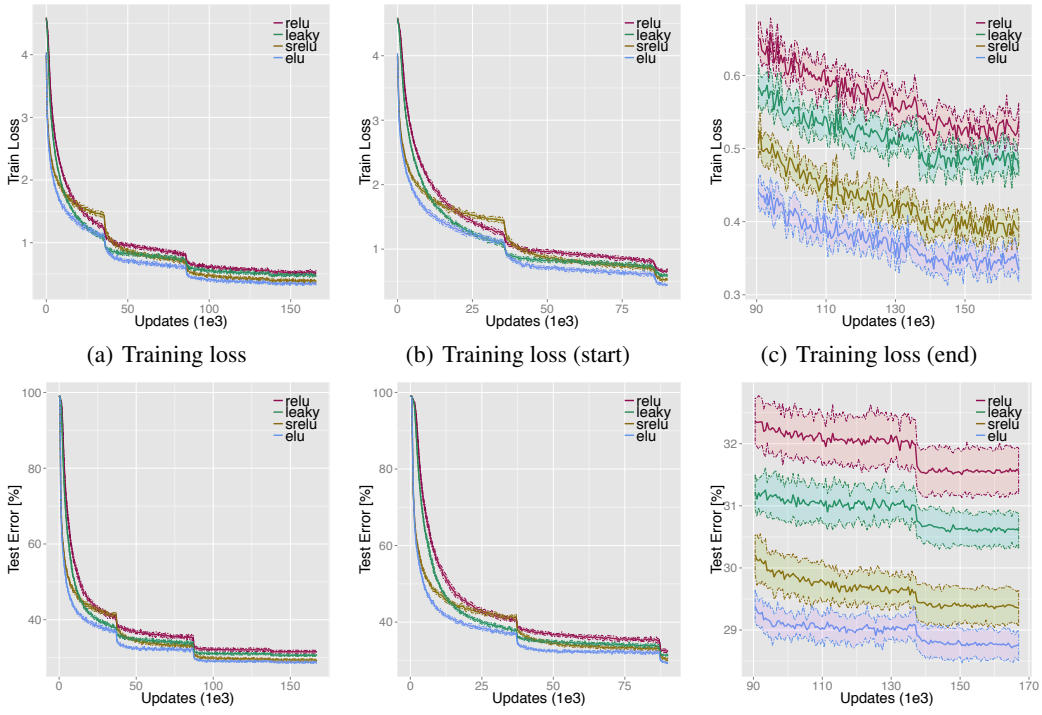
(b) Test set

Figure 3: Autoencoder training on MNIST: Reconstruction error for the test and training data set over epochs, using different activation functions and learning rates. The results are medians over several runs with different random initializations.

To evaluate ELU networks at unsupervised settings, we followed Martens (2010) and Desjardins et al. (2015) and trained a deep autoencoder on the MNIST dataset. The encoder part consisted of four fully connected hidden layers with sizes 1000, 500, 250 and 30, respectively. The decoder part was symmetrical to the encoder. For learning we applied stochastic gradient descent with mini-batches of 64 samples for 500 epochs using the fixed learning rates $(10^{-2}, 10^{-3}, 10^{-4}, 10^{-5})$. Fig. 3 shows, that ELUs outperform the competing activation functions in terms of training / test set reconstruction error for all learning rates. As already noted by Desjardins et al. (2015), higher learning rates seem to perform better.

4.2 Comparison of Activation Functions

In this subsection we show that ELUs indeed possess a superior learning behavior compared to other activation functions as postulated in Section 3. Furthermore we show that ELU networks perform better than ReLU networks with batch normalization. We use as benchmark dataset CIFAR-100 and use a relatively simple convolutional neural network (CNN) architecture to keep the computational complexity reasonable for comparisons.

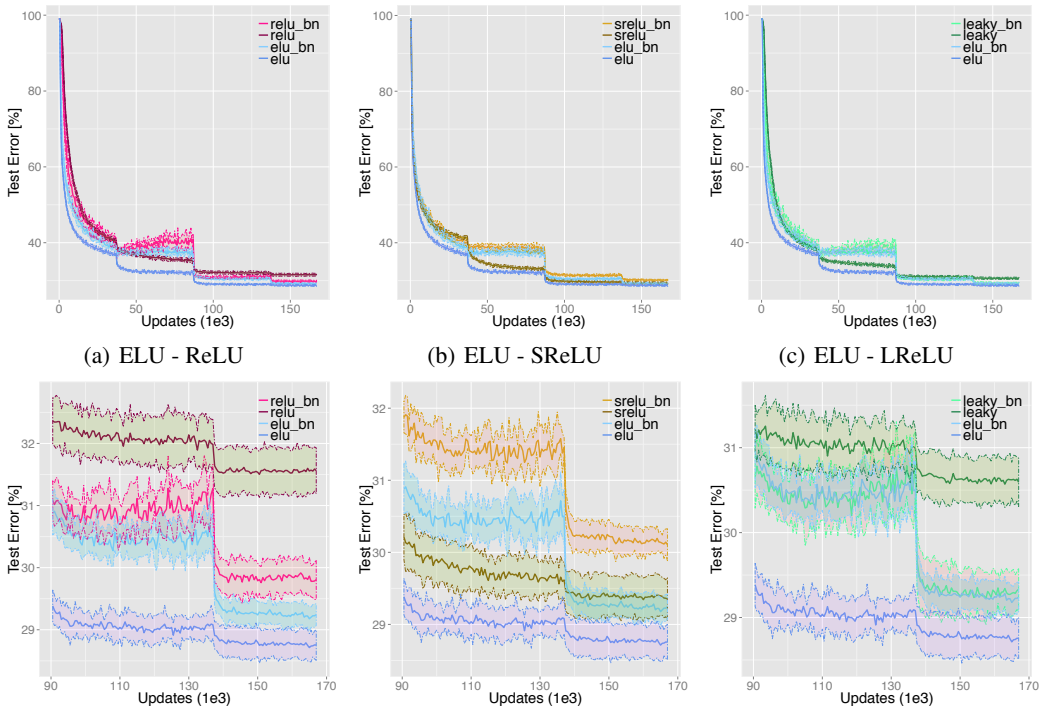


(a) Training loss (b) Training loss (start) (c) Training loss (end)

(d) Test error (e) Test error (start) (f) Test error (end)

Figure 4: Comparison of ReLUs, LReLUs, and SReLUs on CIFAR-100. Panels (a-c) show the training loss, panels (d-f) the test classification error. The ribbon band show the mean and standard deviation for 10 runs along the curve. ELU networks achieved lowest test error and training loss.

The CNN for these CIFAR-100 experiments consists of 11 convolutional layers arranged in stacks of $([1 \times 192 \times 5], [1 \times 192 \times 1, 1 \times 240 \times 3], [1 \times 240 \times 1, 1 \times 260 \times 2], [1 \times 260 \times 1, 1 \times 280 \times 2], [1 \times 280 \times 1, 1 \times 300 \times 2], [1 \times 300 \times 1], [1 \times 100 \times 1])$ layers $\times$ units $\times$ receptive fields. 2×2 max-pooling with a stride of 2 was applied after each stack. For network regularization we used the following drop-out rate for the last layer of each stack $(0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.0)$. The $L2$-weight decay regularization term was set to 0.0005. The following learning rate schedule was applied $(0 - 35k[0.01], 35k - 85k[0.005], 85k - 135k[0.0005], 135k - 165k[0.00005])$ (iterations [learning rate]). For fair comparisons, we used this learning rate schedule for all networks. During previous experiments, this schedule was optimized for ReLU networks, however as ELUs converge faster they would benefit from an adjusted schedule. The momentum term learning rate was fixed to 0.9. The dataset was preprocessed as described in Goodfellow et al. (2013) with global contrast



(a) ELU - ReLU (b) ELU - SReLU (c) ELU - LReLU

(d) ELU - ReLU (end) (e) ELU - SReLU (end) (f) ELU - LReLU (end)

Figure 5: Pairwise comparisons of ELUs with ReLUs, SReLUs, and LReLUs with and without batch normalization (BN) on CIFAR-100. Panels are described as in Fig. 4. ELU networks outperform ReLU networks with batch normalization.

normalization and ZCA whitening. Additionally, the images were padded with four zero pixels at all borders. The model was trained on 32×32 random crops with random horizontal flipping. Besides that, we no further augmented the dataset during training. Each network was run 10 times with different weight initialization. Across networks with different activation functions the same run number had the same initial weights.

Mean test error results of networks with different activation functions are compared in Fig. 4, which also shows the standard deviation. ELUs yield on average a test error of 28.75($\pm$0.24)%, while SReLUs, ReLUs and LReLUs yield 29.35($\pm$0.29)%, 31.56($\pm$0.37)% and 30.59($\pm$0.29)%, respectively. ELUs achieve both lower training loss and lower test error than ReLUs, LReLUs, and SReLUs. Both the ELU training and test performance is significantly better than for other activation functions (Wilcoxon signed-rank test with p-value$<$0.001). Batch normalization improved ReLU and LReLU networks, but did not improve ELU and SReLU networks (see Fig. 5). ELU networks significantly outperform ReLU networks with batch normalization (Wilcoxon signed-rank test with p-value$<$0.001).

4.3 Classification Performance on CIFAR-100 and CIFAR-10

The following experiments should highlight the generalization capabilities of ELU networks. The CNN architecture is more sophisticated than in the previous subsection and consists of 18 convolutional layers arranged in stacks of ($[1 \times 384 \times 3]$, $[1 \times 384 \times 1, 1 \times 384 \times 2, 2 \times 640 \times 2]$, $[1 \times 640 \times 1, 3 \times 768 \times 2]$, $[1 \times 768 \times 1, 2 \times 896 \times 2]$, $[1 \times 896 \times 3, 2 \times 1024 \times 2]$, $[1 \times 1024 \times 1, 1 \times 1152 \times 2]$, $[1 \times 1152 \times 1]$, $[1 \times 100 \times 1]$). Initial drop-out rate, Max-pooling after each stack, $L2$-weight decay, momentum term, data preprocessing, padding, and cropping were as in previous section. The initial learning rate was set to 0.01 and decreased by a factor of 10 after 35k iterations. The mini-batch size was 100. For the final 50k iterations fine-tuning we increased the drop-out rate for *all*

layers in a stack to $(0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.0)$, thereafter increased the drop-out rate by a factor of 1.5 for 40k additional iterations.

Table 1: Comparison of ELU networks and other CNNs on CIFAR-10 and CIFAR-100. Reported is the test error in percent misclassification for ELU networks and recent convolutional architectures like AlexNet, DSN, NiN, Maxout, All-CNN, Highway Network, and Fractional Max-Pooling. Best results are in bold. ELU networks are second best for CIFAR-10 and best for CIFAR-100.

Network	**CIFAR-10 (test error %)**	**CIFAR-100 (test error %)**	**augmented**
AlexNet	18.04	45.80	
DSN	7.97	34.57	$\surd$
NiN	8.81	35.68	$\surd$
Maxout	9.38	38.57	$\surd$
All-CNN	7.25	33.71	$\surd$
Highway Network	7.60	32.24	$\surd$
Fract. Max-Pooling	**4.50**	27.62	$\surd$
ELU-Network	6.55	**24.28**	

ELU networks are compared to following recent successful CNN architectures: AlexNet (Krizhevsky et al., 2012), DSN (Lee et al., 2015), NiN (Lin et al., 2013), Maxout (Goodfellow et al., 2013), All-CNN (Springenberg et al., 2014), Highway Network (Srivastava et al., 2015) and Fractional Max-Pooling (Graham, 2014). The test error in percent misclassification are given in Tab. 1. ELU-networks are the second best on CIFAR-10 with a test error of 6.55% but still they are among the top 10 best results reported for CIFAR-10. ELU networks performed best on CIFAR-100 with a test error of 24.28%. This is the best published result on CIFAR-100, without even resorting to multi-view evaluation or model averaging.

4.4 IMAGENET CHALLENGE DATASET

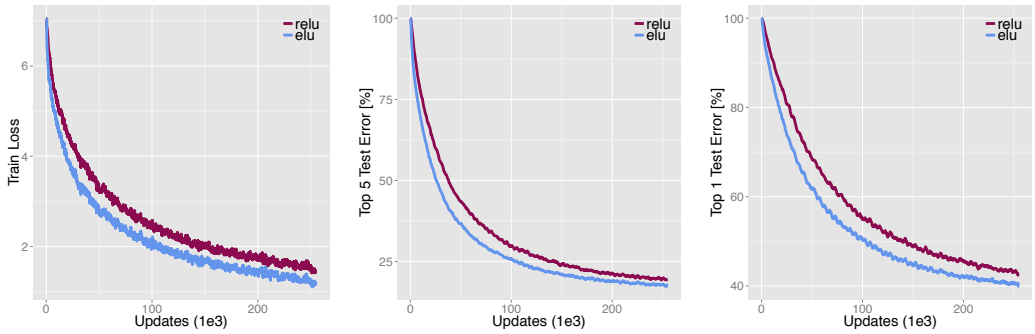
Finally, we evaluated ELU-networks on the 1000-class ImageNet dataset. It contains about 1.3M training color images as well as additional 50k images and 100k images for validation and testing, respectively. For this task, we designed a 15 layer CNN, which was arranged in stacks of $(1 \times 96 \times 6, 3 \times 512 \times 3, 5 \times 768 \times 3, 3 \times 1024 \times 3, 2 \times 4096 \times FC, 1 \times 1000 \times FC)$ layers $\times$ units $\times$ receptive fields or fully-connected (FC). 2×2 max-pooling with a stride of 2 was applied after each stack and spatial pyramid pooling (SPP) with 3 levels before the first FC layer (He et al., 2015). For network regularization we set the $L2$-weight decay term to 0.0005 and used 50% drop-out in the two penultimate FC layers. Images were re-sized to 256×256 pixels and per-pixel mean subtracted. Trained was on 224×224 random crops with random horizontal flipping. Besides that, we did not augment the dataset during training.

Fig. 6 shows the learning behavior of ELU vs. ReLU networks. Panel (b) shows that ELUs start reducing the error earlier. The ELU-network already reaches the 20% top-5 error after 160k iterations, while the ReLU network needs 200k iterations to reach the same error rate. The single-model performance was evaluated on the single center crop with no further augmentation and yielded a top-5 validation error below 10%.

Currently ELU nets are 5% slower on ImageNet than ReLU nets. The difference is small because activation functions generally have only minor influence on the overall training time (Jia, 2014). In terms of wall clock time, ELUs require 12.15h vs. ReLUs with 11.48h for 10k iterations. We expect that ELU implementations can be improved, e.g. by faster exponential functions (Schraudolph, 1999).

5 CONCLUSION

We have introduced the *exponential linear units* (ELUs) for faster and more precise learning in deep neural networks. ELUs have negative values, which allows the network to push the mean activations



(a) Training loss (b) Top-5 test error (c) Top-1 test error

Figure 6: ELU networks applied to ImageNet. The x-axis gives the number of iterations and the y-axis the (a) training loss, (b) top-5 error, and (c) the top-1 error of 5,000 random validation samples, evaluated on the center crop. Both activation functions ELU (blue) and ReLU (purple) lead for convergence, but ELUs start reducing the error earlier and reach the 20% top-5 error after 160k iterations, while ReLUs need 200k iterations to reach the same error rate.

closer to zero. Therefore ELUs decrease the gap between the normal gradient and the unit natural gradient and, thereby speed up learning. We believe that this property is also the reason for the success of activation functions like LReLUs and PReLUs and of batch normalization. In contrast to LReLUs and PReLUs, ELUs have a clear saturation plateau in its negative regime, allowing them to learn a more robust and stable representation. Experimental results show that ELUs significantly outperform other activation functions on different vision datasets. Further ELU networks perform significantly better than ReLU networks trained with batch normalization. ELU networks achieved one of the top 10 best reported results on CIFAR-10 and set a new state of the art in CIFAR-100 without the need for multi-view test evaluation or model averaging. Furthermore, ELU networks produced competitive results on the ImageNet in much fewer epochs than a corresponding ReLU network. Given their outstanding performance, we expect ELU networks to become a real time saver in convolutional networks, which are notably time-intensive to train from scratch otherwise.

Acknowledgment. We thank the NVIDIA Corporation for supporting this research with several Titan X GPUs and Roland Vollgraf and Martin Heusel for helpful discussions and comments on this work.

REFERENCES

Amari, S.-I. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Clevert, D.-A., Unterthiner, T., Mayr, A., and Hochreiter, S. Rectified factor networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015.

Desjardins, G., Simonyan, K., Pascanu, R., and Kavukcuoglu, K. Natural neural networks. *CoRR*, abs/1507.00210, 2015. URL `http://arxiv.org/abs/1507.00210`.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudk, M. (eds.), *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, volume 15, pp. 315–323, 2011.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. *ArXiv e-prints*, 2013.

Graham, Benjamin. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014. URL `http://arxiv.org/abs/1412.6071`.

Grosse, R. and Salakhudinov, R. Scaling up natural gradient by sparsely factorizing the inverse Fisher matrix. *Journal of Machine Learning Research*, 37:2304–2313, 2015. URL `http://jmlr.org/proceedings/papers/v37/grosse15.pdf`. Proceedings of the 32nd International Conference on Machine Learning (ICML15).

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

Hochreiter, S. and Schmidhuber, J. Feature extraction through LOCOCODE. *Neural Computation*, 11(3): 679–714, 1999.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer and Kolen (eds.), *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Journal of Machine Learning Research*, 37:448–456, 2015. URL http://jmlr.org/proceedings/papers/v37/ioffe15.pdf. Proceedings of the 32nd International Conference on Machine Learning (ICML15).

Jia, Yangqing. *Learning Semantic Image Representations at a Large Scale*. PhD thesis, EECS Department, University of California, Berkeley, May 2014. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-93.html.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

Kurita, T. Iterative weighted least squares algorithms for neural networks classifiers. In *Proceedings of the Third Workshop on Algorithmic Learning Theory (ALT92)*, volume 743 of *Lecture Notes in Computer Science*, pp. 77–86. Springer, 1993.

LeCun, Y., Kanter, I., and Solla, S. A. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396–2399, 1991.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In Orr, G. B. and Müller, K.-R. (eds.), *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pp. 9–50. Springer, 1998.

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick W., Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. In *AISTATS*, 2015.

LeRoux, N., Manzagol, P.-A., and Bengio, Y. Topmoumoute online natural gradient algorithm. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T. (eds.), *Advances in Neural Information Processing Systems 20 (NIPS)*, pp. 849–856, 2008.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *CoRR*, abs/1312.4400, 2013. URL http://arxiv.org/abs/1312.4400.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML13)*, 2013.

Martens, J. Deep learning via Hessian-free optimization. In Fürnkranz, J. and Joachims, T. (eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML10)*, pp. 735–742, 2010.

Mayr, A., Klambauer, G., Unterthiner, T., and Hochreiter, S. DeepTox: Toxicity prediction using deep learning. *Front. Environ. Sci.*, 3(80), 2015. doi: 10.3389/fenvs.2015.00080. URL http://journal.frontiersin.org/article/10.3389/fenvs.2015.00080.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. In Fürnkranz, J. and Joachims, T. (eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML10)*, pp. 807–814, 2010.

Olivier, Y. Riemannian metrics for neural networks i: feedforward networks. *CoRR*, abs/1303.0818, 2013. URL http://arxiv.org/abs/1303.0818.

Pascanu, R. and Bengio, Y. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations 2014*, 2014. URL http://arxiv.org/abs/1301.3584. arXiv:1301.3584.

Raiko, T., Valpola, H., and LeCun, Y. Deep learning made easier by linear transformations in perceptrons. In Lawrence, N. D. and Girolami, M. A. (eds.), *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS12)*, volume 22, pp. 924–932, 2012.

Schraudolph, N. N. Centering neural network gradient factor. In Orr, G. B. and Müller, K.-R. (eds.), *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pp. 207–226. Springer, 1998.

Schraudolph, Nicol N. A Fast, Compact Approximation of the Exponential Function. *Neural Computation*, 11: 853–862, 1999.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin A. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL http://arxiv.org/abs/1412.6806.

Srivastava, Rupesh Kumar, Greff, Klaus, and Schmidhuber, Jürgen. Training very deep networks. *CoRR*, abs/1507.06228, 2015. URL http://arxiv.org/abs/1507.06228.

Unterthiner, T., Mayr, A., Klambauer, G., and Hochreiter, S. Toxicity prediction using deep learning. *CoRR*, abs/1503.01445, 2015. URL http://arxiv.org/abs/1503.01445.

Vinyals, O. and Povey, D. Krylov subspace descent for deep learning. In *AISTATS*, 2012. URL http://arxiv.org/pdf/1111.4259v1. arXiv:1111.4259.

Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL http://arxiv.org/abs/1505.00853.

Yang, H. H. and Amari, S.-I. Complexity issues in natural gradient descent method for training multilayer perceptrons. *Neural Computation*, 10(8), 1998.

A INVERSE OF BLOCK MATRICES

Lemma 1. *The positive definite matrix $\boldsymbol{M}$ is in block format with matrix $\boldsymbol{A}$, vector $\mathbf{b}$, and scalar c. The inverse of $\boldsymbol{M}$ is*

$$\boldsymbol{M}^{-1} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{K} & \boldsymbol{u} \\ \boldsymbol{u}^T & s \end{pmatrix} , \tag{16}$$

where

$$\boldsymbol{K} = \boldsymbol{A}^{-1} + \boldsymbol{u}\, s^{-1} \boldsymbol{u}^T \tag{17}$$

$$\boldsymbol{u} = -\, s\, \boldsymbol{A}^{-1}\, \boldsymbol{b} \tag{18}$$

$$s = \left(c - \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b} \right)^{-1} . \tag{19}$$

Proof. For block matrices the inverse is

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{C} \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{K} & \boldsymbol{U} \\ \boldsymbol{U}^T & \boldsymbol{S} \end{pmatrix} , \tag{20}$$

where the matrices on the right hand side are:

$$\boldsymbol{K} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\, \boldsymbol{B}\, \left(\boldsymbol{C} - \boldsymbol{B}^T \boldsymbol{A}^{-1} \boldsymbol{B} \right)^{-1} \boldsymbol{B}^T\, \boldsymbol{A}^{-1} \tag{21}$$

$$\boldsymbol{U} = -\, \boldsymbol{A}^{-1}\, \boldsymbol{B}\, \left(\boldsymbol{C} - \boldsymbol{B}^T \boldsymbol{A}^{-1} \boldsymbol{B} \right)^{-1} \tag{22}$$

$$\boldsymbol{U}^T = -\, \left(\boldsymbol{C} - \boldsymbol{B}^T \boldsymbol{A}^{-1} \boldsymbol{B} \right)^{-1} \boldsymbol{B}^T\, \boldsymbol{A}^{-1} \tag{23}$$

$$\boldsymbol{S} = \left(\boldsymbol{C} - \boldsymbol{B}^T \boldsymbol{A}^{-1} \boldsymbol{B} \right)^{-1} . \tag{24}$$

Further if follows that

$$\boldsymbol{K} = \boldsymbol{A}^{-1} + \boldsymbol{U}\, \boldsymbol{S}^{-1} \boldsymbol{U}^T . \tag{25}$$

We now use this formula for $\boldsymbol{B} = \boldsymbol{b}$ being a vector and $\boldsymbol{C} = c$ a scalar. We obtain

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{K} & \boldsymbol{u} \\ \boldsymbol{u}^T & s \end{pmatrix} , \tag{26}$$

where the right hand side matrices, vectors, and the scalar s are:

$$\boldsymbol{K} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\, \boldsymbol{b}\, \left(c - \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b} \right)^{-1} \boldsymbol{b}^T\, \boldsymbol{A}^{-1} \tag{27}$$

$$\boldsymbol{u} = -\, \boldsymbol{A}^{-1}\, \boldsymbol{b}\, \left(c - \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b} \right)^{-1} \tag{28}$$

$$\boldsymbol{u}^T = -\, \left(c - \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b} \right)^{-1} \boldsymbol{b}^T\, \boldsymbol{A}^{-1} \tag{29}$$

$$s = \left(c - \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b} \right)^{-1} . \tag{30}$$

Again it follows that

$$\boldsymbol{K} = \boldsymbol{A}^{-1} + \boldsymbol{u}\, s^{-1} \boldsymbol{u}^T . \tag{31}$$

A reformulation using $\boldsymbol{u}$ gives

$$\boldsymbol{K} \;=\; \boldsymbol{A}^{-1} \;+\; \boldsymbol{u}\, s^{-1} \boldsymbol{u}^T \tag{32}$$

$$\boldsymbol{u} \;=\; -\, s\, \boldsymbol{A}^{-1}\, \boldsymbol{b} \tag{33}$$

$$\boldsymbol{u}^T \;=\; -\, s\, \boldsymbol{b}^T\, \boldsymbol{A}^{-1} \tag{34}$$

$$s \;=\; \left(c \;-\; \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b}\right)^{-1} \;. \tag{35}$$

□

B Quadratic Form of Mean and Inverse Second Moment

Lemma 2. *For a random variable $\boldsymbol{a}$ holds*

$$\mathrm{E}^T(\boldsymbol{a})\, \mathrm{E}^{-1}(\boldsymbol{a}\, \boldsymbol{a}^T)\, \mathrm{E}(\boldsymbol{a}) \;\leq\; 1 \tag{36}$$

and

$$\left(1 \;-\; \mathrm{E}^T(\boldsymbol{a})\, \mathrm{E}^{-1}(\boldsymbol{a}\, \boldsymbol{a}^T)\, \mathrm{E}(\boldsymbol{a})\right)^{-1} \;=\; 1 \;+\; \mathrm{E}^T(\boldsymbol{a})\, \mathrm{Var}^{-1}(\boldsymbol{a})\, \mathrm{E}(\boldsymbol{a}) \;. \tag{37}$$

Furthermore holds

$$\begin{aligned} &\left(1 \;-\; \mathrm{E}^T(\boldsymbol{a})\, \mathrm{E}^{-1}(\boldsymbol{a}\, \boldsymbol{a}^T)\, \mathrm{E}(\boldsymbol{a})\right)^{-1} \left(1 \;-\; \mathrm{E}_p^T(\boldsymbol{a})\, \mathrm{E}^{-1}(\boldsymbol{a}\, \boldsymbol{a}^T)\, \mathrm{E}(\boldsymbol{a})\right) \\ &=\; 1 \;+\; (\mathrm{E}(\boldsymbol{a}) \;-\; \mathrm{E}_p(\boldsymbol{a}))^T\, \mathrm{Var}^{-1}(\boldsymbol{a})\, \mathrm{E}(\boldsymbol{a}) \;. \end{aligned} \tag{38}$$

Proof. The Sherman-Morrison Theorem states

$$\left(\boldsymbol{A} \;+\; \boldsymbol{b}\, \boldsymbol{c}^T\right)^{-1} \;=\; \boldsymbol{A}^{-1} \;-\; \frac{\boldsymbol{A}^{-1}\, \boldsymbol{b}\, \boldsymbol{c}^T\, \boldsymbol{A}^{-1}}{1 \;+\; \boldsymbol{c}^T \boldsymbol{A}^{-1} \boldsymbol{b}} \;. \tag{39}$$

Therefore we have

$$\begin{aligned} \boldsymbol{c}^T \left(\boldsymbol{A} \;+\; \boldsymbol{b}\, \boldsymbol{b}^T\right)^{-1} \boldsymbol{b} \;&=\; \boldsymbol{c}^T \boldsymbol{A}^{-1} \boldsymbol{b} \;-\; \frac{\boldsymbol{c}^T \boldsymbol{A}^{-1}\, \boldsymbol{b}\, \boldsymbol{b}^T\, \boldsymbol{A}^{-1} \boldsymbol{b}}{1 \;+\; \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b}} \\ &=\; \frac{\boldsymbol{c}^T \boldsymbol{A}^{-1} \boldsymbol{b}\, \left(1 \;+\; \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b}\right) \;-\; \left(\boldsymbol{c}^T \boldsymbol{A}^{-1}\, \boldsymbol{b}\right) \left(\boldsymbol{b}^T \boldsymbol{A}^{-1}\, \boldsymbol{b}\right)}{1 \;+\; \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b}} \\ &=\; \frac{\boldsymbol{c}^T \boldsymbol{A}^{-1} \boldsymbol{b}}{1 \;+\; \boldsymbol{b}^T \boldsymbol{A}^{-1} \boldsymbol{b}} \;. \end{aligned} \tag{40}$$

Using the identity

$$\mathrm{E}(\boldsymbol{a}\, \boldsymbol{a}^T) \;=\; \mathrm{Var}(\boldsymbol{a}) \;+\; \mathrm{E}(\boldsymbol{a})\, \mathrm{E}^T(\boldsymbol{a}) \tag{41}$$

for the second moment and Eq. (40), we get

$$\begin{aligned} \mathrm{E}^T(\boldsymbol{a})\, \mathrm{E}^{-1}(\boldsymbol{a}\, \boldsymbol{a}^T)\, \mathrm{E}(\boldsymbol{a}) \;&=\; \mathrm{E}^T(\boldsymbol{a}) \left(\mathrm{Var}(\boldsymbol{a}) \;+\; \mathrm{E}(\boldsymbol{a})\, \mathrm{E}^T(\boldsymbol{a})\right)^{-1} \mathrm{E}(\boldsymbol{a}) \\ &=\; \frac{\mathrm{E}^T(\boldsymbol{a})\, \mathrm{Var}^{-1}(\boldsymbol{a})\, \mathrm{E}(\boldsymbol{a})}{1 \;+\; \mathrm{E}^T(\boldsymbol{a})\, \mathrm{Var}^{-1}(\boldsymbol{a})\, \mathrm{E}(\boldsymbol{a})} \;\leq\; 1 \;. \end{aligned} \tag{42}$$

The last inequality follows from the fact that $\mathrm{Var}(\boldsymbol{a})$ is positive definite. From last equation, we obtain further

$$\left(1 \;-\; \mathrm{E}^T(\boldsymbol{a}) \mathrm{E}^{-1}(\boldsymbol{a}\, \boldsymbol{a}^T)\, \mathrm{E}(\boldsymbol{a})\right)^{-1} \;=\; 1 \;+\; \mathrm{E}^T(\boldsymbol{a}) \mathrm{Var}^{-1}(\boldsymbol{a})\, \mathrm{E}(\boldsymbol{a}) \;. \tag{43}$$

For the mixed quadratic form we get from Eq. (40)

$$\begin{aligned} \mathrm{E}_p^T(\boldsymbol{a})\, \mathrm{E}^{-1}(\boldsymbol{a}\, \boldsymbol{a}^T)\, \mathrm{E}(\boldsymbol{a}) \;&=\; \mathrm{E}_p^T(\boldsymbol{a}) \left(\mathrm{Var}(\boldsymbol{a}) \;+\; \mathrm{E}(\boldsymbol{a})\, \mathrm{E}^T(\boldsymbol{a})\right)^{-1} \mathrm{E}(\boldsymbol{a}) \\ &=\; \frac{\mathrm{E}_p^T(\boldsymbol{a})\, \mathrm{Var}^{-1}(\boldsymbol{a})\, \mathrm{E}(\boldsymbol{a})}{1 \;+\; \mathrm{E}^T(\boldsymbol{a})\, \mathrm{Var}^{-1}(\boldsymbol{a})\, \mathrm{E}(\boldsymbol{a})} \;. \end{aligned} \tag{44}$$

From this equation follows

$$1 \; - \; \mathrm{E}_p^T(\boldsymbol{a}) \, \mathrm{E}^{-1}(\boldsymbol{a} \, \boldsymbol{a}^T) \, \mathrm{E}(\boldsymbol{a}) \; = \; 1 \; - \; \frac{\mathrm{E}_p^T(\boldsymbol{a}) \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a})}{1 \; + \; \mathrm{E}^T(\boldsymbol{a}) \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a})} \tag{45}$$

$$= \; \frac{1 \; + \; \mathrm{E}^T(\boldsymbol{a}) \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a}) \; - \; \mathrm{E}_p^T(\boldsymbol{a}) \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a})}{1 \; + \; \mathrm{E}^T(\boldsymbol{a}) \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a})} \; = \; \frac{1 \; + \; (\mathrm{E}(\boldsymbol{a}) \; - \; \mathrm{E}_p(\boldsymbol{a}))^T \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a})}{1 \; + \; \mathrm{E}^T(\boldsymbol{a}) \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a})} \; .$$

Therefore we get

$$\begin{aligned} &\left(1 \; - \; \mathrm{E}^T(\boldsymbol{a}) \, \mathrm{E}^{-1}(\boldsymbol{a} \, \boldsymbol{a}^T) \, \mathrm{E}(\boldsymbol{a})\right)^{-1} \; \left(1 \; - \; \mathrm{E}_p^T(\boldsymbol{a}) \, \mathrm{E}^{-1}(\boldsymbol{a} \, \boldsymbol{a}^T) \, \mathrm{E}(\boldsymbol{a})\right) \\ &= \; 1 \; + \; (\mathrm{E}(\boldsymbol{a}) \; - \; \mathrm{E}_p(\boldsymbol{a}))^T \, \mathrm{Var}^{-1}(\boldsymbol{a}) \, \mathrm{E}(\boldsymbol{a}) \; . \end{aligned} \tag{46}$$

□

C VARIANCE OF MEAN ACTIVATIONS IN ELU AND RELU NETWORKS

To compare the variance of median activation in ReLU and ELU networks, we trained a neural network with 5 hidden layers of 256 hidden units for 200 epochs using a learning rate of 0.01, once using ReLU and once using ELU activation functions on the MNIST dataset. After each epoch, we calculated the median activation of each hidden unit on the whole training set. We then calculated the variance of these changes, which is depicted in Figure 7 . The median varies much more in ReLU networks. This indicates that ReLU networks continuously try to correct the bias shift introduced by previous weight updates while this effect is much less prominent in ELU networks.

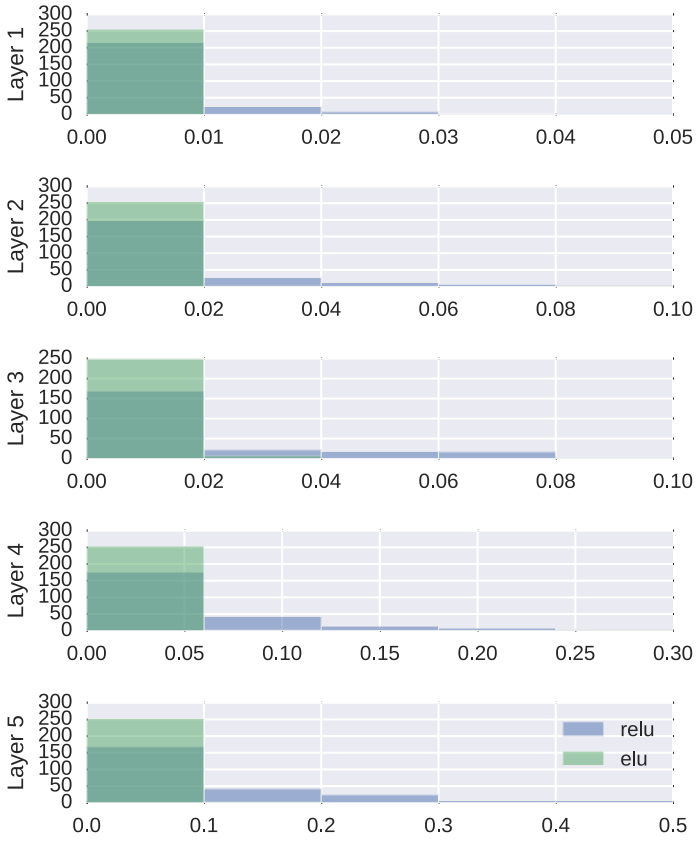


Figure 7: Distribution of variances of the median hidden unit activation after each epoch of MNIST training. Each row represents the units in a different layer of the network.