AutoInit: Analytic Signal-Preserving Weight Initialization for Neural Networks

Garrett Bingham and Risto Miikkulainen

University of Texas at Austin and Cognizant AI Labs

Abstract

AutoInit is a weight initialization algorithm that automatically adapts to different neural network architectures. By analytically tracking the mean and variance of signals as they propagate through the network, AutoInit appropriately scales the weights at each layer to avoid exploding or vanishing signals. AutoInit thus serves as an automatic configuration tool that makes design of new neural network architectures more robust.

Background: Neural Net Signal Propagation

- A layer shifts its input by α and scales the input by a factor of β.
- If the input to the layer has mean μ_{in} and variance ν_{in}, after applying the layer, the output signal will have mean $\mu_{\text{out}} = \alpha + \beta\mu_{\text{in}}$ and variance $\nu_{\text{out}} = \beta^2\nu_{\text{in}}$.
- After L layers, the signal at the final layer has mean and variance

$$\mu_{\text{out}} = \beta^L\mu_{\text{in}} + \alpha(\beta^L + \beta^{L-1} + \cdots + \beta + 1), \quad \nu_{\text{out}} = \beta^{2L}\nu_{\text{in}}. \tag{1}$$

- If $|\beta| > 1$, the network will suffer from a mean shift and exploding signals:

$$\lim_{L\to\infty} \mu_{\text{out}} = \infty, \quad \lim_{L\to\infty} \nu_{\text{out}} = \infty. \tag{2}$$

- If $|\beta| < 1$, the network will suffer from a mean shift and vanishing signals:

$$\lim_{L\to\infty} \mu_{\text{out}} = \alpha/(1-\beta), \quad \lim_{L\to\infty} \nu_{\text{out}} = 0. \tag{3}$$

- AutoInit calculates weight initialization so that $\alpha = 0$ and $\beta = 1$, avoiding the issues of mean shift and exploding/vanishing signals.

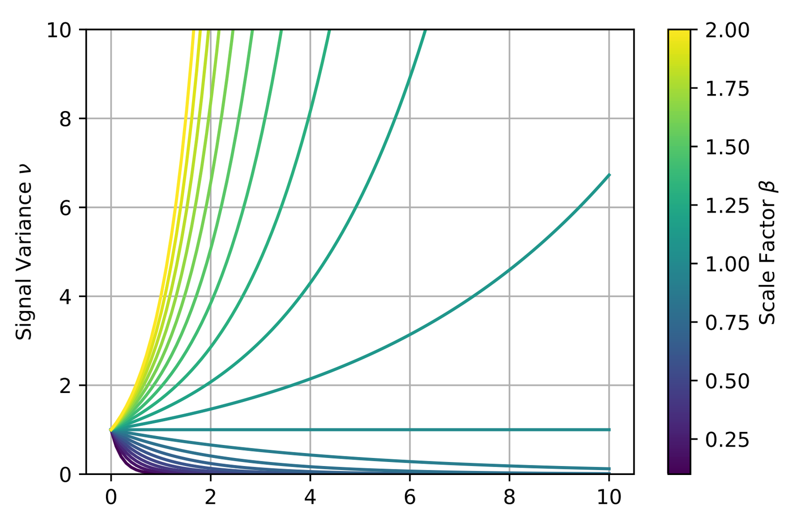

Figure 1: AutoInit maintains $|\beta| = 1$ to avoid exploding or vanishing signals.

Contribution: The AutoInit Framework

- AutoInit uses g functions to map input mean and variance to output mean and variance when a `layer` is applied:

$$g_{\texttt{layer}} : (\mu_{\text{in}}, \nu_{\text{in}}) \mapsto (\mu_{\text{out}}, \nu_{\text{out}}). \tag{4}$$

- Functions are derived for each type of `layer`: $g_{\texttt{Dropout}}$, $g_{\texttt{ReLU}}$, $g_{\texttt{Conv2D},\theta}$, etc.
- If a layer has weights, they are initialized so that the layer output will have zero mean and unit variance in expectation. For example:

$$\theta \sim \mathcal{N}\left(0, 1/\sqrt{\texttt{fan\_in}(\nu_{\text{in}} + \mu_{\text{in}}^2)}\right) \implies g_{\texttt{Conv2D},\theta}(\mu_{\text{in}}, \nu_{\text{in}}) = (0, 1). \tag{5}$$

Algorithm 1: AutoInit

Input: Network with layers L, directed edges E
`output_layers` $= \{l \in L \mid (l, l') \notin E \, \forall \, l' \in L\}$
for `output_layer in output_layers` **do**
  `initialize(output_layer)`

def `initialize(layer)`:
  `layers_in` $= \{l \in L \mid (l, \texttt{layer}) \in E\}$
  $i = 1$
  for `layer_in in layers_in` **do**
    $\mu_{\text{in}_i}, \nu_{\text{in}_i}$ = `initialize(layer_in)`
    $i = i + 1$
  $\mu_{\text{in}} = (\mu_{\text{in}_1}, \mu_{\text{in}_2}, \ldots, \mu_{\text{in}_N})$
  $\nu_{\text{in}} = (\nu_{\text{in}_1}, \nu_{\text{in}_2}, \ldots, \nu_{\text{in}_N})$
  if `layer` has weights θ **then**
    initialize θ s.t. $g_{\texttt{layer},\theta}(\mu_{\text{in}}, \nu_{\text{in}}) = (0, 1)$
    $\mu_{\text{out}}, \nu_{\text{out}} = 0, 1$
  else
    $\mu_{\text{out}}, \nu_{\text{out}} = g_{\texttt{layer}}(\mu_{\text{in}}, \nu_{\text{in}})$
  return $\mu_{\text{out}}, \nu_{\text{out}}$

Robustness to Hyperparameter Variation

- Different layers, activation functions, and hyperparameters affect the signal variance in different ways.
- Ignoring these differences results in inconsistent behavior and vanishing signals.
- AutoInit adapts to these settings automatically, stabilizing signal propagation.

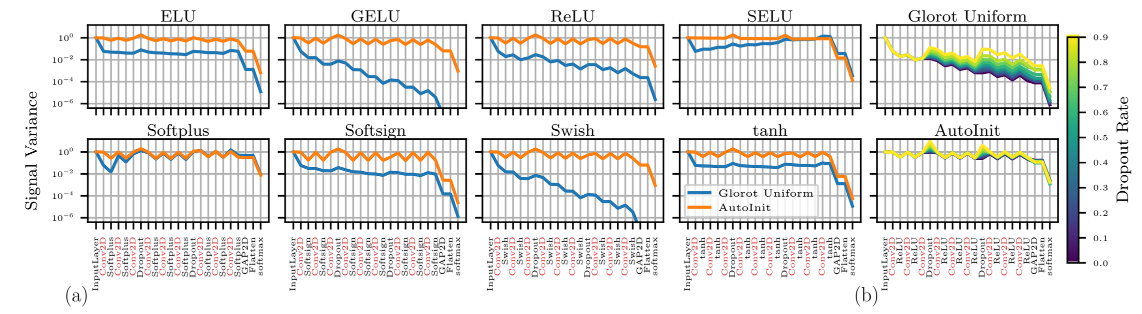

Figure 2: Signal propagation in All-CNN-C networks with different (a) activation functions and (b) dropout rates. With the default initialization, signals often vanish with depth, and their behavior is inconsistent across activation functions and dropout rates. With AutoInit, the variance fluctuates naturally as each layer modifies its input. At layers with weights (marked in red), AutoInit scales the weights appropriately to return the variance to approximately 1.0, stabilizing training in each case.

- In separate experiments, the activation function, dropout rate, weight decay, and learning rate were changed. While one hyperparameter varied, the others were kept at default values.
- AutoInit adapts the initialization to different activation functions and dropout rates, and is robust to changes in weight decay and learning rate.

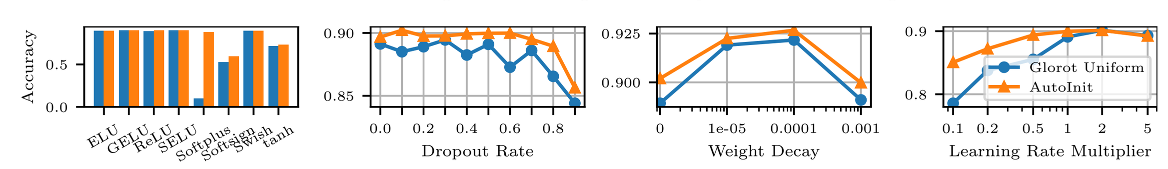

Figure 3: All-CNN-C test accuracy on CIFAR-10. AutoInit results in comparable or better performance in every case.

Stability with Extremely Deep Networks

- Stable signal propagation is crucial, especially for deep networks.
- The default initialization causes exploding signals, even exceeding machine precision on ResNet-812.
- AutoInit stabilizes extremely deep networks, with or without BatchNorm.

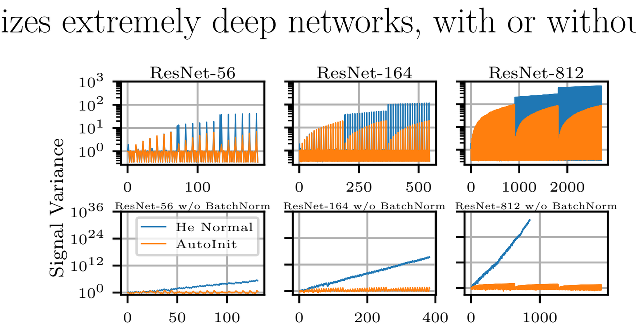

Figure 4: Signal propagation in residual networks. Gaussian input was fed to the networks and empirical variance computed at each layer. Since `ReLU`, `BatchNormalization`, and `Add` are counted as individual layers in this diagram, the total number of layers is different from that in the architecture name (i.e. ResNet-164 has 164 convolutional layers but over 500 total layers). The default initialization causes exploding signals, while AutoInit ensures signal propagation is stable.

- AutoInit allows for a wider range of initial learning rates and improves accuracy.

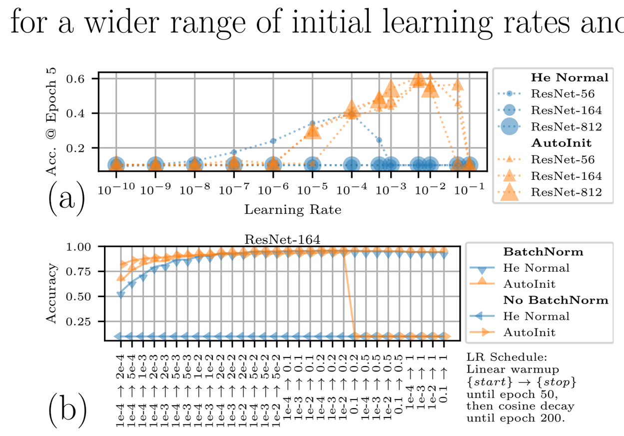

Figure 5: ResNet accuracy on CIFAR-10 with different settings. (a) Accuracy of unnormalized ResNet architectures after five epochs of training with different learning rates and weight initializations. While default initialization makes training difficult in ResNet-56 and impossible at greater depths, AutoInit results in consistent training at all depths. (b) Accuracy of ResNet-164 with a variety of learning rate schedules and initializations. AutoInit is comparable to or outperforms the default initialization in every case.

Scaling up to ImageNet

- AutoInit improves performance with hybrid transformer architectures.
- Accuracy is improved on Imagenette, a 10-class subset of ImageNet.

CoAtNet	w/ GELU	w/ ReLU	w/ SELU	w/ Swish	w/o Norm
Default Init.	89.38	89.22	86.09	88.69	-
Glorot Normal	91.44	91.54	87.59	90.42	**85.89**
Glorot Uniform	91.16	91.18	**88.25**	90.06	85.73
He Normal	88.48	88.05	86.11	88.36	-
He Uniform	88.66	87.87	86.37	88.41	-
LeCun Normal	91.11	90.57	87.80	90.83	-
LeCun Uniform	90.55	90.65	87.67	90.57	-
AutoInit	**92.48**	**92.15**	86.80	**92.28**	85.73

Table 1: CoAtNet top-1 accuracy on Imagenette, shown as median of three runs. The first four experiments vary the activation function, while the fifth removes all normalization layers from the architecture. A "-" indicates that training diverged. AutoInit produces the best model in three of the five settings, and remains stable even without normalization layers.

- AutoInit scales to large datasets like ImageNet.

	top-1	top-5
Default Init.	74.33	91.60
AutoInit	**75.35**	**92.03**

Table 2: ResNet-50 top-1 and top-5 validation accuracy on ImageNet. AutoInit improves performance, even with large and challenging datasets.

Reliability vs. Data-Dependent Initialization

- Because AutoInit does not depend on data samples, it is more reliable than data-dependent initialization methods are.

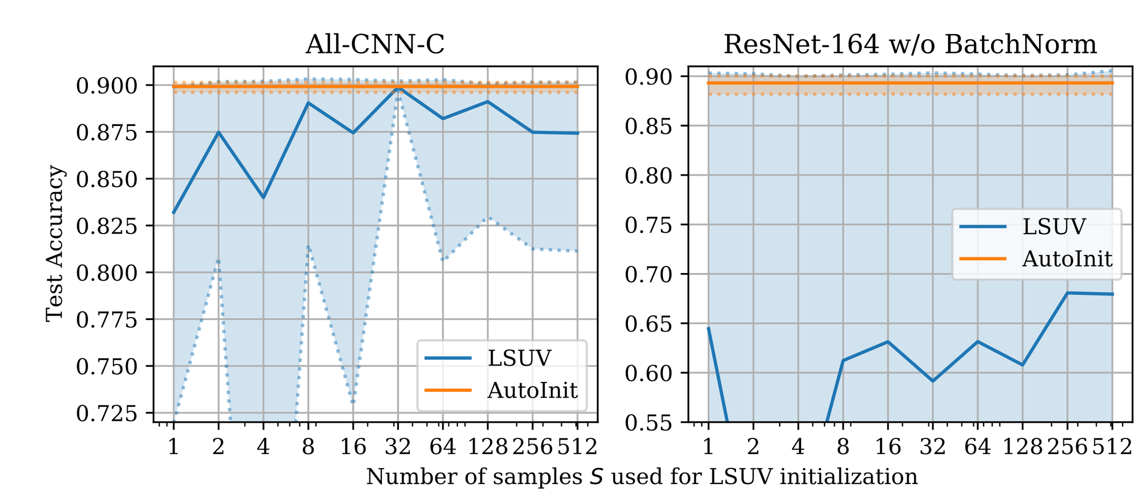

Figure 6: Mean CIFAR-10 test accuracy for AutoInit vs. LSUV with different numbers of samples S. Each evaluation is repeated 10 times; the shaded area shows the maximum and minimum accuracy among all trials. AutoInit is consistent, but LSUV struggles when S is small or the network is deep.

More Effective Neural Architecture Search

- Networks were evolved in five tasks to simulate research and discovery of new architectures.
- Because AutoInit initializes each candidate appropriately, the search is accelerated and better networks are discovered.

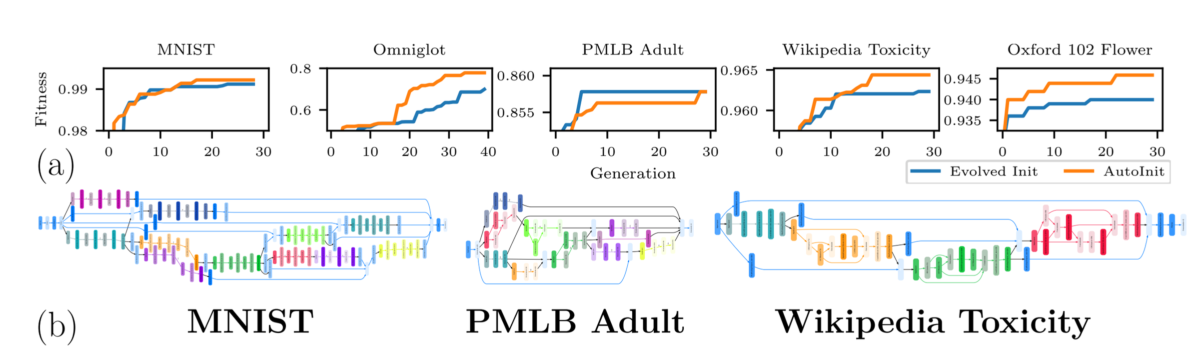

Figure 7: Evaluation of AutoInit with neural architecture search. (a) Performance improvement over generations in the five tasks. AutoInit outperforms the evolved initialization on four tasks and matches it on one. (b) Representative networks evolved with AutoInit. Although the networks are distinct, AutoInit initializes them properly, leading to good performance in each case.

Discovering Better Activation Functions

- Activation function discovery is another AutoML opportunity.
- Proper initialization leads to higher performance and the discovery of better activation functions.

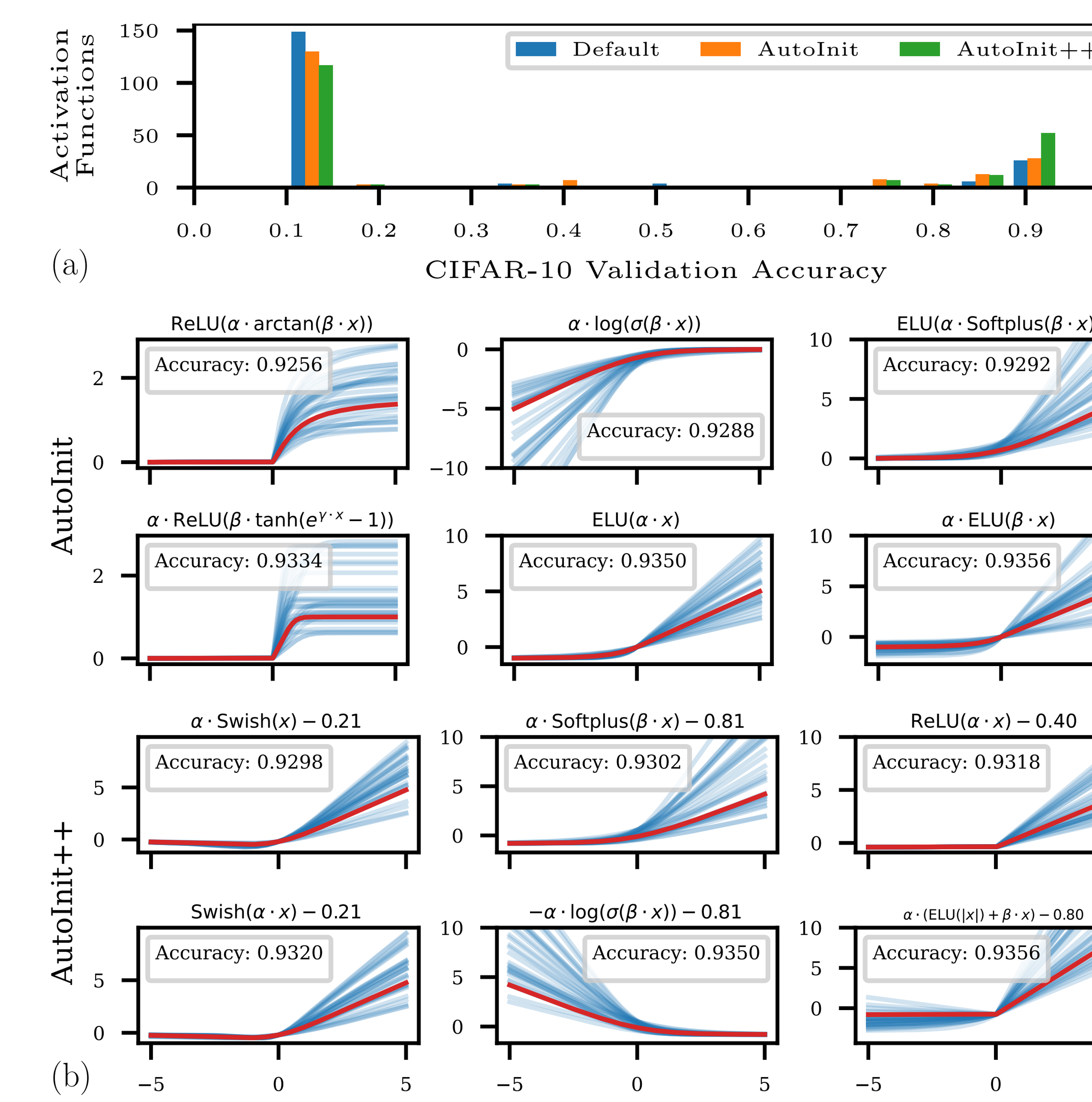

Figure 8: Evaluation of AutoInit with activation function discovery. (a) Distribution of accuracies achieved with 200 activation functions and different weight initialization strategies. AutoInit and AutoInit++ make training more stable and allow more high-performing activation functions to be discovered than the default initialization does. (b) High-performing activation functions. The red line shows the function at initialization, with $\alpha = \beta = \gamma = 1$. The blue lines show the shapes the activation function takes during training, created by sampling α, β, γ from $\mathcal{U}(0.5, 2.0)$. AutoInit's flexibility should turn out useful for developing new activation functions in the future.

AutoInit Software

```
# Install
pip install git+https://github.com/cognizant-ai-labs/autoinit.git

# Import
from autoinit import AutoInit

# Initialize
training_model = AutoInit().initialize_model(training_model)
```

Contact

My website has links to my email, LinkedIn, Google Scholar, and CV.

garrettbingham.com