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 scales its input by α and scales the output by a factor of β.
- If the input to the layer has mean μ_in and variance var, after applying the layer, the output signal will have mean μ_out = αμ_in + βμ_out and variance var_out = var.(1)

- If |β| > 1, the network will suffer from a mean shift and vanishing signals.
- If |β| < 1, the network will suffer from a mean shift and vanishing signals.
- AutoInit calculates weight initializations so that α = 0 and β = 1, avoiding the issues of mean shift and exploding/vanishing signals.

![Image](image1.png)

Figure 1: AutoInit maintains |β| = 1 to avoid exploding or vanishing signals.

Contribution: The AutoInit Framework

- AutoInit uses f functions to map input mean and variance to output mean and variance when a layer is applied.
- Functions are derived for each type of layer, e.g., BatchNorm, BatchGlobalNorm, 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:

\[
\theta \equiv \mathcal{N}(0, 1) \\
\mathcal{N}(\mu, \sigma^2) \rightarrow \mathcal{N}(\mu + \sigma, \sigma^2) \\
\mathcal{N}(\mu, \sigma^2) \rightarrow \mathcal{N}(\mu, \sigma^2)
\]

Algorithm 1: AutoInit

Input: Network with layers \( L \), depth \( \ell \), width \( k \)
Output: Layers \( L \) in output layers \( D \) \( \in \mathbb{N} \) initializations output layers \( D \) \( \in \mathbb{N} \)

Initializations:

- layers-in \( \equiv \{ \ell, (\ell + 1) \} \)
- \( \beta = 1 \)

for layer in layers-in do

- \( \alpha = \mathcal{N}(0, 1) \)
- \( \mu = \mathcal{N}(0, 1) \)
- \( \sigma = \mathcal{N}(0, 1) \)
- \( \beta = \mathcal{N}(0, 1) \)
- \( \alpha = \mathcal{N}(0, 1) \)
- \( \mu = \mathcal{N}(0, 1) \)
- \( \sigma = \mathcal{N}(0, 1) \)
- \( \beta = \mathcal{N}(0, 1) \)

Algorithm 1: AutoInit

Robustness to Hyperparameter Variation

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

![Image](image2.png)

Figure 2: Signal propagation in AlexNet/CNN 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 adapts to accommodate these settings. At layer 6, with weights scaled as in (c), AutoInit scales the weights appropriately to return the variance to approximately 1.0, stabilizing training in each 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.

![Image](image3.png)

Figure 3: Signal propagation in residual networks. Gaussian input was fed to the networks and empirical variance computed at each layer. Since BatchNormalization and Add are coerced as individual layers in this fashion, the total number of layers is different from that in the architecture name (e.g., ResNet-104 has 164 convolutional layers but over 500 total layers). The default initialization causes exploding signals, while AutoInit causes signal propagation to be stable.

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.

![Image](image4.png)

Figure 5: 5-Net CIFAR-10 test accuracy for AutoInit vs. LSTM with different numbers of samples. S. Each evolution is repeated 10 times, the shaded area shows the maximum and minimum accuracy among all trials. AutoInit is consistent, but LSTM struggles when S is small or the network is deep.

Discovering Better Activation Functions

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

![Image](image5.png)

Figure 8: Evaluation of AutoInit with neural architecture search. (a) Performance improvement over generalizing in the fine-tuning. AutoInit outperforms the initialized end-to-end network on fine-tuning and matches it on fine-tuning. (b) Representative network evolved with AutoInit. Although the networks are detect, AutoInit initializes them properly, leading to good performance in each case.

AutoInit Software

- Install pip install git+https://github.com/cognizant-ai-labs/autoinit.git
- Figure from autoinit import AutoInit
- Initialize training_loop = autoinit.AutoInit(initialization=training_model)

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

garrettbingham.com