Journal of Uncertain Systems
Vol.13, No.3, pp.167-170, 2019
Online at: www.jus.org.uk

Smaller Standard Deviation for Initial Weights Improves Performance of Classifying Neural Networks: A Theoretical Explanation of Unexpected Simulation Results

Diego Aguirre, Philip Hassoun, Rafael Lopez, Crystal Serrano
Marcoantonio R. Soto, Andrea Torres, Vladik Kreinovich*
Department of Computer Science, University of Texas at El Paso, 500 W. University
El Paso, Texas 79968, USA

Received 12 April 2019; Revised 7 May 2019

Abstract

Numerical experiments show that for classifying neural networks, it is beneficial to select a smaller deviation for initial weights than what is usually recommended. In this paper, we provide a theoretical explanation for these unexpected simulation results.

Keywords: Keywords: deep learning, neural networks, classification problems

1 Formulation of the Problem

Classifying neural networks: a brief reminder. In deep learning [2], a neural network that classifies into C classes works as follows:

- Computations start with the values $x_1, \ldots, x_v$ describing the object that we need to classify. These values are the input signals for the first layer: $s_{i,1} = x_i$ for $i = 1, \ldots, n_0 \stackrel{\text{def}}{=} v$.
- On each layer ℓ, input signals $s_{1,\ell}, \ldots, s_{n_{\ell-1},\ell}$ to this layer get transformed into outputs

$$s_{i,\ell+1} = \max\left(\sum_{j=1}^{n_{\ell-1}} w_{i,j}^{(\ell)} \cdot s_{j,\ell} - w_{i,0}^{(\ell)}, 0\right),$$

  where $w_{i,j}^{(\ell)}$ are appropriate weights, and n_ℓ denotes the number of neurons in the ℓ-th layer.
- These outputs serve as inputs to the next layer $\ell + 1$.

We do this until we reach the last layer L, where we use *softmax*; namely:

- based on C neural outputs $z_i = s_{i,L}$,
- we compute the probability p_i of being in a i-th class as

$$p_i = \frac{\exp(\beta \cdot z_i)}{\sum_{j=1}^{C} \exp(\beta \cdot z_j)}$$

  for some $\beta > 0$.

*Corresponding author.
Emails: daguirre6@utep.edu (D. Aguirre), pchassoun@miners.utep.edu (P. Hassoun), relopez6@miners.utep.edu (R. Lopez), cserrano5@miners.utep.edu (C. Serrano), mrsoto3@miners.utep.edu (M.R. Soto), aftorres@miners.utep.edu (A. Torres), vladik@utep.edu (V. Kreinovich).

Selecting initial weights is important. We have several objects for which the know the corresponding classification $c^{(k)}$. Each of these objects is characterized by the numerical values $x_1^{(k)}$, ..., $x_v^{(k)}$ of the corresponding quantities.

Training a neural network means selecting the weights $w_{i,j}^{(\ell)}$ for which the classification produced by the neural network is, in some reasonable sense, the closest to the actual classification $c^{(k)}$.

To perform this training, we start with some initial values of all these weights $w_{i,j}^{(\ell)}$, and then we iteratively update them until we get a good match. How fast the network learns depends on how well we selected the initial weights.

If the initial weights are too far from the actual ones, training takes much longer.

How initial weights are selected now. According to the current recommendations [3], weights should be selected layer-by-layer, starting with the input layer.

- For each neuron i in the currently considered layer ℓ, we start with weights $r_{i,j}^{(\ell)}$ uniformly distributed on some interval $[-Z, Z]$.
- Then, we apply Gram-Schmidt orthonormalization to the vectors

$$r_i^{(\ell)} = \left(r_{i,1}^{(\ell)}, \cdots, r_{i,n_{\ell-1}}^{(\ell)}, r_{i,0}^{(\ell)}\right).$$

  Specifically, sequentially, for $i = 1, \ldots, n_\ell$, we compute the new vectors

$$\widetilde{w}_i^{(\ell)} = \frac{r_i^{(\ell)} - \sum\limits_{j=1}^{i-1} \left(r_i^{(\ell)}, \widetilde{w}_j^{(\ell)}\right)}{\left\| r_i^{(\ell)} - \sum\limits_{j=1}^{i-1} \left(r_i^{(\ell)}, \widetilde{w}_j^{(\ell)}\right) \right\|},$$

  where, as usual, $(a, b) \stackrel{\text{def}}{=} \sum\limits_j a_j \cdot b_j$ and $\|a\| \stackrel{\text{def}}{=} \sqrt{\sum\limits_j a_j^2}$.

- Then, we preliminarily select the weights $\widetilde{w}_i^{(\ell)}$ for neurons from this layer, and use the already selected weights for the neurons from the previous layers.
- For each neuron i from the currently analyzed ℓ-th layer, we select a small sample of size K (where K is a pre-selected number) from the list of all available input patterns. For each pattern

$$\left(x_1^{(k)}, \cdots, x_v^{(k)}\right)$$

  from the selected sample, we perform the neural network computations up to this layer, and get the outputs $s_{i,\ell+1}(k)$ of this neuron. We then compute the standard deviation $\sigma_{i,\ell+1}$ of the resulting K values by using the usual statistical formulas:

$$\overline{s}_{i,\ell+1} = \frac{1}{K} \cdot \sum_{k=1}^{K} s_{i,\ell+1}(k)$$

  and

$$\sigma_{i,\ell+1} = \sqrt{\frac{1}{K-1} \cdot \sum_{k=1}^{K} \left(s_{i,\ell+1}(k) - \overline{s}_{i,\ell+1}\right)^2}.$$

  After this, we select the re-scaled values

$$w_{i,\ell}^{(\ell)} = \frac{\widetilde{w}_i^{(\ell)}}{\sigma_{i,\ell+1}}$$

  as the initial values of the weights of the ℓ-th layer.

One can check that if we use these weights for the ℓ-th layer, then for each neuron i on this layer, the standard deviation of the K signals coming from this neuron will be equal to $\sigma_0 = 1$.

Then we freeze these weights and go to the next layer.

To select the weights from the last (linear) layer, we try our best to match the results with the desired outputs.

Empirical observation that needs explaining. One of us (DA) tried to use $\sigma_0 < 1$ in the above algorithm. Specifically, at each layer, at the last step, instead of the weights $\widetilde{w}_i^{(\ell)}/\sigma_{i,\ell+1}$, we selected somewhat different initial weights

$$W_{i,\ell}^{(\ell)} = \sigma_0 \cdot \frac{\widetilde{w}_i^{(\ell)}}{\sigma_{i,\ell+1}}.$$

On several classification examples, he got much better results for $\sigma_0 = 0.5$ than for the usually recommended value $\sigma_0 = 1$. Once he got this result, he tried even smaller values $\sigma_0 = 0.4$ and $\sigma_0 = 0.3$. In turns out, surprisingly, that the smaller σ_0, the better the results.

What we do in this paper. In this paper, we provide a theoretical explanation for this unexpected empirical result.

Comment. This explanation was first announced in [1].

2 Analysis of the Problem

If we use $\sigma_0 < 1$, then on the first layer, instead of the original initial weights $w_{i,j}^{(\ell)}$, we get new weights $W_{i,j}^{(\ell)} = \sigma_0 \cdot w_{i,j}^{(\ell)}$. Then, with the same weights on other layers, we get standard deviation σ_0 on each of them. After L layers, we get new signals $Z_i = \sigma_0 \cdot z_i$.

3 Our Explanation

Until we get to the last layer, we do not use the actual output. So we do now know the actual probabilities $q_1, \ldots, q_C$ of different classes. It is therefore reasonable to select the initial weights so that:

- the resulting probabilities p_i
- are, on average, as close to the actual (unknown) probabilities q_i as possible.

The closeness can be described:

- either by the Euclidean distance

$$\|p - q\|^2 = \sum_i (p_i - q_i)^2 \to \min,$$

- or by any other strictly convex function $C(p, q)$ of p, e.g., by relative entropy.

So, we minimize the expected value $\int C(p, q) \cdot \rho(q)\, dq$.

At this stage, we do not have any information about the probabilities of different classes. So, it is reasonable to assume that this criterion does not change if we simply re-order the classes.

Since the function $C(p, q)$ is convex, there is only one vector p for which this minimum is attained; see, e.g., [4]. Thus, the optimal tuple p should also be invariant under such re-ordering, i.e., $p_i = p_j$ for all i and j. Hence, in the optimal case, we get $p_i = 1/C$ for all i.

The use of $\sigma_0 < 1$ places all z_i closer to 0. Thus, the corresponding softmax values p_i are closer to the optimal values $1/C$. This explains why the results of using $\sigma_0 < 1$ are better.

There is a minor difference between z_i and 0 – and thus, between p_i and the optimal values $1/C$. The smaller σ_0, the smaller this difference.

This explains why the smaller σ_0, the better the results.

Comment. It should be mentioned that while decreasing σ_0 to a smaller positive number makes the classification faster, we cannot decrease this value all the way to $\sigma_0 = 0$. Indeed, in this case, all the weights will be 0, so the weights for all the neurons in each layer will be the same. Since the training is a deterministic process, the weights of all the neurons in each layer will be updated in exactly the same way – thus, all the neurons in each layer will remain identical. In particular, we will have identical signals z_i at the last layer – and thus, such network will always assign equal probabilities $1/C$ to each of C classes.

Acknowledgments

This work was partially supported by the US National Science Foundation via grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence).

References

[1] Aguirre, D., Hassoun, P., Lopez, R., Serrano, C., Soto, M.R., Torres, A., and V. Kreinovich, Smaller standard deviation for initial weights improves neural networks performance: a theoretical explanation of unexpected simulation results, *Abstracts of the 23rd Joint UTEP/NMSU Workshop on Mathematics, Computer Science, and Computational Sciences*, El Paso, Texas, November 3, 2018.

[2] Goodfellow, I., Bengio, Y., and A. Courville, *Deep Leaning*, MIT Press, Cambridge, Massachusetts, 2016.

[3] Mishkin, D., and J. Matas, All you need is a good init, *Tetrahedron*, vol.69, no.14, pp.3013–3018, 2015.

[4] Rockafeller, R.T., *Convex Analysis*, Princeton University Press, Princeton, New Jersey, 1997.