Topic Proposal: An Empirical Study of *Relaxed Activation Identity*'s Effects on First-Order Optimization

Dennis Alexander Mertens Velasquez (i6206990)

1 Introduction

Suppose we have n training examples $Z_0 = [z_0^0, \ldots, z_0^{n-1}]$ denoting inputs and $Z_T = [z_T^0, \ldots, z_T^{n-1}]$ targets, and a feed-forward differentiable circuit of depth T as in figure 1. The circuit is composed of T layers; each a non-linear differentiable[1] function—denoted f_t, where $0 \leq t \leq T-1$. The input and output of each layer are denoted z_t and z_{t+1}, respectively. Additionally, each layer has a corresponding set of parameters θ_t. Without any loss of generality, the relation between input and output is described by

$$z_{t+1} = f_t(x_t, \theta_t). \tag{1}$$

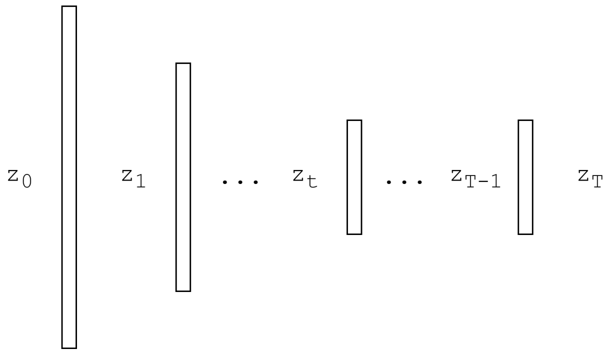

Figure 1: A feed-forward differentiable circuit of depth T; z_0 denotes the input, $z_1, \ldots, z_{T-1}$ denote the hidden activations, and z_T denotes the output. Each layer $t \in \{0, \ldots, T-1\}$ has parameters θ_t and activation function f_t.

When applying backpropagation (BP) and gradient descent (GD), we optimize parameters $\Theta = [\theta_0, \ldots, \theta_{T-1}]$ by nudging them according to their gradients.[2] In such a setting, the

[1] We understand by *differentiable* a function whose first-order gradient is defined everywhere.

[2] To be clear: BP is responsible for propagating the errors through the circuit, and GD is responsible for adjusting the parameters accordingly.

gradients of θ_t depend on the gradients and activations of layers $t+1, \ldots, T-1$. As we will see in section 2, the nonlinearities through which gradients have to flow through, can be destructive. The objective behind this proposal is to transform the sequential dependencies such that gradients flow through less destructive nonlinearities. I think we can achieve this by "relaxing" equation 1.

2 Motivation

...

3 Approach

Extending the the formula introduced in section 1, equation 1, let z_t^i denote the target activation of the tth layer given the ith training example, and $\hat{z}_t^i$ its estimate. For $t \in \{0, T\}$, the targets are given. For $t \in \{1, \ldots, T-1\}$, assume auxiliary trainable parameters $Z_t = [z_t^0, \ldots, z_t^{n-1}]$ and define the hidden activations as

$$\hat{z}_{t+1}^i = f_t(z_t^i, \theta_t). \tag{2}$$

Usually, any activation $\hat{z}_{t+1}^i$ is defined in terms of the preceding $\hat{z}_t^i$, hence the sequential dependence. In contrast, the dependence is broken by introducing an auxiliary z_t^i. Suppose we train such a system by only minimizing the output layer's error. In that case, it will not generalize because only the last layer will learn to map z_{T-1}^i to z_T^i whilst every other part of the circuit will settle for arbitrary configurations. To fix this, we must enforce that

$$\hat{z}_t^i \approx z_t^i, \forall t \in \{1, \ldots, T-1\}, \forall i \in \{0, \ldots, n-1\}. \tag{3}$$

During training, some loss function $\mathcal{L}(Z_T, \hat{Z}_T) = \sum_{i=0}^{n-1} L_y(z_T^i, \hat{z}_T^i)$ is always required, where L_y measures the output error.[3] By extending the loss to include terms that regulate the hidden activations, the condition in equation 3 is enforced. That is,

$$\mathcal{L}(Z_T, \hat{Z}_T) = \sum_{i=0}^{n-1} L_y(z_T^i, \hat{z}_T^i) + \sum_{t=1}^{T-1} \sum_{i=0}^{n-1} L_h(z_t^i, \hat{z}_t^i), \tag{4}$$

where L_h measures the hidden error.

I call this approach *relaxed activation identity* (RAI) because it is based on the idea of—in a sense—relaxing the identity of an activation. Without relaxation, we have $z_t^i = \hat{z}_t^i$, where the distinction between the prediction $\hat{z}_t^i$ and the target z_t^i is redundant. With relaxation, we eventually have $z_t^i \approx \hat{z}_t^i$. Yet, at times, these two may differ. Hence, the activation's identity is "relaxed".

[3]Usually, loss functions are augmented with a regularization term, but it is irrelevant in this context, and thus not included.

4 Preliminary Evidence

To reproduce the following experiments, find the Jupyter notebooks `rai-demo.ipynb` and `rai-grads.ipynb` in the GitHub repository at https://github.com/damertensv/rai. All experiments are based on the multilayer perceptron (MLP) [2]. To indicate the structure of a particular MLP, we write $(l_0, \ldots, l_T)$ for the layer dimensions, where l_t is a positive integer, followed by a mention of the activation functions used.

4.1 Shallow Circuit Comparison

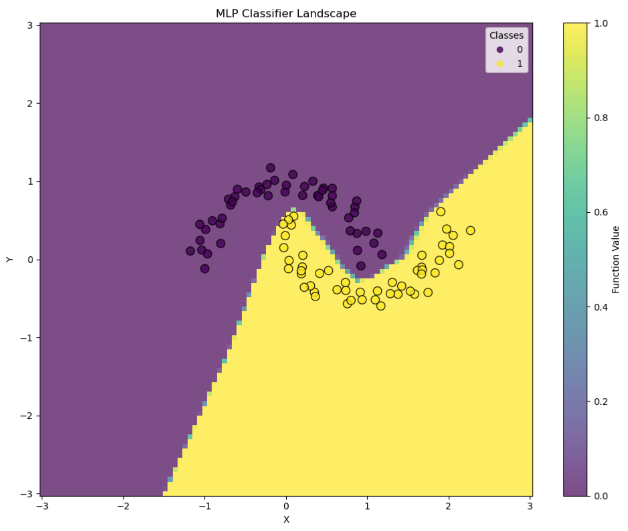

Figure 2: An MLP trained on the Moons dataset from Scikit Learn [3] with layer dimensions (2, 42, 100, 100, 42, 1), relu hidden activations, and sigmoid output activation.

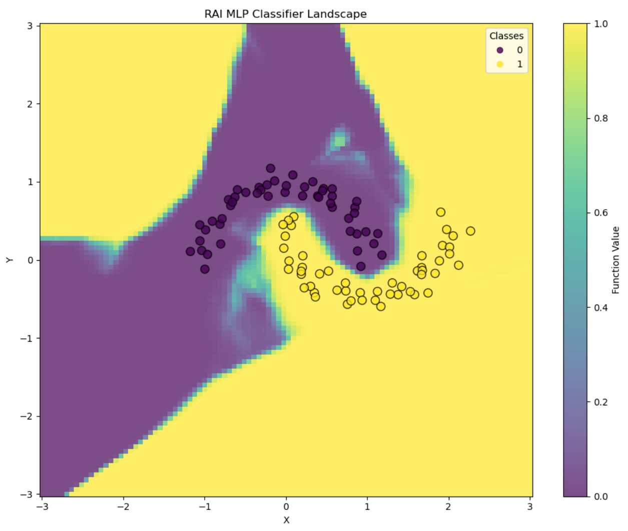

Figure 3: A RAI MLP trained on the Moons dataset from Scikit Learn [3] with layer dimensions (2, 42, 100, 100, 42, 1), relu hidden activations, and sigmoid output activation.

4.2 Deep Circuit Comparison

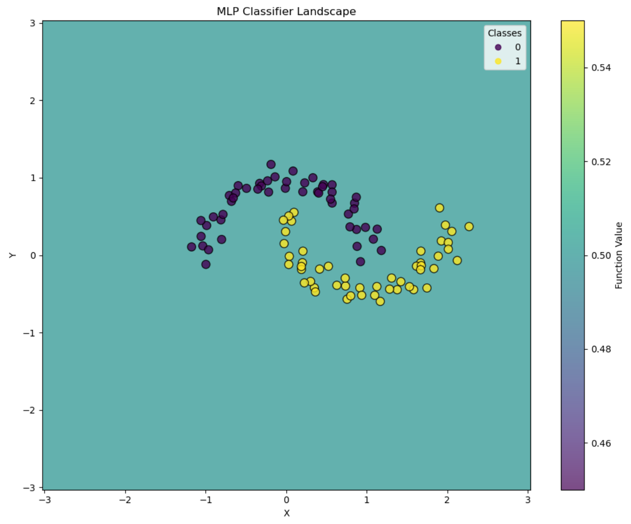

Figure 4: An MLP trained on the Moons dataset from Scikit Learn [3] with layer dimensions (2, 100, . . . , 100, 1), 100 hidden layers, and sigmoid activations everywhere.

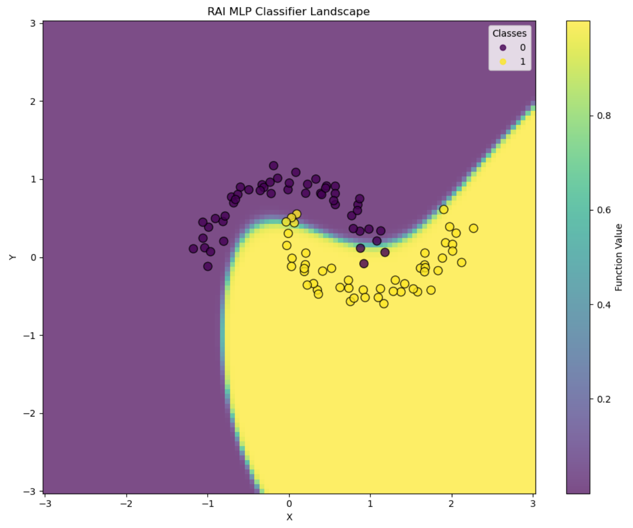

Figure 5: A RAI MLP trained on the Moons dataset from Scikit Learn [3] with layer dimensions (2, 100, . . . , 100, 1), 100 hidden layers, and sigmoid activations everywhere.

5 Preliminary Discussion

From the landscapes obtained in the shallow experiment, illustrated in figures 2 and 3, we can see that RAI retains the ability to generalize, albeit it seems to follow slightly different inductive biases—based on the qualitative differences around the decision boundaries.

From the landscapes obtained in the deep experiment, illustrated in figures 4 and 5, we can see that the standard application of BP and GD fails to train the deep MLP. In contrast, RAI manages to train the same model and without the need for explicit tricks such as skip connections [1].

Although it is not visible in the plots, for anyone running the notebooks `rai-demo.ipynb` and `rai-grads.ipynb`, it should be clear that RAI exhibits a weaker bias on the shape of the decision boundary w.r.t. the standard application of BP and GD. While the MLP settles for similar boundaries across different i.i.d. runs, RAI does not. In higher dimensions, this is likely to be an issue. Hence, the use of regularization techniques will be necessary for serious applications.

Finally, since RAI requires caching the intermediate activations, the approach is incompatible with batched, mini-batched, and stochastic gradient descent methods. At least at first glance. Though there might be workarounds to be discovered.

5.1 Research Questions

...

References

[1] Kaiming He et al. *Deep Residual Learning for Image Recognition.* 2015. arXiv: `1512.03385 [cs.CV]`. URL: `https://arxiv.org/abs/1512.03385`.

[2] Fionn Murtagh. "Multilayer perceptrons for classification and regression". In: *Neurocomputing* 2.5 (1991), pp. 183–197. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/0925-2312(91)90023-5`. URL: `https://www.sciencedirect.com/science/article/pii/0925231291900235`.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.