 doi:10.20944/preprints202209.0231.v1

Article

Evidence-Based Regularization for Neural Networks

Giuseppe Nuti [1,†,‡], Andreea-Ingrid Cross [1,‡] and Philipp Rindler [2,‡]*

[1] UBS Investment Bank
[2] UBS Business Solutions
* Correspondence: giuseppe.nuti@ubs.com (G.N.), andreea-ingrid.cross@ubs.com (A.C.), philipp.rindler@ubs.com (P.R.)
† Current address: 1285 Avenue Of The Americas, New York City, 10019 New York, United States
‡ These authors contributed equally to this work.

Abstract: Numerous approaches address over-fitting in neural networks: by imposing a penalty on the parameters of the network (L1, L2, etc); by changing the network stochastically (drop-out, Gaussian noise, etc.); or by transforming the input data (batch normalization, etc.). In contrast, we aim to ensure that a *minimum amount of supporting evidence* is present when fitting the model parameters to the training data. This, at the single neuron level, is equivalent to ensuring that both sides of the separating hyperplane (for a standard artificial neuron) have a minimum number of data points — noting that these points need not belong to the same class for the inner layers. We firstly benchmark the results of this approach on the standard Fashion-MINST dataset, comparing it to various regularization techniques. Interestingly, we note that by nudging each neuron to divide, at least in part, its input data, the resulting networks make use of each neuron, avoiding a hyperplane completely on one side of its input data (which is equivalent to a constant into the next layers). To illustrate this point, we study the prevalence of saturated nodes throughout training, showing that neurons are activated more frequently and earlier in training when using this regularization approach. A direct consequence of the improved neuron activation is that deep networks are now easier to train. This is crucially important when the network topology is not known *a priori* and fitting often remains stuck in a suboptimal local minima. We demonstrate this property by training a network of increasing depth (and constant width): most regularization approaches will result in increasingly frequent training failures (over different random seeds) whilst the proposed *evidence-based regularization* significantly outperforms in its ability to train deep networks.

Keywords: neural networks; regularization; deep networks

Citation: Nuti, G.; Cross, A.; Rindler, P. Title. *Preprints* **2022**, *1*, 0. https://doi.org/

1. Introduction

Regularization is a supplementary element added to neural networks in order to improve their ability to generalize — but not necessarily the performance on training data [1, Chapter 5] (see [2] for a comprehensive overview of various approaches to regularization). Broadly speaking, there are five categories of regularization:

- *Regularization of the training data*: This form of regularization involves augmentation of the data fed to the layers of a neural network or the introduction of transformations to expand the observed set of training samples. The most popular methods in this category are batch normalization [3] and layer normalization [4]. A related form of regularization is the addition of randommess to input data or the labels. One of the most popular methods in this category is the addition of Gaussian noise to input data [5,6], label smoothing [7] and Dropout [8,9].
- *Regularization via the network architecture*: Whereby the network topology is selected to match certain assumptions regarding the data-generating process. Such regularization limits the search space of models or introduces certain invariances. Popular methods in this category are limiting the size (either breadth or depth) of the network, convolutional or pooling layers [10,11], or skip connections [12,13].
- *Regularization term added to the loss function*: This is a term that is independent of the targets which changes the loss, and thereby the gradient step, in a certain manner.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

Common approaches are weight decay by adding the norm of the weights to the loss (typically either L1, L2, or the sum of those) [14,15] and weight smoothing by adding the norm of gradients [15].

- *Regularization via the error function itself*: This includes approaches to make optimization robust to class imbalance [16] or multi-task learning [17,18].
- *Regularization via the optimization procedure*: The final form of regularization is via the optimization procedure itself — or, more precisely, the interplay between loss function, regularization terms, and optimization iterations. The most common forms of regularization in this category are mini-batch optimization [19], early stopping [20], initialization approach [21,22], learning-rate scheduling [23], and the specific update rules such as Momentum, RMSProp, or Adam [24].

2. Evidence-Based Regularization

The basic idea underpinning this approach to regularization is simple: for each neuron, we aim to have a minimum amount of evidence to support the fitted hyperplane dividing the data. To illustrate the need for sufficient evidence, Fig. 1 presents the same (2-dimensional) inputs to a neuron, with the starting hyperplane as a dotted line, randomly set at initialization. The standard back-propagation algorithm would likely result in the split on the left (Fig. 1, left). Since this neuron may represent the original data transformed by numerous layers, we may ask if the left hyperplane could be leading to an over-fitting of the data. Conversely, if we add a penalty when the number of observations on each side of the hyperplane falls below a user-defined threshold, the fitted neuron would likely result in the right configuration (Fig.1, right) which is supported by a larger number of points and thus may represent a better generalization of the data.

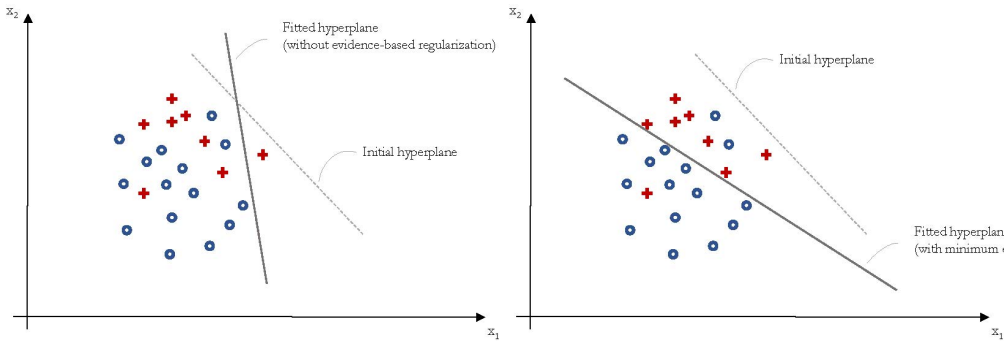

Figure 1. An example of a trained hyperplane potentially over-fitting the data (left) versus a hyperplane supported by more evidence (right).

Mathematically, we have a wide range of choices to penalize for a low number of data points on either side, noting that it can depend on the activation function if we are to insert the penalty after activation. For a $y_i = \tanh(x_i)$ activation, where y_i represents the output of the ith training sample's activation function, the regularization penalty, r,applicable to the entire batch, can be defined as:

$$r = \min\left[\left(-\sum^{i\in B} \mathbb{1}_{y_i<0}\, y_i\right) - \lambda, \left(\sum^{i\in B} \mathbb{1}_{y_i>0}\, y_i\right) - \lambda, 0\right] \tag{1}$$

Where λ is the minimum number of observations we require on each side of the hyperplane and B is the number of samples in the batch[1]. To back-propagate through the

[1] In Eq. 1, the term $\mathbb{1}_{\mathcal{A}}$ stands for the indicator function which returns 1 when the condition $\mathcal{A}$ is true, zero otherwise.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

min function, we use the standard $softmin$ substitute[2] or directly use the min operator if implementing this in an environment that supports auto-differentiation, such as *TensorFlow*.

One interesting property of this regularization is that it nudges the dividing hyperplane towards the neuron's data if it finds itself outside at initialization. This property is key in ensuring that each neuron is actively used in the network and, as we will show in Sections 3.1 and 3.2, in aiding the training of deep networks that may be prone (fully or in part) to neurons that are saturated and thus have near-zero gradient.

Importantly, this approach to normalization goes across a mini-batch and is not applicable to single-sample updates. Moreover, the minimum number of observations, λ, which is an intuitive parameter to set, needs to be viewed with respect to the batch size: as an example, a $\lambda = 1$ may be quite appropriate for an image recognition task with a batch size of 1,000 (as the process may not be overly noisy), whilst a financial data application may require $\lambda = 10$ or more for the same batch size.[3] Other activation functions can be derived in a similar way to Eq. 1 when the output is bound. For unbounded activation, such as *ReLU*, we would need to bound the output in some way. Interestingly, if we consider a radial basis function activation, defined for a centroid c over d input dimensions as $y = e^{-\sum(x_d - c_d)^2}$, this regularization ensures that the sum of activated samples is greater or equal to λ — in essence, a minimum number of neighbors if we think of this neuron as a *k-Nearest Neighbor* algorithm.

Finally, a note on extrapolation: this regularization is primarily focused on ensuring that there is a minimum amount of evidence when setting each neuron's hyperplane. When it comes to extrapolation, this approach will not yield a network that can generalize better: if we are concerned with querying samples that may be significantly different from the training inputs, we can employ specialized techniques, such as Selective Classification [25], specifically designed to address this issue.

3. Empirical Results

Before analyzing the effects of this approach on neuron activation, we explore how it performs as a regularization method. Using the *Fashion-MNIST* [26] dataset with a 5% label noise [27][28] on a neural network (with a batch size of 1,000, 5 hidden layers, 10 hidden units and 50 epochs)[4], we test the effectiveness of the Evidence-Based Regularization against standard regularization methods: L_1, L_2, $Dropout$ as well as against a non-regularized network[5]. We use the default parameters for all the regularization methods (i.e: the *regularization parameter* is equal to 0.01 for both L_1 and L_2 techniques and the *dropout rate* is equal to 0.5 for the *Dropout* method, 10 threshold for the Evidence-Based Regularization method).

In Figure 2 we can observe the training and validation accuracy results of the Evidence-Based Regularization outperforming the traditional regularization methods.

To analyse how this approach performs as the signal-to-noise ratio decreases (in this case disitorting the inputs), we inject a Gaussian noise component (zero mean; increasing standard deviation) to the normalized input data — using the default parametrization for all the regularization methods. The results in Table 1 support the use of this approach as a performant regularization method.

---

[2] To implement the differentiable version of the min operator, use the softmin version: $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{B} e^{z_j}} \quad for\ i = 1, 2, \ldots, B$. In our implementation, we set $z_i = \alpha\, y_i$ with $\alpha = -0.95/\lambda$ to automatically scale the strength of the min function w.r.t. the minimum number of observations, λ.

[3] We use a default of $\lambda = 0.10\, B/c$, where B is the batch size and c is the number of categories or $c = 1$ for regression problems (noting that for categorization problems that have significantly imbalanced classes, B/c is set to the number of samples in the batch that belongs to the category with the *least* representation).

[4] We use seed 9218 for our data initialisation

[5] Each regularization method is tested on the same network architecture described above.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

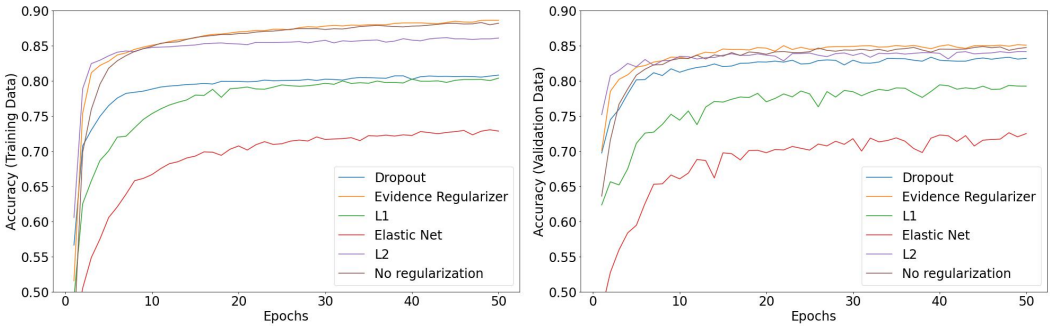

Figure 2. Training (left) and validation (right) accuracy for different regularization approaches.

Table 1. Training and validation accuracy for Fashion-MNIST data with increasing noise for various regularization methods (EBR: Evidence-Based Regularization).

Std. Dev.	0	0.1	0.25	0.5	0.75	1	2	3	4
Training Set									
L1	0.80	0.80	0.79	0.76	0.69	0.66	0.32	0.27	0.10
L2	0.86	0.86	0.85	0.82	0.80	0.78	0.68	0.58	0.46
Dropout	0.80	0.80	0.79	0.77	0.75	0.71	0.60	0.48	0.39
EBR	0.89	0.88	0.86	0.84	0.81	0.78	0.68	0.59	0.51
Validation Set									
L1	0.80	0.79	0.78	0.75	0.64	0.58	0.24	0.21	0.10
L2	0.84	0.84	0.84	0.83	0.82	0.80	0.70	0.60	0.49
Dropout	0.82	0.82	0.83	0.81	0.80	0.77	0.68	0.58	0.54
EBR	0.85	0.85	0.85	0.85	0.83	0.82	0.74	0.65	0.55

3.1. Large learning rates

Setting an appropriate learning rate for training is a critical choice to create a robust model. A small learning rate risks overfitting but a large learning rate risks divergence during training [23]. A common strategy to balance the two is to use cycle learning [29], where the learning rate starts out at a lower limit, then increases to a maximum over a set period of epochs, and finally decreases again to a lower limit several magnitudes below the initial learning rate.

We hypothesize that our proposed regularizer acts as a natural *deterrent* against rogue learning with high learning rates in the early stages of training. The reason is that the additional gradient applied to nodes with a low activation on either side of its hyperplane ensures that the learning algorithm cannot take steps that would result in extreme weights.

To test this hypothesis, we train randomly initialized neural networks[6] with high and low learning rates on the Fashion-MNIST data. To ensure that there is no contamination from adaptive learning rates, we use vanilla stochastic gradient descent (without momentum). We train the network for 50 epochs with and without regularization and compare the cross-entropy loss (Figure 3) as well as the resulting accuracy (Figure 4). We compare results for a high learning rate of 1.0 and a low learning rate of 0.0001.

The results clearly show that, without regularization, a high learning rate results in erratic improvements of the loss function while low learning rates result in much better behaved training. However, the accuracy results show that the low learning rate results in very slow progress on accuracy of the model. Conversely, applying the regularizer with a high learning rate results in smooth learning and rapid progression on accuracy — observable in the early epochs of training.

[6] We use the same network topology detailed in Sec. 3.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

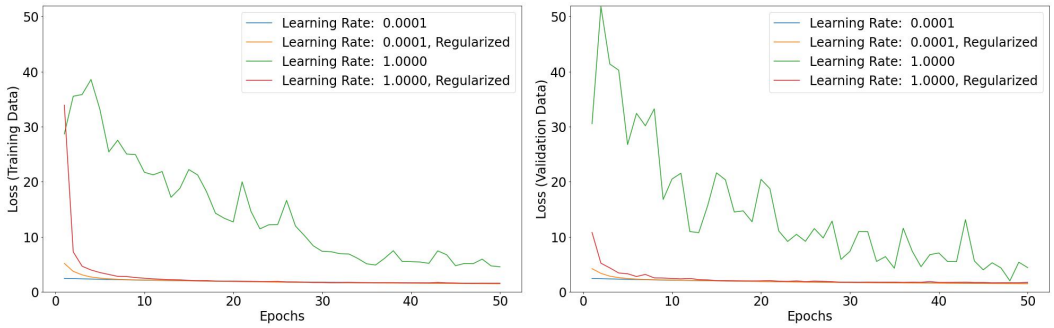

Figure 3. Comparison of cross-entropy loss for different learning rates with and without regularization (left panel shows results for training data, right panel for validation data).

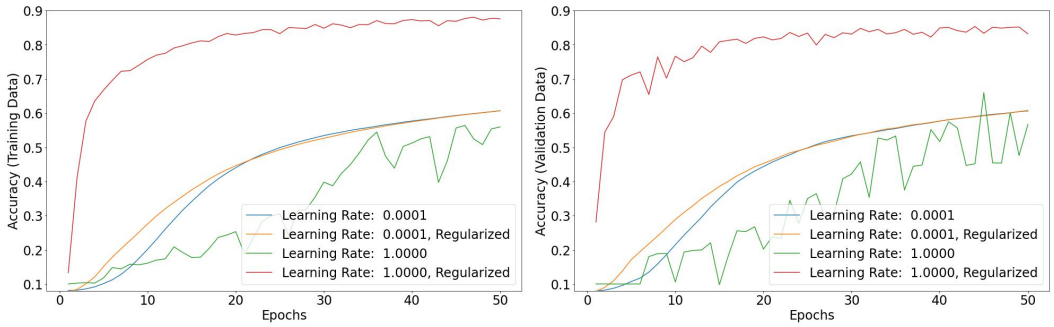

Figure 4. Comparison of accuracy for different learning rates with and without regularization (left panel shows results for training data, right panel for validation data).

3.2. Preventing vanishing and exploding gradients

Interestingly, the proposed regularizer ensures that poorly initialized neurons receive an additional *nudge* during backpropagation. Visually, if the hyperplane of the neuron lies too far away from the data and therefore the activation as defined in (C) is too low then the gradient of that neuron is increased in the direction which increases activation. A similar issue is encountered in weight initialization. At their core, all approaches to weight initialization have in common that they seek to prevent vanishing or exploding gradients during training. Our regularizer can be seen as a gradient stabilizer since it ensures that poorly initializer nodes are pushed to a more *meaningful* configuration. We also suspect that the regularizer prevents vanishing or exploding gradients as such behavior would indicate that a node does not cut through the data but rather lies on one side of the data.

We test this by examining the training progress for different levels of variance for the initializer, with and without the EBR. The stabilizing effect of the EBR should allow for higher variances in the initialization and still result in training progress as the EBR will prevent nodes from remaining dead, i.e. stuck with vanishing or exploding gradients. We use the same network as in other tests and fit the network to the Fashion MNIST data.

The results show that training of the network with the EBR progresses virtually unchanged for variances as high as 5 times the *stable* amount. This shows that the EBR has the property of stabilizing training and makes training progress less dependent on the initial state of the network.

3.3. Training deeper networks

The improved activation highlighted above has a beneficial impact on our ability to train deep and narrow networks — where the vanishing/exploding gradient can hinder the ability of a network to converge onto an acceptable solution (see [30] for a thorough review of this phenomena). To illustrate this, we explore how the Evidence-Based Regularization technique performs, compared to the other traditional regularization methods,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

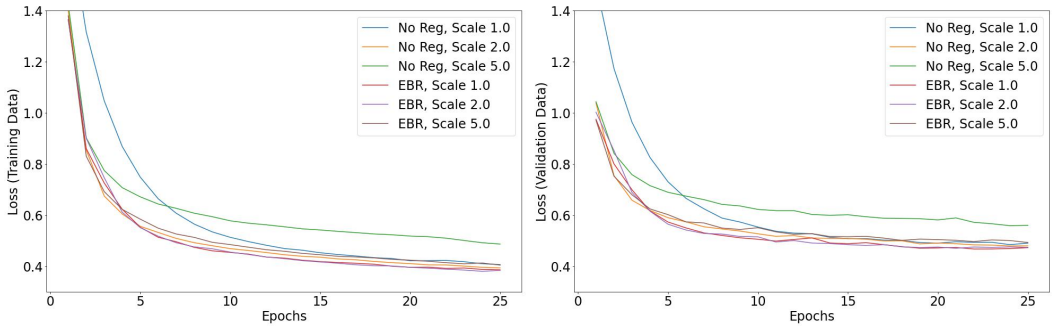

Figure 5. Comparison of cross-entropy loss for different variance scales of the initalizer for layers with and without regularization (left panel shows results for training data, right panel for validation data).

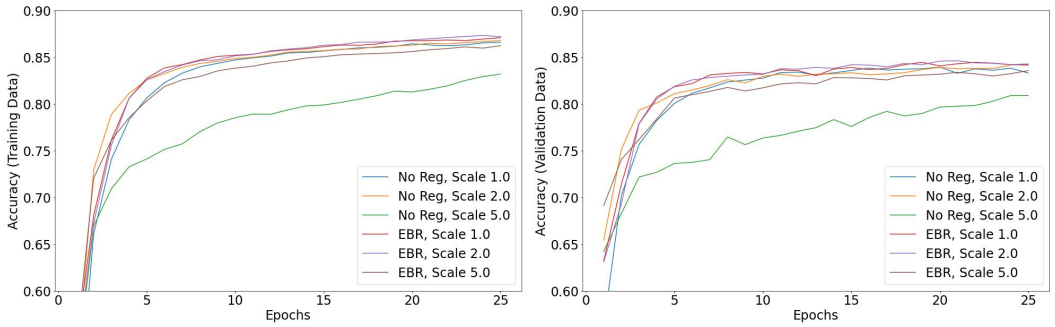

Figure 6. Comparison of accuracy for different variance scales of the initalizer for layers with and without regularization (left panel shows results for training data, right panel for validation data).

when using a narrow but relatively deep neural network: starting from 5 layers and up to a depth of 30, for a width of 10. We perform 100 tests using different seeds for each depth (the 100 different seeds are kept constant across depths and are included in appendix for reproducibility of results), reporting the average validation accuracy at the end of the training period.

We notice from Fig. 7 that the Evidence-Based Regularization technique gracefully drops its accuracy as we increase the network depth, with smaller standard errors across the accuracy results. Conversely, other regularization techniques, such as *Dropout*, degrade significantly from depth 24 and beyond.[7]

4. Concluding Remarks

We propose a simple regularization method based on ensuring that a sufficient amount of evidence supports each neuron's fitted parameters within an artificial neural network. We show that it performs well compared to the currently available regularization methods. Importantly, we show how this approach nudges each neuron's hyperplane towards separating its input data — at least to the extent specified by the regularization parameter. This has various beneficial effects: *(a)* by ensuring that each neuron contributes to the overall solution (less saturated neurons with zero gradient); *(b)* by allowing for more latitude in setting the learning rate since possible large gradient steps that would otherwise bring the separating hyperplane outside of the data will be naturally moved back within the input data; and finally, *(c)* by improving training of deep and narrow neural networks which would otherwise be at risk of getting stuck in a suboptimal local minima. An interesting open question remains around how this approach can be accurately normalized

[7] Please see Appendix B for the same experiments run over the Digits MNIST dataset, which shows similar results.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

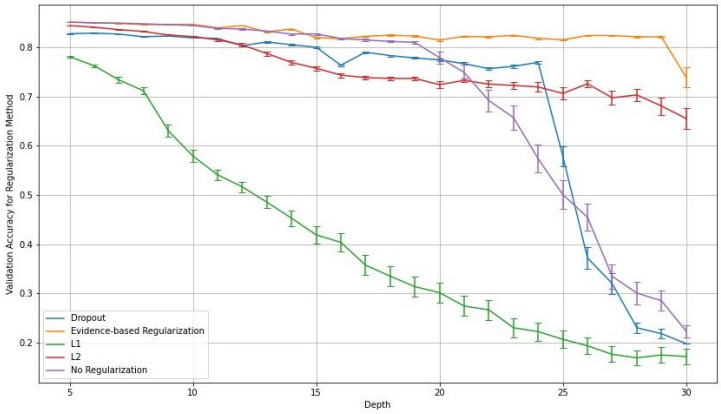

Figure 7. Validation Accuracy (and standard error) for different regularization methods for a network with Width=10.

w.r.t. the number of inputs to a neuron: each additional input would naturally require a higher number of supporting points (loosely following Occam's Razor) yet the additional regularization depends both on the number of inputs and the data itself.

Author Contributions: Conceptualization, G.N.; implementation, A.C. and P.R.; formal analysis, G.N., A.C. and P.R.; writing —original draft preparation, A.C. and P.R.; writing—review and editing, G.N. and P.R.; visualization, A.C. and P.R.; supervision, G.N.; project administration, G.N.. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Only publicly available data has been used for this study. The Fashion-MNIST data [26] is freely available.

Acknowledgments: In this section you can acknowledge any support given which is not covered by the author contribution or funding sections. This may include administrative and technical support, or donations in kind (e.g., materials used for experiments).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

EBR	Evidence-Based Regularization
MNIST	Modified National Institute of Standards and Technology
ReLU	Rectified Linear Unit

Appendix A *TensorFlow* Implementation

We chose Tensorflow [31] for implementation due to its simple interface via Keras [32] and its autodifferentiation abilities. We implemented the *Evidence Regularizer* as a layer that adds the minimal activation for each node to the loss of the layer while passing its input through. We add metrics for the activation on the positive and negative side of the layer as well as a count of the number of nodes with activation below the chosen threshold. These metrics allow us to track the impact of the *Evidence Regularizer* on the layer and the neural network. By using only Tensorflow operations and the *add_loss* method, Tensorflow's autodiff mechanism take care of the logic of adjusting the gradient size in the backward step during optimization. The complete implementation via Keras is:

```
import tensorflow as tf

class EvidenceRegularizerLayer(tf.keras.layers.Layer):
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1


```
    def __init__(self, threshold: float, cutoff: float):
        super().__init__()
        self.threshold = threshold
        self.cutoff = cutoff

    def get_config(self):
        return dict(threshold=self.threshold, cutoff=self.cutoff)

    def call(self, inputs):
        positive_activation = tf.reduce_sum(tf.maximum(inputs, self.cutoff), axis=0)
        negative_activation = tf.reduce_sum(tf.maximum(-inputs, self.cutoff), axis=0)

        node_activation = tf.minimum(positive_activation, negative_activation)
        regularization = tf.minimum(node_activation - self.threshold, 0.0)

        batch_size = tf.cast(tf.shape(inputs)[0], dtype=tf.float32)
        self.add_loss(-tf.reduce_sum(regularization) / batch_size)

        self.add_metric(
            tf.reduce_min(positive_activation),
            name=f"minimal_positive_activation_{
             self._name.split('_')[-1]}",aggregation="mean",
        )
        self.add_metric(
            tf.reduce_min(negative_activation),
            name=f"minimal_negative_activation_{
                self._name.split('_')[-1]}",aggregation="mean",
        )
        self.add_metric(
            tf.math.count_nonzero(
                  tf.math.less(node_activation, self.threshold),
                  dtype=tf.float32
            ),
            name=f"number_of_nodes_below_threshold_{
               self._name.split('_')[-1]}", aggregation="mean",
        )

        return inputs
```

Alternatively, the regularizer can also be implemented as a *Regularizer* class as follows — noting that the below implementation does not require auto-differentiation as the penalty is coded using the *softmax* function:

```
class EvidenceRegularizer(tf.keras.regularizers.Regularizer):

    def __init__(self, threshold: float, cutoff: float):
        self.threshold = threshold
        self.cutoff = cutoff

    def get_config(self):
        return dict(threshold=self.threshold, cutoff=self.cutoff)

    def __call__(self, x):
        positive_activation = tf.reduce_sum(tf.maximum(x, self.cutoff), axis=0)
        negative_activation = tf.reduce_sum(tf.maximum(-x, self.cutoff), axis=0)
        node_activation = tf.minimum(positive_activation, negative_activation)
        regularization = tf.minimum(node_activation - self.threshold, 0.0)
        return -tf.reduce_sum(regularization)
```

For *tanh* activation, the cutoff parameter has to be set to 0.0. For sigmoid activation, the cutoff parameter is 0.5.

Seed numbers used to reproduce the results in Section 3.3:

```
9218, 2059, 6670, 3606, 2077, 7205, 7736, 1593, 9788, 2621, 5193, 4173,
6735, 3156, 7767, 1635, 8803, 9828, 619, 2155, 7277, 5234, 5240, 6778,
651, 4750, 8847, 5263, 4248, 8345, 6813, 7844, 9382, 3260, 5822, 4306,
5845, 4311, 4825, 9433, 7392, 7403, 8433, 8951, 5880, 3321, 9983, 3845,
1294, 2832, 7448, 8984, 5406, 8991, 801, 4402, 5431, 3387, 1345, 9544,
5450, 4938, 1868, 3409, 4948, 5467,  349, 9060, 2917, 2923, 8562, 6520,
1408, 4481, 9093,  392, 7048, 1419, 8076, 3982, 8594, 5524, 2969,  410,
8602, 2971,  945, 8116, 8628, 9659, 6078, 1472,  961, 9160, 5068, 7632,
2007, 6634, 2546, 1019
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

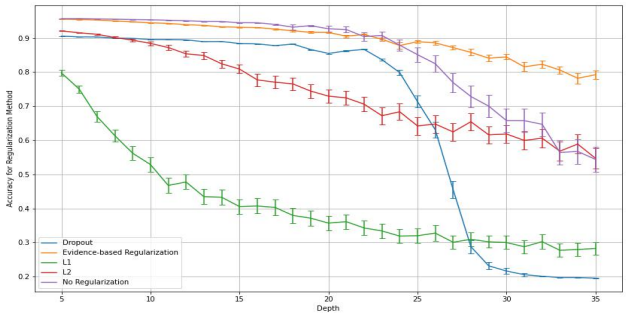

Figure A1. Training Accuracy (and standard error) for different regularization methods for a network with Width=10.

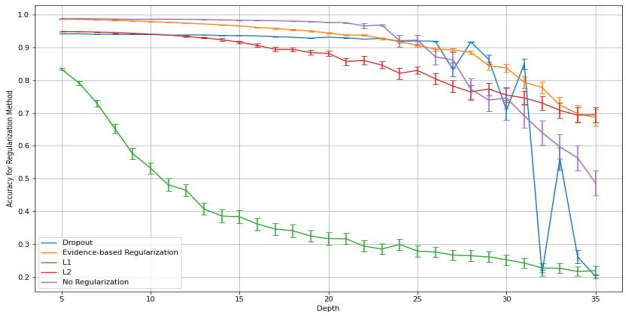

Figure A2. Training Accuracy (and standard error) for different regularization methods for a network with Width=20.

Appendix B Digit MNIST experiments for two narrow neural networks (widths 10 and 20) up to depth 35

Appendix C Detailed Specifications

Each node in a neural network layer is defined by a hyperplane in $\mathbb{R}^d$ with weights $\mathbf{w}$ and an activation function s from $\mathbb{R}$ to $\mathbb{R}$:

$$f(\mathbf{x}) = s(\langle \mathbf{w}, \mathbf{x} \rangle_{w_0}) \tag{A1}$$

where $\langle \mathbf{w}, \mathbf{x} \rangle_{w_0} = w_0 + \sum_{i=1}^{d} w_i x_i$.

We say that a neuron has been activated for some inputs if its output is far from 0. We define the activation of a neuron for some data $\mathcal{X} = \left\{ \mathbf{x}^{(j)} \right\}_{j=1}^{n}$ as

$$\mathcal{A}_{\mathcal{X}}(f) = \min\left(\sum_{j=1}^{n} \max\left(f\left(\mathbf{x}^{(j)}\right), 0 \right), \sum_{j=1}^{n} \max\left(-f\left(\mathbf{x}^{(j)}\right), 0 \right) \right) \tag{A2}$$

The evidence-based regularized output of the neuron is then

$$f_r(\mathbf{x}) = f(\mathbf{x}) + \min(\mathcal{A}_{\mathcal{X}}(f) - \lambda, 0) \tag{A3}$$

where λ is a threshold chosen according to the noise in the data. The gradient of the regularized node is therefore

$$\frac{\partial f_r}{\partial w} = \frac{\partial f}{\partial w} + \frac{\partial \mathcal{A}_{\mathcal{X}}(f)}{\partial w} \mathbb{1}_{\lambda < \mathcal{A}_{\mathcal{X}}(f)}. \tag{A4}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 15 September 2022 doi:10.20944/preprints202209.0231.v1

The formulation ensures that the gradient of the node is increased when the activation on one side of the hyperplane is below the threshold λ. On the other hand, if the activation is above the threshold, no regularization occurs. Therefore, the usual knowledge gain in each back-propagation update step includes a nudge into a direction that involves a meaningful split by the hyperplane of the input data. For the value of λ, we found that good values are 10% of the batch size for regression problems. For classification problems, the threshold should be 10% of the batch size multiplied by the fraction of the smallest class. For example, if the smallest class is 20% of the data and a batch size of 1'000 is used, then the threshold λ should be $10\% \times 20\% \times 1'000 = 20$.

For sigmoid activation, (C) has to be changed to

$$\mathcal{A}_{\mathcal{X}}(f) = \min\left(\sum_{j=1}^{n} \max\left(f\left(\mathbf{x}^{(j)}\right), 0.5\right), \sum_{j=1}^{n} \max\left(-f\left(\mathbf{x}^{(j)}\right), 0.5\right)\right) \tag{A5}$$

i.e. the cutoff between positive and negative activation is around the 0.5 mid-point, rather than 0 for tanh activation.

For ReLU-type activations, the formulation of activation has to be adapted: Such activation functions are unbounded on one side and do not have a symmetry point around which to define positive and negative activation. We propose to use a softcount of the number of nodes with positive and negative values prior to activation using the tanh function for counting. (C) then becomes:

$$\mathcal{A}_{\mathcal{X}}(f) = \min\left(\sum_{j=1}^{n} \max\left(\tanh\left(\mathbf{x}^{(j)}\right), 0\right), \sum_{j=1}^{n} \max\left(-\tanh\left(\mathbf{x}^{(j)}\right), 0\right)\right). \tag{A6}$$

The activation is therefore calculated as if a tanh activation function had been used.

References

1. Goodfellow, I.J.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.
2. Kukacka, J.; Golkov, V.; Cremers, D. Regularization for Deep Learning: A Taxonomy, 2017. https://doi.org/10.48550/ARXIV.1710.10686.
3. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International conference on machine learning. PMLR, 2015, pp. 448–456.
4. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer normalization. *arXiv preprint arXiv:1607.06450* **2016**.
5. Bishop, C.M.; et al. *Neural networks for pattern recognition*; Oxford university press, 1995.
6. DeVries, T.; Taylor, G.W. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538* **2017**.
7. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.
8. Hinton, G.E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR* **2012**, *abs/1207.0580*, [1207.0580].
9. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* **2014**, *15*, 1929–1958.
10. Fukushima, K.; Miyake, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*; Springer, 1982; pp. 267–285.
11. Rumelhard, D.E.; McClelland, J.L.; PDP Research Group. Parallel distributed processing: Explorations in the microstructures of cognition. Volume 1: Foundations. In *Parallel distributed processing. Vol. 1.*; IEEE, MIT Press, 1988.
12. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.
13. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.
14. Plaut, D.C.; et al. Experiments on Learning by Back Propagation., 1986.
15. Lang, K.; Hinton, G.E. Dimensionality reduction and prior knowledge in e-set recognition. *Advances in neural information processing systems* **1989**, *2*.
16. Lin, T.Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal loss for dense object detection. In Proceedings of the Proceedings of the IEEE international conference on computer vision, 2017, pp. 2980–2988.

17. Caruana, R. A dozen tricks with multitask learning. In *Neural networks: tricks of the trade*; Springer, 1998; pp. 165–191.
18. Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* **2017**.
19. Hardt, M.; Recht, B.; Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. In Proceedings of the International conference on machine learning. PMLR, 2016, pp. 1225–1234.
20. Prechelt, L. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks* **1998**, *11*, 761–767.
21. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
22. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.
23. Smith, L.N. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018. https://doi.org/10.48550/ARXIV.1803.09820.
24. Wilson, A.C.; Roelofs, R.; Stern, M.; Srebro, N.; Recht, B. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems* **2017**, *30*.
25. Geifman, Y.; El-Yaniv, R. Selective Classification for Deep Neural Networks. In Proceedings of the Proceedings of the 31st International Conference on Neural Information Processing Systems; Curran Associates Inc.: Red Hook, NY, USA, 2017; NIPS'17, Long Beach, California, USA, p. 4885–4894.
26. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017. https://doi.org/10.48550/ARXIV.1708.07747.
27. Menon, A.K.; van Rooyen, B.; Natarajan, N. Learning from Binary Labels with Instance-Dependent Corruption, 2016. https://doi.org/10.48550/arXiv.1605.00751.
28. Patrini, G.; Rozza, A.; Menon, A.; Nock, R.; Qu, L. Making Deep Neural Networks Robust to Label Noise: a Loss Correction Approach, 2016. https://doi.org/10.48550/arXiv.1609.03683.
29. Smith, L.N. Cyclical Learning Rates for Training Neural Networks, 2015. https://doi.org/10.48550/ARXIV.1506.01186.
30. Lu, Z.; Pu, H.; Wang, F.; Hu, Z.; Wang, L. The Expressive Power of Neural Networks: A View from the Width. In Proceedings of the Proceedings of the 31st International Conference on Neural Information Processing Systems; Curran Associates Inc.: Red Hook, NY, USA, 2017; NIPS'17, Long Beach, California, USA, p. 6232–6240.
31. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.
32. Chollet, F.; et al. Keras. https://keras.io, 2015.