Probabilistic RPROP

Andrii Zadaianchuk & Lukas Balles
Max Planck Institute for Intelligent Systems, Tübingen, Germany
`[azadaianchuk|lballes]@tue.mpg.de`

Abstract

Training with mini-batches is necessary for large-scale tasks where it is not possible to calculate the loss on the whole data set. Existing stochastic optimization methods for such tasks need a lot of efforts to tune their parameters such as learning rate. An adaptive optimization algorithm RPROP is fast and easy to tune. However, it fails in a mini-batch setting. We studied the reasons of RPROP failure and estimated the probabilities of wrong adaptations and updates. We proposed a modification of RPROP that uses these probabilities to minimize average changes caused by the noise in the stochastic gradients. We showed that it is stable to noise and possible to use with small mini-batches. The performance of our algorithm is better than RPROP with large mini-batch, but it is still lower than current state-of-the-art optimizers such as well-tuned SGD with momentum.

1 Introduction

Deep neural networks (DNNs) recently reach the state-of-the-art performance for many tasks connected with high dimensional data, such as image or speech recognition (Deng et al., 2013, Krizhevsky et al., 2012). The optimization problem for these tasks can be formulated as

$$\min_{\theta} \mathcal{L}(\theta), \qquad \mathcal{L}(\theta) = \frac{1}{M}\sum_{i=1}^{M} l(\theta; x_i), \tag{1}$$

where M is the size of a data set and $l(\theta; x_i)$ is a loss function computed for one data point. A dataset for such tasks is too large to compute the full loss and gradients, thus one needs to use mini-batches for training the model. We can interpret the estimation of the full loss and gradient values with mini-batches as the full loss and gradient with additive noise. From this view, the mini-batch loss and gradient are random variables, and the loss minimization is a stochastic optimization task. In this setting, we do not have access to the loss gradient $\nabla\mathcal{L}(\theta)$ that is equal to

$$\nabla\mathcal{L}(\theta) = \frac{1}{M}\sum_{i=1}^{M} \nabla l(\theta; x_i). \tag{2}$$

Instead, we can compute its unbiased estimator - a stochastic gradient

$$g(\theta) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \nabla l(\theta; x_i). \tag{3}$$

where $\mathcal{B}$ is mini-batch indexes.

There are several stochastic optimization algorithms that use stochastic loss and gradient and converge to the solutions with good generalization ability. Momentum modifications of SGD (Nesterov, 1983, Polyak, 1964) accelerate the convergence of SGD and help to go across flat regions of a loss function. They ware used to get state-of-the-art results with Deep Residual Networks (He et al., 2015). However, tuning of these methods is challenging. For example, one often needs to adjust step size during training and find the best schedule function, not just constant learning rate (Bergstra and Bengio, 2012, Loshchilov and Hutter, 2016).

A common approach to simplify tuning of the learning rate is to adapt the step size of the update for every parameter. There are several different ways to adapt the step size (Kingma and Ba, 2014, Riedmiller and Braun, 1993, Tan et al., 2016). One can use the estimation of the stochastic gradient variance, shrinking the step sizes in the directions with the huge variance. ADAM (Kingma and Ba, 2014) and AdaDelta (Zeiler, 2012) are notable examples of such methods.

RPROP (Riedmiller and Braun, 1993) is another adaptive optimization algorithm. It changes the size of the optimization step using the information about the product of current and previous gradients signs. This algorithm and its modifications (Igel and Hüsken, 2000) were used a lot during the 1990s because they are fast and easy to tune. However, they all rely on the exact value of gradients and fail to optimize the stochastic objective functions (Tieleman and Hinton., 2012).

There were several current attempts to modify RPROP optimizer for training of DNNs (Mosca and Magoulas, 2015, 2017). The authors introduce new optimizer WAME, which uses moving average of RPROP adaptations to adapt the step size. They showed that in some setting WAME can get lower training loss that ADAM and RMSPROP. However, in their work, there is no information about the stability of their optimizer to noise in the stochastic gradient and about the size of the batches that they used during training. In addition, the authors introduce additional global learning rate, without mentioning the way how they tune it. Because of this, it is hard to conclude something about the stability and the generalization ability of this method.

The main goals of this project were to study the reasons of RPROP failure in the stochastic case and to develop a probabilistic version of RPROP algorithm that combines the advantages of RPROP and is possible to use with mini-batches.

The project consists of several parts. First, we studied the RPROP algorithm performance while optimizing CNN model on highly used in the image recognition datasets: CIFAR-10 and MNIST. We estimated the variance of the stochastic gradient similar to (Kingma and Ba, 2014) and used the normal assumption to calculate the probabilities of the wrong update. Secondly, we modified RPROP algorithm to make it stable to noise. For this, we used calculated probabilities of the mistake to scale the adaptation factor and sign of the stochastic gradient. This scaling helps to minimize the average changes in the behaviour of RPROP algorithm due to noise. Finally, we compared the performance of our modification with the original RPROP as well as with the fine-tuned SGD with momentum.

2 Resilient backPROPagation (RPROP)

2.1 Description of RPROP

The basic idea of RPROP is to increase the step size if we move in the right direction and to decrease it if we go trough a local minimum. To control that we are going in the right direction we can check that the sign of the partial derivative is the same $\frac{\partial \mathcal{L}^{(t-1)}}{\partial \theta_{ij}^{(t-1)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial \theta_{ij}^{(t)}} > 0$. If we go trough the local minimum the sign of the partial derivative will change, so the product of partial derivatives will be negative $\frac{\partial \mathcal{L}^{(t-1)}}{\partial \theta_{ij}^{(t-1)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial \theta_{ij}^{(t)}} < 0$. The exact form of the RPROP update rule is

$$\begin{aligned} \theta_{ij}^{(t)} &= \theta_{ij}^{(t)} + \Delta\theta_{ij}^{(t)} \\ \Delta\theta_{ij}^{(t)} &= -\operatorname{sign}\left(\frac{\partial \mathcal{L}^{(t)}}{\partial \theta_{ij}^{(t)}}\right) \cdot \Delta_{ij}^{(t)}, \end{aligned} \tag{4}$$

where $\Delta_{ij}^{(t)}$ is adapted step size. It is changed according to the sign of the product of partial derivatives.

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+} \cdot \Delta_{ij}^{(t-1)}, & \frac{\partial \mathcal{L}^{(t-1)}}{\partial \theta_{ij}^{(t-1)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial \theta_{ij}^{(t)}} > 0 \\ \eta^{-} \cdot \Delta_{ij}^{(t-1)}, & \frac{\partial \mathcal{L}^{(t-1)}}{\partial \theta_{ij}^{(t-1)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial \theta_{ij}^{(t)}} < 0 \\ \Delta_{ij}^{(t-1)}, & \text{otherwise,} \end{cases} \tag{5}$$

where $\eta^{-} < 1, \eta^{+} > 1$.

There are several variants of RPROP algorithm (Igel and Hüsken, 2000). The main difference in RPROP modifications is presence or absence of a backward step. The backward step is done when we go through the local minimum and want to return back to the previous parameters values. It is performed by subtracting previous update $\Delta w_{ij}^{(t-1)}$. In our experiments we use the version without backward step because it saves memory usage and it is more stable comparable to RPROP with backward step (Igel and Hüsken, 2000). Look at the Algorithm 1 to see the exact description of the RPROP version that we used in our experiments.

Algorithm 1 RPROP

```
Require: initial value $\theta_0$, $\Delta_0$, $\Delta_{min}$, $\Delta_{max}$, $\eta^-$,$\eta^+$ number of steps $T$
 1: Initialize $g_{old} = 0$, $\Delta = \Delta_0$, $\theta = \theta_0$
 2: for $t = 1, \dots, T$ do
 3:     for all parameters do
 4:         Evaluate stochastic gradient $g$
 5:         Take signs $s = \mathrm{sign}(g)$ and $s_{old} = \mathrm{sign}(g_{old})$
 6:         Take product of signs $prod = s_{old} \cdot s$
 7:         if $prod < 0$ then
 8:             Update $\Delta = \max(\eta^- \cdot \Delta, \Delta_{min})$
 9:             Set $g = 0$
10:         else if $prod > 0$ then
11:             Update $\Delta = \min(\eta^+ \cdot \Delta, \Delta_{max})$
12:         end if
13:         Update $\theta = \theta - \Delta s$
14:         $g_{old} = g$
15:     end for
16: end for
```

2.2 RPROP fails to optimize stochastic approximation of the loss function

There are several optimization algorithms where the usage of small mini-batches gives an improvement compared with the usage of large batches (such as the whole dataset) (Ge et al., 2015, Zhang et al., 2017). However, an attempt to use RPROP to train CNN with mini-batches shows that this algorithm is highly sensitive to the amount of noise in loss function evaluation. The covariance of the stochastic gradient $\mathrm{cov}[g(\theta)] = \frac{\mathrm{cov}[\nabla l_i(\theta)]}{|\mathcal{B}|}$ is inversely proportional to the batch size $|\mathcal{B}|$. Therefore, we can study the dependence of the optimizer on noise in the stochastic gradient by training our model with different batch sizes. Look at Fig. 3 to see the dependence of accuracy and loss on the batch size $|\mathcal{B}|$.

2.3 Why does it fail?

There are several reasons why the behaviour of the RPROP algorithm is much worse for high-variance stochastic gradients. The first problem is connected with the adaptation of step sizes. RPROP needs the sign of the product of the full gradients, whereas in stochastic case we have access only to the sign of the product of the stochastic gradients that is also a random variable. So the probability

$$P\left(\mathrm{sign}(g(\theta^{(t)}))\,\mathrm{sign}(g(\theta^{(t-1)})) \neq \mathrm{sign}(\nabla\mathcal{L}(\theta^{(t)}))\,\mathrm{sign}(\nabla\mathcal{L}(\theta^{(t-1)}))\right) \tag{6}$$

determines how often we will change our step size by mistake. This probability is closer to $\frac{1}{2}$ when the variance is larger. Thus, in the noisy case, we will more often change the size of optimization step Δ by mistake. Accumulation of such mistakes leads to step sizes that are far from optimal values making the optimizer inefficient or unstable.

Another problem is that sometimes we make the step in a wrong direction. As our direction is determined by the sign of the stochastic gradient there is probability

$$P\left(\mathrm{sign}(g(\theta^{(t)})) \neq \mathrm{sign}(\nabla\mathcal{L}(\theta^{(t)}))\right) \tag{7}$$

of making step in the wrong direction. See the (Balles and Hennig, 2017) for more details. Estimation of this probabilities can help us to distinguish between reliable updates and random noisy updates. In next section, we show how to estimate these two probabilities and use them to minimize the damage caused by noise in the stochastic gradient.

3 Probabilistic RPROP

In this section we describe how can we estimate the variance of the stochastic gradient, and use the variance estimate to scale the adaptation factor. We discuss the choice of different adaptation functions and find such function that improve stability and convergence of ProbRPROP algorithm.

3.1 Variance estimation

There are several different ways to estimate the variance of the stochastic gradient. First method uses moving averages of the first and second moment of the stochastic gradient to estimate the variance. This method of variance estimation was used in (Kingma and Ba, 2014). In our experiments we used the same moving average constant $\beta_1 = \beta_2 = 0.99$. This constant defines the effective time horizon over which the gradients assumed to come from the same distribution, so there is no reasons to set different constants for first and second moment of the stochastic gradient. It is important to include bias correction to deal with bias after setting initial values of moving averages to zero.

Another method of variance estimation is "mini-batch" variance estimation that uses single mini-batch to estimate the variance of the stochastic gradient. For details look at (Balles and Hennig, 2017).

For this project we tried both variance estimators. For final experiments, we used moving averages variance estimation as it is easy to implement and the results for this estimator was not worse than the results for mini-batch variance estimator.

3.2 Probability of sign change

After we estimated the variance of the stochastic gradient, we can use it to find the probability ρ of sign change. The stochastic gradient $g(\theta^{(t)})$ is approximately normally distributed

$$g(\theta^{(t)}) \sim \mathcal{N}(\Delta\mathcal{L}(\theta^{(t)}), \operatorname{cov}[g(\theta^{(t)})])$$

as the average of independent identically distributed $\nabla l_i(\theta^{(t)})$. Assuming non-informative prior over $\nabla\mathcal{L}^{(t)}$ we can use Bayes theorem to find distribution over $\nabla\mathcal{L}^{(t)}$ given $g(\theta^{(t)})$

$$g^{(t)} \sim \mathcal{N}(\nabla\mathcal{L}^{(t)}, \operatorname{cov}[g^{(t)}]) \Rightarrow \nabla\mathcal{L}^{(t)} \sim \mathcal{N}(g^{(t)}, \operatorname{cov}[g^{(t)}])$$

Then $s_{ij}^{(t)} = \operatorname{sign} \nabla\mathcal{L}_{ij}^{(t)} \nabla\mathcal{L}_{ij}^{(t-1)}$ is Bernoulli random variable.

$$s_{ij}^{(t)} = \begin{cases} -1, & \text{with probability } \rho_{ij}^{(t)} \\ 1, & \text{with probability } 1 - \rho_{ij}^{(t)} \end{cases} \tag{8}$$

Look at Formula 9 in the Lemma 1 for the formula of sign change probability ρ. For typical values of sign change probability you can look at Fig. 1, there you see the histogram of the probabilities of the sign change evolving through iterations of training.

Lemma 1. *If $X \sim \mathcal{N}(\mu_1, \sigma_1^2)$, $Y \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then for independent X and Y*

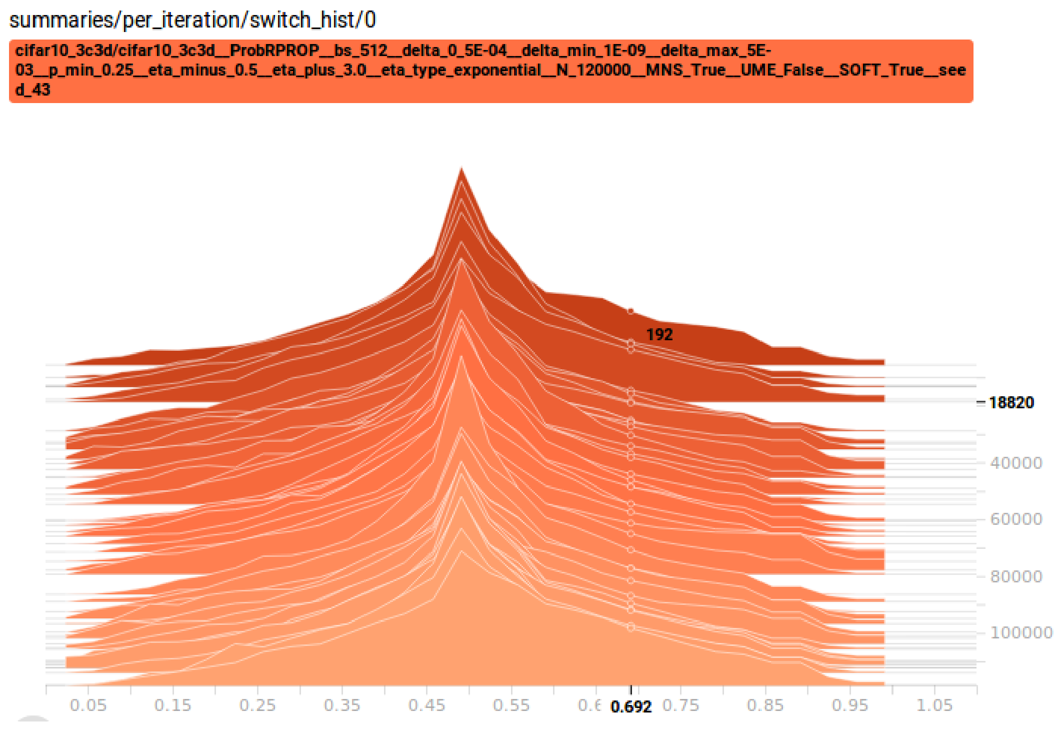
$$\begin{aligned} \rho &= \mathbf{P}[\operatorname{sign}(X) \neq \operatorname{sign}(Y)] \\ &= \frac{1}{4}\left(1 + \operatorname{erf}\left(\frac{-\mu_1}{\sqrt{2}\sigma_1}\right)\right)\left(1 + \operatorname{erf}\left(\frac{\mu_2}{\sqrt{2}\sigma_2}\right)\right) + \frac{1}{4}\left(1 + \operatorname{erf}\left(\frac{\mu_1}{\sqrt{2}\sigma_1}\right)\right)\left(1 + \operatorname{erf}\left(\frac{-\mu_2}{\sqrt{2}\sigma_2}\right)\right). \end{aligned} \tag{9}$$

Proof.

$$\rho = \mathbf{P}[\operatorname{sign}(X) \neq \operatorname{sign}(Y)] = \mathbf{P}[X < 0]\mathbf{P}[Y > 0] + \mathbf{P}[X > 0]\mathbf{P}[Y < 0] \tag{10}$$

Lets find the probability that formal variable $Z \sim \mathcal{N}(\mu, \sigma^2)$ in bigger and lowwer than 0. Let Φ be the CDF of the normal distribution. Then,

$$\mathbf{P}[Z < 0] = \Phi\left(\frac{0 - \mu}{\sigma}\right) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\mu}{\sqrt{2}\sigma}\right)\right). \tag{11}$$


summaries/per_iteration/switch_hist/0

cifar10_3c3d/cifar10_3c3d__ProbRPROP__bs_512__delta_0_5E-04__delta_min_1E-09__delta_max_5E-03__p_min_0.25__eta_minus_0.5__eta_plus_3.0__eta_type_exponential__N_120000__MNS_True__UME_False__SOFT_True__seed_43

Figure 1: Histograms of sign change probabilities. For all parameters of our model, at every iteration we have some probability of sign change. Every histogram represents the distribution of this probabilities at concrete iteration. Deeper and darker histograms correspond to the smaller iterations

$$\begin{aligned}\mathbf{P}[Z > 0] &= 1 - \mathbf{P}[Z \leq 0] = 1 - \Phi\left(\frac{0-\mu}{\sigma}\right) \\ &= 1 - \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\mu}{\sqrt{2}\sigma}\right)\right) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right)\right),\end{aligned} \tag{12}$$

where the last step used the anti-symmetry of the error function. Applying Eq. 11 and 12 to Eq. 10 we proof the lemma. □

3.3 Adaptation function

The main idea of ProbRPROP is the usage of the probability of sign change to make the behaviour of RPROP more stable to the noise. We can make this by modifying adaptation rule in Eq. 8. New adaptation rule can be described as

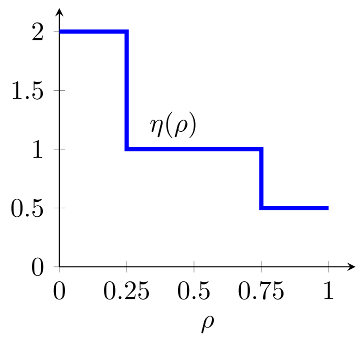
$$\Delta_{ij}^{(t)} = \eta(\rho) \cdot \Delta_{ij}^{(t-1)}, \tag{13}$$

where $\eta(\rho)$ is dependence of the adaptation factor on probability of sign change ρ.

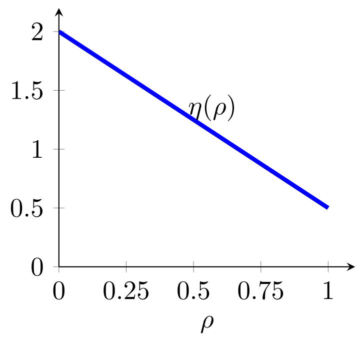
When probability ρ is near zero we are sure that there was no sign change, so we want our algorithm to behave as deterministic RPROP in case $\frac{\partial \mathcal{L}^{(t-1)}}{\partial \theta_{ij}^{(t-1)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial \theta_{ij}^{(t)}} > 0$. When probability ρ is near one the algorithm behaves as deterministic RPROP in case $\frac{\partial \mathcal{L}^{(t-1)}}{\partial \theta_{ij}^{(t-1)}} \cdot \frac{\partial \mathcal{L}^{(t)}}{\partial \theta_{ij}^{(t)}} < 0$. For other probabilities, we can scale the size of the adaptation factor η counting that we do not want to change Δ heavily if we are not sure about sign change.

We propose several different types of the adaptation function $\eta(\rho)$. For example, we can use linear function that goes throuth η^+ in 0 and η^- in 1.

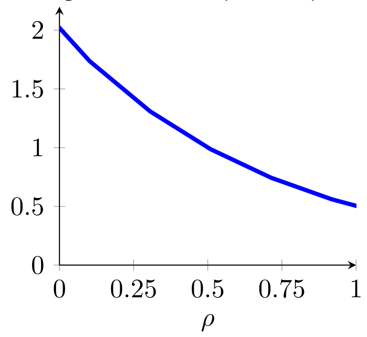
$$\eta(\rho, \eta^+, \eta^-) = \rho\eta^- + (1-\rho)\eta^+. \tag{14}$$



(a) Step function with $\eta^+ = 2, \eta^- = \frac{1}{2}, p_{min} = 0.25$.



(b) Linear function with $\eta^+ = 2, \eta^- = \frac{1}{2}$.



(c) Exponential function with $\eta^+ = 2$.

Figure 2: Dependence of the adaptation factor η on probability of sign change ρ. a) shows step adaptation that does not charge step size if the probability is near 0.5, so when we are not sure about presents or absence of sign change. In b) the adaptation factor is different for every probability. It is the linear approximation between η^+ and η^-. c) represents exponential adaptation function that satisfies stability criteria.

To balance average random increase and decrease of step size we want our adaptation factors to be inverse e.g. $\eta^+ = \frac{1}{\eta^-}$ for probabilities symmetric around 0.5. For such probabilities we want to have inverse adaptation factors $\eta(0.5 + p) = \frac{1}{\eta(0.5-p)}$. The adaptation function that fulfils this property is

$$\eta(\rho, \eta^+) = e^{-2 \ln \eta^+ \cdot \left(\rho - \frac{1}{2}\right)}. \tag{15}$$

Such adaptation function helps to make ProbRPROP insensitive to different batch sizes. In Figure 2 you can see all cases of the adaptation function $\eta(\rho)$, that we used in our experiments.

For exact version of ProbRPROP algorithm look at Algorithm 2. We do not save values of gradients g and their variance r^2 because we do need to save the only their combination - a relative variance. So we save only

$$p^+ = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{g}{\sqrt{2}r}\right)\right)$$

that is the function of the relative variance. This decreases memory usage of the optimizer.

3.4 Soft sign

In addition to the usage of the probability of the sign change, we can use the variance of the gradient to adapt the size of the current update step of the parameters. If the variance is too large we cannot rely on the calculated direction, so we want to make a small step in such direction. As described in (Balles and Hennig, 2017) the optimal step size $d = 2p^+ - 1$. Look at Algorithm 3 for the pseudo code of PropRPROP algorithm with the soft sign. It differs from Algorithm 2 in one line only.

Algorithm 2 ProbRPROP

```
Require: $\theta_0$, $\Delta_0$, $\Delta_{min}$, $\Delta_{max}$, $\eta^-$, $\eta^+$, $p_{min}$ number of steps $T$, $\varepsilon$
 1: Initialize $p^+_{old} = 0.5$, $\Delta_{old} = \Delta_0$,
 2: for $t = 1, \dots, T$ do
 3:     Compute stochastic gradient and its variance $g, r^2$
 4:     Compute $p^+ = \mathbf{P}[Z > 0] = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{g}{\sqrt{2}r}\right)\right)$
 5:     Evaluate probability of sign change $\rho = (1 - p^+)p^+_{old} + (1 - p^+_{old})p^+$
 6:     Compute step size adaptation factor (e.g. Eq. 15) $\eta = \eta(\rho, \eta^+, \eta^-, p_{min})$
 7:     Update $\Delta = \Delta \cdot \eta$
 8:     $\theta = \theta - \Delta \cdot \operatorname{sign}(g)$
 9:     $\Delta_{old} = \Delta$
10:     $p^+_{old} = p^+$
11: end for
```

Algorithm 3 ProbRPROP with soft sign

```
Require: $\theta_0$, $\Delta_0$, $\Delta_{min}$, $\Delta_{max}$, $\eta^-$, $\eta^+$, $p_{min}$ number of steps $T$, $\varepsilon$
 1: Initialize $p^+_{old} = 0.5$, $\Delta_{old} = \Delta_0$,
 2: for $t = 1, \dots, T$ do
 3:     Compute stochastic gradient and its variance $g, r^2$
 4:     Compute $p^+ = \mathbf{P}[Z > 0] = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{g}{\sqrt{2}r}\right)\right)$
 5:     Take soft sign $d = 2p^+ - 1$
 6:     Evaluate probability of sign change $\rho = (1 - p^+)p^+_{old} + (1 - p^+_{old})p^+$
 7:     Compute step size adaptation factor (e.g. Eq. 15) $\eta = \eta(\rho, \eta^+, \eta^-, p_{min})$
 8:     Update $\Delta = \Delta \cdot \eta$
 9:     $\theta = \theta - \Delta \cdot d$
10:     $\Delta_{old} = \Delta$
11:     $p^+_{old} = p^+$
12: end for
```

4 Experiments

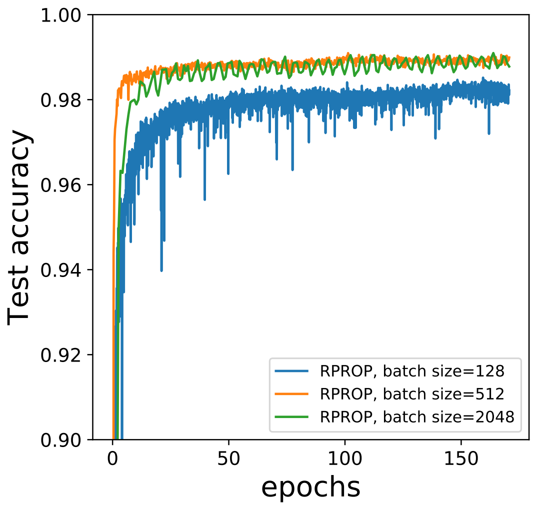
We have done several computational experiments to compare properties of the original RPROP (Algorithm 1) and ProbRPROP with the soft sign (Algorithm 3). We use different batch sizes (from 32 to 2048) to show the dependence of the performance of studied optimizers to the amount of noise in stochastic gradient estimate. In addition, we compared the results of our algorithm to well-tuned SGD with momentum.

We chose the soft sign version of ProbRPROP (Algorithm 3) because it simultaneously decreases the effect of the adaptation mistakes (Eq. 6) and the wrong signs updates (Eq. 7). Also, this version showed better experimental results during early comparison of these methods.

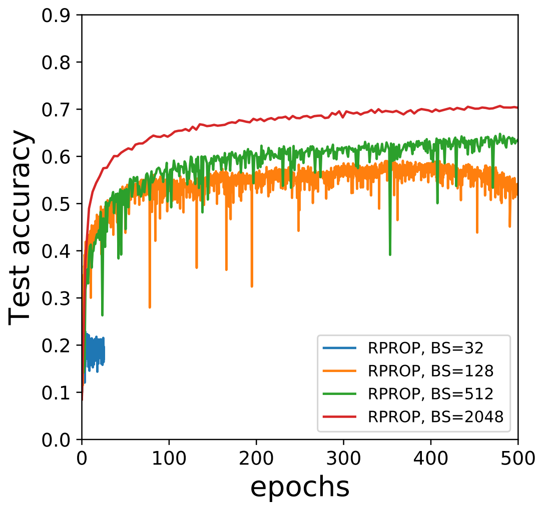
The same $\Delta_{min} = 10^{-9}$ and $\Delta_{max} = 0.5$ were used for Algorithm 1 and 3. As we do not have any global learning rate, we want to be sure that the interval $[\Delta_{min}, \Delta_{max}]$ is wide enough. This guarantees that the performance of our optimizer determined by the right adaptive procedure, not just saturation to the good Δ_{min} or Δ_{max} values.

In all experiments, where it was possible to choose both η^+ and η^- we chose $\eta^+ = \frac{1}{\eta^-}$ as it helps to balance the random increase and decrease of the step size. Additional experiments showed that this indeed helps to get the best test accuracy in comparison with different η^+ for fixed η^- (Part B of the Supplementary materials). We checked the properties for three adaptation functions presented in Figure 2. We tuned η^+ using grid search in interval $[1.1, 2]$.

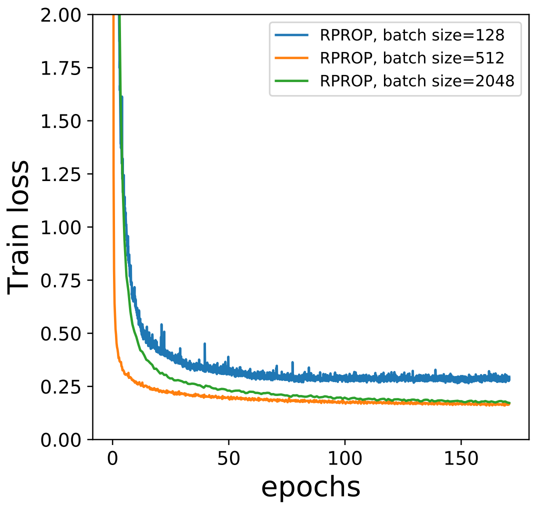
You can find our TensorFlow implementation for RPROP and ProbRPROP optimizers in GitHub: https://github.com/zadaianchuk/ProbabilisticRPROP.



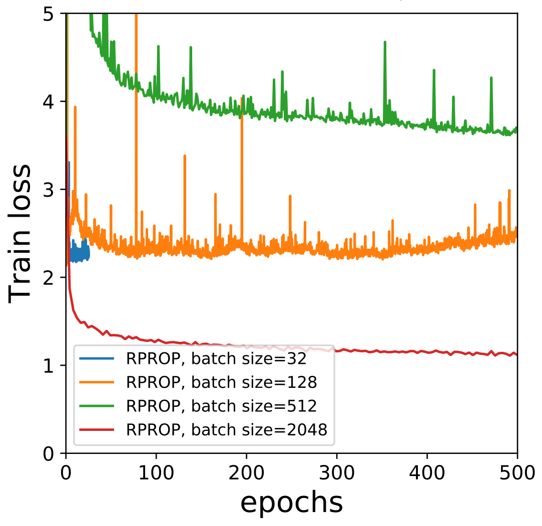
(a) MNIST. Test accuracy



(b) CIFAR-10. Test accuracy



(c) MNIST. Training loss



(d) CIFAR-10. Training loss

Figure 3: RPROP algorithm is not stable to the noise in the stochastic gradient. It works only for large mini-batches. In a) and c) you see the test accuracy and training loss for MNIST data set. For small batch sizes, you can see that RPROP algorithm converges to smaller test accuracies. Whereas for large batch sizes there is convergence to similar values as SGD with momentum. In b) and d) you see the test accuracy and training loss for the CIFAR-10 data set. The accuracy is significantly lower than momentum accuracy even for large batch sizes.

4.1 Experimental Set-Up

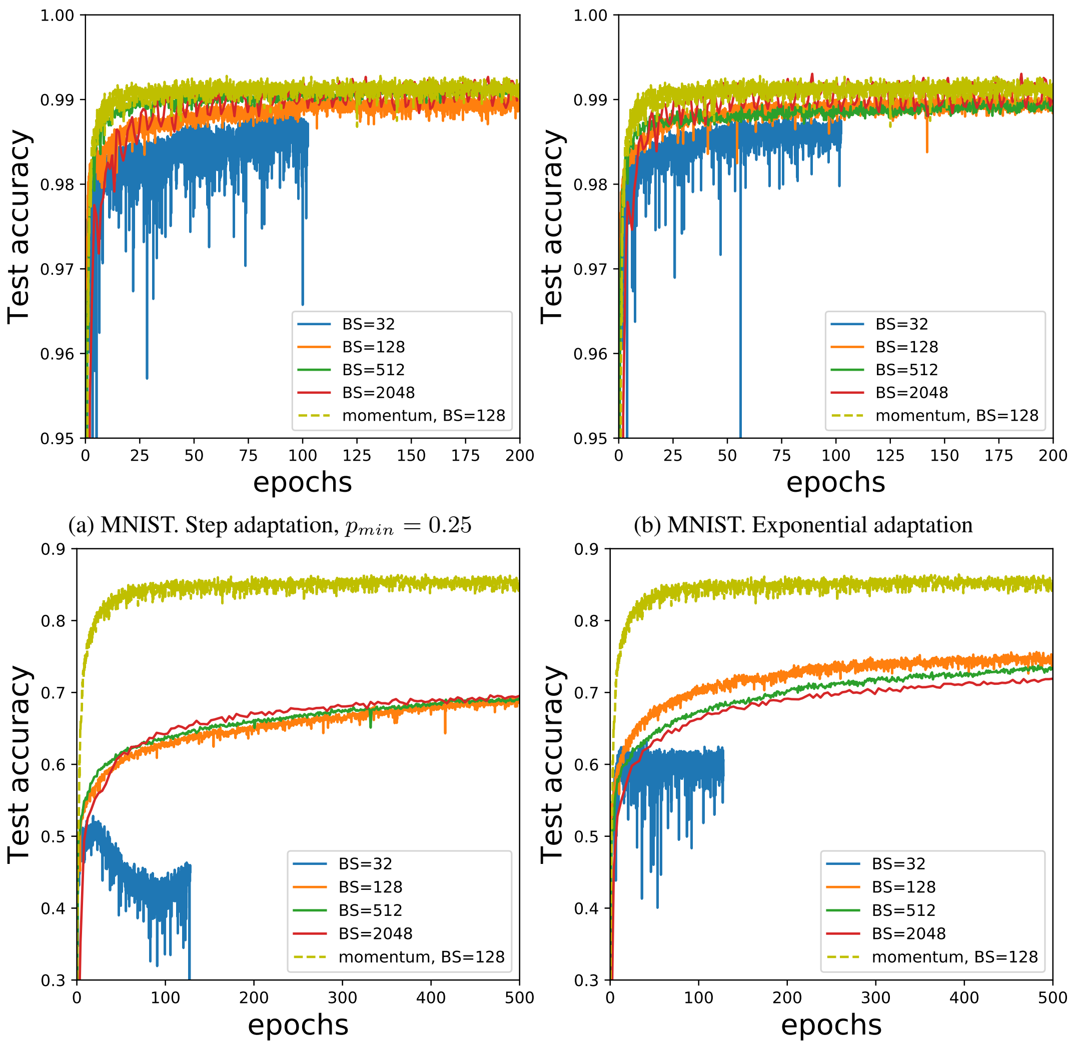
We tested the optimizers on two problems: CNN training on the MNIST and CIFAR-10 datasets. On MNIST, we used CNN with two convolutional layers, interspersed with max-pooling, and two fully-connected layers. On CIFAR-10, we used similar CNN with three convolutional layers and three fully-connected layers (the details are described in part A of the Supplementary materials). We used a cross-entropy loss function. Both networks were regularized by $L2$-regularization.

4.2 Results

First, we looked at the performance of the naive implementation of RPROP algorithm, where the full gradients where substituted for the stochastic gradients. Look at the Figure 3 to see the dependence of test accuracy and training loss on numbers of training epochs. As expected, the performance of such optimizer highly depends on the variance of the stochastic gradient. We also see that for CIFAR-10 small mini-batch training the loss starts to increase simultaneously with the accuracy decrease, so there is no convergence.

Next, we use our implementation of the ProbRPROP algorithm to check its stability to the noise. We tested the performance of ProbRPROP for three different adaptation functions. During usage of a linear adaptation function, probabilistic RPROP failed to optimize the loss function showing nearly random test accuracy. In Figure **??** you see the results for step and exponential adaptation functions. We see that the results for different mini-batches are similar and better than the RPROP results with large batch size. In the same figure, you see the results for SGD with momentum optimizer. For the CIFAR-10 data set SGD with momentum still performs better.

One important advantage of the RPROP algorithm is the absence of global learning rate and independence of initial learning rate. We want to study how sensitive our modification to initial learning rate Δ_0. In Figure 5 you see that ProbRPROP algorithm converges to the same solution starting from a wide range of initial step sizes Δ_0 from 10^{-5} to 10^{-2}.



(a) MNIST. Step adaptation, $p_{min} = 0.25$

(b) MNIST. Exponential adaptation

(c) CIFAR-10. Step adaptation, $p_{min} = 0.25$

(d) CIFAR-10. Exponential adaptation

Figure 4: ProbRPROP algorithm is more stable than RPROP. It fails to optimize the loss function only for very small batch size 32. The test accuracy of ProbRPROP is better than RPROP with large batch sizes. However, it is still smaller than test accuracy of SGD with momentum.

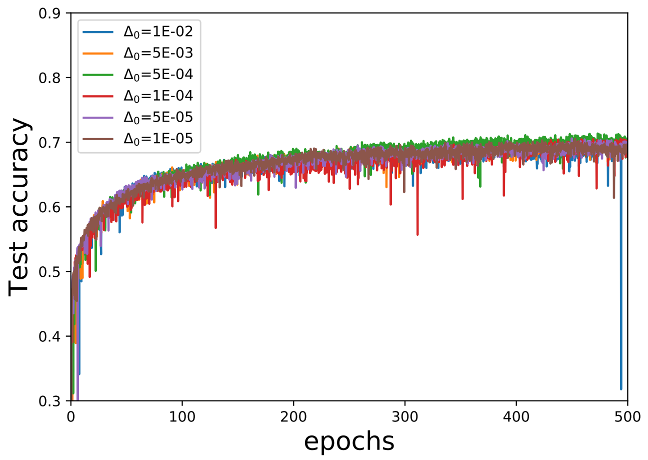


Figure 5: Probabilistic RPROP algorithm convergence does not depend on initial learning rate Δ_0. The test accuracies for ProbRPROP with exponential adaptation function during training of CIFAR-10 with batch equal to 128 are almost the same for large range of Δ_0.

5 Conclusion and Discussion

In this paper, we studied the RPROP algorithm as a stochastic optimizer. We showed that naive implementation of the RPROP algorithm with the replacement of the gradients to the stochastic gradients works much worse that deterministic RPROP and is sensitive to the stochastic gradient variance. This is due to the inaccurate adaptation of step size and wrong direction of updates caused by the variance in the stochastic gradients.

We developed a modification of the RPROP algorithm that is stable to the noise in the stochastic gradient. The performance of this modification is better than RPROP with large batch size. However, it is lower than the performance of SGD with momentum. In addition, we showed that it is insensitive to initial step size that also simplifies usage of this optimizer.

Because of the limited time for this project, we haven't studied the performance of this optimizer in bigger data sets such as CIFAR-100 or ImageNet. It would be good to look at the performance of ProbRPROB algorithm on some other tasks such as natural language processing tasks. In addition, it can be useful to study the performance of ProbRPROP on the set of small stochastic optimization tasks with typical problems such as saddle points and flat regions (e.g. Unit Tests for Stochastic Optimization(Schaul et al., 2013)). Also, it is desirable to show that this algorithm works well for different machine learning models.

There are several possible future directions for this project. First, it is important to develop the connection with recent work on Hyper gradient descent (Baydin et al., 2017) as it uses similar update rule to find optimal learning rate in every iteration. Secondly, similar to (Balles and Hennig, 2017) it is possible to incorporate momentum inside the calculation of the soft sign. We expect that this can improve the results for our optimizer as it was in (Balles and Hennig, 2017).

References

L. Balles and P. Hennig. Follow the signs for robust stochastic optimization. *CoRR*, abs/1705.07774, 2017. URL `http://arxiv.org/abs/1705.07774`.

A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. D. Wood. Online learning rate adaptation with hypergradient descent. *CoRR*, abs/1703.04782, 2017. URL `http://arxiv.org/abs/1703.04782`.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, Feb. 2012. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2188385.2188395`.

L. Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), May 2013*, May 2013.

R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points — online stochastic gradient for tensor decomposition. In P. Grünwald, E. Hazan, and S. Kale, editors, *Proceedings of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, pages 797–842, Paris, France, 03–06 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v40/Ge15.html`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

C. Igel and M. Hüsken. Improving the rprop learning algorithm, 01 2000.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL `http://arxiv.org/abs/1608.03983`.

A. Mosca and G. D. Magoulas. Adapting resilient propagation for deep learning. *CoRR*, abs/1509.04612, 2015. URL `http://arxiv.org/abs/1509.04612`.

A. Mosca and G. D. Magoulas. Adapting resilient propagation for deep learning. *ESANN 2017 Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2017. URL `https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2017-50.pdf`.

Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). 1983.

B. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL `http://www.sciencedirect.com/science/article/pii/0041555364901375`.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, 1993. doi: 10.1109/ICNN.1993.298623.

T. Schaul, I. Antonoglou, and D. Silver. Unit tests for stochastic optimization. *CoRR*, abs/1312.6055, 2013. URL `http://arxiv.org/abs/1312.6055`.

C. Tan, S. Ma, Y.-H. Dai, and Y. Qian. Barzilai-borwein step size for stochastic gradient descent. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 685–693. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6286-barzilai-borwein-step-size-for-stochastic-gradient-descent.pdf`.

T. Tieleman and G. Hinton. Rmsprop. divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning, Lecture 6.5.*, 2012.

M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL `http://arxiv.org/abs/1212.5701`.

C. Zhang, Q. Liao, A. Rakhlin, B. Miranda, N. Golowich, and T. A. Poggio. Theory of deep learning iii: Generalization properties of sgd. 2017.

Supplementary materials

A. Network architecture

MNIST

We train convolutional neural network with two convolutional layers (32 filters of size 5×5, 64 filters of size 5×5) and two fully-connected layers of 1024 and 10 units with ReLU activation. The output layer has 10 units with softmax activation. We use cross-entropy loss function and apply $L2$-regularization on all weights, but not the biases.

CIFAR-10

We train a convolutional neural network (CNN) with three convolutional layers (64 filters of size 5×5, 96 filters of size 3×3, and 128 filters of size 3×3) interspersed with max-pooling over 3×3 areas with stride 2. Two fully-connected layers with 512 and 256 units follow. We use ReLU activation function for all layers. The output layer has 10 units for the 10 classes of CIFAR-10 with softmax activation. We use cross-entropy loss function and apply $L2$-regularization on all weights, but not the biases.

B. Optimal η^+ for fixed η^-

Here we want to show additional motivation to choose $\eta^+ = \frac{1}{\eta^-}$. For this, we train RPROP optimizer with fixed η^- and different η^+. During training, we use large mini-batches (2048) and the CIFAR-10 data set. As you see in Figure 6 for smaller η^+ the accuracies are significantly worse, whereas for bigger η^+ the optimizer completely fails. The possible reason for this is unbalanced decrease and increase of step size that results in saturation of the parameters to the Δ_{min} or Δ_{max}.

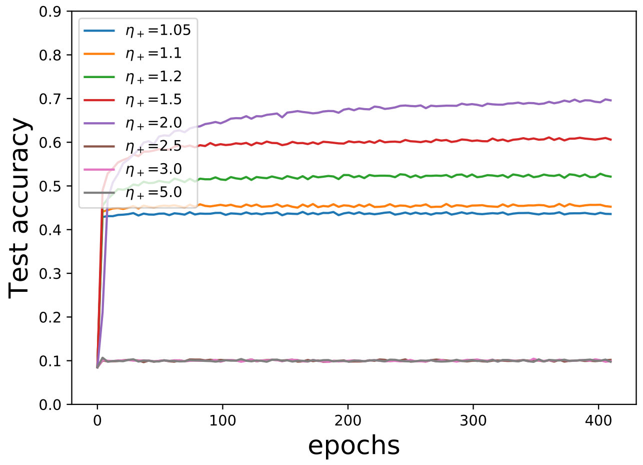


Figure 6: Different runs of the RPROP algorithm with $\eta^- = \frac{1}{2}$. For tested $\eta^+ > \frac{1}{\eta^-}$ algorithm failed to optimize the loss function.