Incorporating Nesterov Momentum into *Adam*

Timothy Dozat

1 Introduction

When attempting to improve the performance of a deep learning system, there are more or less three approaches one can take: the first is to improve the structure of the model, perhaps adding another layer, switching from simple recurrent units to LSTM cells [4], or–in the realm of NLP–taking advantage of syntactic parses (e.g. as in [13, et seq.]); another approach is to improve the initialization of the model, guaranteeing that the early-stage gradients have certain beneficial properties [3], or building in large amounts of sparsity [6], or taking advantage of principles of linear algebra [15]; the final approach is to try a more powerful learning algorithm, such as including a decaying sum over the previous gradients in the update [12], by dividing each parameter update by the L_2 norm of the previous updates for that parameter [2], or even by foregoing first-order algorithms for more powerful but more computationally costly second order algorithms [9]. This paper has as its goal the third option—improving the quality of the final solution by using a faster, more powerful learning algorithm.

2 Related Work

2.1 Momentum-based algorithms

Gradient descent is a simple, well-known, and generally very robust optimization algorithm where the gradient of the function to be minimized with respect to the parameters ($\nabla f(\theta_{t-1})$) is computed, and a portion η of that gradient is subtracted off of the parameters:

Algorithm 1 Gradient Descent

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{g}_t$
```

Classical momentum [12] accumulates a decaying sum (with decay constant μ) of the previous gradients into a momentum vector **m**, and using that instead of the true gradient. This has the advantage of accelerating gradient descent learning along dimensions where the gradient remains relatively consistent across training steps and slowing it along turbulent dimensions where the gradient is significantly oscillating.

Algorithm 2 Classical Momentum

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \mathbf{g}_t$
$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{m}_t$
```

[14] show that *Nesterov's accelerated gradient* (NAG) [11]–which has a provably better bound than gradient descent–can be rewritten as a kind of improved momentum. If we can substitute the definition for $\mathbf{m}_t$ in place of the symbol $\mathbf{m}_t$ in the parameter update as in (2)

$$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{m}_t \tag{1}$$

$$\theta_t \leftarrow \theta_{t-1} - \eta \mu \mathbf{m}_{t-1} - \eta \mathbf{g}_t \tag{2}$$

we can see that the term $\mathbf{m}_{t-1}$ doesn't depend on the current gradient $\mathbf{g}_t$—so in principle, we can get a superior step direction by applying the momentum vector to the parameters *before* computing the gradient. The authors provide empirical evidence that this algorithm is superior to the gradient descent, classi-

Algorithm 3 Nesterov's accelerated gradient

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta\mu\mathbf{m}_{t-1})$
$\mathbf{m}_t \leftarrow \mu\mathbf{m}_{t-1} + \mathbf{g}_t$
$\theta_t \leftarrow \theta_{t-1} - \eta\mathbf{m}_t$
```

cal momentum, and Hessian-Free [9] algorithms for conventionally difficult optimization objectives.

2.2 L_2 norm-based algorithms

[2] present *adaptive subgradient descent* (AdaGrad), which divides η of every step by the L_2 norm of all previous gradients; this slows down learning along dimensions that have already changed significantly and speeds up learning along dimensions that have only changed slightly, stabilizing the model's representation of common features and allowing it to rapidly "catch up" its representation of rare features.[1]

Algorithm 4 AdaGrad

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\mathbf{n}_t \leftarrow \mathbf{n}_{t-1} + \mathbf{g}_t^2$
$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\mathbf{g}_t}{\sqrt{\mathbf{n}_t} + \varepsilon}$
```

One notable problem with AdaGrad is that the norm vector $\mathbf{n}$ eventually becomes so large that training slows to a halt, preventing the model from reaching the local minimum; [16] go on to motivate *RMSProp*, an alternative to AdaGrad that replaces the sum in $\mathbf{n}_t$ with a decaying mean parameterized here by ν. This allows the model to continue to learn indefinitely.

Algorithm 5 RMSProp

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\mathbf{n}_t \leftarrow \nu\mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2$
$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\mathbf{g}_t}{\sqrt{\mathbf{n}_t} + \varepsilon}$
```

2.3 Combination

One might ask if combining the momentum-based and norm-based methods might provide the advantages of both. In fact, [5] successfully do so with *adaptive moment estimation* (Adam), combining classical momentum (using a decaying mean instead of a decaying sum) with RMSProp to improve performance on a number of benchmarks. In their algorithm, they include initialization bias correction terms, which offset some of the instability that initializing $\mathbf{m}$ and $\mathbf{n}$ to $\mathbf{0}$ can create.

Algorithm 6 Adam

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\mathbf{m}_t \leftarrow \mu\mathbf{m}_{t-1} + (1-\mu)\mathbf{g}_t$
$\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1-\mu^t}$
$\mathbf{n}_t \leftarrow \nu\mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2$
$\hat{\mathbf{n}}_t \leftarrow \frac{\mathbf{n}_t}{1-\nu^t}$
$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{n}}_t} + \varepsilon}$
```

[5] also include an algorithm AdaMax that replaces the L_2 norm with the L_∞ norm, removing the need for $\hat{\mathbf{n}}_t$ and replacing $\mathbf{n}_t$ and θ_t with the following updates:

$$\mathbf{n}_t \leftarrow \max(\nu\mathbf{n}_{t-1}, |\mathbf{g}_t|)$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\mathbf{n}_t + \varepsilon}$$

We can generalize this to RMSProp as well, using the L_∞ norm in the denominator instead of the L_2 norm, giving what might be called the *MaxaProp* algorithm.

3 Methods

3.1 NAG revisited

Adam combines RMSProp with classical momentum. But as [14] show, NAG is in general superior to classical momentum—so how would we go about modifying Adam to use NAG instead? First, we rewrite the NAG algorithm to be more straightforward and efficient to implement at the cost of some intuitive readability. Momentum is most effective with a warming schedule, so for completeness we parameterize μ by t as well. Here, the vector $\bar{\mathbf{m}}$ contains the gradient update for the cur-

Algorithm 7 NAG rewritten

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\mathbf{m}_t \leftarrow \mu_t\mathbf{m}_{t-1} + \mathbf{g}_t$
$\bar{\mathbf{m}}_t \leftarrow \mathbf{g}_t + \mu_{t+1}\mathbf{m}_t$
$\theta_t \leftarrow \theta_{t-1} - \eta\bar{\mathbf{m}}_t$
```

[1] Most implementations of this kind of algorithm include an ε parameter to keep the denominator from being too small and resulting in an irrecoverably large step

rent timestep $\mathbf{g}_t$ in addition to the momentum vector update for the next timestep $\mu_{t+1}\mathbf{m}_t$, which needs to be applied before taking the gradient at the next timestep. We don't need to apply the momentum vector for the current timestep anymore because we already applied it in the last update of the parameters, at timestep $t-1$.

3.2 Applying NAG to Adam

Ignoring the initialization bias correction terms for the moment, Adam's update rule can be written in terms of the previous momentum/norm vectors and current gradient update as in (3).

$$\begin{aligned}\theta_t \leftarrow & \theta_{t-1} - \eta \frac{\mu \mathbf{m}_{t-1}}{\sqrt{\nu \mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2} + \varepsilon} \\ & - \eta \frac{(1-\mu)\mathbf{g}_t}{\sqrt{\nu \mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2} + \varepsilon}\end{aligned} \tag{3}$$

In rewritten NAG, we would take the first part of the step and apply it before taking the gradient of the cost function f–however, the denominator depends on $\mathbf{g}_t$, so we can't take advantage of the trick used in NAG for this equation. However, ν is generally chosen to be very large (normally $> .9$), so the difference between $\mathbf{n}_{t-1}$ and $\mathbf{n}_t$ will in general be very small. We can then replace $\mathbf{n}_t$ with $\mathbf{n}_{t-1}$ without losing too much accuracy:

$$\begin{aligned}\theta_t \leftarrow & \theta_{t-1} - \eta \frac{\mu \mathbf{m}_{t-1}}{\sqrt{\mathbf{n}_{t-1}} + \varepsilon} \\ & - \eta \frac{(1-\mu)\mathbf{g}_t}{\sqrt{\nu \mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2} + \varepsilon}\end{aligned} \tag{4}$$

The the first term in the expression in (4) no longer depends on $\mathbf{g}_t$, meaning here we *can* use the Nesterov trick; this give us the following expressions for $\bar{\mathbf{m}}_t$ and θ_t:

$$\bar{\mathbf{m}}_t \leftarrow (1-\mu_t)\mathbf{g}_t + \mu_{t+1}\mathbf{m}_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\bar{\mathbf{m}}_t}{\sqrt{\mathbf{v}_t} + \varepsilon}$$

All that's left is to determine how to include the initialization bias correction terms, taking into consideration that $\mathbf{g}_t$ comes from the current timestep but $\mathbf{m}_t$ comes from the subsequent timestep. This gives us the following, final form of the Nesterov-accelerated adaptive moment estimation (Nadam) algorithm. AdaMax can make use of the Nesterov acceleration trick identically (NadaMax).

Algorithm 8 Nesterov-accelerated adaptive moment estimation

```
$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\hat{\mathbf{g}} \leftarrow \frac{\mathbf{g}_t}{1-\prod_{i=1}^{t}\mu_i}$
$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + (1-\mu)\mathbf{g}_t$
$\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1-\prod_{i=1}^{t+1}\mu_i}$
$\mathbf{n}_t \leftarrow \nu \mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2$
$\hat{\mathbf{n}}_t \leftarrow \frac{\mathbf{n}_t}{1-\nu^t}$
$\bar{\mathbf{m}}_t \leftarrow (1-\mu_t)\hat{\mathbf{g}}_t + \mu_{t+1}\hat{\mathbf{m}}_t$
$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\bar{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{n}}_t} + \varepsilon}$
```

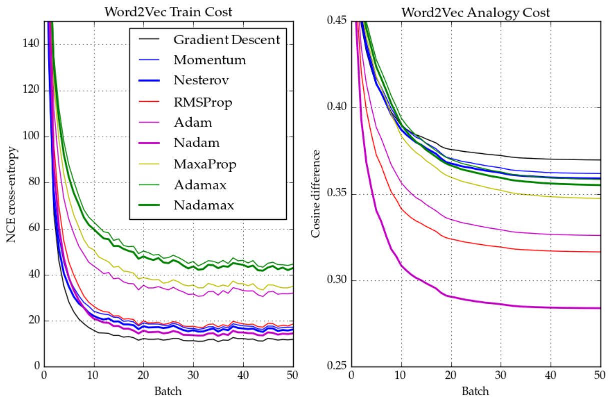
4 Experiments

To test this algorithm, we compared the performance of nine algorithms–GD, Momentum, NAG, RMSProp, Adam, Nadam, MaxaProp, AdaMax, and Nadamax–on three benchmarks–word2vec [10], MNIST image classification [7], and LSTM language models [17]. All algorithms used $\nu = .999$ and $\varepsilon = 1e^{-8}$ as suggested in [5], with a momentum schedule given by $\mu_t = \mu(1 - .5 \times .96^{\frac{t}{250}})$ with $\mu = .99$, similar to the recommendation in [14]. Only the learning rate η differed across algorithms and experiments. The algorithms were all coded using Google's TensorFlow [1] API and the experiments were done using the built-in TensorFlow models, making only small edits to the default settings. All algorithms used initialization bias correction.

4.1 Word2Vec

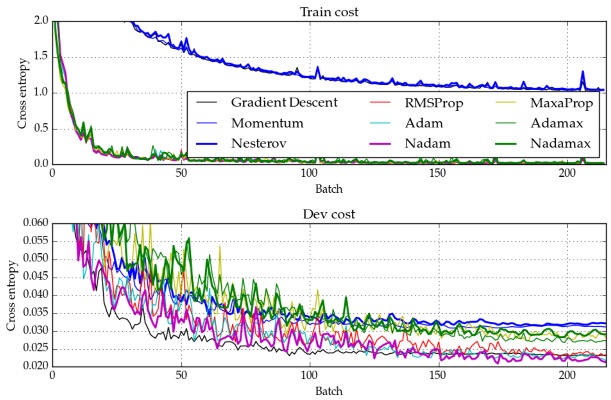
Word2vec [10] word embeddings were trained using each of the nine algorithms. Approximately 100MB of cleaned text[2] from Wikipedia were used as the source text, and any word not in the top 50000 words was replaced with `UNK`. 128-dimensional vectors with a left and right context size of 1 were trained using noise-contrastive estimation with 64 negative samples. Validation was done using the word analogy task; we report the average cosine differene ($\frac{1-\text{cossim}(\mathbf{x},\mathbf{y})}{2}$) between the analogy vector and the embedding of the correct answer to the analogy. The best results were achieved when, each column of the embedding vector had a 50% chance to be dropped during training. The results are in Figure 1.

[2] `http://mattmahoney.net/dc/text8.zip`



	GD	Mom	NAG
Test loss	.368	.361	.358
	RMS	Adam	Nadam
Test loss	.316	.325	**.284**
	Maxa	A-max	N-max
Test loss	.346	.356	.355

Figure 1: Training of word2vec word embeddings



	GD	Mom	NAG
Test loss	.0202	.0263	.0283
	RMS	Adam	Nadam
Test loss	**.0172**	.0175	.0183
	Maxa	A-max	N-max
Test loss	.0195	.0231	.0204

Figure 2: Training of handwritten digit recognition

The methods with RMSProp produced word vectors that represented relationships between words significantly better than the other methods, but RMSProp with Nesterov momentum (Nadam) clearly outperformed RMSProp with no momentum and with classical momentum (Adam). Notice also that the algorithms with NAG consistently outperform the algorithms witn classical momentum.

4.2 Image Recognition

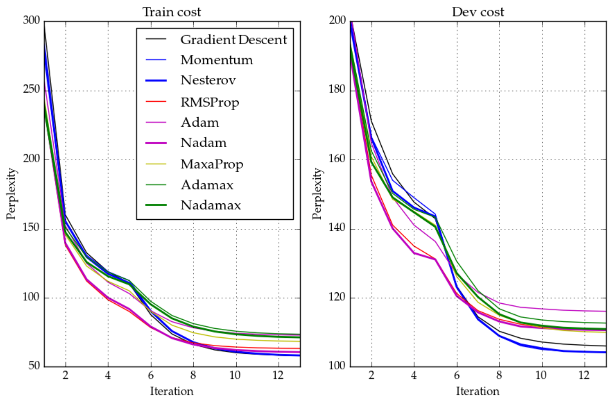
MNIST [7] is a classic benchmark for testing algorithms. We train a CNN with two convolutional layers of 5×5 filters of depth 32 and depth 64 alternating with 2×2 max pooling layers with a stride of 2, followed by a fully connected layer of 512 nodes and a softmax layer on top of that–this is the default model in TensorFlow's basic MNIST CNN model. The annealing schedule was modified to decay faster and the model was trained for more epochs in order to smooth out the error graphs. The results are given in Figure 2.

In this benchmark, Nadam receives the lowest development error–as predicted–but on the test set, all momentum-based methods underperform their simple counterparts without momentum. However, [5] use a larger CNN and find that Adam achieves the lowest performance. Furthermore, the hyperparameters here have not been tuned for this task (except the global learning rate), and in other training runs Nadam performed noticeably better with a higher value of ε.

4.3 LSTM Language Model

The final benchmark we present is the task of predicting the next word of a sentence given the previous words in a sentence. The dataset used here comes from the Penn TreeBank [8], and the model uses a two-layer stacked LSTM with 200 cells at each layer, reproducing [17] on a smaller scale. All default settings in the small TensorFlow implementation were retained, with the exception that the model included L_2 regularization in order to offset the lack of dropout. The results of training are presented in Figure 3.

In this benchmark, algorithms that incorporated RMSProp or MaxaProp did noticeably worse than gradient descent and those that only use momentum, and tuning hyperparameters makes little difference. It's unclear why this is–it's conceivable that the assumptions RMS/MaxaProp make about how features are represented don't hold in the case of language models, LSTMs, or recurrent NNs more broadly. It's also curious that classical momentum outperformed Nesterov momentum, although that may be due to the relatively small size of the models. Crucially, even though Nadam wasn't the op-



	GD	Mom	NAG
Test perp	100.8	**99.3**	99.8
	RMS	Adam	Nadam
Test perp	106.7	111.0	105.5
	Maxa	A-max	N-max
Test perp	106.3	108.5	107.0

Figure 3: Training of a Penn TreeBank language model

timal algorithm in this case, it significantly outperformed Adam, supporting our hypothesis that using Nesterov momentum will improve Adam.

5 Discussion

While the results are a little messy, we can infer a number of things: the first is that the hitherto untested MaxaProp-based methods, including AdaMax and NadaMax, generally do worse than at least some of the other algorithms. In the word2vec task, the algorithms with Nesterov momentum consistently achieve better results than the ones with only classical momentum, supporting the intuition behind the hypothesis that Nadam should be better than Adam. We also see that Nadam produces by far the best word vectors for this task when dropout is included (which is not normally done but in our experiments categorically improves results). With the MNIST task, we find that including RMSProp makes a significant difference in the quality of the final classifier, likely owing to RMSProp's ability to adapt to different depths. Nadam performed best on the development set, but contra [5], RMSProp surpassed it on the test set. This may be due to the smaller size of our model, the relative easiness of the task (CIFAR-10 might make a better benchmark in a less time-sensitive setting), our choice to only use default settings, random fluctuations that arise from dropout, or possibly even slight overfitting (which can be clearly seen in Nadam–but not in RMSProp–when hyperparameters *are* tuned). Finally, in the LSTM-LM, we find that using RMSProp hinders performance, even when hyperparameters are extensively tuned, but when it is included anyway, adding Nesterov momentum to it as in Nadam improves it over not using any momentum or only using classical momentum.

In two of the three benchmarks, we found a rather significant difference between the performance of models trained with Adam and that of those trained with Nadam—why might this be? When the denominator of RMSProp is small, the effective step size of a parameter update becomes very large, and when the denominator is large, the model requires a lot of work to change what it's learned–so ensuring the quality of the step direction and reducing as much noise as possible becomes critical. We can understand classical momentum as being a version of Nesterov momentum that applies an old, outdated momentum vector that only uses past gradients to the parameter updates, rather than the most recent, up-to-date momentum vector computed using the current gradient as well. It logically follows then that Nesterov momentum would produce a gradient update superior to classical momentum, which RMSProp can then take full advantage of.

6 Conclusion

Adam essentially combines two algorithms known to work well for different reasons–momentum, which points the model in a better direction, and RMSProp, which adapts how far the model goes in that direction on a per-parameter basis. However, Nesterov momentum is theoretically and often empirically superior to momentum—so it makes sense to see how this affects the result of combining the two approaches. We compared Adam and Nadam (Adam with Nesterov momentum), in addition to a slew of related algorithms, and find the in most cases the improvement of Nadam over Adam is fairly dramatic. While Nadam wasn't always the best algorithm to chose, it seems like if one is going to use momentum with RMSProp to train their model, they may as well use Nesterov momentum.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[3] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015.

[7] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[8] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[9] James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[11] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[12] Boris Teodorovich Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[13] Richard Socher, Cliff C Lin, Andrew Y Ng, and Chris Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[14] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.

[15] Sachin S Talathi and Aniket Vartak. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*, 2015.

[16] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

[17] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.