Optimization for Deep Learning

Sebastian Ruder
PhD Candidate, INSIGHT Research Centre, NUIG
Research Scientist, AYLIEN
@seb_ruder

Advanced Topics in Computational Intelligence
Dublin Institute of Technology

24.11.17

Agenda

Introduction

- Gradient descent is a way to minimize an objective function $J(\theta)$
  - $\theta \in \mathbb{R}^d$: model parameters
  - η: learning rate
  - $\nabla_\theta J(\theta)$: gradient of the objective function with regard to the parameters
- Updates parameters **in opposite direction** of gradient.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

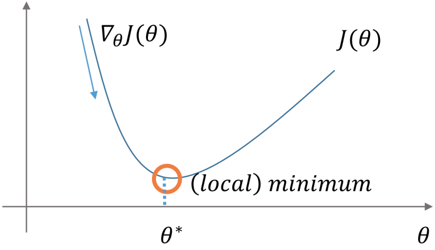


Figure: Optimization with gradient descent

Gradient descent variants

1. Batch gradient descent
2. Stochastic gradient descent
3. Mini-batch gradient descent

Difference: Amount of data used per update

Batch gradient descent

- Computes gradient with the **entire** dataset.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(
    loss_function, data, params)
  params = params - learning_rate * params_grad
```

Listing 1: Code for batch gradient descent update

- Pros:
  - Guaranteed to converge to **global** minimum for **convex** error surfaces and to a **local** minimum for **non-convex** surfaces.
- Cons:
  - **Very slow**.
  - Intractable for datasets that **do not fit in memory**.
  - **No online learning**.

Stochastic gradient descent

- Computes update for **each** example $x^{(i)} y^{(i)}$.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(
      loss_function, example, params)
    params = params - learning_rate * params_grad
```

Listing 2: Code for stochastic gradient descent update

- Pros
  - **Much faster** than batch gradient descent.
  - Allows **online learning**.
- Cons
  - **High variance** updates.

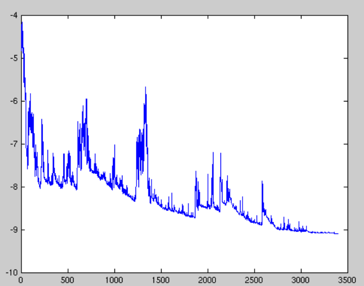


Figure: SGD fluctuation (Source: Wikipedia)

Batch gradient descent vs. SGD fluctuation

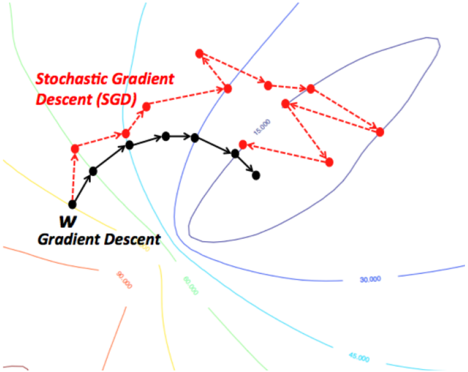


Figure: Batch gradient descent vs. SGD fluctuation (Source: wikidocs.net)

- SGD shows same convergence behaviour as batch gradient descent if learning rate is **slowly decreased (annealed)** over time.

Mini-batch gradient descent

- Performs update for every **mini-batch** of n examples.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(
      loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Listing 3: Code for mini-batch gradient descent update

- Pros
  - **Reduces variance** of updates.
  - Can exploit **matrix multiplication** primitives.
- Cons
  - **Mini-batch size** is a hyperparameter. Common sizes are 50-256.
- Typically the algorithm of choice.
- Usually referred to as SGD even when mini-batches are used.

Method	Accuracy	Update Speed	Memory Usage	Online Learning
Batch gradient descent	Good	Slow	High	No
Stochastic gradient descent	Good (with annealing)	High	Low	Yes
Mini-batch gradient descent	Good	Medium	Medium	Yes

Table: Comparison of trade-offs of gradient descent variants

Challenges

- Choosing a **learning rate**.
- Defining an **annealing schedule**.
- Updating features to **different extent**.
- **Avoiding suboptimal minima**.

Gradient descent optimization algorithms

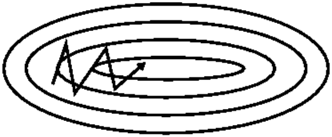
1. Momentum
2. Nesterov accelerated gradient
3. Adagrad
4. Adadelta
5. RMSprop
6. Adam
7. Adam extensions

Momentum

- SGD has trouble navigating **ravines**.
- Momentum [Qian, 1999] helps SGD **accelerate**.
- Adds a fraction γ of the update vector of the past step v_{t-1} to current update vector v_t. Momentum term γ is usually set to 0.9.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{1}$$

(a) SGD without momentum



(b) SGD with momentum

Figure: Source: Genevieve B. Orr

- **Reduces updates** for dimensions whose gradients **change directions**.
- **Increases updates** for dimensions whose gradients **point in the same directions**.

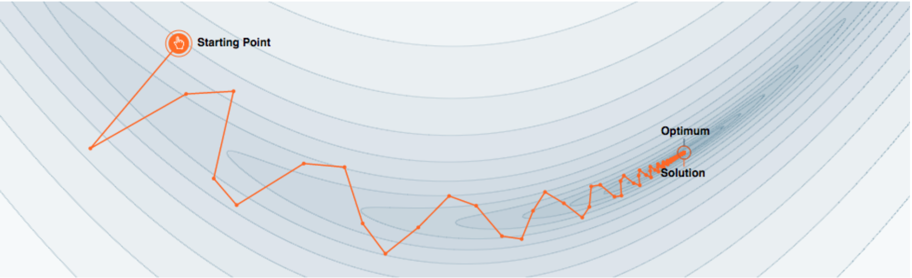


Figure: Optimization with momentum (Source: distill.pub)

Nesterov accelerated gradient

- Momentum **blindly accelerates** down slopes: First computes gradient, then makes a big jump.
- Nesterov accelerated gradient (NAG) [Nesterov, 1983] first makes a big jump in the direction of the previous accumulated gradient $\theta - \gamma v_{t-1}$. Then measures where it ends up and makes a correction, resulting in the complete update vector.

$$\begin{aligned} v_t &= \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{2}$$

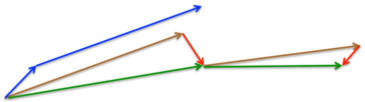


Figure: Nesterov update (Source: G. Hinton's lecture 6c)

Adagrad

- Previous methods: **Same learning rate** η for all parameters θ.
- Adagrad [Duchi et al., 2011] **adapts** the learning rate to the parameters (**large** updates for **infrequent** parameters, **small** updates for **frequent** parameters).
- SGD update: $\theta_{t+1} = \theta_t - \eta \cdot g_t$
  - $g_t = \nabla_{\theta_t} J(\theta_t)$
- Adagrad divides the learning rate by the **square root of the sum of squares of historic gradients**.
- Adagrad update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{3}$$

  - $G_t \in \mathbb{R}^{d \times d}$: diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t
  - ϵ: smoothing term to avoid division by zero
  - $\odot$: element-wise multiplication

- Pros
  - Well-suited for dealing with **sparse data**.
  - Significantly **improves robustness** of SGD.
  - Lesser need to manually tune learning rate.
- Cons
  - **Accumulates squared gradients** in denominator. Causes the learning rate to **shrink** and become **infinitesimally small**.

Adadelta

- Adadelta [Zeiler, 2012] restricts the window of accumulated past gradients to a **fixed size**. SGD update:

$$\begin{aligned} \Delta\theta_t &= -\eta \cdot g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t \end{aligned} \tag{4}$$

- Defines **running average** of squared gradients $E[g^2]_t$ at time t:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \tag{5}$$

  - γ: fraction similarly to momentum term, around 0.9
- Adagrad update:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{6}$$

- Preliminary Adadelta update:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{7}$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{8}$$

- Denominator is just root mean squared (RMS) error of gradient:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t \tag{9}$$

- Note: **Hypothetical units do not match**.
- Define **running average of squared parameter updates** and RMS:

$$\begin{aligned} E[\Delta\theta^2]_t &= \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \\ RMS[\Delta\theta]_t &= \sqrt{E[\Delta\theta^2]_t + \epsilon} \end{aligned} \tag{10}$$

- Approximate with $RMS[\Delta\theta]_{t-1}$, replace η for **final Adadelta update**:

$$\begin{aligned} \Delta\theta_t &= -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t \end{aligned} \tag{11}$$

RMSprop

- Developed independently from Adadelta around the same time by Geoff Hinton.
- Also divides learning rate by a **running average of squared gradients**.
- RMSprop update:

$$\begin{aligned} E[g^2]_t &= \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \tag{12}$$

  - γ: decay parameter; typically set to 0.9
  - η: learning rate; a good default value is 0.001

Adam

- Adaptive Moment Estimation (Adam) [Kingma and Ba, 2015] also stores **running average of past squared gradients** v_t like Adadelta and RMSprop.
- Like Momentum, stores **running average of past gradients** m_t.

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
\end{aligned}
\tag{13}
$$

  - m_t: first moment (mean) of gradients
  - v_t: second moment (uncentered variance) of gradients
  - β_1, β_2: decay rates

- m_t and v_t are initialized as 0-vectors. For this reason, they are biased towards 0.
- Compute bias-corrected first and second moment estimates:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{14}$$

- Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{15}$$

Adam extensions

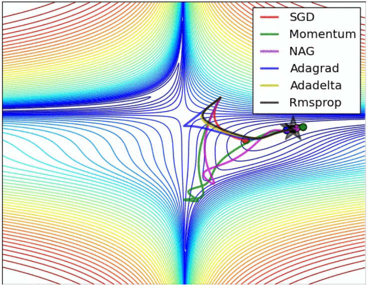
1. AdaMax [Kingma and Ba, 2015]
    - Adam with ℓ_∞ norm
2. Nadam [Dozat, 2016]
    - Adam with Nesterov accelerated gradient

Update equations

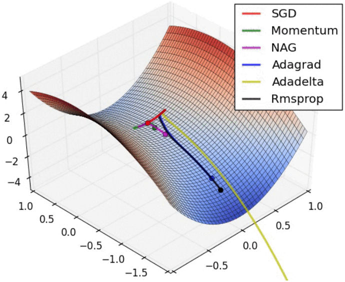
Method	Update equation
SGD	$g_t = \nabla_{\theta_t} J(\theta_t)$ $\Delta\theta_t = -\eta \cdot g_t$ $\theta_t = \theta_t + \Delta\theta_t$
Momentum	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta g_t$
NAG	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
Adagrad	$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
Adadelta	$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
RMSprop	$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Adam	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

Table: Update equations for the gradient descent optimization algorithms.

Visualization of algorithms



(a) SGD optimization on loss surface contours



(b) SGD optimization on saddle point

Figure: Source and full animations: Alec Radford

Which optimizer to choose?

- Adaptive learning rate methods (Adagrad, Adadelta, RMSprop, Adam) are **particularly useful for sparse features**.
- Adagrad, Adadelta, RMSprop, and Adam work well in similar circumstances.
- [Kingma and Ba, 2015] show that bias-correction helps Adam **slightly outperform RMSprop**.

Parallelizing and distributing SGD

1. Hogwild! [Niu et al., 2011]
   - Parallel SGD updates on CPU
   - Shared memory access **without parameter lock**
   - Only works for **sparse input data**
2. Downpour SGD [Dean et al., 2012]
   - **Multiple replicas** of model on subsets of training data run in parallel
   - Updates sent to parameter server; **updates fraction of model parameters**
3. Delay-tolerant Algorithms for SGD [Mcmahan and Streeter, 2014]
   - Methods also adapt to **update delays**
4. TensorFlow [Abadi et al., 2015]
   - Computation graph is split into a **subgraph for every device**
   - Communication takes place using Send/Receive node pairs
5. Elastic Averaging SGD [Zhang et al., 2015]
   - **Links parameters elastically** to a center variable stored by parameter server

Additional strategies for optimizing SGD

1. Shuffling and Curriculum Learning [Bengio et al., 2009]
   - Shuffle training data after every epoch to **break biases**
   - Order training examples to **solve progressively harder problems**; infrequently used in practice
2. Batch normalization [Ioffe and Szegedy, 2015]
   - **Re-normalizes every mini-batch** to zero mean, unit variance
   - Must-use for computer vision
3. Early stopping
   - "*Early stopping (is) beautiful free lunch*" (Geoff Hinton)
4. Gradient noise [Neelakantan et al., 2015]
   - Add Gaussian noise to gradient
   - Makes model **more robust to poor initializations**

Outlook

1. Tuned SGD vs. Adam
2. SGD with restarts
3. Learning to optimize
4. Understanding generalization in Deep Learning
5. Case studies

Tuned SGD vs. Adam

- Many recent papers use **SGD with learning rate annealing**.
- SGD with tuned learning rate and momentum is **competitive with Adam** [Zhang et al., 2017b].
- Adam **converges faster**, but **underperforms SGD** on some tasks, e.g. Machine Translation [Wu et al., 2016].
- Adam with **2 restarts and SGD-style annealing** converges faster and outperforms SGD [Denkowski and Neubig, 2017].
- **Increasing the batch size** may have the same effect as decaying the learning rate [Smith et al., 2017].

SGD with restarts

- At each restart, the learning rate is initialized to some value and decreases with cosine annealing [Loshchilov and Hutter, 2017].
- Converges $2\times$ to $4\times$ faster with comparable performance.

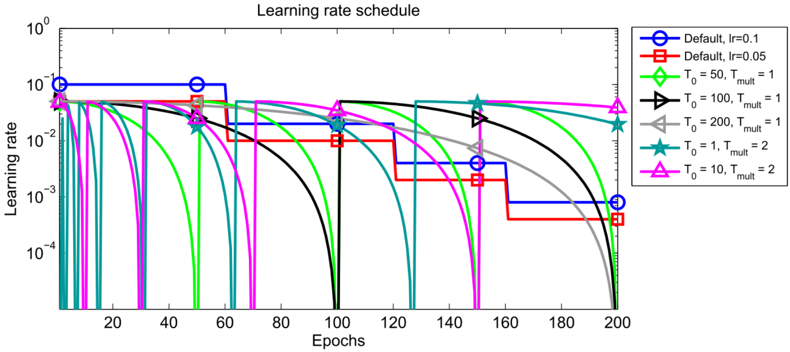


Figure: Learning rate schedules with warm restarts [Loshchilov and Hutter, 2017]

Snapshot ensembles

1. Train model until convergence with cosine annealing schedule.
2. Save model parameters.
3. Perform warm restart and repeat steps 1-3 M times.
4. Ensemble saved models.

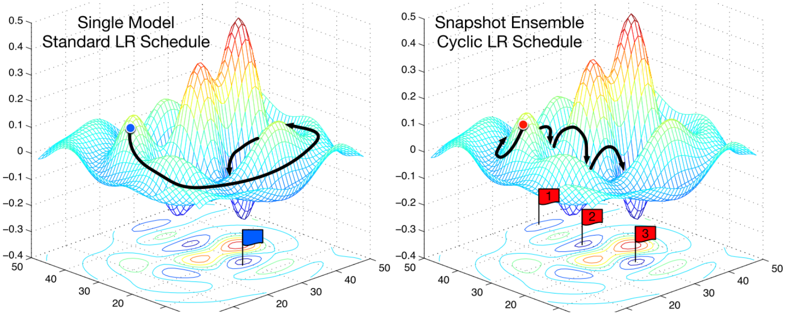


Figure: SGD vs. snapshot ensemble [Huang et al., 2017]

Learning to optimize

- Rather than manually defining an update rule, **learn it**.
- Update rules outperform existing optimizers and transfer across tasks.

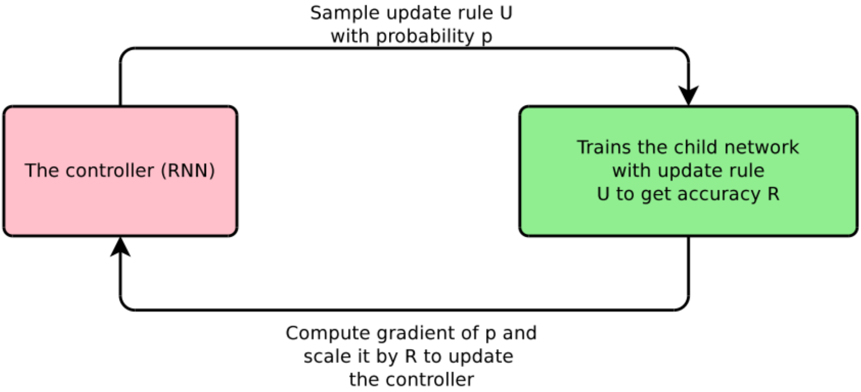


Figure: Neural Optimizer Search [Bello et al., 2017]

Discovered update rules

- **PowerSign**:

$$\alpha^{f(t)*\text{sign}(g)*\text{sign}(m)} * g \tag{16}$$

  - α: often e or 2
  - f: either 1 or a decay function of the training step t
  - m: moving average of gradients
  - Scales update by $\alpha^{f(t)}$ or $1/\alpha^{f(t)}$ depending on whether the direction of the gradient and its running average agree.

- **AddSign**:

$$(\alpha + f(t) * \text{sign}(g) * \text{sign}(m)) * g \tag{17}$$

  - α: often 1 or 2
  - Scales update by $\alpha + f(t)$ or $\alpha - f(t)$.

Understanding generalization in Deep Learning

- **Optimization** is closely tied to **generalization**.
- The number of possible local minima **grows exponentially with the number of parameters** [Kawaguchi, 2016].
- Different local minima **generalize to different extents**.
- Recent insights in understanding generalization:
  - Neural networks can **completely memorize random inputs** [Zhang et al., 2017a].
  - Sharp minima found by batch gradient descent have **high generalization error** [Keskar et al., 2017].
  - Local minima that generalize well can be **made arbitrarily sharp** [Dinh et al., 2017].
- Several submissions at ICLR 2018 on understanding generalization.

Case studies

- Deep Biaffine Attention for Neural Dependency Parsing [Dozat and Manning, 2017]
  - Adam with $\beta_1 = 0.9$, $\beta_2 = 0.9$
  - Report large positive impact on final performance of lowering β_2
- Attention is All You Need [Vaswani et al., 2017]
  - Adam with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$, learning rate η
  - $\eta = d_{\text{model}}^{-0.5} \cdot \min(\mathit{step\_num}^{-0.5}, \mathit{step\_num} \cdot \mathit{warmup\_steps}^{-1.5})$
  - $\mathit{warmup\_steps} = 4000$

Thank you for attention!
For more details and derivations of the gradient descent optimization algorithms, refer to [Ruder, 2016].

Bibliography I

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Man, D., Monga, R., Moore, S., Murray, D., Shlens, J., Steiner, B., Sutskever, I., Tucker, P., Vanhoucke, V., Vasudevan, V., Vinyals, O., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2015).
TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

[Bello et al., 2017] Bello, I., Zoph, B., Vasudevan, V., and Le, Q. V. (2017).
Neural Optimizer Search with Reinforcement Learning.
In *Proceedings of the 34th International Conference on Machine Learning*.

Bibliography II

[Bengio et al., 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009).
Curriculum learning.
Proceedings of the 26th annual international conference on machine learning, pages 41–48.

[Dean et al., 2012] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M. A., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012).
Large Scale Distributed Deep Networks.
NIPS 2012: Neural Information Processing Systems, pages 1–11.

[Denkowski and Neubig, 2017] Denkowski, M. and Neubig, G. (2017).
Stronger Baselines for Trustable Results in Neural Machine Translation.
In *Workshop on Neural Machine Translation (WNMT)*.

Bibliography III

[Dinh et al., 2017] Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017).
Sharp Minima Can Generalize For Deep Nets.
In *Proceedings of the 34 th International Conference on Machine Learning*.

[Dozat, 2016] Dozat, T. (2016).
Incorporating Nesterov Momentum into Adam.
ICLR Workshop, (1):2013–2016.

[Dozat and Manning, 2017] Dozat, T. and Manning, C. D. (2017).
Deep Biaffine Attention for Neural Dependency Parsing.
In *ICLR 2017*.

Bibliography IV

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011).
Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.
Journal of Machine Learning Research, 12:2121–2159.

[Huang et al., 2017] Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. (2017).
Snapshot Ensembles: Train 1, get M for free.
In *Proceedings of ICLR 2017*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015).
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.
arXiv preprint arXiv:1502.03167v3.

Bibliography V

[Kawaguchi, 2016] Kawaguchi, K. (2016).
Deep Learning without Poor Local Minima.
In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*.

[Keskar et al., 2017] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017).
On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.
In *Proceedings of ICLR 2017*.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. L. (2015).
Adam: a Method for Stochastic Optimization.
International Conference on Learning Representations, pages 1–13.

Bibliography VI

[Loshchilov and Hutter, 2017] Loshchilov, I. and Hutter, F. (2017).
SGDR: Stochastic Gradient Descent with Warm Restarts.
In *Proceedings of ICLR 2017*.

[Mcmahan and Streeter, 2014] Mcmahan, H. B. and Streeter, M. (2014).
Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning.
Advances in Neural Information Processing Systems (Proceedings of NIPS), pages 1–9.

[Neelakantan et al., 2015] Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. (2015).
Adding Gradient Noise Improves Learning for Very Deep Networks.
pages 1–11.

Bibliography VII

[Nesterov, 1983] Nesterov, Y. (1983).
A method for unconstrained convex minimization problem with the rate of convergence o(1/k2).
Doklady ANSSSR (translated as Soviet.Math.Docl.), 269:543–547.

[Niu et al., 2011] Niu, F., Recht, B., Christopher, R., and Wright, S. J. (2011).
Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent.
pages 1–22.

[Qian, 1999] Qian, N. (1999).
On the momentum term in gradient descent learning algorithms.
Neural networks : the official journal of the International Neural Network Society, 12(1):145–151.

Bibliography VIII

[Ruder, 2016] Ruder, S. (2016).
An overview of gradient descent optimization algorithms.
arXiv preprint arXiv:1609.04747.

[Smith et al., 2017] Smith, S. L., Kindermans, P.-J., and Le, Q. V. (2017).
Don't Decay the Learning Rate, Increase the Batch Size.
In *arXiv preprint arXiv:1711.00489*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention Is All You Need.
In *Advances in Neural Information Processing Systems*.

Bibliography IX

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016).
Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.
arXiv preprint arXiv:1609.08144.

[Zeiler, 2012] Zeiler, M. D. (2012).
ADADELTA: An Adaptive Learning Rate Method.
arXiv preprint arXiv:1212.5701.

Bibliography X

[Zhang et al., 2017a] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017a).
Understanding deep learning requires rethinking generalization.
In *Proceedings of ICLR 2017*.

[Zhang et al., 2017b] Zhang, J., Mitliagkas, I., and Ré, C. (2017b).
YellowFin and the Art of Momentum Tuning.
In *arXiv preprint arXiv:1706.03471*.

[Zhang et al., 2015] Zhang, S., Choromanska, A., and LeCun, Y. (2015).
Deep learning with Elastic Averaging SGD.
Neural Information Processing Systems Conference (NIPS 2015), pages 1–24.