НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Stochastic optimization

Stochastic optimization methods overview and their application in large-scale problems

Сон Андрей, БПМИ142

abson@edu.hse.ru

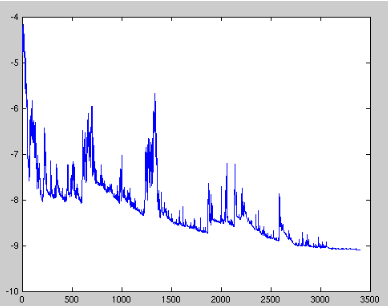
Национальный исследовательский университет
«Высшая школа экономики»

14 марта 2017 г.

Gradient descent variants

1. (Batch) gradient descent — $\theta_{i+1} = \theta_i - \eta \cdot \nabla_\theta J(\theta)$
2. Stochastic («online») gradient descent — $\theta_{i+1} = \theta_i - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}, y^{(i)}\right)$
3. Mini-batch gradient descent — $\theta_{i+1} = \theta_i - \eta \cdot \nabla_\theta J(\theta; x^{(i;i+n)}; y^{(i;i+n)})$

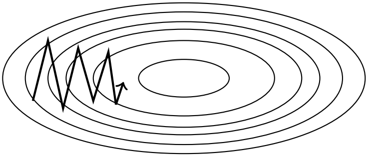
Pros/cons and challenges of SGD



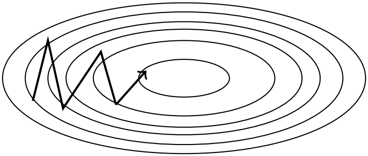
- fast speed (esp. with sparse data) and posibility of «online» use
- high variance causes heavy fluctuation
- proper choice of learning rate
- non-convex to optimization

Momentum(heavy-ball) method

David E. Rumelhart, Geoffrey E. Hinton, & Ronald J. Williams, (1986), doi:10.1038/323533a0



SGD without momentum



SGD with momentum

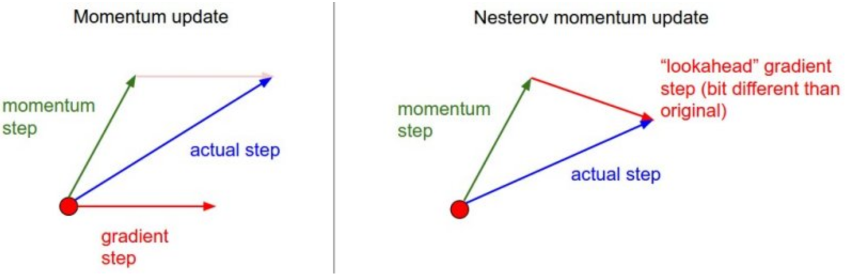
$$\begin{cases} \upsilon_t = \gamma \upsilon_{t-1} + \eta \cdot \nabla_\theta J(\theta) \\ \theta_{t+1} = \theta_t - \upsilon_t \end{cases}$$

γ – *momentum term*, usually set to 0.9 or a similar value (0.995, 0.999) ($\gamma < 1$).

Nesterov accelerated gradient (AGM)

Nesterov, Y. (1983), A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. Doklady ANSSSR

$$\begin{cases} v_t = \gamma v_{t-1} + \eta \cdot \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta_{t+1} = \theta_t - v_t \end{cases}$$



Theorem (Nesterov)

$$f(x_k) - f(x^*) \leq \frac{2L||x_1 - x^*||^2}{k^2}$$

[1] Convergence rate $O(1/k^2)$ is optimal for first-order methods, link

Mathematical formulation and intuition

- Ilya Sutskever, (2013), Training Reccurent Neural Networks, Ph.D. Thesis, link
- Yoshua Bengio, Nicolas Boulanger-Lewandowski and Razvan Pascanu, (2012), Advances in optimizing recurrent networks, arXiv:1212.0901v2

Reducing constant factor

- Optimized gradient method (OGM) – D. Kim, J. A. Fessler, Optimized first-order methods for smooth convex minimization, (2016), doi:10.1007/s10107-015-0949-3

[1] Nesterov lecture

Nesterov accelerated gradient (AGM)

Application/Becnhmarks

task	0(SGD)	0.9N	0.99N	0.995N	0.999N	0.9M	0.99M	0.995M	0.999M	SGDC	HF†	HF*
Curves	0.48	0.16	0.096	0.091	**0.074**	0.15	0.10	0.10	0.10	0.16	0.058	0.11
Mnist	2.1	1.0	**0.73**	0.75	0.80	1.0	0.77	0.84	0.90	0.9	0.69	1.40
Faces	36.4	14.2	8.5	7.8	**7.7**	15.3	8.7	8.3	9.3	NA	7.5	12.0

problem	biases	0	0.9N	0.98N	0.995N	0.9M	0.98M	0.995M
add $T = 80$	0.82	0.39	0.02	0.21	**0.00025**	0.43	0.62	0.036
mul $T = 80$	0.84	0.48	0.36	0.22	**0.0013**	0.029	0.025	0.37
mem-5 $T = 200$	2.5	1.27	1.02	0.96	**0.63**	1.12	1.09	0.92
mem-20 $T = 80$	8.0	5.37	2.77	0.0144	**0.00005**	1.75	0.0017	0.053

Learning rate adaptation

- *Step decay* — reduce the learning rate by some factor every few epochs
- *Bold driver* — if the error decreased, increase α by small proportion (1-5%), otherwise descrease sharply (typcially by 50%)
- *Exponential decay* — $\alpha_k = \alpha_0 e^{-\beta k}$, α_0, β – hyperparametres
- *Annealing schedule* — $\alpha_k = \frac{\alpha_0}{1+\beta k}$, α_0, β – hyperparametres

AdaGrad (adaptive gradient algorithm)

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, link

Per-parameter update:

$$g_{k,i} = \nabla_\theta J(\theta_i)$$

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta}{\sqrt{G_{k,ii} + \epsilon}} \cdot g_{k,i},$$

$G \in \mathbb{R}^{d \times d}$ — diag. matrix of sum of squares $\nabla_\theta J(\theta_i)$ up to k step

Vectorized implementation:

$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_k$$

Pros/cons and challenges

- Well-suited for dealing with sparse data
- No need to manually tune the learning rate. Most implementations use a default value of 0.01.
- Accumulation of the squared gradients in the denominator causes the learning rate to shrink

AdaGrad (adaptive gradient algorithm)

Application/Benchmarks

Since it is well-suited for dealing with sparse data, the most frequest application include NLP and image recognition

- Recognize cats in Youtube videos, wired article
- GloVe word embeddings — Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation, doi:10.3115[2]
-

	SGD	MOMENTUM	ADAGRAD
$\epsilon = 1e^{0}$	**2.26%**	89.68%	43.76%
$\epsilon = 1e^{-1}$	2.51%	**2.03%**	2.82%
$\epsilon = 1e^{-2}$	7.02%	2.68%	**1.79%**
$\epsilon = 1e^{-3}$	17.01%	6.98%	5.21%
$\epsilon = 1e^{-4}$	58.10%	16.98%	12.59%

[2] GloVe overwiew in comparison with w2v — link

Resticting evaluating sum of squared gradient to fixed size *n* and compute it with exponentially weighted moving average.

$$
\begin{aligned}
E[g^2]_k &= \gamma E[g^2]_{k-1} + (1-\gamma) g_t^2 \\
\theta_{k+1} &= \theta_k - \frac{\eta}{\sqrt{E[g^2]_k + \epsilon}} g_k = \theta_k - \frac{\eta}{RMS[g]_k} g_k
\end{aligned}
$$

AdaDelta

Pros/cons and challenges

- no need to set a default learning rate:

$$\Delta\theta_k = -\frac{\eta}{RMS[g]_k} g_k$$

$$E[\Delta\theta^2]_k = \gamma E[\Delta\theta^2]_{k-1} + (1-\gamma)\Delta\theta_k^2$$

$$RMS[\Delta\theta]_k = \sqrt{E[\Delta\theta^2]_k + \epsilon}$$

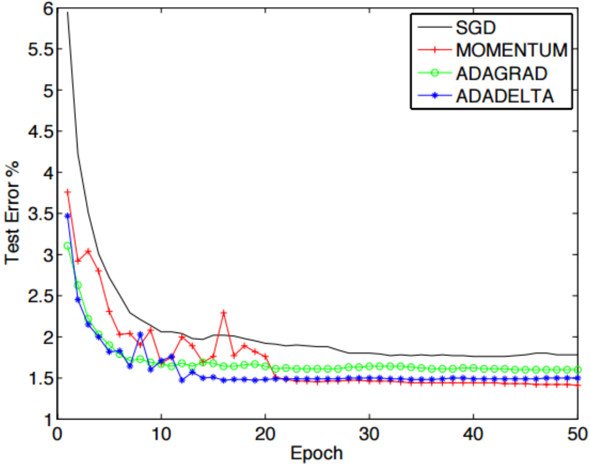
$$\Delta\theta_k = -\frac{RMS[\Delta\theta]_{k-1}}{RMS[g]_k} g_k$$

$\theta_{k+1} = \theta_k + \Delta\theta_k$

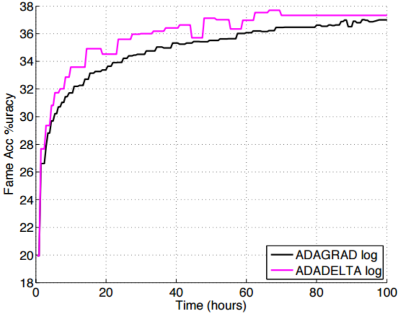
- Combining with Nesterov momentum (Ilya Sutskever, 2012)

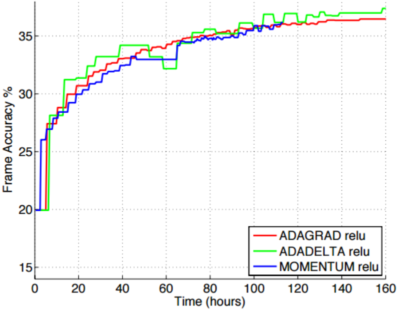
AdaDelta

Application/Benchmarks



	$\rho = 0.9$	$\rho = 0.95$	$\rho = 0.99$
$\epsilon = 1e^{-2}$	2.59%	2.58%	2.32%
$\epsilon = 1e^{-4}$	2.05%	1.99%	2.28%
$\epsilon = 1e^{-6}$	1.90%	**1.83%**	2.05%
$\epsilon = 1e^{-8}$	2.29%	2.13%	2.00%





RMSprop (Root Mean Square Propagation)

Geoff Hinton, Coursera class Lecture 6c (Unpublished), link

Technically, it's AdaDelta with parametres $\gamma = 0.9, \eta = 0.001$

$$E[g^2]_k = 0.9E[g^2]_{k-1} + 0.1g_k^2$$
$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Adam (Adaptive Moment Estimation)

Kingma, D. P., & Ba, J. L. (2014) Adam: A Method for Stochastic Optimization, arXiv:1412.6980

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k,\ \hat{m}_k = \frac{m_t}{1-\beta_1^k}$$

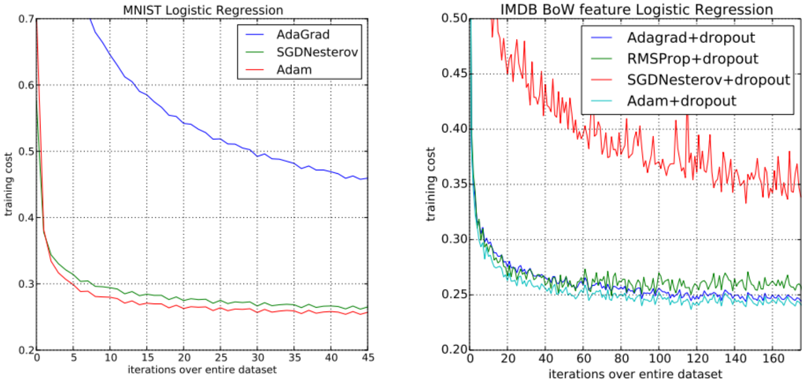
$$\upsilon_k = \beta_2 \upsilon_{k-1} + (1-\beta_2) g_k^2,\ \hat{\upsilon}_k = \frac{\upsilon_k}{1-\beta_2^k}$$

$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{\hat{\upsilon}_k + \epsilon}} \hat{m}_k$$

Proposed hyperparametres — $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$

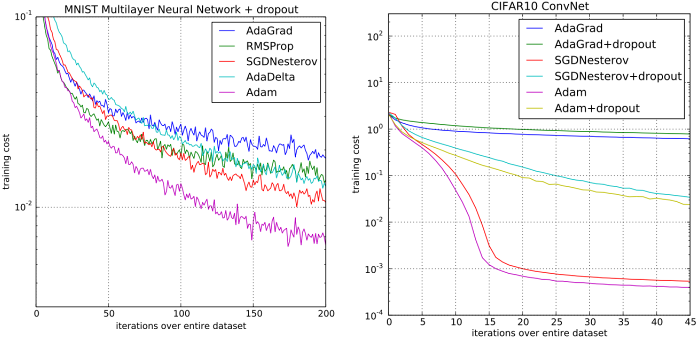
Adam (Adaptive Moment Estimation)

Application/Benchmarks



Adam (Adaptive Moment Estimation)

Application/Benchmarks

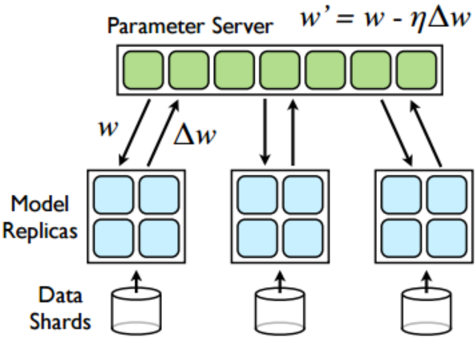


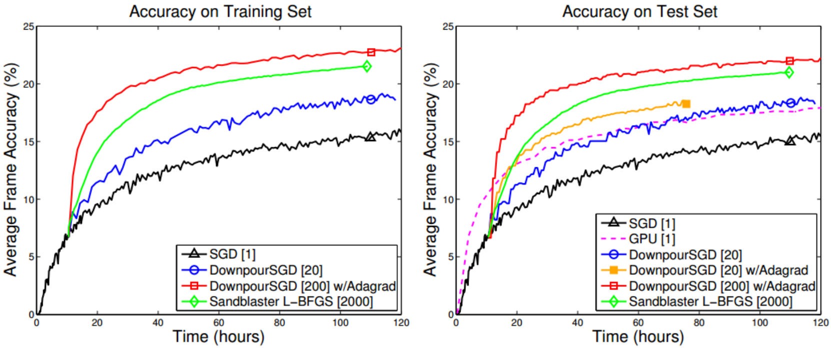
Visualization of algorithms

SGD optimization on loss surface contours
SGD optimization on saddle point

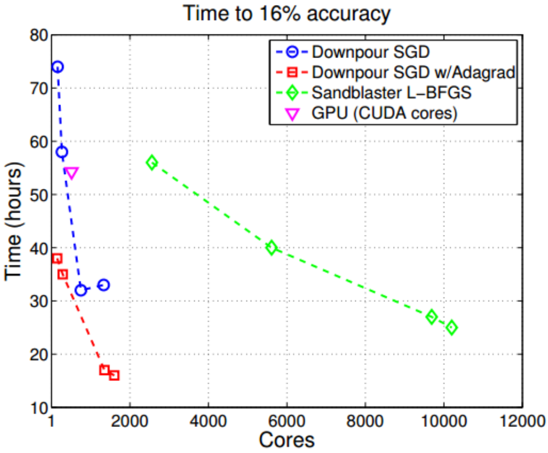
Parallelizing and distributing SGD

- Hogwild! – Feng Niu, Benjamin Recht, Christopher Re, Stephen J. Wright. (2011). HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent arXiv:1106.5730
- DistBelief(Downpour SGD)/Tensorflow – NIPS2012, Google Research Blog





Accuracy on Training Set
Average Frame Accuracy (%)
Time (hours)
SGD [1]
DownpourSGD [20]
DownpourSGD [200] w/Adagrad
Sandblaster L-BFGS [2000]
Accuracy on Test Set
Average Frame Accuracy (%)
Time (hours)
SGD [1]
GPU [1]
DownpourSGD [20]
DownpourSGD [20] w/Adagrad
DownpourSGD [200] w/Adagrad
Sandblaster L-BFGS [2000]


Time to 16% accuracy
Downpour SGD
Downpour SGD w/Adagrad
Sandblaster L–BFGS
GPU (CUDA cores)
Time (hours)
80
70
60
50
40
30
20
10
1
2000
4000
6000
8000
10000
12000
Cores

Elastic Averaging SGD (EASGD)

Yann LeCun. (2015) Deep learning with Elastic Averaging SGD, arXiv:1412.6651

Idea: inventing auxiliary «center variables» for «exploration»

$$\theta_{i,k+1} = \theta_k^i - \eta\left(\nabla_\theta J(\theta; x^{(i)}) + \rho(x_{i,k+1} - \widetilde{x}_k)\right)$$

$$\widetilde{x}_{k+1} = \widetilde{x}_t + \eta \sum_{i=1}^{p} \rho\left(x_k^{(i)} - \widetilde{x}_k\right)$$

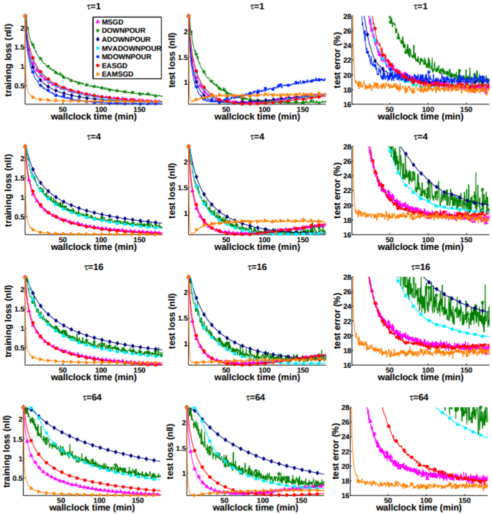
$\alpha = \eta\rho, \quad \beta = p\alpha$

$$\theta_{i,k+1} = \theta_k^i - \eta\nabla_\theta J(\theta; x^{(i)}) - \alpha(x_{i,k+1} - \widetilde{x}_k)$$

$$\widetilde{x}_{k+1} = (1-\beta)\widetilde{x}_t + \beta\left(\frac{1}{p}\sum_{i=1}^{p}\widetilde{x}_k^{(i)}\right)$$

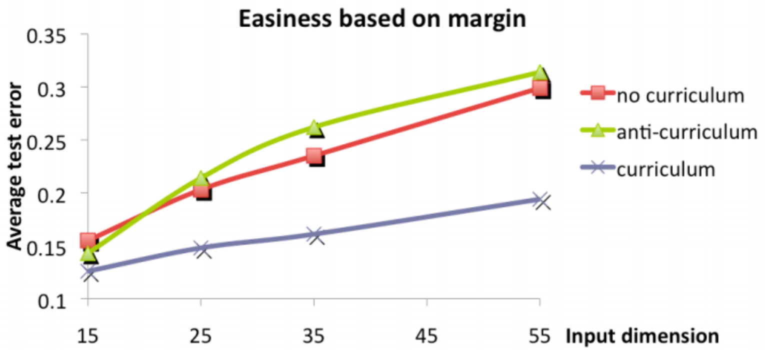
Elastic Averaging SGD (EASGD)

Application/Benchmarks

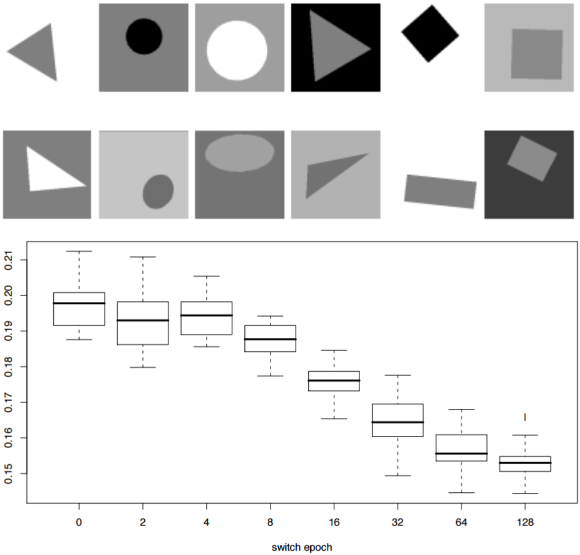


Aditional possible tweaks

- Shuffling and Curriculum Learning — Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning, doi:10.1145/1553374.1553380, link
  Idea: Sort examples by «difficulty», not shuffling

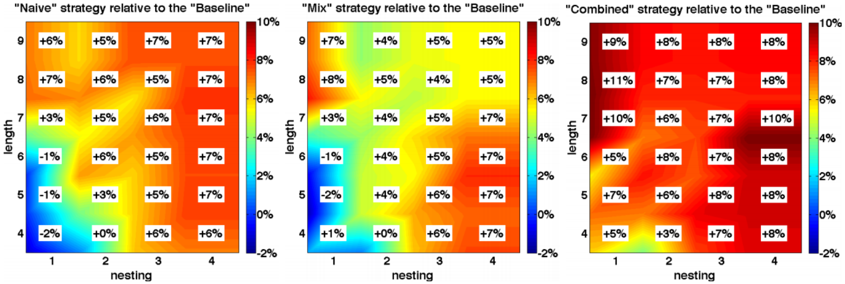


Curriculum learning



Curriculum learning

Wojciech Zaremba, Ilya Sutskever. (2014) Learning to Execute, arXiv:1410.4615



Aditional possible tweaks

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

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

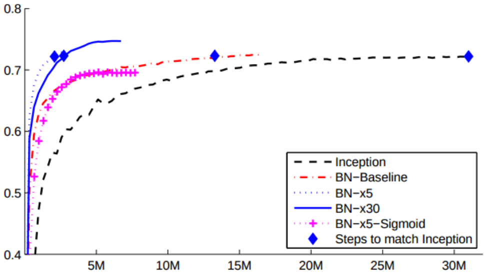
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Accelerating BN Networks

- Increase learning rate
- Remove Dropout
- Accelerate the laerning rate decay
- Thorough shuffling of training examples

Batch normalization

Application/Benchmarks



Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Aditional possible tweaks

Gradient noise, Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., & Martens, J. (2015). Adding Gradient Noise Improves Learning for Very Deep Networks arXiv:1511.06807

$$g_{k,i} = g_{k,i} + \mathcal{N}(0, \sigma_k^2)$$

$$\sigma_k^2 = \frac{\eta}{(1+k)^\gamma}$$

Setting	Best Test Accuracy	Average Test Accuracy
No Noise	89.9%	43.1%
With Noise	96.7%	52.7%
No Noise + Dropout	11.3%	10.8%

Experiment 2: Simple Init, Gradient Clipping Threshold = 100

No Noise	90.0%	46.3%
With Noise	96.7%	52.3%

Experiment 3: Simple Init, Gradient Clipping Threshold = 10

No Noise	95.7%	51.6%
With Noise	97.0%	53.6%

Experiment 4: Good Init (Sussillo & Abbott, 2014) + Gradient Clipping Threshold = 10

No Noise	97.4%	92.1%
With Noise	97.5%	92.2%

Experiment 5: Good Init (He et al., 2015) + Gradient Clipping Threshold = 10

No Noise	97.4%	91.7%
With Noise	97.2%	91.7%

Experiment 6: Bad Init (Zero Init) + Gradient Clipping Threshold = 10

No Noise	11.4%	10.1%
With Noise	94.5%	49.7%

Resources

References

1. S. Ruder, (2016), An overview of gradient descent optimization algoritms, arXiv:1609.04747v1 or blog post

For detailed information

1. L´eon Bottou, Frank E. Curtis†, Jorge Noceda, (2016) Optimization Methods for Large-Scale Machine Learning, arXiv:1606.04838v1
2. Optimization I lecture (Youtube)

Спасибо за внимание!

Stochastic optimization

Сон Андрей, БПМИ142

abson@edu.hse.ru

Национальный исследовательский университет
«Высшая школа экономики»

14 марта 2017 г.