An Introduction to Optimization Methods in Deep Learning

Yuan YAO
HKUST

Acknowledgement

- Feifei Li, Stanford cs231n
- Ruder, Sebastian (2016). An overview of gradient descent optimization algorithms. arXiv:1609.04747.
  - http://ruder.io/deep-learning-optimization-2017/

Image Classification

Example Dataset: **CIFAR10**

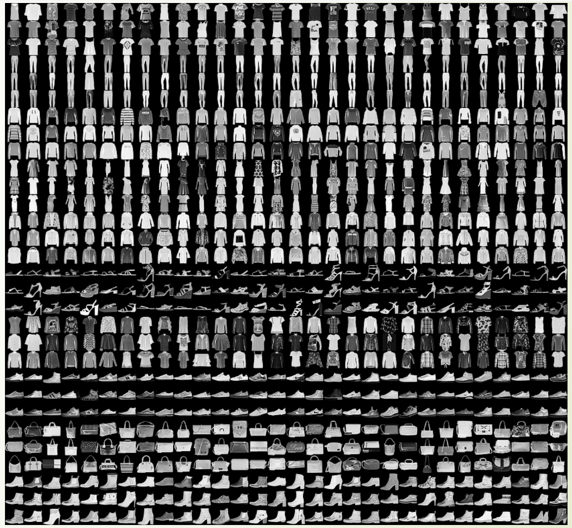
10 classes
50,000 training images
10,000 testing images

Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Example Dataset: **Fashion MNIST**

28x28 grayscale images
60,000 training and 10,000 test examples
10 classes

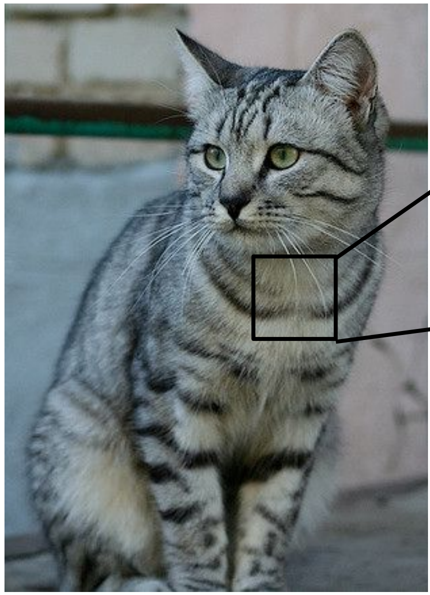
index	0	1	2	3	4	5	6	7	8	9
Type	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot



Jason WU, Peng XU, and Nayeon LEE

The Challenge of Human-Instructing-Computers

The Problem: Semantic Gap



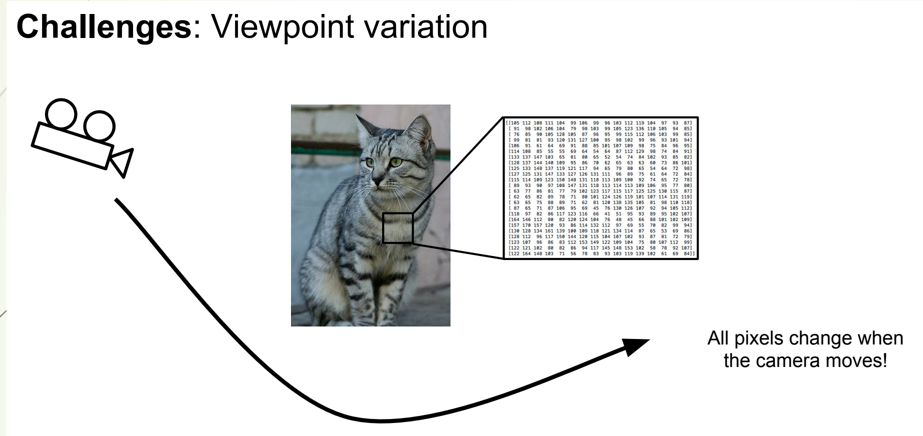

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Complex Invariance



Euclidean transform

Large scale deformation

Complex Invariance

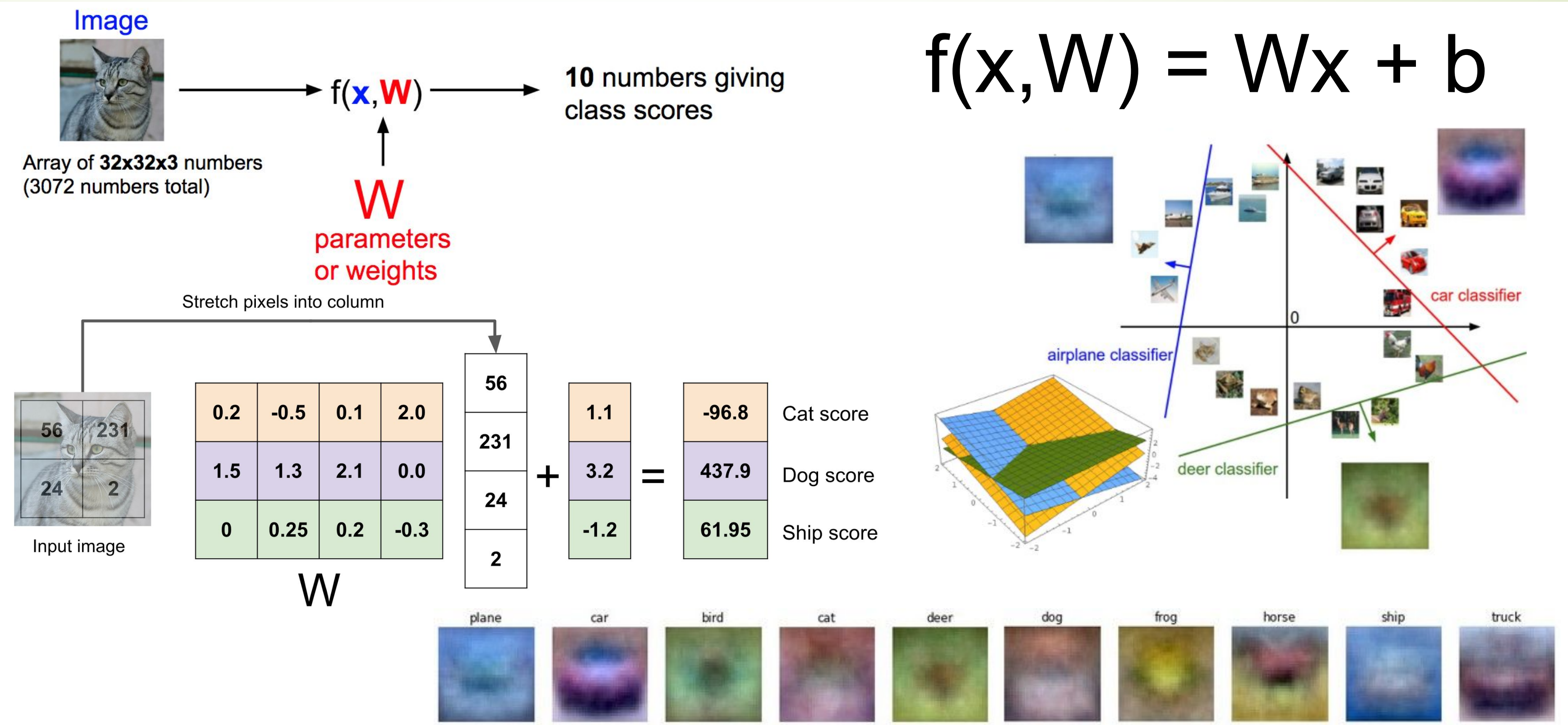
Challenges: Illumination

Challenges: Background Clutter

Challenges: Occlusion

Challenges: Intraclass variation

Data Driven Learning of the invariants: linear discriminant/classification



(Empirical) Loss or Risk Function

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

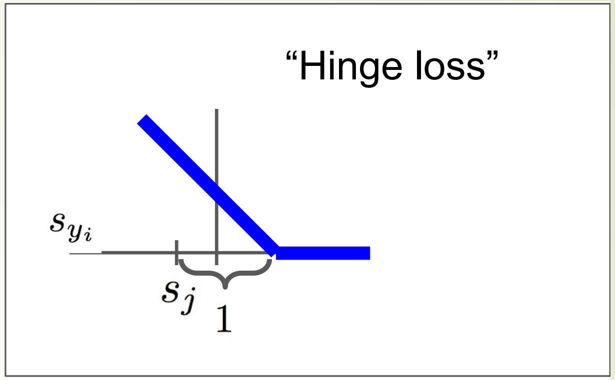
$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Hing Loss



Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

			
cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Cross Entropy (Negative Log-likelihood) Loss

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities	
cat	**3.2**		**24.5**		**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

unnormalized log probabilities

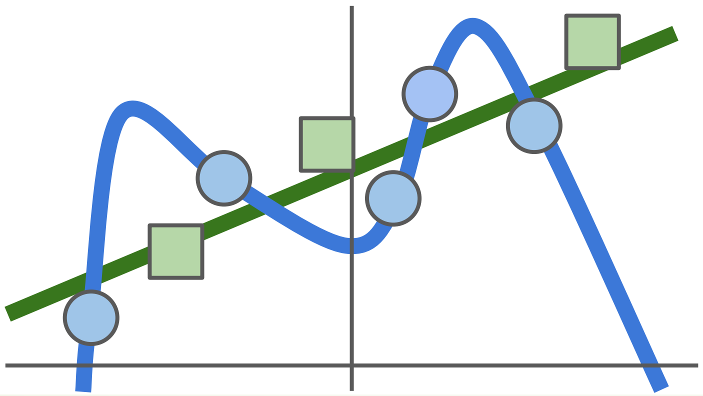
probabilities

Loss + Regularization

$$\underbrace{L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

Regularizations

- Explicit regularization
  - L2-regularization
  - L1-regularization (Lasso)
  - Elastic-net (L1+L2)
  - Max-norm regularization
- Implicit regularization
  - Dropout
  - Batch-normalization
  - Earlystopping

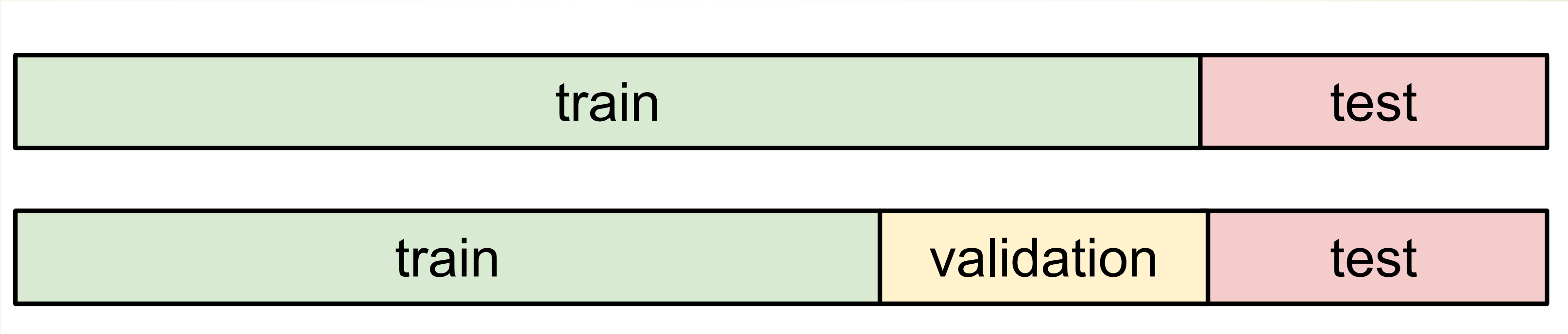
$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

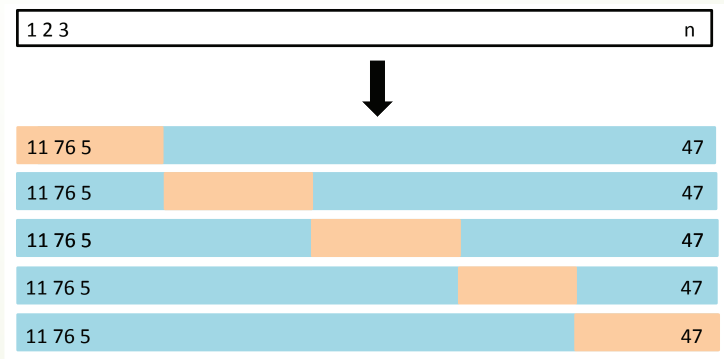
$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

Hyperparameter (Regularization) Tuning

Data rich:



Data poverty: cross-validation



Recap

How do we find the best W?

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

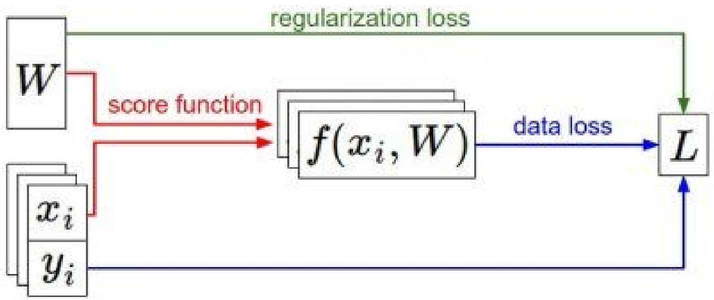
Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss



In regression, square loss is often used instead.

Optimization Methods to find minima of the Loss Landscape?

Gradient Descent Method

- 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: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

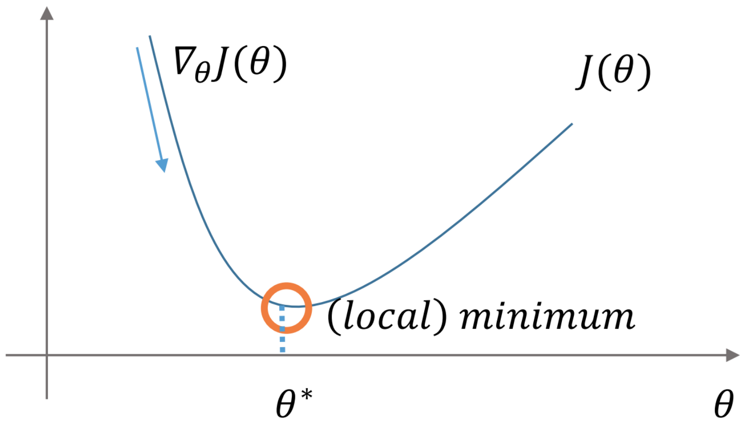


Figure: Optimization with gradient descent

Gradient Descent Variants

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

- Difference: how much data we use in computing the *gradients*

Batch Gradient Descent

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

```
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** objective function and to a **stationary/critical** point for **non-convex** ones.
  - Exponentially fast (linear) convergence rates in **strongly convex** landscape
  - Sublinear convergence rates in **convex** landscape
- Cons:
  - Slow in big data.
  - Intractable for big datasets that do **not fit in memory**.
  - No **online** learning.

Stochastic Gradient Descent

- Computes update for each example $(x^{(i)}, y^{(i)})$, usually uniformly sampled from the training dataset
- Update equation:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

- The expectation of stochastic gradient is the batch gradient

```
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:
  - Guaranteed to converge to **global** minimum for **convex** losses and to a local optima for **non-convex** ones, may **escape saddle** points polynomially fast
  - O(1/k) convergence rates in convex losses, possibly dimension-free
  - Much faster than batch in big data
  - Online learning algorithms
- Cons:
  - High variance in gradients and outcomes

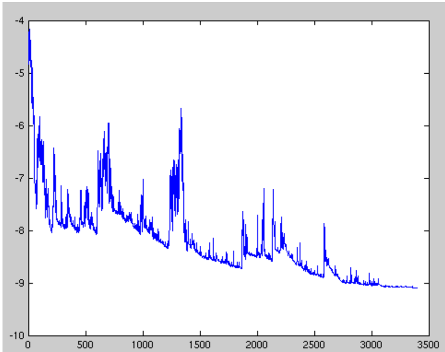


Figure: SGD fluctuation (Source: Wikipedia)

Batch GD vs. Stochastic GD

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

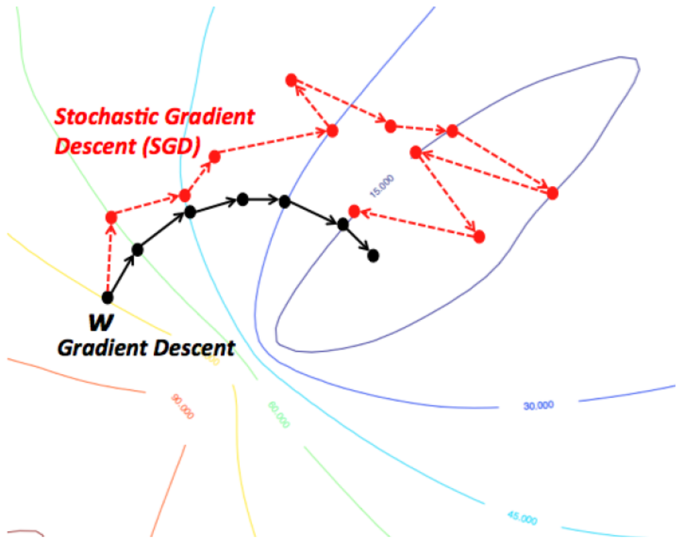


Figure: Batch gradient descent vs. SGD fluctuation (Source: wikidocs.net)

Mini-batch Gradient Descent

- Performs update for every **mini-batch** of random n examples.
- Update equation:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

- The expectation of gradient is the same as the batch gradient

```
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** in deep learning 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 (learning rate decay) schedule.
- Escaping saddles and suboptimal minima.

Variants of Gradient Descent Algorithms

- Momentum
- Nesterov accelerated gradient
- Adagrad
- Adadelta
- RMSprop
- Adam
- 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**.

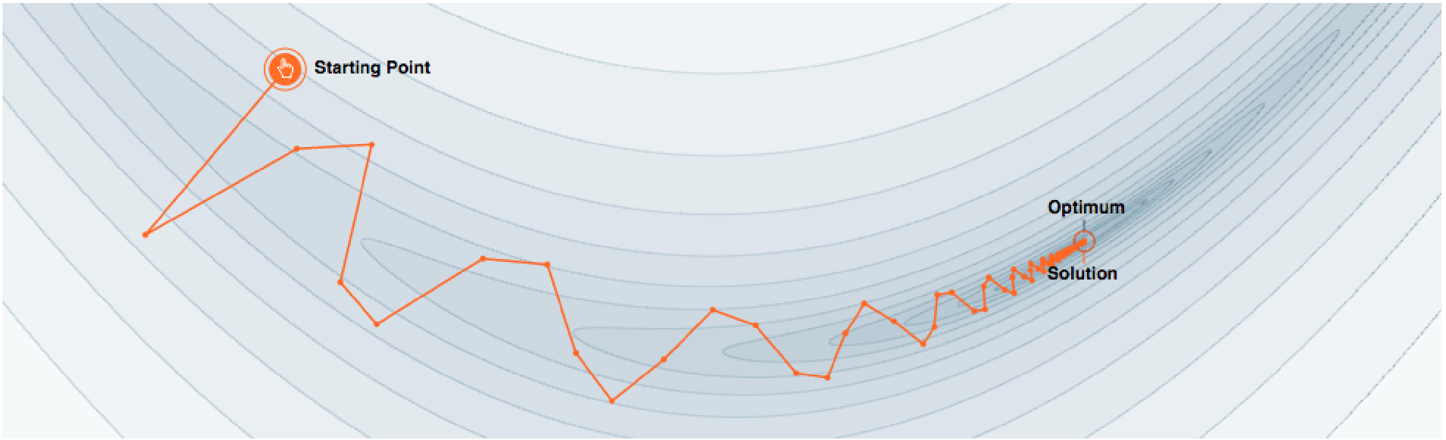


Figure: Optimization with momentum (Source: distill.pub)

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.

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

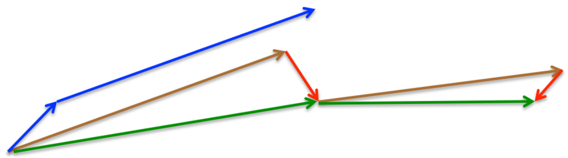


Figure: Nesterov update (Source: G. Hinton's lecture 6c)

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:

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

- Adam update rule:

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

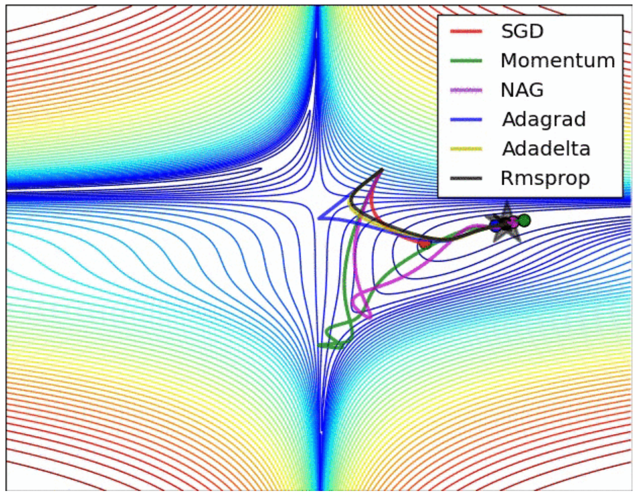
Adam Extensions

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

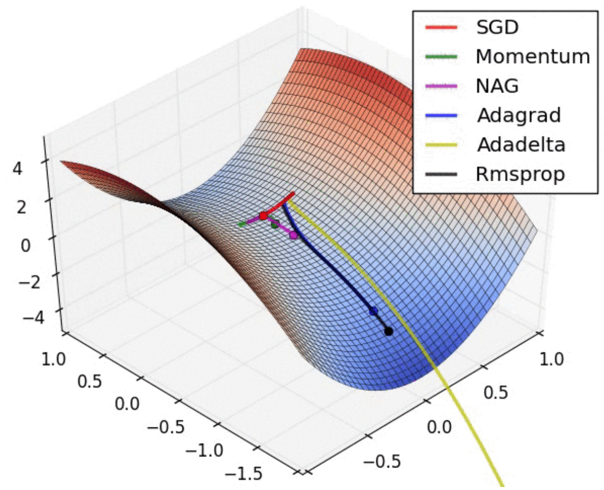
Update Equations

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	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

Visualization of algorithms



(a) SGD optimization on loss surface contours



(b) SGD optimization on saddle point

Figure: Source and full animations: Alec Radford

Comparisons

- 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.

Parallel and Distributed SGD

- **Hogwild!** [**Niu et al., 2011**]
  - Parallel SGD updates on CPU
  - Shared memory access without parameter lock Only works for sparse input data
- 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
- Delay-tolerant Algorithms for SGD [**Mcmahan and Streeter, 2014**]
  - Methods also adapt to update delays
- TensorFlow [**Abadi et al., 2015**]
  - Computation graph is split into a subgraph for every device
  - Communication takes place using Send/Receive node pairs
- Elastic Averaging SGD [**Zhang et al., 2015**]
  - Links parameters elastically to a center variable stored by parameter server

Additional Strategies for SGD

- 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
- Batch normalization [**Ioffe and Szegedy, 2015**]
  - Re-normalizes every mini-batch to zero mean, unit variance
  - Must-use for computer vision
- Early stopping
  - "*Early stopping (is) beautiful free lunch*" (**Geoff Hinton**)
- Gradient noise [**Neelakantan et al., 2015**]
  - Add Gaussian noise to gradient
  - Makes model more robust to poor initializations
  - Escape saddles or local optima

Adam vs. Tuned SGD

- 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 oscillates and may underperform SGD on some tasks, e.g. Machine Translation [Wu et al., 2016].
- Adam with 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].

Reference

- [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.
- [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).

Reference

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

Reference

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

Thank you!