Optimization for Training Deep Models

Vishnu Lokhande

Department of Computer Science and Engineering
University at Buffalo, SUNY
vishnulo@buffalo.edu

March 3, 2025

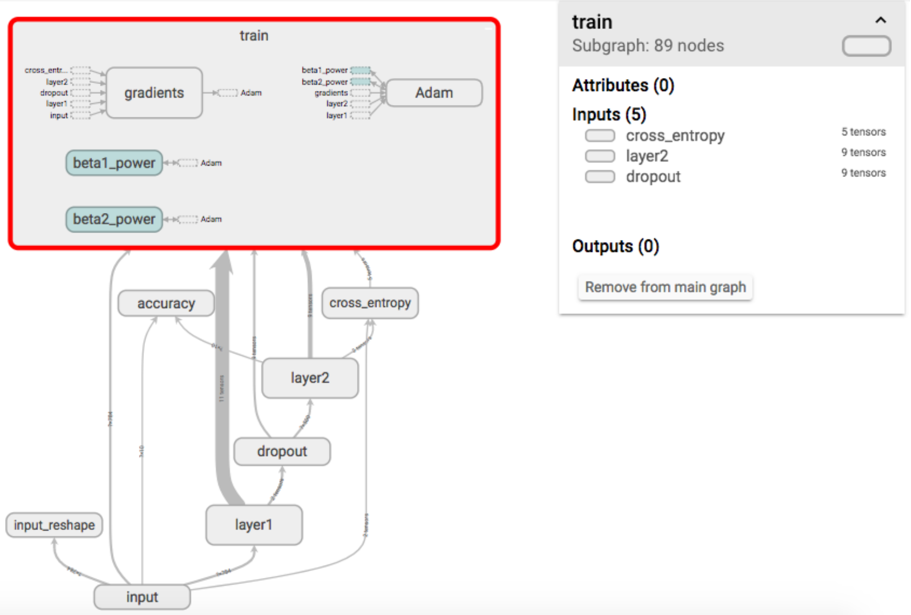
[1]

[1] Adapted and modified from https://project.inria.fr/deeplearning/files/2016/05/optimization-training-deep.pdf

Implementation of the 2 Attempts in Quiz

- Each attempt lasts for 3 mins, if not specified.
- In attempt 1, you can answer as many questions as you want.
- If you are not able to finish all the questions in attempt 1, you can continue submitting in attempt 2.

Optimizer: A Special Node in a Computational Graph



train

Subgraph: 89 nodes

Attributes (0)

Inputs (5)

cross_entropy	5 tensors
layer2	9 tensors
dropout	9 tensors

Outputs (0)

Outline

- Difference between learning and pure optimization
- Challenges in network optimization
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rate

Outline

- Difference between learning and pure optimization
- Challenges in network optimization
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rate

ML Training vs. Pure Optimization

1. ML acts indirectly:
   - Care about some performance measure $\mathcal{P}$ which may be intractable, *e.g.*, expected loss w.r.t. intractable data distribution.
   - Reduce to a different cost function $J(\boldsymbol{\theta})$ in the hope that doing so will reduce $\mathcal{P}$.
2. Pure optimization:
   - Minimizing J is a goal in itself.
3. Optimizing Deep models: specialization on structure of ML objective function.

Generalization Error (Loss Function)

- Typically the loss function is a simple average over the training set:

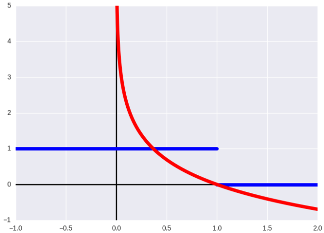
$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{\text{data}}} L(f(\mathbf{x};\boldsymbol{\theta}), y) = \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), y^{(i)}) \ ,$$

  - L: the per-example loss function
  - $f(\mathbf{x};\boldsymbol{\theta})$: predicted output for input $\mathbf{x}$
  - $\hat{p}_{\text{data}}$: empirical distribution
  - y: target output
  - N: number of training examples

Surrogate Loss[2]

An alternative loss that is smooth and easier than the original loss.

1. Often, minimizing the real loss is intractable, *e.g.*, 0-1 loss.
2. Minimizing a surrogate loss instead, *e.g.*, the negative log-likelihood is a surrogate for the 0-1 loss.
3. Sometimes, the surrogate loss may learn more:
   - Test error 0-1 loss keeps decreasing even after training 0-1 loss is zero.
   - Even if the training 0-1 loss is zero, it can be improved by pushing the classes even farther from each other.



[2] See https://arxiv.org/pdf/1802.03688 and https://towardsdatascience.com/reasons-why-surrogate-loss-functions-are-pivotal-for-classification-in-machine-learning-e33974ce6d29 for more details.

Batch and Minibatch Algorithms

1. In machine learning, the objective function typically decomposes as a sum, each summand relates to one data point.
2. Full gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x};\boldsymbol{\theta}), y)$$

3. Computing the expectation exactly at each update is very expensive (batch algorithm).
4. Use randomly sampled minibatches instead (minnibatch algorithm):
   - unbiased estimate
   - computational efficient
   - called stochastic algorithms

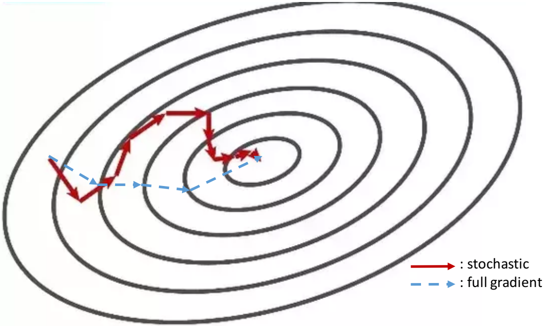
$$\nabla_{\boldsymbol{\theta}} \tilde{J}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$$

Stochastic Gradient Descent (SGD)

$$\nabla_{\boldsymbol{\theta}} \tilde{J}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \triangleq \mathbf{g}_t$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \epsilon_t \, \mathbf{g}_t$$

Eventually ends up a (local) optima.

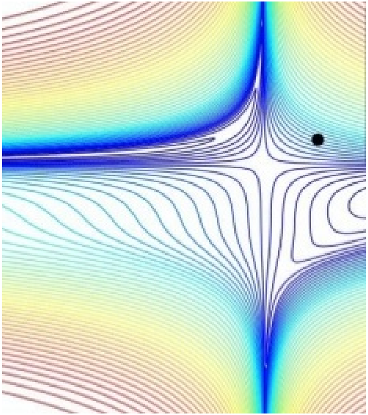


Outline

- Difference between learning and pure optimization
- Challenges in network optimization
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rate

III Conditioning

1. Condition number: how a change in input affects output.
2. High value greatly propagate errors, may cause the gradient to become “stuck”.



When ill Conditioning is a Problem

1. Let $f(\mathbf{x})$ be the loss. According to Taylor expansion:

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^T \mathbf{g} + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(0)})^T \mathbf{H}(\mathbf{x} - \mathbf{x}^{(0)})$$

2. An SGD update with learning rate ϵ changes the objective function to an amount of:

$$f(\mathbf{x}) - f(\mathbf{x}_0) = f(\mathbf{x}_0 - \epsilon\,\mathbf{g}) - f(\mathbf{x}_0) \approx \frac{1}{2}\epsilon^2\,\mathbf{g}^T\,\mathbf{H}\,\mathbf{g} - \epsilon\,\mathbf{g}^T\,\mathbf{g}$$

3. When **H** is ill-conditioned such that $\frac{1}{2}\epsilon^2\,\mathbf{g}^T\,\mathbf{H}\,\mathbf{g} \geq \epsilon\,\mathbf{g}^T\,\mathbf{g}$, the objective function does not decrease with SGD update.

Local Minima

1. In convex problems, any local minimum is a global minimum.
2. Non-convex functions may have several local minima:
   - Neural networks are non-convex.
   - However, does not seem a major problem in practice.
3. Local minima is still an open problem.
4. But nowadays researchers believe most local minima have low cost objective values (loss) [Choromanska et al., 2015, Dauphin et al., 2014, Goodfellow et al., 2015, Saxe et al., 2013], or even that all local minima are global minima [Kawaguchi, NIPS 2016].

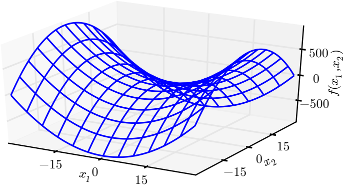
Model Identifiability

1. A model is *identifiable* if a large training set yields a unique set of parameters, *e.g.*, one-to-one mapping.
2. Models with latent variable are often not identifiable.
3. Neural nets are not identifiable.
    - *e.g.*, the input and output weights of any ReLU or maxout units may be scaled to produce the same result.
4. Require appropriate initialization.

Saddle Points

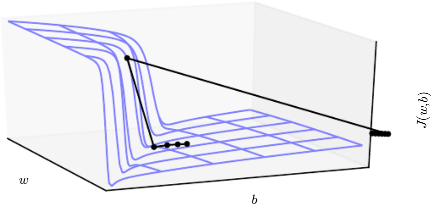
1. Saddle point appears to be a more sever problem since its function value is typically high.
2. The chance of containing saddle points in high dimensional space is exponentially high.
3. For first-order methods such as SGD, intuitively it seems to be problematic:
   - But studies show it is not[a].

[a] https://arxiv.org/abs/1902.00247



Cliffs and Exploding Gradients

1. Very steep cliffs cause gradients to explode:
   - Go to far away.
2. More often in recurrent networks.
3. Need algorithms with adaptive stepsizes.



Outline

- Difference between learning and pure optimization
- Challenges in network optimization
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rate

Recap: Stochastic Gradient Descent

1. Minibatch of training samples $\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)}\}$ with targets $\{y^{(i)}\}$.
2. At each iteration, computer stochastic gradient:

$$\bar{\mathbf{g}} = \frac{1}{n}\nabla_{\boldsymbol{\theta}}\left(\sum_{i=1}^{n} L(f(\mathbf{x}^{(i)}, y^{(i)}))\right)$$

3. Apply update:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon_k \bar{\mathbf{g}}$$

Learning Rate for SGD

1. It is common to decay the learning rate linearly until iteration τ.

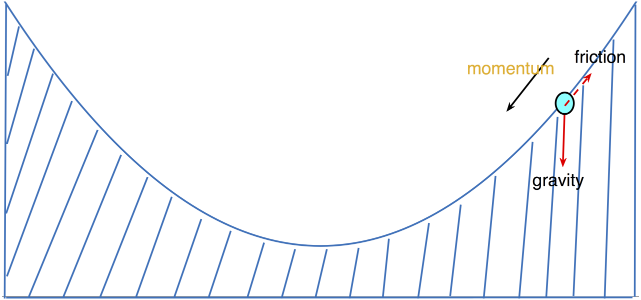
$$\epsilon_k = \left(1 - \frac{k}{\tau}\right)\epsilon_0 + \frac{k}{\tau}\epsilon_\tau$$

2. τ should allow a few hundred passes through the training set.
3. ϵ_τ should be roughly 1% of ϵ_0.
4. After τ, it is common to leave ϵ constant.
5. Also have many other learning-rate-decay schemes.

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

SGD with Momentum

1. Momentum aims primarily to solve two problems: 1) poor conditioning of the Hessian; and 2) variance in stochastic gradient.
2. Consider a ball rolling on a friction surface, with location θ and momentum $\mathbf{p} = m\mathbf{v}$, where we assume the mass $m = 1$, leaving only the velocity $\mathbf{v}$.



SGD with Momentum

1. Momentum accumulates previous gradients:

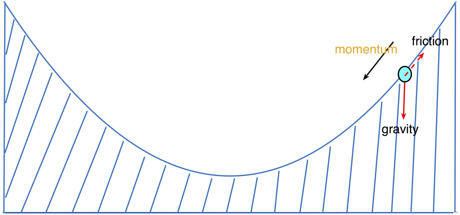
$$\mathbf{v} = \underbrace{\mathbf{v}}_{\text{old velocity}} - \underbrace{(1-\alpha)\,\mathbf{v}}_{\text{friction}} - \underbrace{\epsilon\nabla_{\boldsymbol{\theta}}\left(\sum_{i=1}^{n} L(f(\mathbf{x}^{(i)}, y^{(i)}))\right)}_{\text{gravity}=\bar{\mathbf{g}}}$$

$$= \alpha\,\mathbf{v} - \epsilon\bar{\mathbf{g}}(\boldsymbol{\theta})$$

2. Update location with velocity:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{v}$$

- $\alpha \in [0, 1)$: momentum coefficient
- with enough **v**, the particle can move out of local modes



SGD with Momentum

1. Momentum accumulates previous gradients:

$$\mathbf{v} = \underbrace{\mathbf{v}}_{\text{old velocity}} - \underbrace{(1-\alpha)\,\mathbf{v}}_{\text{friction}} - \underbrace{\epsilon\nabla_{\boldsymbol{\theta}}\left(\sum_{i=1}^{n} L(f(\mathbf{x}^{(i)}, y^{(i)}))\right)}_{\text{gravity}=\bar{\mathbf{g}}}$$

$$= \alpha\,\mathbf{v} - \epsilon\bar{\mathbf{g}}(\boldsymbol{\theta})$$

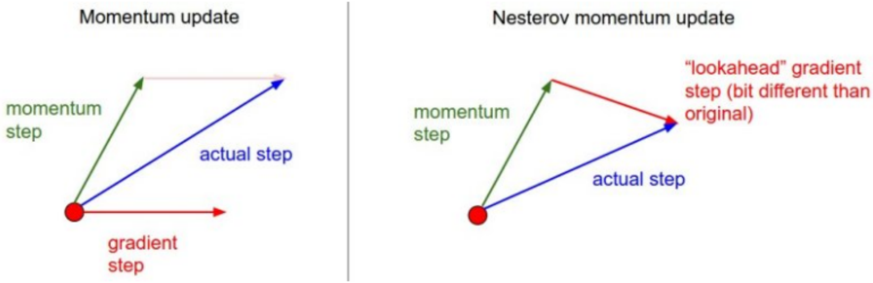
2. Update location with velocity:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{v}$$

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Nesterov's Accelerated Momentum (NAG)

- A slightly different version of the momentum update.
  - Enjoys stronger theoretical converge guarantees for convex functions.
  - Consistently works slightly better than standard momentum in practice.

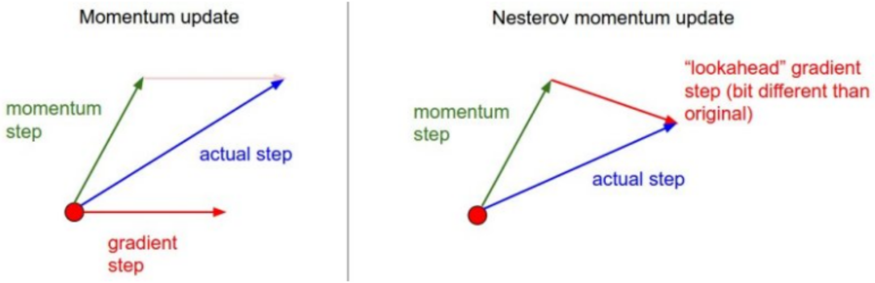


Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

http://ruder.io/optimizing-gradient-descent/

Nesterov's Accelerated Momentum (NAG)

$$\mathbf{v} = \alpha\,\mathbf{v} - \epsilon\bar{\mathbf{g}}(\boldsymbol{\theta} + \alpha\,\mathbf{v})$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{v}$$



Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

http://ruder.io/optimizing-gradient-descent/

Outline

- Difference between learning and pure optimization
- Challenges in network optimization
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rate

Parameter Initialization

1. Deep learning methods are strongly affected by initialization.
2. It can determine if the algorithm converges at all.
3. Network optimization is still not well understood.
4. Modern techniques are simple and heuristic.

Breaking symmetry

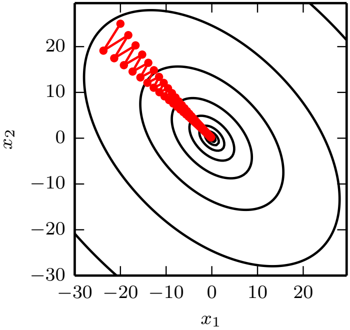
1. Initialization must break symmetry between units:
   - Units with same inputs and activations will be updated the same way, leading to the same values.
   - Even if using dropouts, it is better to avoid symmetry.
2. Common initialization choices are random (preferable) and orthogonal matrices:
   - Gaussian initialization: typically with mean 0 and small variance.
   - Uniform initialization (Xavier initialization): $W_{ij} \sim \text{Uniform}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$.
   - Sparse initialization: Martens (2010) propose to have exactly k non-zero weights at each layer.
   - [Saxe et al., 2013] recommend using orthogonal weight matrices for initialization, which avoids vanishing gradients.
3. Biases are usually constants chosen heuristically, *e.g.*, 0.1 for ReLU, 1 for LSTM.

Outline

- Difference between learning and pure optimization
- Challenges in network optimization
- Basic algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rate

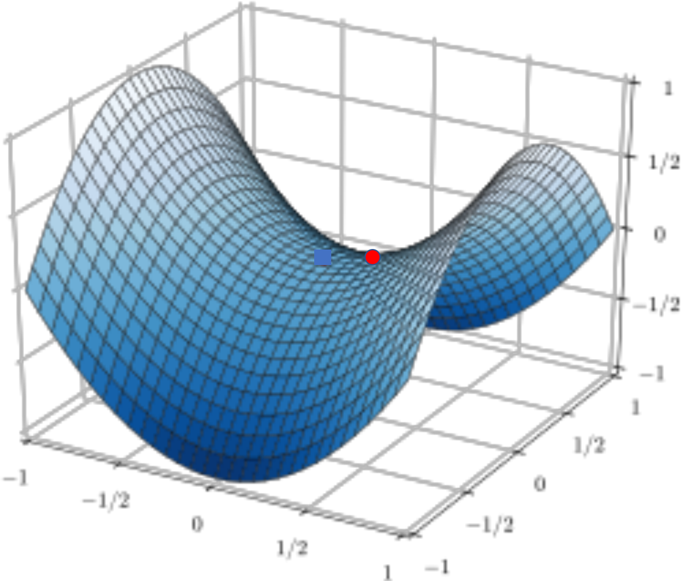
Adaptive Learning Rate

1. Learning rate is one of the hyperparameter that impacts the most.
2. The gradient is highly sensitive to some directions.
3. In high-dimensional space, it makes sense to use separate rates for each parameter.



Adaptive Learning Rate

In what direction should the learning rate be larger?



AdaGrad [Duchi et al., 2011]

1. Scale the gradient according to the historical norms.
2. Learning rates of parameters with high partial derivatives decrease fast.
3. Enforces progress in more gently sloped directions.

AdaGrad algorithm

- Accumulate squared gradients:

$$\mathbf{r} = \mathbf{r} + \bar{\mathbf{g}} \odot \bar{\mathbf{g}}$$

- Element-wise update (δ is a small constant for stabilization):

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \bar{\mathbf{g}}$$

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

RMSProp [Hinton, 2012]

1. Modification of AdaGrad to perform better on non-convex problems.
2. AdaGrad accumulates since beginning, gradient may be too small before reaching a convex structure:
   - gradients canceled out by historical gradients.
3. RMSProp uses an exponentially weighted moving average:
   - emphasize on recent gradients.

RMSProp algorithm

- Accumulate squared gradients (ρ is the decay rate):

$$\mathbf{r} = \rho\,\mathbf{r} + (1 - \rho)\bar{\mathbf{g}} \odot \bar{\mathbf{g}}$$

- Element-wise update:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \bar{\mathbf{g}}$$

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Adam [Kingma and Ba, 2014]

1. Adaptive Moments, variation of RMSProp + Momentum.
2. Momentum is incorporated directly as an estimate of the first order moment.
3. Also add bias correction to the moments, in the sense that the estimation of the moments are unbiased.

Adam algorithm

- Update time step: $t = t + 1$
- Update biased moment estimates:

$$s = \rho_1 s + (1 - \rho_1)\bar{\mathbf{g}}; \quad r = \rho_2 r + (1 - \rho_2)\bar{\mathbf{g}} \odot \bar{\mathbf{g}}$$

- Correct biases:

$$\hat{s} = \frac{s}{1 - \rho_1^t}; \quad \hat{r} = \frac{r}{1 - \rho_2^t}$$

- Update parameters: $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon \frac{\hat{s}}{\delta + \sqrt{\hat{r}}}$

Bias Correction in Adam

- Take **r** for example:

$$\mathbf{r}_t = (1-\rho_2)\sum_{i=1}^{t}\rho_2^{t-i}\,\mathbf{g}_i^2$$

$$\Rightarrow \mathbb{E}[\mathbf{r}_t] = \mathbb{E}\left[(1-\rho_2)\sum_{i=1}^{t}\rho_2^{t-i}\,\mathbf{g}_i^2\right]$$

Assume $\mathbb{E}[\mathbf{g}_i^2]$ is stationary, *i.e.*, $\mathbb{E}[\mathbf{g}_i^2]$ are equal for $\forall i$

$$= \mathbb{E}[\mathbf{g}_i^2](1-\rho_2)\sum_{i=1}^{t}\rho_2^{t-i}$$

$$= \mathbb{E}[\mathbf{g}_i^2](1-\rho_2^t)$$

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Adamw: Adam with Weight Decay[3]

- Apply weight decay on the gradient:

$$\bar{g} = \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}) - \lambda \boldsymbol{\theta}_{t-1}$$

Adamw algorithm

- Update time step: $t = t + 1$
- Weight decay on gradient:

$$\bar{g} = \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}) - \lambda \boldsymbol{\theta}_{t-1}$$

- Update biased moment estimates:

$$s = \rho_1 s + (1 - \rho_1)\bar{\mathbf{g}}; \quad r = \rho_2 r + (1 - \rho_2)\bar{\mathbf{g}} \odot \bar{\mathbf{g}}$$

- Correct biases: $\hat{s} = \frac{s}{1-\rho_1^t}; \quad \hat{r} = \frac{r}{1-\rho_2^t}$
- Update parameters: $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon(\frac{\hat{s}}{\delta + \sqrt{\hat{r}}} + \lambda \boldsymbol{\theta})$

[3] https://arxiv.org/abs/1711.05101

Visualization Comparison for Different Optimization Algorithms

https://emiliendupont.github.io/2018/01/24/optimization-visualization/

Visualization Comparison

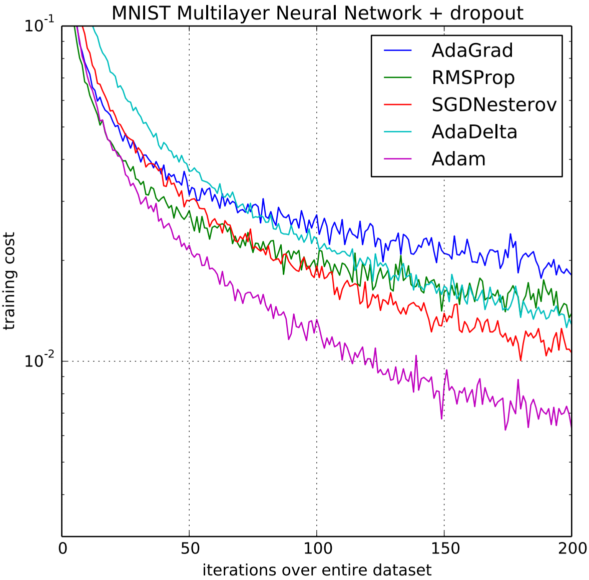
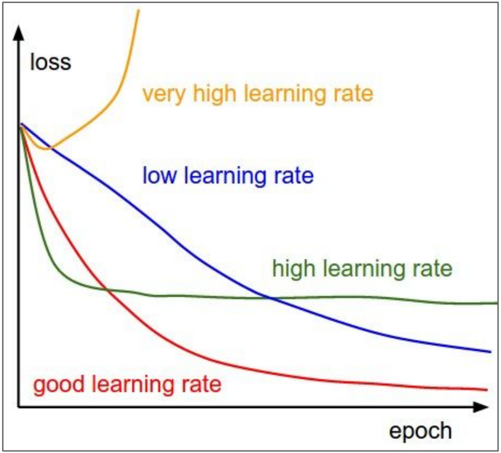


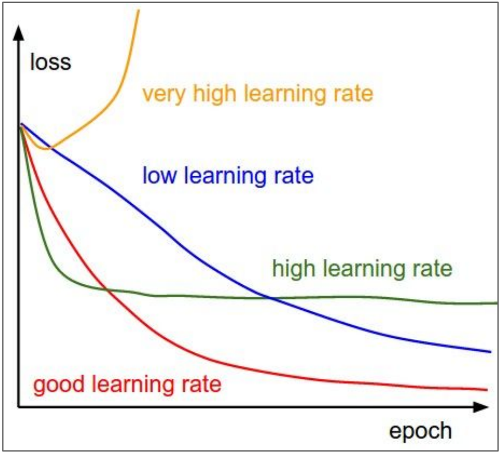
Image credit: David Carlson

Learning Rates



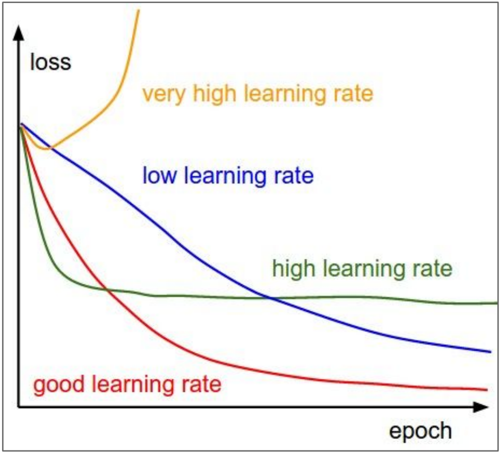
- How to choose the learning rate?
- Learning rate decay over time:
  1. step decay, decay learning rate by half every several epochs.
  2. exponential decay: $\alpha = \alpha_0 e^{-kt}$.
  3. linearly decay: $\alpha = \frac{\alpha_0}{1+kt}$.
  4. $\cdots$

Learning Rates



- How to choose the learning rate?
- Learning rate decay over time:
  1. step decay, decay learning rate by half every several epochs.
  2. exponential decay: $\alpha = \alpha_0 e^{-kt}$.
  3. linearly decay: $\alpha = \frac{\alpha_0}{1+kt}$.
  4. $\cdots$

Learning Rates



- How to choose the learning rate?
- Learning rate decay over time:
  1. step decay, decay learning rate by half every several epochs.
  2. exponential decay: $\alpha = \alpha_0 e^{-kt}$.
  3. linearly decay: $\alpha = \frac{\alpha_0}{1+kt}$.
  4. $\cdots$

Convergence

- Given an arbitrary, unknown sequence of convex cost functions $f_1(\boldsymbol{\theta}), f_2(\boldsymbol{\theta}), \cdots, f_T(\boldsymbol{\theta})$.
- At each time t, our goal is to predict the parameter $\boldsymbol{\theta}_t$ and evaluate it on a previously unknown cost function f_t.
- Evaluate the algorithm using the regret:

$$R(T) \triangleq \sum_{t=1}^{T} [f_t(\boldsymbol{\theta}_t) - f_t(\boldsymbol{\theta}^*)] \ ,$$

where $\boldsymbol{\theta}^* \triangleq \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{t=1}^{T} f_t(\boldsymbol{\theta})$ is the best possible solution.

Regret Bounds for Adaptive Gradient Methods

For some particular class of functions and data features,

$$R(T) = O\left(\log d\sqrt{T}\right)$$

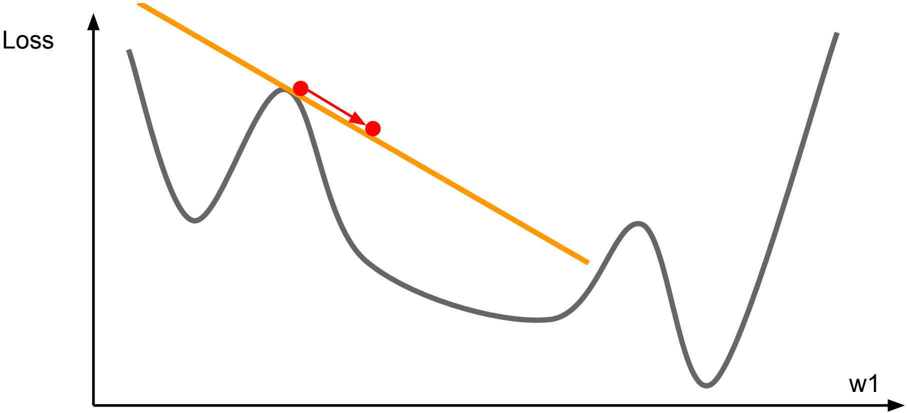
Duchi *et al.*, 2011; Kingma & Ba, 2015

Second Order Methods (Optional)

Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

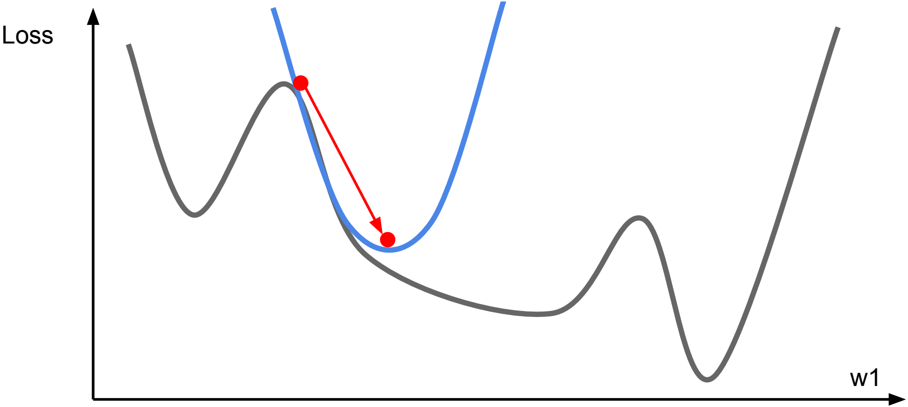
First-Order Optimization

- Step on the gradient direction.
- Hard to choose stepsize.



Second-Order Optimization

- Use gradient and Hessian to form a quadratic approximation of the original objective function at each point.
- Step to the minima of the approximation.



Second-Order Optimization

- Second Taylor-expansion:

$$J(\boldsymbol{\theta}) = J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Solving for the critic point, we arrive the Newton parameter update equation:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Second-Order Optimization

- Second Taylor-expansion:

$$J(\boldsymbol{\theta}) = J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Solving for the critic point, we arrive the Newton parameter update equation:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Q1: What is nice about this update?

Second-Order Optimization

- Second Taylor-expansion:

$$J(\boldsymbol{\theta}) = J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Solving for the critic point, we arrive the Newton parameter update equation:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Q2: Is it good for deep learning?

Second-Order Optimization

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

1. Quasi-Newton methods (BFGS most popular):
   - Instead of inverting the Hessian ($O(n^3)$), approximate inverse Hessian with rank 1 updates over time ($O(n^2)$ each).
2. L-BFGS (Limited memory BFGS):
   - Does not form/store the full inverse Hessian.

BFGS

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

$$\mathbf{g}_n = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_n), \qquad \mathbf{s}_n = \boldsymbol{\theta}_n - \boldsymbol{\theta}_{n-1}$$

$$\mathbf{y}_n = \mathbf{g}_n - \mathbf{g}_{n-1}, \qquad \rho_n = \frac{1}{\mathbf{y}_n^T \mathbf{s}_n}$$

- Update the pseudo-Hessian in each iteration as:

$$\mathbf{H}_{n+1}^{-1} = \left(\mathbf{I} - \rho_n \mathbf{y}_n \mathbf{s}_n^T\right) \mathbf{H}_n^{-1} \left(\mathbf{I} - \rho_n \mathbf{s}_n \mathbf{y}_n^T\right) + \rho_n \mathbf{s}_n \mathbf{s}_n^T$$

- Since in parameter update, we only need to calculate terms like $H^{-1}\,\mathbf{b}$, with $\mathbf{b}$ a vector:
  - We don't need to store the huge matrix $\mathbf{H}$.
  - Only need to store the history of $\{\mathbf{g}_n, \mathbf{s}_n, \mathbf{y}_n, \rho_n\}$, then use them to reconstruct $\mathbf{H}_n^{-1}$.
  - Time complexity reduces to $O(d^2)$.
  - L-BFGS only store a limited $\{\mathbf{g}_n, \mathbf{s}_n, \mathbf{y}_n, \rho_n\}$.

In Practice

1. L-BFGS usually works very well in full batch, deterministic mode, *i.e.*, if you have a single, deterministic $f(\theta)$, then L-BFGS will probably work very nicely.
2. Does not transfer very well to mini-batch setting:
    - Givess bad results.
    - Adapting L-BFGS to large-scale, stochastic setting is an active area of research.

Li *et al.*, ICML 2011