Optimization

Rowel Atienza

rowel@eee.upd.edu.ph

University of the Philippines

Optimization

Finding the parameters, $\boldsymbol{\theta}$, of a neural network that significantly reduce the loss function $L(\boldsymbol{\theta})$

Measured in terms of a performance measure, P, on the entire training set and some regularization terms

P is the one that makes Optimization in Machine Learning different from just pure optimization as the end goal itself.

Optimization

Loss function from an empirical distribution p'_{data} (over the training set):

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p'_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y})$$

Where $f(\boldsymbol{x};\boldsymbol{\theta})$ is prediction and $(\boldsymbol{x},\boldsymbol{y})\sim p'_{data}$ are dataset samples

Empirical Risk Minimization

$$L(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p'_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y}) = \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right),\boldsymbol{y}^{(i)}\right)$$

m is the number of samples or batch size

Minibatch Stochastic

Using entire training set is expensive and has no linear return
Use of minibatch stochastic (small subset of entire training set) offers many advantages:

Suitable for parallelization; GPU memory limits batch size

GPUs perform better on power of 2 sizes, 32 to 256

Small batches offer regularizing effects; improves generalization error

Small batch size increases gradient variance; learning rate must be reduced

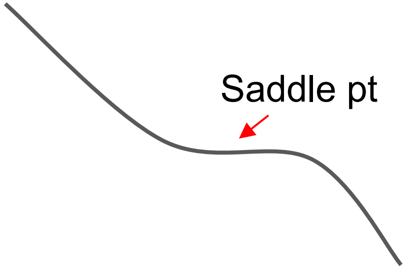
Shuffle minibatch, make minibatches independent improves training

Challenges

Convex loss function: any local minimum is a global minimum

Non-convex: any local minimum is not necessarily a global minimum. We settle for a local minimum that satisfies our performance metrics.

Challenges



Saddle points: found in high-dimensional models

Hessian matrix both have positive and negative eigenvalues

Saddle pts common in high dim space; Local min in low dim space

Can be easily overcome by SGD

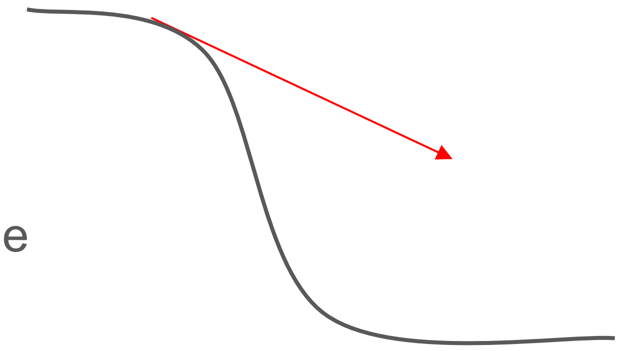
Challenges

Cliff

Gradient descent proposes a large change thus missing the minimum - Exploding Gradient

Solution is to use **gradient clipping** - capping the gradient

Common problem is Recurrent Neural Networks



Challenges

Long Term Dependencies (eg RNN, LSTM)

Performing the same computation many times

Applying the same $\boldsymbol{W}$ t-times

$$\boldsymbol{W}^t = (\boldsymbol{V} diag(\boldsymbol{\lambda}) V^{-1})^t = \boldsymbol{V} diag(\boldsymbol{\lambda})^t \boldsymbol{V}^{-1}$$

if $|\boldsymbol{\lambda}| < 1$, the term vanishes as t increases

if $|\boldsymbol{\lambda}| > 1$, the term explodes as t increases

Gradients are influenced by $diag(\boldsymbol{\lambda})$

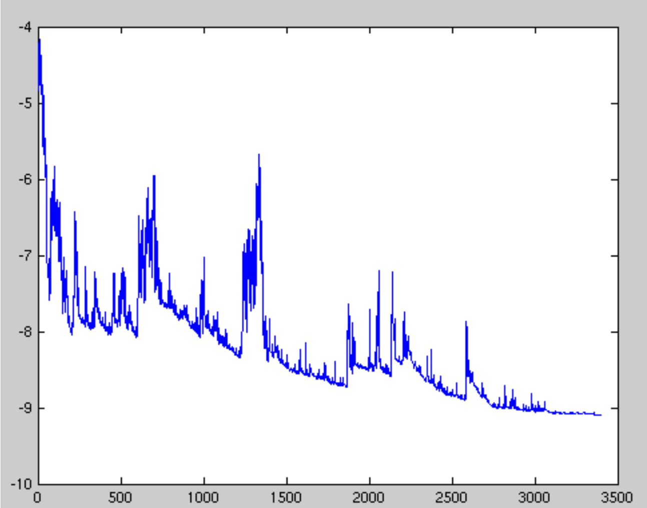
Collectively known as the **vanishing or exploding gradients problem**

Challenges

Inexact gradients due to noisy, biased estimates or small batch size

Optimization does not necessarily lead to a critical pt (global, local or saddle).

Most of the time, only near zero gradient points with resulting acceptable performance

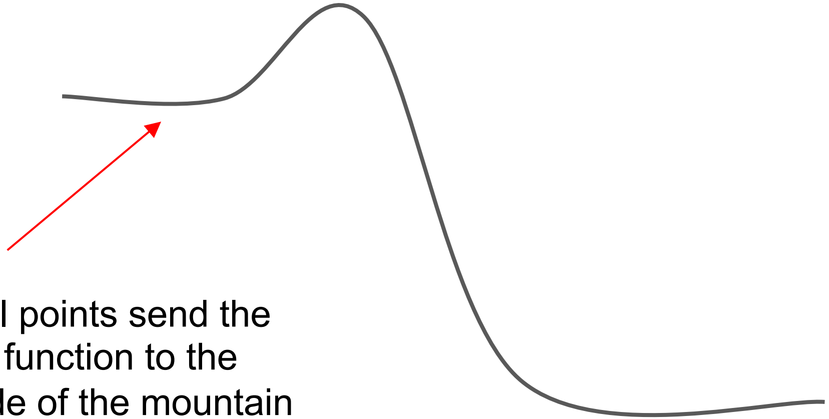


Noisy gradients

Challenges

Wrong side of the mountain: gradient descent will not find the minimum

Solution: algorithm for choosing the initial points (initial parameters)



Bad initial points send the objective function to the wrong side of the mountain

Parameter Initialization - Weights

Easiest : Sample from a Gaussian distribution with zero mean and small standard deviation (e.g. std = 0.01)

Other initializations:

Glorot, He, LeCun

Glorot

Assume a network layer with m inputs, n outputs

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \quad Glorot\ Uniform$$

$$W \sim \mathcal{N}\left(w; 0, \sqrt{\frac{2}{m+n}}\right) \quad Glorot\ Normal$$

[Glorot & Bengio, AISTATS 2010 - http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

He

Assume a network layer with m inputs

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{m}}, \sqrt{\frac{6}{m}}\right) \quad He\ Uniform$$

$$W \sim \mathcal{N}\left(w; 0, \sqrt{\frac{2}{m}}\right) \quad He\ Normal$$

[He et al., http://arxiv.org/abs/1502.01852]

Parameter Initialization - Biases

Set biases to zero (only for linear activation)

Use small values (eg 0.1) for ReLU activation

1 for LSTM forget state

Optimization Algorithms

Stochastic Gradient Descent

with Momentum

Adaptive Gradient (AdaGrad)

RMSProp

with Momentum

Adaptive Moment (Adam)

Stochastic Gradient Descent

Instead of using the whole training set, we use a minibatch of m iid samples

Gradually decrease learning rate during training since after some time, the gradient due to noise is more significant

Typical initial learning rate values [0.1 to 0.001]

Typical learning rate decay: cosine, multi-step

Stochastic Gradient Descent (SGD) Algorithm

```
Require: Learning Rate Scheduler: $\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \ldots\ \boldsymbol{\epsilon}_k$
Require: Initial Parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion is not met do
    Sample a minibatch { $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}$ } with corresponding targets { $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}$ }
    Compute gradient estimate: $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \boldsymbol{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
    Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon_k \boldsymbol{g}$
    $k = k + 1$
  end while
```

Momentum on SGD for Speed Improvement

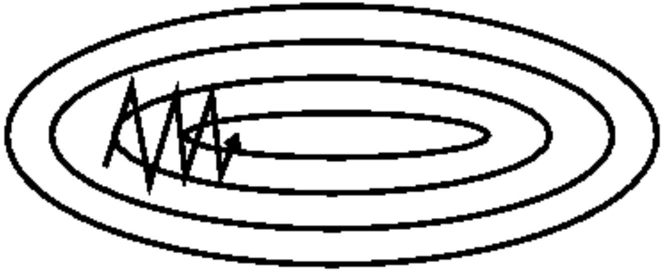
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \boldsymbol{\epsilon g}$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

Where $\boldsymbol{v}$ is the accumulator of gradient $\boldsymbol{g}$; $\boldsymbol{v}$ includes influence of past gradients, $\boldsymbol{g}$

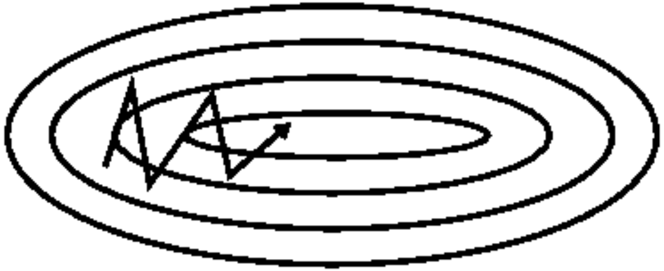
α is momentum [0,1); typical 0.5, 0.9 and 0.99

α is larger compared to $\boldsymbol{\epsilon}$, the bigger is the influence of past $\boldsymbol{g}$’s; similar to snowballing effect

Momentum on SGD for Speed Improvement



SGD without Momentum



SGD with Momentum

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

SGD Algorithm with Momentum

```
Require: Learning Rate Scheduler: $\epsilon_1, \epsilon_2, \ldots\ \epsilon_k$
Require: Initial Parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion is not met do
    Sample a minibatch { $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}$ } with corresponding targets { $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)}$ }
    Compute gradient estimate $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \boldsymbol{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
    Compute velocity: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
    $k = k + 1$
  end while
```

Momentum on SGD for Speed Improvement

Nesterov Momentum: Loss is evaluated after the momentum is applied

$$\boldsymbol{\theta} \rightarrow (\boldsymbol{\theta} + \alpha \boldsymbol{v})$$

SGD Algorithm with Nesterov Momentum

```
Require: Learning Rate Scheduler: $\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \dots\ \boldsymbol{\epsilon}_k$
Require: Initial Parameter $\boldsymbol{\theta}$
    $k \leftarrow 1$
    while stopping criterion is not met
        Sample a minibatch $\{ \boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)} \}$ with corresponding targets $\{ \boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(m)} \}$
        Compute gradient estimate $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \boldsymbol{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}\right), \boldsymbol{y}^{(i)}\right)$
        Compute velocity: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \boldsymbol{\epsilon} \boldsymbol{g}$
        Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
        $k = k + 1$
    end while
```

Adaptive Learning Rates

AdaGrad (Adaptive Gradient) [Duchi et al 2011]: learning rate decrease is inversely proportional to the square root of the sum of all the historical squared values of the gradient

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$$

$$\Delta\boldsymbol{\theta} \leftarrow -\boldsymbol{g}\frac{\in}{\delta + \sqrt{\boldsymbol{r}}}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$$

Where δ = small constant (eg 10^{-7})

AdaGrad Algorithm

```
Require: Learning Rate: $\boldsymbol{\epsilon}$, Initial Parameter $\boldsymbol{\theta}$, Small constant $\delta = 10^{-7}$ for numerical stability
    $\boldsymbol{r} \leftarrow \mathbf{0}$
    while stopping criterion is not met do
        Sample a minibatch $\{ \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)} \}$ with corresponding targets $\{ \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)} \}$
        Compute gradient estimate $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \boldsymbol{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
        Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
        Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\boldsymbol{g} \frac{\in}{\delta + \sqrt{\boldsymbol{r}}}$
        Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
    end while
```

Adaptive Learning Rates

RMSProp [Hinton 2012] is like AdaGrad but replaces gradient accumulation with exponentially weighted moving average; Suitable for nonconvex optimization

$$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$$

$$\Delta\boldsymbol{\theta} \leftarrow -\boldsymbol{g}\frac{\in}{\delta + \sqrt{\boldsymbol{r}}}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$$

where δ is a small constant (eg 10^{-6}); ρ is the decay rate (e.g. 0.9)

Discard history from extreme past. Effective and practical for deep neural nets

RMSProp Algorithm

```
Require: Learning Rate: $\boldsymbol{\epsilon}$, Initial Parameter $\boldsymbol{\theta}$, Small constant $\delta = 10^{-6}$ for numerical stability, Decay rate $\rho$ (eg 0.9)
    $\boldsymbol{r} \leftarrow \mathbf{0}$
    while stopping criterion is not met do
        Sample a minibatch $\{ \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)} \}$ with corresponding targets $\{ \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)} \}$
        Compute gradient estimate $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \boldsymbol{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
        Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
        Compute param update: $\Delta \boldsymbol{\theta} \leftarrow -\boldsymbol{g} \frac{\in}{\delta + \sqrt{\boldsymbol{r}}}$
        Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
    end while
```

Adaptive Learning Rates

Adam (Adaptive Moments) [Kingma and Ba 2014]

first moment: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g}$, $\boldsymbol{s}' \leftarrow \frac{\boldsymbol{s}}{(1-{\rho_1}^t)}$

Built-in 1st-order momentum & correction

second moment: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\, \boldsymbol{g} \odot \boldsymbol{g}$, $\boldsymbol{r}' \leftarrow \frac{\boldsymbol{r}}{(1-{\rho_2}^t)}$

Built-in 2nd-order momentum & correction

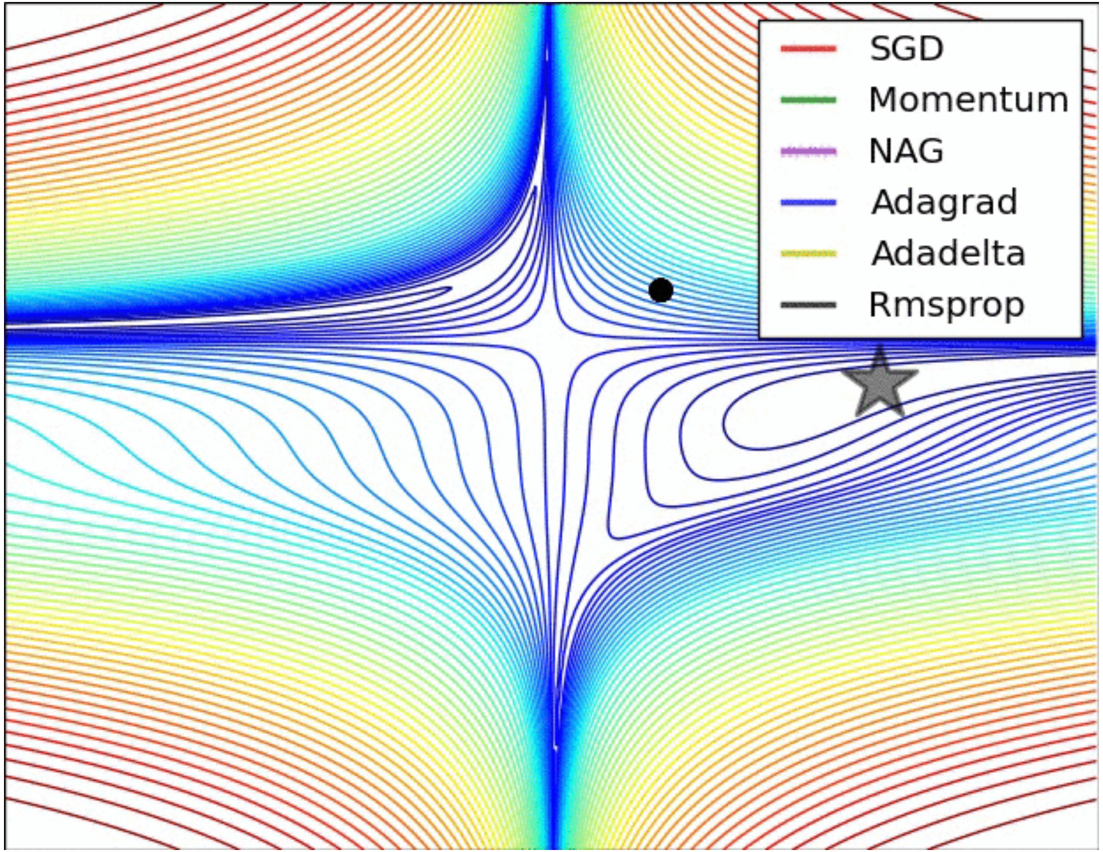
$$\boldsymbol{\Delta\theta} \leftarrow -\boldsymbol{s}' \frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}'}}, \ \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\Delta\theta}$$

where δ is a small constant for numerical stabilization (eg 10^{-8}), ρ_1 and $\rho_2 \in [0, 1)$ (suggest: $\rho_1 = 0.9$, $\rho_2 = 0.999$), t is time step, $\boldsymbol{\epsilon}$ is suggested to be 0.001

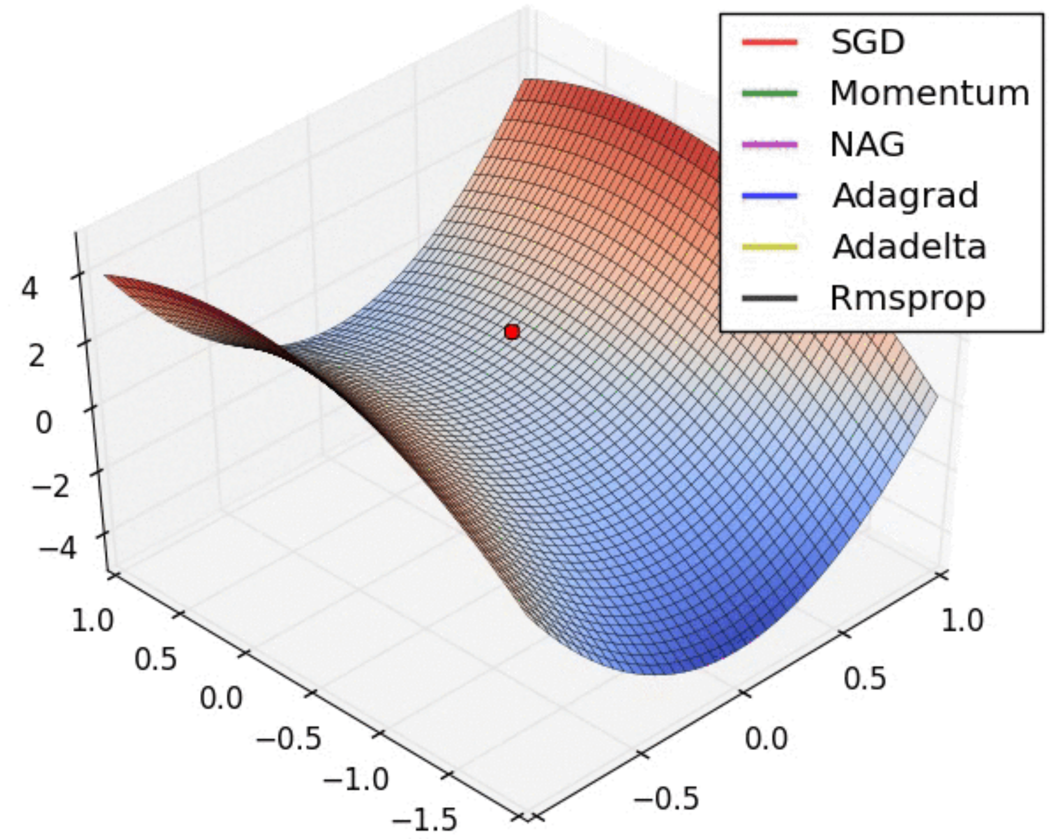
Adam Algorithm

Require: Learning Rate: $\boldsymbol{\epsilon}$ (eg 0.001), Initial Parameter $\boldsymbol{\theta}$, δ = small constant for numerical stabilization (eg 10^{-8}), ρ_1 and ρ_2 ∈ [0, 1) (suggest: ρ_1 = 0.9, ρ_2 = 0.999), t is time step, $\boldsymbol{\epsilon}$ is suggested to be 0.001

```
$\boldsymbol{r} \leftarrow \boldsymbol{0}, \boldsymbol{s} \leftarrow \boldsymbol{0}, t \leftarrow 0$
while stopping criterion is not met do
    Sample a minibatch { $\boldsymbol{x}^{(1)}$, ..., $\boldsymbol{x}^{(m)}$ } with corresponding targets { $\boldsymbol{y}^{(1)}$, ..., $\boldsymbol{y}^{(m)}$ }
    Compute gradient estimate $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \boldsymbol{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
    $t = t + 1$
    Compute 1st-moment and correction: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}, \boldsymbol{s}' \leftarrow \frac{\boldsymbol{s}}{(1 - \rho_1{}^{t})}$
    Compute 2nd-moment and correction: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\, \boldsymbol{g} \odot \boldsymbol{g},\ \boldsymbol{r}' \leftarrow \frac{\boldsymbol{r}}{(1 - \rho_2{}^{t})}$
    Compute and apply update: $\Delta\boldsymbol{\theta} \leftarrow -\boldsymbol{s}' \frac{\in}{\delta + \sqrt{\boldsymbol{r}'}}, \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while
```



Behavior on loss surface



Behavior on saddle point

Reference

Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016, http://www.deeplearningbook.org

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

In Summary

SGD with Momenntum and Adam are usually the default go to optimizers

No clear winner on which optimizer is the best performing

Usually it depends on the task

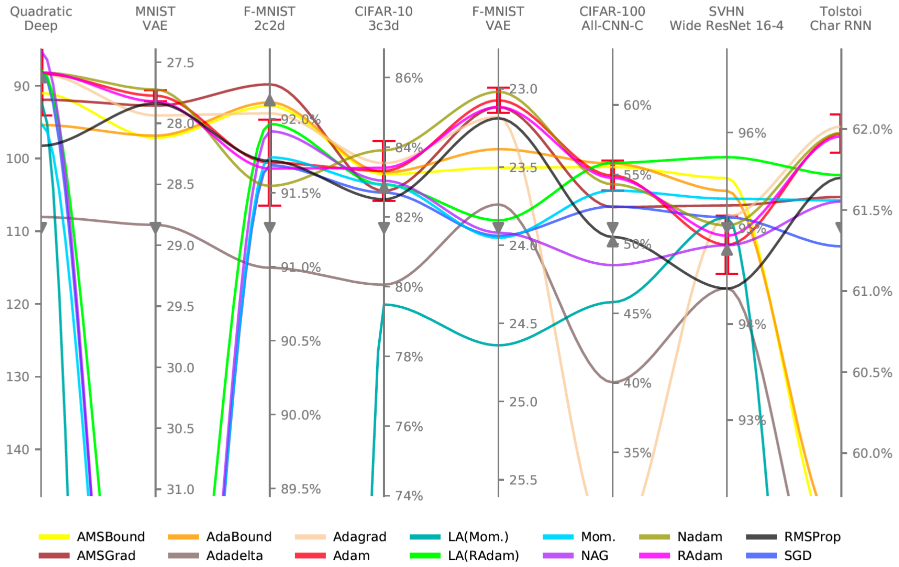


Figure 4: Mean test set performance over 10 random seeds of all tested optimizers on all eight optimization problems using the *large budget* for tuning and *no learning rate schedule*. One standard deviation for the *tuned* ADAM optimizer is shown with a red error bar (**I**; error bars for other methods omitted for legibility). The performance of *untuned* ADAM (▼) and ADABOUND (▲) are marked for reference. The upper bound of each axis represents the best performance achieved in the benchmark, while the lower bound is chosen in relation to the performance of ADAM with default parameters.

Schmidt, Robin M., Frank Schneider, and Philipp Hennig. "Descending through a Crowded Valley--Benchmarking Deep Learning Optimizers." *arXiv preprint arXiv:2007.01547* (2020).

Batch Normalization

Applied layer-wise

To maintain zero mean and variance of 1 for activation outputs

Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift).

Allows use of larger learning rate in deep models w/o causing instability

Applied to any input or hidden layer

$\mathbf{H}' = (\mathbf{H} - \boldsymbol{\mu})/\boldsymbol{\sigma}$ (row-wise operation, one sample is one row)

where **H** matrix is a minibatch of activations of the layer to normalize

$\boldsymbol{\mu}$ vector of mean of each unit, $\boldsymbol{\sigma}$ vector of std of each unit (broadcast op)

Batch Normalization

At training time,

$\boldsymbol{\mu} = 1/m\sum_i \mathbf{H}_{i,:}$

$\boldsymbol{\sigma} = \mathrm{sqrt}(\delta + 1/m\sum_i (\mathbf{H}\text{-}\boldsymbol{\mu})_i{}^2)$

$\delta = 10^{-8}$ for numerical stability

In retrospect, this operations discourages **μ** and **σ** from attempting to propose large changes

In practice, **μ** and **σ** are stored as running averages so we do not have to compute (1) and (2) all the time.