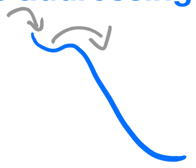
You read about momentum, AdaGrad, RMSProp, Adam.
What challenge(s) of optimization are these methods addressing?

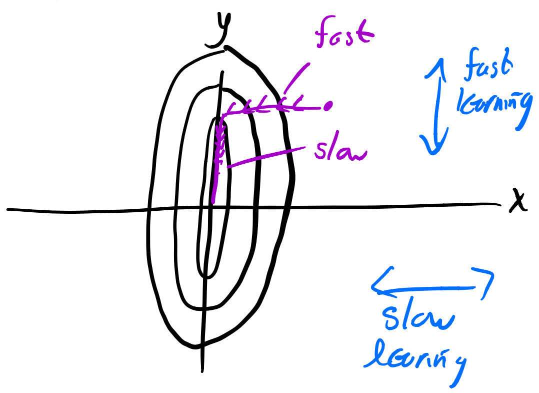
Avoid getting stuck in local minima



Need adaptive learning rates
—- In some directions learning should be faster
—- in some directions learning should be slower
Ideally we could decide which directions are which and tune learning rate accordingly

Objective function may not be stationary (changes over time)

$$\min_{x,y} f(x,y) := \underset{\text{high curvature}}{x^2} + \underset{\text{slow curv}}{\varepsilon y^2} \quad w/ \ \varepsilon \ll 1$$



Recall: If learning rate $\eta > \frac{2}{\text{max curvature}}$

then GD diverged

With just y problem, we could use l.r. $\frac{2}{\varepsilon}$

How does momentum work?

accelerate learning
exponentially weighted moving avg
related to physics 1st order ODE

$$\min_x f(x)$$

v - velocity $\quad \alpha$ - decay rate

$$\begin{cases} V_{t+1} = \alpha V_t \underbrace{- \eta \nabla f(X_t)}_{\eta_t} & (*) \\ X_{t+1} = X_t + V_{t+1} \end{cases}$$

How is this a exponentially weighted moving avg?

$$V_1 = \alpha \cdot V_0 + \eta_0$$

$$V_2 = \alpha V_1 + \eta_1 = \alpha^2 V_0 + \alpha \eta_0 + \eta_1$$

$$\vdots$$

$$V_t = \alpha^t V_0 + \underbrace{\alpha^{t-1}}_{\text{exponentially small weight on past gradient}} \eta_0 + \alpha^{t-2} \eta_1 + \cdots + \underbrace{\eta_{t-1}}_{\text{more weight here}}$$

Roughly, how many past iterations are relevant under parameter α?

If $\alpha \approx 1$, very large window

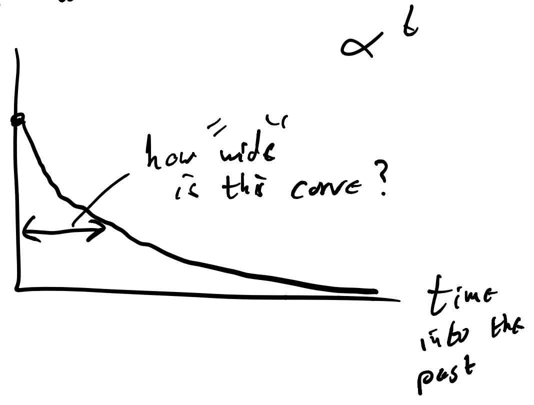
If $\alpha \approx 0$, very small window

window size given by "half life" of this decay process

$$\alpha^t = 1/2$$

$$t \lg \alpha = -\lg 2$$

$$t = -\frac{\lg 2}{\lg \alpha}$$



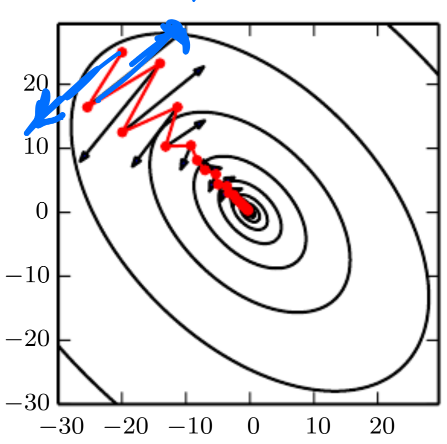
8.3.2 Momentum

Ideas: exponential moving average, physics interpretation, relation to challenges of optimization , relation to SGD

While stochastic gradient descent remains a popular optimization strategy, learning with it can sometimes be slow. The method of momentum (Polyak, 1964) is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move

with momentum, ∇f is being averaged over multiple time steps.

avg these, so then opposite directions "cancel" resulting in smaller effective learning rate in that direction



In the dir. $\nwarrow\searrow$, all derivs are in same direction so avg is also in that same dir without much change in size

Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function. At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point. We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides. Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon. Compare also figure 4.6, which shows the behavior of gradient descent without momentum.

in their direction. The effect of momentum is illustrated in figure 8.5.

Formally, the momentum algorithm introduces a variable $\boldsymbol{v}$ that plays the role of velocity—it is the direction and speed at which the parameters move through parameter space. The velocity is set to an exponentially decaying average of the negative gradient. The name **momentum** derives from a physical analogy, in which the negative gradient is a force moving a particle through parameter space, according to Newton's laws of motion. Momentum in physics is mass times velocity. In the momentum learning rule, we assume unit mass, so the velocity vector $\boldsymbol{v}$

may also be regarded as the momentum of the particle. A hyperparameter $\alpha \in [0, 1)$ determines how quickly the contributions of previous gradients exponentially decay. The update rule is given by

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right), \tag{8.15}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{8.16}$$

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while
```

The velocity $\boldsymbol{v}$ accumulates the gradient elements $\nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$. The larger α is relative to ϵ, the more previous gradients affect the current direction. The SGD algorithm with momentum is given in algorithm 8.2.

Previously, the size of the step was simply the norm of the gradient multiplied by the learning rate. Now, the size of the step depends on how large and how aligned a *sequence* of gradients are. The step size is largest when many successive gradients point in exactly the same direction. If the momentum algorithm always observes gradient $\boldsymbol{g}$, then it will accelerate in the direction of $-\boldsymbol{g}$, until reaching a terminal velocity where the size of each step is

$$\frac{\epsilon ||\boldsymbol{g}||}{1 - \alpha}. \tag{8.17}$$

It is thus helpful to think of the momentum hyperparameter in terms of $\frac{1}{1-\alpha}$. For example, $\alpha = 0.9$ corresponds to multiplying the maximum speed by 10 relative to the gradient descent algorithm.

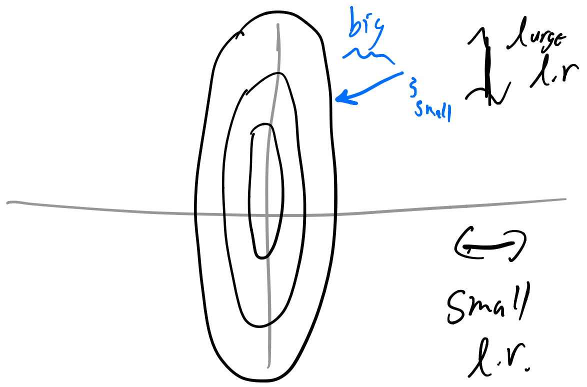
Common values of α used in practice include 0.5, 0.9, and 0.99. Like the learning rate, α may also be adapted over time. Typically it begins with a small value and is later raised. Adapting α over time is less important than shrinking ϵ over time.

What is the idea behind the idea behind AdaGrad?

over all time

Use individual learning rates for each parameter
Set each learning rate inversely proportional to sum of the square of previous gradients (of that parameter)

Downweight things with large gradients
Because those will typically be directions
Of high curvature (which we want low learning Rates)



Sparsity
- What is an example of a problem where some gradients are "sparse" (many coefficients are zero and a few are nonzero)

Eg. Natural Language Processing. - Consider the problem of detecting whether a review of a product is positive or negative . Input to your classifier ("bag of words" - each coordinate corresponds to one word, and there is a 1 for each word in the review).
Any individual review will have very sparse features

In the context of convex optimization, the AdaGrad algorithm enjoys some desirable theoretical properties. Empirically, however, for training deep neural network models, the accumulation of squared gradients *from the beginning of training* can result in a premature and excessive decrease in the effective learning rate. AdaGrad performs well for some but not all deep learning models.

8.5.2 RMSProp

The **RMSProp** algorithm (Hinton, 2012) modifies AdaGrad to perform better in the nonconvex setting by changing the gradient accumulation into an exponentially weighted moving average. AdaGrad is designed to converge rapidly when applied to a convex function. When applied to a nonconvex function to train a neural network,

the learning trajectory may pass through many different structures and eventually arrive at a region that is a locally convex bowl. AdaGrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure. RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl, as if it were an instance of the AdaGrad algorithm initialized within that bowl.

Algorithm 8.4 The AdaGrad algorithm

```
**Require:** Global learning rate $\epsilon$
**Require:** Initial parameter $\boldsymbol{\theta}$
**Require:** Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  **while** stopping criterion not met **do**
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
  **end while**
```

Algorithm 8.5 The RMSProp algorithm

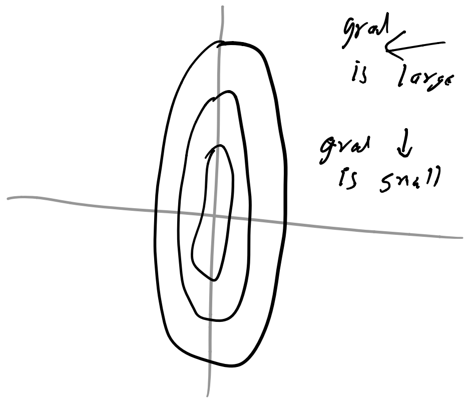
```
**Require:** Global learning rate $\epsilon$, decay rate $\rho$
**Require:** Initial parameter $\boldsymbol{\theta}$
```

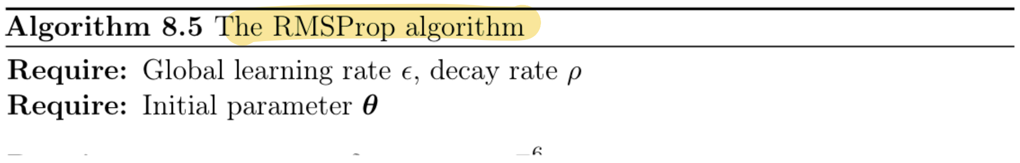
What is the idea of RMSProp?

Use of exponential moving average.

RMS - room mean square

Method - divide the learning rate of each parameter by root mean square value of its past gradients (computed by an exponential moving average)





Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ

Require: Initial parameter $\boldsymbol{\theta}$

```
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small
  numbers
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
  end while
```

RMSProp is shown in its standard form in algorithm 8.5 and combined with Nesterov momentum in algorithm 8.6. Compared to AdaGrad, the use of the moving average introduces a new hyperparameter, ρ, that controls the length scale of the moving average.

Empirically, RMSProp has been shown to be an effective and practical optimization algorithm for deep neural networks. It is currently one of the go-to optimization methods being employed routinely by deep learning practitioners.

8.5.3 Adam

Adam (Kingma and Ba, 2014) is yet another adaptive learning rate optimization algorithm and is presented in algorithm 8.7. The name "Adam" derives from the phrase "adaptive moments." In the context of the earlier algorithms, it is perhaps best seen as a variant on the combination of RMSProp and momentum with a few important distinctions. First, in Adam, momentum is incorporated directly as an estimate of the first-order moment (with exponential weighting) of the gradient. The most straightforward way to add momentum to RMSProp is to apply momentum to the rescaled gradients. The use of momentum in combination with rescaling does not have a clear theoretical motivation. Second, Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin (see algorithm 8.7). RMSProp also incorporates an estimate of the

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

What is the idea behind the Adam optimizer?

Adam adapts learning rates of each parameter in terms

Exponentially weighted average of squared gradients (for each parameter)
Has momentum (exponential moving average of gradients)

There is a bias correction term which corrects for fact that initial value of moving averages is 0

Updates with bias-corrected velocity divided by root bias corrected second moments

Bias correction

$$m_{t+1} = \beta m_t + (1-\beta) g_t$$

$$m_1 = \beta m_0 + (1-\beta) g_0$$

$$m_2 = \beta m_1 + (1-\beta) g_1 = \beta^2 m_0 + \beta(1-\beta) g_0 + (1-\beta) g_1$$

$$\vdots$$

$$m_t = \beta^t \overset{0}{\cancel{m_0}} + \frac{\beta^{t-1}(1-\beta) g_0 + \beta^{t-2}(1-\beta) g_1 + \dots + (1-\beta) g_{t-1}}{\beta^{t-1}(1-\beta) + \beta^{t-2}(1-\beta) + \dots + (1-\beta)}$$

$$(1-\beta)\sum_{i=0}^{t-1} \beta^i = \frac{1-\beta^t}{1-\beta} \sim (1-\beta^t)$$

Stepsizes are approximately bounded by step size hyperparameter

$$\Delta_t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad \text{“} \lesssim \text{”} \; \alpha$$

Can actually be $> \alpha$

Typical cases: $m_t \approx \mathbb{E}[g]$, $v_t \approx \mathbb{E}[g^2]$

$$\mathbb{E}(g)^2 \leq \mathbb{E}(g^2)$$

So $\Delta_t \lesssim \alpha$ typically

Sparse case:

$g_1 = 0$
$g_2 = 0$
$g_{t-1} = 0$
$g_t \neq 0$

$$m_t = (1-\beta_1) g_t$$

$$v_t = (1-\beta_2) g_t^2$$

$$\Delta_t = \alpha \frac{(1-\beta_1)\cancel{g_t}}{\sqrt{(1-\beta_2)\cancel{g_t^2}}} = \alpha \frac{(1-\beta_1)}{\sqrt{1-\beta_2}}$$

If $\beta_1 = 0.9$
$\beta_2 = 0.999$

$$\Delta_t \approx \alpha \cdot \sqrt{10} \approx 3\alpha$$

Does not require stationary objective

Adagrad – maintains running <u>sum</u> of g^2

RMSProp & Adam – moving exponential avg of g^2

Works with sparse gradients?

Step size is controlled even in case of extreme sparsity

Need to remember many past values

Note: $\beta_2 = 0.999$ value is high, so long windows for averaging, so can handle sparse gradients

Naturally forms a step size annealing

$$\Delta_t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

As training progresses,

$$\frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \to 0$$

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma[*]
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba[*]
University of Toronto
jimmy@psi.utoronto.ca

ABSTRACT

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.

1 INTRODUCTION

Stochastic gradient-based optimization is of core practical importance in many fields of science and engineering. Many problems in these fields can be cast as the optimization of some scalar parameterized objective function requiring maximization or minimization with respect to its parameters. If the function is differentiable w.r.t. its parameters, gradient descent is a relatively efficient optimization method, since the computation of first-order partial derivatives w.r.t. all the parameters is of the same computational complexity as just evaluating the function. Often, objective functions are stochastic. For example, many objective functions are composed of a sum of subfunctions evaluated at different subsamples of data; in this case optimization can be made more efficient by taking gradient steps w.r.t. individual subfunctions, i.e. stochastic gradient descent (SGD) or ascent. SGD proved itself as an efficient and effective optimization method that was central in many machine learning success stories, such as recent advances in deep learning (Deng et al., 2013; Krizhevsky et al., 2012; Hinton & Salakhutdinov, 2006; Hinton et al., 2012a; Graves et al., 2013). Objectives may also have other sources of noise than data subsampling, such as dropout (Hinton et al., 2012b) regularization. For all such noisy objectives, efficient stochastic optimization techniques are required. The focus of this paper is on the optimization of stochastic objectives with high-dimensional parameters spaces. In these cases, higher-order optimization methods are ill-suited, and discussion in this paper will be restricted to first-order methods.

We propose *Adam*, a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name *Adam* is derived from adaptive moment estimation. Our method is designed to combine the advantages of two recently popular methods: AdaGrad (Duchi et al., 2011), which works well with sparse gradients, and RMSProp (Tieleman & Hinton, 2012), which works well in on-line and non-stationary settings; important connections to these and other stochastic optimization methods are clarified in section 5. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing.

[*]Equal contribution. Author ordering determined by coin flip over a Google Hangout.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

momentum

In section 2 we describe the algorithm and the properties of its update rule. Section 3 explains our initialization bias correction technique, and section 4 provides a theoretical analysis of Adam's convergence in online convex programming. Empirically, our method consistently outperforms other methods for a variety of models and datasets, as shown in section 6. Overall, we show that Adam is a versatile algorithm that scales to large-scale high-dimensional machine learning problems.

2 ALGORITHM

See algorithm 1 for pseudo-code of our proposed algorithm *Adam*. Let $f(\theta)$ be a noisy objective function: a stochastic scalar function that is differentiable w.r.t. parameters θ. We are interested in minimizing the expected value of this function, $\mathbb{E}[f(\theta)]$ w.r.t. its parameters θ. With $f_1(\theta), ..., , f_T(\theta)$ we denote the realisations of the stochastic function at subsequent timesteps $1, ..., T$. The stochasticity might come from the evaluation at random subsamples (minibatches) of datapoints, or arise from inherent function noise. With $g_t = \nabla_\theta f_t(\theta)$ we denote the gradient, i.e. the vector of partial derivatives of f_t, w.r.t θ evaluated at timestep t.

The algorithm updates exponential moving averages of the gradient (m_t) and the squared gradient (v_t) where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the 1st moment (the mean) and the 2nd raw moment (the uncentered variance) of the gradient. However, these moving averages are initialized as (vectors of) 0's, leading to moment estimates that are biased towards zero, especially during the initial timesteps, and especially when the decay rates are small (i.e. the βs are close to 1). The good news is that this initialization bias can be easily counteracted, resulting in bias-corrected estimates $\widehat{m}_t$ and $\widehat{v}_t$. See section 3 for more details.

Note that the efficiency of algorithm 1 can, at the expense of clarity, be improved upon by changing the order of computation, e.g. by replacing the last three lines in the loop with the following lines: $\alpha_t = \alpha \cdot \sqrt{1 - \beta_2^t} / (1 - \beta_1^t)$ and $\theta_t \leftarrow \theta_{t-1} - \alpha_t \cdot m_t / (\sqrt{v_t} + \hat{\epsilon})$.

2.1 ADAM'S UPDATE RULE

An important property of Adam's update rule is its careful choice of stepsizes. Assuming $\epsilon = 0$, the effective step taken in parameter space at timestep t is $\Delta_t = \alpha \cdot \widehat{m}_t / \sqrt{\widehat{v}_t}$. The effective stepsize has two upper bounds: $|\Delta_t| \le \alpha \cdot (1 - \beta_1) / \sqrt{1 - \beta_2}$ in the case $(1 - \beta_1) > \sqrt{1 - \beta_2}$, and $|\Delta_t| \le \alpha$

otherwise. The first case only happens in the most severe case of sparsity: when a gradient has been zero at all timesteps except at the current timestep. For less sparse cases, the effective stepsize will be smaller. When $(1-\beta_1) = \sqrt{1-\beta_2}$ we have that $|\widehat{m}_t/\sqrt{\widehat{v}_t}| < 1$ therefore $|\Delta_t| < \alpha$. In more common scenarios, we will have that $\widehat{m}_t/\sqrt{\widehat{v}_t} \approx \pm 1$ since $|\mathbb{E}[g]/\sqrt{\mathbb{E}[g^2]}| \leq 1$. The effective magnitude of the steps taken in parameter space at each timestep are approximately bounded by the stepsize setting α, i.e., $|\Delta_t| \lessapprox \alpha$. This can be understood as establishing a *trust region* around the current parameter value, beyond which the current gradient estimate does not provide sufficient information. This typically makes it relatively easy to know the right scale of α in advance. For many machine learning models, for instance, we often know in advance that good optima are with high probability within some set region in parameter space; it is not uncommon, for example, to have a prior distribution over the parameters. Since α sets (an upper bound of) the magnitude of steps in parameter space, we can often deduce the right order of magnitude of α such that optima can be reached from θ_0 within some number of iterations. With a slight abuse of terminology, we will call the ratio $\widehat{m}_t/\sqrt{\widehat{v}_t}$ the *signal-to-noise* ratio (SNR). With a smaller SNR the effective stepsize Δ_t will be closer to zero. This is a desirable property, since a smaller SNR means that there is greater uncertainty about whether the direction of $\widehat{m}_t$ corresponds to the direction of the true gradient. For example, the SNR value typically becomes closer to 0 towards an optimum, leading to smaller effective steps in parameter space: a form of automatic annealing. The effective stepsize Δ_t is also invariant to the scale of the gradients; rescaling the gradients g with factor c will scale $\widehat{m}_t$ with a factor c and $\widehat{v}_t$ with a factor c^2, which cancel out: $(c \cdot \widehat{m}_t)/(\sqrt{c^2 \cdot \widehat{v}_t}) = \widehat{m}_t/\sqrt{\widehat{v}_t}$.

3 Initialization bias correction

As explained in section 2, Adam utilizes initialization bias correction terms. We will here derive the term for the second moment estimate; the derivation for the first moment estimate is completely analogous. Let g be the gradient of the stochastic objective f, and we wish to estimate its second raw moment (uncentered variance) using an exponential moving average of the squared gradient, with decay rate β_2. Let $g_1, ..., g_T$ be the gradients at subsequent timesteps, each a draw from an underlying gradient distribution $g_t \sim p(g_t)$. Let us initialize the exponential moving average as $v_0 = 0$ (a vector of zeros). First note that the update at timestep t of the exponential moving average $v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (where g_t^2 indicates the elementwise square $g_t \odot g_t$) can be written as a function of the gradients at all previous timesteps:

$$v_t = (1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2 \tag{1}$$

We wish to know how $\mathbb{E}[v_t]$, the expected value of the exponential moving average at timestep t, relates to the true second moment $\mathbb{E}[g_t^2]$, so we can correct for the discrepancy between the two. Taking expectations of the left-hand and right-hand sides of eq. (1):

$$\mathbb{E}[v_t] = \mathbb{E}\left[(1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2\right] \tag{2}$$

$$= \mathbb{E}[g_t^2] \cdot (1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} + \zeta \tag{3}$$

$$= \mathbb{E}[g_t^2] \cdot (1-\beta_2^t) + \zeta \tag{4}$$

where $\zeta = 0$ if the true second moment $\mathbb{E}[g_i^2]$ is stationary; otherwise ζ can be kept small since the exponential decay rate β_1 can (and should) be chosen such that the exponential moving average assigns small weights to gradients too far in the past. What is left is the term $(1-\beta_2^t)$ which is caused by initializing the running average with zeros. In algorithm 1 we therefore divide by this term to correct the initialization bias.

In case of sparse gradients, for a reliable estimate of the second moment one needs to average over many gradients by chosing a small value of β_2; however it is exactly this case of small β_2 where a lack of initialisation bias correction would lead to initial steps that are much larger.

4 CONVERGENCE ANALYSIS

We analyze the convergence of Adam using the online learning framework proposed in (Zinkevich, 2003). Given an arbitrary, unknown sequence of convex cost functions $f_1(\theta)$, $f_2(\theta)$,..., $f_T(\theta)$. At each time t, our goal is to predict the parameter θ_t and evaluate it on a previously unknown cost function f_t. Since the nature of the sequence is unknown in advance, we evaluate our algorithm using the regret, that is the sum of all the previous difference between the online prediction $f_t(\theta_t)$ and the best fixed point parameter $f_t(\theta^*)$ from a feasible set $\mathcal{X}$ for all the previous steps. Concretely, the regret is defined as:

$$R(T) = \sum_{t=1}^{T} [f_t(\theta_t) - f_t(\theta^*)] \tag{5}$$

where $\theta^* = \arg\min_{\theta \in \mathcal{X}} \sum_{t=1}^{T} f_t(\theta)$. We show Adam has $O(\sqrt{T})$ regret bound and a proof is given in the appendix. Our result is comparable to the best known bound for this general convex online learning problem. We also use some definitions simplify our notation, where $g_t \triangleq \nabla f_t(\theta_t)$ and $g_{t,i}$ as the i^{th} element. We define $g_{1:t,i} \in \mathbb{R}^t$ as a vector that contains the i^{th} dimension of the gradients over all iterations till t, $g_{1:t,i} = [g_{1,i}, g_{2,i}, \cdots, g_{t,i}]$. Also, we define $\gamma \triangleq \frac{\beta_1^2}{\sqrt{\beta_2}}$. Our following theorem holds when the learning rate α_t is decaying at a rate of $t^{-\frac{1}{2}}$ and first moment running average coefficient $\beta_{1,t}$ decay exponentially with λ, that is typically close to 1, e.g. $1 - 10^{-8}$.

Theorem 4.1. *Assume that the function f_t has bounded gradients, $\|\nabla f_t(\theta)\|_2 \le G$, $\|\nabla f_t(\theta)\|_\infty \le G_\infty$ for all $\theta \in R^d$ and distance between any θ_t generated by Adam is bounded, $\|\theta_n - \theta_m\|_2 \le D$, $\|\theta_m - \theta_n\|_\infty \le D_\infty$ for any $m, n \in \{1, ..., T\}$, and β_1, $\beta_2 \in [0, 1)$ satisfy $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$. Let $\alpha_t = \frac{\alpha}{\sqrt{t}}$ and $\beta_{1,t} = \beta_1 \lambda^{t-1}, \lambda \in (0, 1)$. Adam achieves the following guarantee, for all $T \ge 1$.*

$$R(T) \le \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\widehat{v}_{T,i}} + \frac{\alpha(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{d} \|g_{1:T,i}\|_2 + \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha(1-\beta_1)(1-\lambda)^2}$$

Our Theorem 4.1 implies when the data features are sparse and bounded gradients, the summation term can be much smaller than its upper bound $\sum_{i=1}^{d} \|g_{1:T,i}\|_2 << dG_\infty\sqrt{T}$ and $\sum_{i=1}^{d} \sqrt{T\widehat{v}_{T,i}} << dG_\infty\sqrt{T}$, in particular if the class of function and data features are in the form of section 1.2 in (Duchi et al., 2011). Their results for the expected value $\mathbb{E}[\sum_{i=1}^{d} \|g_{1:T,i}\|_2]$ also apply to Adam. In particular, the adaptive method, such as Adam and Adagrad, can achieve $O(\log d\sqrt{T})$, an improvement over $O(\sqrt{dT})$ for the non-adaptive method. Decaying $\beta_{1,t}$ towards zero is important in our theoretical analysis and also matches previous empirical findings, e.g. (Sutskever et al., 2013) suggests reducing the momentum coefficient in the end of training can improve convergence.

Finally, we can show the average regret of Adam converges,

Corollary 4.2. *Assume that the function f_t has bounded gradients, $\|\nabla f_t(\theta)\|_2 \le G$, $\|\nabla f_t(\theta)\|_\infty \le G_\infty$ for all $\theta \in R^d$ and distance between any θ_t generated by Adam is bounded, $\|\theta_n - \theta_m\|_2 \le D$, $\|\theta_m - \theta_n\|_\infty \le D_\infty$ for any $m, n \in \{1, ..., T\}$. Adam achieves the following guarantee, for all $T \ge 1$.*

$$\frac{R(T)}{T} = O(\frac{1}{\sqrt{T}})$$

This result can be obtained by using Theorem 4.1 and $\sum_{i=1}^{d} \|g_{1:T,i}\|_2 \le dG_\infty\sqrt{T}$. Thus, $\lim_{T\to\infty} \frac{R(T)}{T} = 0$.

5 RELATED WORK

Optimization methods bearing a direct relation to Adam are RMSProp (Tieleman & Hinton, 2012; Graves, 2013) and AdaGrad (Duchi et al., 2011); these relationships are discussed below. Other stochastic optimization methods include vSGD (Schaul et al., 2012), AdaDelta (Zeiler, 2012) and the natural Newton method from Roux & Fitzgibbon (2010), all setting stepsizes by estimating curvature from first-order information. The Sum-of-Functions Optimizer (SFO) (Sohl-Dickstein et al., 2014) is a quasi-Newton method based on minibatches, but (unlike Adam) has memory requirements linear in the number of minibatch partitions of a dataset, which is often infeasible on memory-constrained systems such as a GPU. Like natural gradient descent (NGD) (Amari, 1998), Adam employs a preconditioner that adapts to the geometry of the data, since $\widehat{v}_t$ is an approximation to the diagonal of the Fisher information matrix (Pascanu & Bengio, 2013); however, Adam's preconditioner (like AdaGrad's) is more conservative in its adaption than vanilla NGD by preconditioning with the square root of the inverse of the diagonal Fisher information matrix approximation.

RMSProp: An optimization method closely related to Adam is RMSProp (Tieleman & Hinton, 2012). A version with momentum has sometimes been used (Graves, 2013). There are a few important differences between RMSProp with momentum and Adam: RMSProp with momentum generates its parameter updates using a momentum on the rescaled gradient, whereas Adam updates are directly estimated using a running average of first and second moment of the gradient. RMSProp also lacks a bias-correction term; this matters most in case of a value of β_2 close to 1 (required in case of sparse gradients), since in that case not correcting the bias leads to very large stepsizes and often divergence, as we also empirically demonstrate in section 6.4.

AdaGrad: An algorithm that works well for sparse gradients is AdaGrad (Duchi et al., 2011). Its basic version updates parameters as $\theta_{t+1} = \theta_t - \alpha \cdot g_t / \sqrt{\sum_{i=1}^{t} g_t^2}$. Note that if we choose β_2 to be infinitesimally close to 1 from below, then $\lim_{\beta_2 \to 1} \widehat{v}_t = t^{-1} \cdot \sum_{i=1}^{t} g_t^2$. AdaGrad corresponds to a version of Adam with $\beta_1 = 0$, infinitesimal $(1 - \beta_2)$ and a replacement of α by an annealed version $\alpha_t = \alpha \cdot t^{-1/2}$, namely $\theta_t - \alpha \cdot t^{-1/2} \cdot \widehat{m}_t / \sqrt{\lim_{\beta_2 \to 1} \widehat{v}_t} = \theta_t - \alpha \cdot t^{-1/2} \cdot g_t / \sqrt{t^{-1} \cdot \sum_{i=1}^{t} g_t^2} = \theta_t - \alpha \cdot g_t / \sqrt{\sum_{i=1}^{t} g_t^2}$. Note that this direct correspondence between Adam and Adagrad does not hold when removing the bias-correction terms; without bias correction, like in RMSProp, a β_2 infinitesimally close to 1 would lead to infinitely large bias, and infinitely large parameter updates.

6 Experiments

To empirically evaluate the proposed method, we investigated different popular machine learning models, including logistic regression, multilayer fully connected neural networks and deep convolutional neural networks. Using large models and datasets, we demonstrate Adam can efficiently solve practical deep learning problems.

We use the same parameter initialization when comparing different optimization algorithms. The hyper-parameters, such as learning rate and momentum, are searched over a dense grid and the results are reported using the best hyper-parameter setting.

6.1 Experiment: Logistic Regression

We evaluate our proposed method on L2-regularized multi-class logistic regression using the MNIST dataset. Logistic regression has a well-studied convex objective, making it suitable for comparison of different optimizers without worrying about local minimum issues. The stepsize α in our logistic regression experiments is adjusted by $1/\sqrt{t}$ decay, namely $\alpha_t = \frac{\alpha}{\sqrt{t}}$ that matches with our theoratical prediction from section 4. The logistic regression classifies the class label directly on the 784 dimension image vectors. We compare Adam to accelerated SGD with Nesterov momentum and Adagrad using minibatch size of 128. According to Figure 1, we found that the Adam yields similar convergence as SGD with momentum and both converge faster than Adagrad.

As discussed in (Duchi et al., 2011), Adagrad can efficiently deal with sparse features and gradients as one of its main theoretical results whereas SGD is low at learning rare features. Adam with $1/\sqrt{t}$ decay on its stepsize should theoratically match the performance of Adagrad. We examine the sparse feature problem using IMDB movie review dataset from (Maas et al., 2011). We pre-process the IMDB movie reviews into bag-of-words (BoW) feature vectors including the first 10,000 most frequent words. The 10,000 dimension BoW feature vector for each review is highly sparse. As suggested in (Wang & Manning, 2013), 50% dropout noise can be applied to the BoW features during

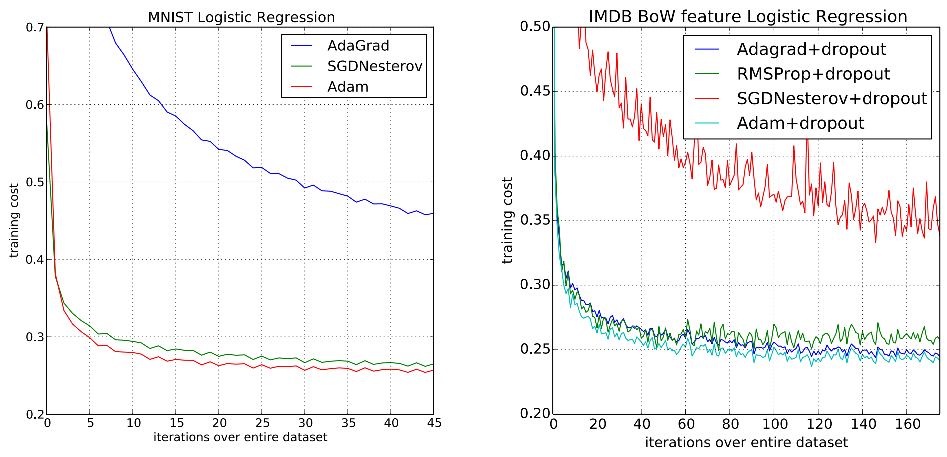


Figure 1: Logistic regression training negative log likelihood on MNIST images and IMDB movie reviews with 10,000 bag-of-words (BoW) feature vectors.

training to prevent over-fitting. In figure 1, Adagrad outperforms SGD with Nesterov momentum by a large margin both with and without dropout noise. Adam converges as fast as Adagrad. The empirical performance of Adam is consistent with our theoretical findings in sections 2 and 4. Similar to Adagrad, Adam can take advantage of sparse features and obtain faster convergence rate than normal SGD with momentum.

6.2 Experiment: Multi-layer Neural Networks

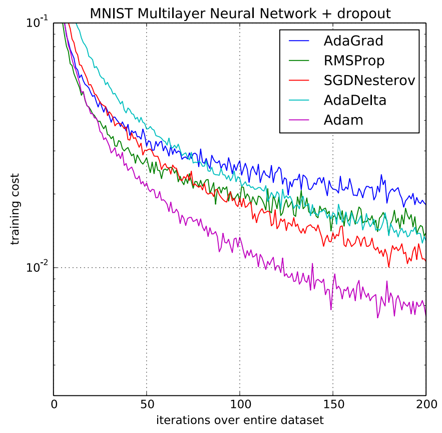
Multi-layer neural network are powerful models with non-convex objective functions. Although our convergence analysis does not apply to non-convex problems, we empirically found that Adam often outperforms other methods in such cases. In our experiments, we made model choices that are consistent with previous publications in the area; a neural network model with two fully connected hidden layers with 1000 hidden units each and ReLU activation are used for this experiment with minibatch size of 128.

First, we study different optimizers using the standard deterministic cross-entropy objective function with L_2 weight decay on the parameters to prevent over-fitting. The sum-of-functions (SFO) method (Sohl-Dickstein et al., 2014) is a recently proposed quasi-Newton method that works with minibatches of data and has shown good performance on optimization of multi-layer neural networks. We used their implementation and compared with Adam to train such models. Figure 2 shows that Adam makes faster progress in terms of both the number of iterations and wall-clock time. Due to the cost of updating curvature information, SFO is 5-10x slower per iteration compared to Adam, and has a memory requirement that is linear in the number minibatches.

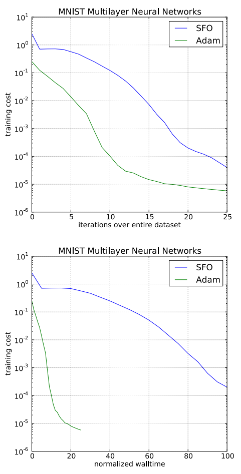
Stochastic regularization methods, such as dropout, are an effective way to prevent over-fitting and often used in practice due to their simplicity. SFO assumes deterministic subfunctions, and indeed failed to converge on cost functions with stochastic regularization. We compare the effectiveness of Adam to other stochastic first order methods on multi-layer neural networks trained with dropout noise. Figure 2 shows our results; Adam shows better convergence than other methods.

6.3 Experiment: Convolutional Neural Networks

Convolutional neural networks (CNNs) with several layers of convolution, pooling and non-linear units have shown considerable success in computer vision tasks. Unlike most fully connected neural nets, weight sharing in CNNs results in vastly different gradients in different layers. A smaller learning rate for the convolution layers is often used in practice when applying SGD. We show the effectiveness of Adam in deep CNNs. Our CNN architecture has three alternating stages of 5x5 convolution filters and 3x3 max pooling with stride of 2 that are followed by a fully connected layer of 1000 rectified linear hidden units (ReLU's). The input image are pre-processed by whitening, and

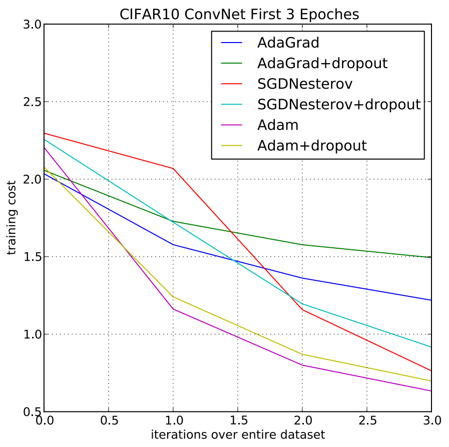


(a)



(b)

Figure 2: Training of multilayer neural networks on MNIST images. (a) Neural networks using dropout stochastic regularization. (b) Neural networks with deterministic cost function. We compare with the sum-of-functions (SFO) optimizer (Sohl-Dickstein et al., 2014)



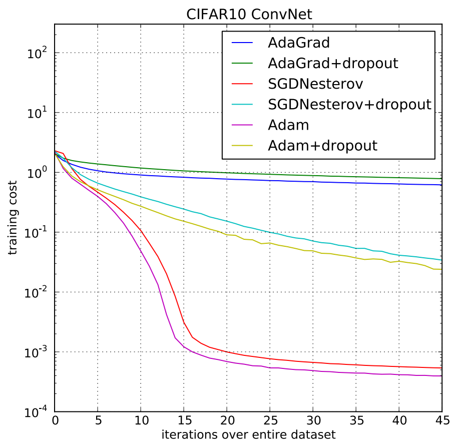
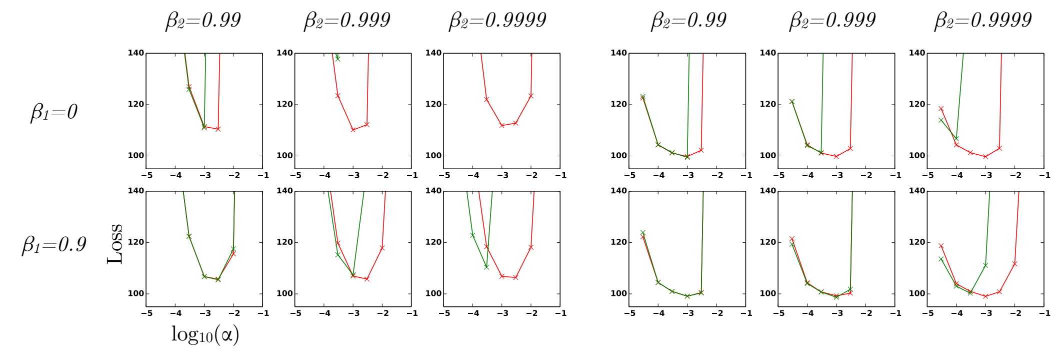


Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

dropout noise is applied to the input layer and fully connected layer. The minibatch size is also set to 128 similar to previous experiments.

Interestingly, although both Adam and Adagrad make rapid progress lowering the cost in the initial stage of the training, shown in Figure 3 (left), Adam and SGD eventually converge considerably faster than Adagrad for CNNs shown in Figure 3 (right). We notice the second moment estimate $\widehat{v}_t$ vanishes to zeros after a few epochs and is dominated by the ϵ in algorithm 1. The second moment estimate is therefore a poor approximation to the geometry of the cost function in CNNs comparing to fully connected network from Section 6.2. Whereas, reducing the minibatch variance through the first moment is more important in CNNs and contributes to the speed-up. As a result, Adagrad converges much slower than others in this particular experiment. Though Adam shows marginal improvement over SGD with momentum, it adapts learning rate scale for different layers instead of hand picking manually as in SGD.



(a) after 10 epochs (b) after 100 epochs

Figure 4: Effect of bias-correction terms (red line) versus no bias correction terms (green line) after 10 epochs (left) and 100 epochs (right) on the loss (y-axes) when learning a Variational Auto-Encoder (VAE) (Kingma & Welling, 2013), for different settings of stepsize α (x-axes) and hyper-parameters β_1 and β_2.

6.4 Experiment: bias-correction term

We also empirically evaluate the effect of the bias correction terms explained in sections 2 and 3. Discussed in section 5, removal of the bias correction terms results in a version of RMSProp (Tieleman & Hinton, 2012) with momentum. We vary the β_1 and β_2 when training a variational auto-encoder (VAE) with the same architecture as in (Kingma & Welling, 2013) with a single hidden layer with 500 hidden units with softplus nonlinearities and a 50-dimensional spherical Gaussian latent variable. We iterated over a broad range of hyper-parameter choices, i.e. $\beta_1 \in [0, 0.9]$ and $\beta_2 \in [0.99, 0.999, 0.9999]$, and $\log_{10}(\alpha) \in [-5, ..., -1]$. Values of β_2 close to 1, required for robustness to sparse gradients, results in larger initialization bias; therefore we expect the bias correction term is important in such cases of slow decay, preventing an adverse effect on optimization.

In Figure 4, values β_2 close to 1 indeed lead to instabilities in training when no bias correction term was present, especially at first few epochs of the training. The best results were achieved with small values of $(1-\beta_2)$ and bias correction; this was more apparent towards the end of optimization when gradients tends to become sparser as hidden units specialize to specific patterns. In summary, Adam performed equal or better than RMSProp, regardless of hyper-parameter setting.

7 Extensions

7.1 AdaMax

In Adam, the update rule for individual weights is to scale their gradients inversely proportional to a (scaled) L^2 norm of their individual current and past gradients. We can generalize the L^2 norm based update rule to a L^p norm based update rule. Such variants become numerically unstable for large p. However, in the special case where we let $p \to \infty$, a surprisingly simple and stable algorithm emerges; see algorithm 2. We'll now derive the algorithm. Let, in case of the L^p norm, the stepsize at time t be inversely proportional to $v_t^{1/p}$, where:

$$v_t = \beta_2^p v_{t-1} + (1-\beta_2^p)|g_t|^p \tag{6}$$

$$= (1-\beta_2^p)\sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p \tag{7}$$