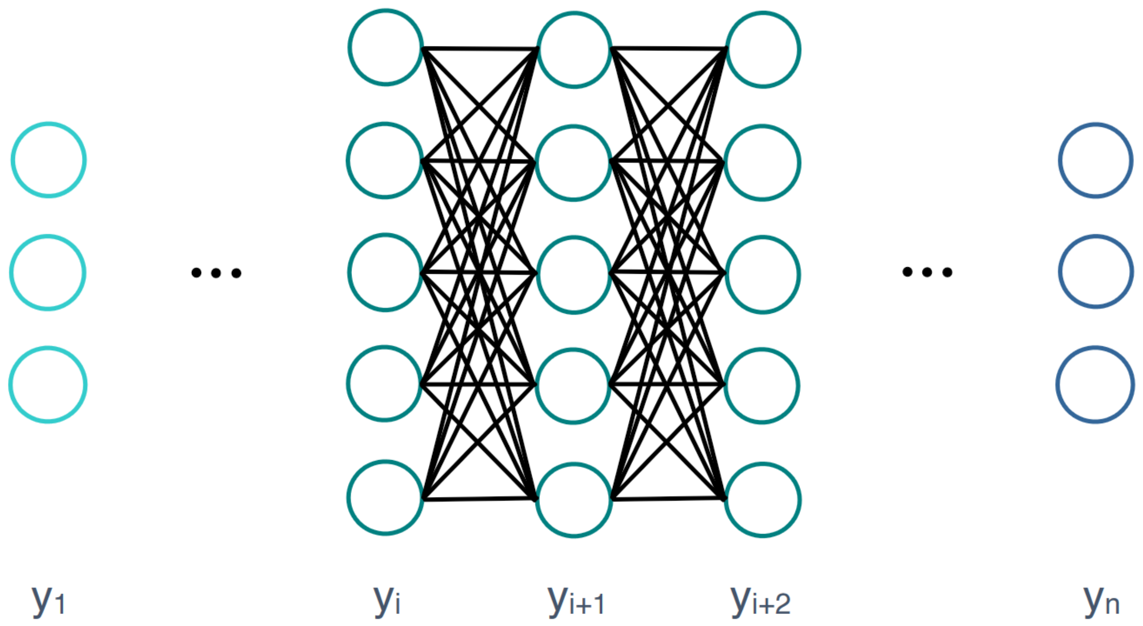
# ADAM: A method for stochastic optimization

Kingma & Ba, 2014 (158k citations)

Mattea Busato, Stefano Mauloni, Daniyar Zakarin

16th November, 2023

# Brief Recap on Neural Networks:




The **loss function** is a function of the input data, the network weights and the expected output.

The loss function therefore **defines the task** for which the model is intended and the weights **need to be trained** to fit the task.

# Gradient descent variants:

$$\theta_t = \theta_{t-1} - \eta * \boxed{\frac{\partial L}{\partial \theta_{t-1}}}$$

(N total datapoints)

→ N data points → **Classic Gradient Descent**

→ 1 data point → **Stochastic Gradient Descent**

→ k data points → **Mini Batch SGD**

Depending on the amount of data, we make a **trade-off** between the **accuracy** of the weights' update and the **computational time** it takes to perform an update.

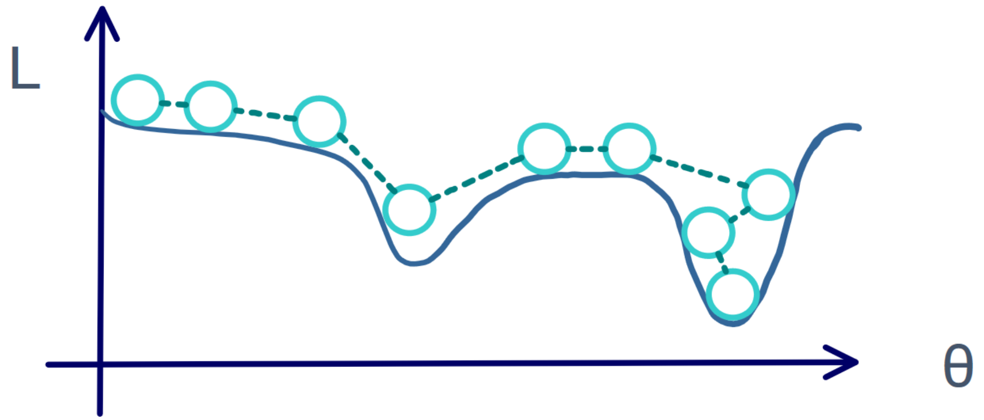
# SGD with Momentum:

Update rule: $\begin{cases} \theta_t = \theta_{t-1} + v_t \\ v_t = \rho v_{t-1} - \eta g(\theta_{t-1}) \end{cases} \implies \theta_t = \theta_{t-1} + \rho v_{t-1} - \eta g(\theta_{t-1})$

We introduce **momentum** $\rho$:

1. It helps accelerate SDG in the relevant direction
2. It smooths out the trajectory by mitigating the stochasticity of the motion of $\theta$ in parameter space

The choice of learning rate $\eta$ is made a bit **less crucial** by the fact that the training procedure is **adaptive** to the particular loss landscape.

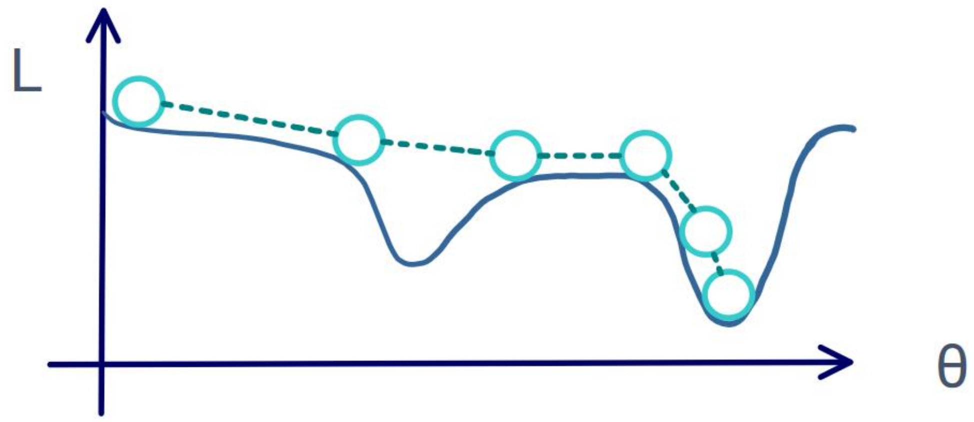


# AdaGrad (Adaptive Gradient):

Update rule: $$\theta_{i,t} = \theta_{i,t-1} - \frac{\eta}{\varepsilon + \sqrt{\sum_{j=1}^{t-1} g(\theta_{i,j})^2}} * g(\theta_{i,t-1})$$

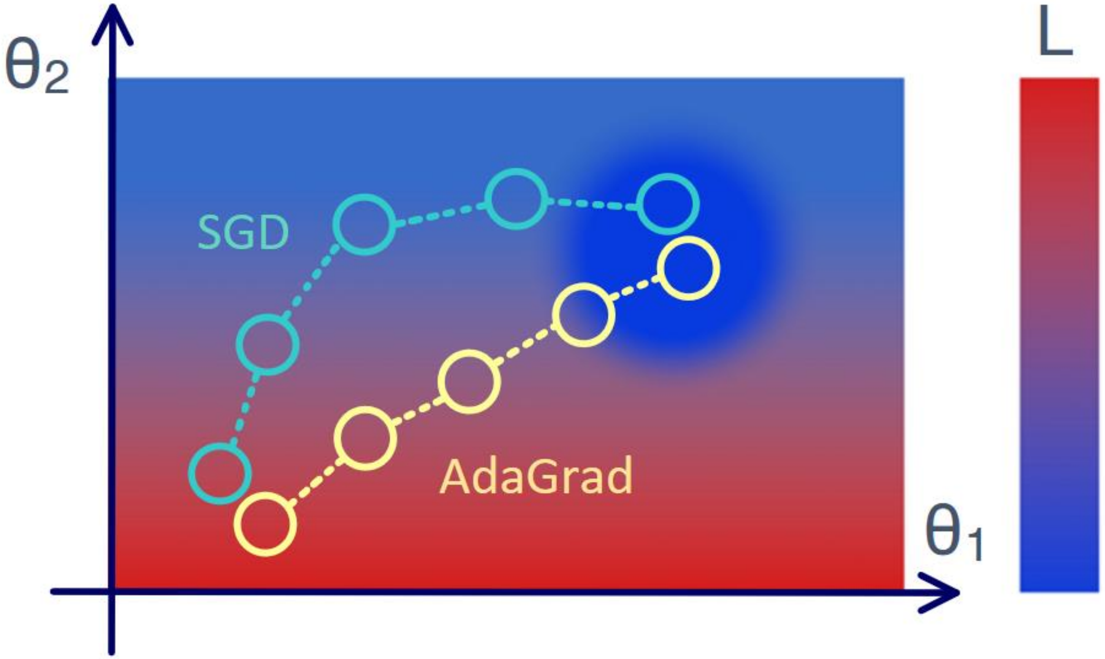
To simplify the notation: $$\begin{cases} \theta_{i,t} = \theta_{i,t-1} - \dfrac{\eta}{\varepsilon + \sqrt{v_{i,t}}} * g(\theta_{i,t-1}) \\ v_{i,t} = v_{i,t-1} + g(\theta_{i,t-1})^2 \end{cases}$$

AdaGrad adapts the learning rate of **each parameter** differently as training progresses.



# AdaGrad (Adaptive Gradient):

The **idea** is that if a parameter has changed significantly, it must have made a lot of progress towards the target and if it has not changed much, it should keep getting updated with greater emphasis.



# RMSProp (Root Mean Square Propagation)

Update rule:

$$\begin{cases} \theta_t = \theta_{t-1} - \dfrac{\eta}{\varepsilon + \sqrt{v_t}} * g(\theta_{t-1}) \\ v_t = \beta v_{t-1} + (1-\beta) g(\theta_{t-1})^2 \end{cases}$$

We introduce a **discount parameter** $\beta \in [0, 1]$:

1. It controls how much of the previous term $v_{t-1}$ is remembered
2. It allows the scaling down and scaling up of the learning rate

We retain the benefits of the **decaying learning** rate without the risk of suffering a permanently decayed rate

# What is ADAM?

- ADAM is a stochastic gradient-descent optimization method.
- It consists, as we have seen with other methods, in updating weights based on the value of the gradient computed at each time step.

## Characteristics:

1. Straightforward to implement
2. Computationally efficient
3. Uses little memory
4. Adapts to the parameters landscape

# What is ADAM?

Bocconi AI&Neuroscience Student Association

- ADAM is a stochastic gradient-descent optimization method.
- It consists, as we have seen with other methods, in updating weights based on the value of the gradient computed at each time step.

## Requirements:

- Stochastic objective function
- Function must be differentiable wrt its parameters $\theta$

# Notation

- $\theta$ = weights
- $v$ = velocity
- $\rho, \eta$ = fixed parameters
- $g(\theta) = \nabla_\theta f(\theta)$ = gradient of $f$ wrt $\theta$

# Update rule of "Classic" SGD with momentum:

$$\begin{cases} \theta_t = \theta_{t-1} + v_t \\ v_t = \rho v_{t-1} - \eta g(\theta_{t-1}) \end{cases} \Longrightarrow \theta_t = \theta_{t-1} + \rho v_{t-1} - \eta g(\theta_{t-1})$$

# Additional Notation

- $\beta_1,\ \beta_2$ = discount parameters (how much of the previous is remembered)
- $m$ = first moment estimate
- $v$ = second moment estimate
- $\epsilon$ = numerical stabilizer (avoids divisions by zero)

# Update rule of ADAM:

- 1st moment vector:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g(\theta_{t-1})$$

- 2nd moment vector:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g(\theta_{t-1})^2$$

$$\Longrightarrow \quad \theta_t = \theta_{t-1} - \frac{\alpha}{\epsilon + \sqrt{\hat{v}_t}} \hat{m}_t$$

# Why $\hat{v}_t$ and $\widehat{m}_t$?

The algorithm computes **exponential moving** (not just decaying) averages of the gradient ($m_t$) and the squared gradient ($v_t$).

We need to inizialize $\theta_0$, usually as a vector of zeros. Hence, $m_t$ and $v_t$ will be **biased towards 0** $\in \mathbb{R}^{\mathbf{n}}$ (initialization bias).

$m_t$ and $v_t$ are biased moment estimates. Bias is stronger:

- At initial timesteps (small $t$)
- When decay rates are small ($\beta_1,\ \beta_2$ close to 1)

To solve this, ADAM introduces **bias-corrected** 1st and 2nd moment estimates:

$$\widehat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{\mathbf{1} - \boldsymbol{\beta}_1{}^t} \qquad \widehat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{\mathbf{1} - \boldsymbol{\beta}_2{}^t}$$

**NB**: They are magnification of the standard estimates for small t, while they tend to be equal to the biased as t becomes larger and larger

# Algorithm

Good default settings: $\alpha = 0.1,\ \ \beta_1 = 0.9,\ \ \beta_2 = 0.999,\ \epsilon =\ 10^{-8}$

Require: $\alpha,\ \beta_1,\ \beta_2,\ f(\theta),\ \theta_0$
Require: $m_0 \longleftarrow 0,\ v_0 \longleftarrow 0,\ t \longleftarrow 0$

While $\theta_t$ not converged do:

$$t \longleftarrow t + 1$$

$$g_t \longleftarrow \nabla_\theta f_t(\theta_{t-1})$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

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

$$\widehat{m}_t = m_t / (1 - {\beta_1}^t)$$

$$\hat{v}_t = v_t / (1 - {\beta_2}^t)$$

$$\theta_t = \theta_{t-1} - \frac{\alpha\, \widehat{m}_t}{\epsilon + \sqrt{\hat{v}_t}}$$

Return $\theta_t$

# Update Rule: 3 Properties

## 1: Boundedness

Assuming $\epsilon = 0$, the effective stepsize in parameter space is:

$$|\Delta_t| = \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}}$$

It is **bounded** by two upper bounds:

$$\begin{cases} |\Delta_t| \leq \alpha \dfrac{(1-\beta_1)}{\sqrt{(1-\beta_2)}} & if \;\; (1-\beta_1) > \sqrt{(1-\beta_2)} \\ |\Delta_t| \leq \alpha & if \;\; (1-\beta_1) \leq \sqrt{(1-\beta_2)} \end{cases}$$

Since the first happens only in very peculiar cases (sparse gradients), in general we can assume $|\Delta_t| \lessapprox \boldsymbol{\alpha}$

NB: It establishes a *trust region* outside of which the current gradient does not provide sufficient information. Hence, **we never move outside of it**:

# 2: Invariance to rescaling of the gradients

Multiplying $g$ by a factor c will make no difference in the magnitude of the effective stepsize.

Rescaling the gradients $g$ by a factor $c$:

$$\boldsymbol{m_t'} = \beta_1 m_{t-1}' + (1-\beta_1) c g_{t-1} = \beta_1 c m_{t-1} + (1-\beta_1) c g_{t-1} = \boldsymbol{c m_t}$$

$$\boldsymbol{v_t'} = \beta_2 {v'}_{t-1} + (1-\beta_2)(c g_{t-1})^2 = \beta_2 c^2 v_{t-1} + (1-\beta_2) c^2 (g_{t-1})^2 = \boldsymbol{c^2 v_t}$$

$$|\Delta_{\boldsymbol{t}}|' = \alpha \frac{\widehat{m_t}'}{\sqrt{\widehat{v_t}'}} = \alpha \frac{m_t'}{\sqrt{v_t'}} \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} = \alpha \frac{c m_t}{\sqrt{c^2 v_t}} \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} = \alpha \frac{m_t}{\sqrt{v_t}} \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} = |\Delta_{\boldsymbol{t}}|$$

# 3: “Cautiousness”

We define signal-to-noise ratio (SNR) $:= \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}}$

SNR $\approx 0 \implies \Delta_t \approx 0$

It is desirable, since this means that there is uncertainty whether the true direction of the gradient corresponds to the one of $\widehat{m_t}$.

This usually happens near a point of minimum, so this prevents jumping around it.

# Initialization Bias Correction

Bocconi AI & Neuroscience Student Association

**Derivation of $\boldsymbol{v_t}$** (analogous for $\boldsymbol{m_t}$)

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

2: algebra:

$$\mathbb{E}(v_t) = \mathbb{E}\left((1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\cdot g_i^2\right) = \mathbb{E}(g_t^2)(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i} + \zeta$$

$$= \mathbb{E}(g_t^2)(1-\beta_2^t) + \zeta$$ bias

$\zeta = 0$ if the gradient is stationary (namely, constant), otherwise it is still very close to zero since gradients far in the past have small values. **We ignore it**.

# Initialization Bias Correction

**Example:** (with $\boldsymbol{m_t}$)

$t = 1$:

$$m_1 = \beta_1 m_0 + (1 - \beta_1) g_1$$

We take out $\beta_1 m_0$ and divide by $(1 - \beta_1)$ to get the same expectation as $g_1$,
This yields:

$$\hat{m}_1 = \frac{m_1 - \beta_1 m_0}{(1 - \beta_1)}\,, \qquad \mathbb{E}(\hat{m}_1) = \mathbb{E}(g_1)$$

But $\beta_1 m_0 = 0$, thus:

$$\hat{m}_1 = \frac{m_1}{(1 - \beta_1)}$$

Note that in this trivial case $\zeta = 0$.

# Does it actually converge?

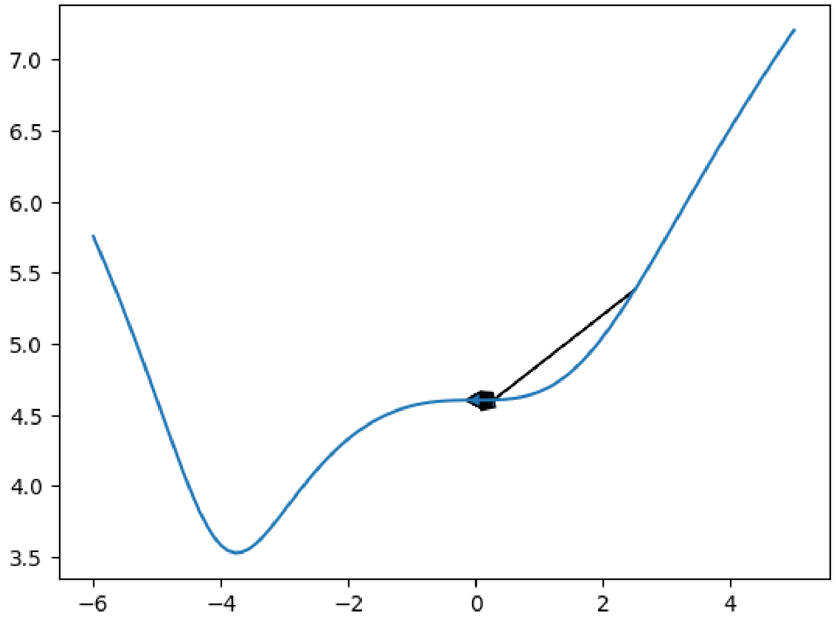
**Yes** (proved in the paper), assuming bounded gradients.

**Not always** in practice. It depends at a great extent to the choice of $\beta_2$.

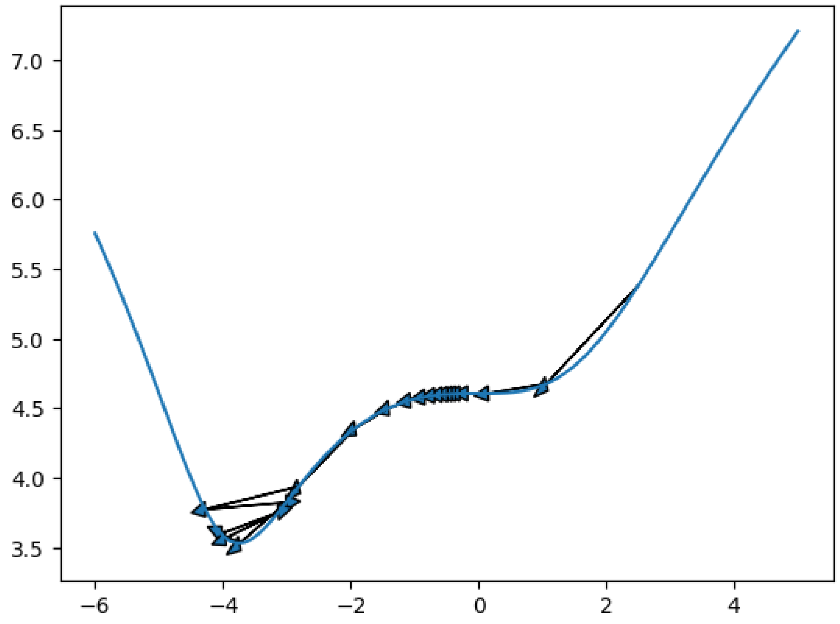
In general, we take $\beta_2$ large enough (very close to 1) to ensure convergence.

# Power of Momentum: Acceleration

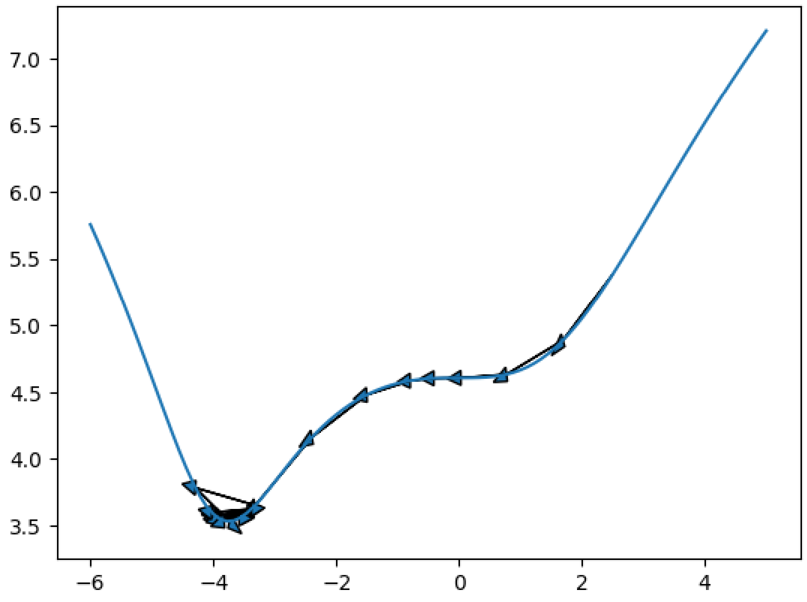
Gradient Descent



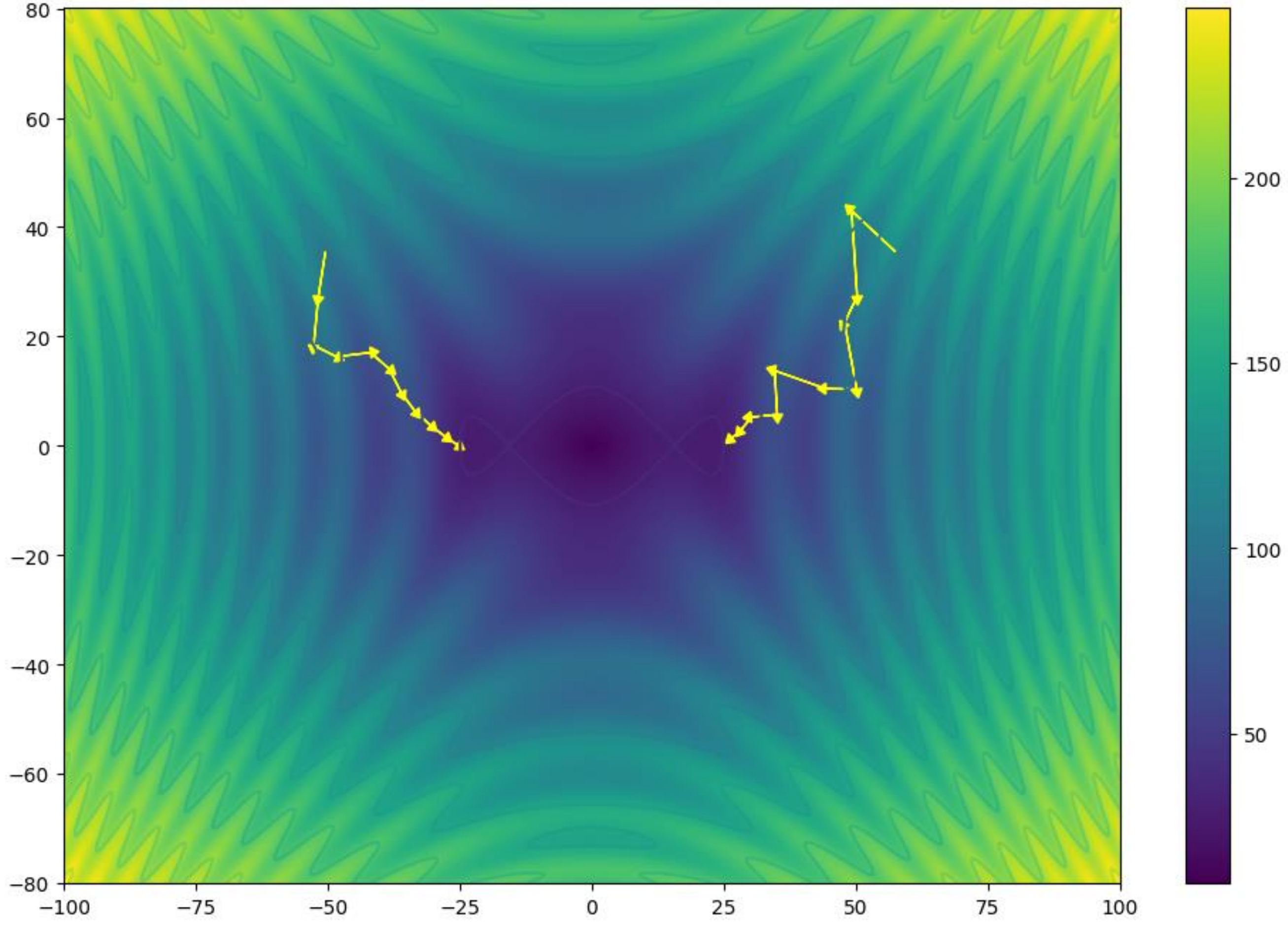

Gradient Descent with Momentum



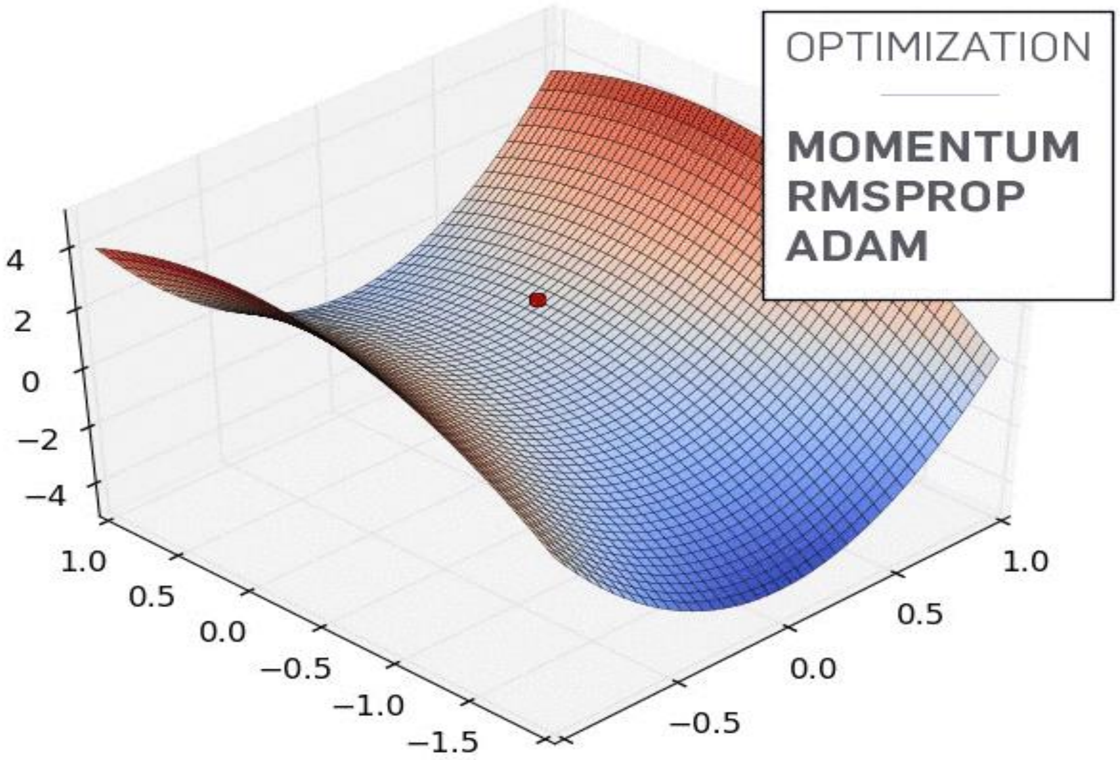

ADAM



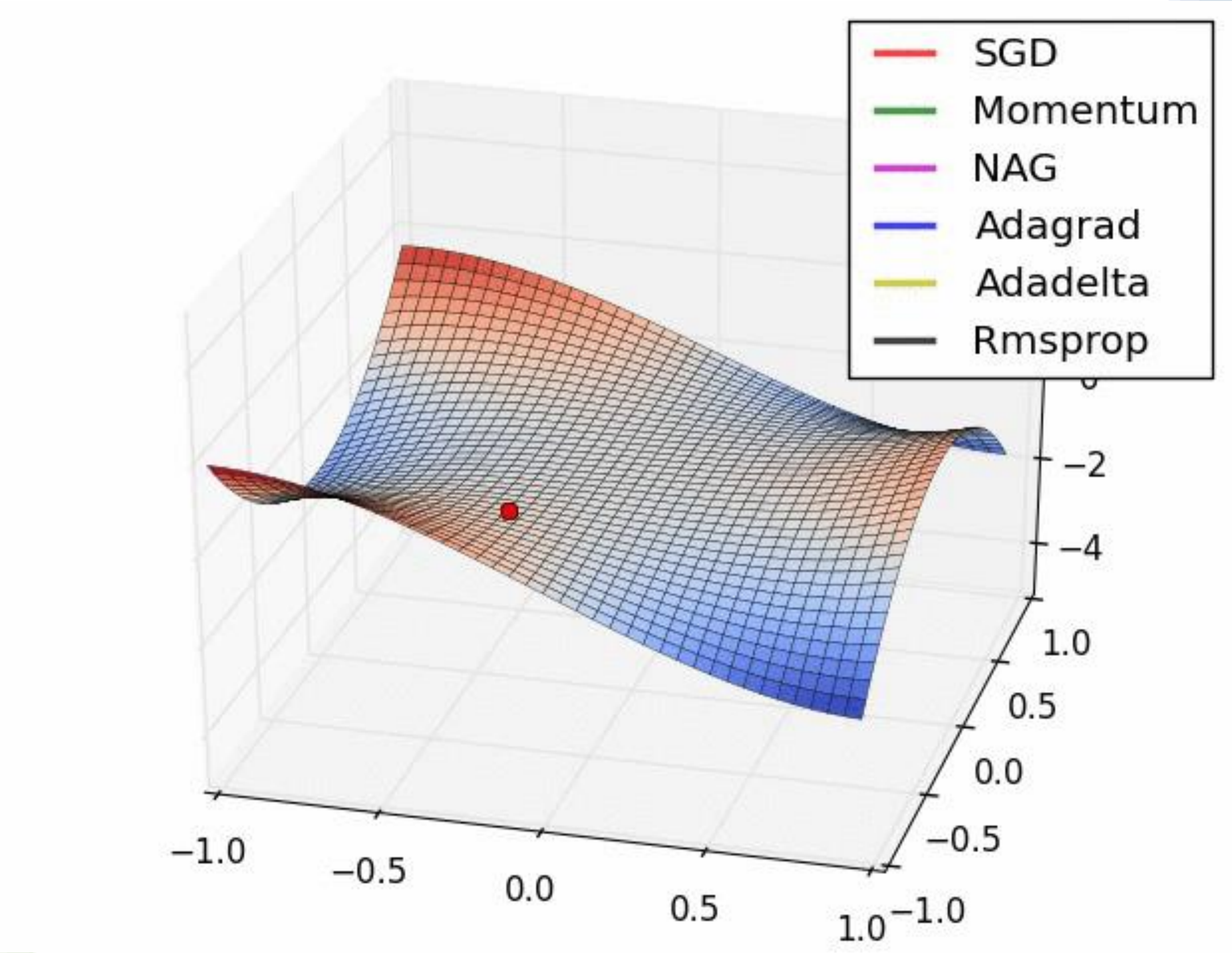
# Power of Momentum: Noise Cancelling



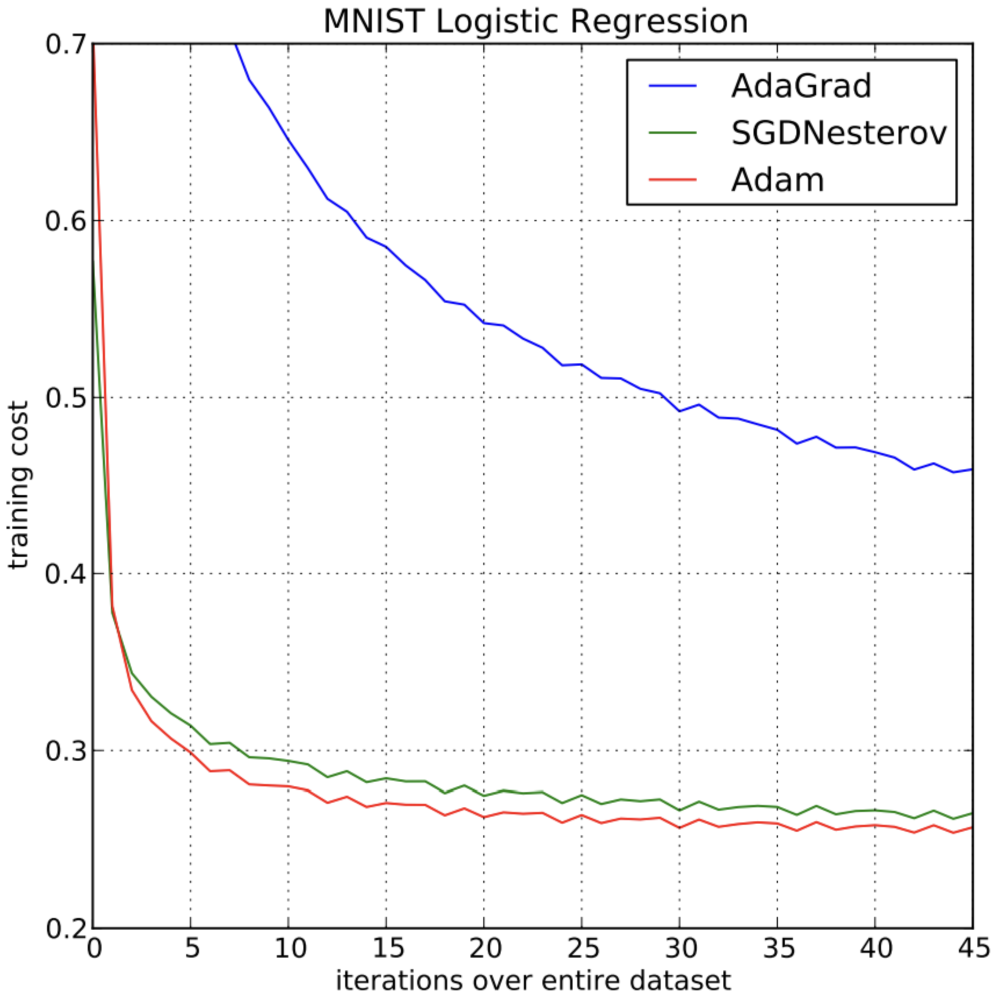
# Second moment helps with sparsity



# Second moment helps with sparsity



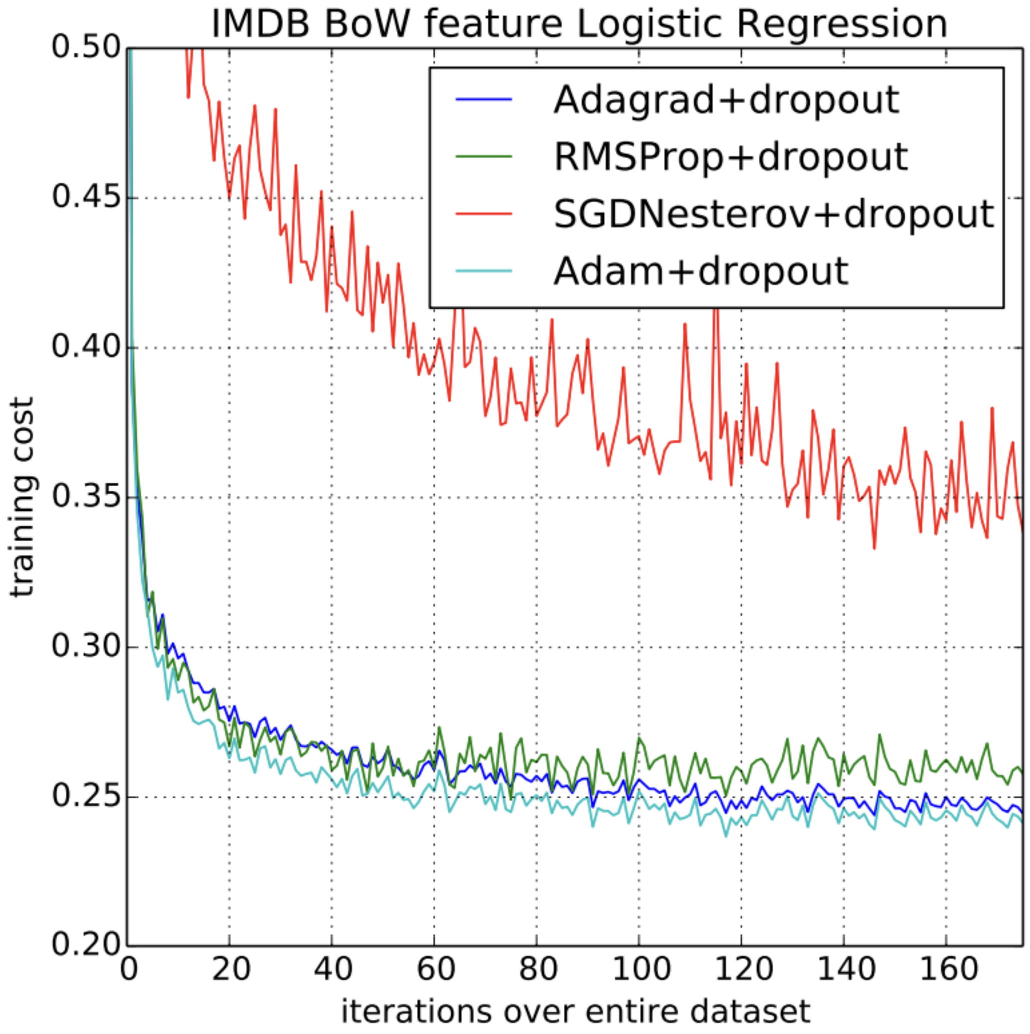
# Logistic Regression: MNIST




- Authors use L2-Regularized Multi Class Logistic regression on MNIST image dataset

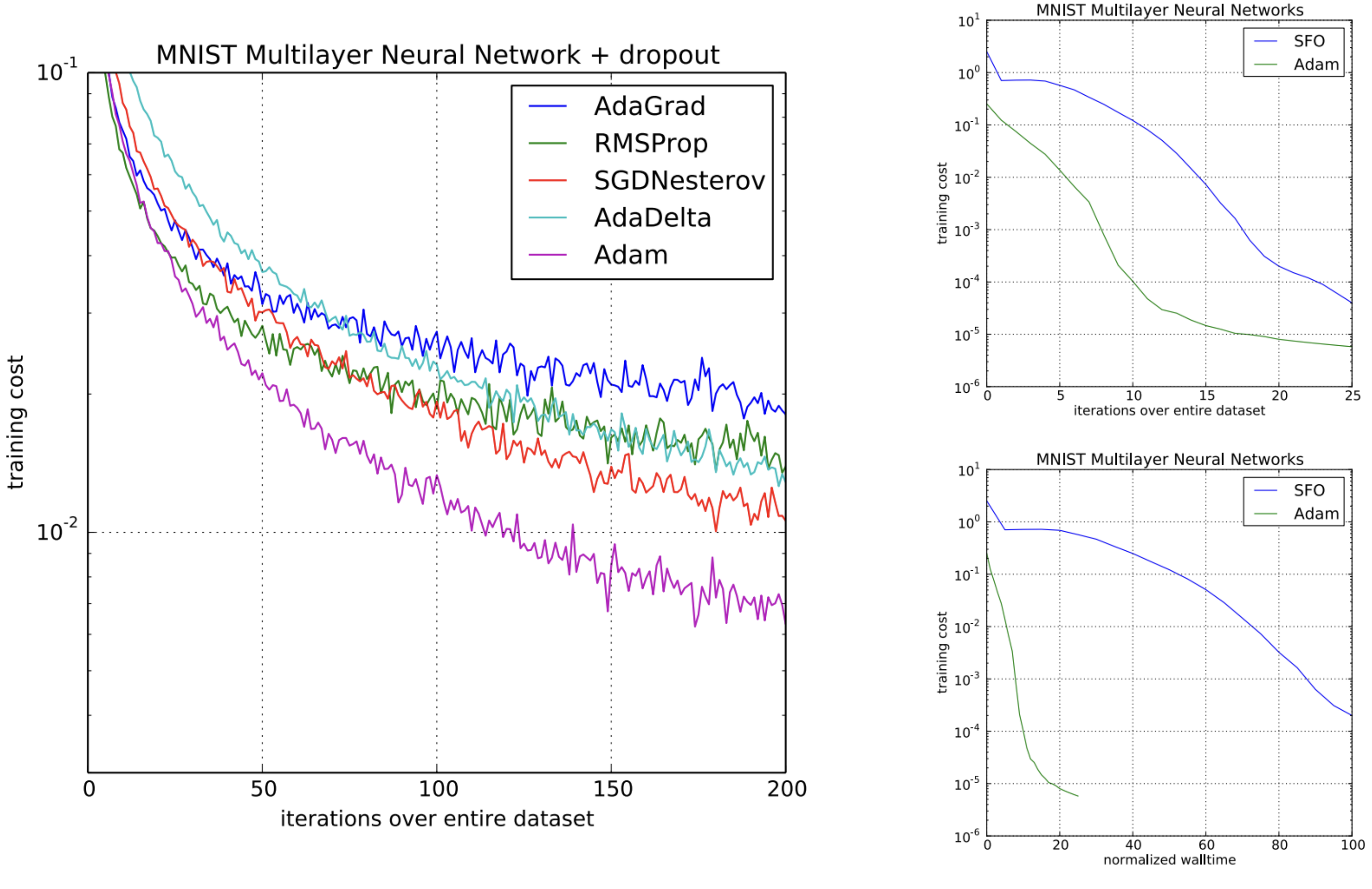
- 1 / sqrt(t) learning rate decay was used in agreement with theoretical convergence result

# Logistic Regression: IMDB reviews




- IMDB movie reviews were preprocessed to bag-of-words (BoW) feature vectors including 10000 most frequent words
- Adagrad performs well compared to SGD on sparse data and sparse gradients
- Notice blue & pink lines are smoother

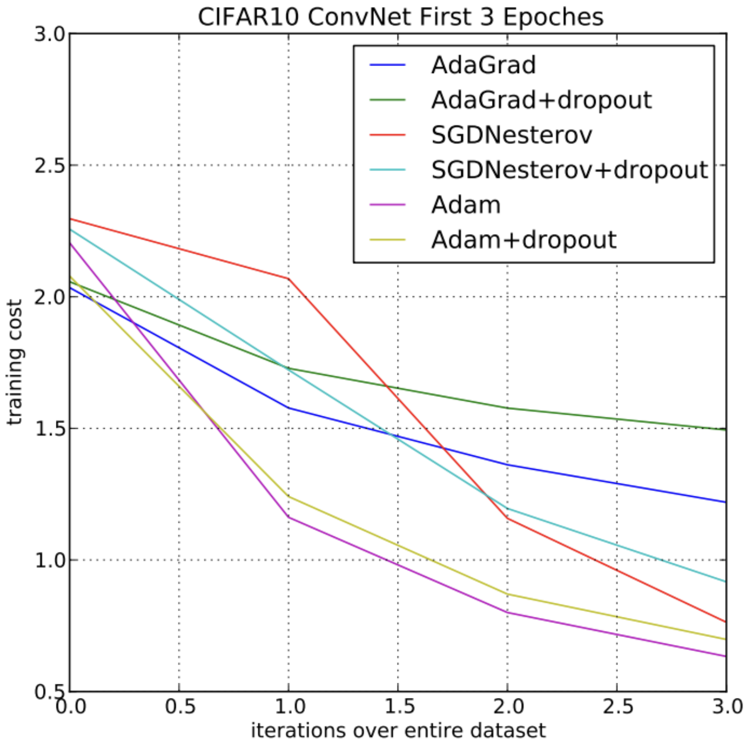
# Feed-forward Neural Networks

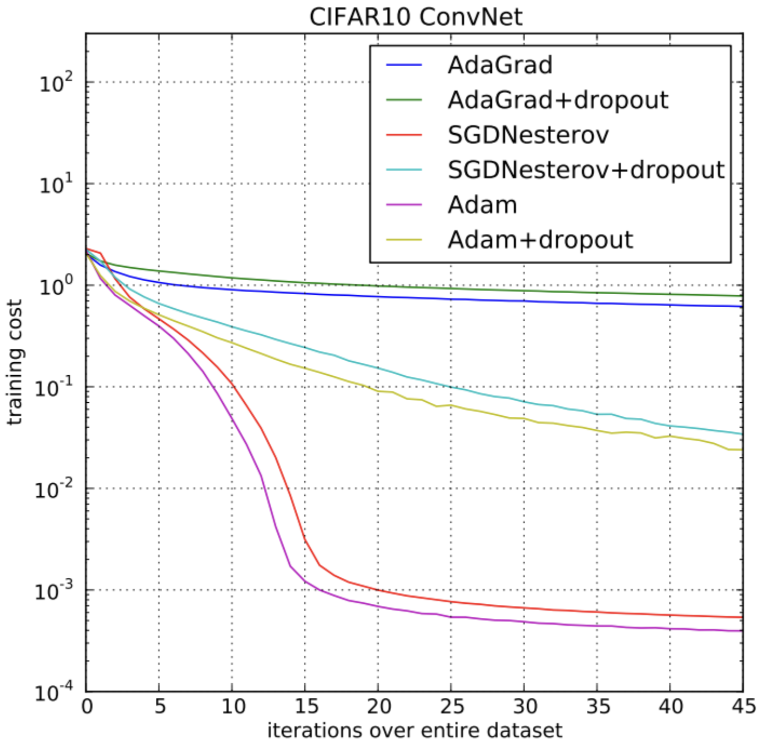



(a) (b)

- The Sum of Functions (SFO) method quasi-Newton method that works with minibatches of data (Sohl-Dickstein et al., 2014)

# Convolutional Neural Networks




- Second momentum vanishes
- Adam does not require handpicked layer learning rates

# ADAM after 2014

- Attention is All you Need, 2017
- U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019
- An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2020

"ADAM does well where others fail, although not without drawbacks..." © Alex from Stack Exchange

Bocconi
AI&Neuroscience
Student
Association

# People behind ADAM

# Thanks

Mattea Busato, Stefano Mauloni, Daniyar Zakarin