An Overview of Gradient Descent Optimization Algorithms

Matteo Sordello

University of Pennsylvania

Credits: A big part of this presentation is borrowed from Sebastian Ruder

October 18, 2018

Expected vs. Empirical Loss

- $(x_i, y_i) \in \mathbb{R}^{d_x} \times \mathbb{R}$ for $i = 1, ..., n$, we call x the input and y the output.
- $f(\theta, x, y)$ is the loss (risk) function. Two popular examples
  - Linear regression: $f(\theta, x, y) = (y - x \cdot \theta)^2$
  - Logistic regression: $f(\theta, x, y) = \log\left(1 + e^{-yx \cdot \theta}\right)$

Expected vs. Empirical Loss

- $(x_i, y_i) \in \mathbb{R}^{d_x} \times \mathbb{R}$ for $i = 1, ..., n$, we call x the input and y the output.
- $f(\theta, x, y)$ is the loss (risk) function. Two popular examples
  - Linear regression: $f(\theta, x, y) = (y - x \cdot \theta)^2$
  - Logistic regression: $f(\theta, x, y) = \log\left(1 + e^{-yx\cdot\theta}\right)$
- The **expected** loss is

$$L(\theta) = \mathbb{E}_{(x,y)}\left[f(\theta, x, y)\right]$$

Expected vs. Empirical Loss

- $(x_i, y_i) \in \mathbb{R}^{d_x} \times \mathbb{R}$ for $i = 1, ..., n$, we call x the input and y the output.
- $f(\theta, x, y)$ is the loss (risk) function. Two popular examples
  - Linear regression: $f(\theta, x, y) = (y - x \cdot \theta)^2$
  - Logistic regression: $f(\theta, x, y) = \log\left(1 + e^{-yx\cdot\theta}\right)$
- The **expected** loss is

$$L(\theta) = \mathbb{E}_{(x,y)}\left[f(\theta, x, y)\right]$$

- The **empirical** loss is

$$L_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} f(\theta, x_i, y_i) := \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$$

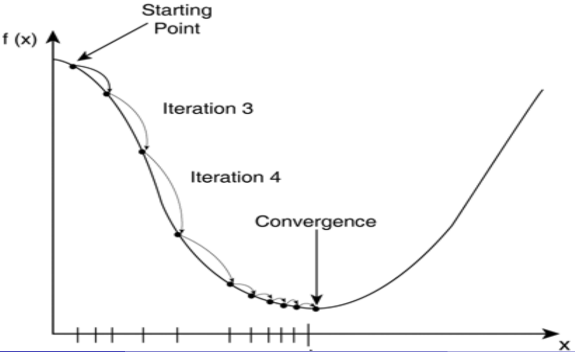
Gradient descent

- Gradient descent is a way to minimize an objective function $F(\theta)$.
  - $\theta \in \mathbb{R}^d$ are the model parameters
  - η_t is the learning rate at iteration t.
  - $\nabla_\theta F(\theta)$ is the gradient of the objective function with respect to the parameters

Gradient descent

- Gradient descent is a way to minimize an objective function $F(\theta)$.
  - $\theta \in \mathbb{R}^d$ are the model parameters
  - η_t is the learning rate at iteration t.
  - $\nabla_\theta F(\theta)$ is the gradient of the objective function with respect to the parameters
- The parameters get updated in the **opposite** direction of the gradient, following the equation

$$\theta_{t+1} = \theta_t - \eta_t \cdot \nabla_\theta F(\theta_t)$$



Variants

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

They only differ in the amount of data used for each update.

Batch gradient descent

At every iteration we compute the gradient using the whole dataset. The update equation is

$$\theta_{t+1} = \theta_t - \eta_t \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta f_i(\theta_t)$$

Batch gradient descent

At every iteration we compute the gradient using the whole dataset. The update equation is

$$\theta_{t+1} = \theta_t - \eta_t \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta f_i(\theta_t)$$

- **Pros**: Guaranteed to converge to global minimum for convex error surfaces and to a local minimum for non-convex surfaces.
- **Cons**:
  - Very slow
  - Intractable for datasets that do not fit in memory
  - No online learning

Stochastic gradient descent

We compute an update for each data point. The update equation is

$$\theta_{t+1} = \theta_t - \eta_t \cdot \nabla_\theta f_{i_t}(\theta_t)$$

where $i_t \in \{1, .., n\}$.

Stochastic gradient descent

We compute an update for each data point. The update equation is

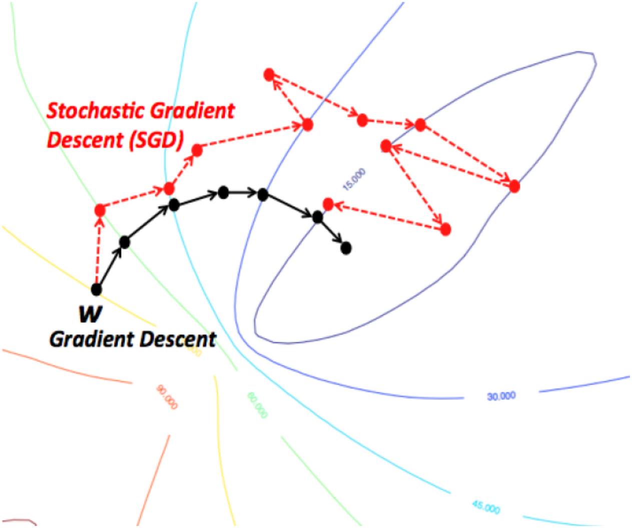
$$\theta_{t+1} = \theta_t - \eta_t \cdot \nabla_\theta f_{i_t}(\theta_t)$$

where $i_t \in \{1, .., n\}$.

- **Pros**:
  - Much faster than batch gradient descent
  - Allows online learning
- **Cons**: High variance updates.

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

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



Mini-batch gradient descent

At every gradient update a mini-batch of b data points is used. Let B be the set indices, the update equation is

$$\theta_{t+1} = \theta_t - \eta_t \cdot \frac{1}{b} \sum_{i \in B} \nabla_\theta f_i(\theta_t)$$

Mini-batch gradient descent

At every gradient update a mini-batch of b data points is used. Let B be the set indices, the update equation is

$$\theta_{t+1} = \theta_t - \eta_t \cdot \frac{1}{b} \sum_{i \in B} \nabla_\theta f_i(\theta_t)$$

- **Pros**: Reduces variance of updates
- **Cons**: Mini-batch size is a hyperparameter. Common sizes are 50-256

- This is the algorithm used in practice most of the times in real applications.

Method	Accuracy	Update Speed	Memory Usage	Online Learning
Batch gradient descent	Good	Slow	High	No
Stochastic gradient descent	Good (with annealing)	High	Low	Yes
Mini-batch gradient descent	Good	Medium	Medium	Yes

Table: Comparison of trade-offs of gradient descent variants

Challenges

- Choosing a learning rate η_t (constant or decreasing? How fast?)

Challenges

- Choosing a learning rate η_t (constant or decreasing? How fast?)
- Updating features to different extent

Challenges

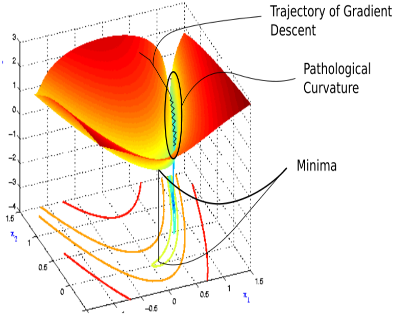
- Choosing a learning rate η_t (constant or decreasing? How fast?)
- Updating features to different extent
- Avoiding suboptimal minima

Gradient descent optimization algorithms

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

Ravine

A ravine is a slope landform of relatively steep (cross-sectional) sides.



Momentum

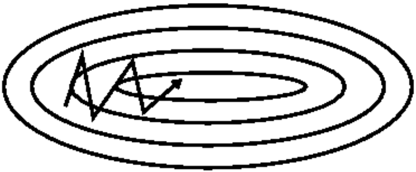
- SGD has trouble navigating ravines
- Momentum [Qian, 1999] helps SGD accelerate
- A fraction γ of the update vector at step $t-1$ is added to the gradient

$$v_t = \gamma v_{t-1} + \eta_t \nabla_\theta f(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_t$$

or equivalently $\theta_{t+1} = \theta_t - \eta_t \nabla_\theta f(\theta_t) - \gamma(\theta_{t-1} - \theta_t)$

Figure: SGD without momentum

Figure: SGD with momentum



Advantages of momentum

IDEA: a ball rolling down a hill accumulating momentum

- Reduces updates for dimensions whose gradients change directions
- Increases updates for dimensions whose gradients point in the same directions

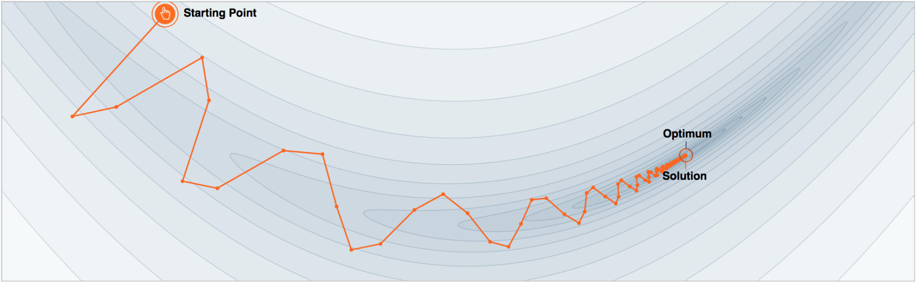


Figure: Optimization with momentum (Source: distill.pub)

Nesterov accelerated gradient

Problem with momentum: the ball has no notion of where it's going

- Momentum blindly accelerates down slopes: First computes gradient, then makes a big jump

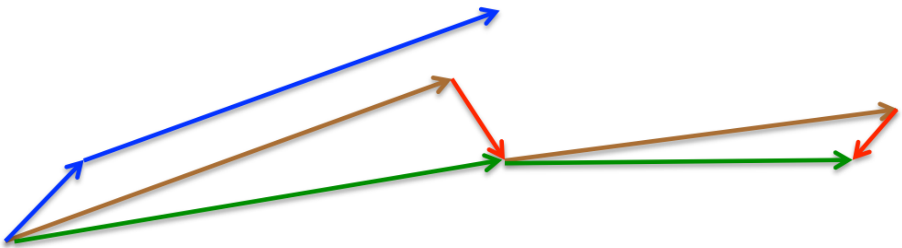
Nesterov accelerated gradient

Problem with momentum: the ball has no notion of where it's going

- Momentum blindly accelerates down slopes: First computes gradient, then makes a big jump
- Nesterov accelerated gradient (NAG) [Nesterov, 1983] first makes a big jump in the direction of the previous accumulated gradient $\theta_t - \gamma v_{t-1}$. Then measures where it ends up and makes a correction, resulting in the complete update vector.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta_t \nabla_\theta f(\theta_t - \gamma v_{t-1}) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned}$$

Nesterov update



- small blue vector: current gradient
- brown vector: previous accumulated gradient
- big blue vector: current accumulated gradient
- red vector: gradient after the big jump
- green vector: NAG update

Adagrad

Notation: $g_t = \nabla_\theta f(\theta_t)$.

Problem: up to now the learning rate η_t was the same for each parameter.

Adagrad

Notation: $g_t = \nabla_\theta f(\theta_t)$.

Problem: up to now the learning rate η_t was the same for each parameter.

- Adagrad [Duchi et al., 2011] **adapts** the learning rate to the parameters (large updates for infrequent parameters, small updates for frequent parameters)
- It divides the learning rate by the square root of the sum of squares of historic gradients. Its update is

$$\theta_{t+1} = \theta_t - \frac{\eta_t}{\sqrt{G_t + \epsilon}} \cdot g_t$$

  where G_t is the diagonal matrix where each entry is the sum of the squares of the gradients up to time t.
- ϵ is a smoothing term to avoid division by 0.

- **Pros**:
  - Well-suited for dealing with sparse data
  - Significantly improves robustness of SGD
  - Lesser need to manually tune learning rate
- **Cons**: Accumulates squared gradients in denominator. Causes the learning rate to shrink and become infinitesimally small

Adadelta

Idea: only keep a window of accumulated past squared gradients (**inefficient**)

- Defines running average of squared gradients $avg(g^2)_t$ at time t with

$$avg(g^2)_t = \gamma \cdot avg(g^2)_{t-1} + (1-\gamma)g_t^2$$

  The parameter γ is similar to the momentum term, usually $\gamma = 0.9$.

- The preliminary Adadelta upgrade is then

$$\theta_{t+1} = \theta_t - \frac{\eta_t}{\sqrt{avg(g^2)_t + \epsilon}} \cdot g_t$$

Units matching with Adadelta

Note

As well as in SGD, Momentum, or Adagrad, hypothetical units do not match. If $f(\theta)$ is unitless, then

$$g = \frac{\partial f(\theta)}{\partial \theta} \propto \frac{1}{\text{units of } \theta} \quad \Rightarrow \quad \frac{\eta_t}{\sqrt{avg(g^2)_t + \epsilon}} \cdot g_t \quad \text{is unitless}$$

Units matching with Adadelta

Note

As well as in SGD, Momentum, or Adagrad, hypothetical units do not match. If $f(\theta)$ is unitless, then

$$g = \frac{\partial f(\theta)}{\partial \theta} \propto \frac{1}{\text{units of } \theta} \quad \Rightarrow \quad \frac{\eta_t}{\sqrt{avg(g^2)_t + \epsilon}} \cdot g_t \quad \text{is unitless}$$

- Define $\Delta\theta_t = \theta_{t+1} - \theta_t$ and a running average of squared parameter updates

$$avg(\Delta\theta^2)_t = \gamma \cdot avg(\Delta\theta^2)_{t-1} + (1-\gamma)\Delta\theta_t^2$$

- The final Adadelta update is

$$\theta_{t+1} = \theta_t - \frac{\sqrt{avg(\Delta\theta^2)_{t-1} + \epsilon}}{\sqrt{avg(g^2)_t + \epsilon}} \cdot g_t$$

Comments on Adadelta

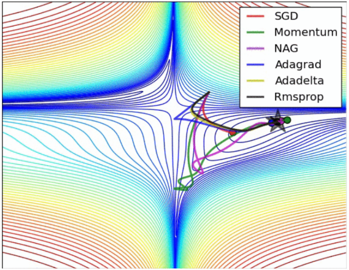
- In the final formula we use $avg(\Delta\theta^2)_{t-1}$ instead of $avg(\Delta\theta^2)_t$ because it is not known
- Adadelta uses the hyperparameters γ and ϵ, but it's **robust** so they don't need to be tuned
- The learning rate η_t does not have to be specified!

RMSprop

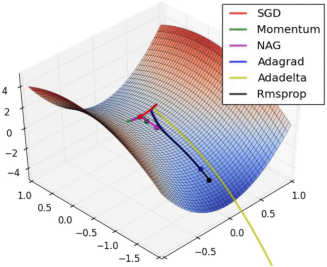
Developed independently from Adadelta around the same time by Geoff Hinton. Its updates are the same as in the preliminary Adadelta

$$\theta_{t+1} = \theta_t - \frac{\eta_t}{\sqrt{avg(g^2)_t + \epsilon}} \cdot g_t$$

Some cool animation



(a) SGD optimization on loss surface contours



(b) SGD optimization on saddle point

Link to animations

Adam (Adaptive Moment Estimation)

- Like Adadelta and RMSprop, also Adam [Kingma and Ba, 2015] stores a running average of **past squared gradients**

$$avg(g^2)_t = \beta_2 \cdot avg(g^2)_{t-1} + (1 - \beta_2) g_t^2$$

- Moreover, similar to Momentum, it also stores an exponentially decaying average of **past gradients**

$$avg(g)_t = \beta_1 \cdot avg(g)_{t-1} + (1 - \beta_1) g_t$$

They are estimates of the first and second moments of the gradient (hence the name). The parameters β_1 and β_2 are decay rates.

If initialized as vectors of zeros, $avg(g^2)_t$ and $avg(g)_t$ are biased towards zero. Compute bias-corrected first and second moment estimates

$$m_t = \frac{avg(g)_t}{1 - \beta_1^t}$$

$$v_t = \frac{avg(g^2)_t}{1 - \beta_2^t}$$

The Adam update is

$$\theta_{t+1} = \theta_t - \frac{\alpha_t}{\sqrt{v_t} + \epsilon} \cdot m_t$$

Why the bias correction

$$
\begin{aligned}
\mathbb{E}\left[avg(g^2)_t\right] &= \mathbb{E}\left[(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\cdot g_i^2\right] \\
&= \mathbb{E}\left[g_t^2\right]\cdot(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}+\zeta \\
&= \mathbb{E}\left[g_t^2\right]\cdot(1-\beta_2^t)+\zeta
\end{aligned}
$$

- $\zeta = 0$ if $\mathbb{E}\left[g_i^2\right]$ is stationary, otherwise it can be kept small assigning small weights to gradients too far in the past.

Convergence analysis

- $f_t(\theta)$ is a sequence of convex cost functions of unknown nature
- $\theta^* = \text{argmin}_\theta \sum_{t=1}^T f_t(\theta)$
- The goal is to show that the regret

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

is sublinear in T.

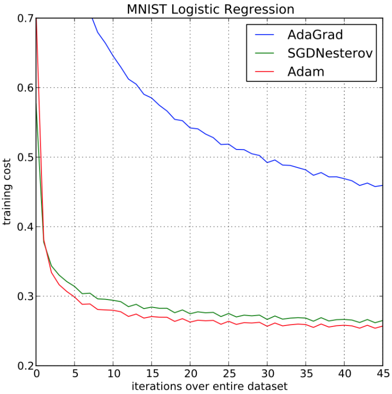
Convergence analysis

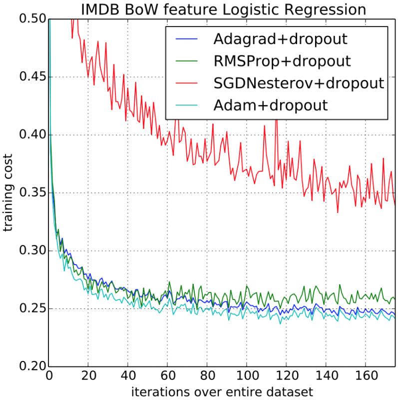
Theorem

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

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

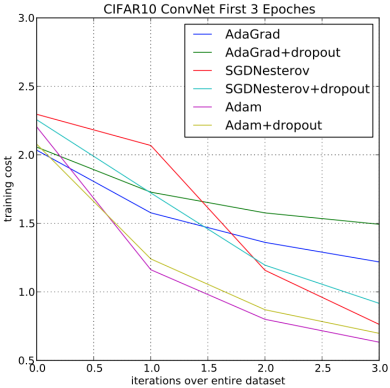
Experiment: Logistic Regression

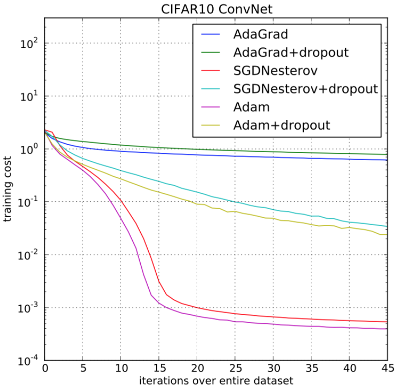




- stepsize is $\alpha_t = \alpha/\sqrt{t}$.

Experiment: Convolutional Neural Networks





Extensions: Adamax

- Instead of L^2 norm we can consider the L^p norm
- It is usually unstable, but for $p \to \infty$ the algorithm is surprisingly simple and stable. Define $v_t = \beta_2^p \cdot v_{t-1} + (1 - \beta_2^p) \cdot |g_t|^p$ and

$$u_t = \lim_{p \to \infty} (v_t)^{1/p} \quad \Rightarrow \quad u_t = \max(\beta_2 \cdot u_{t-1}, |g_t|_1)$$

- Note that u_t now does not need bias correction

The update rule for Adamax is

$$\theta_{t+1} = \theta_t - \frac{\eta_t}{u_t} \cdot m_t$$

Extensions: Temporal Averaging

The last iterate is noisy, instead one can use:

- Polyak-Ruppert averaging

$$\bar{\theta}_t = \frac{1}{t}\sum_{k=1}^{t}\theta_k$$

- running average

$$\bar{\theta}_t = \beta \cdot \bar{\theta}_{t-1} + (1-\beta)\theta_t$$

  starting from $\bar{\theta}_0 = 0$. Initalization bias can again be corrected by the estimator

$$\hat{\theta}_t = \frac{\bar{\theta}_t}{1-\beta^t}$$

Additional Strategies

- **Nadam**: combine Adam and NAG modifying the momentum term m_t.

Additional Strategies

- **Nadam**: combine Adam and NAG modifying the momentum term m_t.
- **Hogwild!**: a way to perform SGD updates in parallel. It only works if the input data is sparse, as in this case it is unlikely that processors will overwrite useful information.

Additional Strategies

- **Nadam**: combine Adam and NAG modifying the momentum term m_t.
- **Hogwild!**: a way to perform SGD updates in parallel. It only works if the input data is sparse, as in this case it is unlikely that processors will overwrite useful information.
- **Early stopping**: "beautiful free lunch". Always monitor error on a validation set during training and stop (with some patience) if your validation error does not improve enough.

Additional Strategies

- **Nadam**: combine Adam and NAG modifying the momentum term m_t.
- **Hogwild!**: a way to perform SGD updates in parallel. It only works if the input data is sparse, as in this case it is unlikely that processors will overwrite useful information.
- **Early stopping**: "beautiful free lunch". Always monitor error on a validation set during training and stop (with some patience) if your validation error does not improve enough.
- **Gradient noise**: adding noise to each gradient update

$$g_{t,i} = g_{t,i} + N(0, \sigma_t^2) \qquad \text{where} \quad \sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

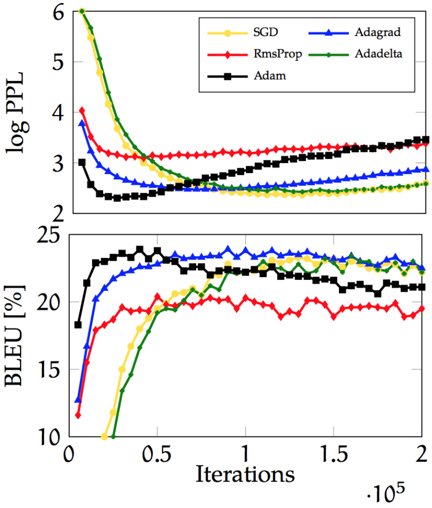
  makes networks more robust to poor initialization and helps training particularly deep and complex networks. They use $\gamma = 0.55$.

Comparison on Neural Machine Translation

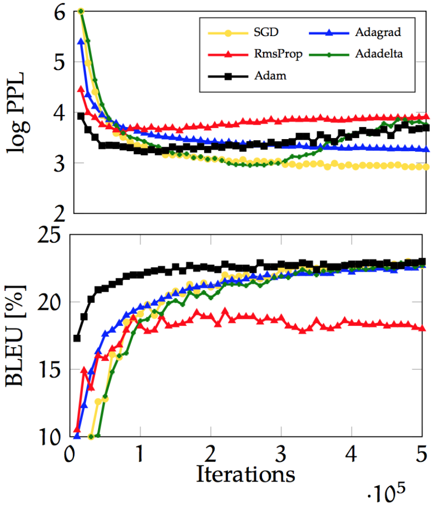
- The goal is to minimize the cross-entropy over the training samples
- Two tasks:
  - English $\rightarrow$ Romanian, 604K pairs of bilingual sentences with 16.8M English and 17.7M Romanian words
  - German $\rightarrow$ English, 4.2M sentence pairs with 133M German and 125M English words
- The hyperparameters are always the ones proposed in the original publications
- First single algorithm, then a combination of them

Comparison on Neural Machine Translation

- The goal is to minimize the cross-entropy over the training samples
- Two tasks:
  - English $\rightarrow$ Romanian, 604K pairs of bilingual sentences with 16.8M English and 17.7M Romanian words
  - German $\rightarrow$ English, 4.2M sentence pairs with 133M German and 125M English words
- The hyperparameters are always the ones proposed in the original publications
- First single algorithm, then a combination of them
- How to evaluate performance
  - PPL: perplexity, strictly related to the entropy, we want to minimize it
  - BLEU: bilingual evaluation understudy, measure of similarity between texts, we want to maximize it



(a) En⟶Ro



(b) De⟶En

Figure 1: log PPL and BLEU score of all optimizers on validation sets.

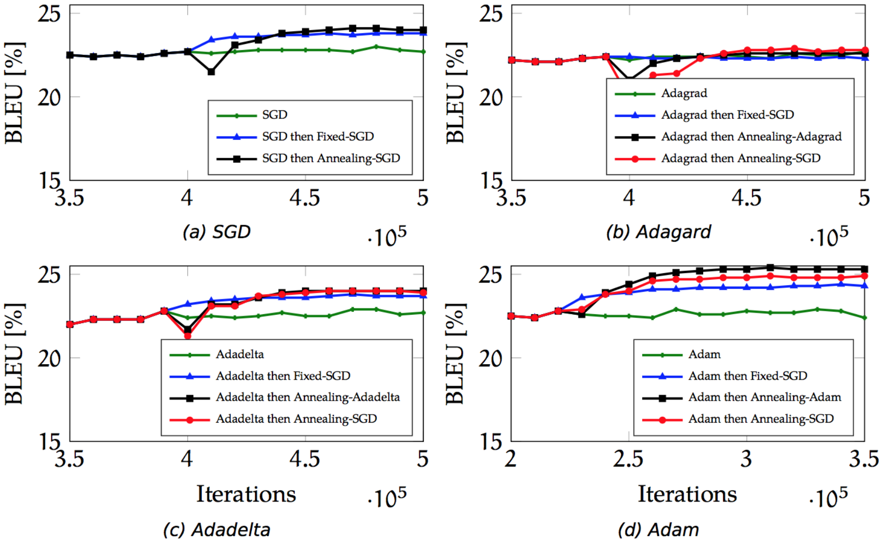


Figure 3: BLEU of optimizers followed by the combinations on the val. set for De→En. The representation of x-axis of Adam is different as it is faster.

Bibliography

1. Ruder S (2016). An overview of gradient descent optimization algorithms. *arXiv preprint* arXiv:1609.04747
2. Kingma DP, Ba J (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint*: arXiv:1412.6980
3. Bahar P, Alkhouli T, Peter JT, Brix CJS, Ney H (2017) Empirical Investigation of Optimization Algorithms in Neural Machine Translation. *The Prague Bulletin of Mathematical Linguistics*, 108, p. 13-25 .