We are starting at 14:00!

Grab a seat and get ready

Saturdays.AI
Kigali

Saturdays.AI
Kigali

#4 Optimization

AI Saturdays Kigali

Agenda

14:00 - 14:45: Random search is costly

14:45 - 15:30: Poor conditioning

15:00 - 16:00: Non-convexity

16:30 - 17:00: Costly full gradient

16:00 - 16:30: Break

16:30 - 17:00: Hyperparameter tuning

17:30 - 18:00: Challenges & Next steps

Saturdays.AI
Kigali

5 optimization challenges & solutions

1. Random search
2. Poor conditioning
3. Non-convexity
4. Full gradients are expensive to compute
5. Hyperparameter tuning

1. Gradient descent
2. Momentum
3. Overparametrization
4. Stochastic gradient descent, mini-batches
5. Adaptive methods

Saturdays.AI
Kigali

Random search is costly

$\Rightarrow$

Gradient descent

"How to optimize high-dimensional objectives"

How do we minimize an objective?

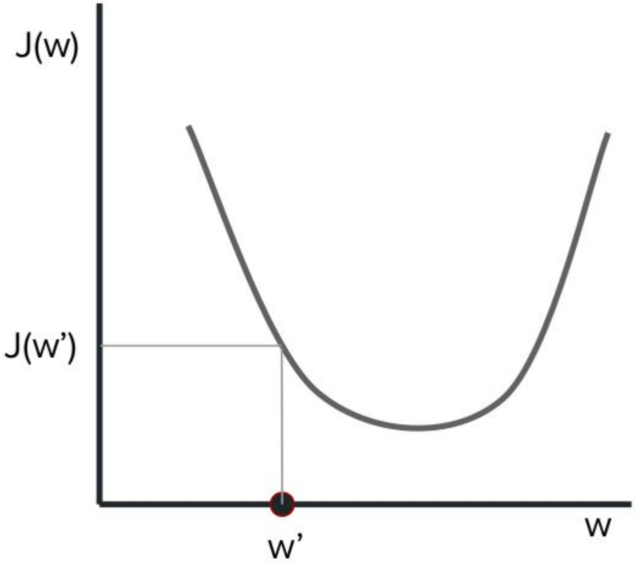
Simple, 1D objective, J(w)

Simple algorithm:

Only two ways to move: left and right

Look to your left and right!

Pick the direction that makes J smaller



Saturdays.AI
Kigali

Random search for general objectives

What if our optimization variable, w, is high-dimensional?

E.g. $w \in \mathbb{R}^d$

Algorithm: sample random points around current w

If random point, w', yields lower objective (i.e. J(w')<J(w))

Accept w' as new position and store it in w

It is a **derivative-free algorithm**: only uses function evaluations

Saturdays.AI Kigali

Random search: A curse of dimensionality

What if our optimization variable, w, is high-dimensional? $w \in \mathbb{R}^d$

- 1D: {left, right}
- 2D: {left-forward, left-backward, right-forward, right-backward}
- 3D: {left-forward-up, left-backward-up, right-forward-up, right-backward-up,
- left-forward-down, left-backward-down, right-forward-down, right-backward-down}
- ….

Exponential growth with respect to dimension

Saturdays.AI Kigali

Random search: A curse of dimensionality

What if our optimization variable, w, is high-dimensional? $w \in \mathbb{R}^d$

How many function evaluations to get ε-close to minimum?

in the order of $(1/\epsilon)^d$

Why? Probability of finding an improved point randomly decreases with dimension

Lab: Try your hand in the notebook!

Iteration complexity depends on dimension, d, i.e. not dimension-free

“Derivative-free vs. dimension-free: choose one!”

Saturdays.AI
Kigali

Gradient descent: walk down the mountain

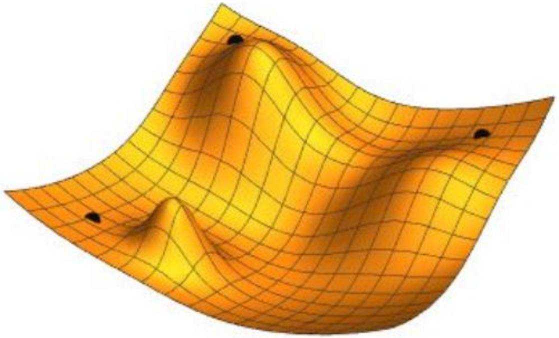
Algorithm:

Compute gradient (it points uphill)

Do step in opposite direction of gradient

Step size (**learning rate**), η

$$w_{t+1} = w_t - \eta \nabla J(w_t)$$



Dimension-free iteration complexity!

Can more efficiently handle models of many parameters

Saturdays.AI Kigali

Gradient descent to train our model

Objective: Empirical risk on the whole training set

$$\hat{R}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_w(x_i))$$

$$\begin{aligned} w_{t+1} &= w_t - \eta \nabla \hat{R}(w_t) \\ &= w_t - \eta \nabla \left(\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_w(x_i)) \right) \\ &= w_t - \eta \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(y_i, f_w(x_i)) \end{aligned}$$

Practice

Implement gradient descent

Use PyTorch automatic differentiation capabilities to compute the gradient

[Coding Exercise 3]

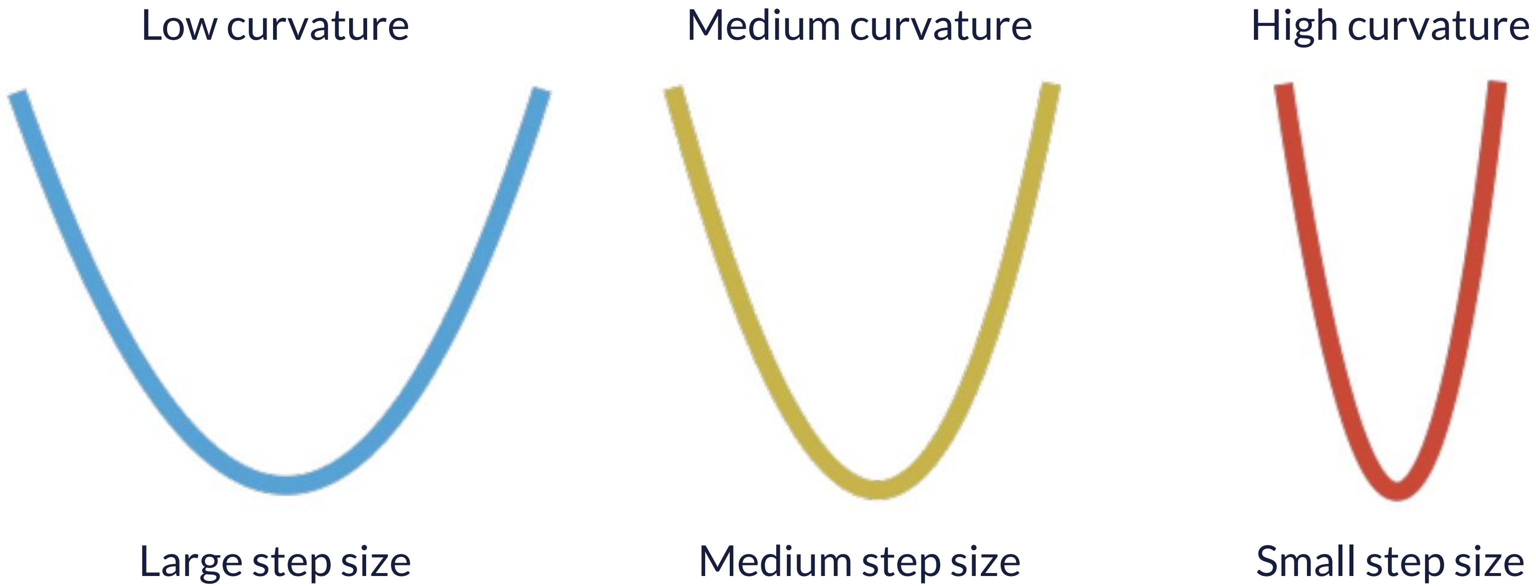
Poor conditioning

$\Rightarrow$

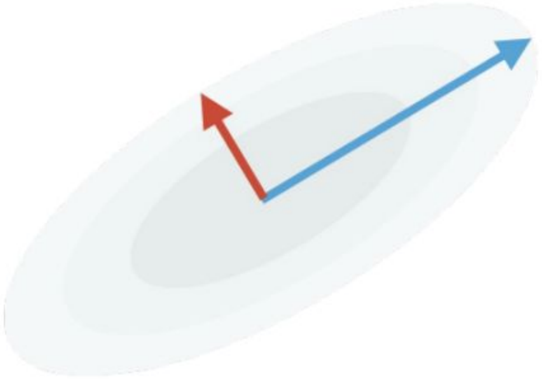
momentum

"I find it hard to select a good step size"

Curvature of 1D objective and step size



Conditioning of multidimensional objective



Low curvature High curvature

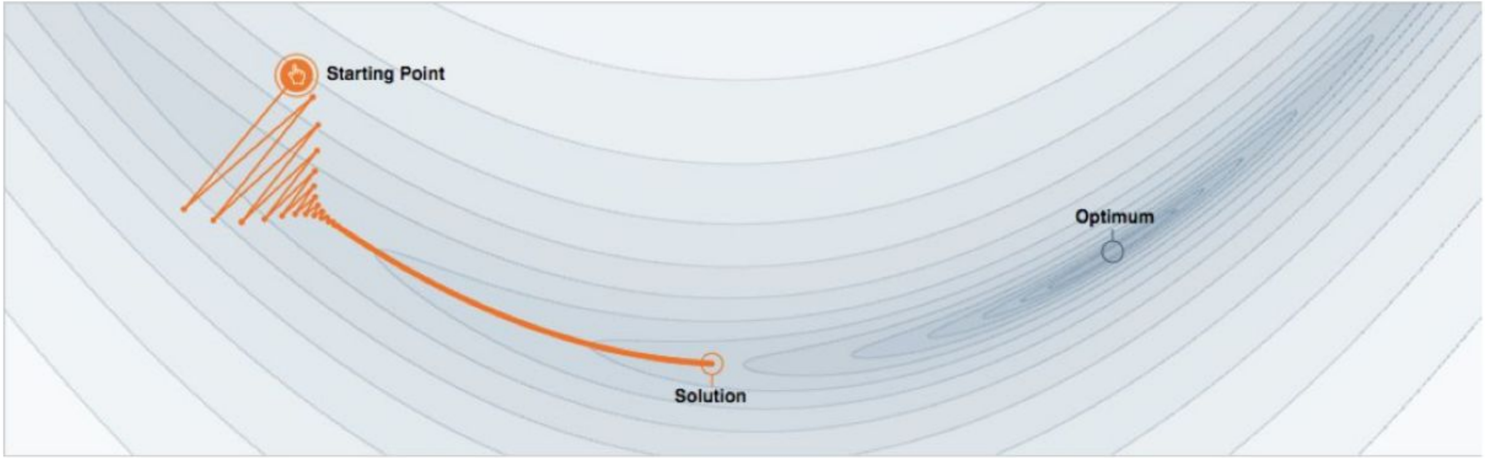
Large step size Small step size

Step size: Need large for **one direction**
Need small for **other direction**

Poor conditioning ⇒ Slow convergence

Poor conditioning and gradient descent

Gradient descent: Moves slowly along **flat directions**
Oscillates along **sharp directions**

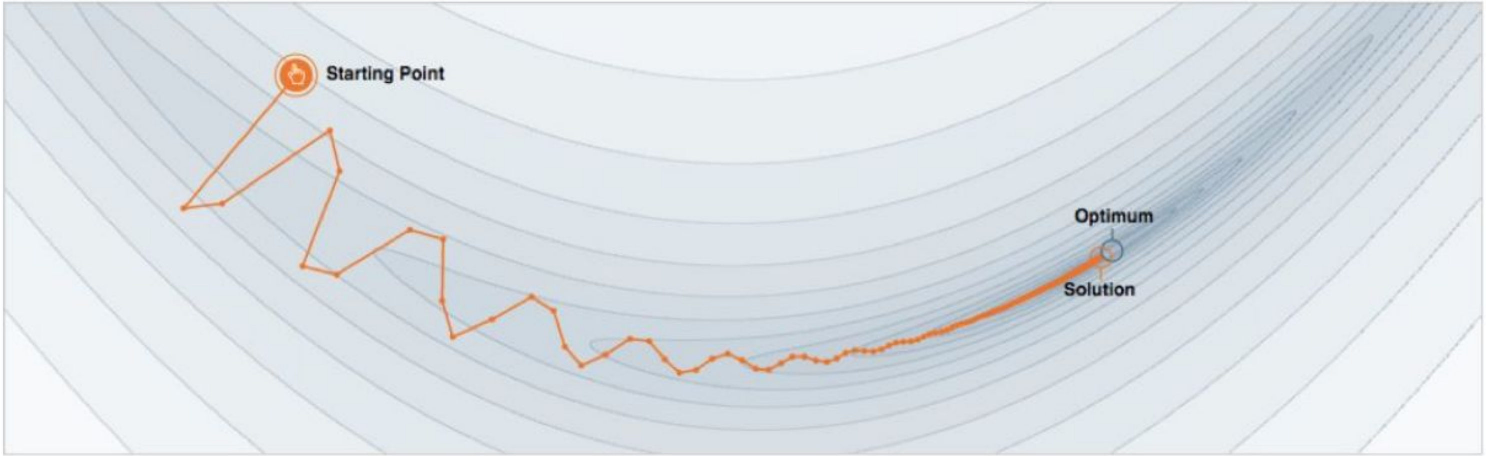


[Distill.pub]

Saturdays.AI
Kigali

Poor conditioning and momentum

Momentum: Accelerates along **flat directions**
Slows down along **sharp directions**



[Distill.pub]

Saturdays.AI
Kigali

Momentum

Momentum algorithm:

Do a **gradient descent step**

Apply the update from the last iteration, only smaller (**momentum step**)

$$w_{t+1} = w_t - \eta \nabla J(w_t) + \beta(w_t - w_{t-1})$$

- Guaranteed to accelerate convergence on very simple problems
    - Equivalent to improving conditioning
- Known in practice to make problems like training NNs easier

Practice

Implement momentum

Implement the momentum update

[Coding Exercise 4]

Non-convexity

$\Rightarrow$

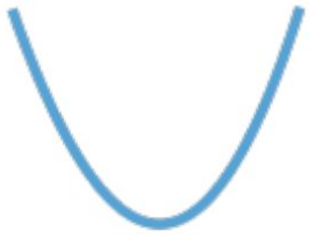
overparametrization

Convexity and non-convexity

Convex functions:

Easy to find global minimum

Gradient descent just works



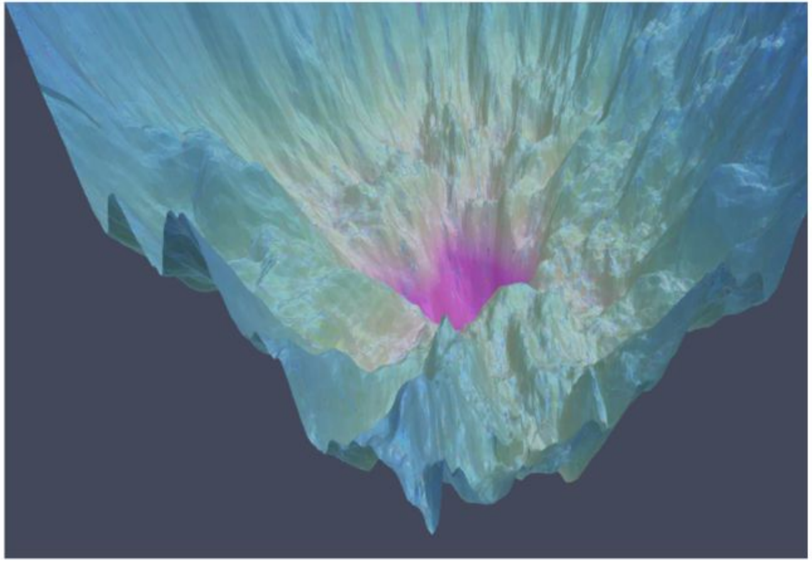
Deep learning:

Non-convex training objective

Not guaranteed to efficiently find global minimum

Non-convexity of “loss landscape”

- Loss landscape
  - The surface of our objective
- Seriously non-convex
  - many suboptimal local minima
  - no guarantees on finding an optimum
  - might be ok to not find the global optimum

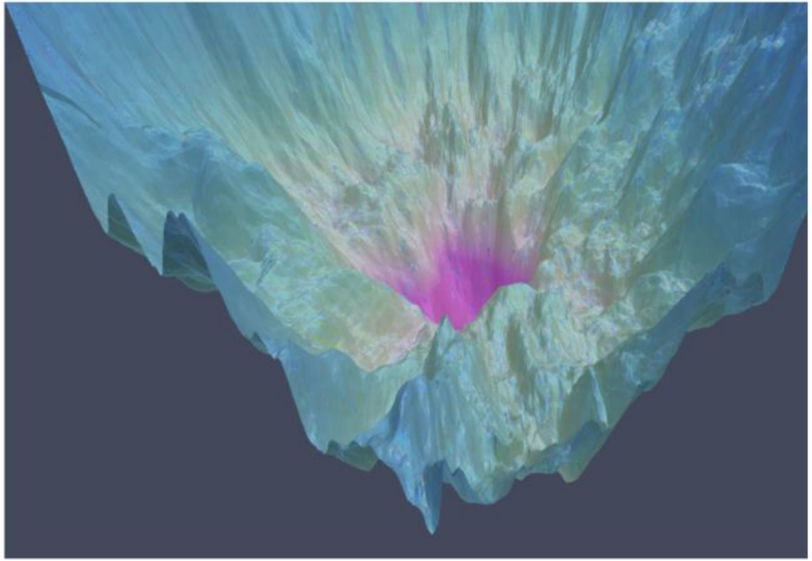


Part of the losslandscape.com project by Javier Ideami

Saturdays.AI
Kigali

Interactive break

- **Lab**: Take a minute to play with the interactive visualization!
- Click on the (i) button on the top right
  - Read the available functionality
- Click on the “gradient descent” button, fifth on bottom left
  - Click on landscape to start runs
  - Initialization matters!!
- Play with learning rate (bottom right)



Part of the losslandscape.com project by Javier Ideami

Saturdays.AI Kigali

Overparametrization

Are all models equally sensitive to initialization?

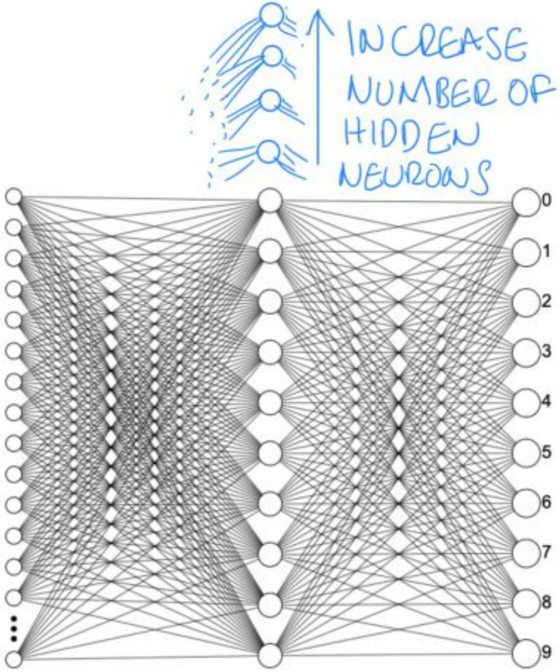
- No! Wider networks are less sensitive/easier to train
- Ample empirical and theoretical evidence

Lab: Increase the size of your MLP's hidden layer

- Study how it becomes less sensitive to initialization

Increasing the network's parameters can have negative effects

- Overfitting: Surprisingly, in many cases it doesn't happen
- Increased memory and computational complexity



Costly full gradients

$\Rightarrow$

mini-batches

Cost of full-batch gradient descent

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(y_i, f_w(x_i))$$

- Big networks, computing the gradient for one example is costly
- The gradient for the full training set (a.k.a. **full-batch**) is n times that

- Do we really need to see all n examples to do one single step?

Saturdays.AI
Kigali

Mini-batching: use a few examples per step

- Computing over 60K examples on MNIST for a single exact update is too expensive.
  - Even worse for bigger datasets

- We use **mini-batches**:
  - A subset, B_t, of the training set
  - Different at each step, t
  - noisy estimate of the true gradient
    - stochastic gradients

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(y_i, f_w(x_i))$$

$$\Downarrow$$

$$w_{t+1} = w_t - \eta \frac{1}{|B_t|} \sum_{i \in B_t} \nabla \ell(y_i, f_w(x_i))$$

Saturdays.AI
Kigali

Mini-batching: discussion

- Gradient updates are worse (less accurate)
- We can do many of them in the same amount of time as one full gradient step
  - Great speedups in practice
- We might not find the same minimum, but resulting models are still great

- We want the mini-batches to be representative of the data distribution
  - Mini-batches are often selected at random,
  - or in order after initial shuffling of training set

Saturdays.AI
Kigali

How should we choose the mini-batch size?

- too small batch (e.g. SGD): bounces around a lot, and can lead to slower convergence to a minimum
- too big batch won't fit on the GPU

- Simple rule of thumb:
  - Pick the largest batch size that fits in the GPU

Saturdays.AI
Kigali

Practice

Implement minibatch sampling

Produce IID subsets of the training set of the desired size

[Coding Exercise 6]

Break

Saturdays.AI
Kigali

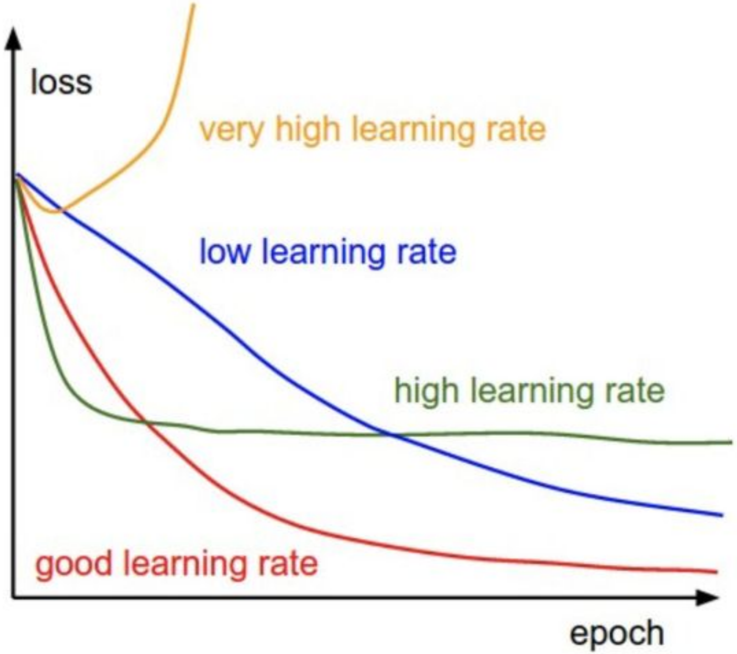
Hyperparameter tuning

⇒

adaptive methods

Saturdays.AI
Kigali

Importance of learning rate (step size)



Can make a big difference

- Varies by model/dataset
- Takes time

Image credit:
Stanford CS231 website

Saturdays.AI
Kigali

Adjusting the learning rate

- If you learn too fast
    - you see wild variations on your loss curve
    - you converge towards solutions with huge (+ or -) weights (or NaN values)
- If you learn too slowly
    - convergence takes forever

A partial solution: decrease the rate if your loss varies wildly;
otherwise increase it

Might need to go faster initially, slower later

Saturdays.AI
Kigali

Learning rate schedules

$$w_{t+1} = w_t - \eta_t \nabla J(w_t)$$

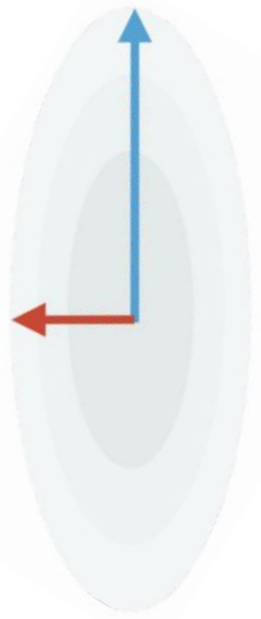
- Polynomial schedules, e.g. → $\eta_t = \frac{\alpha}{c+t}, \; \eta_t = \frac{\alpha}{c+\sqrt{t}}$
- Exponential
- Stepwise decay
- Cosine/cyclical schedules
- ...
- Still: need to tune hyperparameters
  one learning rate for all weights

Saturdays.AI
Kigali

Poor conditioning and weight-specific LRs

- Different layers can have gradients of drastically different magnitudes
    - especially in deep nets
    - poor conditioning (Section 4)
- Some gradients can become enormous

- Ideas?
    - **clip gradients**: if greater than specified magnitude
    - **individual learning rates** for different weights or layers



Saturdays.AI
Kigali

RMSprop

Uses a moving average
instead of sum used by Adagrad

$$[w_{t+1}]_i = [w_t]_i - \frac{\eta}{\sqrt{[v_{t+1}]_i + \epsilon}}[\nabla J(w_t)]_i$$

Moving average can be useful on
non-convex objectives

$$[v_{t+1}]_i = \alpha[v_t]_i + (1-\alpha)[\nabla J(w_t)]_i^2$$

Adam [Kingma, Ba, 2015] adds momentum
to RMSProp (+couple more tricks)

Extremely successful

Saturdays.AI
Kigali

Practice

Implement RMSprop

Implement the update of the RMSprop optimizer

[Coding Exercise 7]

Challenges & Next steps!

Saturdays.AI
Kigali

Kahoot!

Any questions?

Saturdays.AI
Kigali

Saturdays.AI
Kigali

THANKS

kigali@saturdays.ai
coming soon
coming soon