Optimization

Ioannis Mitliagkas

neuromatch academy

Notebook/lab:
Jose Gallego-Posada

Hello from today's lecturer

Ioannis Mitliagkas
Assistant professor University of Montréal
Core member of Mila

Amateur musician,
professional researcher in ML and optimization

Eternally dreaming of mediterranean summers

Credits

- NeuroMatch academy organizers and staff
- Content creator: Jose Gallego-Posada
- Some material from original slides by Lyle Ungar and Konrad Kording

Introduction: Optimization

Section 1

What is it?

Why is it important?

What is optimization about?

What we optimize

How we optimize it

Picking the loss function

- **Technical questions**
  - MSE vs. CrossEntropy -- for multi-class problem
  - Accuracy vs. AUC -- for class imbalance
- **Societal questions**
  - Fairness: Optimal for whom?
  - Unintended consequences

Unintended consequences

Cobras in India

Game play

[OpenAI.com]

What is optimization about?

What we optimize

How we optimize it

How expensive can training NNets be?

“Based on information released by Google, we estimate that, at list-price, training **the 11B parameter variant of T5 costs well above $1.3 million for a single run**. Assuming 2-3 runs of the large model and hundreds of the small ones, the (list-) price tag for the entire project may have been **$10 million**”

- Sharir, Peleg & Shoam, 2020

Summary of today

1. Why optimization is important
2. Case study: MLP classification
3. Gradient descent
4. Momentum
5. Non-convexity
6. Mini-batches
7. Adaptive methods
8. Putting it all together
9. Conclusion and ethical considerations

Case study: MLP classification

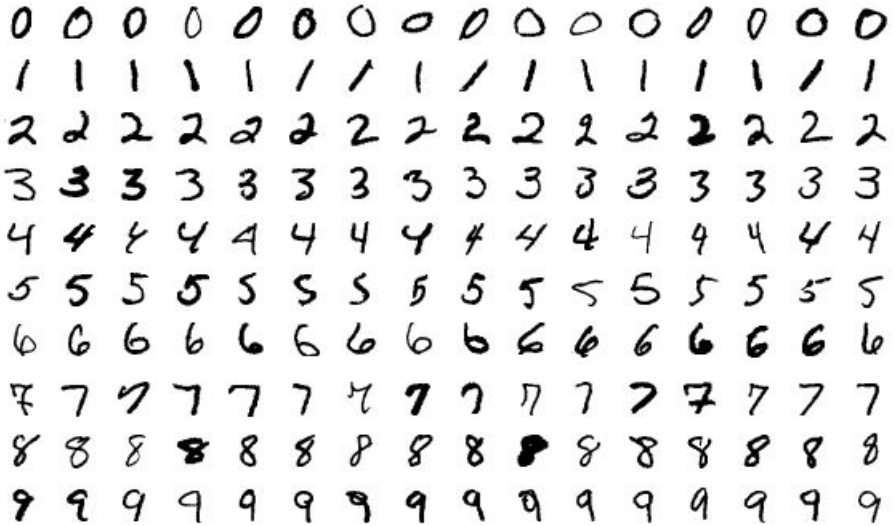
Section 2

A running example for today

- Data
- Model
- Loss
- Empirical risk minimization

Dataset - MNIST

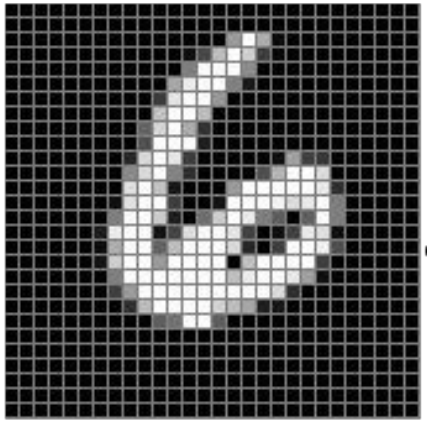
Modified National Institute of Standards and Technology database
-- Yann LeCun, 1998



[Wikipedia]

Dataset - MNIST

- 60,000 training images
- 10,000 testing images
- Each example, (x,y):
  - x: Image of 28x28 pixels
  - y: Label in {0, 1, 2, ..., 9}
- Each pixel: grayscale value {0, 1, ..., 255}



[ml4a.net]

Data

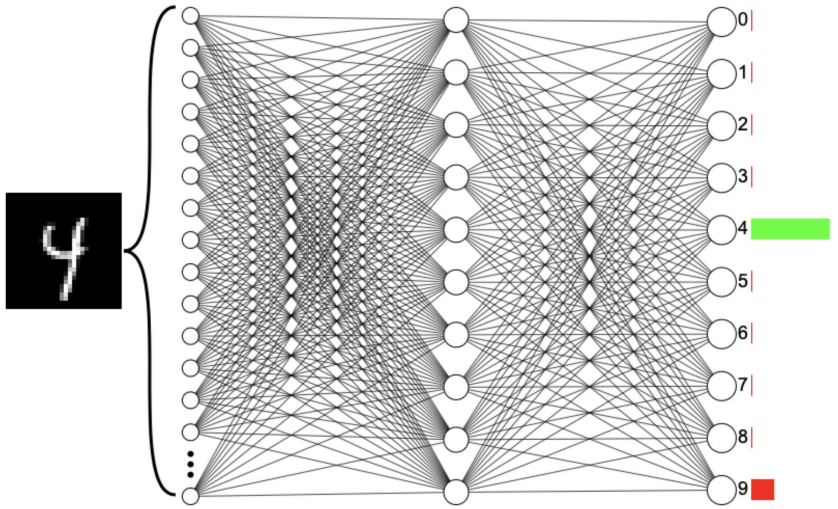
- Training set, S, of n examples $\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

- Drawn i.i.d. from data distribution $(x_i, y_i) \sim D$

Model - MLP

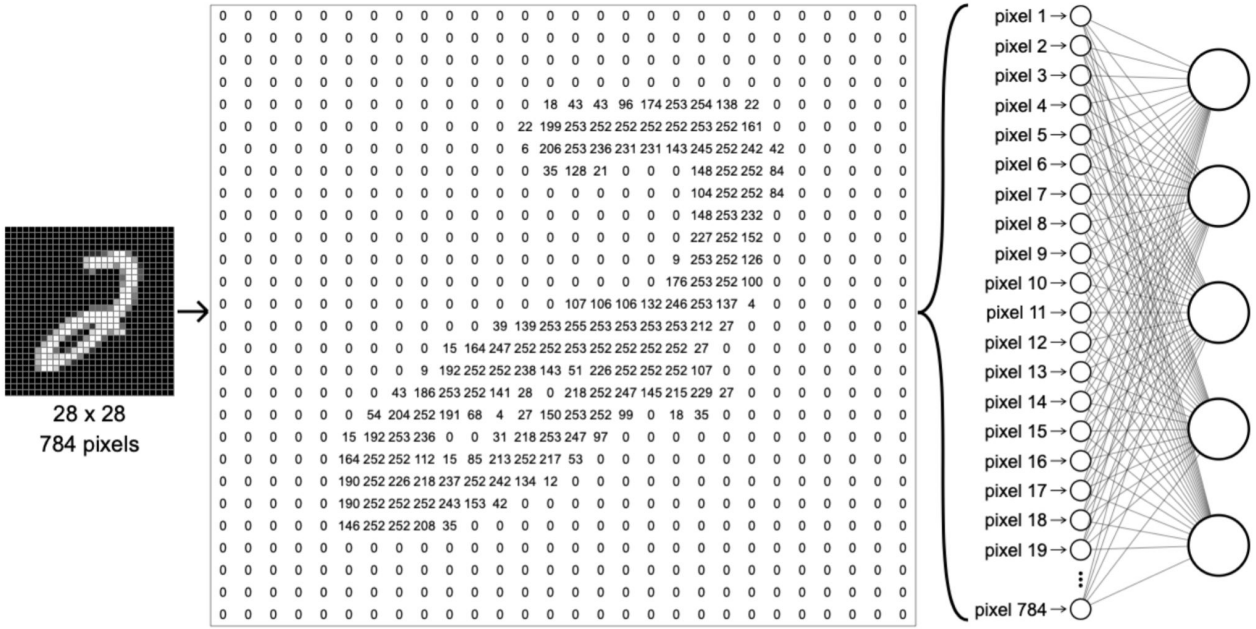
- Multi-layer perceptron (Week 1, Day 3)
- Example: one hidden layer
- 10 outputs (10-class classification)
- Takes vector as input

- Image is a grid (matrix). How does this work?
  - Week 2, Day 1: Convolutional NNs
  - Simple approach for now:
  - Stack pixels in long vector



[ml4a.net]

Model - Stacking pixels into input vector



[ml4a.net]

MLP and prediction

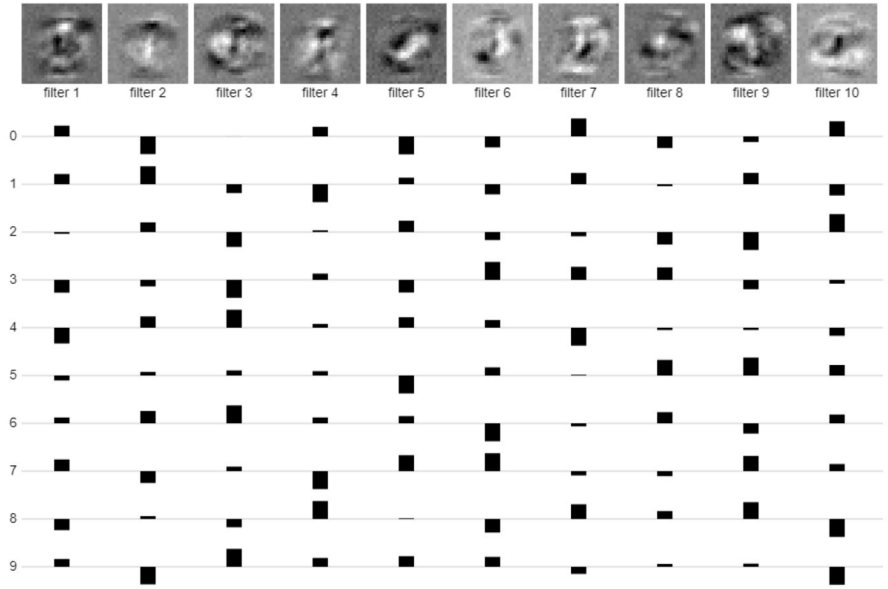
- MLP makes prediction $\hat{y}_i = f_w(x_i)$
  - Parametrized by weights w (and biases b)
  - Composition of modules:

    affine transformation & non-linearity
  - Softmax (Day 3)

$$f(x) = \text{softmax}(W_L \sigma(W_{L-1} \sigma(\ldots \sigma(W_0 x + b_0) \ldots) + b_{L-1}) + b_L)$$

Visualization of MLP parameters

- 1-hidden layer network
  - 10 hidden units
  - Showing ‘filters’ and their contribution to each of the 10 output neurons



[ml4a.net]

Losses

- Example (x_i, y_i)
- Model makes prediction $\hat{y}_i = f_w(x_i)$
- Losses
  - Mean square error (Day 2)
  - 0-1 loss (accuracy)
  - Cross-entropy loss (Day 3)

$\ell(y_i, \hat{y}_i)$

Goal: learn model that generalizes well

$$R(w) = \mathbb{E}_{(x,y)}[\ell(y, f_w(x))]$$

- **Risk**: expected loss of our model on unseen data
  - Coming from data distribution
- For fixed loss, ℓ, this is the quantity we want to minimize

- But: We do not have access to the data distribution!

Surrogate objective: Minimize empirical risk

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

- **Empirical risk** tells us how badly we are doing on the training set
  - Not always representative of performance on unseen data (our goal)
  - For n large enough, it is a good approximation
- Surrogate goal: find values for weights, w, which achieve low empirical risk

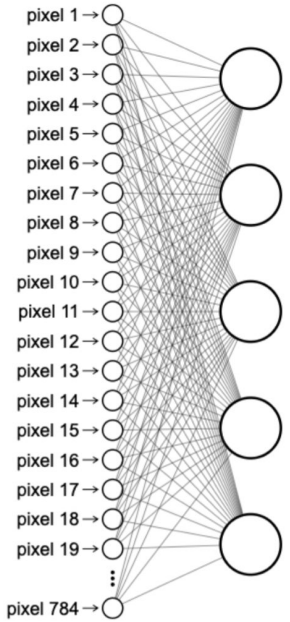
Today: Our running example

Classification on MNIST

We start with simple, linear model
(no hidden layers yet)

We want to learn parameters

Where do we start?



[ml4a.net]

Today: 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

Random search is costly

⇒

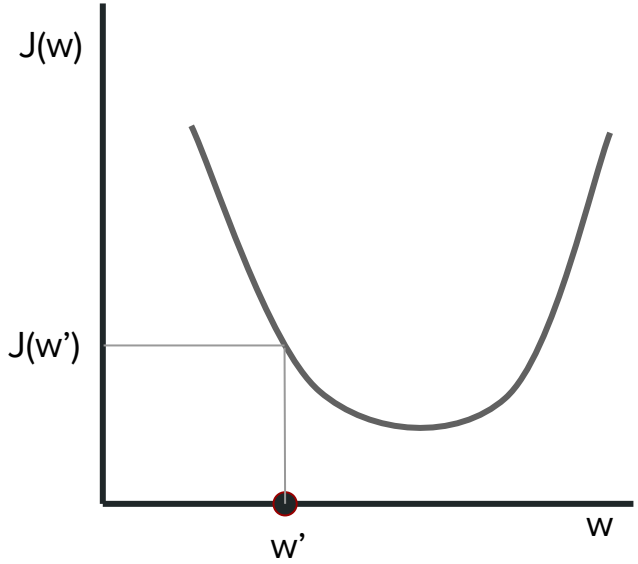
Gradient descent

Section 3

“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

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

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

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, dimension-free: choose one!"

Gradient descent: walk down the mountain

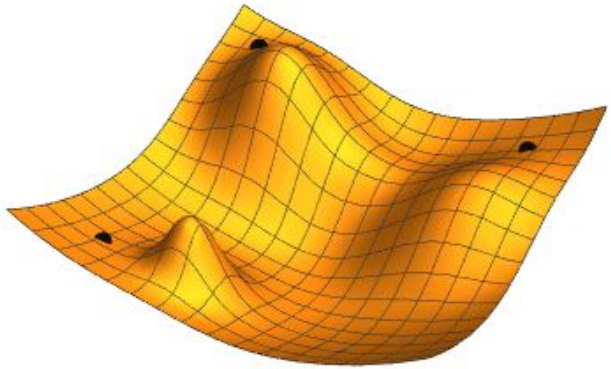
Algorithm:

Compute gradient (it points uphill)

Do step in opposite direction of gradient

Step size (**learning rate**), η

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



[Wikimedia commons]

Dimension-free iteration complexity!

Can more efficiently handle models of many parameters

Gradient descent to train our model

Objective: Empirical risk on the whole training set

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

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

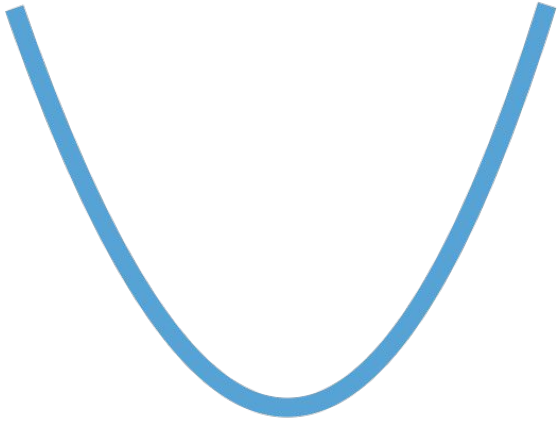
Poor conditioning
⇒
momentum

Section 4

“I find it hard to select a good step size”

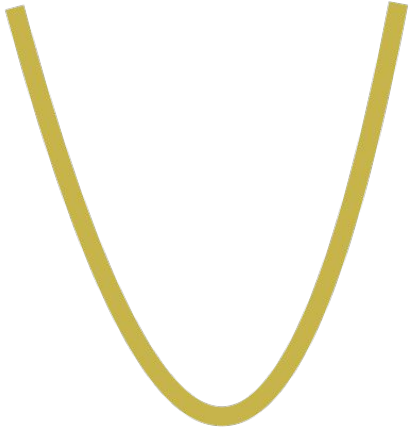
Curvature of 1D objective and step size

Low curvature



Large step size

Medium curvature

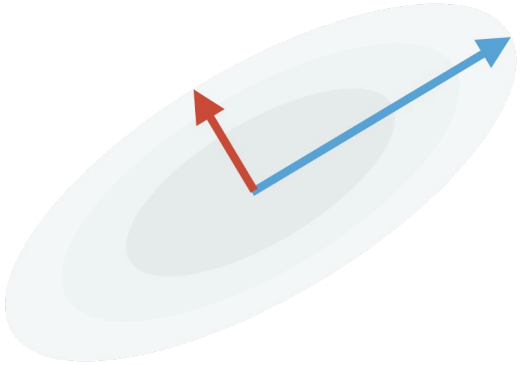


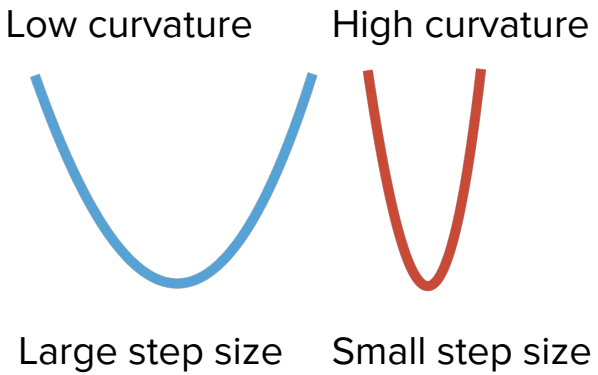
Medium step size

High curvature

Small step size

Conditioning of multidimensional objective





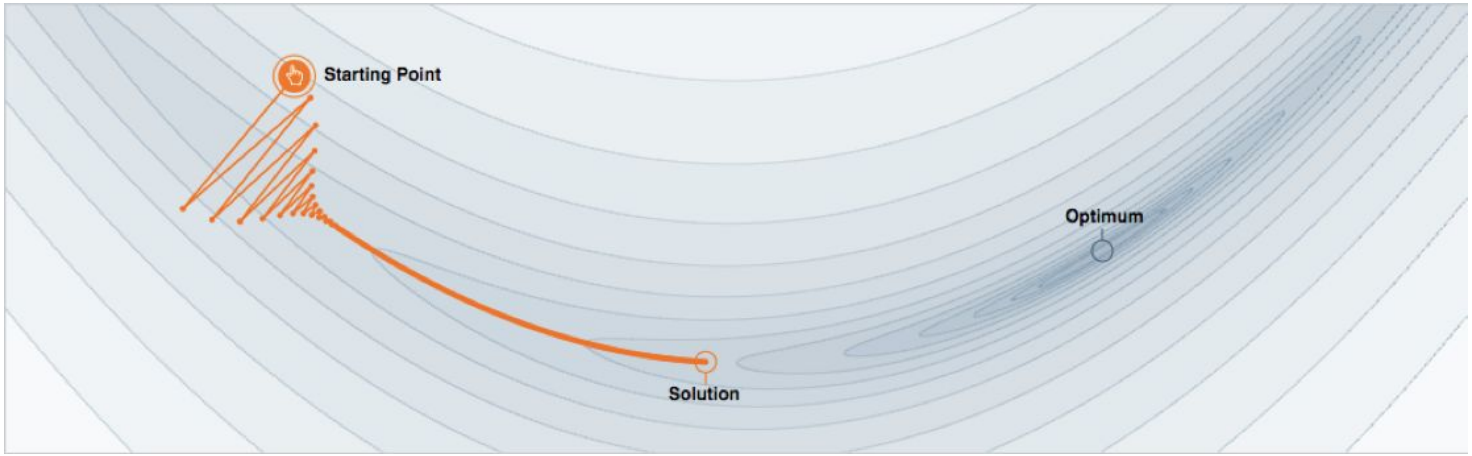
Step size: Need large for **one direction**
Need small for **other direction**

Poor conditioning ⇒ Slow convergence

Poor conditioning and gradient descent

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

Oscillates along **sharp directions**

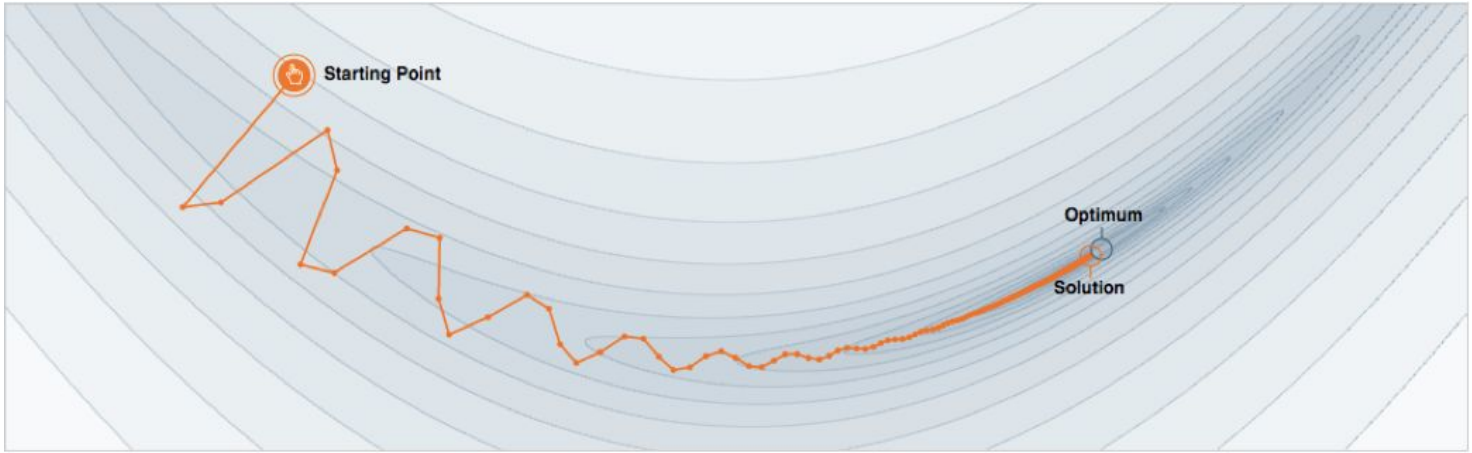


[Distill.pub]

Poor conditioning and momentum

Momentum: Accelerates along **flat directions**

Slows down along **sharp directions**



[Distill.pub]

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

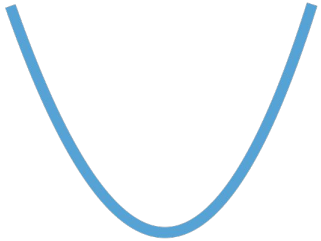
Non-convexity
⇒
overparametrization

Section 5

Convexity and non-convexity

Convex functions:

- Easy to find global minimum
- Gradient descent just works

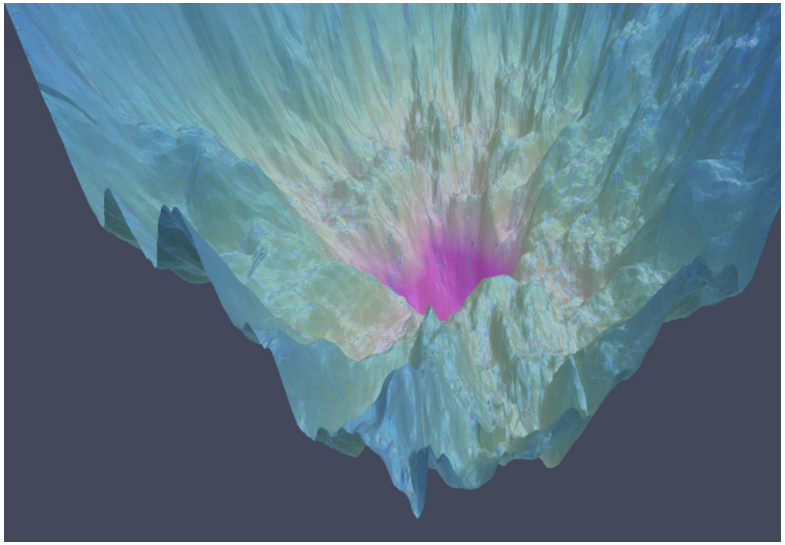


Deep learning:

- Non-convex training objective
- Not guaranteed to efficiently find global minimum

Non-convexity of “loss landscape”

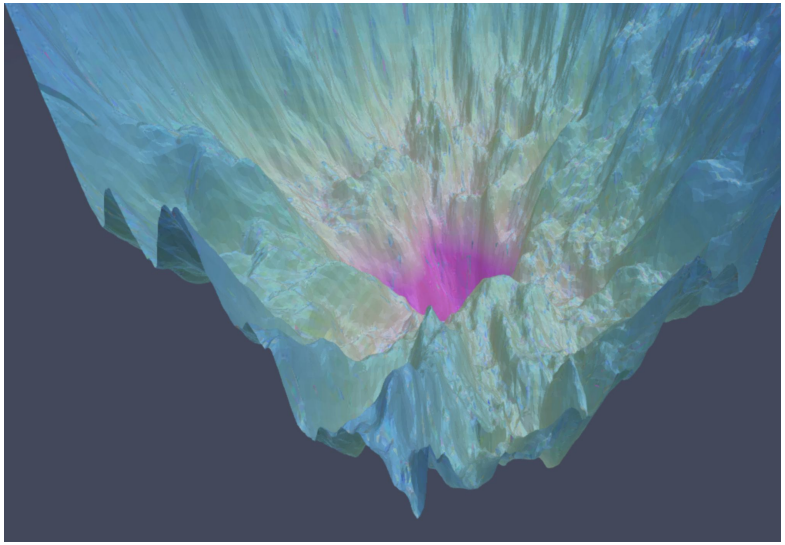
- Loss landscape
  - The surface of our objective
- Seriously non-convex
  - many suboptimal local minima
  - no guarantees on finding an optimum
  - might be ok to not find the global optimum



Part of the losslandscape.com project by Javier Ideami

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

Overparametrization

Are all models equally sensitive to initialization?

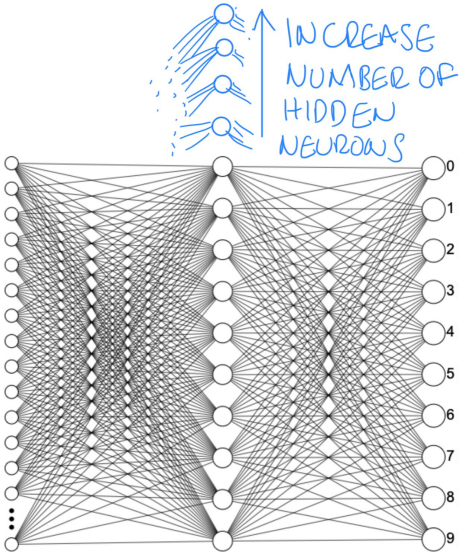
- No! Wider networks are less sensitive/easier to train
- Ample empirical and theoretical evidence

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 ⇒ mini-batches

Section 6

Cost of full-batch gradient descent

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

- 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?

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

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

- 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

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

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

How should we choose the mini-batch size?

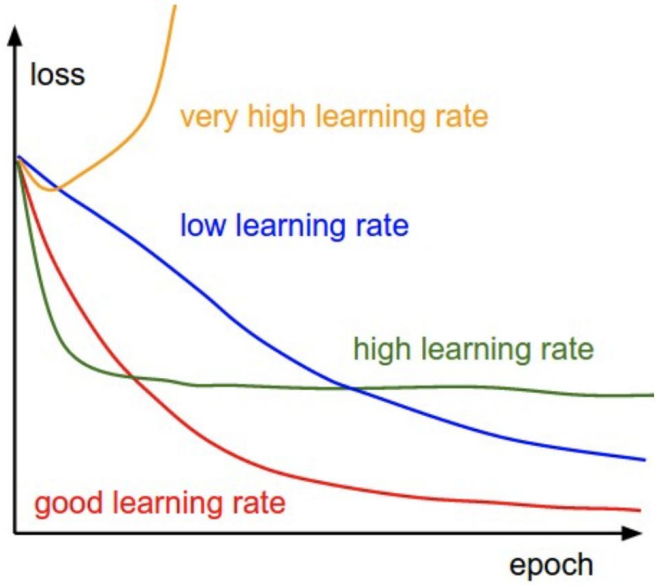
- too small batch (e.g. SGD): bounces around alot, and can lead to slower convergence to a minimum
- too big batch won't fit on the GPU

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

Hyperparameter tuning
⇒
adaptive methods

Section 7

Importance of learning rate (step size)



Can make a big difference

- Varies by model/dataset
- Takes time

Image credit:
Stanford CS231 website

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

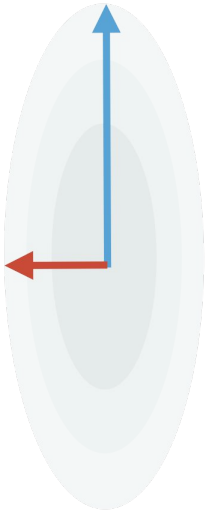
Learning rate schedules

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

- 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

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

Adagrad [Duchi et al., 2011]

- **Ada**ptive **grad**ient algorithm
- Adapts the learning rate
  for each parameter
- Using running sum of past gradients
- Typically used in stochastic form:
  - Instead of full gradient, use gradient from mini-batch

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

$$[v_{t+1}]_i = \sum_{s=1}^{t} [\nabla J(w_s)]_i^2$$

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

Putting it all together

Section 8

Today: 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

Leveraging Pytorch

1. Define the model and optimizer

```
model = MLP(in_dim, out_dim, hidden_dims)
optimizer = torch.optim.Adam(model.parameters(), lr)
```

2. Load data and sample batches

```
train_loader =
torch.utils.data.DataLoader(train_set, batch_size)
```

3. Automatic differentiation

```
loss = loss_fn(model(batch_ins), batch_target)
loss.backward()
```

4. Update model parameters

```
optimizer.step()
```

Putting it all together

In this lab section, we invite you to outperform our baseline on our running example of MNIST classification

1. Tweak data hyperparameters
2. Tune optimization settings (learning rate, momentum, etc)
3. Try different optimization methods

Ethical concerns

Section 9

The biases baked into ML

- **The choices made in machine learning have many consequences**
  - some from the training data: how it was selected and what the labels are
  - some from the loss function selected.

What do we want to optimize?

https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

Algorithmic fairness

Setting: Two groups, a “protected class” and a general population

Fairness Criteria:

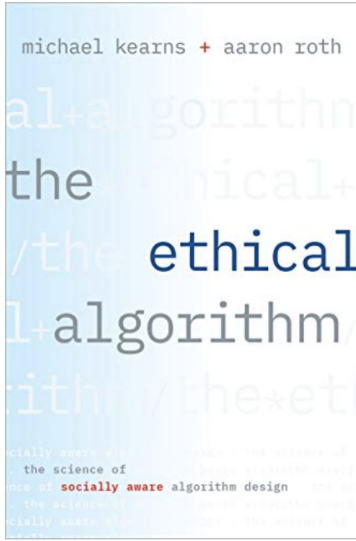
same prediction accuracy in the two groups

same false positive rate

same percentage labeled “True”

the protected class label is not used in the prediction

In general, these criteria are incompatible



Online ads

March 2019: Facebook stops allowing use of race, gender or age when targeting ads for housing, employment and credit.

Will this mean that race, gender and age will no longer affect who is shown such ads?

The objectives we use introduce bias

- Keep an eye out for those unintended consequences!
- Take a critical look of your results
- Discuss with domain experts if it makes sense
- Have fun :)