Lecture Notes for

Machine Learning in Python

[👨‍🏫, 👨‍💻, 🐍, 👨‍🔬]

Professor Eric Larson

Optimizing Neural Networks

Class Logistics and Agenda

- Logistics
  - Grading
  - Flipped Module
- Agenda:
  - Practical Multi-layer Architectures
  - Programming Examples and Adaptive Eta's
- Next Time: More MLPs

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn

Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,

Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn

Detailed mathematics for simple optimization intuition for advanced optimization

Neural Networks and Back Prop.

Numpy

Detailed mathematics for NN operations

Wide and Deep Networks | **Convolutional Networks** | **Recurrent Networks**

Keras, Tensorflow

Intuition, Detailed implement.

Ethics in Language Models

ConceptNet

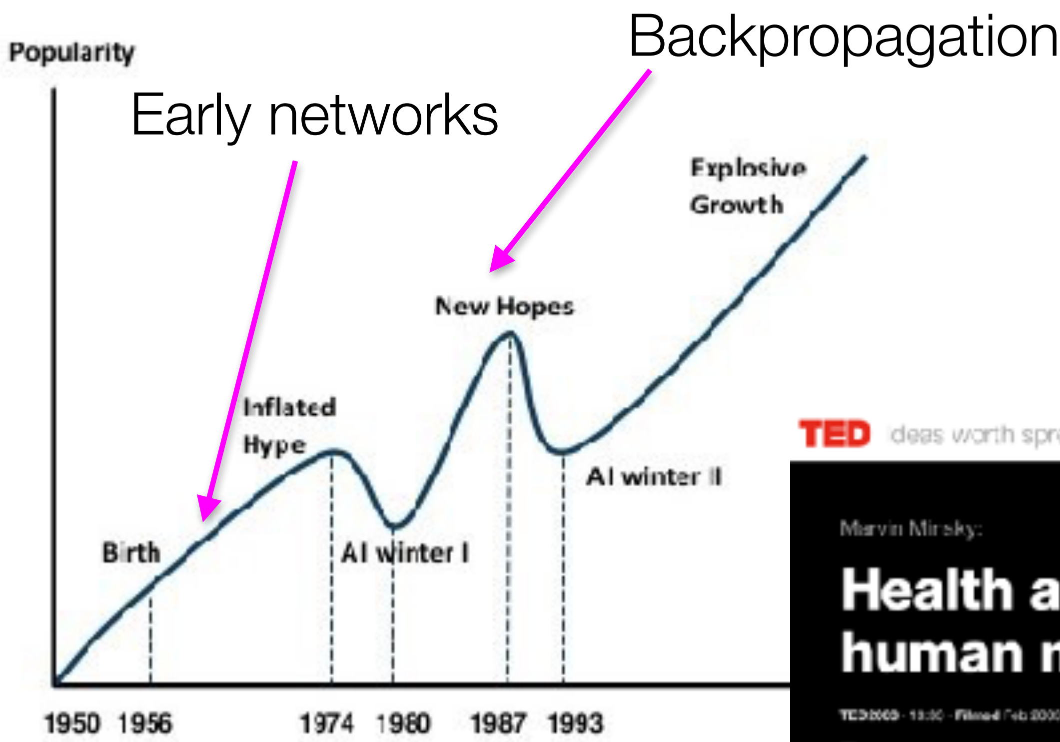
Case studies

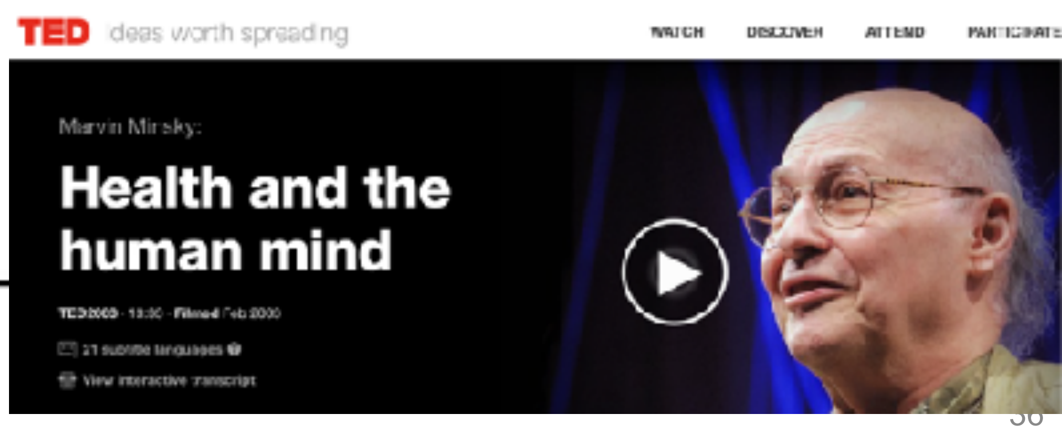
Semester Summary, so far!

- Adaline network, Widrow and Hoff, 1960
  - iterative linear regression
- Perceptron
  - *with sigmoid*: logistic regression
- One-versus-all implementation is the same as having $\mathbf{w}_{class}$ be rows of weight matrix, $\mathbf{W}$
  - works in adaline
  - works in logistic regression

these networks were created in the 50's and 60's
but were abandoned
why were they not used?

The First AI Winter (if not covered already)





The Rosenblatt-Widrow-Hoff Dilemma

- 1960’s: Rosenblatt got into a public academic argument with Marvin Minsky and Seymour Papert

  "Given an elementary α-perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always yield a solution to C(W) in finite time…”

- Minsky and Papert publish limitations paper, 1969:

  “the style of research being done on the perceptron is doomed to failure because of these limitations.”

- Widrow and Rosenblatt try to build bigger networks without limitations and fail
  - Neural Networks research **basically stops** for **17 years**
- **Until**: researchers revisit training bigger networks
  - Neural Networks with multiple layers

Stable Training of Multi-layer Architectures: history

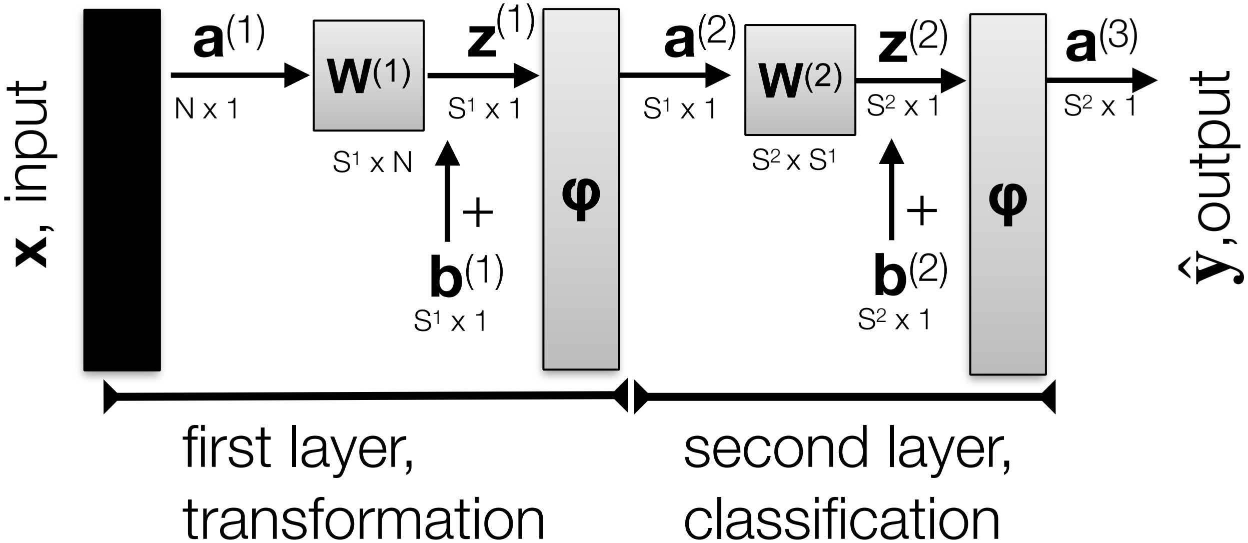
- 1986: *Rumelhart*, *Hinton*, and *Williams* popularize gradient calculation for multi-layer network
  - *technically* introduced by Werbos in ~1982
- **difference**: Rumelhart *et al.* validated ideas with a computer
- until this point no one could train a multiple layer network consistently
- algorithm is popularly called **Back-Propagation**
- wins pattern recognition prize in 1993, becomes de-facto machine learning algorithm until: SVMs and Random Forests in ~2004
- would eventually see a resurgence for its ability to train algorithms for Deep Learning applications: **Hinton is widely considered the founder of deep learning**

David Rumelhart

Geoffrey Hinton

Review: Back propagation

- Optimize all weights of network at once
- Steps:
    1. Forward propagate to get all $\mathbf{Z}^{(l)}$, $\mathbf{A}^{(l)}$
    2. Get final layer gradient
    3. Back propagate sensitivities
    4. Update each $\mathbf{W}^{(l)}$



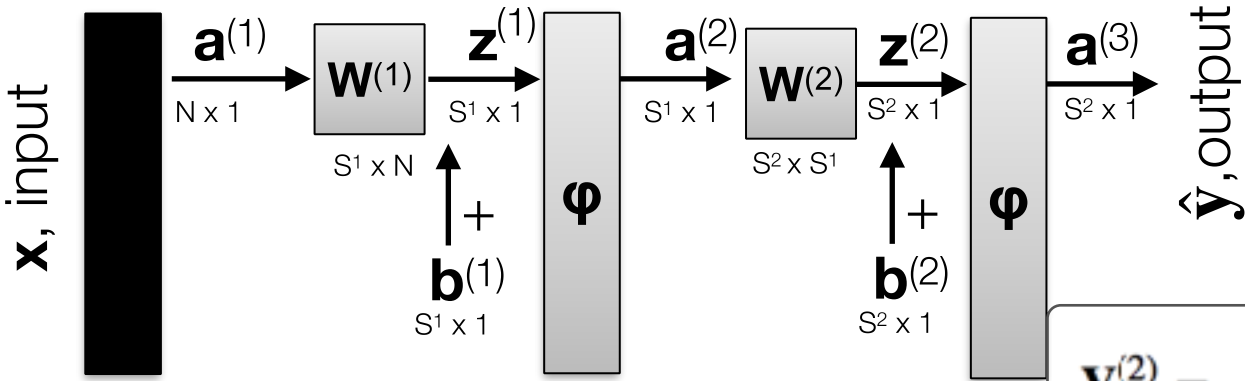
$$J(\mathbf{W}) = \left\| \mathbf{Y} - \hat{\mathbf{Y}} \right\|^2$$

$$w_{i,j}^{(l)} \leftarrow w_{i,j}^{(l)} - \eta \frac{\partial J(\mathbf{W})}{\partial w_{i,j}^{(l)}}$$

$$b_i^{(l)} \leftarrow b_i^{(l)} - \eta \frac{\partial J(\mathbf{W})}{\partial b_i^{(l)}}$$

**Recall from Flipped Assignment!

Review: Back Propagation Summary



1. Forward propagate to get **Z**, **A**
2. Get final layer gradient
3. Back propagate sensitivities
4. Update each $\mathbf{W}^{(l)}, \mathbf{b}^{(l)}$

$$\mathbf{V}^{(2)} = -2(\mathbf{Y} - \mathbf{A}^{(3)}) * \mathbf{A}^{(3)} * (1 - \mathbf{A}^{(3)})$$

$$\nabla^{(2)} = \mathbf{V}^{(2)} \cdot [\mathbf{A}^{(2)}]^T$$

$$\mathbf{V}^{(1)} = \mathbf{A}^{(2)} * (1 - \mathbf{A}^{(2)}) * [\mathbf{W}^{(2)}]^T \cdot \mathbf{V}^{(2)}$$

$$\nabla^{(1)} = \mathbf{V}^{(1)} \cdot [\mathbf{A}^{(1)}]^T$$

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \nabla^{(l)}$$

Where is the problem of **vanishing gradients** introduced?

**Recall from Flipped Assignment!

07a. MLP Neural Networks with bias.ipynb

same as Flipped Assignment!
with regularization
and vectorization
and mini-batching

A. $\mathbf{z} = \mathbf{W} \cdot \mathbf{a}_{bias}$ old notebooks

B. $\mathbf{z} = \mathbf{W} \cdot \mathbf{a} + \mathbf{b}$ new notebook!

Optimization Heuristics

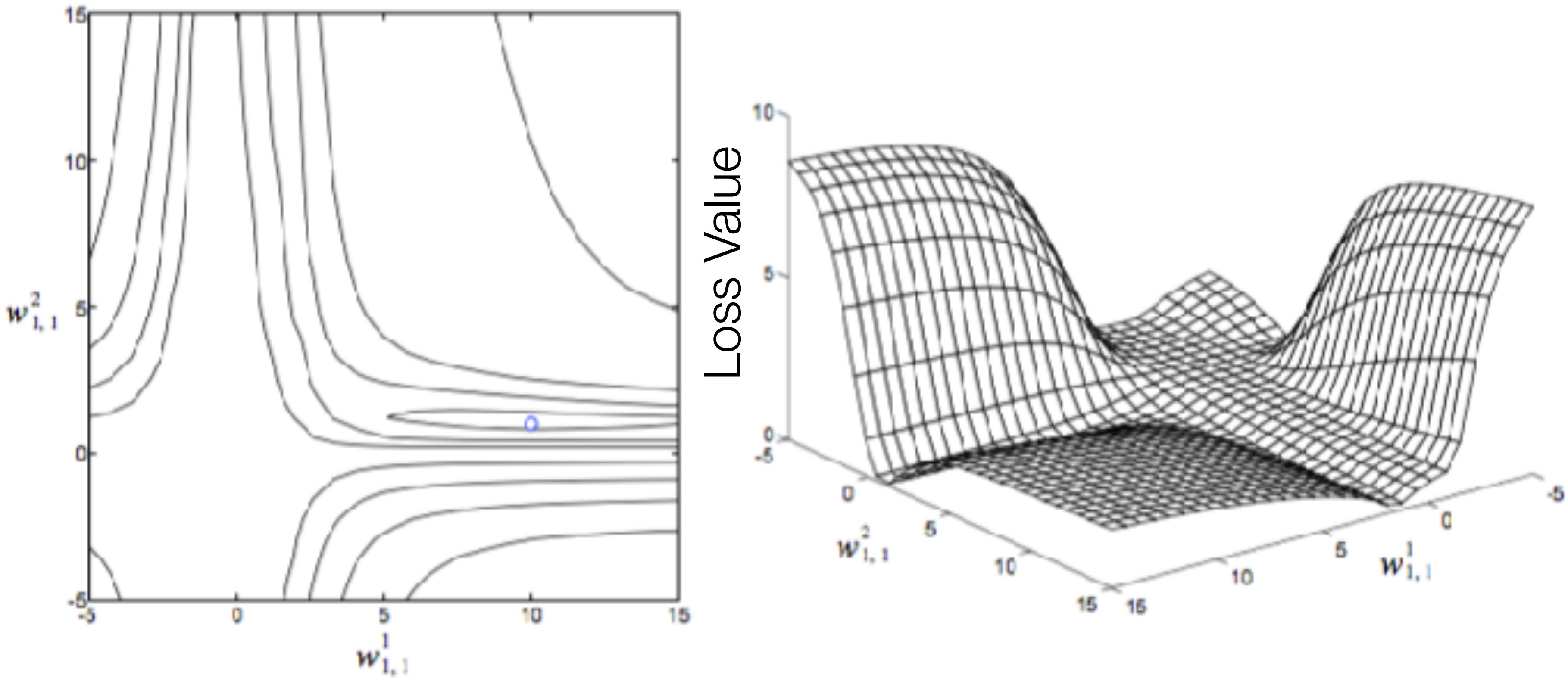
```
def print_message(num_of_times) {
    for i in range(num_of_times) {
        print("Bython is awesome!");
    }
}

if __name__ == "__main__" {
    print_message(10);
}
```

Bython

Python with braces. Because Python is awesome, but whitespace is awful.

Bython is a Python preprosessor which translates curly brackets into indentation.

A new Loss landscape

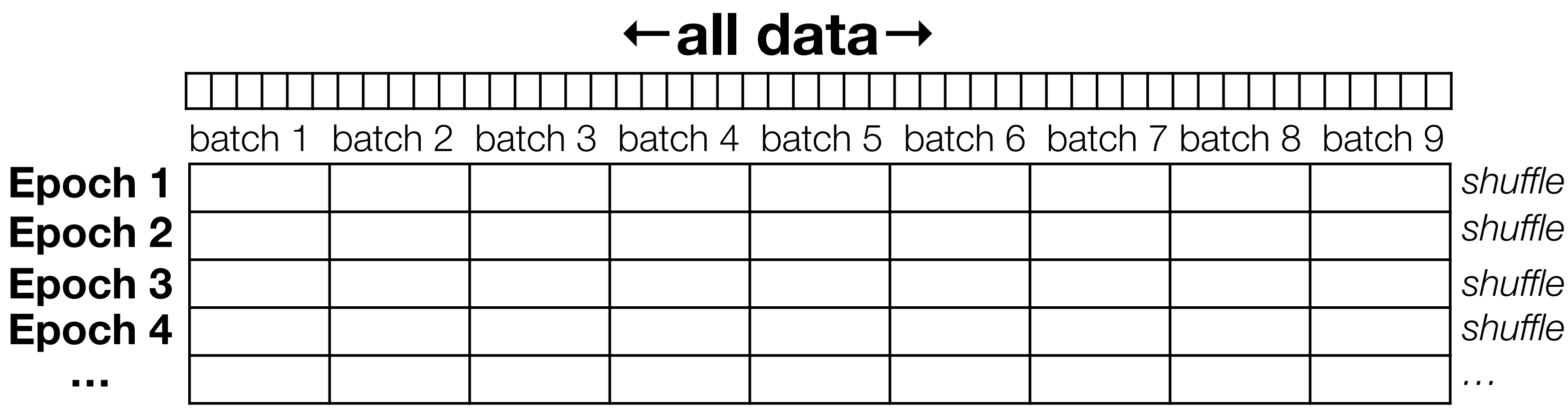


What are some optimization problems?

A. There are many local optima that gradients will be fooled by
B. There are many interconnected parameters that change each other
C. There are large flat areas in the loss function, where the gradient is small
D. All of the above

Mini-batching

- Numerous instances to find one gradient update
  - **solution**: mini-batch

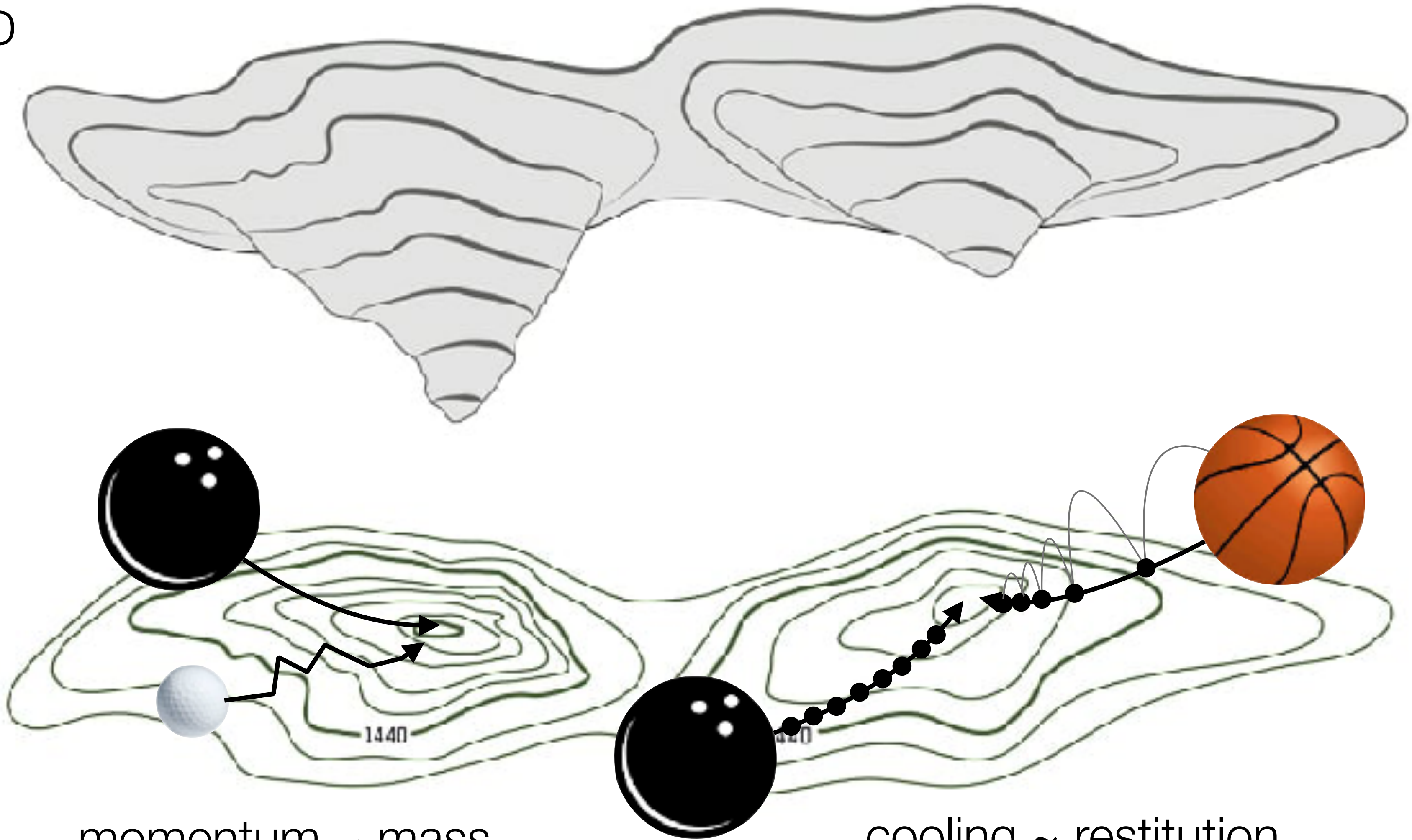


shuffle ordering ***each epoch*** *and update W's after* ***each batch***

- **Remaining problems**: there might be many local optima…
  - **solutions**:
    - momentum
    - adaptive learning rate (cooling)

Momentum and Cooling Intuition

3D



momentum ~ mass

cooling ~ restitution

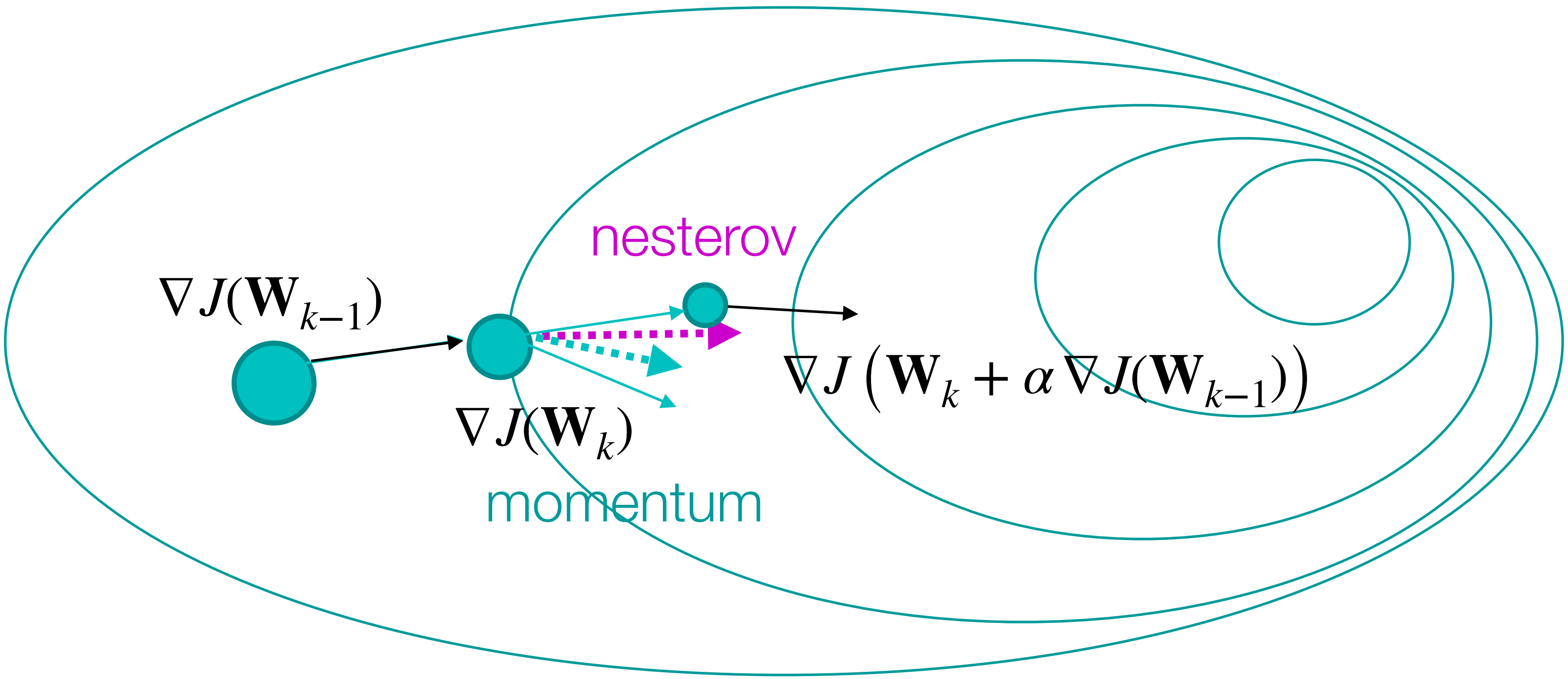
Topological

Momentum

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \rho_k$$

- Momentum $\rho_k = \eta \cdot \nabla J(\mathbf{W}_k) + \alpha \cdot \rho_{k-1}$

- Nesterov's Accelerated Gradient $\rho_k = \underbrace{\beta \nabla J \left(\mathbf{W}_k + \alpha \nabla J(\mathbf{W}_{k-1}) \right)}_{\text{step twice}} + \alpha \nabla J(\mathbf{W}_{k-1})$

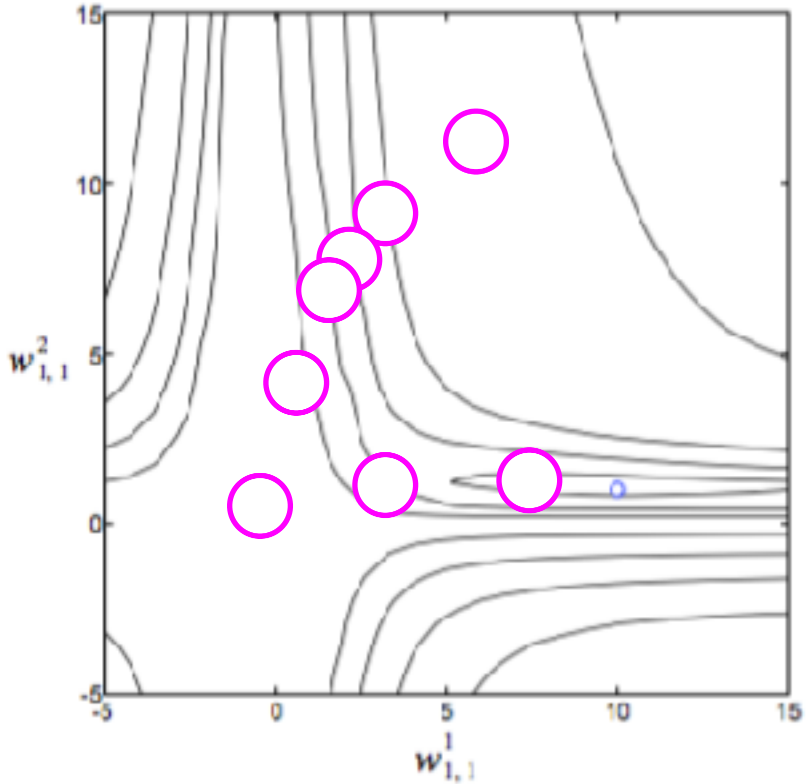


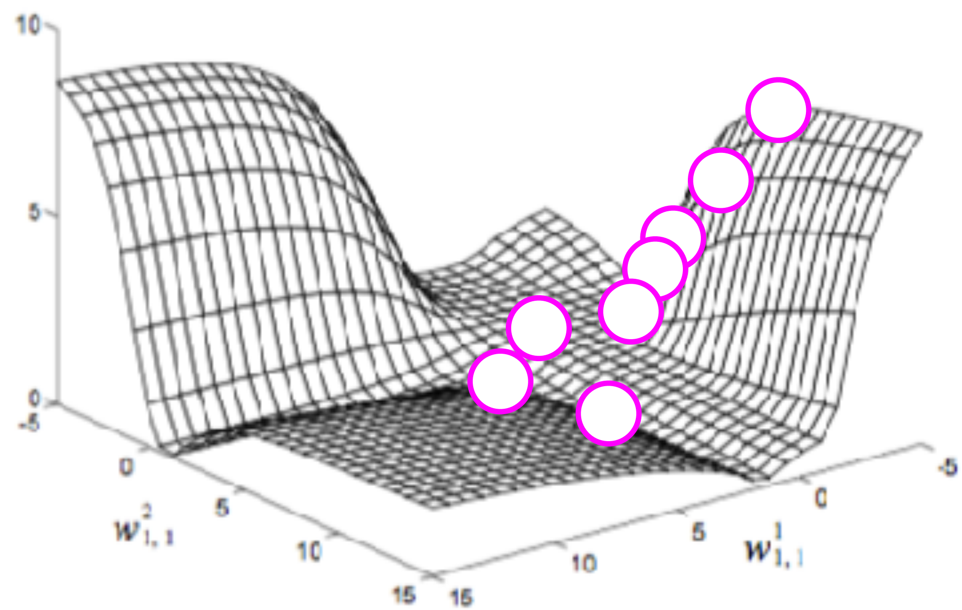
Cooling (Learning Rate Reduction)

- Fixed Reduction at Each Epoch, k

$\eta_k = \eta_0 \cdot d^{\lfloor \frac{k_{max}}{k} \rfloor}$	drop by d every k_d epochs
$\eta_k = \eta_0^{(1+k \cdot d)}$	drop a little every epoch

- Adjust on Plateau
  - make smaller when J rapidly changes
  - make bigger when J not changing much





Learning Rate Schedules

- Many scheduling rate functions exist and can be different for each application
- Some first increase plate and then decrease

- Cosine
- Linear Warm up

Demo

```
07. MLP Neural Networks.ipynb
```

comparison:

mini-batch

- momentum
- adaptive learning rate

L-BFGS (if time)

$$\rho_k = \eta \cdot \nabla J(\mathbf{W}_k) + \alpha \cdot \rho_{k-1}$$

$$\eta_k = \eta_0^{(1+k \cdot d)}$$