Lecture Notes for

Machine Learning in Python

⟦👨‍🏫, 👨‍💻, 🐍, 👨‍🔬⟧

Professor Eric Larson

Optimizing Neural Networks

Class Logistics and Agenda

- Logistics
  - Grading
  - Team canvas assignment turn in
- 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

Review: Back propagation history

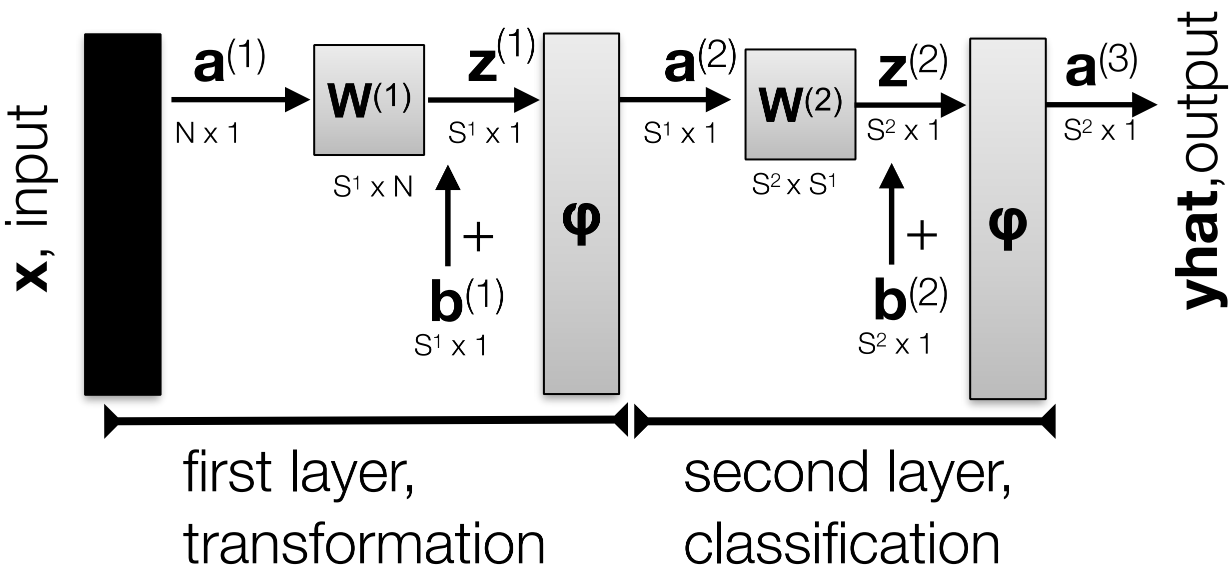
- 1986: *Rumelhart*, *Hinton*, and *Williams* popularize gradient calculation for multi-layer network
  - *actually* introduced by Werbos in 1982
- **difference**: Rumelhart *et al.* validated ideas with a computer
- until this point no one could train a multiple layer network consistently
- algorithm 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 $\mathbf{W}^{(l)}$

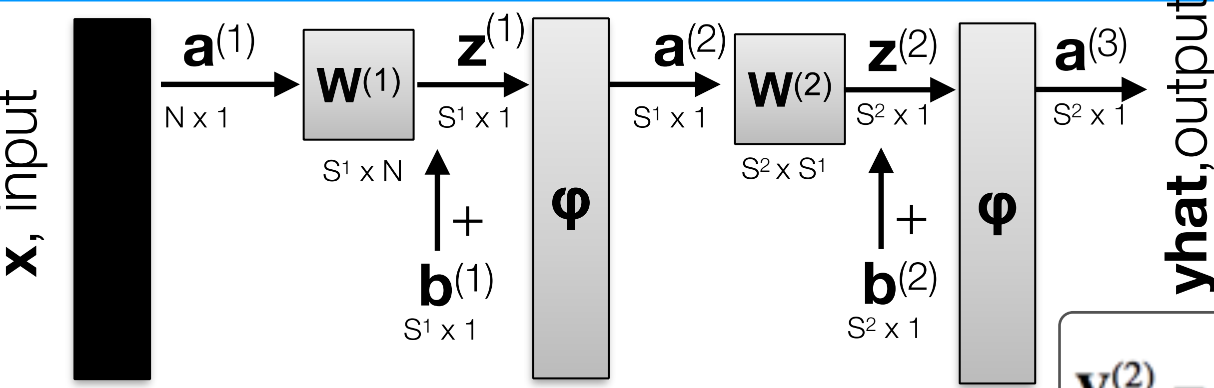


$$J(\mathbf{W}) = \| \mathbf{Y} - \hat{\mathbf{Y}} \|^2$$

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

**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!

Lightning Demo

`07a. MLP Neural Networks with bias.ipynb`

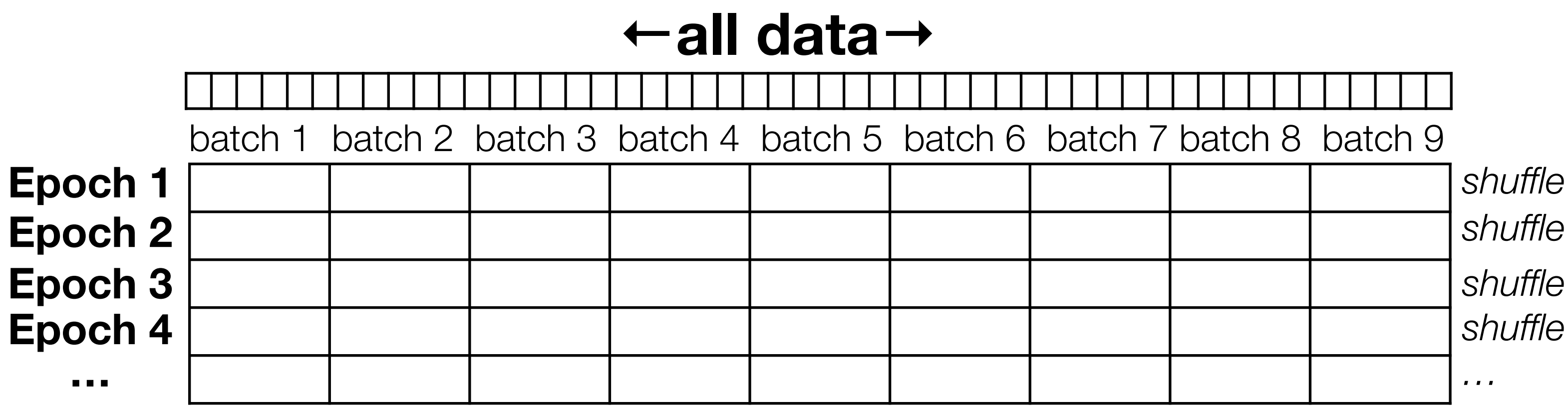
same as Flipped Assignment!
with regularization
and vectorization
and mini-batching

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

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

Mini-batching

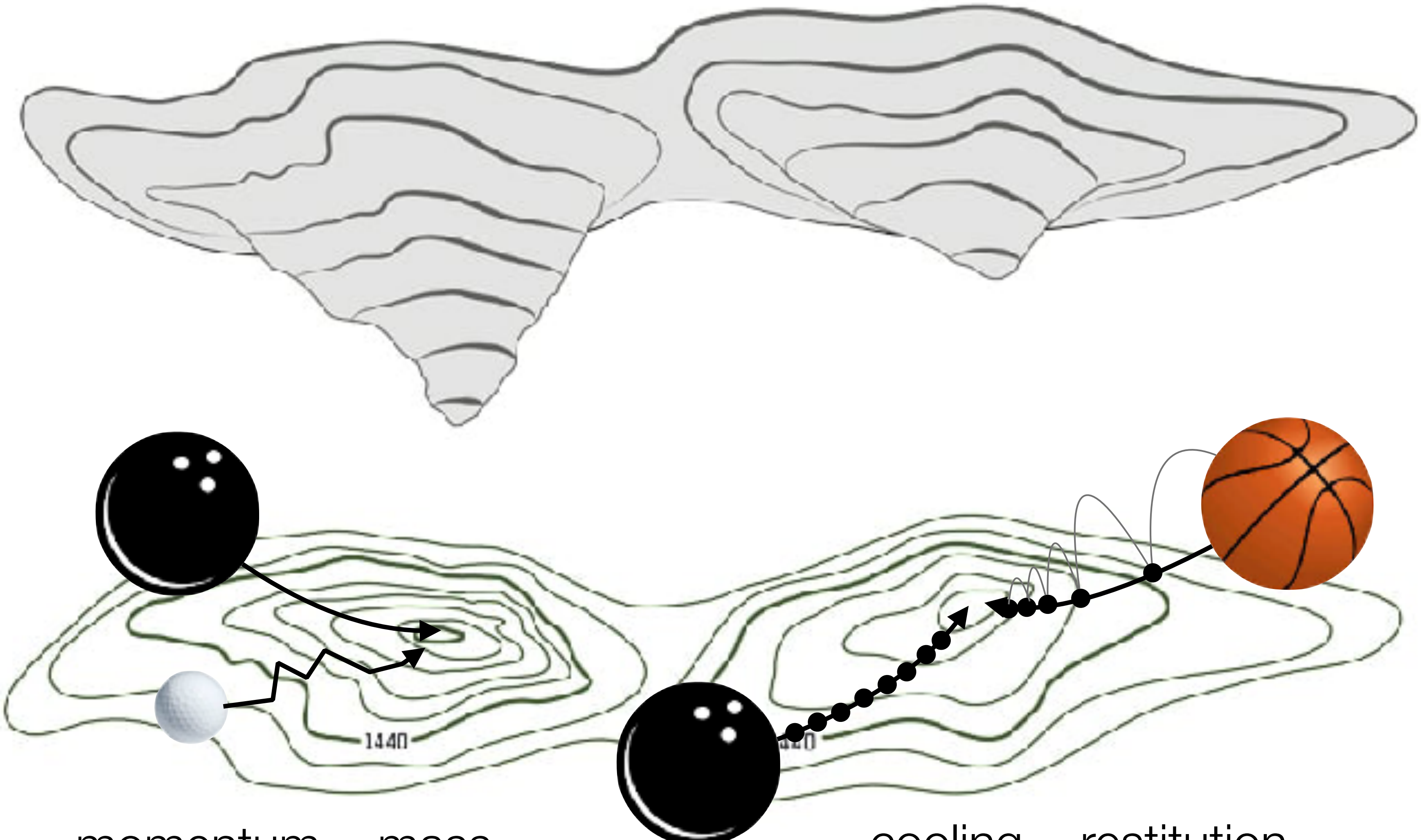
- Numerous instances to find one gradient update
  - **solution**: mini-batch



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

- **new problem**: mini-batch gradient updates can be erratic and there might be many local optima…
  - **solutions**:
    - momentum
    - adaptive learning rate (cooling)

Momentum and Cooling Intuition



momentum ~ mass

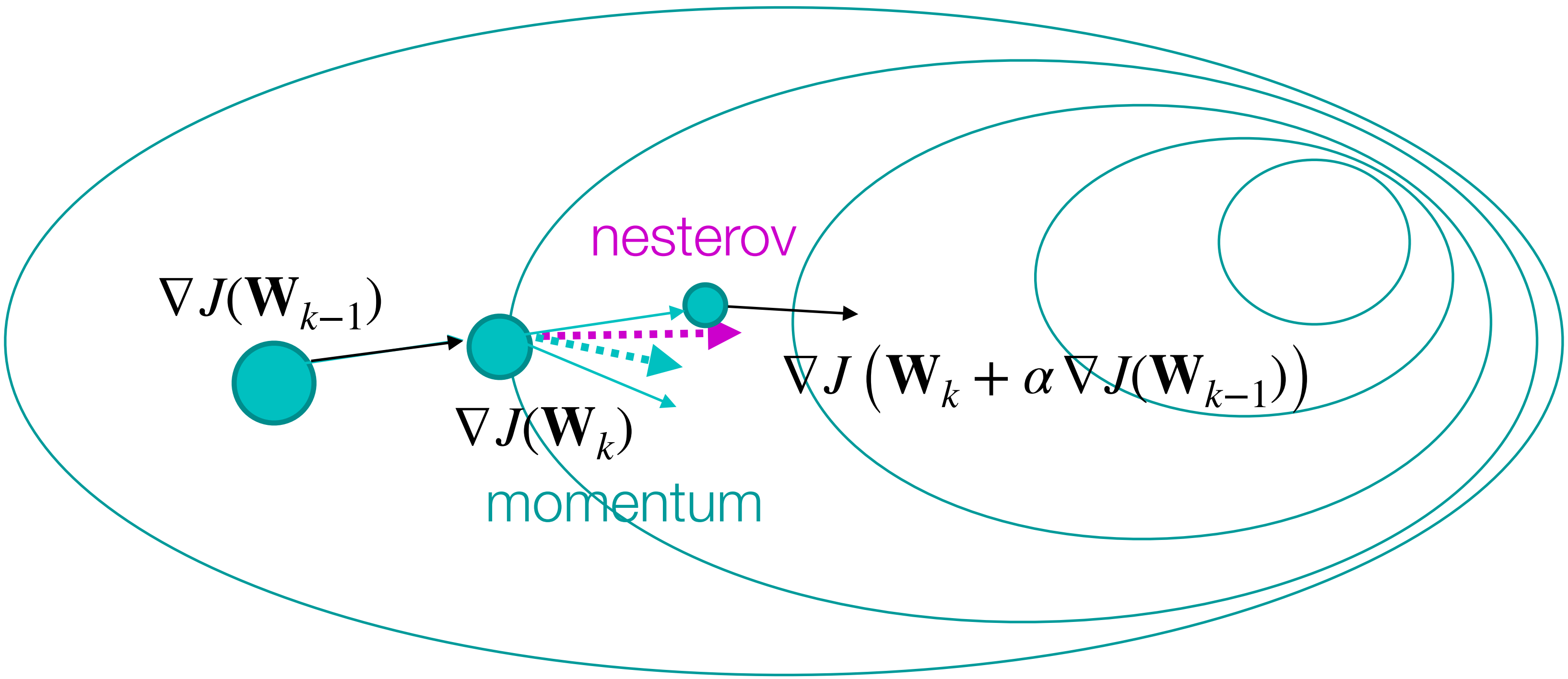
cooling ~ restitution

Momentum

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

- Momentum $\rho_k = \alpha \nabla J(\mathbf{W}_k) + \beta \nabla J(\mathbf{W}_{k-1})$

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

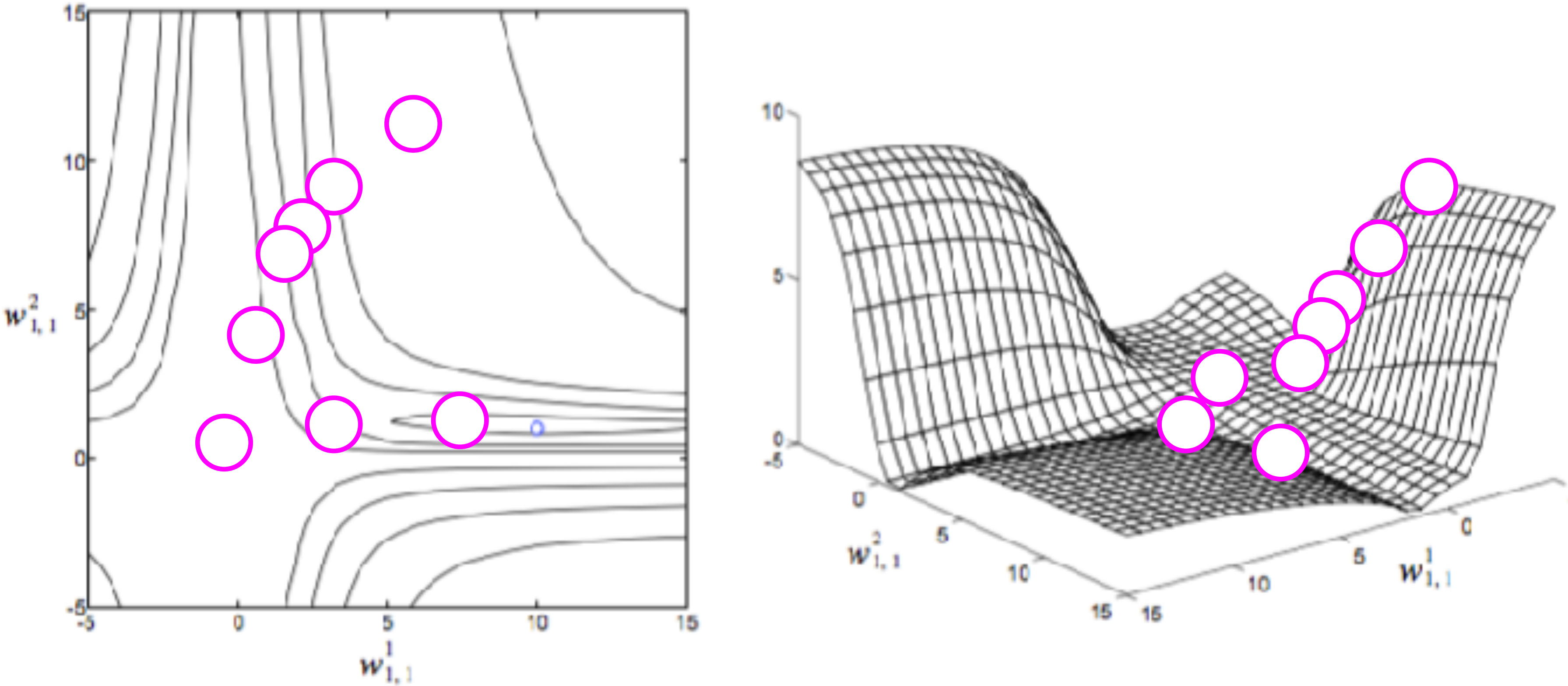


Cooling (Learning Rate Reduction)

- Fixed Reduction at Each Epoch, k
- Adjust on Plateau
  - make smaller when J rapidly changes
  - make bigger when J not changing much

$$\eta_k = \eta_0 \cdot d^{\lfloor \frac{k_{max}}{k} \rfloor}$$ drop by d every k_d epochs

$$\eta_k = \eta_0^{(1+k \cdot d)}$$ drop a little every epoch



07. MLP Neural Networks.ipynb

comparison:
mini-batch
- momentum
- adaptive learning rate

L-BFGS (if time)