Lecture Notes for

Machine Learning in Python

⟦👨‍🏫, 👨‍💻, 🐍, 👨‍🔬⟧

Professor Eric Larson

Neural Network Optimization and Activation

Class Logistics and Agenda

- Agenda:
  - More optimization techniques
    - Momentum
    - Adaptive learning rates
    - Initialization
    - More activations: Tanh, ReLU, SiLU
  - Programming Examples

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

Last Time

`07. MLP Neural Networks.ipynb`

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

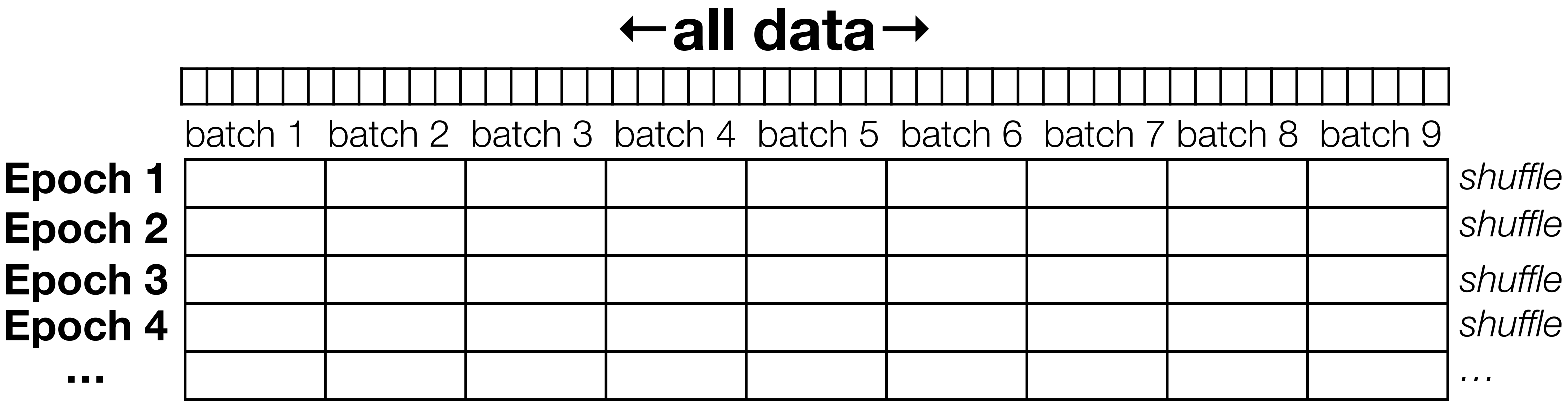
Self test: Should we see examples where:

A. $\mathbf{z} = \mathbf{W} \cdot \mathbf{a}_{bias}$ where bias is concatenated, and $\mathbf{W}$ incorporates bias term?

B. $\mathbf{z} = \mathbf{W} \cdot \mathbf{a} + \mathbf{b}$ where we separate out the bias explicitly ?

Mini-batching

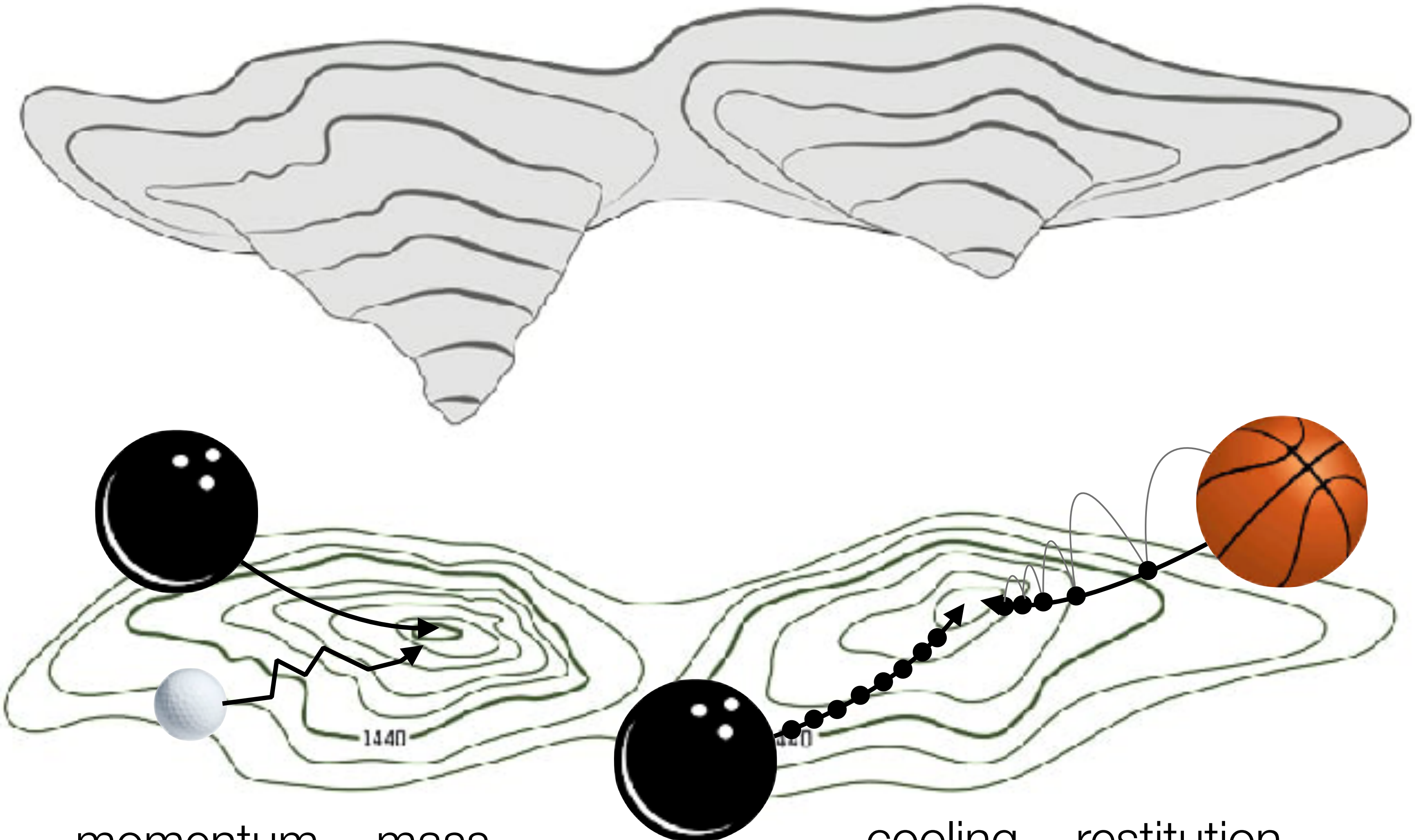
- Numerous instances to find one gradient update
  - **solution**: mini-batch



*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

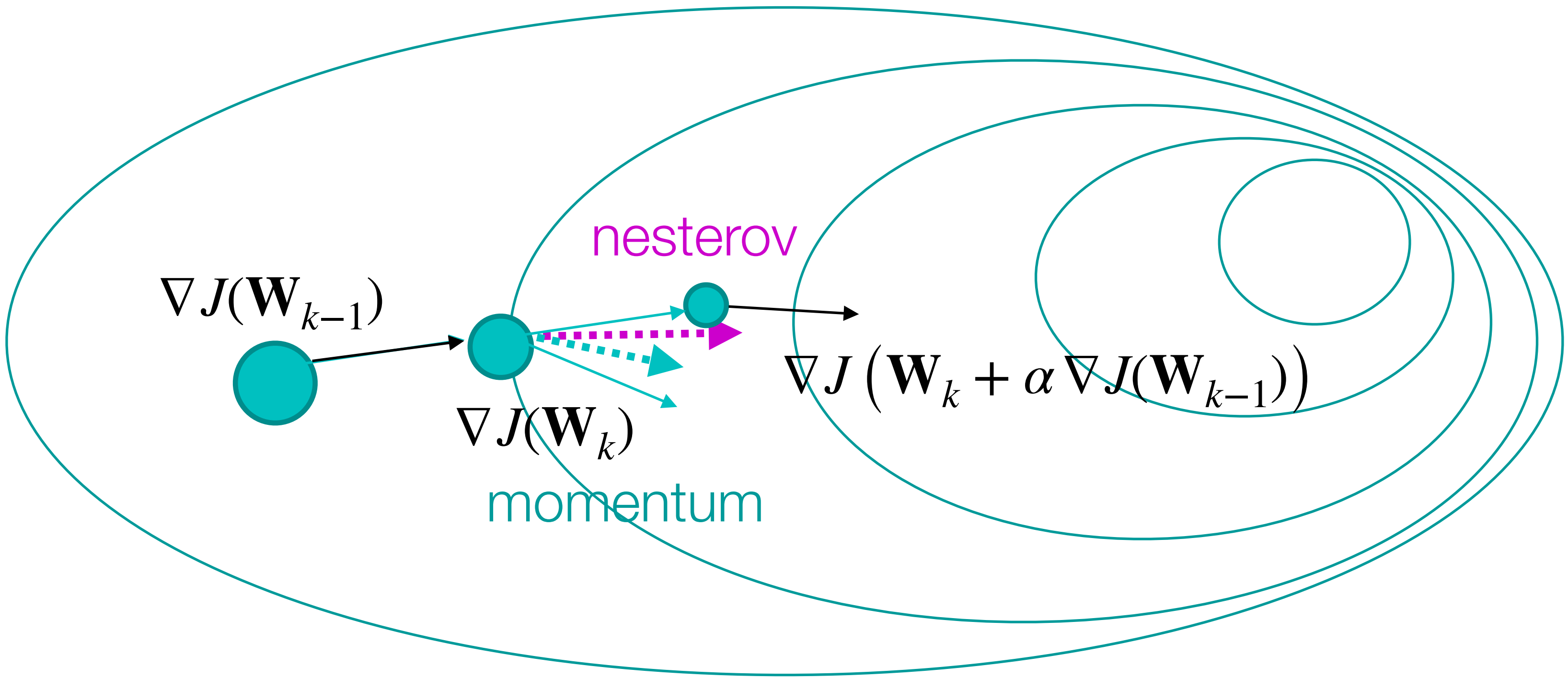
cooling ~ restitution

Momentum

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

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

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



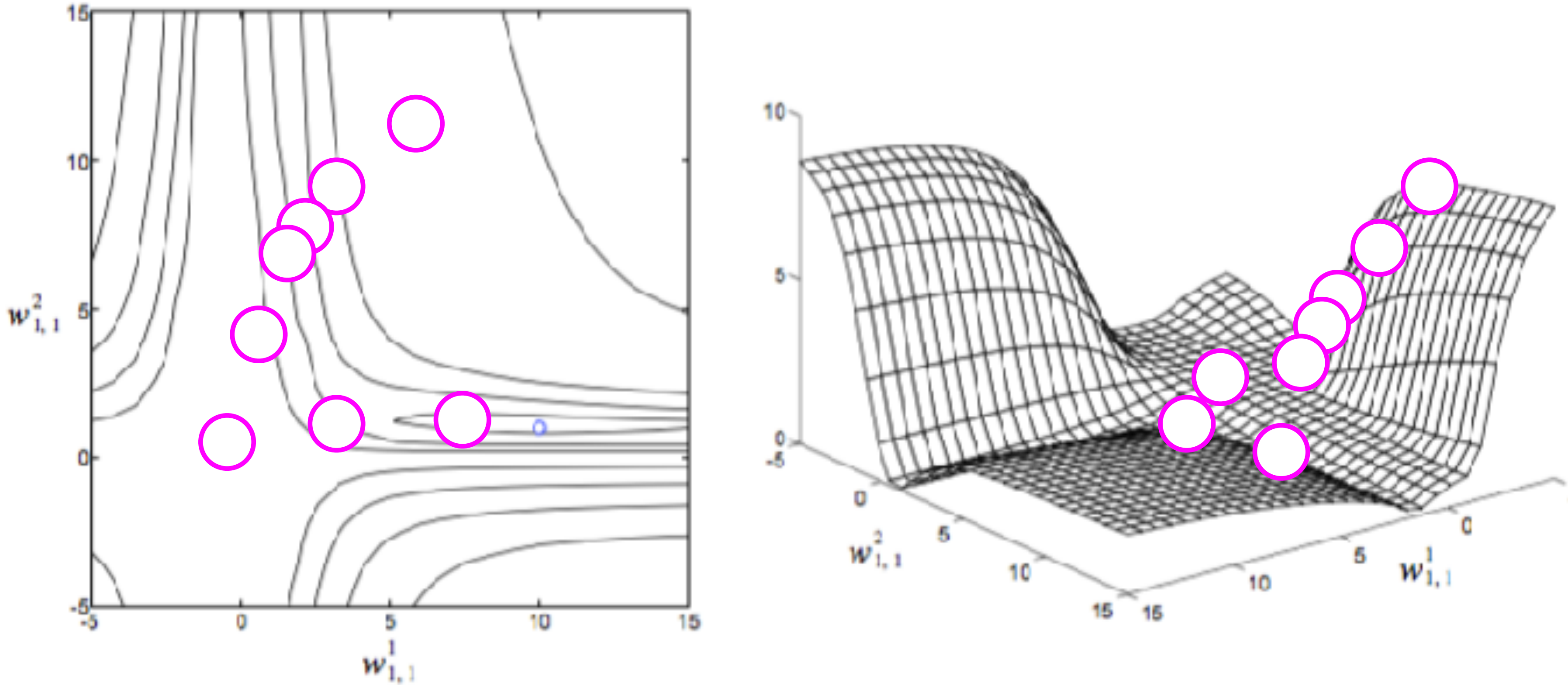
Cooling (Learning Rate Reduction)

- 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



Demo

`07. MLP Neural Networks.ipynb`

comparison:

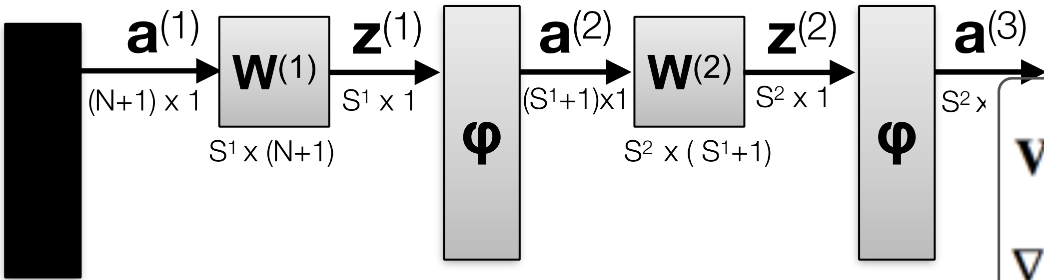
mini-batch

- momentum
- adaptive learning rate

L-BFGS (if time)

Objective Function

Changing the Objective Function



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

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

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

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

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

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

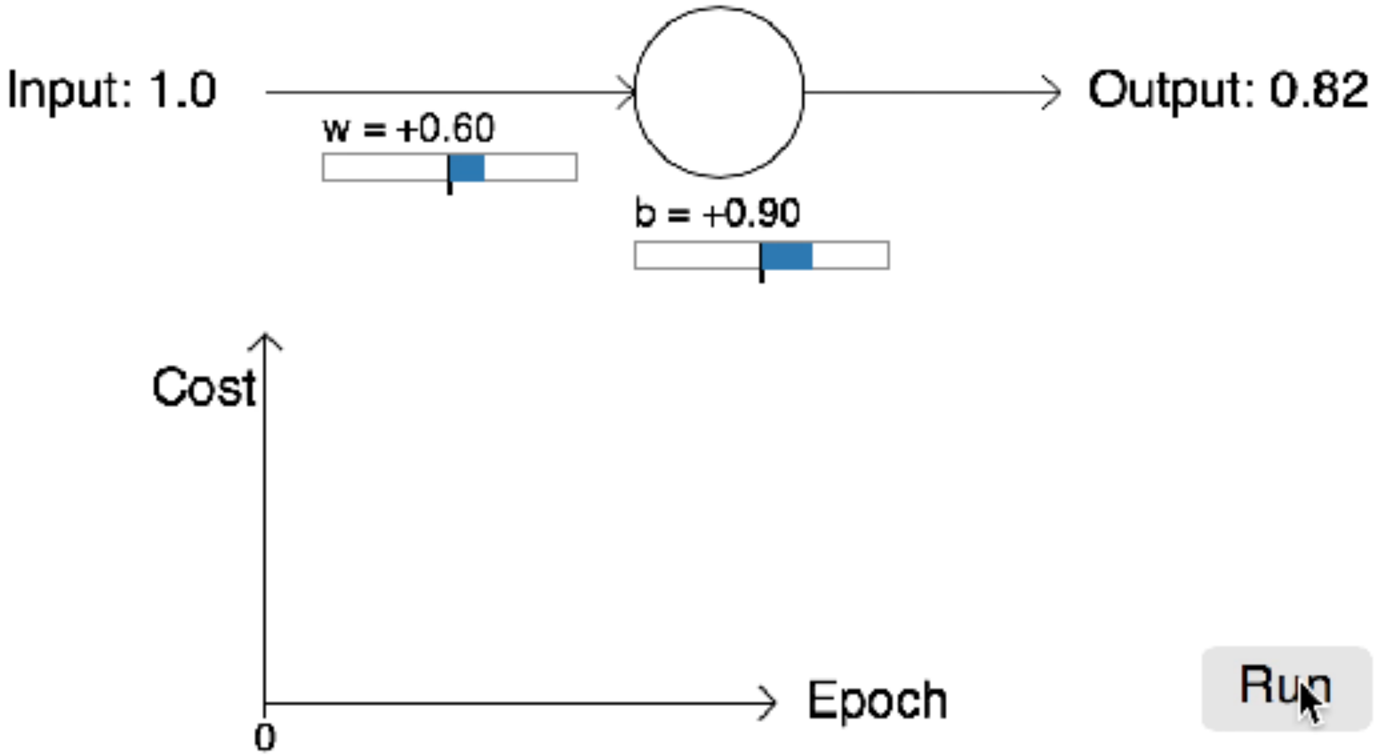
- **Self Test**:
  True or False: If we change the cost function, $J(\mathbf{W})$, we only need to update the final layer sensitivity calculation, $\mathbf{V}^{(2)}$, of the back propagation steps. The remainder of the algorithm is unchanged.
  - A. True
  - B. False

Practical Implementation of Architectures

- MSE

$$J(\mathbf{W}) = \sum_{k}^{M} (\mathbf{y}^{(k)} - \mathbf{a}^{(L)})^2$$

least squares objective,
tends to slow training initially

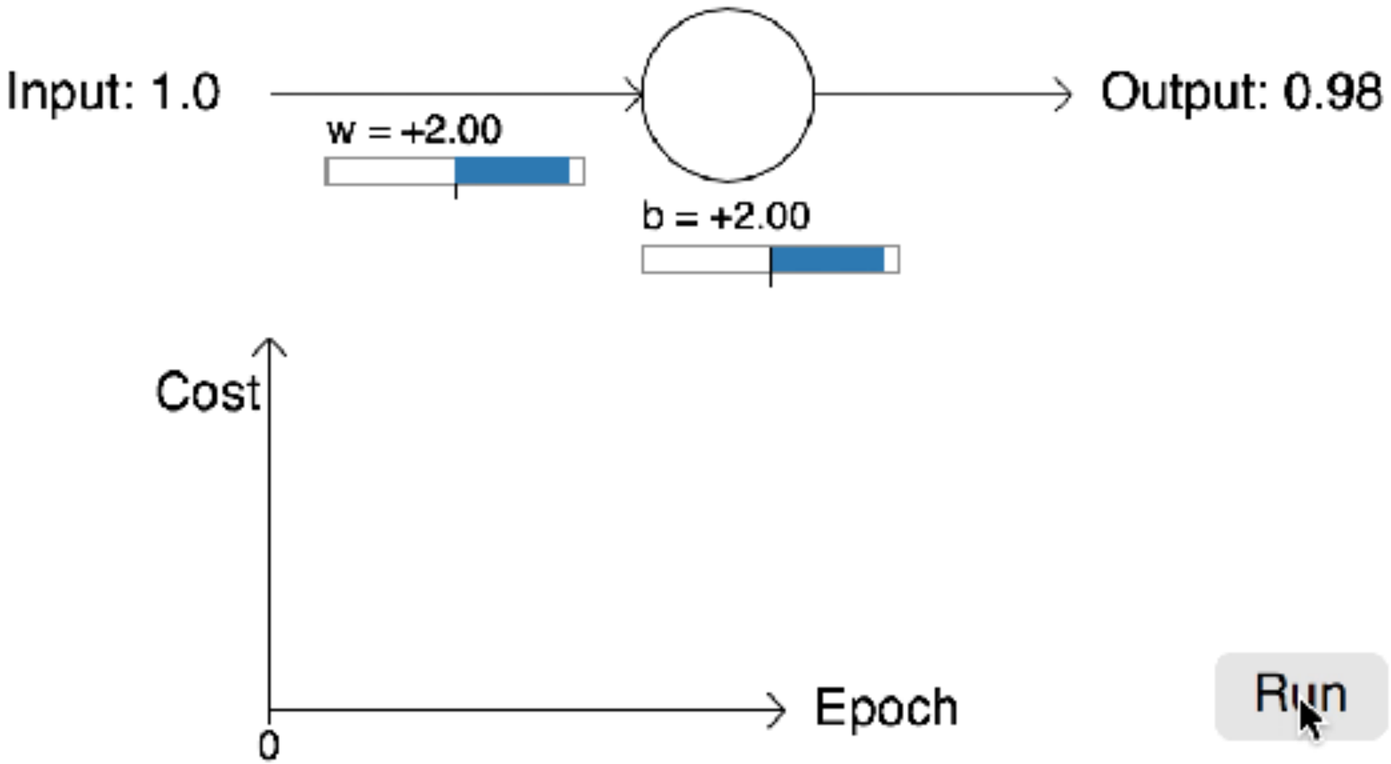


Neural Networks and Deep Learning, Michael Nielson, 2015

- MSE

$$J(\mathbf{W}) = \sum_{k}^{M} (\mathbf{y}^{(k)} - \mathbf{a}^{(L)})^2$$

least squares objective,
tends to slow training initially

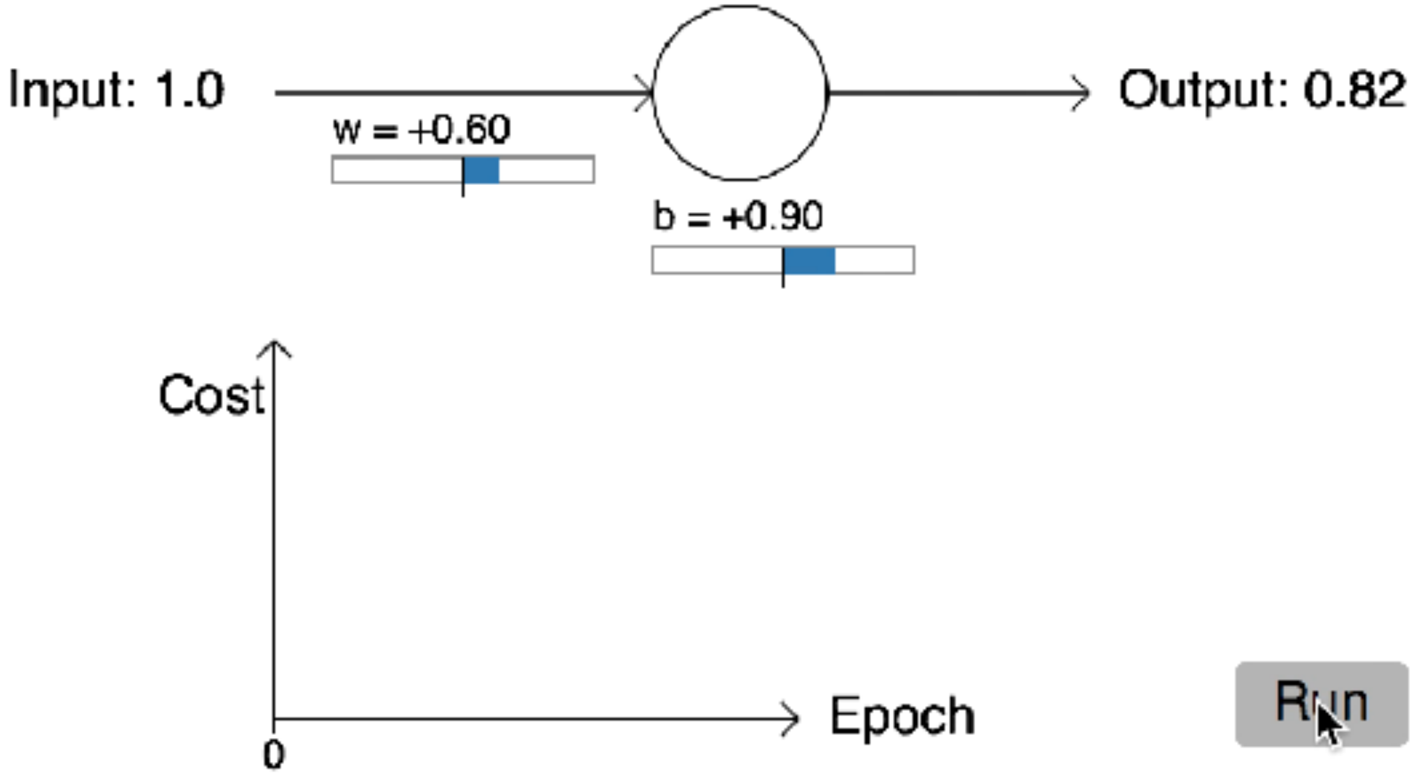


Neural Networks and Deep Learning, Michael Nielson, 2015

- Negative of MLE: **Binary Cross entropy**

$$J(\mathbf{W}) = -\left[\mathbf{y}^{(i)}\ln([\mathbf{a}^{(L+1)}]^{(i)}) + (1-\mathbf{y}^{(i)})\ln(1-[\mathbf{a}^{(L+1)}]^{(i)})\right]$$

speeds up initial training



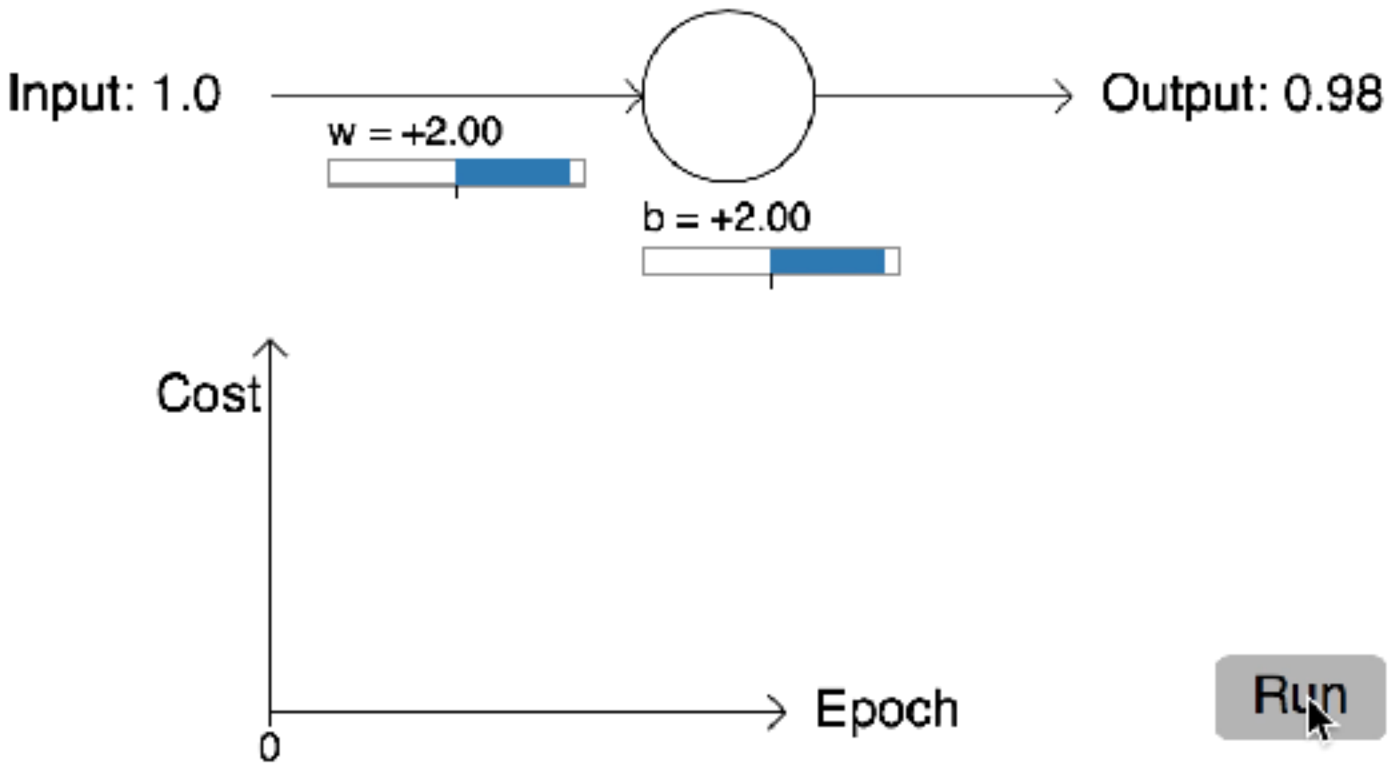
Neural Networks and Deep Learning, Michael Nielson, 2015

Practical Implementation of Architectures

- Negative of MLE: **Binary Cross entropy**

$$J(\mathbf{W}) = -\left[\mathbf{y}^{(i)}\ln([\mathbf{a}^{(L+1)}]^{(i)}) + (1-\mathbf{y}^{(i)})\ln(1-[\mathbf{a}^{(L+1)}]^{(i)})\right]$$

speeds up initial training



Neural Networks and Deep Learning, Michael Nielson, 2015

Practical Implementation of Architectures

$$J(\mathbf{W}) = -\left[\mathbf{y}^{(i)}\ln([\mathbf{a}^{(L+1)}]^{(i)}) + (1-\mathbf{y}^{(i)})\ln(1-[\mathbf{a}^{(L+1)}]^{(i)})\right]$$

likely to speed up initial training

$$\left[\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(L)}}\right]^{(i)} = -\frac{\partial}{\partial \mathbf{z}^{(L)}}\left[\mathbf{y}^{(i)}\ln([\mathbf{a}^{(L+1)}]^{(i)}) + (1-\mathbf{y}^{(i)})\ln(1-[\mathbf{a}^{(L+1)}]^{(i)})\right]$$

only $\mathbf{a}$ has dependence on $\mathbf{z}$

$$= -\left[\mathbf{y}^{(i)}\frac{\partial}{\partial \mathbf{z}^{(L)}}\left(\ln([\mathbf{a}^{(L+1)}]^{(i)})\right) + (1-\mathbf{y}^{(i)})\frac{\partial}{\partial \mathbf{z}^{(L)}}\left(\ln(1-[\mathbf{a}^{(L+1)}]^{(i)})\right)\right]$$

$$= -\left[\mathbf{y}^{(i)}\frac{1}{[\mathbf{a}^{(L+1)}]^{(i)}}\left(\frac{\partial}{\partial \mathbf{z}^{(L)}}[\mathbf{a}^{(L+1)}]^{(i)}\right) + \frac{(1-\mathbf{y}^{(i)})}{1-[\mathbf{a}^{(L+1)}]^{(i)}}\left(-\frac{\partial}{\partial \mathbf{z}^{(L)}}[\mathbf{a}^{(L+1)}]^{(i)}\right)\right]$$

$$= -\left[\mathbf{y}^{(i)}\frac{1}{\cancel{[\mathbf{a}^{(L+1)}]^{(i)}}}\left(\cancel{[\mathbf{a}^{(L+1)}]^{(i)}}(1-[\mathbf{a}^{(L+1)}]^{(i)})\right) - \frac{(1-\mathbf{y}^{(i)})}{\cancel{1-[\mathbf{a}^{(L+1)}]^{(i)}}}\left([\mathbf{a}^{(L+1)}]^{(i)}\cancel{(1-[\mathbf{a}^{(L+1)}]^{(i)})}\right)\right]$$

$$= -\left[\mathbf{y}^{(i)}\left(1-[\mathbf{a}^{(L+1)}]^{(i)}\right) - (1-\mathbf{y}^{(i)})\left([\mathbf{a}^{(L+1)}]^{(i)}\right)\right]$$

$$= -\left[\mathbf{y}^{(i)} \cancel{-\mathbf{y}^{(i)}[\mathbf{a}^{(L+1)}]^{(i)}} - [\mathbf{a}^{(L+1)}]^{(i)} \cancel{+[\mathbf{a}^{(L+1)}]^{(i)}\mathbf{y}^{(i)}}\right] \qquad = [\mathbf{a}^{(L+1)}]^{(i)} - \mathbf{y}^{(i)}$$

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

Practical Implementation of Architectures

- Back to our old friend: **Cross entropy**

$$J(\mathbf{W}) = -\left[\mathbf{y}^{(i)} \ln([\mathbf{a}^{(L+1)}]^{(i)}) + (1-\mathbf{y}^{(i)})\ln(1-[\mathbf{a}^{(L+1)}]^{(i)})\right]$$

likely to speed up initial training

$$\left[\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(L)}}\right]^{(i)} = [\mathbf{a}^{(L+1)}]^{(i)} - \mathbf{y}^{(i)}$$

$$\left[\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}}\right]^{(i)} = [\mathbf{a}^{(3)}]^{(i)} - \mathbf{y}^{(i)}$$

two layer network

$$\mathbf{A}^{(3)} - \mathbf{Y}$$

new update

```
# vectorized backpropagation
V2 = (A3-Y_enc) # <- this is only line t
V1 = A2*(1-A2)*(W2.T @ V2)

grad2 = V2 @ A2.T
grad1 = V1[1:,:] @ A1.T
```

bp-5

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

08. Practical_NeuralNets.ipynb

Demo

cross entropy

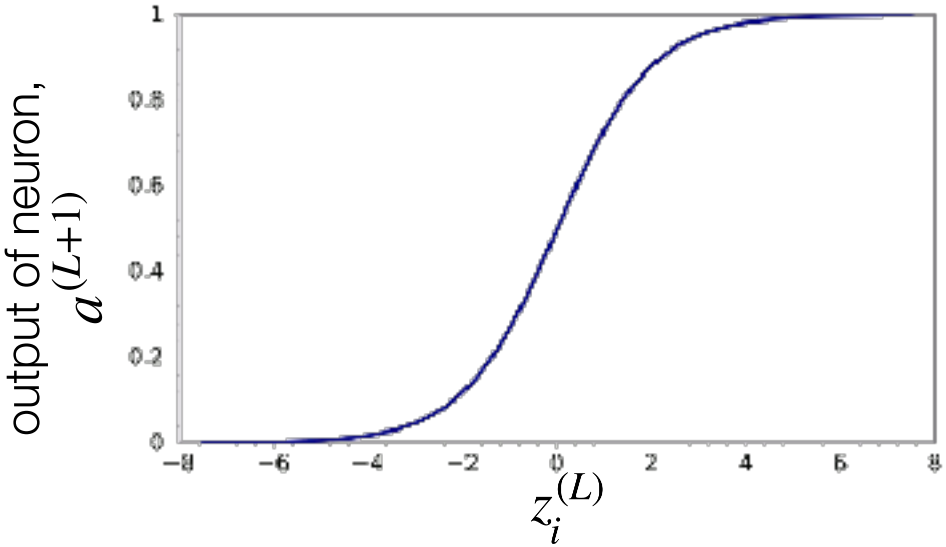
Practical Implementation of Architectures

Formative Self Test

- for adding Gaussian random variables, variances add together

$$\mathbf{a}^{(L+1)}=\phi(\mathbf{W}^{(L)}\mathbf{a}^{(L)})$$ assume each element of $\mathbf{a}$ is Gaussian

- If you initialized the weights, $\mathbf{W}$, with too large variance, you would expect the output of the neuron, $\mathbf{a}^{(L+1)}$, to be:
    - A. saturated to “1”
    - B. saturated to “0”
    - C. could either be saturated to “0” or “1”
    - D. would not be saturated

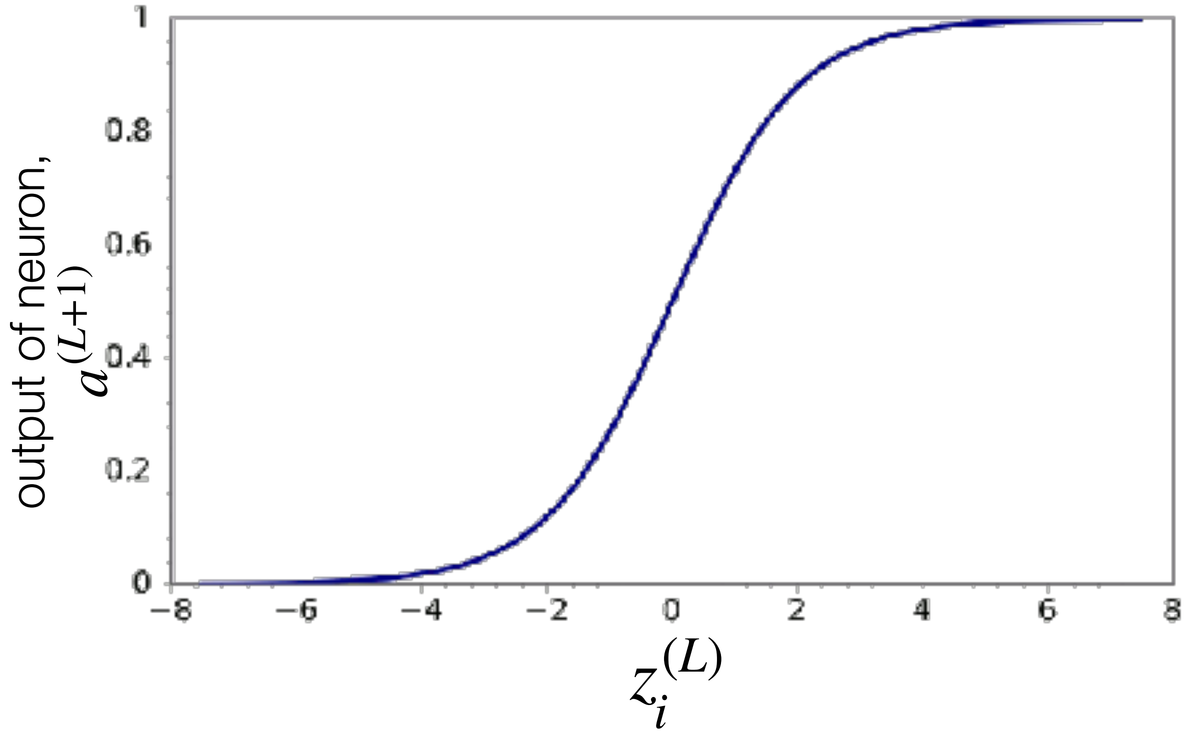


Formative Self Test

- for adding Gaussian distributions, variances add together

$\mathbf{a}^{(L+1)}=\phi(\mathbf{W}^{(L)}\mathbf{a}^{(L)})$ assume each element of $\mathbf{a}$ is Gaussian

- What is the derivative of a saturated sigmoid neuron?
  - A. zero
  - B. one
  - C. $a \times (1 - a)$
  - D. it depends

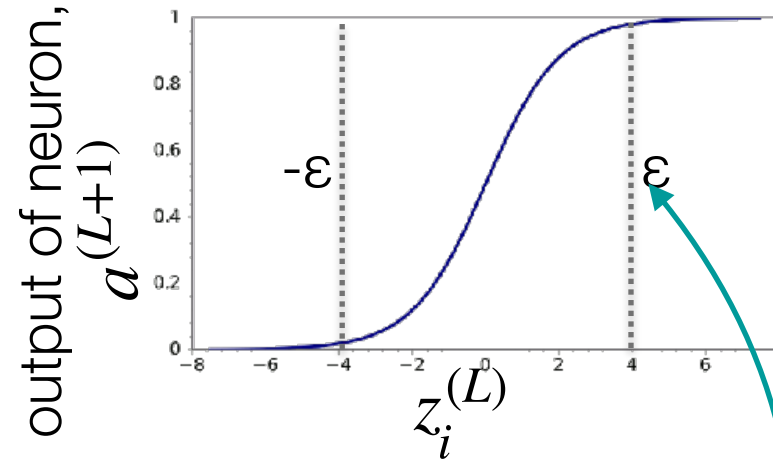


Practical Implementation of Architectures

- **Weight initialization**
  - try not to **saturate** your neurons right away!

$$\mathbf{a}^{(L+1)} = \phi(\mathbf{z}^{(L)})$$
$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{a}^{(L)} \qquad z_i^{(L)} = \sum_j^{n^{(L)}} w_{ij} a_j^{(L)}$$

each row is summed before sigmoid



want each $z_i^{(L)}$ to be between -ε<Σ<ε for no saturation

solution: squash initial weights magnitude

- one choice: each element of **W** selected from a Gaussian with **zero mean** and **specific standard deviation**

$$w_{ij}^{(L)} \approx \mathcal{N}\left(0,\ 4 \cdot \sqrt{\frac{1}{n^{(L)}}}\right)$$

For a sigmoid if, $-\epsilon < z_i^{(L)} < \epsilon$ where $\epsilon = 4$

then $a^{(L+1)}$ is well distributed [0,1]