acm.ai

Feeding Forward and Backpropagation

Advanced Track Workshop #2

Anonymous feedback: forms.gle/QQbKfqEoLnviAooZ9

GitHub: github.com/uclaacmai/advanced-track-winter21

Attendance Code: bobsleigh

Officers Helping w/ Advanced Track

Sudhanshu Agrawal
(he/him)

Jenson Choi
(he/him)

Adithya Nair
(he/him)

acm.ai

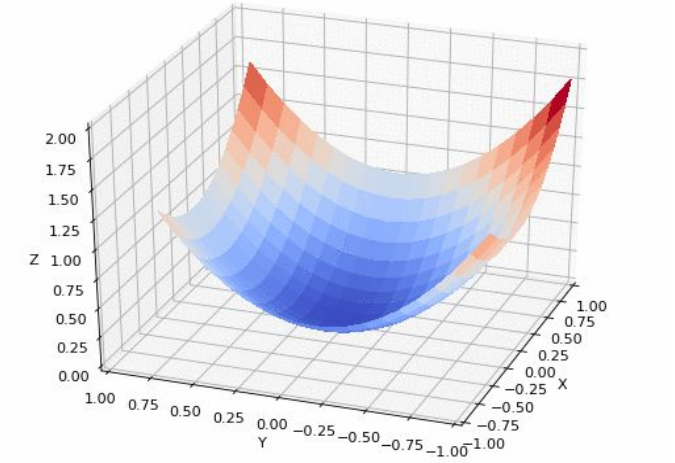
Advanced Track Tentative Schedule

- Week 1: Logistic Regression + Feedforward Networks Intro
- Week 2: Logistic Regression Review + Feedforward/Backpropagation
- Week 3: Convolutional Neural Networks
- Week 4: Convolutional Neural Networks 2
- Week 5: Recurrent Neural Networks (LSTMs)
- Week 6: Introduction to PyTorch
- Week 7: Guided Project
- Week 8: Guided Project

Thursdays 7:00 - 9:00 PM PDT

Gradient Descent

- Iterations of gradient descent continue until the cost function converges to a local minimum



acm.ai

Partial Derivative Exercise

Let $h(y, z) = ye^{-z}$. Calculate $\frac{\partial h}{\partial y}$ and $\frac{\partial h}{\partial z}$.

$$\frac{\partial h}{\partial y} = e^{-z}$$
$$\frac{\partial h}{\partial z} = -ye^{-z}$$

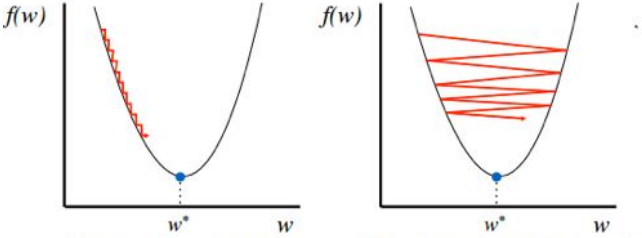
Let $z(s, u) = s^2 + su + u^2$. Calculate $\frac{\partial z}{\partial s}$ and $\frac{\partial z}{\partial u}$.

$$\frac{\partial z}{\partial s} = 2s + u$$
$$\frac{\partial z}{\partial u} = s + 2u$$

acm.ai

Learning Rate

- Determines by how much we adjust weights with respect to loss gradient
- Smaller the learning rate, the slower we converge to cost function minimum

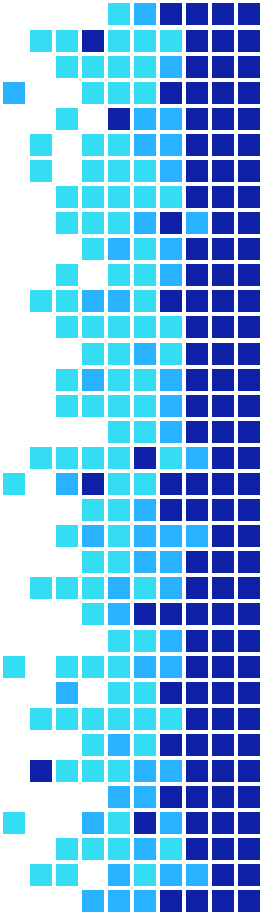


Too small: converge very slowly

Too big: overshoot and even diverge

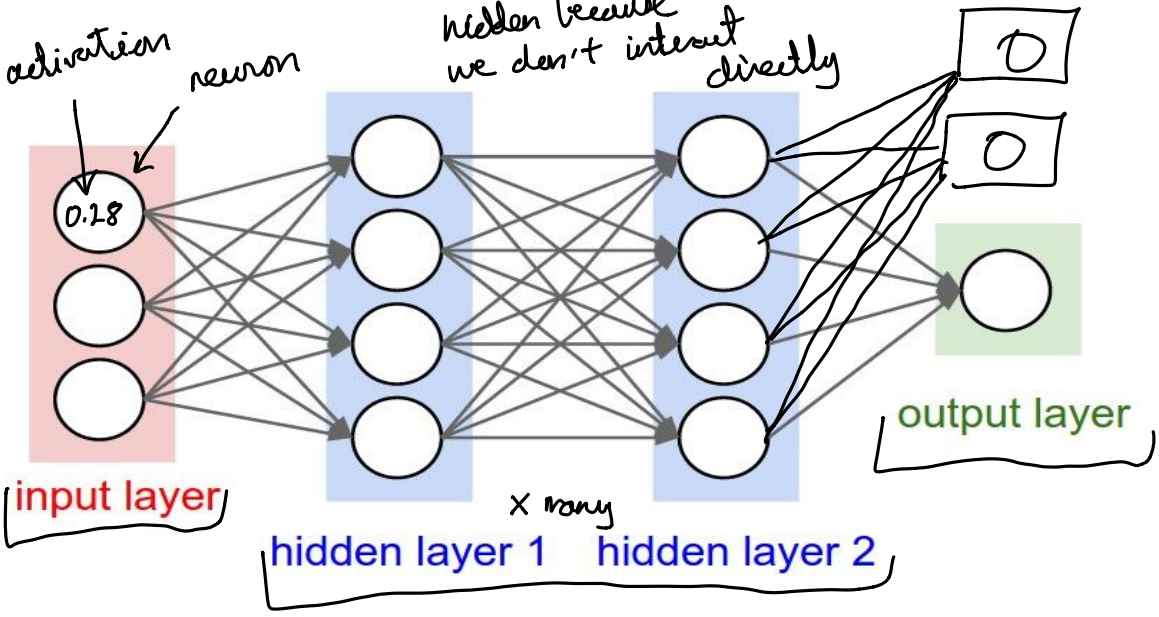
- Traditional default: 0.1 or 0.01
- ADAM, Momentum, and RMSprop are variants of gradient descent that help overcome these problems (attend Advanced++ for more!)
- Simulation made by ACM Hack

2. Neural Networks: Overview

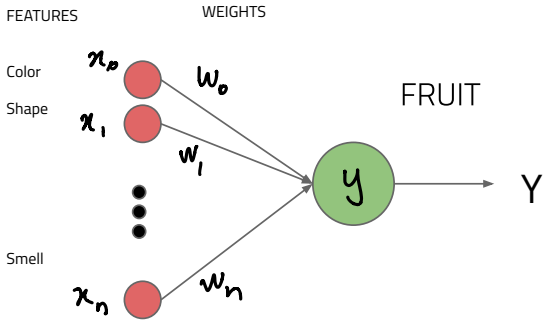


acm.ai

Neural Networks



Artificial Neuron

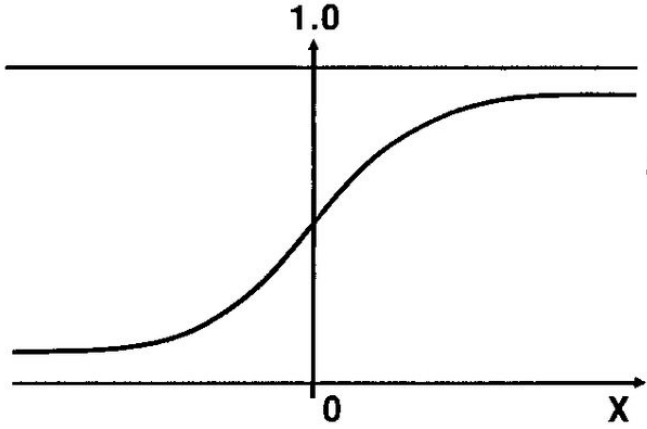


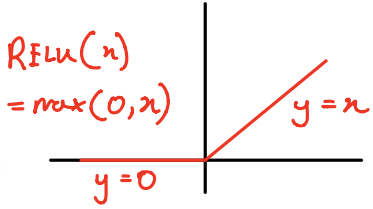
- Logistic Regression
  - Weights Vector w
  - Activation Function σ
  - Bias b
- Recall: Logistic Regression Equation
  - Y = σ(xw + b)

$$y = \sigma(w_0 x_0 + w_1 x_1 + \cdots w_n x_n + b)$$

Activation

- Measure of how positive the relevant weighted sum is
- $Y = \sigma(xw + b)$



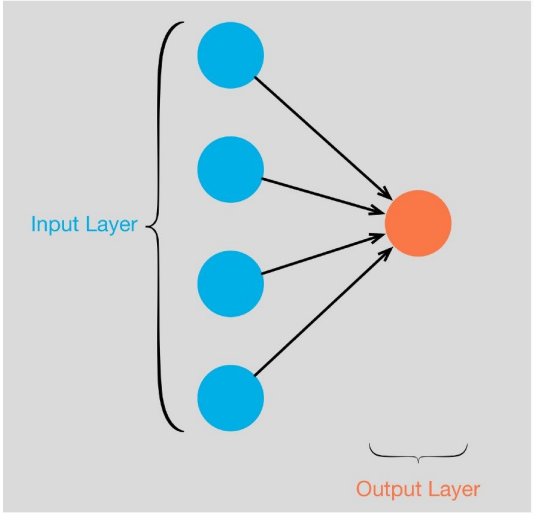


Sigmoid

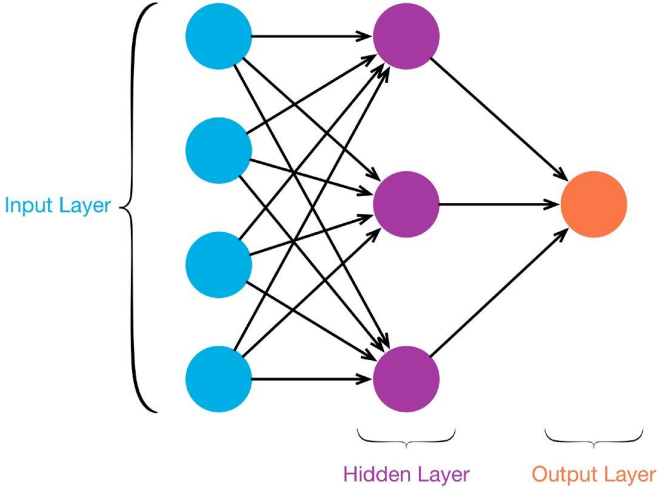
$$f(x) = \frac{1}{1+e^{-x}}$$

Recap

Logistic Regression

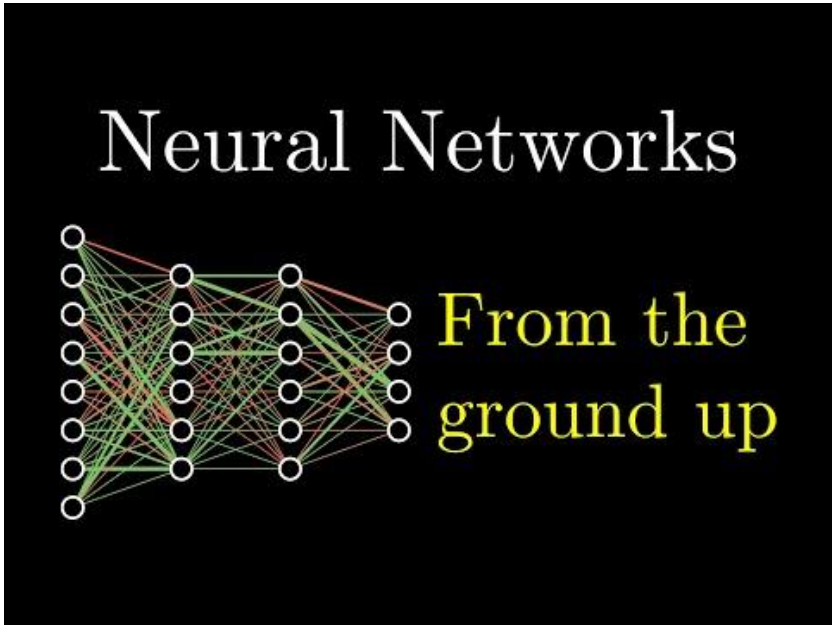


Single layer NN



Why do we need biases?

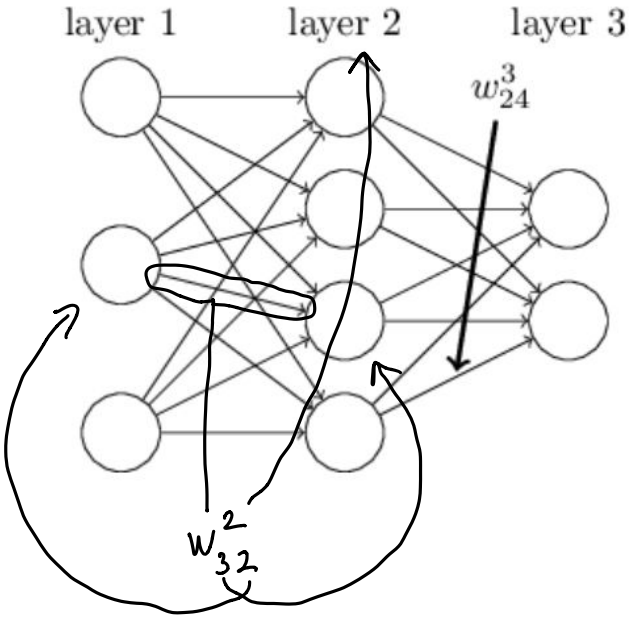
Timestamp 8:42



Training for a single input

1. Define architecture (fully-connected, CNN, RNN) called hyperparameters.
2. Determine number of inputs, hidden layers, and outputs
3. Initialize weights and biases ← random, pretty bad at first
4. FEEDFORWARD through the network to compute loss
5. Determine the output error
6. BACKPROPAGATE the output layer to compute cost function gradient
7. Update weights and biases using the gradient

Notation



read as "the weight connected to j^{th} neuron in l from k^{th} neuron in $l-1$."

w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

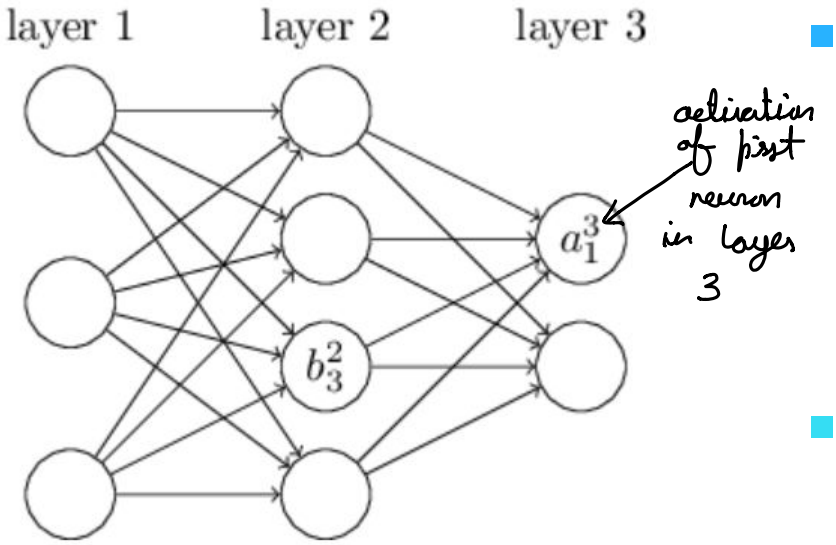
The diagrams we use on these next few slides are from ch. 2 of Michael Nielsen's *Neural Networks and Deep Learning* free (CC licensed) online book. Check it out here!

acm.ai

Notation

$a \rightarrow$ activation

$b \rightarrow$ bias



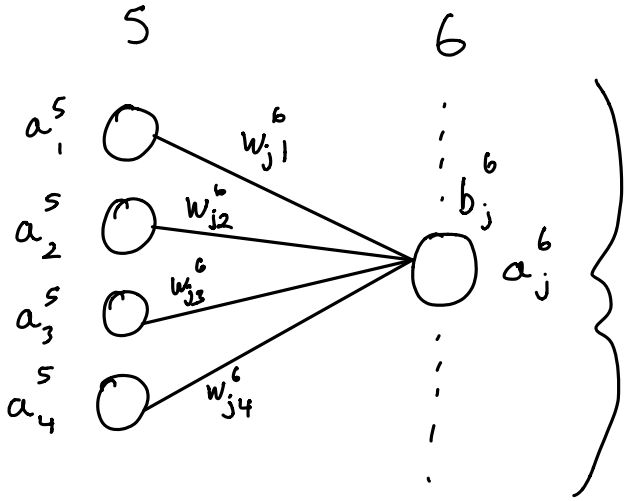
- The superscript refers to the layer being referenced
- The subscript refers to the neuron in the CURRENT layer being referenced

One Layer

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right),$$

- This equation provides the activation (a number ranging from 0-1) of the “j”th neuron in the “l”th layer
- The “l” superscript refers to the layer being referenced
- The “j” subscript refers to the neuron in the CURRENT layer being referenced
- The “k” subscript refers to the neuron in the PREVIOUS layer being referenced

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$



$k : 1 \ldots 4$

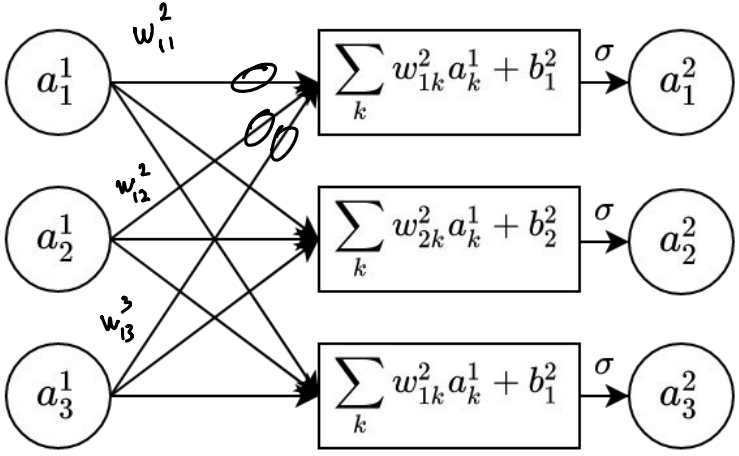
$l : 6$

$j : 1 \ldots n$

One Layer

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right),$$

- This equation provides the activation (a number ranging from 0-1) of the “j”th neuron in the “l”th layer



Matrix Notation

Make everything compact.

$$a^l = \sigma(w^l a^{l-1} + b^l).$$

- w^l is a (j x k) MATRIX of weights
  - Each ROW can be thought of as a (1 x k) VECTOR of weights connecting every neuron in the previous layer to a single neuron in the current layer
- a^{l-1} is a (k x 1) VECTOR of activations that holds the activations of each neuron in the previous layer
- b^l is a (j x 1) VECTOR of biases that holds the biases of each neuron in the current layer
- This equation provides a (j x 1) VECTOR of activations that holds the activation of each neuron in the current layer

$$a^l = \sigma\left(w^l a^{l-1} + b^l\right)$$

$l-1$ has k neurons

l has j neurons

all weights connected to 1st neuron in layer l →

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ w_{21} & w_{22} & \cdots & w_{2j} \\ \vdots & \ddots & \ddots & \vdots \\ w_{k1} & \cdots & \cdots & w_{kj} \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_k^{l-1} \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_j^l \end{bmatrix}$$

↑ all weights from first neuron of layer $l-1$

Matrix Notation

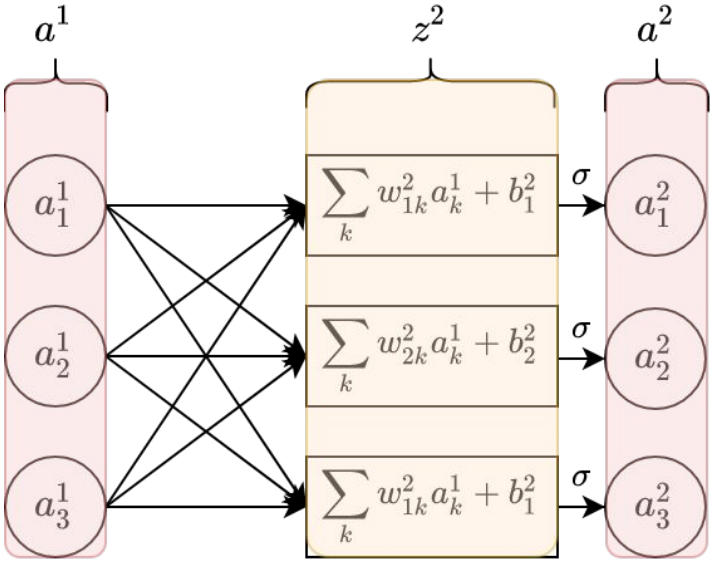
$$z^l \equiv w^l a^{l-1} + b^l \qquad a^l = \sigma(z^l).$$

- The quantity z^l is referred to as the (j x 1) “weighted input vector” for a given layer
  - Each element of the vector holds the weighted input to a single neuron in the layer
- Weighted inputs are used heavily in the math behind backpropagation

Matrix Notation

$$z^l \equiv w^l a^{l-1} + b^l \qquad a^l = \sigma(z^l).$$

- Here's how the matrix notation maps to the equation we showed earlier



Example of the matrices

$$z^l \equiv w^l a^{l-1} + b^l$$

- How many neurons are in layer l?
- How many neurons are in layer l-1?

3

5

$$\begin{pmatrix} z_1^l \\ z_2^l \\ z_3^l \end{pmatrix} = \begin{pmatrix} w_{11}^l & w_{12}^l & w_{13}^l & w_{14}^l & w_{15}^l \\ w_{21}^l & w_{22}^l & w_{23}^l & w_{24}^l & w_{25}^l \\ w_{31}^l & w_{32}^l & w_{33}^l & w_{34}^l & w_{35}^l \end{pmatrix} \begin{pmatrix} a_1^{l-1} \\ a_2^{l-1} \\ a_3^{l-1} \\ a_4^{l-1} \\ a_5^{l-1} \end{pmatrix} + \begin{pmatrix} b_1^l \\ b_2^l \\ b_3^l \end{pmatrix}$$

Feedforward

$$a^l = \sigma(w^l a^{l-1} + b^l).$$

- Compute the result of the above equation for (l = 2, 3, ... L)
  - L refers to the output layer
  - Why do we start at the second layer?
- Ultimately, feeding forward maps our inputs to our outputs
  - This allows us to compute the cost of our network
  - We need the cost in order to perform backpropagation and ultimately train the network

acm.ai

Poll

Backpropagation

Adapted from Nielsen's book: "Neural Networks and Deep Learning"
Reference: http://neuralnetworksanddeeplearning.com/chap2.html
Proofs: https://tinyurl.com/PaymonProof

What is backpropagation?

- "The backward propagation of errors"
  - Error and Cost are two different quantities
- To perform backpropagation, we need to take the partial derivatives of the cost with respect to the weights and biases:

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2, \quad \partial C / \partial w \quad \partial C / \partial b$$

Error

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

- δ^l_j refers to the error of the “j”th neuron at the “l”th layer
- Error is defined in terms of the weighted input (z) as opposed to the activation (a) for algebraic convenience
- Error is calculated as an intermediate step to finding the weight and bias partials
  - This makes sense because the weighted input (z) is a function of the weights and the biases

The Hadamard Product

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \times 3 \\ 2 \times 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Step 1: Compute output error

Component form: $\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).$ Vector form: $\delta^L = \nabla_a C \odot \sigma'(z^L).$

$\nabla_a C$ is a (j x 1) vector where each element holds

$\frac{\partial C}{\partial a_j^L}$ for the "j"th neuron in the output layer "L"

Compute $\frac{\partial C}{\partial a_j^L}$ for our quadratic cost function

$$\partial C / \partial a_j^L = (a_j^L - y_j),$$

Step 1 Visualized

Vector form: $\delta^L = \nabla_a C \odot \sigma'(z^L).$

$$\delta^L = \begin{pmatrix} \frac{\partial C}{\partial a_1^L} \\ \frac{\partial C}{\partial a_2^L} \\ \frac{\partial C}{\partial a_3^L} \\ \frac{\partial C}{\partial a_4^L} \\ \frac{\partial C}{\partial a_5^L} \end{pmatrix} \odot \begin{pmatrix} \sigma'(z_1^L) \\ \sigma'(z_2^L) \\ \sigma'(z_3^L) \\ \sigma'(z_4^L) \\ \sigma'(z_5^L) \end{pmatrix}$$

Step 2: Compute the errors of other layers

Vector form: $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l),$

- Once we have the output layer error vector, we can use this recursive equation to find the layer error for every other layer in the network by calculating backward from the output layer
- We compute up until the first hidden layer (the input layer does not have an error associated with it)

Step 2 Visualized

Vector form : $$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l),$$

Backpropagation from layer $l+1$, with 5 neurons, to layer l with 7 neurons.

$$\delta^l = \begin{pmatrix} w_{1,1}^{l+1} & w_{2,1}^{l+1} & w_{3,1}^{l+1} & w_{4,1}^{l+1} & w_{5,1}^{l+1} \\ w_{1,2}^{l+1} & w_{2,2}^{l+1} & w_{3,2}^{l+1} & w_{4,2}^{l+1} & w_{5,2}^{l+1} \\ w_{1,3}^{l+1} & w_{2,3}^{l+1} & w_{3,3}^{l+1} & w_{4,3}^{l+1} & w_{5,3}^{l+1} \\ w_{1,4}^{l+1} & w_{2,4}^{l+1} & w_{3,4}^{l+1} & w_{4,4}^{l+1} & w_{5,4}^{l+1} \\ w_{1,5}^{l+1} & w_{2,5}^{l+1} & w_{3,5}^{l+1} & w_{4,5}^{l+1} & w_{5,5}^{l+1} \\ w_{1,6}^{l+1} & w_{2,6}^{l+1} & w_{3,6}^{l+1} & w_{4,6}^{l+1} & w_{5,6}^{l+1} \\ w_{1,7}^{l+1} & w_{2,7}^{l+1} & w_{3,7}^{l+1} & w_{4,7}^{l+1} & w_{5,7}^{l+1} \end{pmatrix} \begin{pmatrix} \delta_1^{l+1} \\ \delta_2^{l+1} \\ \delta_3^{l+1} \\ \delta_4^{l+1} \\ \delta_5^{l+1} \end{pmatrix} \odot \begin{pmatrix} \sigma'(z_1^l) \\ \sigma'(z_2^l) \\ \sigma'(z_3^l) \\ \sigma'(z_4^l) \\ \sigma'(z_5^l) \\ \sigma'(z_6^l) \\ \sigma'(z_7^l) \end{pmatrix}$$

Step 3: Compute the weight and bias partials

Component forms: $\frac{\partial C}{\partial b^l_j} = \delta^l_j.$ $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j.$

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Step 4: Update the weights and biases

- The method we will use to update the weights and biases of a network given these weight and bias partials is called Stochastic Gradient Descent (SGD)

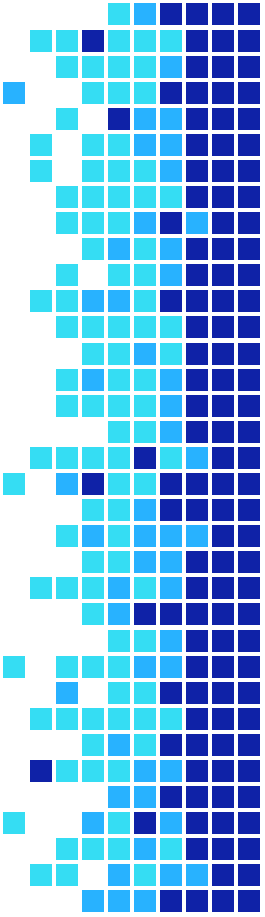
$$w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T, \qquad b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}.$$

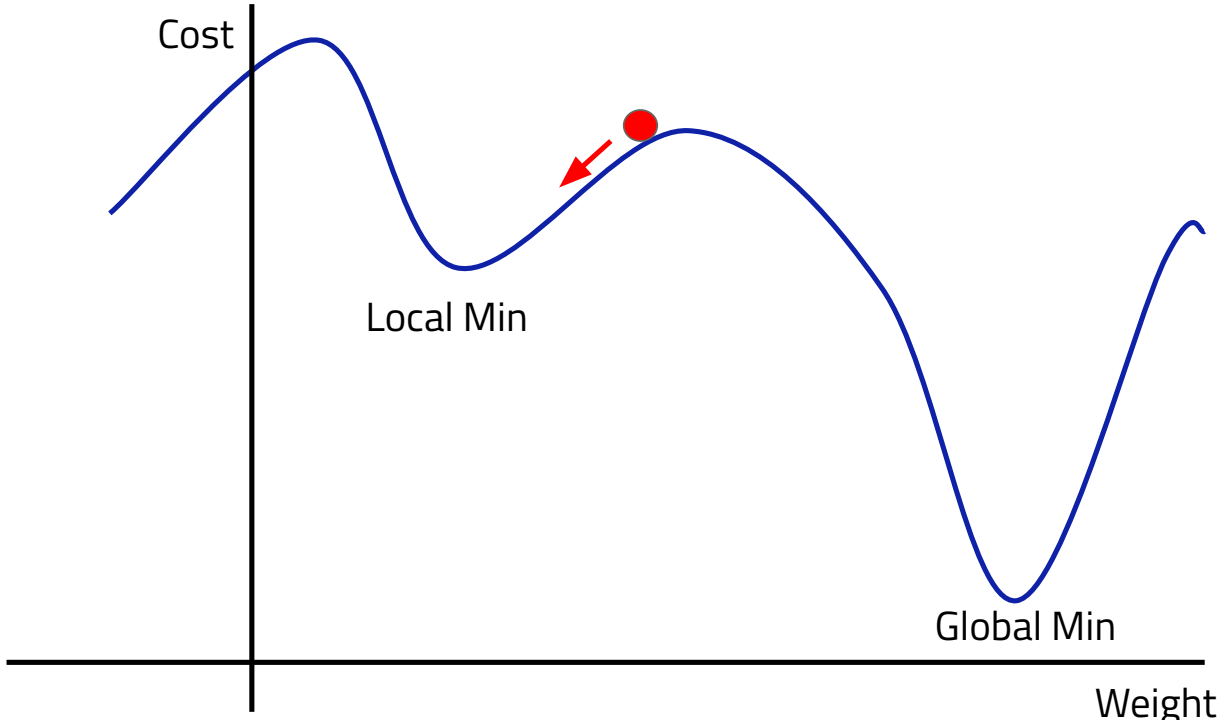
- "η" is the learning rate hyperparameter
- "m" is the mini-batch size hyperparameter
  - A single mini-batch contains "m" training samples
- "x" denotes the index of the training sample within a batch

acm.ai

Poll

Understanding Stochastic Gradient Descent




Cost
Local Min
Global Min
Weight

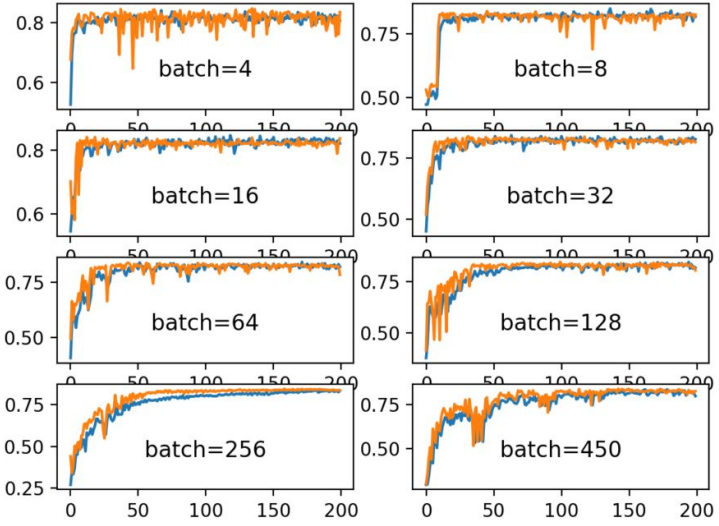
Understanding SGD (hyperparameters)

- To perform SGD, we need to choose network *hyperparameters*
  - Hyperparameters control the training process or the model
  - Hyperparameters are provided by the users, not tuned by the training algorithm
    - In this sense weights and biases are not hyperparameters

- Some examples:
  - The learning rate “η”
  - The mini-batch size “m”
  - The number of hidden layers in the network

Changing Batch Size

- Smaller sizes may learn faster, but be more volatile

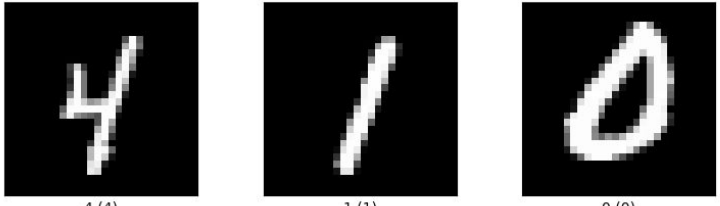


Plots from *How to Control the Stability of Training Neural Networks With the Batch Size*, Machine Learning Mastery by Jason Brownlee

Understanding SGD (dataset structure)

- For supervised learning, the data we use to train our network will be comprised of pairings of inputs with a corresponding expected output

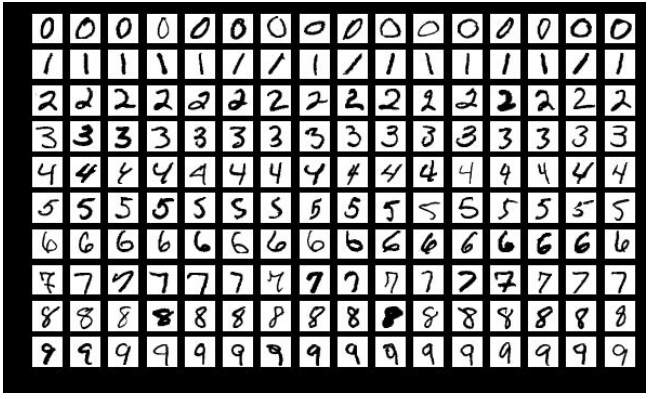
- Example: For the MNIST handwritten digit dataset, a single pairing would consist of a single (28x28) array of pixels representing a handwritten digit and an integer denoting what the handwritten digit is



acm.ai

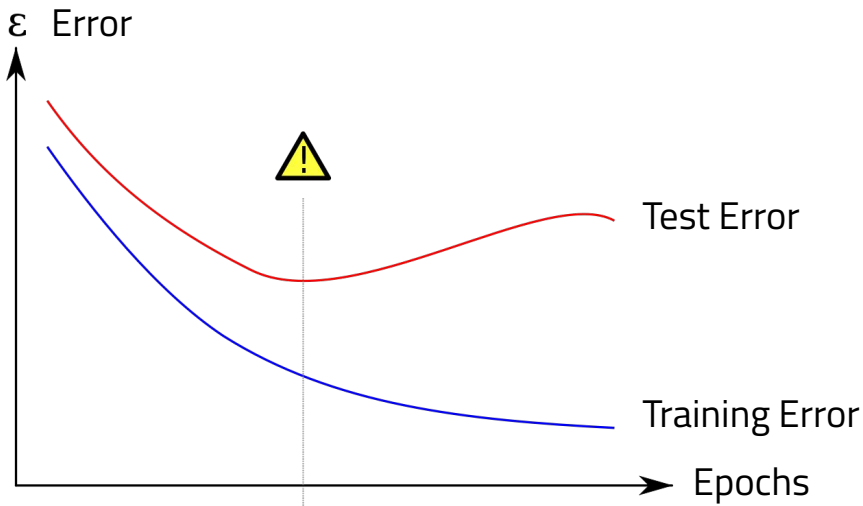
Understanding SGD (dataset structure)

- MNIST example continued
  - This dataset in its totality contains 70,000 of these pairing
  - The first 60,000 pairings make up the “training set”
  - The remaining 10,000 pairings make up the “testing set”



Why a test set?

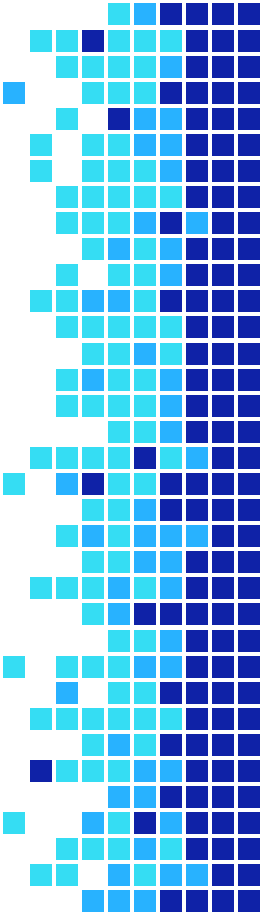
- Our goal is not to memorize the data
- We want a model that can generalize to data it hasn't seen before



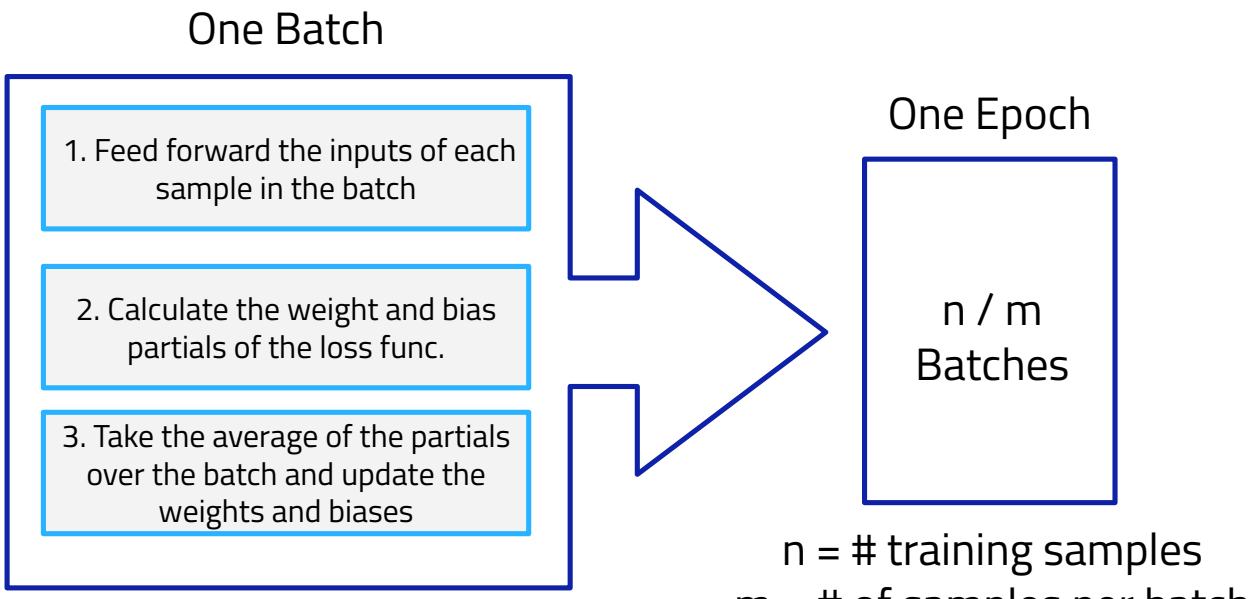
Validation vs. Test Sets

- Sometimes a portion of the training samples is reserved as a “validation set”, which helps us choose hyperparameters for our model

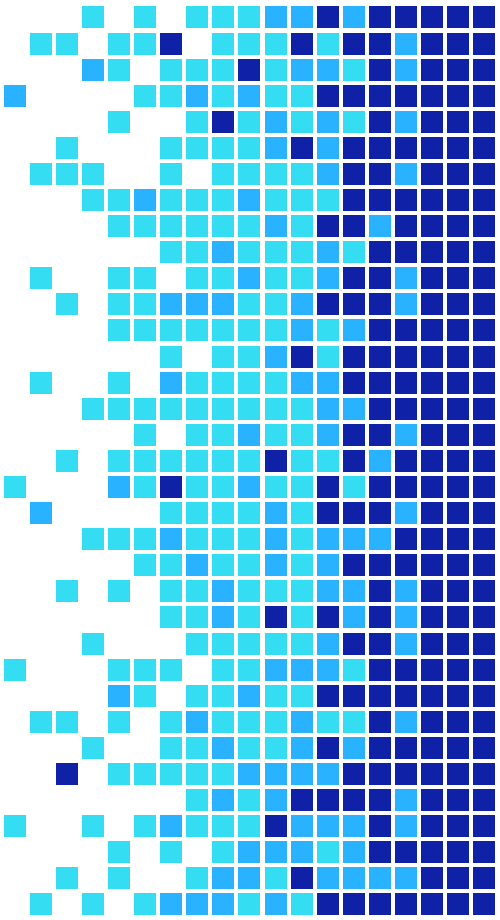
- Why do we want to separate the validation and the test set?



SGD: Overview of One Epoch



Discussion Time!



Some Questions to Consider:

1. Provide two examples of hyperparameters you may have to tweak in your network and the implications that each hyperparameter has on how quickly and efficiently the network can learn.

Some Questions to Consider:

2. Suppose we're performing SGD on our network for training on the MNIST digit classification problem. Our batch size is 10 samples, our training set has 60,000 images, our test set has 10,000 images, and we wish to train over 40 epochs.
   a. How many images are used in each gradient approximation?
   b. How many times is the network going to update its weights and biases?
   c. How many times will the network feedforward?
   d. How many times will the network backpropagate?

Some Questions to Consider:

3. There are 2 models that we've randomly initialized the weights and biases of with different hyperparameters. To train on the MNIST dataset. (Recall MNIST has 60,000 training images and 10,000 testing images.)

Model 1 uses a learning rate of 0.01, a batch size of 40, and trains over 200 epochs.

Model 2 uses a learning rate of 0.1, a batch size of 5, and trains over 1,000 epochs.

Which do you think will achieve better training accuracy? Test accuracy?

Discussion Answers

- https://tinyurl.com/advworksheet2

Conda environment setup

Please read through and complete the steps on this document to make sure you're all set up for the project we'll be doing toward the end of Advanced Track. It should only take ~20 minutes and will teach you valuable foundational knowledge that every computer/data scientist should be familiar with. Please try to do it by the sixth workshop, which is when we'll begin using machine learning tools hands on. We'll also likely hold a short optional session where we'll go through this setup.

Resources

- More on:
  - Activation Functions: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
  - Neural Network Theory:
  - https://www.youtube.com/watch?v=aircAruvnKk&t=820s

Resources (contd.)

- Gradient Descent:
  - https://www.youtube.com/watch?v=IHZwWFHWa-w
- Backpropagation Theory/Math:
  - https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=1s
  - https://www.youtube.com/watch?v=tIeHLnjs5U8

Thank you all for coming!

Anonymous Feedback: forms.gle/QQbKfqEoLnviAooZ9

Facebook Group: www.facebook.com/groups/uclaacmai/