CS/ENGR M148 L14: Backpropagation

Sandra Batista

Administrative News

This week in discussion section:

In lab this week you'll be learning about PyTorch to apply neural networks to your projects.

No project check-ins this week.

Lecture on 11/25/24 will be via Zoom for the holiday.

Midterm grades posted. Grade distribution on piazza. Regrade requests due by 6 pm 11/20/24.

PS3 quiz today!

Will post final project report guidelines this week also with PS4.

Extra credit Final Exam Review Question Code Bank:

https://forms.gle/XdC97wxwWd8QuTR9A

Questions due by 11:59 pm PT on 11/25/24.

We'll share questions with solutions during week 10 for final exam review.

Join our slido for the week…

https://app.sli.do/event/nCV57u4mC7eUMit9euSBr2

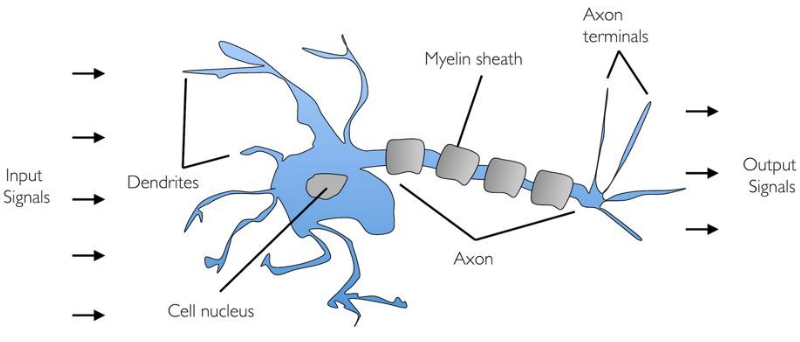
Today's Learning Objectives

Students will be able to:

- Review: Describe what a neural network (NN) is
- Apply the forward algorithm on the MNIST NN
- Trace Backpropagation on a small NN
- Understand the role of gradient descent in training NN

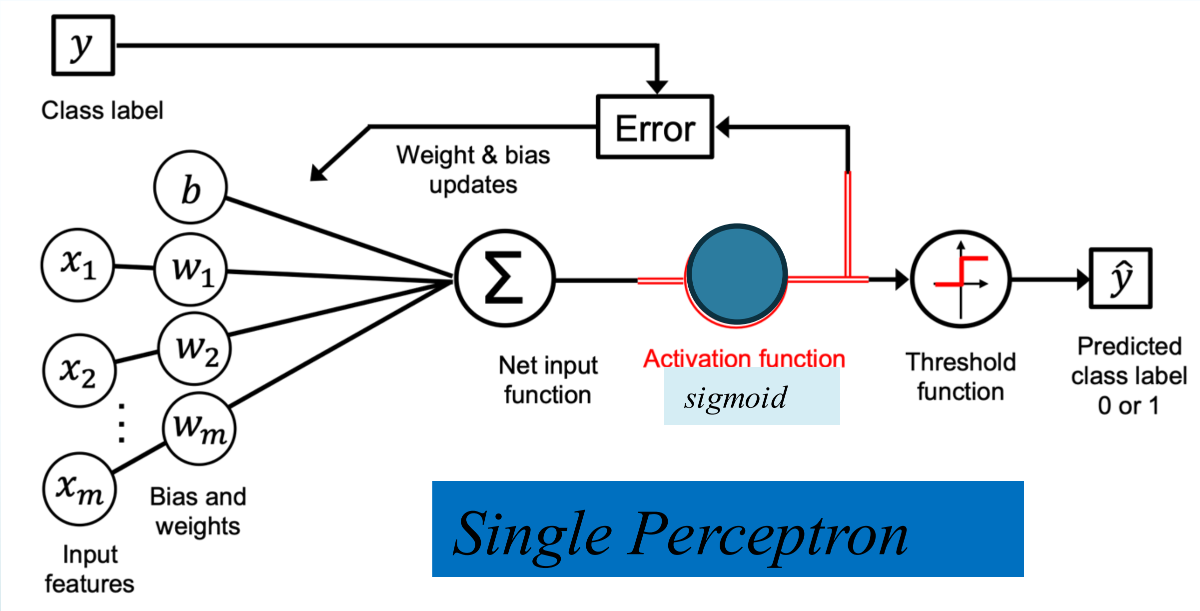
Neural Networks (NN)

- *Based on models for how brain works using artificial neurons*
- *Many varied successful applications such as mood recognition in pictures, modeling virus mutations, and predicting needed medical resources*
- *Used to model complex, nonlinear models*
- *We'll focus on using them for **classification**.*



What is a neural network?

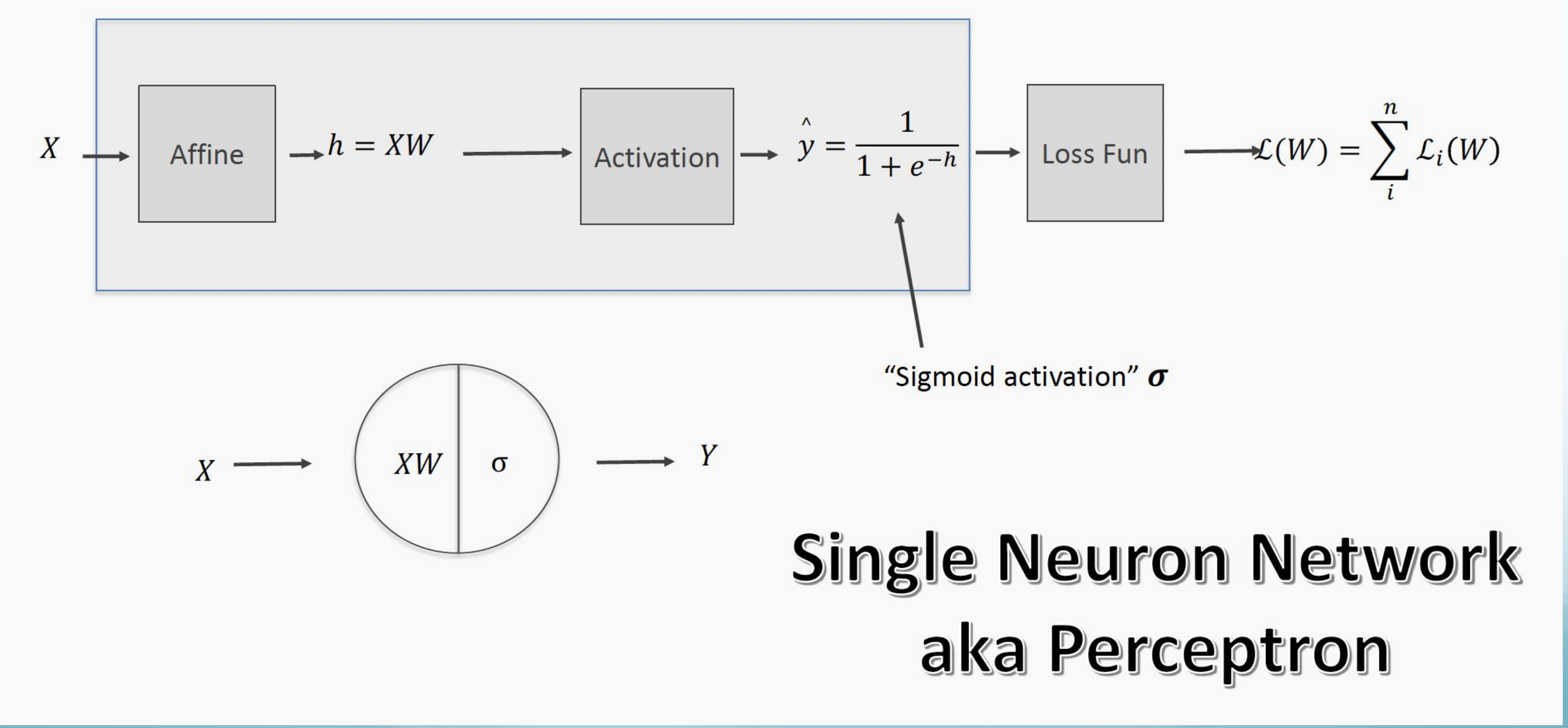
- *A **neural network** consists of layers of nodes or **artificial neurons***



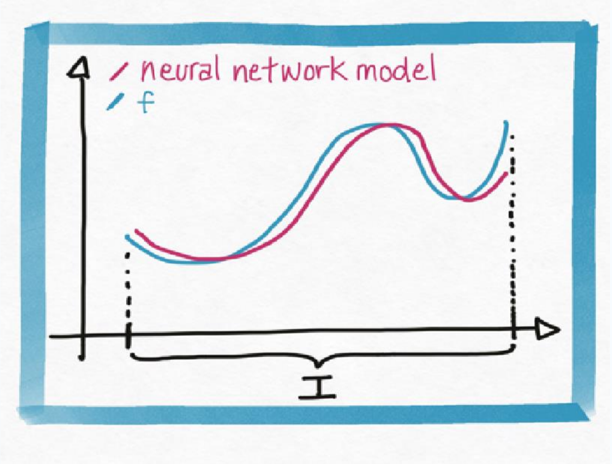
A single neuron can be a logistic regression or linear unit or other activation functions.

What is a neural network?

- *A **neural network** consists of layers of nodes or **artificial neurons***



Neural Networks as Universal Approximators



Theorem:

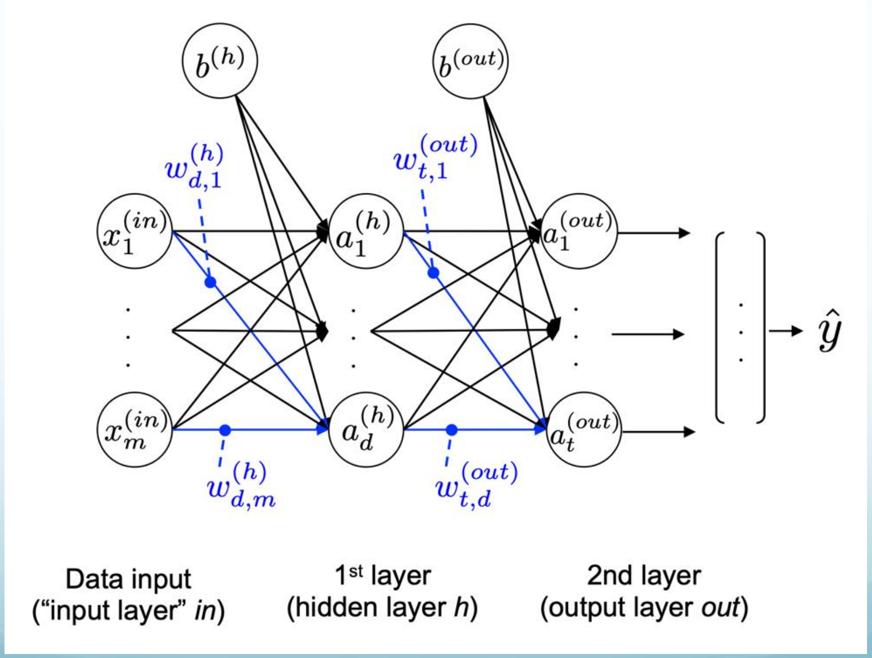
For any continuous function f defined on a bounded domain, we can find a neural network that approximates f with an arbitrary degree of accuracy.

One hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy.

A neural network can approximate non-linear functions either for regression or classification.

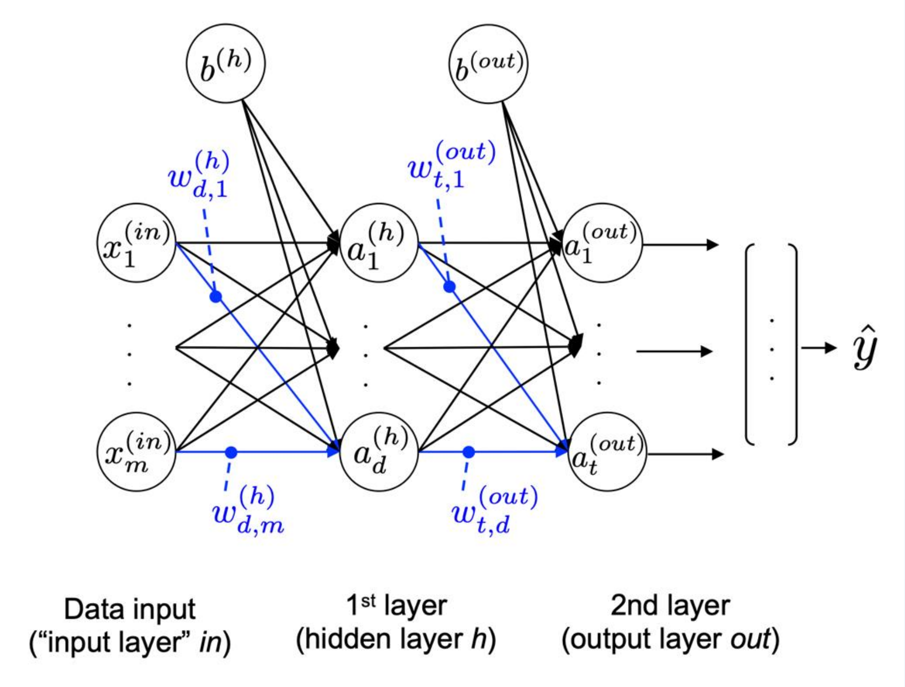
Multilayer Perceptron

- *A neural network is a* combination *of neurons such as logistic regression (or other types) units.*
- *A* ***multilayer perceptron (MLP)*** *is a fully connected network of neurons.*
- *An MLP is a multilayer* ***feedforward*** *NN because each layer is input to the next.*
- *A network with more than one hidden layer is a* ***deep NN.***

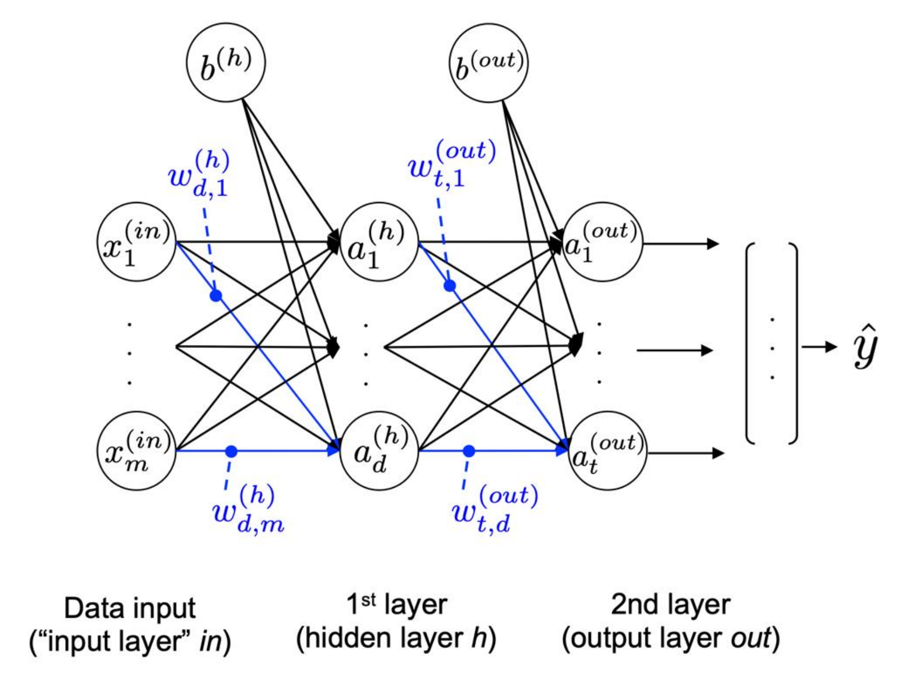


[Raschka et al 2022]

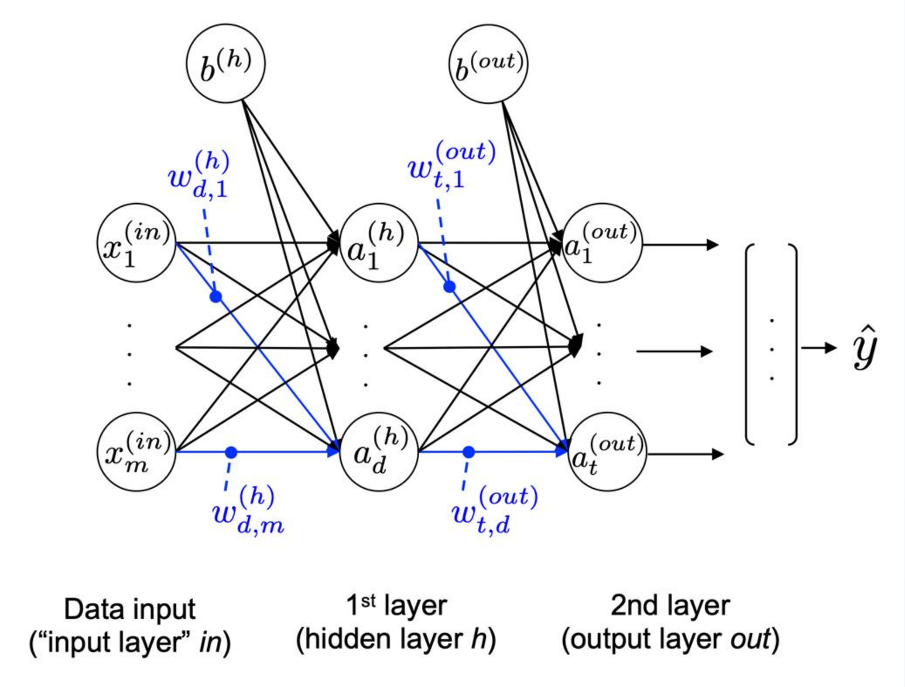
Multilayer Perceptron



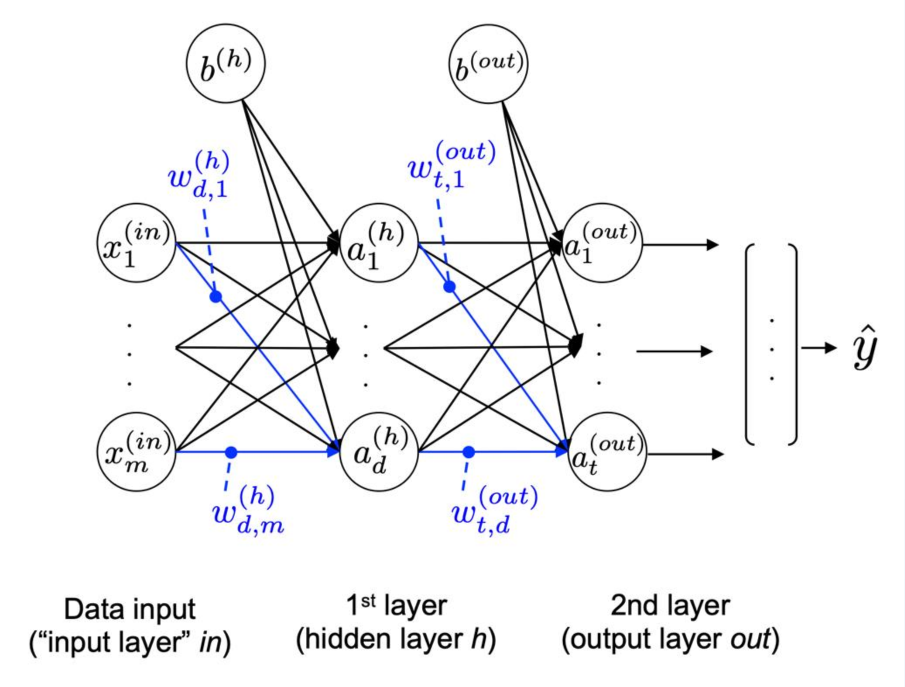
Multilayer Perceptron



Multilayer Perceptron



Composing a complex function



Activation function

$$h = f(W\,{:}\,X + b)$$

The activation function should:

- Provide non-linearity
- Ensure gradients remain large through hidden unit

Examples:

- sigmoid
- ReLU
- identity

Today’s Learning Objectives

Students will be able to:

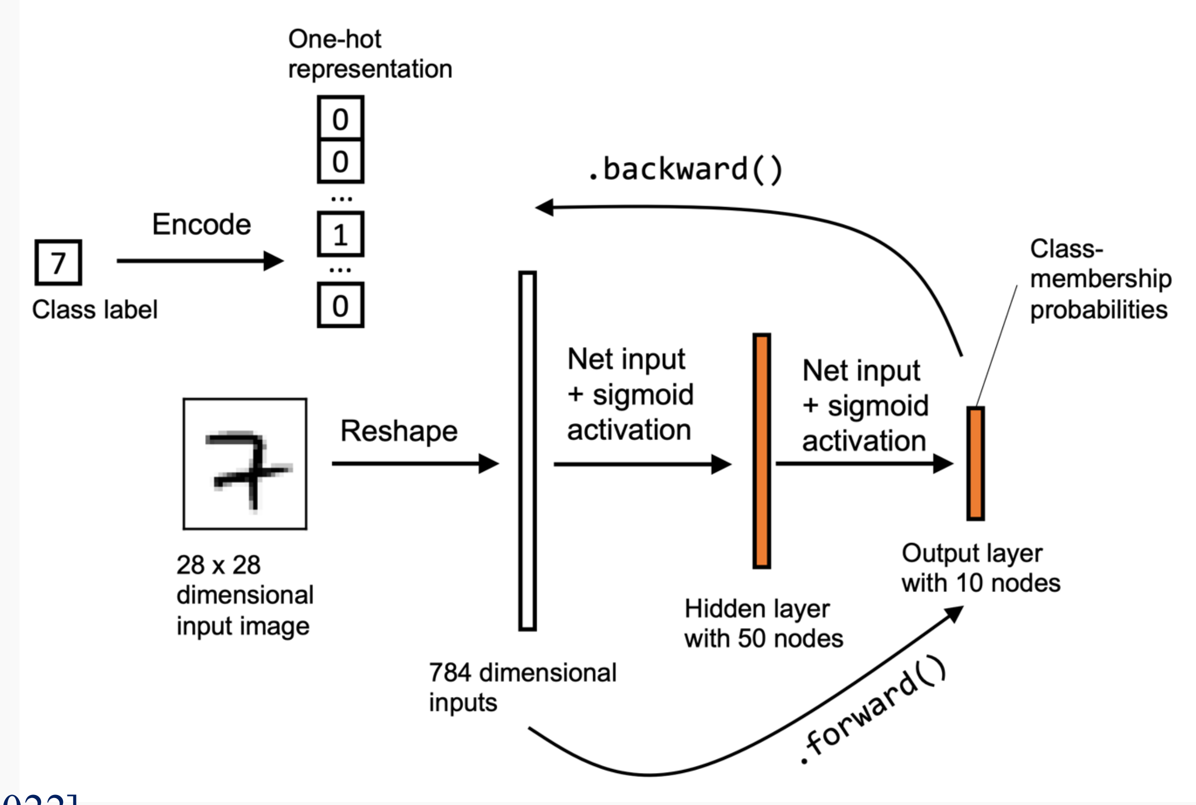
- ✓ Review: Describe what a neural network (NN) is
- Apply the forward algorithm on the MNIST NN
- Trace Backpropagation on a small NN
- Understand the role of gradient descent in training NN

Modified National Institute of Standards and Technology (MNIST) Classification NN from Scratch

Hand-written digit recognition: MNIST data

MNIST NN

Today we'll work on forward algorithm..
Next lecture backpropagation, gradient descent, training, and evaluating network performance



[Raschka et al 2022]

MNIST NN utility code

How would the following sample labels be one-hot encoded: [1, 0,2] ?

```
def sigmoid(z):
    return 1. / (1. + np.exp(-z))


def int_to_onehot(y, num_labels):

    ary = np.zeros((y.shape[0], num_labels))
    for i, val in enumerate(y):
        ary[i, val] = 1

    return ary
```


[Raschka et al 2022]

MNIST NN initialization

Initialize weights and biases

```
class NeuralNetMLP:

    def __init__(self, num_features, num_hidden, num_classes, random_seed=123):
        super().__init__()

        self.num_classes = num_classes

        # hidden
        rng = np.random.RandomState(random_seed)

        self.weight_h = rng.normal(
            loc=0.0, scale=0.1, size=(num_hidden, num_features))
        self.bias_h = np.zeros(num_hidden)

        # output
        self.weight_out = rng.normal(
            loc=0.0, scale=0.1, size=(num_classes, num_hidden))
        self.bias_out = np.zeros(num_classes)
```

[Raschka et al 2022]

Forward Algorithm

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

[Goodfellow et al 2016]

MNIST NN Forward

How do we use the output for prediction?

```
def forward(self, x):
    # Hidden layer
    # input dim: [n_examples, n_features] dot [n_hidden, n_features].T
    # output dim: [n_examples, n_hidden]
    z_h = np.dot(x, self.weight_h.T) + self.bias_h
    a_h = sigmoid(z_h)

    # Output layer
    # input dim: [n_examples, n_hidden] dot [n_classes, n_hidden].T
    # output dim: [n_examples, n_classes]
    z_out = np.dot(a_h, self.weight_out.T) + self.bias_out
    a_out = sigmoid(z_out)
    return a_h, a_out
```


[Raschka et al 2022]

Today's Learning Objectives

Students will be able to:

- ✓ Review: Describe what a neural network (NN) is
- ✓ Apply the forward algorithm on the MNIST NN
- Trace Backpropagation on a small NN
- Understand the role of gradient descent in training NN

Loss Functions

As with other models, we have choice of loss functions, such as

1. MSE

2. Negative Log Likelihood (Binary Cross Entropy) or its generalization for multi-class classification

We'll look at MSE for MNIT

How to calculate MSE for MNIST

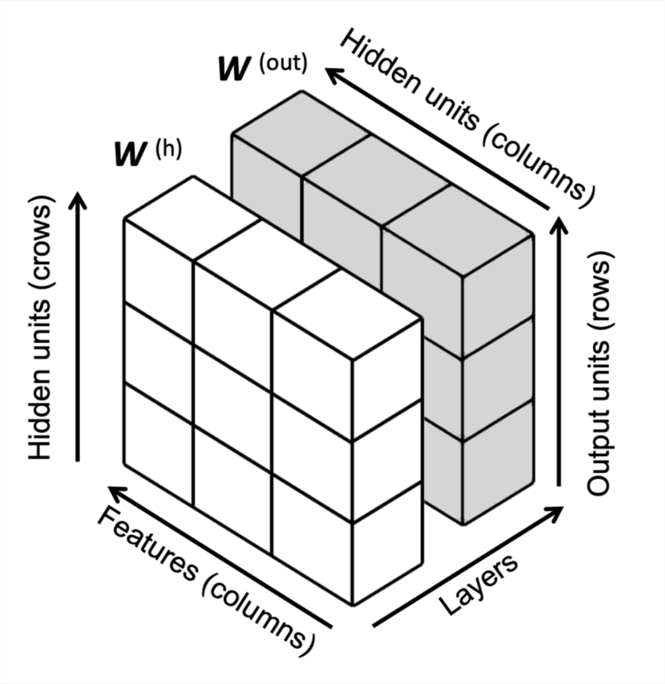
We have a vector of y labels that are one-hot encoded.

If image is 2: [0,0, 1, 0,0,0,0,0,0,0]

Output from NN = [.1,.03,.9, .1, .1,.1,.1,.3,.1,.1]

Minimize Loss

As with any model, we want to minimize our loss function. To do so we need to calculate partial derivative of our loss function with respect to all the weights in the model.



[Raschka et al 2022]

Applying the Chain Rule

$$\frac{d}{dx}[f(g(x))] = \frac{df}{dg} \cdot \frac{dg}{dx}$$

$$\frac{d}{dx}[f(g(h(u(v(x)))))] = \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{du} \cdot \frac{du}{dv} \cdot \frac{dv}{dx}$$

[Raschka et al 2022]

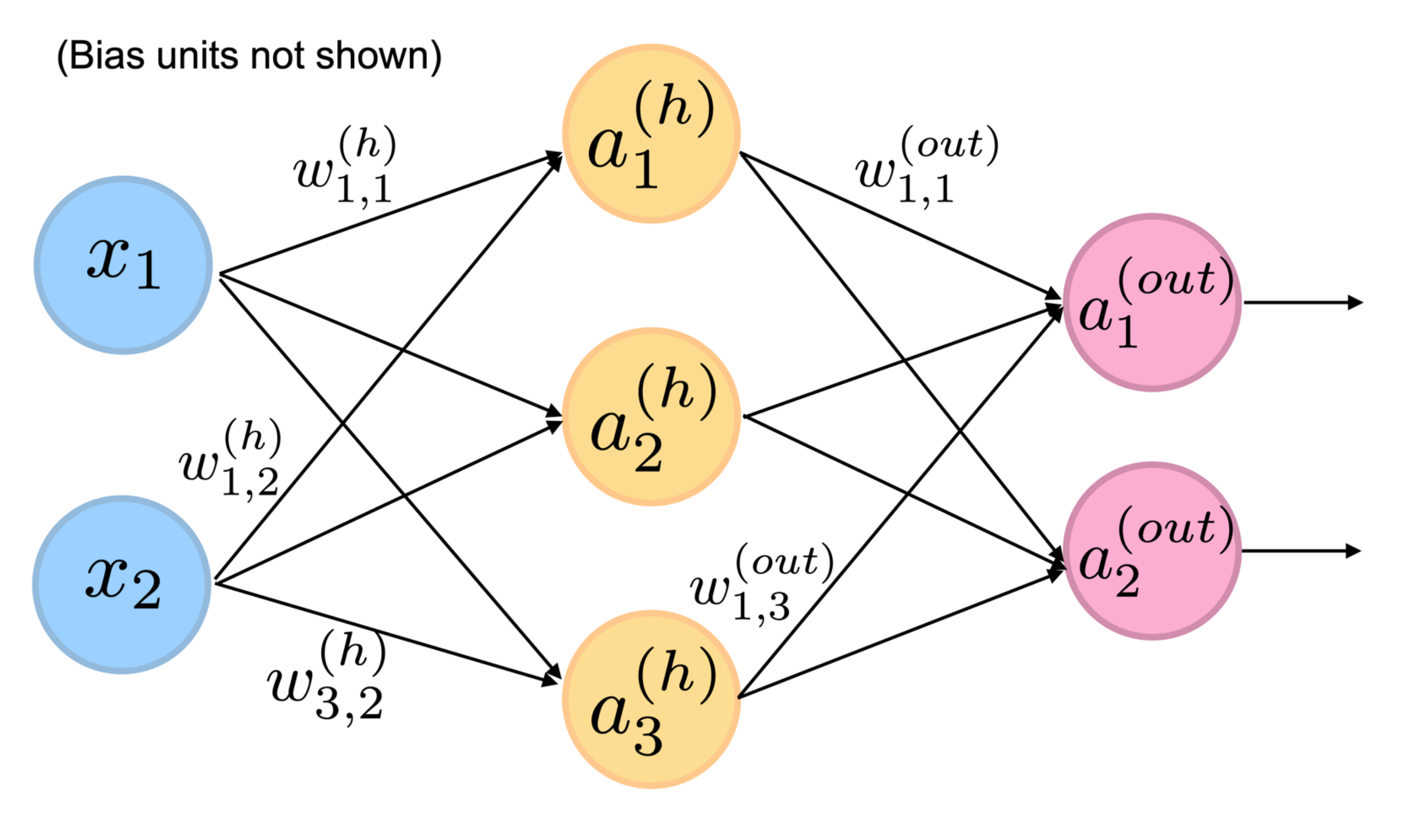
Backprop Algorithm

1. *Compute the gradient of loss function with respect to output layer*
2. *For each layer*
   i. *Convert gradient on layer's output into a gradient with respect to the net input before activation*
   ii. *Compute gradients on weights and biases by using gradient of the net input with respect to weights*
   iii. *Propagate the gradients backwards in NN (i.e. save to reuse for gradients in lower layers)*

Gradients give us how much output layer or weight should change To reduce loss.

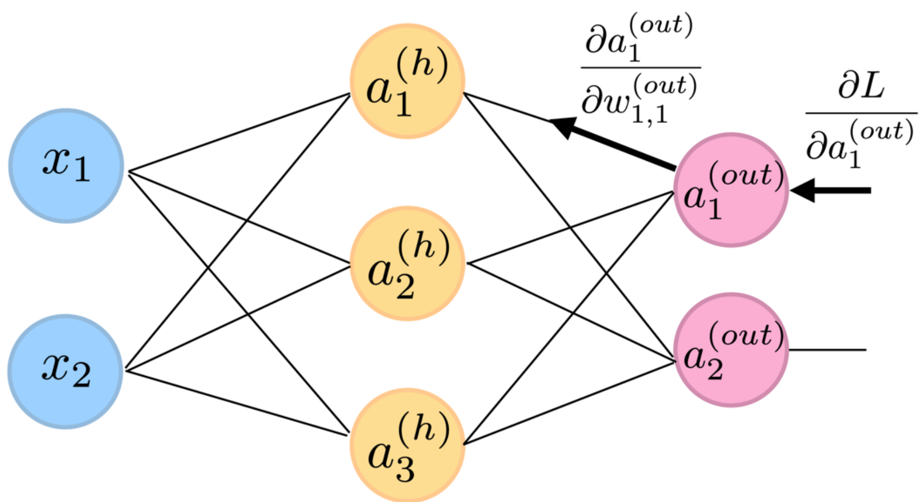
[Goodfellow et al 2016]

Tracing Backprop



[Raschka et al 2022]

Tracing Backprop



Gradient for output layer weight:

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} \cdot \frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

[Raschka et al 2022]

Calculating the partial derivatives

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} \cdot \frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

[Raschka et al 2022]

Calculating the partial derivatives

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} \cdot \frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

[Raschka et al 2022]

Calculating the partial derivatives

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} \cdot \frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

[Raschka et al 2022]

Calculating the partial derivatives

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} \cdot \frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

[Raschka et al 2022]

Calculating the partial derivatives

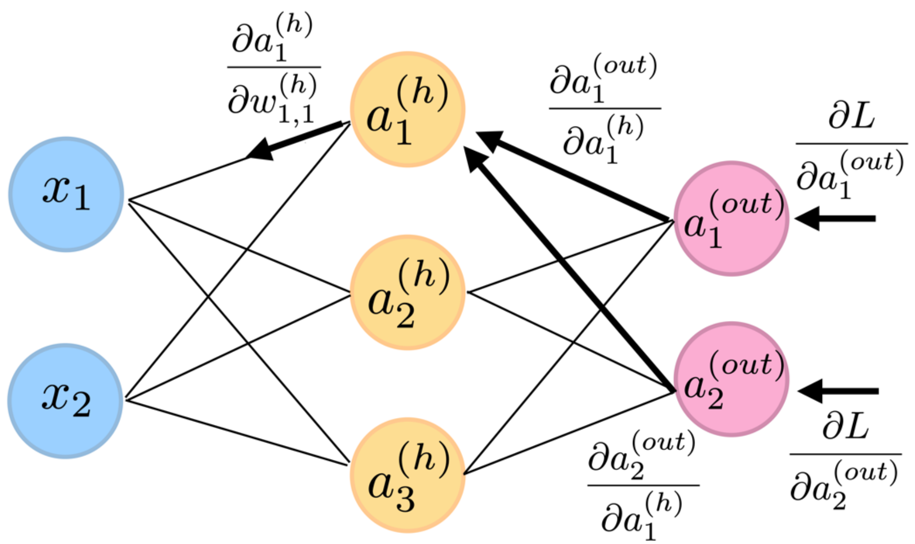
$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} \cdot \frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

Update weights for gradient descent

Reuse gradient of loss with respect to net output

[Raschka et al 2022]

Tracing Backprop

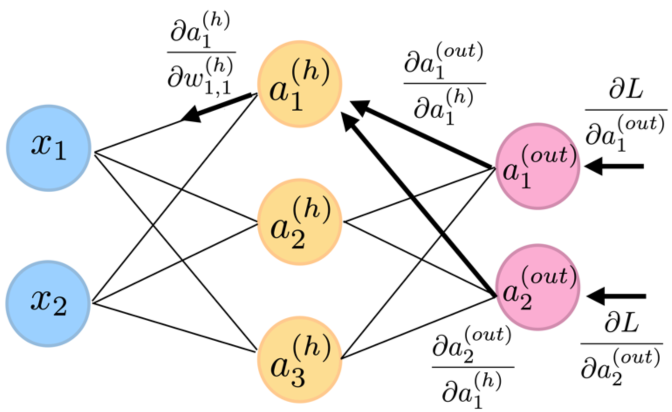


Gradient for hidden layer weight:

$$
\begin{aligned}
\frac{\partial L}{\partial w_{1,1}^{(h)}} &= \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial a_1^{(h)}} \cdot \frac{\partial a_1^{(h)}}{\partial w_{1,1}^{(h)}} \\
&+ \frac{\partial L}{\partial a_2^{(out)}} \cdot \frac{\partial a_2^{(out)}}{\partial a_1^{(h)}} \cdot \frac{\partial a_1^{(h)}}{\partial w_{1,1}^{(h)}}
\end{aligned}
$$

[Raschka et al 2022]

Tracing Backprop



Gradient for hidden layer weight:

$$\frac{\partial L}{\partial w_{1,1}^{(h)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial a_1^{(h)}} \cdot \frac{\partial a_1^{(h)}}{\partial w_{1,1}^{(h)}} + \frac{\partial L}{\partial a_2^{(out)}} \cdot \frac{\partial a_2^{(out)}}{\partial a_1^{(h)}} \cdot \frac{\partial a_1^{(h)}}{\partial w_{1,1}^{(h)}}$$

[Raschka et al 2022]

Today's Learning Objectives

Students will be able to:

- ✔ Review: Describe what a neural network (NN) is
- ✔ Apply the forward algorithm on the MNIST NN
- ✔ Trace Backpropagation on a small NN
- Understand the role of gradient descent in training NN

Training to minimize the loss function

Question: What is the mathematical function that describes the slope?

Derivative

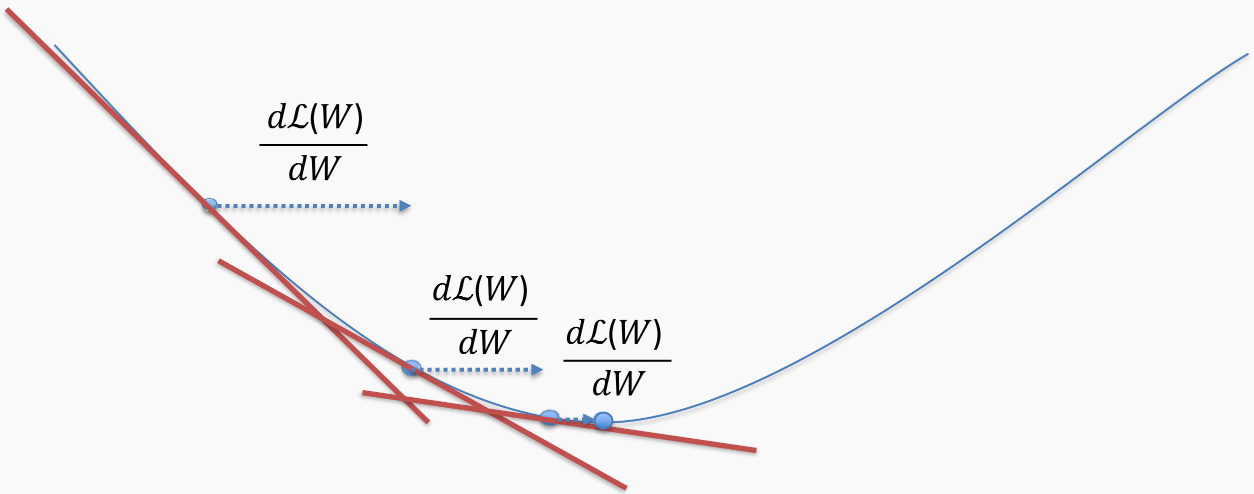
Question: How do we generalize this to more than one predictor?

Take the derivative with respect to each coefficient and do the same sequentially

Question: What do you think is a good approach for telling the model how to change (what is the step size) to become better?

Gradient Descent

If the step is proportional to the slope then you avoid overshooting the minimum. How?



Backprop Algorithm

1. *Compute the gradient of loss function with respect to output layer*
2. *For each layer*
   i. Convert gradient on layer's output into a gradient with respect to the net input before activation
   ii. Compute gradients on weights and biases by using gradient of the net input with respect to weights
   iii. Propagate the gradients backwards in NN (i.e. save to reuse for gradients in lower layers)

Gradients give us how much output layer or weight should change To reduce loss.

[Goodfellow et al 2016]

Gradient Descent

1. *Initialize small weights*
2. *For each sample*
   i. Apply the forward algorithm
   ii. Apply backprop
   iii. Use the gradients of the weights and biases to update the weights and biases

Repeat step 2 for a number of ***epochs,*** *complete pass through Training data*

[Goodfellow et al 2016]

Updating the weights

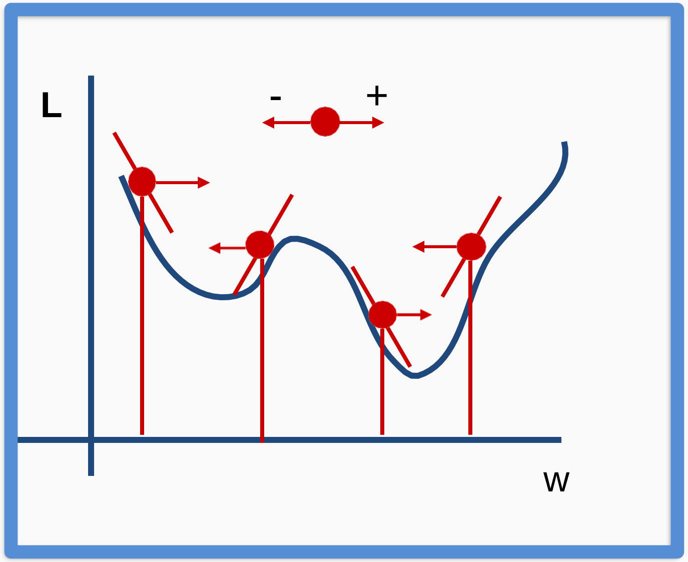
For each weight

$$\Delta w = \frac{\partial L}{\partial w}$$

$$w^{new} = w^{old} - \eta \Delta w$$

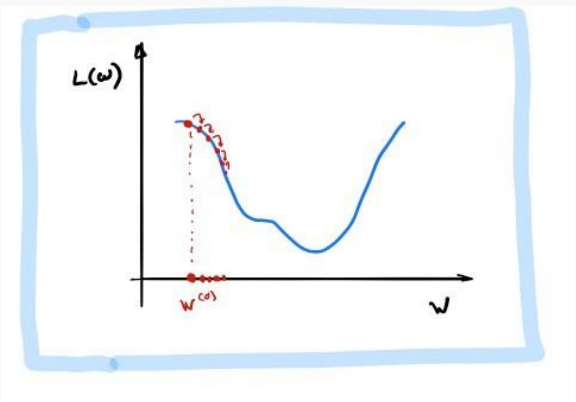
Gradient Descent

- Algorithm for optimization of first order to finding a minimum of a function.
- It is an iterative method.
- L is decreasing much faster in the direction of the negative derivative.
- The learning rate is controlled by the magnitude of η.

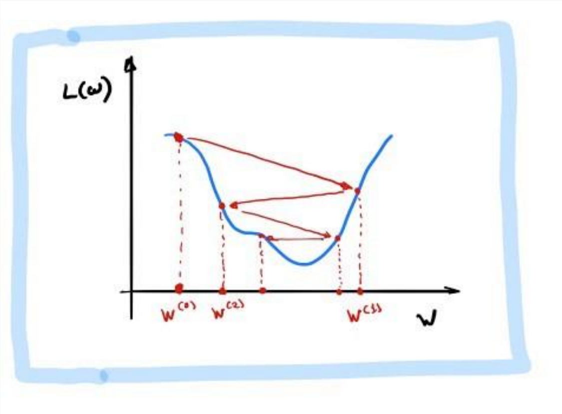


Learning Rate

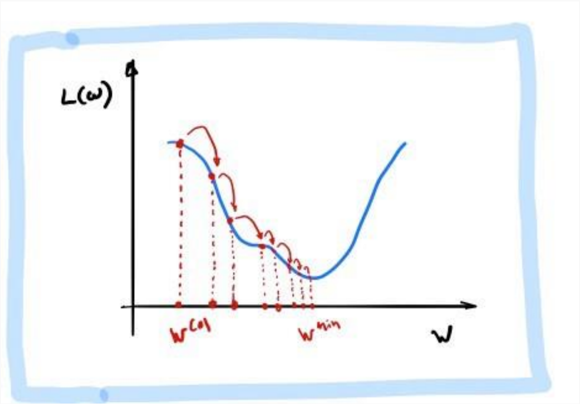
Our choice of the learning rate has a significant impact on the performance of gradient descent.



When η is too small, the algorithm makes very little progress.



When η is too large, the algorithm may overshoot the minimum and has crazy oscillations.

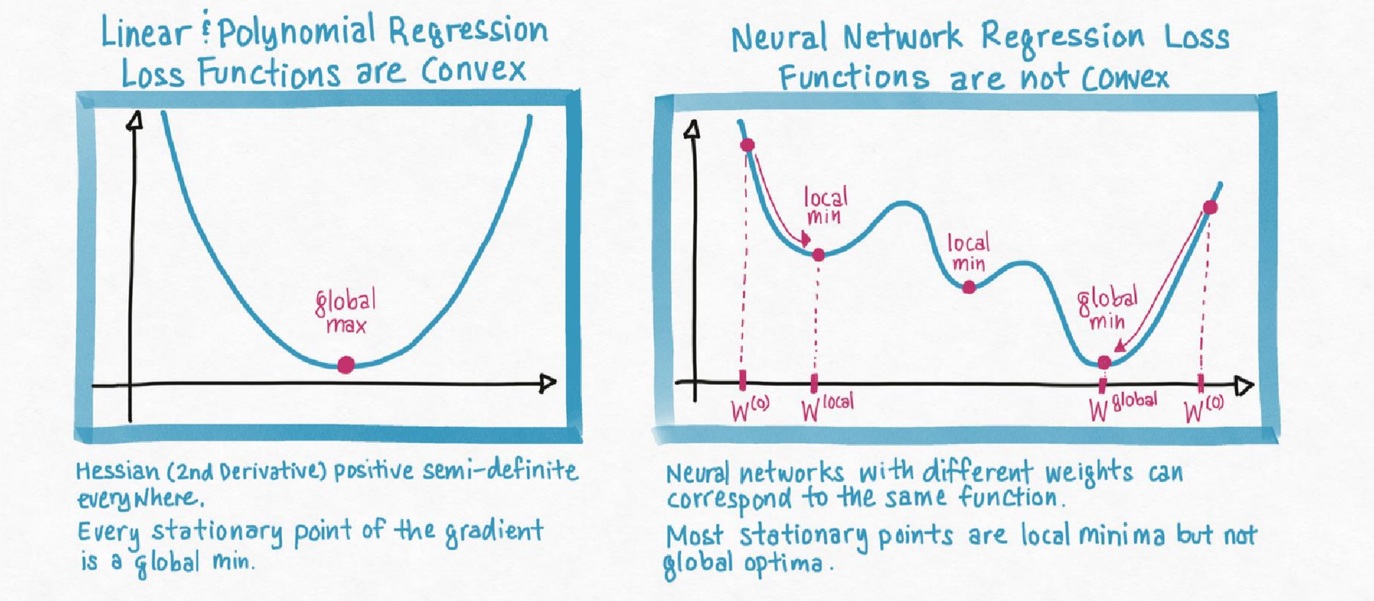


When η is appropriate, the algorithm will find the minimum. The algorithm **converges**!

Local vs Global Minima

If we choose η correctly, then gradient descent will converge to a stationary point. But will this point be a **global minimum**?

If the function is convex then the stationary point will be a global minimum.



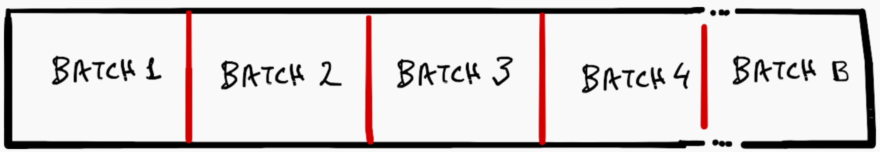
Batch and Stochastic Gradient Descent

Instead of using all the training examples for every step,

For **batch gradient descent,**
In each epoch use a subset or **batch** of samples. (These can be randomly selected). Use average of the gradients from the batches to update the weights

For **stochastic gradient descent,**
Use a randomly selected sample to update the weights a sample at a time. (Like batch, but here size = 1)

DATA



calculate $L_1 \Rightarrow \frac{\partial L_1}{\partial w} \Rightarrow w^* = w - \eta \frac{\partial L_1}{\partial w}$

calculate $L_2 \Rightarrow \frac{\partial L_2}{\partial w} \Rightarrow w^* = w - \eta \frac{\partial L_2}{\partial w}$

$L_B \Rightarrow \frac{\partial L_B}{\partial w} \Rightarrow w^* = w - \eta \frac{\partial L_B}{\partial w}$

COMPLETE DATA $\Rightarrow$ ONE EPOCH

RESHUFFLE DATA AND REPEAT

Your turn:
MNIST NN

Please get the Jupyter notebook for GPU data:

Go to Notebook (also on BruinLearn):
https://colab.research.google.com/drive/10BkOu_bVBs3af2NXFrD48iK8s3MJIScj?usp=sharing

Save a copy to your Google Drive and keep notes there...

Today's Learning Objectives

Students will be able to:

- ✓ Review: Describe what a neural network (NN) is
- ✓ Apply the forward algorithm on the MNIST NN
- ✓ Trace Backpropagation on a small NN
- ✓ Understand the role of gradient descent in training NN

Citations:.

Goodfellow, I., Bengio, Y.,, Courville, A. (2016). *Deep Learning*. MIT Press.

Sebastian **Raschka**, Yuxi (Hayden) **Liu**, and Vahid Mirjalili. **Machine Learning** with PyTorch and Scikit-Learn. Packt Publishing, **2022**.

Baharan Mirzasoleiman, UCLA CS M148 Winter 2024 Lecture 13 Notes

Thank You