COMP 5630/6630:Machine Learning

Lecture 7: Neural Network, Backpropagation, MLP Demo

Slide Adapted: Dr. Aakur, : Kian Katanforoosh

Introduction to Neural Networks

Introduction to Neural Network

- The Perceptron algorithm for binary classification
  - How perceptron is different from logistic regression (LR)
  - Training steps a perceptron
  - Extending LR to multiclass LR
    - Extending perceptron to multilayer perceptron
- Forward Pass
  - Defining propagation equations
  - Defining loss function
- How to train
- Backward propagation

Introduction to Neural Network

- The Perceptron algorithm for binary classification
  - How perceptron is different from logistic regression (LR)
  - Training steps a perceptron
  - Extending LR to multiclass LR
    - Extending perceptron to multilayer perceptron
  - Forward Pass
    - Defining propagation equations
    - Defining loss function
  - How to train
  - Backward propagation

The Perceptron: Supervised Binary Classifier

- Task: Predict whether an input image contains a cat (1) or not cat (0)
- Consider modifying the logistic regression method to "force" it to output values that are either 0 or 1 or exactly.
  - Change the definition of g to be the threshold function of logistic regression:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- Let $h(x) = g(\theta^T x)$ as before but using this modified definition of g, and if we use the update rule

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

The Perceptron Algorithm: Difference w.r.t LR

- In the 1960s, this “perceptron” was argued to be a rough model for how individual neurons in the brain work.
- Although the perceptron may be similar to the other algorithms we have seen, it is actually a very different type of algorithm than logistic regression.
  - The hypothesis in logistic regression provides a measure of uncertainty in the occurrence of a binary outcome based on a linear model.
  - The output from a *step function* can of course not be interpreted as any kind of probability.
  - Since a step function is *not differentiable*, it is not possible to train a perceptron using the same algorithms that are used for logistic regression.

Introduction to Neural Network

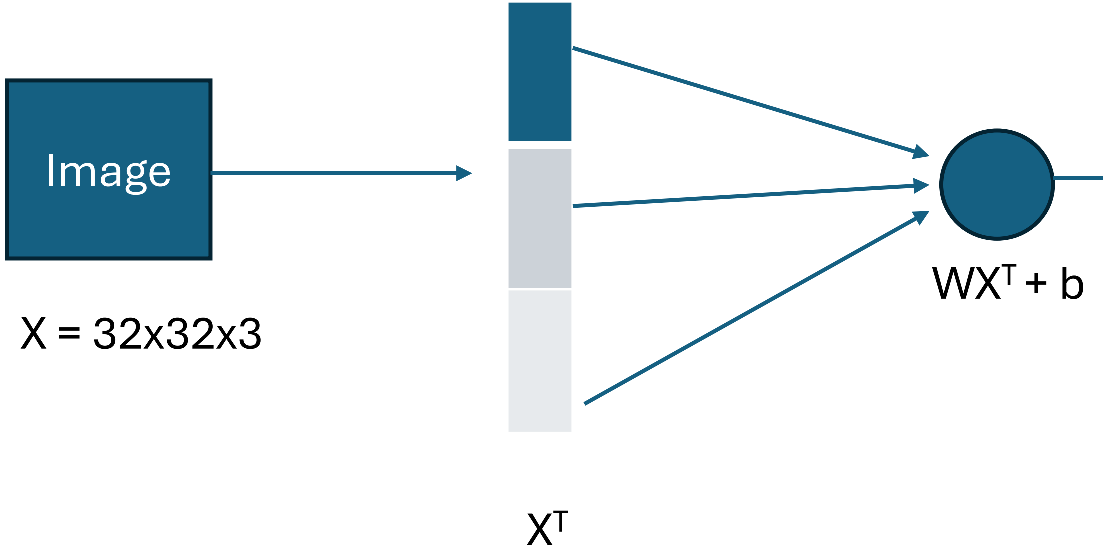
- The Perceptron algorithm for binary classification
  - How perceptron is different from logistic regression (LR)
  - Training steps a perceptron
  - Extending LR to multiclass LR
    - Extending perceptron to multilayer perceptron

  - Forward Pass
    - Defining propagation equations
    - Defining loss function

  - How to train
  - Backward propagation

The Perceptron Algorithm: Training Idea

- Task: Predict whether an input image contains a cat (1) or not (0)
- Image representation from Lecture 2: Linear Classifier



$y^{\wedge}$ =sigmoid ($WX^T + b$)

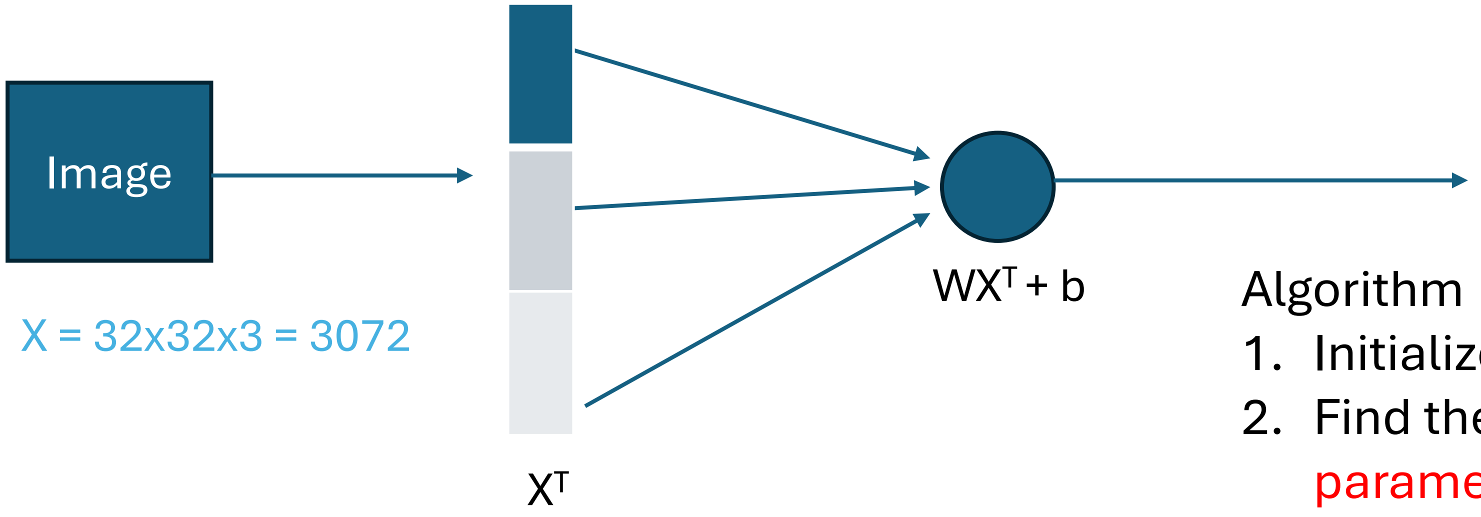
Algorithm

1. Initialize W and b
2. Find the optimal value of W and b (learning parameters)
3. Predict $y^{\wedge}$

How many parameters are in this model?

The Perceptron Algorithm

- Task: Predict whether an input image contains a cat (1) or not cat (0)
- Image representation from Lecture 2: Linear Classifier



$\hat{y}$ =sigmoid (WX^T + b)

Algorithm
1. Initialize W and b
2. Find the optimal value of W and b (learn parameters)
3. Predict $\hat{y}$

How many parameters are in this model?

W = 1x 3072, X^T = 3072x 1, b = 1

Parameters = 3072 + 1

Vocabulary of Neural Network

- Neuron = linear + activation
  - In our example
    - Linear= $WX^T + b$
    - Activation= sigmoid on the linear output

- Model: architecture + parameter
  - In our example
    - Model = logistic regression
    - Parameter = 3073

Vocabulary of Neural Network: Activation Functions

- Neuron = linear + activation (non-linear)
- The z_j is a linear component, corresponds to outputs of inputs from previous layer
  - or input layer of network
- Each activation a_j transforms z_j using differentiable nonlinear activation functions
- Three examples of activation functions:

1. Logistic sigmoid

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

2. Hyperbolic tangent

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

3. Rectified Linear unit

$$f(x) = \max(0, x)$$

- Some functions: step, softplus $\log(1 + e^x))$, leak RELU, softmax

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Activation Functions: Examples

- Task: Predict the age of the animal of the image
  - *What changes will you make in the previous network and why?*

Activation Functions: Examples

- Task: Predict the age of the animal of the image

  - What changes will you make in the previous network?

  - Sigmoid activation function will not be a valid choice
    - ReLU: the age is non-negative
    - The identity or linear activation function

Introduction to Neural Network

- The Perceptron algorithm for binary classification
  - How perceptron is different from logistic regression (LR)
  - Training steps a perceptron
  - Extending LR to multiclass LR
    - Extending perceptron to multilayer perceptron

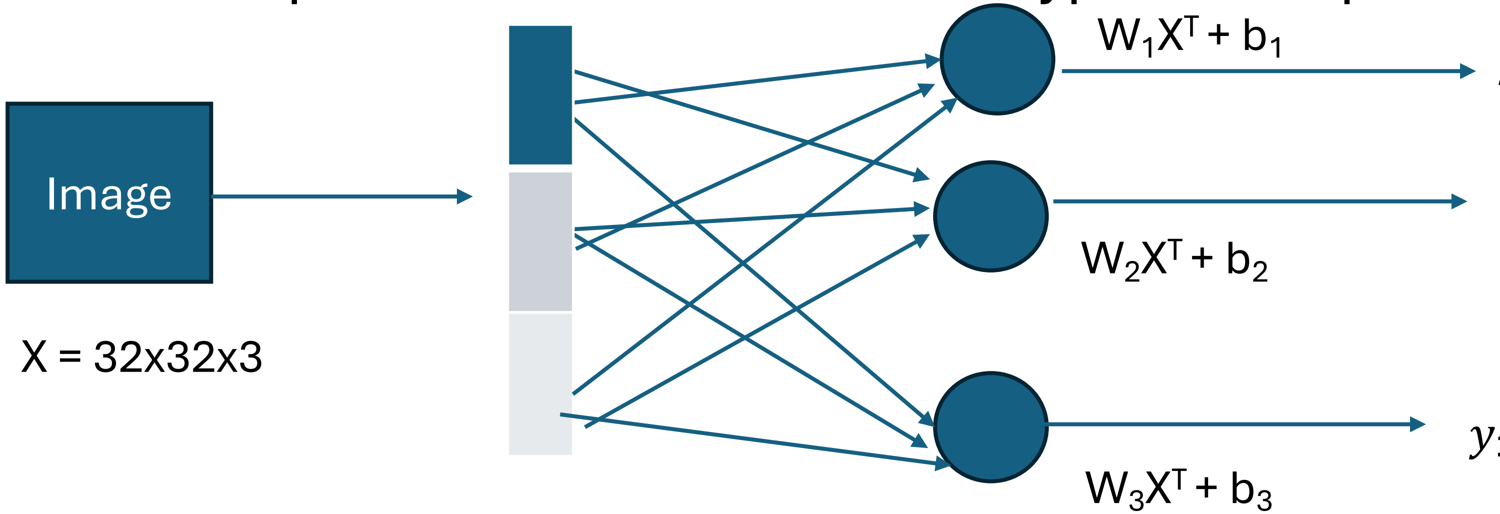
  - Forward Pass
    - Defining propagation equations
    - Defining loss function

  - How to train
  - Backward propagation

Multiclass Logistic Regression & Neural Network

Task: Predict whether an input image contains a cat/sheep/dog. *Image may contain multiple types of animals.*

Extend the previous network for three types of outputs



$\hat{y_1}$ =sigmoid ($W_1X^T + b_1$)

$\hat{y_2}$ =sigmoid ($W_2X^T + b_2$)

$\hat{y_3}$ =sigmoid ($W_3X^T + b_3$)

X = 32x32x3

X^T

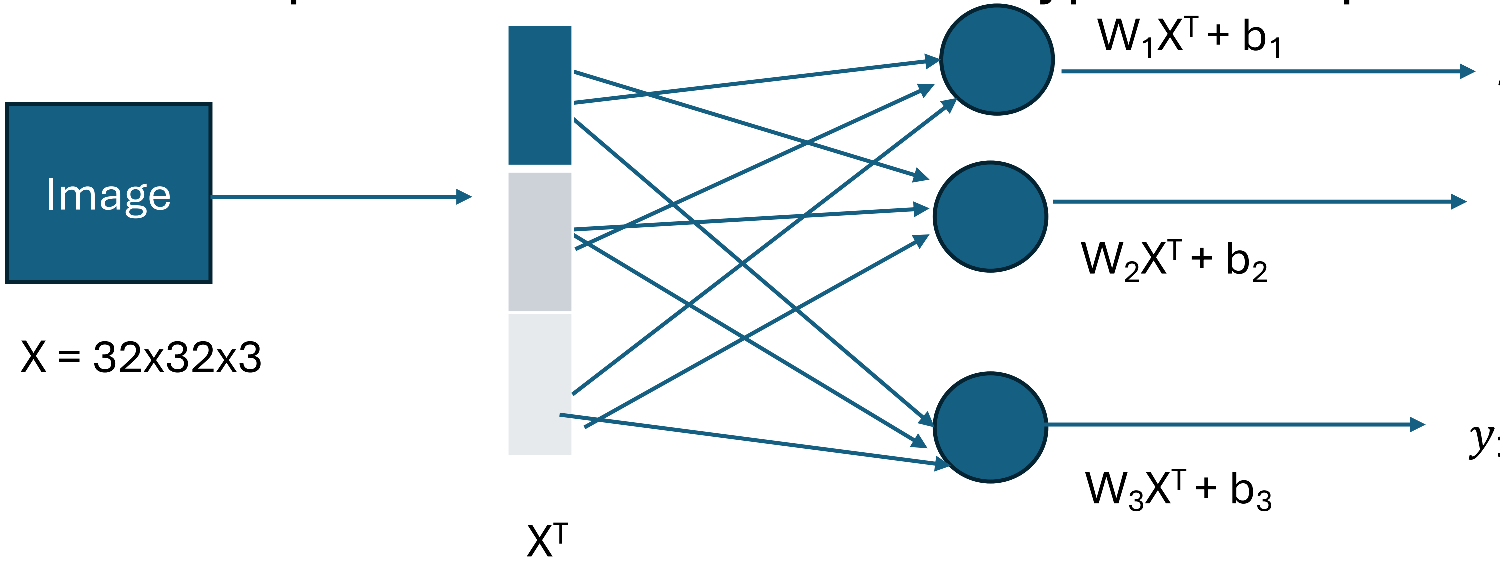
Assignment 1 question answer hint

How many parameters are in this model?
Can this network classify if an image contains BOTH cat and dog?

Multiclass Logistic Regression & Neural Network

Task: Predict whether an input image contains a cat/sheep/dog. *Image may contain multiple types of animals.*

Extend the previous network for three types of outputs



$y_1^{\wedge}$ =sigmoid ($W_1X^T + b_1$)

$y_2^{\wedge}$ =sigmoid ($W_2X^T + b_2$)

$y_3^{\wedge}$ =sigmoid ($W_3X^T + b_3$)

How many parameters are in this model? = 3 x prev. model
Can this network classify if an image contains BOTH cat and dog? YES. As each neuron is independent

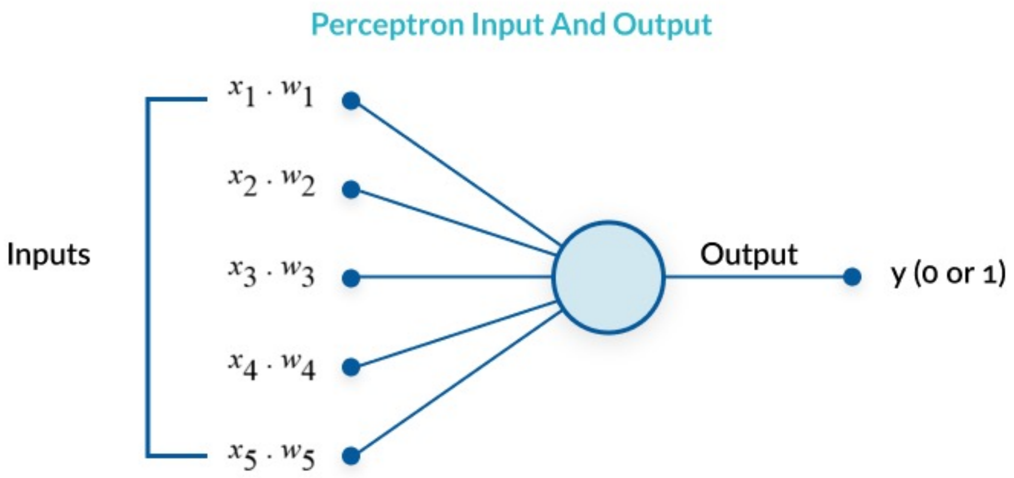
Assignment 1 question answer hint

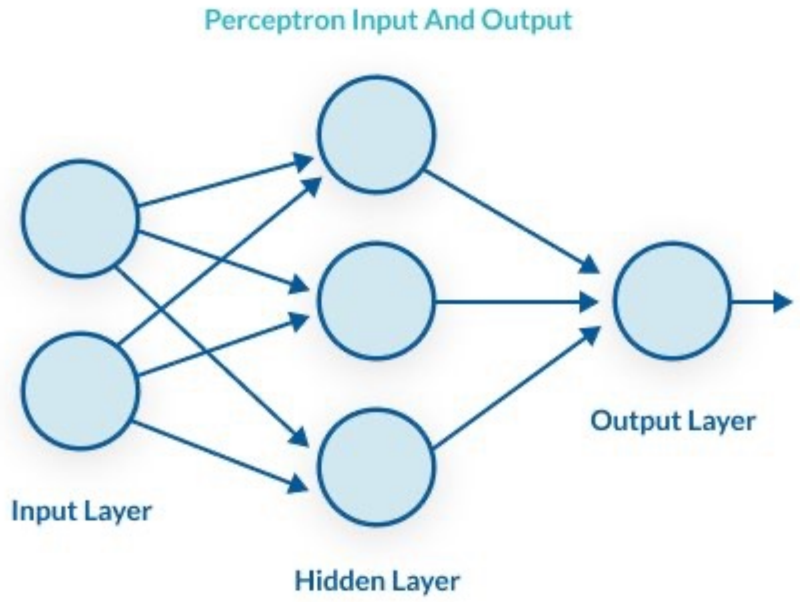
Activation Function Selection

- Determined by the nature of the data and the assumed distribution of the target variables
- For standard regression problems the activation function is the identity function so that $y_k=a_k$
  - *(e.g. predicting the age of the animal in an input image)*
- For multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that $y_k=\sigma(a_k)$
  - *(e.g. predicting whether an input image contains a cat/sheep/dog. Image may contain multiple types of animals)*
- For multiclass problems, a softmax activation function of the form:

$$\boxed{\frac{\exp(a_k)}{\sum_j \exp(a_j)}}$$

Perceptron vs Multilayer Perceptron (MLP)

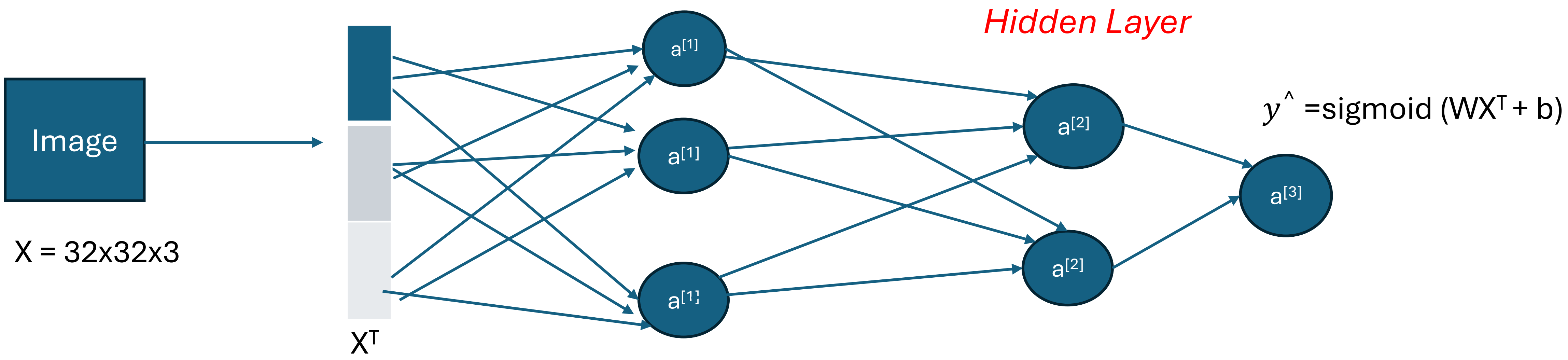




Why do we need more layers?

Multilayer Perceptron

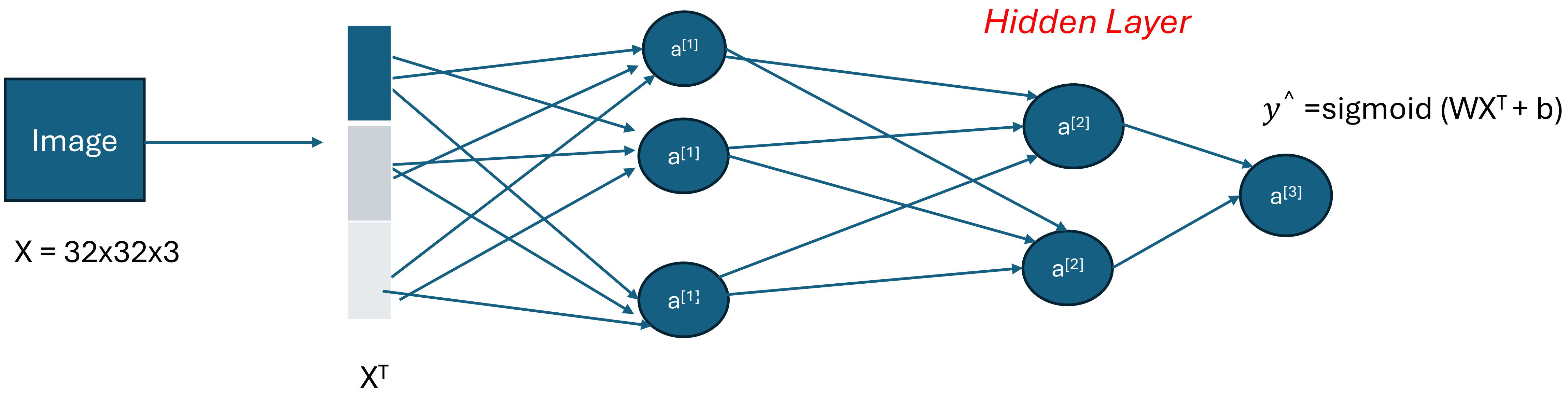
Task: Predict whether an input image contains a cat or not



- The first layer processes the input.
- The two neuron in the hidden layer may identify two different features of a cat, such as eye color or size of the head. We don't know how the layers operate (thus called hidden layer).
- Assumption: given enough data and adding more layers, the network can accurately identify the image

Multilayer Perceptron

Task: Predict whether an input image contains a cat or not

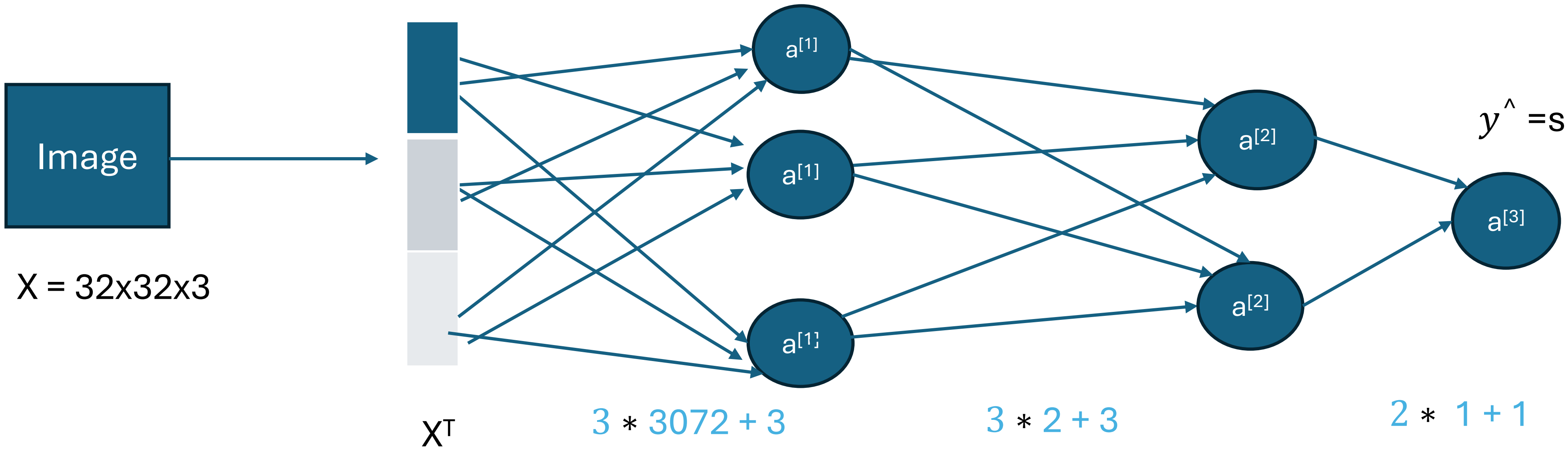


How many parameters are in this model?

Multilayer Perceptron

Task: Predict whether an input image contains a cat or not

$\hat{y}$ =sigmoid ($WX^T + b$)



$3 * 3072 + 3$ $3 * 2 + 3$ $2 * 1 + 1$

How many parameters are in this model?

Introduction to Neural Network

- The Perceptron algorithm for binary classification
  - How perceptron is different from logistic regression (LR)
  - Training steps a perceptron
  - Extending LR to multiclass LR
    - Extending perceptron to multilayer perceptron

  - Forward Pass
    - Defining propagation equations
    - Defining loss function

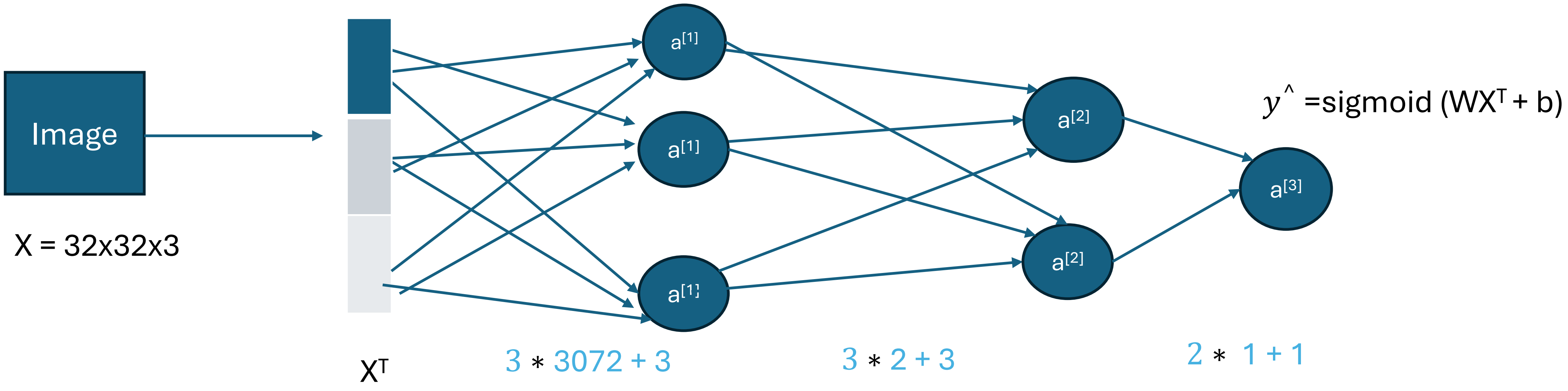
  - How to train
  - Backward propagation

Notations and Forward Pass

- Output of a layer l is given by $$z^{[\ell]} = W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}$$
- Where W^l is the weights of the layer, b^l is the bias and a^l is the activation $$a^{[\ell]} = g^{[\ell]}(z^{[\ell]})$$
- Neuron = linear ($z^{[l]}$)+ activation ($a^{[l]}$)

Task: How to Train this MLP Network

Task: Predict whether an input image contains a cat or not



Steps: How to Train this MLP Network

- Forward Pass
  - Defining propagation equations
  - Defining loss function

- How to train
  - Gradient Descent
  - Backward propagation

Defining Propagation Equations

- X = [n x f]
- b =[n x1]

- First layer: W_1, bias = b_1
- Second layer : W_2, bias = b_2
- Third layer: W_3, bias = b_3

The Forward Pass

- Output of layer 1: $z^{[1]} = W^{[1]}x + b^{[1]}$

  $a^{[1]} = g(z^{[1]});$ $a^{[1]}$ = activation function

- Output of layer 2: $z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

  $a^{[2]} = g(z^{[2]});$ $a^{[2]}$ = activation function

- Output of layer 3: $z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}$

  $a^{[3]} = g(z^{[3]});$ $a^{[3]}$ = activation function

- Output of neural network: $y^{\wedge} = a^{[3]}$

Defining the Loss

- Task: Binary classification

- Loss: binary cross entropy or log loss

- L = $-[(1 - y)\log(1 - \ y^{\wedge}\ ) \ + \ y.\log y^{\wedge}]$

How to Train this MLP Network?

- Gradient Descent + Backpropagation!

- What is Backpropagation

  - Backprop or Backpropagation is a way to train multilayer neural networks using gradients

Gradient Descent

- 1. Provide random value for weight and biases

- 2. Update weights by gradient descent of the ith layer
  - Wi = wi - $\alpha\frac{\partial L}{\partial w_i}$
  - bi = bi - $\alpha\frac{\partial L}{\partial b_i}$

  - Repeat until convergence

Gradient Descent

- We have three weight matrices, Wi, i = 1, 2, 3.

- We need to compute loss and update weights for each weight matrix

- Wi = wi - $\alpha\frac{\partial L}{\partial w_i}$

What will be the ordering of updating the weight matrices? Justify your answer.

Gradient Descent and Backpropagation

- We have three weight matrices, Wi, i = 1, 2, 3.

- We need to compute loss and update weights for each weight matrix
- Wi = wi - $\alpha\frac{\partial L}{\partial w_i}$
- *What will be the ordering of updating the weight matrices? Justify your answer*
  - *W_3, W_2, W_1. The loss, L is computed from the output and the last layer of the network, W_3. So, it is straightforward to compute.*
  - *The relationship between L and W_1 is not straightforward to compute*
  - *This is called backpropagation, as it calculates gradients backwards through the network*

dL/dW$_3$

- $\frac{\partial \mathrm{L}}{\partial w_3} = \frac{\partial}{\partial w_3}[-[(1-y)\log(1- \ y^{\wedge}\ ) \ + \ y.\log y^{\wedge}]\,]$

$$= -\,[(1-y)\frac{\partial}{\partial w_3}\log(1- \ y^{\wedge}\ ) \ + \ y\frac{\partial}{\partial w_3}\log y^{\wedge}]$$

dL/dW$_3$

- $\frac{\partial L}{\partial w_3} = -\,[\frac{1-y}{(1-y^{\wedge})}\frac{\partial}{\partial w_3}(1 - y^{\wedge}) + \frac{y}{y^{\wedge}}\frac{\partial}{\partial w_3}y^{\wedge}]$ *[Law of calculus]*

$= -\,[\frac{-(1-y)}{(1-y^{\wedge})}\frac{\partial y^{\wedge}}{\partial w_3} + \frac{y}{y^{\wedge}}\frac{\partial y^{\wedge}}{\partial w_3}]$ *[Law of calculus, differentiating $\frac{\partial}{\partial w_3}(1 - y^{\wedge})$]*

$= [\frac{(1-y)}{(1-y^{\wedge})}\frac{\partial y^{\wedge}}{\partial w_3} - \frac{y}{y^{\wedge}}\frac{\partial y^{\wedge}}{\partial w_3}]$

$= [\frac{y^{\wedge}(1-y)-y(1-y^{\wedge})}{y^{\wedge}(1-y^{\wedge})}]\frac{\partial y^{\wedge}}{\partial w_3}$

dL/dW$_3$

- $\frac{\partial L}{\partial w_3} = [\frac{y^{\wedge}(1-y)-y(1-y^{\wedge})}{y^{\wedge}(1-y^{\wedge})}]\frac{\partial y^{\wedge}}{\partial w_3}$

$= [\frac{y^{\wedge}-yy^{\wedge}-y+yy^{\wedge}}{y^{\wedge}(1-y^{\wedge})}]\frac{\partial y^{\wedge}}{\partial w_3}$

$= [\frac{y^{\wedge}-y}{y^{\wedge}(1-y^{\wedge})}]\frac{\partial y^{\wedge}}{\partial w_3}$

We will compute $\frac{\partial y^{\wedge}}{\partial w_3}$ and then put its value to $\frac{\partial L}{\partial w_3}$

dL/dW$_3$

- $\frac{\partial y^{\wedge}}{\partial w_3} = \frac{\partial a_3}{\partial w_3} = \frac{\partial g(z_3)}{\partial w_3}$

If g is a sigmoid function, $g'(z_3)= g(z_3)(1- g(z_3))$

$$\frac{\partial g(z_3)}{\partial w_3} = g(z_3)(1- g(z_3)) \frac{\partial z_3}{\partial w_3}$$

$$= a_3(1- a_3) \frac{\partial}{\partial w_3}(w_3 a_2 + b_3)$$ *[Replacing values from forward pass equations]*

$$= a_3(1- a_3)\, a_2$$

$$= y^{\wedge}(1 - y^{\wedge})\, a_2$$

dL/dW$_3$

$$\frac{\partial L}{\partial w_3} = \left[\frac{(y^{\wedge} - y)}{y^{\wedge}(1 - y^{\wedge})}\right] y^{\wedge}(1 - y^{\wedge})\, a_2$$ [Replacing value of $\frac{\partial y^{\wedge}}{\partial w_3}$]

$$\frac{\partial L}{\partial w_3} = (y^{\wedge} - y)\, a_2$$

$$\frac{\partial L}{\partial w_3} = (a^{[3]} - y)\, a_2{}^{T}$$ (vectoral solution)

dL/db_3

- Similar derivation

$$\frac{\partial L}{\partial b_3} = (a^{[3]} - y)$$

Finding dL/dW_2

- We need to use the chain rule from calculus.

Why?

Finding dL/dW_2

- We need to use the chain rule from calculus.
- Why?

  - There is no direct relationship between the weights W_2 and the loss L.

  - We cannot differentiate a variable by another if there is no direct relationship
    - Else it would be 0
    - Not true if the variable/function is composite

Finding dL/dW_2

- We know loss is dependent on $y^\wedge$ = $a^{[3]}$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{?} \cdot \frac{?}{\partial w_2}$$

We know $a^{[3]} = g(z^{[3]})$.

Putting this value to make differentiable w.r.t. $z^{[3]}$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{?} \cdot \frac{?}{\partial w_2}$$

Finding dL/dW$_2$

- $z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}$
- $z^{[3]}$ is dependent on $a^{[2]}$. So, we add $\frac{\partial z_3}{\partial a_2}$ to make the component differentiable w.r.t. $a^{[2]}$
- $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{?} \cdot \frac{?}{\partial w_2}$

Finding dL/dW$_2$

$a^{[2]} = g(z^{[2]})$;

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{?} \cdot \frac{?}{\partial w_2}$$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$. So the $\frac{\partial L}{\partial w_2}$ becomes

$$\frac{\partial L}{\partial w_2} = \underbrace{\frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3}}_{\text{We already know this value}} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

Finding dL/dW_2

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

Rewrite $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_3}$

Recap: dL/dW$_3$

- $\frac{\partial y^{\wedge}}{\partial w_3} = \frac{\partial a_3}{\partial w_3} = \frac{\partial g(z_3)}{\partial w_3}$

If g is a sigmoid function, $g'(z_3)= g(z_3)(1- g(z_3))$

$$\frac{\partial g(z_3)}{\partial w_3} = g(z_3)(1- g(z_3))\frac{\partial z_3}{\partial w_3}$$

$$= a_3(1- a_3)\frac{\partial}{\partial w_3}(w_3 a_2 + b_3)$$

$$= a_3(1- a_3)\, a_2$$

$$= y(1 - y^{\wedge})\, a_2$$

Thus, $\frac{\partial z_3}{\partial w_3} = a_2$

Finding dL/dW_2

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

Rewrite $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_3}$

We know $\frac{\partial L}{\partial w_3} = (a^{[3]} - y)\, a_2 T$

[See the final form of $\frac{\partial L}{\partial w_3}$]

Thus, $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_3} = (a^{[3]} - y)\, a_2{}^{T}$

$\boxed{\text{Because } \frac{\partial z_3}{\partial w_3} = a_2, \text{ so } \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} = = (a^{[3]} - y)}$

Finding dL/dW_2

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

Thus,

$$\frac{\partial L}{\partial w_2} = (a_3 - y) \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

$$= (a_3 - y).\ w_{3.}\ g'(z_2).\ a_1$$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$$\frac{\partial z_2}{\partial w_2} = a_1$$

$a^{[2]} = g(z^{[2]})$

$$\frac{\partial a_2}{\partial z_2} = g'(z^{[2]})$$

$z_3 = W^{[3]} a^{[2]} + b^{[3]}$

$$\frac{\partial z_3}{\partial a_2} = W_3$$

Finding dL/dW$_1$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$ *[Chain rule of derivative]*

$$= (a_3 - y) .\ w_3 .g'(z_2).\ w_2 .g'(z_1).x$$

$$\frac{\partial L}{\partial b_1} = (a_3 - y) .\ w_3 .\ g'(z_2).\ w_2 .g'(z_1)$$

Improving the Neural Network

Improving the Neural Network

- Initialization
  - Normalization
  - Mitigating vanishing and exploding gradient

- Optimization

- Initialization
  - Normalization
  - Mitigating vanishing and exploding gradient
- Optimization

Normalizing Input

- Normalizing can help improve the speed and accuracy of the machine learning model.

- Performed on input features before feeding them into a NN
- Ensures features are in same scale
- Ensure all features contribute equally
- Neural networks perform better with smaller values (0-1), easier to train

Normalizing Input

- Z-score
  - For each feature type
    - convert each value into its corresponding z-score by subtracting mean from every value and dividing by standard deviation of all values (mean normalization)

- Min-max normalization
  - For each feature type
    - divide each entry by maximum possible value (min-max scaling)

- Initialization
  - Normalization
  - Mitigating vanishing and exploding gradient
- Optimization

Recap: The Forward Pass

- Output of layer 1: $z^{[1]} = W^{[1]}x + b^{[1]}$

  $a^{[1]} = g(z^{[1]});$ $a^{[1]}$ = activation function

- Output of layer 2: $z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

  $a^{[2]} = g(z^{[2]});$ $a^{[2]}$ = activation function

- Output of layer 3: $z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}$

  $a^{[3]} = g(z^{[3]});$ $a^{[3]}$ = activation function

- Output of neural network: $y^{\wedge} = a^{[3]}$

Vanishing and Exploding Gradient

- [From forward pass equations, replace $a^{[l]}$ values gradually]

- $y^{\wedge}$ = $a^{[3]} = W^{[3]} W^{[2]} W^{[1]} x + \ldots.$

- Weight matrices are initialized randomly and updated in the training process

- *How does the initialization of weight matrices impact the output?*

Vanishing Gradient

- $y^{\wedge} = a^{[3]} = W^{[3]} W^{[2]} W^{[1]} x$
- Assume all weight matrices, $W^{[L]}$ are initialized with a value little below than the identity matrix
- $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L}$

Multiplicative value of the resulting weight matrix will be very small, $y^{\wedge}$ will be close to zero (or vanish)

Exploding Gradient

- $y^{\wedge} = a^{[3]} = W^{[3]} W^{[2]} W^{[1]} x$

- Assume all weight matrices, $W^{[L]}$ are initialized with a value little higher than the identity matrix

- $\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L}$

Multiplicative value of the resulting weight matrix will be very big, $y^{\wedge}$ will explode

Solution to Vanishing and Exploding Gradients

- Initialize weight matrices will values ~ 1.
- In practice,

$W^{[l]}$ = np.random.randn(shape)*nsqrt($\frac{1}{n^{l-1}}$)

Number of inputs, n, coming to layer l

  - Initialize weight of the layer with the number of inputs coming to the layer
  - Works well for sigmoid activation function

  - ReLU
  - $W^{[l]}$ = np.random.randn(shape)*nsqrt($\frac{2}{n^{l-1}}$)

- Initialization
  - Normalization
  - Mitigating vanishing and exploding gradient

- Optimization

Optimization

- Batch Gradient descent
  - Updates gradient using the entire dataset

- Stochastic Descent
  - Updates gradient for each training example
  - As updates frequently, may stuck in local minima

- Trade-off between these two: mini-batch gradient descent
- *Why do we need mini-batch gradient descent?*

Optimization: Limitations

- If we have a lot of training example, say 1 million images
  - Batch gradient descent will take a long time to compute gradient once
  - Stochastic gradient descent will update frequently

- A trade-off solution
  - Update gradients, may be batches of 10,000 examples
  - That will give an approximate direction of gradient

Mini-Batch Gradient Descent: Definition, Advantages, and Disadvantages

- Mini-batch gradient descent splits the training dataset into small batches that are used to calculate ML model error and update gradients.

- Advantages
  - Update frequency is higher than batch gradient descent, allowing faster convergence in large dataset
  - Avoids local minima.
  - Gradient updates are computationally efficient than stochastic gradient descent.

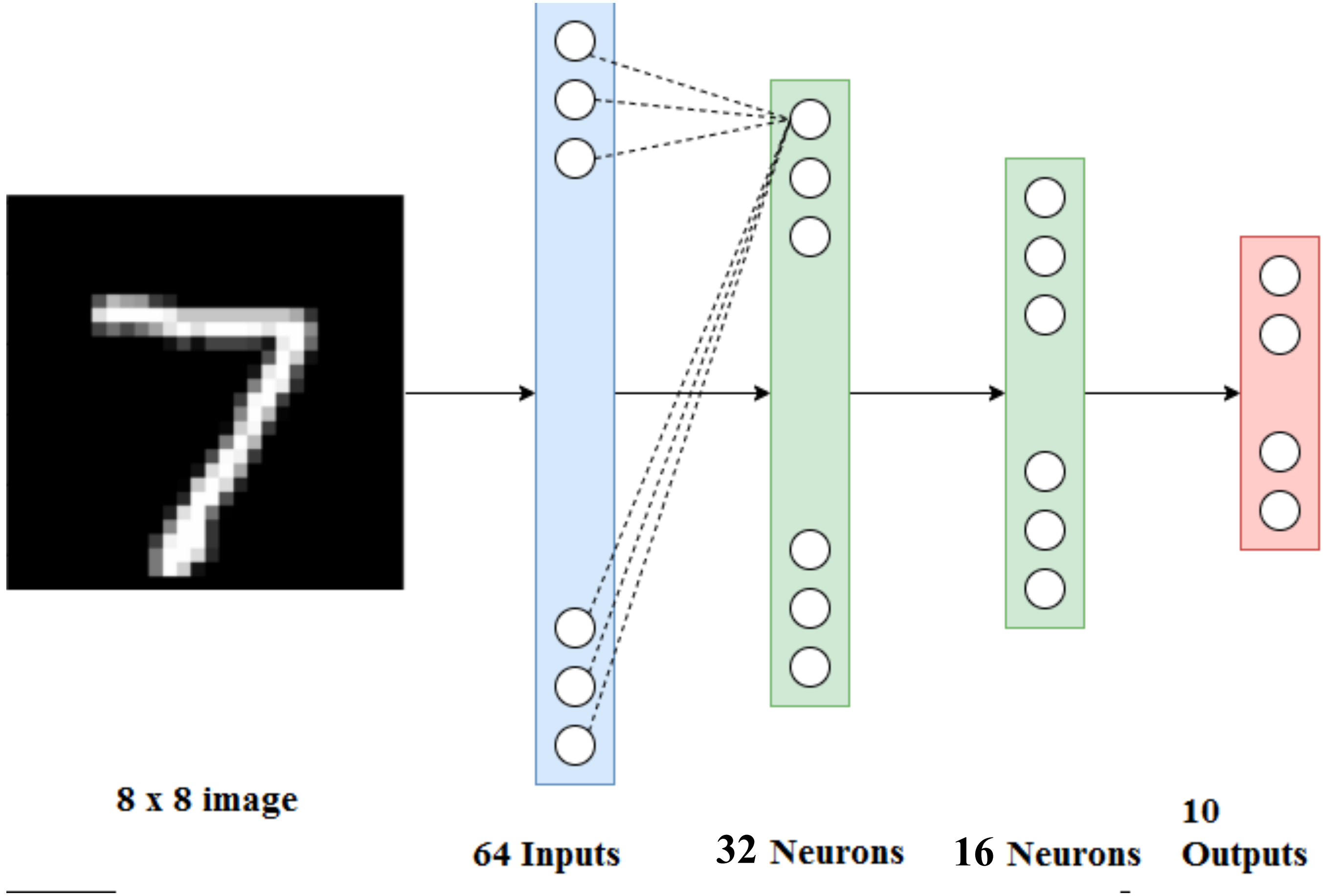
- Disadvantages
  - Requires an additional hyperparameter tuning: “batch size” for the learning algorithm.
  - Error information need to be accumulated across mini-batches of training examples like batch gradient descent.

Adapted from: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Colab Demo: MLP

- https://proceedings.mlr.press/v133/wang21a/wang21a.pdf
- https://2025.msrconf.org/track/msr-2025-mining-challenge
- https://2024.msrconf.org/track/msr-2024-mining-challenge?#Call-for-Mining-Challenge-Papers-

Task: Digit Classification on the MNIST Dataset



https://github.com/effat/MLP-Demo/blob/main/MLP.ipynb