NEURAL NETWORKS TRAINING

DR. FARHAD RAZAVI

OUTLINE

- Training Neural Network with TensorFlow
- Different activation functions
- Multiclass classification using Neural Network

MODEL TRAINING STEPS

1. Specify how to compute output given input x and parameters w, b (define model)

$$f_{\vec{w},b}(\vec{x}) = ?$$

2. Specify loss and cost:

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$

3. Train on data to minimize $J(\vec{w}, b)$.

Logistic Regression

```
z = np.dot(w, x) + b
f_x = 1/(1+np.exp(-z))
```

```
#logistic loss

loss = -y * np.log(f_x)
       -(1-y) * np.log(1 - f_x)
```

```
w = w - alpha * dj_dw
b = b - alpha * dj_db
```

Neural Network

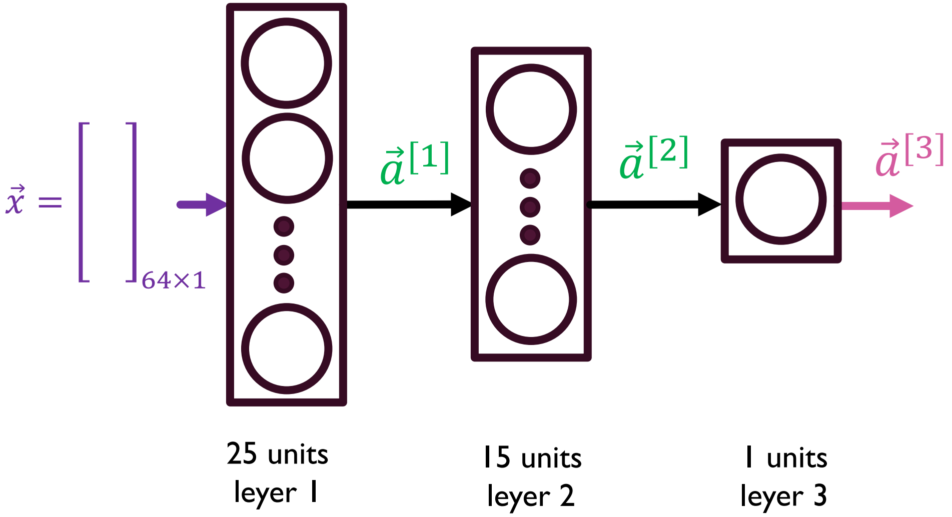
```
model = Sequential([
        Dense(...)
        Dense(...)
        Dense(...)
                    ])
```

```
#binary cross entropy

model.compile(
    loss=BinaryCrossentropy())
```

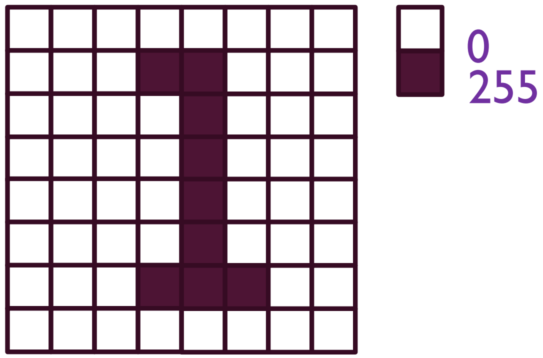
```
model.fit(X, y, epoch=100)
```

$$J(W, B)$$

HANDWRITTEN DIGIT RECOGNITION



Probability of being a handwritten ‘1’



$$\vec{a}^{[1]} = \begin{cases} f\left(\vec{w}_1^{[1]} . \vec{x} + b_1^{[1]}\right) \\ \vdots \\ f\left(\vec{w}_{25}^{[1]} . \vec{x} + b_{25}^{[1]}\right) \end{cases}$$

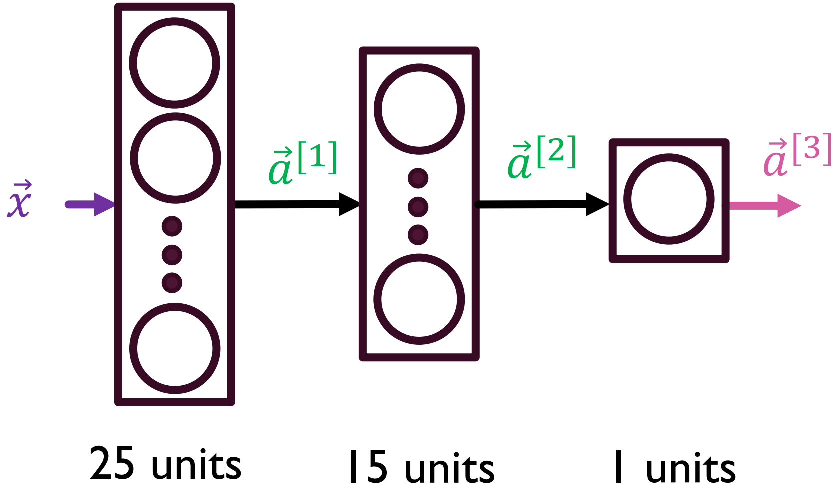
$$\vec{w}_1^{[1]} = \left[\quad\right]_{64\times1}, \vec{w}_2^{[1]} = \left[\quad\right]_{64\times1} \cdots, \vec{w}_{25}^{[1]} = \left[\quad\right]_{64\times1}$$

$$\boldsymbol{W}^{[1]} = \left[\vec{w}_1^{[1]}, \vec{w}_2^{[1]}, \cdots, \vec{w}_{25}^{[1]}\right]_{64\times25} \qquad \vec{b}^{[1]} = \left[b_1^{[1]}, b_2^{[1]}, \cdots, b_{25}^{[1]}\right]_{1\times25}$$

$$\boldsymbol{W}^{[2]} = \left[\vec{w}_1^{[2]}, \vec{w}_2^{[2]}, \cdots, \vec{w}_{15}^{[2]}\right]_{25\times15} \qquad \vec{b}^{[2]} = \left[b_1^{[2]}, b_2^{[2]}, \cdots, b_{25}^{[2]}\right]_{1\times15}$$

$$\boldsymbol{W}^{[3]} = \left[\vec{w}_1^{[3]}\right]_{15\times1} \qquad \vec{b}^{[3]} = \left[b_1^{[3]}\right]_{1\times1}$$

TENSOR FLOW IMPLEMENTATION (TRAINING)

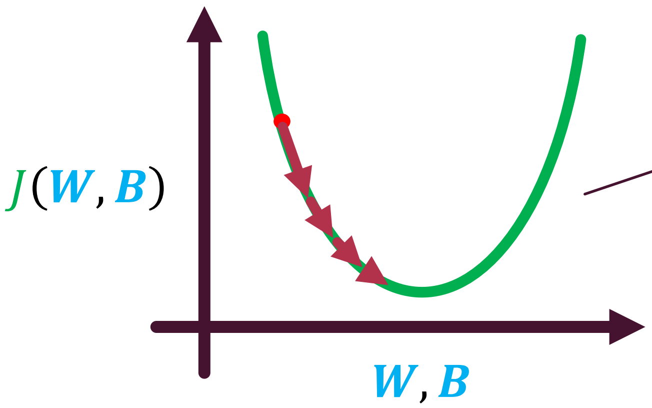



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(units = 25, activation='sigmoid'),
    Dense(units = 15, activation='sigmoid'),
    Dense(units = 1 , activation='sigmoid')])
```

$f_{W,B}(\vec{x})$

```
from tensorflow.keras.losses import BinaryCrossentropy

model.compile(loss=BinaryCrossentropy())

model.fit(X, Y, epochs = 100)
```

$L(f_{W,B}(\vec{x}^{(i)}), y^{(i)})$

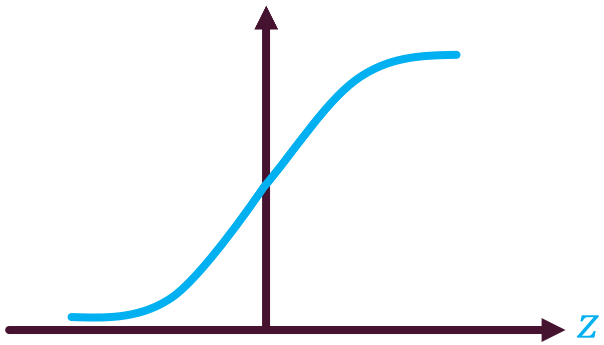


$$J(W,B) = \frac{1}{m}\sum_{i=1}^{m} L(f_{W,B}(\vec{x}^{(i)}), y^{(i)})$$

ALTERNATIVES TO SIGMOID FUNCTION

$$z = \vec{w}.\vec{x} + b$$

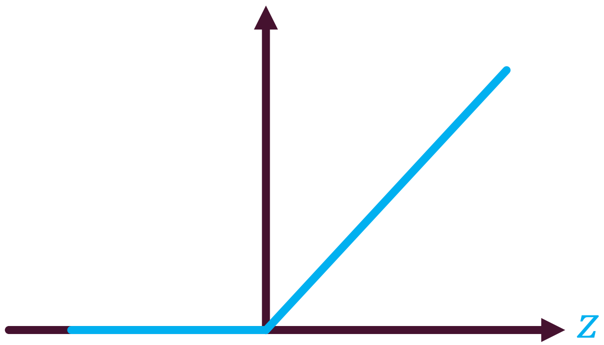
Sigmoid



$$g(z) = \frac{1}{1+e^{-z}}$$

$$0 < g(z) < 1$$

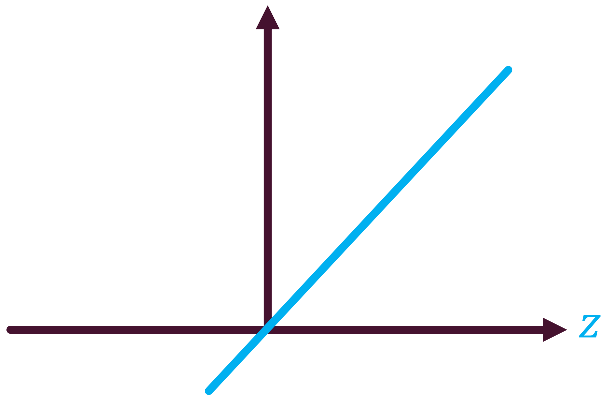
ReLU (Rectified Linear Unit)



$$g(z) = \max(0, z)$$

$$0 \leq g(z)$$

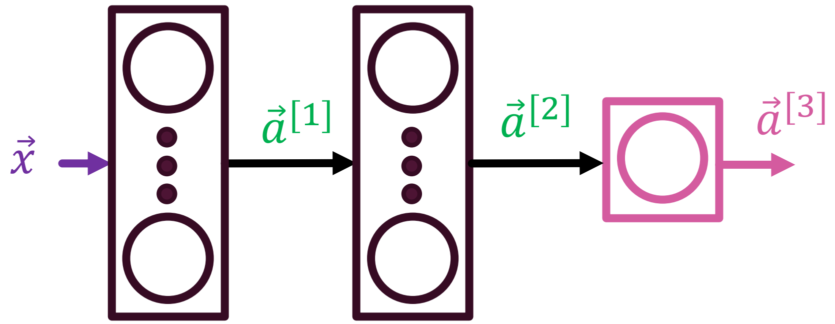
Linear activation function



$$g(z) = z = \vec{w}.\vec{x} + b$$

CHOOSING ACTIVATION FUNCTION (OUTPUT LAYER)

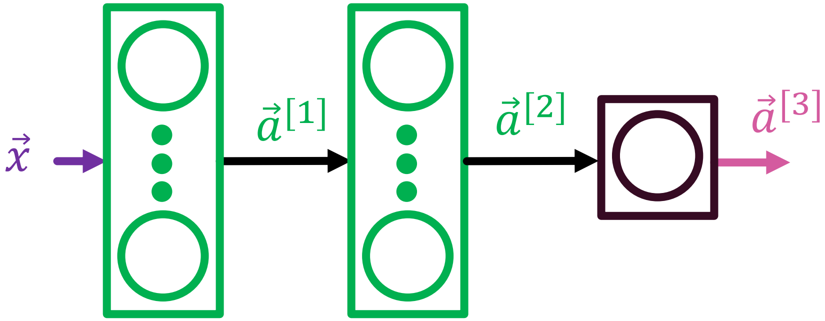
- The most natural way to choose the activation function in the output layer is to adapt it to the labels or target values of our training data.



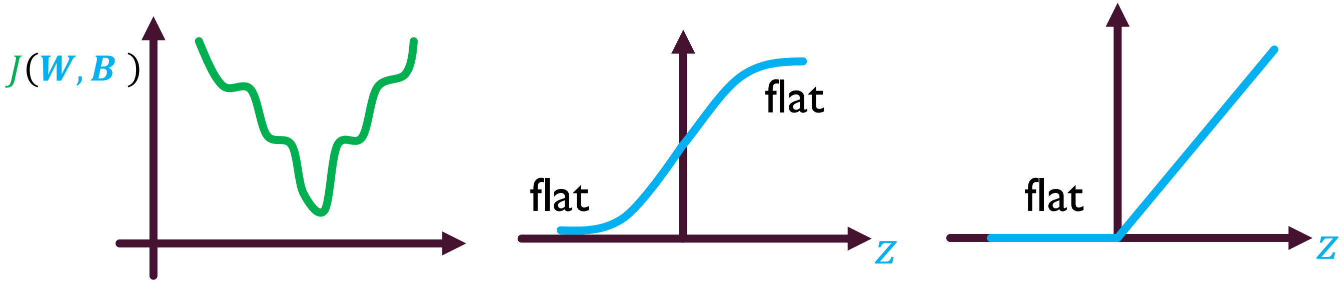
- If we are predicting True or False: → sigmoid activation is a natural choice (0,1)
- If we are predicting the stock prices changes with respect to the previous day → linear activation (+ and -)
- If we are predicting the price of a home → ReLU activation (price of home is always positive +)

CHOOSING ACTIVATION FUNCTION (HIDDEN LAYER)

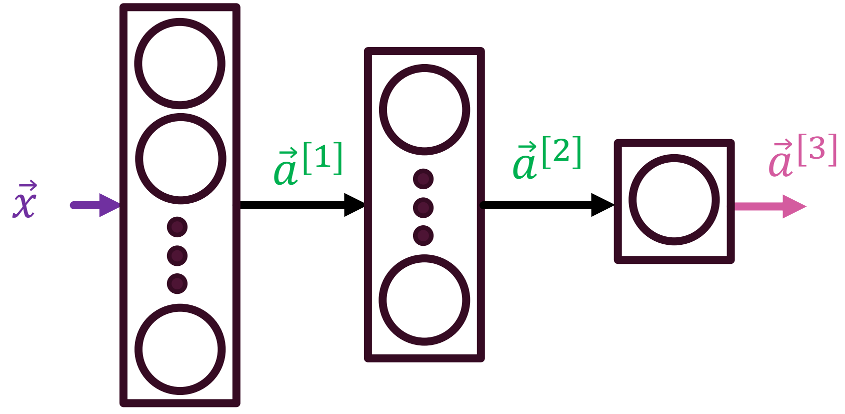
- Nowadays, the most natural way to choose the activation function in the hidden layer is to choose ReLU.



- Computationally they are far more efficient. Sigmoid requires computation of exponential values.
- It can be shown that the flatness of cost function can be significantly reduced if ReLU is used.
- This will help with faster convergence.



ACTIVATION FUNCTION IN TENSORFLOW




```python
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

binary classification

```python
model = Sequential([
    Dense(units = 25, activation='relu'),
    Dense(units = 15, activation='relu'),
    Dense(units = 1 , activation='sigmoid')])
```

regression y ±

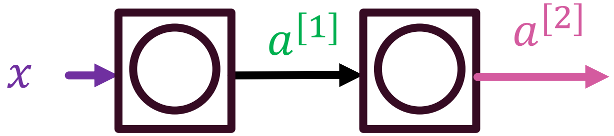
```python
model = Sequential([
    Dense(units = 25, activation='relu'),
    Dense(units = 15, activation='relu'),
    Dense(units = 1 , activation='linear')])
```

regression y≥0

```python
model = Sequential([
    Dense(units = 25, activation='relu'),
    Dense(units = 15, activation='relu'),
    Dense(units = 1 , activation='relu')])
```

- Read on other activation functions such as tanh, Leaky ReLU and swish … on https://www.tensorflow.org/api_docs/python/tf/keras/activations/

USING LINEAR ACTIVATION IN NEURAL NETWORK

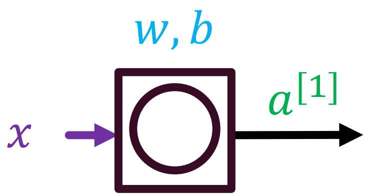


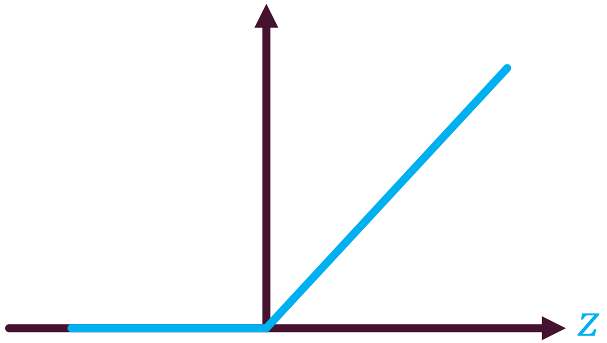
- A network of neurons with linear activation function can be reduced to one linear activation function.

- $a^{[1]} = w_1^{[1]} x + b_1^{[1]}$
- $a^{[2]} = w_1^{[2]} a^{[1]} + b_1^{[2]} = w_1^{[2]} \left(w_1^{[1]} x + b_1^{[1]} \right) + b_1^{[2]}$
- $a^{[2]} = \underbrace{w_1^{[2]} w_1^{[1]}}_{w} x + \underbrace{w_1^{[2]} b_1^{[1]} + b_1^{[2]}}_{b} = wx + b$

- Don't use the linear activation function in the hidden layers.

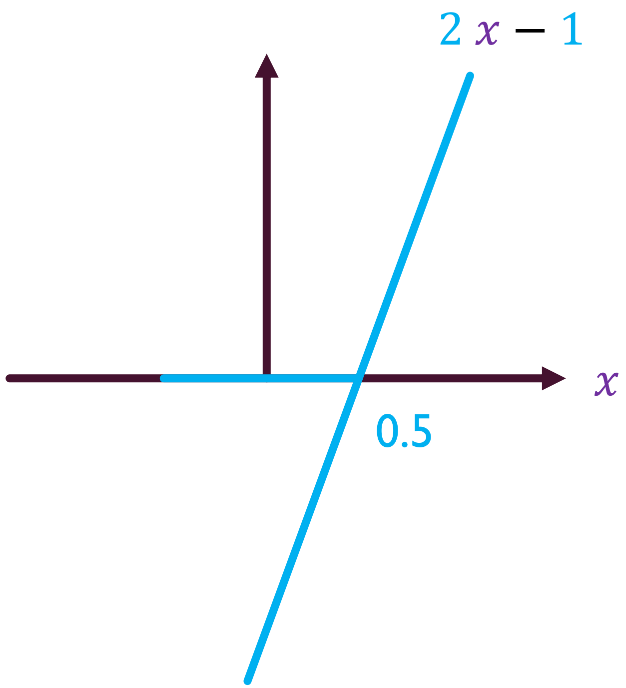
RELU PARAMETERS





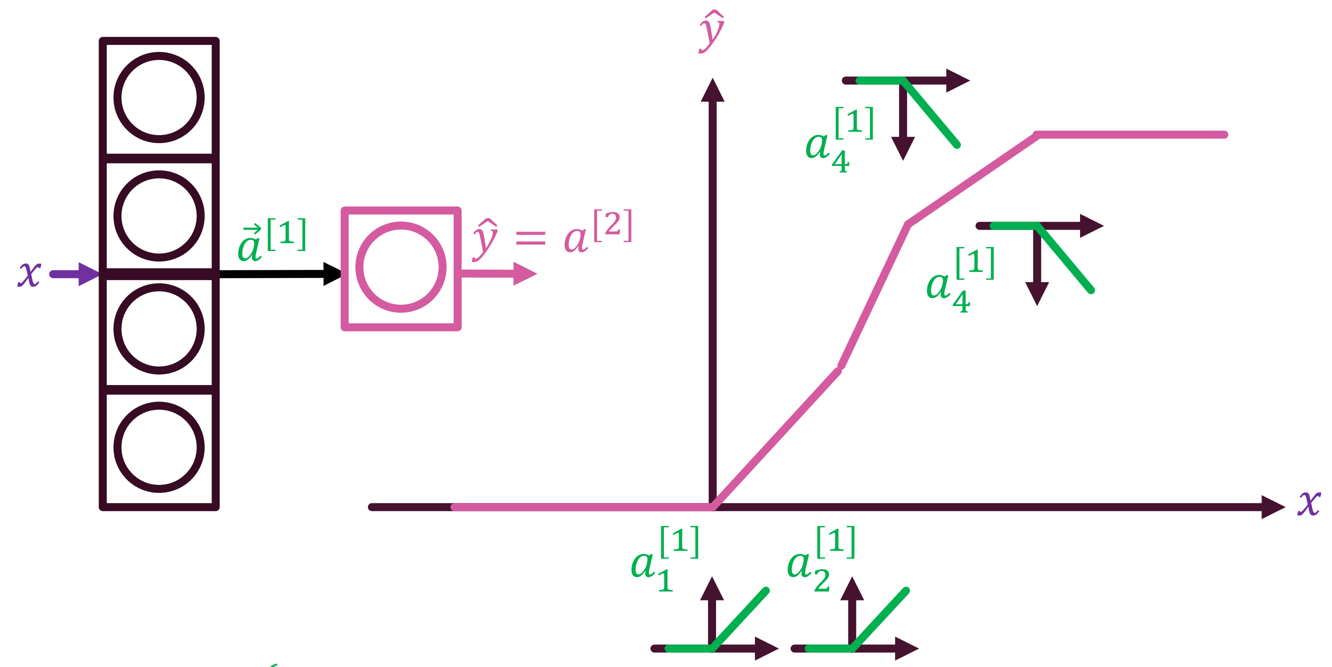
$$z = w\,x + b$$

$$a^{[1]} = \mathrm{ReLU}(z) = \max(0,\ z) = \max(0, w\,x + b)$$



$$w = 2; b = -1;\ a^{[1]} = \max(0, 2\,x - 1)$$

APPROXIMATING ANY FUNCTION



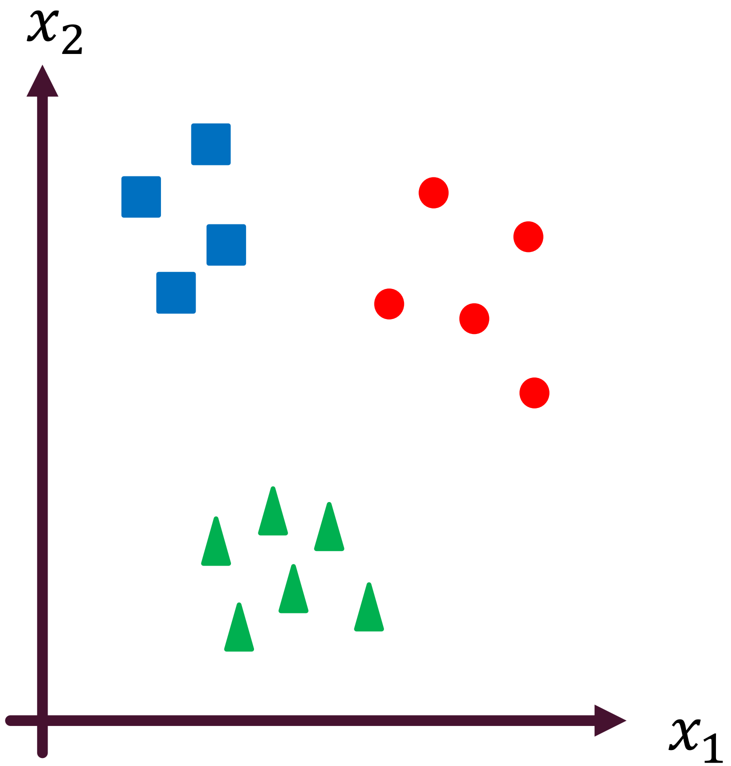
$$\vec{a}^{[1]} = \begin{cases} \text{relu}\left(w_1^{[1]}x + b_1^{[1]}\right) = \text{relu}(x-0) \\ \text{relu}\left(w_2^{[1]}x + b_2^{[1]}\right) = \text{relu}(x-1) \\ \text{relu}\left(w_3^{[1]}x + b_3^{[1]}\right) = \text{relu}(x-2) \\ \text{relu}\left(w_4^{[1]}x + b_4^{[1]}\right) = \text{relu}(x-3) \end{cases}$$

$$\vec{w}_1^{[2]} = [1,1,-1,-1]; b_1^{[2]} = 0$$

$$\hat{y} = a^{[2]} = \vec{w}_1^{[2]}.\vec{a}^{[1]} + b_1^{[2]} = \text{relu}(x) + \text{relu}(x-1) - \text{relu}(x-2) - \text{relu}(x-3)$$

- ReLU is a function with a very simple nonlinearity.
- Any piece-wise linear function can be modeled with a ReLU activation.
- One just need to add enough neurons on a layer to create complex models.

MULTICLASS CLASSIFICATION



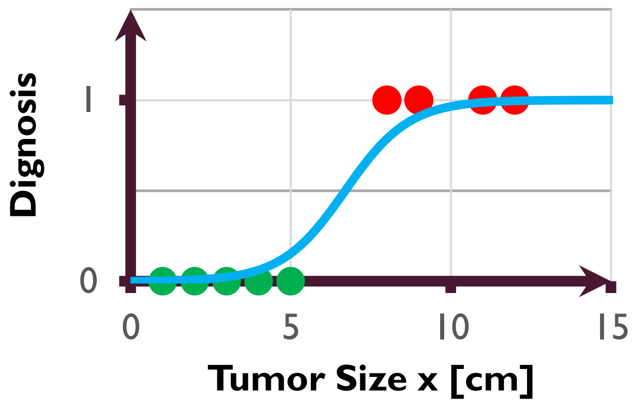
MNIST

SOFTMAX

- Recall that for logistic regression we used the sigmoid function to model the following probability of a new sample $\vec{x}$ belonging to category $y = 1$.

$$a_1 = g(z) = g(\vec{w}.\vec{x} + b) = \frac{1}{1 + e^{-z}} = \text{P}(y = 1|\vec{x})$$

$$a_2 = 1 - a_1 = \text{P}(y = 0|\vec{x})$$



- Let's generalize this with more than two categories.
- Category 1:

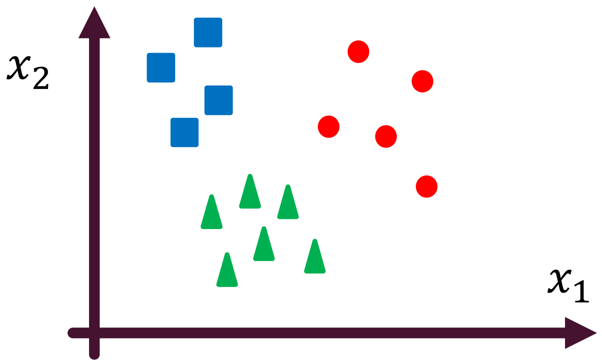
$$a_1 = \text{P}(y = 1|\vec{x})$$

- Category 2:

$$a_2 = \text{P}(y = 2|\vec{x})$$

- Category 3:

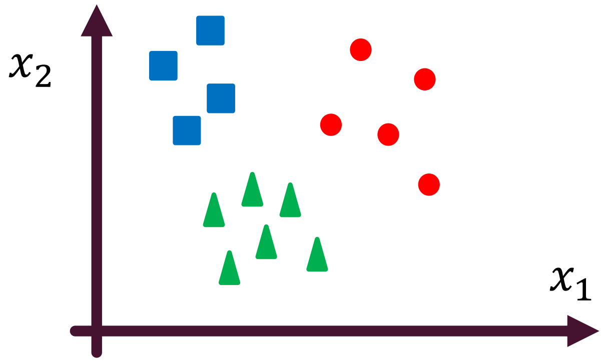
$$a_3 = \text{P}(y = 3|\vec{x})$$



SOFTMAX

- For each category we will define a different z parameter.

$$z_1 = \vec{w}_1.\vec{x} + b_1$$
$$z_2 = \vec{w}_2.\vec{x} + b_2$$
$$z_3 = \vec{w}_3.\vec{x} + b_3$$



- Then the probability of the feature belonging to each category can be modeled as.

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} = \mathrm{P}(y = 1|\vec{x}) \qquad 0.5$$
$$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}} = \mathrm{P}(y = 2|\vec{x}) \qquad 0.2$$
$$a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}} = \mathrm{P}(y = 3|\vec{x}) \qquad 0.3$$

SOFTMAX GENERALIZED

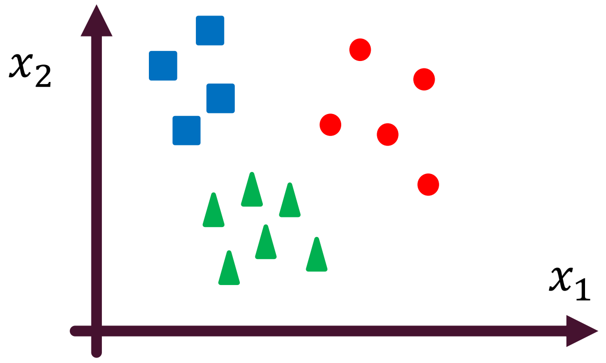
- For K different category each category we will define a different z parameter.

$$z_j = \vec{w}_j . \vec{x} + b_j \qquad\qquad j = 1, \dots, K$$

- Then the probability of the feature belonging to each category can be modeled as.

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} = \frac{e^{z_j}}{e^{z_1} + e^{z_2} + \cdots + e^{z_K}} = \mathrm{P}(y = j|\vec{x})$$

$$\sum_{j=1}^{K} a_j = 1$$



SOFTMAX COST

- For Logistic Regression:

$$a_1 = g(z) = \frac{1}{1+e^{-z}} = \mathrm{P}(y=1|\vec{x})$$

$$a_2 = 1 - a_1 \qquad = \mathrm{P}(y=0|\vec{x})$$

- We defined the loss as follow:

$$Loss = \begin{cases} -\log\big(g(z)\big) & if \quad y=1 \\ -\log\big(1-g(z)\big) & if \quad y=0 \end{cases}$$

$$J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}[Loss]$$

SOFTMAX COST

- For Logistic Regression:

$$a_1 = g(z) = \frac{1}{1+e^{-z}} = \mathrm{P}(y=1|\vec{x})$$

$$a_2 = 1 - a_1 \qquad = \mathrm{P}(y=0|\vec{x})$$

- We defined the loss as follow:

$$Loss = \begin{cases} -\log(a_1) & if \quad y=1 \\ -\log(1-a_1) & if \quad y=0 \end{cases}$$

$$Loss = \begin{cases} -\log(a_1) & if \quad y=1 \\ -\log(a_2) & if \quad y=0 \end{cases}$$

- For softmax regression:

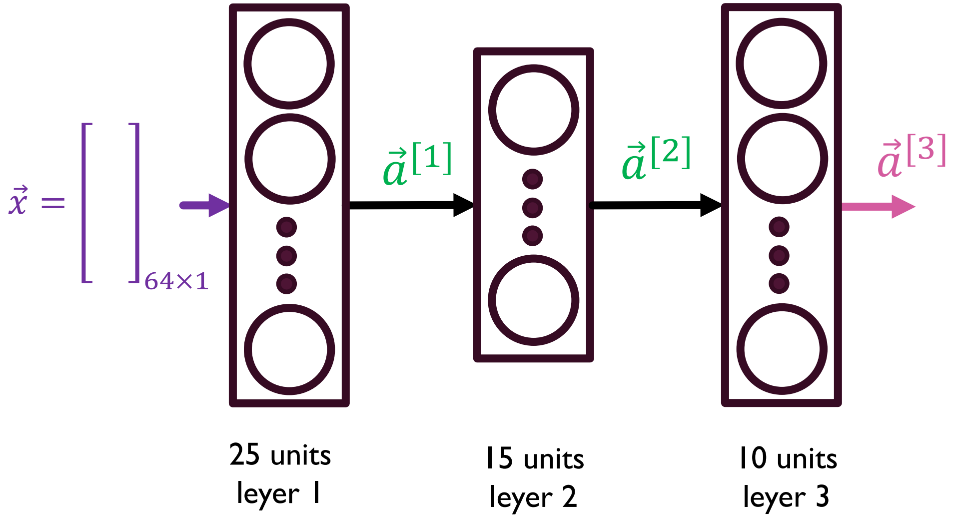
$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} = \mathrm{P}(y=j|\vec{x}) \qquad j = 1, \ldots, K$$

- Similarly, the loss will be defined as:

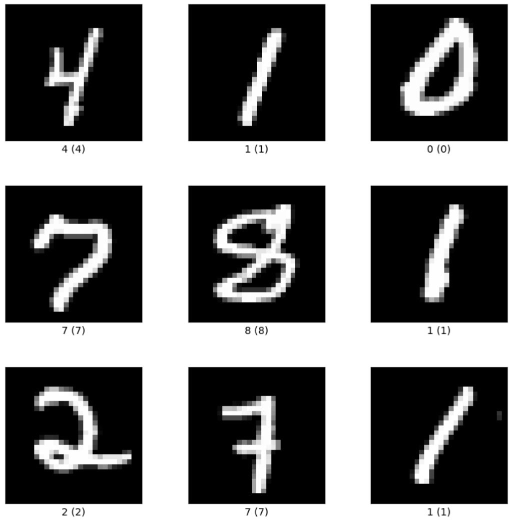
$$Loss = \begin{cases} -\log(a_1) & if \quad y=1 \\ -\log(a_2) & if \quad y=2 \\ \vdots & \\ -\log(a_K) & if \quad y=K \end{cases}$$

- This is called Crossentropy loss.

HANDWRITTEN DIGIT RECOGNITION



$$\vec{a}^{[1]} = \begin{cases} f\left(\vec{w}_1^{[1]}.\vec{x} + b_1^{[1]}\right) \\ \vdots \\ f\left(\vec{w}_{25}^{[1]}.\vec{x} + b_{25}^{[1]}\right) \end{cases}$$



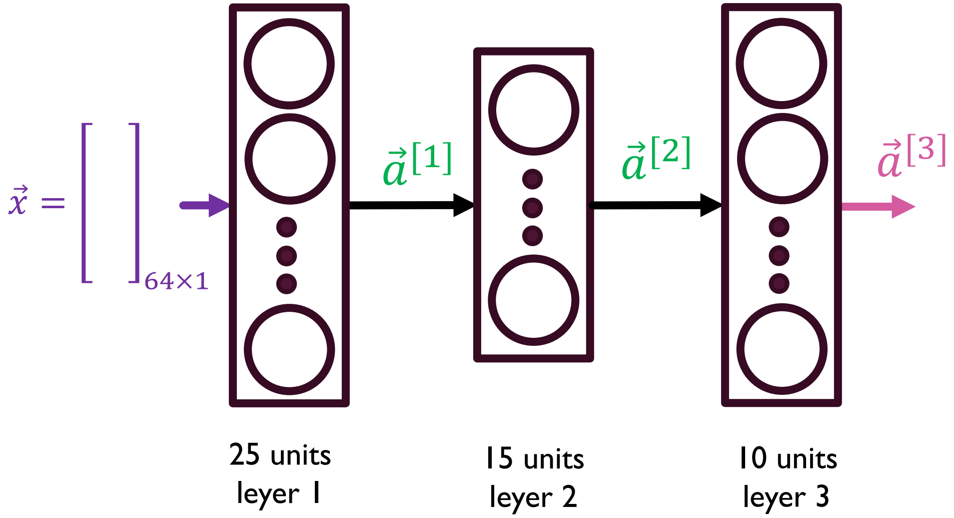
$$\vec{w}_1^{[1]} = \begin{bmatrix} \ \end{bmatrix}_{64\times1}, \vec{w}_2^{[1]} = \begin{bmatrix} \ \end{bmatrix}_{64\times1} \cdots, \vec{w}_{25}^{[1]} = \begin{bmatrix} \ \end{bmatrix}_{64\times1}$$

$$\boldsymbol{W}^{[1]} = \left[\vec{w}_1^{[1]}, \vec{w}_2^{[1]}, \cdots, \vec{w}_{25}^{[1]}\right]_{64\times25} \qquad \vec{b}^{[1]} = \left[b_1^{[1]}, b_2^{[1]}, \cdots, b_{25}^{[1]}\right]_{1\times25}$$

$$\boldsymbol{W}^{[2]} = \left[\vec{w}_1^{[2]}, \vec{w}_2^{[2]}, \cdots, \vec{w}_{15}^{[2]}\right]_{25\times15} \qquad \vec{b}^{[2]} = \left[b_1^{[2]}, b_2^{[2]}, \cdots, b_{25}^{[2]}\right]_{1\times15}$$

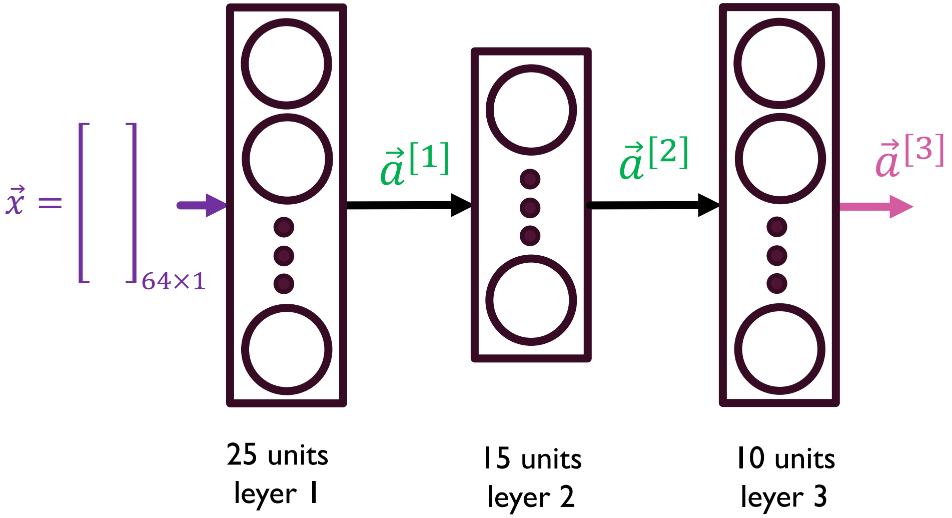
$$\boldsymbol{W}^{[3]} = \left[\vec{w}_1^{[3]}, \vec{w}_2^{[3]}, \cdots, \vec{w}_{10}^{[3]}\right]_{15\times10} \qquad \vec{b}^{[3]} = \left[b_1^{[3]}, b_2^{[3]}, \cdots, b_{10}^{[3]}\right]_{1\times10}$$

TENSORFLOW IMPLEMENTATION




```python
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(units = 25, activation='relu'),
    Dense(units = 15, activation='relu'),
    Dense(units = 10, activation='softmax')])


from tensorflow.keras.losses import
        SparseCategoricalCrossentropy

model.compile(loss=SparseCategoricalCrossentropy())

model.fit(X, Y, epochs = 100)
```

TENSORFLOW IMPLEMENTATION (MORE ACCURATE)




```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(units = 25, activation='relu'),
    Dense(units = 15, activation='relu'),
    Dense(units = 10, activation='linear')])


from tensorflow.keras.losses import
        SparseCategoricalCrossentropy

model.compile(
   loss=SparseCategoricalCrossentropy(from_logits=True))

model.fit(X, Y, epochs = 100)

logit = model(X)

f_x = tf.nn.softmax(logit)
```

Due to taking the log of small numbers numerically is more accurate and deals with roundoff error better.

REFERENCE

- Advanced Learning Algorithms, Andrew Ng, Stanford Online, DeepLearning.AI

https://towardsdatascience.com/animations-of-logistic-regression-with-python-31f8c9cb420