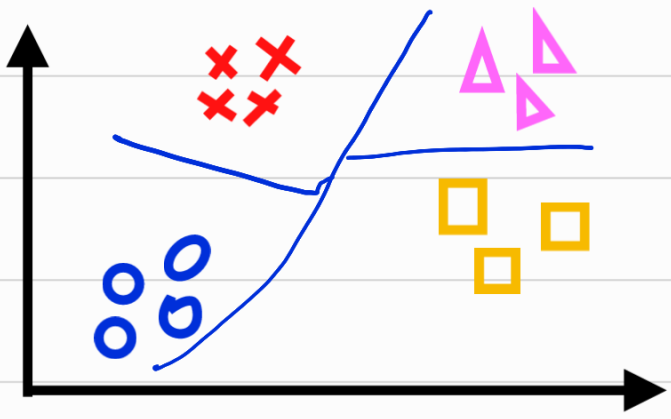
MNIST

$\hat{y}$ = 0, 1, 2, 3, ...

more than 2 problem values



SoftMax Regression

eg:- 4 outputs $y = 1, 2, 3, 4$

$z_1 = \vec{w}_1 \cdot \vec{x} + b_1$

$z_2 = \vec{w}_2 \cdot \vec{x} + b_2$

$z_3 = \vec{w}_3 \cdot \vec{x} + b_3$

$z_4 = \vec{w}_4 \cdot \vec{x} + b_4$

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 1 \mid \vec{x})$$

$$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + \dots}$$

$$a_3 = \frac{e^{z_3}}{\dots}$$

$$a^4 \dots$$

for N possible values $y=0 \to n$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} \longrightarrow a_1 + a_2 + \dots a_N = 1$$

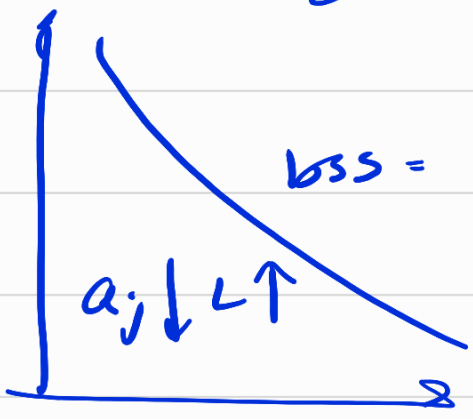
$a_2 = 1 - a_1$

so $\text{loss} = \underbrace{-y \log a_1}_{\text{if } y=1} \underbrace{-(1-y) \log(1-a_1)}_{\text{if } y=0}$ (a_2)

Softmax reg.:

$y=1 \Rightarrow -\log a_1$

$y=2 \Rightarrow -\log a_2$

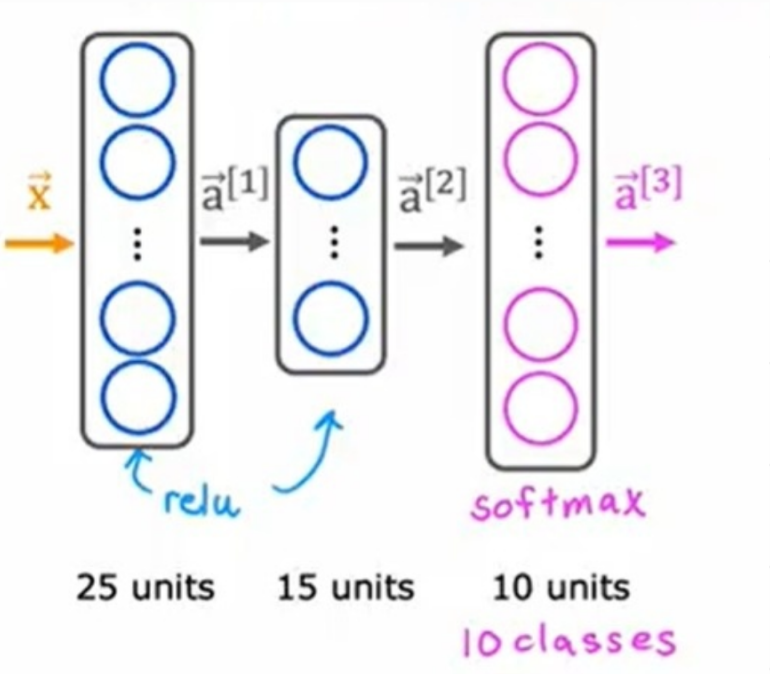


Multiclass model training → cross entropy

$\text{loss} = -\log(a_x)$ ← neuron of true class

others ignored

Neural Network with Softmax output



$$z_1^{[3]} = w_1^{[3]} \cdot a^{[2]} + b_1^{[3]}$$

$$\vdots$$

$$z_{10}^{[3]} = w_{10}^{[3]} \cdot a^{[2]} + b_{10}^{[3]}$$

1st neu.

$$a_1^{[3]} = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \dots + e^{z_{10}^{[3]}}}$$

→ SparseCategoricalCrossentropy

↓

each is 1 of categories

*But there's a better way in TensorFlow

MNIST with softmax

① specify the model

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

② specify loss and cost

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

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

```
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)
```

Numerical Roundoff errors

```
In [1]: x1 = 2.0 / 10000
        print(f"{x1:.18f}") # print 18 digits to the right of decimal point

        0.000200000000000000

In [2]: x2 = 1 + (1/10000) - (1 - 1/10000)
        print(f"{x2: .18f}")

         0.000199999999999978

In [ ]:
```

original loss $= -y \log a - (1-y) \log(1-a)$ ← $a = g(z)$

$$\text{loss} = -y \log\left(\frac{1}{1+e^{-z}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-z}}\right)$$

change

```
    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)
```

'softmax' → 'linear'

from_logits=True

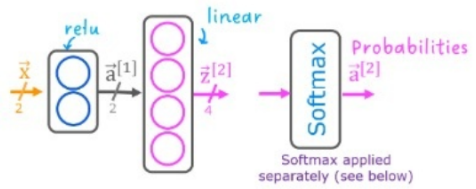
logits = model(X) ← not $a_1, a_2, \ldots a_{10}$

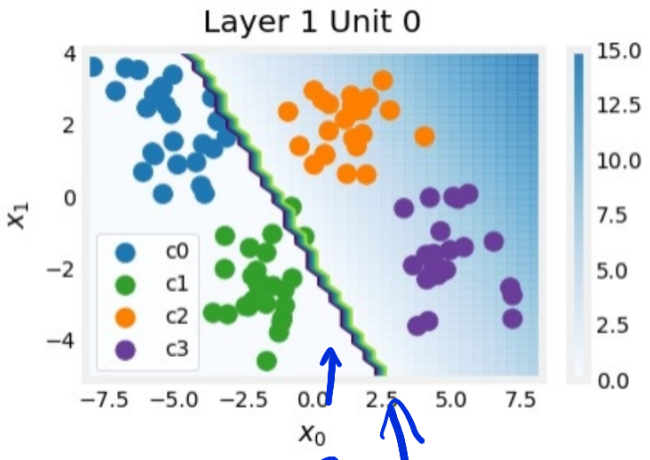
but $z_1, z_2 \ldots z_{10}$

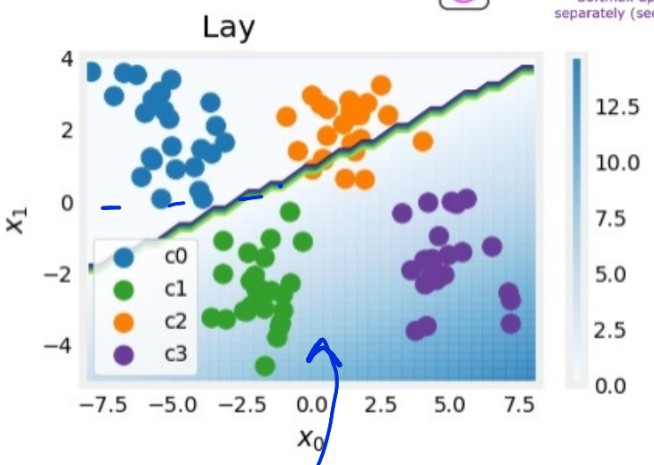
f_x = tf.nn.softmax(logits)

logits = scores (by each neuron) → highest score wins

1st Layer – ReLU



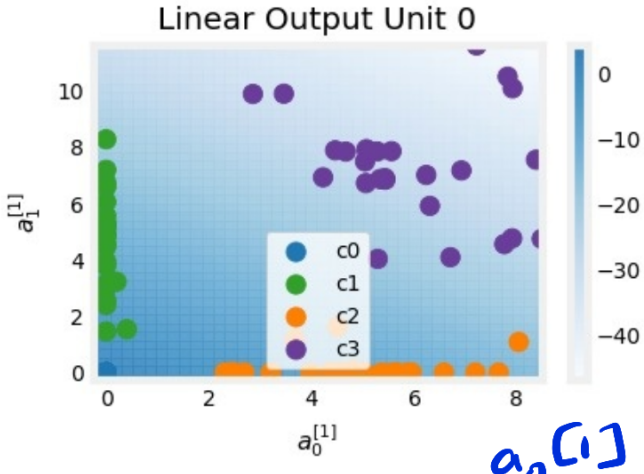


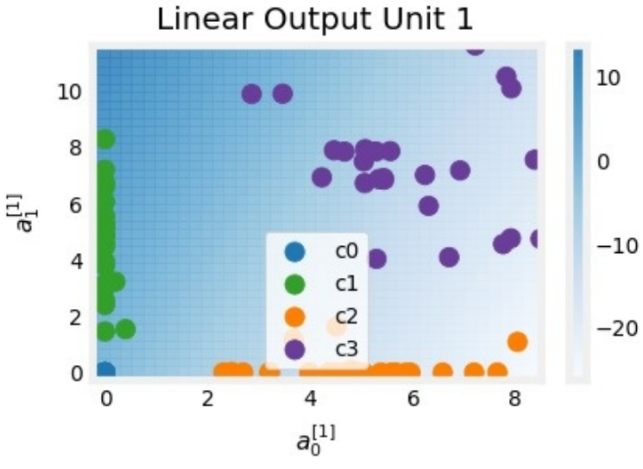


① if side or side

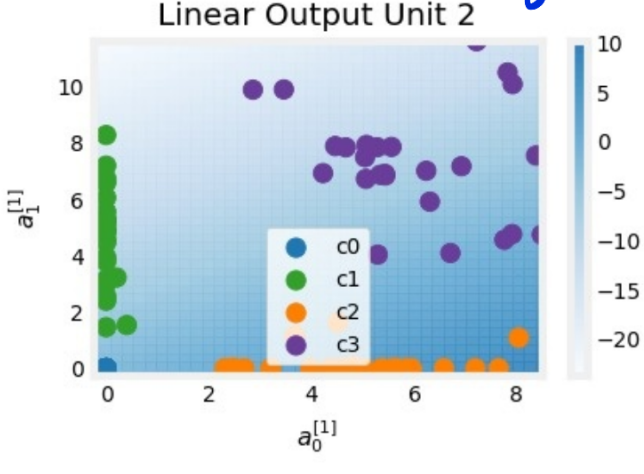
② side side

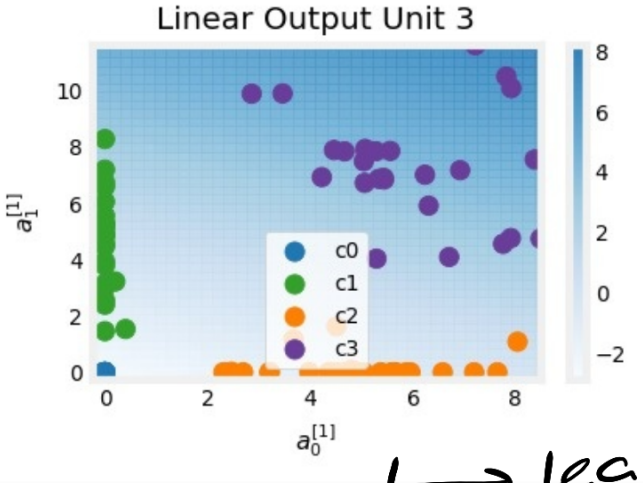
(a_0, a_1) $=0$ $=0$ — 4 cat.
>0 >0





$a_0[1]$





computes scores with

$z_k = w_{k0} a_0 + w_{k1} a_1 \ldots + b$

↳ learns weights to minimize loss

relatively make big than others

then prediction = $\text{argmax}(z_0, z_1, z_2, z_3)$

* classification with considering high score like a competition