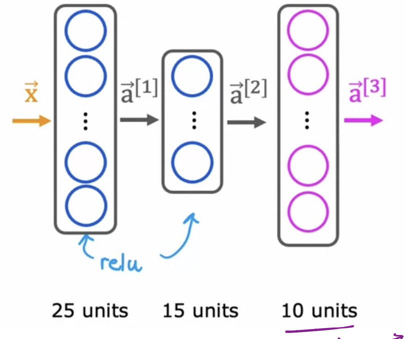
10 diff. categories = 10 diff. z's



$$\overbrace{z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}} \longrightarrow a_1^{[3]} = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \ldots + e^{z_{10}}}$$

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

$$z_2^{[3]} = \vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]} \longrightarrow a_2^{[3]} = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + \ldots + e^{z_{10}}}$$

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

$\vdots$

$$z_{10}^{[3]} = \vec{w}_{10}^{[3]} \cdot \vec{a}^{[2]} + b_{10}^{[3]} \longrightarrow a_{10}^{[3]} = \frac{e^{z_{10}}}{e^{z_1} + e^{z_2} + \ldots + e^{z_{10}}}$$

$P(y=10|\vec{x})$

softmax

with 10 different categories

logistics regression was a function of only one z.
$a_1^{[3]}$ would have been, $a_1^{[3]} = g(z_1^{[3]})$, $a_2^{[3]} = g(z_2^{[3]})$, ... etc.

softmax is a function of all z's, $a_1^{[3]} = g(z_1^{[3]})$, $a_2^{[3]} = g(z_1^{[3]})$
... etc.

Basic Implementation of softmax

```
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)
```

Note: better (recommended) version later.

Since the computer only has a finite amt. of memory to store each floating point no. certain complex calculations can result in round off errors.

The previous softmax implementation algorithm can be improved keeping this in mind.

More numerically accurate implementation of logistic loss.

Logistics Regression

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

$$\text{loss} = -y\log(a) - (1-y)\log(1-a)$$

More accurate loss

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

not allowing an intermediate term instead letting tensorflow handle it.

Instead of explicitely computing 'a' we give tensorflow the freedom to compute everything by stating the formula directly.

In code:

```
model = Sequential ([
                    Dense(units = 25 , activation = "relu"),
                    Dense (units = 15 , activation = "relu"),
                    Dense (units = 1 , activation = "~~sigmoid~~" linear)
                    ])
```

"don't need sigmoid anymore"

```
~~model. compile (loss = BinaryCrossEntropy ())~~
model. compile ( loss = BinaryCrossEntropy (from_logits = True))
```

puts the sigmoid function and the loss function togethere

logit = z

$$\underbrace{loss = -y \log\Big(\underbrace{\frac{1}{1+e^{-z}}}_{\text{activation function}}\Big) - (1-y)\log\Big(1 - \frac{1}{1+e^{-z}}\Big)}_{+\ \text{loss function}}$$

Only one downside is that the code becomes less legible.

Applying this method to softmax regression.

Old Method

$$(a_1, \ldots, a_{10}) = g(z_1, \ldots, z_{10})$$

$$\text{Loss} = L(\vec{a}, y) \begin{cases} -\log a_1, & \text{if } y=1 \\ \vdots \\ -\log a_{10}, & \text{if } y=10 \end{cases}$$

Better Method

$$L(\vec{a}, y) = \begin{cases} -\log \dfrac{e^{z_1}}{e^{z_1}+\ldots+e^{z_{10}}}, & \text{if } y=1 \\ \vdots \\ -\log \dfrac{e^{z_{10}}}{e^{z_1}+\ldots+e^{z_{10}}}, & \text{if } y=10 \end{cases}$$

tensorflow can avoid some unnecessary calculations when we define everything implicitly.

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

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

```
model    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') ])
loss     from tensorflow.keras.losses import
           SparseCategoricalCrossentropy
         model.compile(...,loss=SparseCategoricalCrossentropy(from_logits=True) )
fit      model.fit(X,Y,epochs=100)
predict  logits = model(X)
         f_x = tf.nn.softmax(logits)
```

← instead of outputting $a_1, \ldots, a_{10}$ it outputs $z_1, \ldots, z_{10}$

Basically, in model.compile (from_logits = False) the loss function in model.compile expects the inputs to already be in the form of probability distribution by using final activation function as 'softmax'.

But, in from_logits = True means that it expects raw input (z) and calculates loss with the $\frac{\sum_{j=1} e^{z_j}}{e^{z_1} + \ldots + e^{z_N}}$ function embedded in it. We use tf.nn.softmax later to get probability distribution.

Besides, SparseCategoricalCrossentropy, we have CategoricalCrossentropy which means that the output $\hat{y}$ wouldn't be a concrete value like 1, 2, 3, ... it would be a one-hot encoded vector where only the correct label is 1 and other N-1 choices are 0.

eg. There are 10 categories :- 1 2 3 4 5 6 7 8 9 10

$\vec{x}$ has true label 4, then $\hat{y} = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$