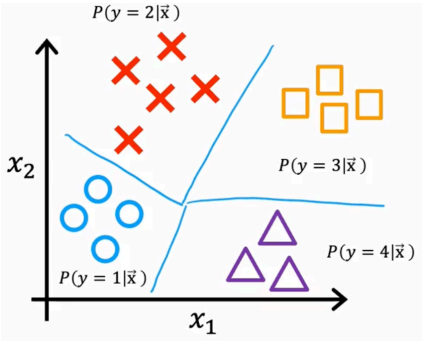
Multiclass Classification

Multiclass Classification Overview

- Multiclass classification deals with problems where there are multiple possible output labels, not just binary outcomes (0 or 1).
- Examples include recognizing handwritten digits (0-9), classifying diseases, or identifying defects in manufactured products.

Data Representation

- In multiclass classification, the dataset can have multiple classes represented by different symbols (e.g., circles, crosses, triangles).
- The goal is to estimate the probability of each class for a given input, rather than just two outcomes.



$$P(y=1|\vec{x}), P(y=2|\vec{x}), \ldots, P(y=4|\vec{x})$$

Softmax Regression

(N possible outputs)

Softmax regression is like a more advanced version of logistic regression, which helps us classify things into multiple categories instead of just two. Assume the output variable y can take on one of the values in the set $\{1, 2, 3, \ldots, N\}$. Let a_j represent the probability that $y = j$, for each class $j = 1, 2, \ldots, N$. The model uses a set of parameters $\vec{w}_j$ and b_j for each class j. Based on these definitions, the **softmax regression** algorithm operates as follows:

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \qquad a_j = g(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{N} e^{z_i}} = P(y = j|\vec{x})$$

Therefore, the sum of all probabilites is 1:

$$a_1 + a_2 + \ldots + a_N = 1$$

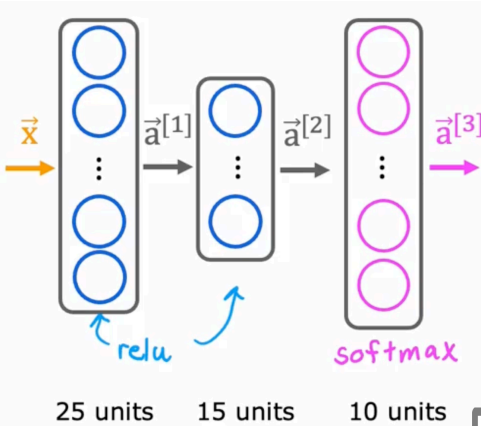
Loss Function (Crossentropy)

The loss function for softmax regression is defined as the negative log of the predicted probabilities for the actual class.

$$loss(a_1, a_2, \ldots, a_N) = \begin{cases} -\log a_1 & y = 1 \\ -\log a_2 & y = 2 \\ \vdots & \\ -\log a_N & y = N \end{cases}$$

Neural Network with Softmax Output

Example: Handwritten Digit Classification needs to predict 10 digits



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layer 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)
```

Normal version (recommended: don't use)

Improved Implemention of Softmax

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic Regression:

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

Original Loss:

$loss = -y\log(a) - (1 - y)\log(1 - a)$

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=20, activation='sigmoid')
                  ])
model.compile(loss=BinaryCrossEntropy())
```

More accurate loss (in code):

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

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=20, activation='linear')
                  ])
model.compile(loss=BinaryCrossEntropy(from_logits=True))
```

More numerically accurate implementation of logistic loss:

Softmax regression:

$(a_1, a_2, \ldots, a_N) = g(z_1, z_2, \ldots, z_N)$

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=20, activation='softmax'),
                  ])
model.compile(loss=SparseCategoricalCrossEntropy())
```

$$loss = L(\vec{a}, y) =$$
$$\begin{cases} -\log a_1 & y = 1 \\ -\log a_2 & y = 2 \\ \vdots & \\ -\log a_N & y = N \end{cases}$$

More accurate:

$$L(\vec{a}, y) =$$
$$\begin{cases} -\log \frac{e^{z1}}{e^{z1}+e^{z2}+\cdots+e^{zN}} & y = 1 \\ -\log \frac{e^{z2}}{e^{z1}+e^{z2}+\cdots+e^{zN}} & y = 2 \\ \vdots & \\ -\log \frac{e^{zN}}{e^{z1}+e^{z2}+\cdots+e^{zN}} & y = N \end{cases}$$

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=20, activation='linear'),
                  ])
model.compile(loss=SparseCategoricalCrossEntropy(from_logits=True))
```

MNIST (more numerically accurate)

```
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)  <- not a1...a10
        f_x = tf.nn.softmax(logits)
```

Logistic Regression (more numerically accurate)

```
model   model = Sequential([
          Dense(units=25, activation='sigmoid'),
          Dense(units=15, activation='sigmoid'),
          Dense(units=1, activation='linear')
                  ])
        from tensorflow.keras.losses import
          BinaryCrossentropy
loss    model.compile(..., BinaryCrossentropy(from_logits=True))
        model.fit(X,Y,epochs=100)
fit     logit = model(X)
predict f_x = tf.nn.sigmoid(logit)
```

Classification with multiple outputs

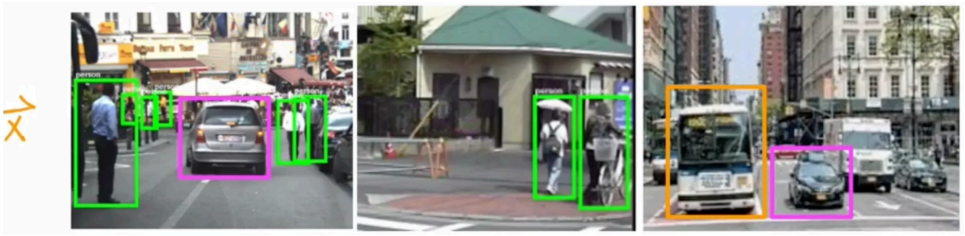
Multiclass Classification

- Multi-class classification involves a single output label that can take on one of several categories (e.g., handwritten digit classification).

Multi-Label Classification

- Multi-label classification allows for multiple labels to be associated with a single input (e.g., detecting cars, buses, and pedestrians in an image).

Multi-label classification

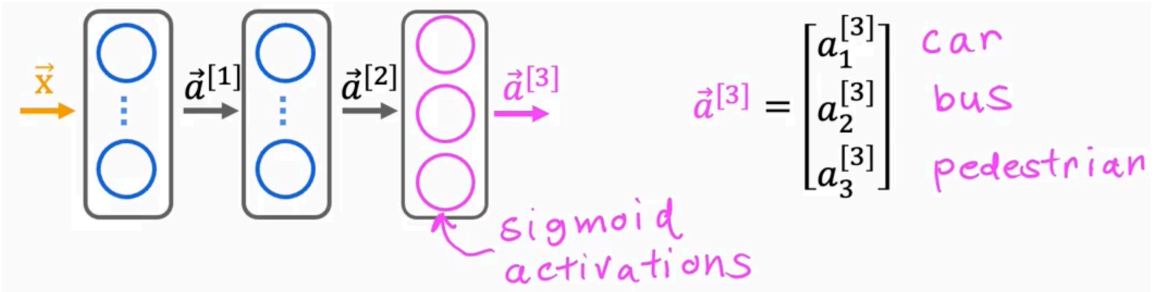


Is there a car?	Yes	No	Yes
Is there a bus?	No	No	Yes
Is there a pedestrian?	Yes	Yes	No
Math representation	$y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$	$y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

Building Neural Networks for Multi-Label Classification

- One approach is to create separate neural networks for each label (e.g., one for cars, one for buses, one for pedestrians).

- Alternatively, a single neural network can be trained to detect all labels simultaneously, using a vector of outputs and a **sigmoid activation function** for each output node.

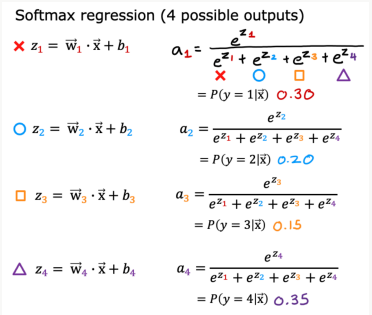


`SparseCategorialCrossentropy` or `CategoricalCrossEntropy`

`Tensorflow` has two potential formats for target values and the selection of the loss defines which is expected.

- `SparseCategorialCrossentropy:` expects the target to be an integer corresponding to the index. For example, if there are 10 potential target values, y would be between 0 and 9.
- `CategoricalCrossEntropy:` Expects the target value of an example to be one-hot encoded where the value at the target index is 1 while the other $N-1$ entries are zero. An example with 10 potential target values, where the target is 2 would be `[0,0,1,0,0,0,0,0,0,0]`.

?



QUES: For a multiclass classification task that has 4 possible outputs, the sum of all the activations adds up to 1. For a multiclass classification task that has 3 possible outputs, the sum of all the activations should add up to ....

- More than 1
- 1 👍🏻👍🏻👍🏻
- It will vary, depending on the input x.
- Less than 1

❓ **QUES:** For multiclass classification, the cross entropy loss is used for training the model. If there are 4 possible classes for the output, and for a particular training example, the true class of the example is class 3 ($y = 3$), then what does the cross entropy loss simplify to? [Hint: This loss should get smaller when a_3 gets larger.]

- $\frac{z_3}{z_1+z_2+z_3+z_4}$
- $\frac{-\log a_1 - \log a_2 - \log a_3 - \log a_4}{4}$
- $-\log(a_3)$ 👍👍
- z_3

Explain: When the true label is 3, then the cross entropy loss for that training example is just the negative of the log of the activation for the third neuron of the softmax. All other terms of the cross entropy loss equation $-\log(a1), -\log(a2), and -\log(a4)$ are ignored

❓ **QUES:** For multiclass classification, the recommended way to implement softmax regression is to set `from_logits=True` in the loss function, and also to define the model's output layer with...

- a 'linear' activation 👍👍👍
- a 'softmax' activation

Explain: Yes! Set the output as linear, because the loss function handles the calculation of the softmax with a more numerically stable method.