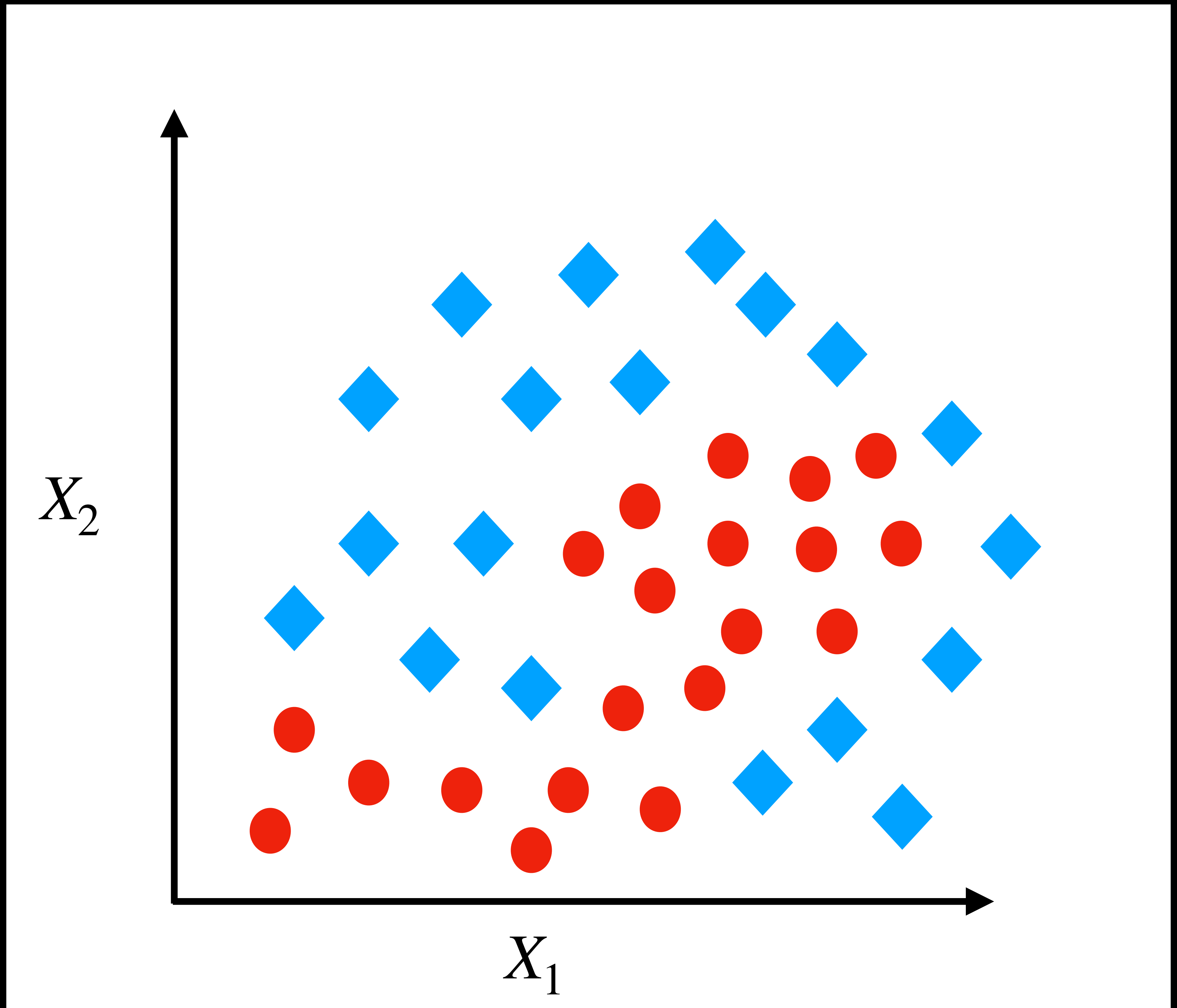
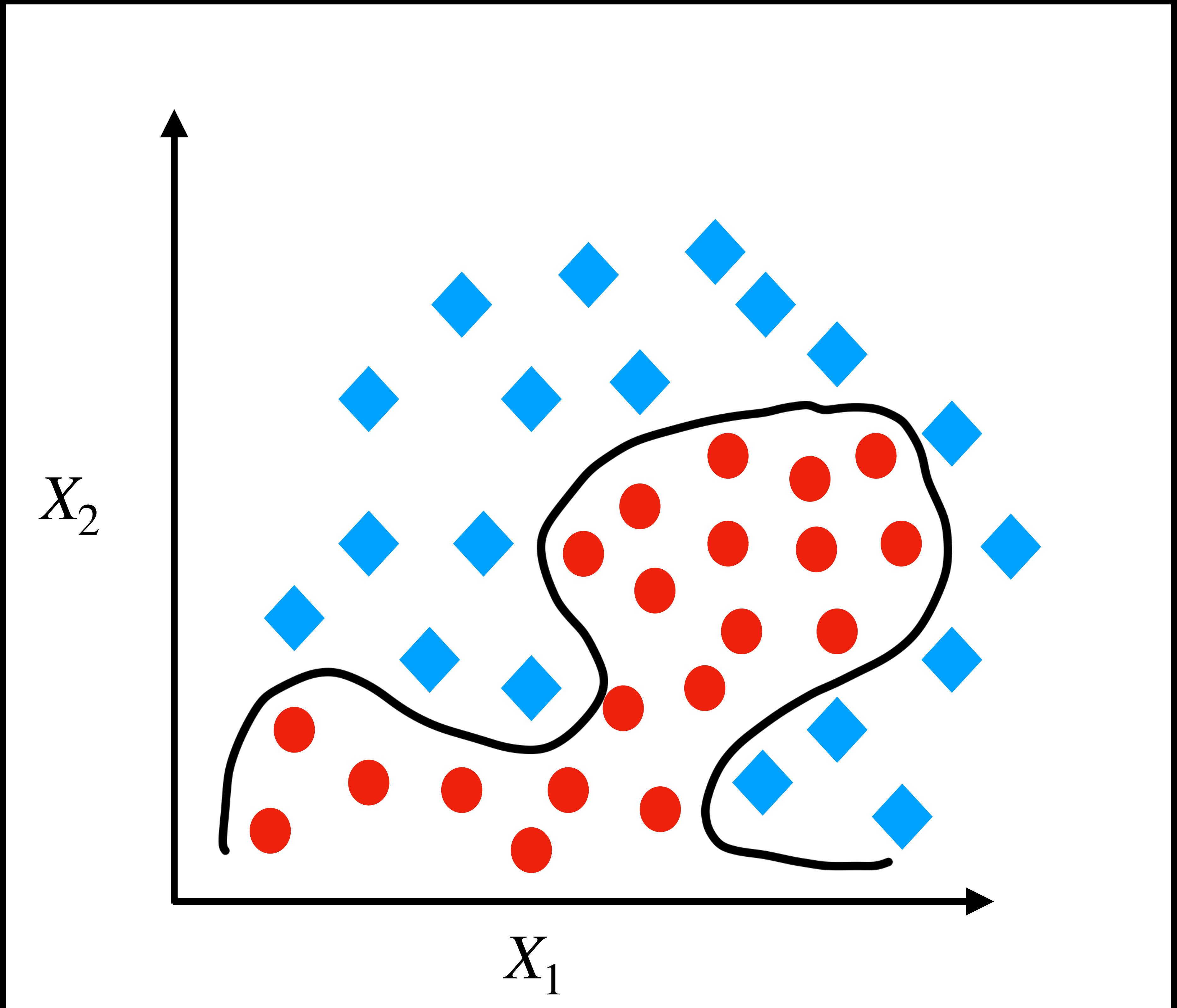
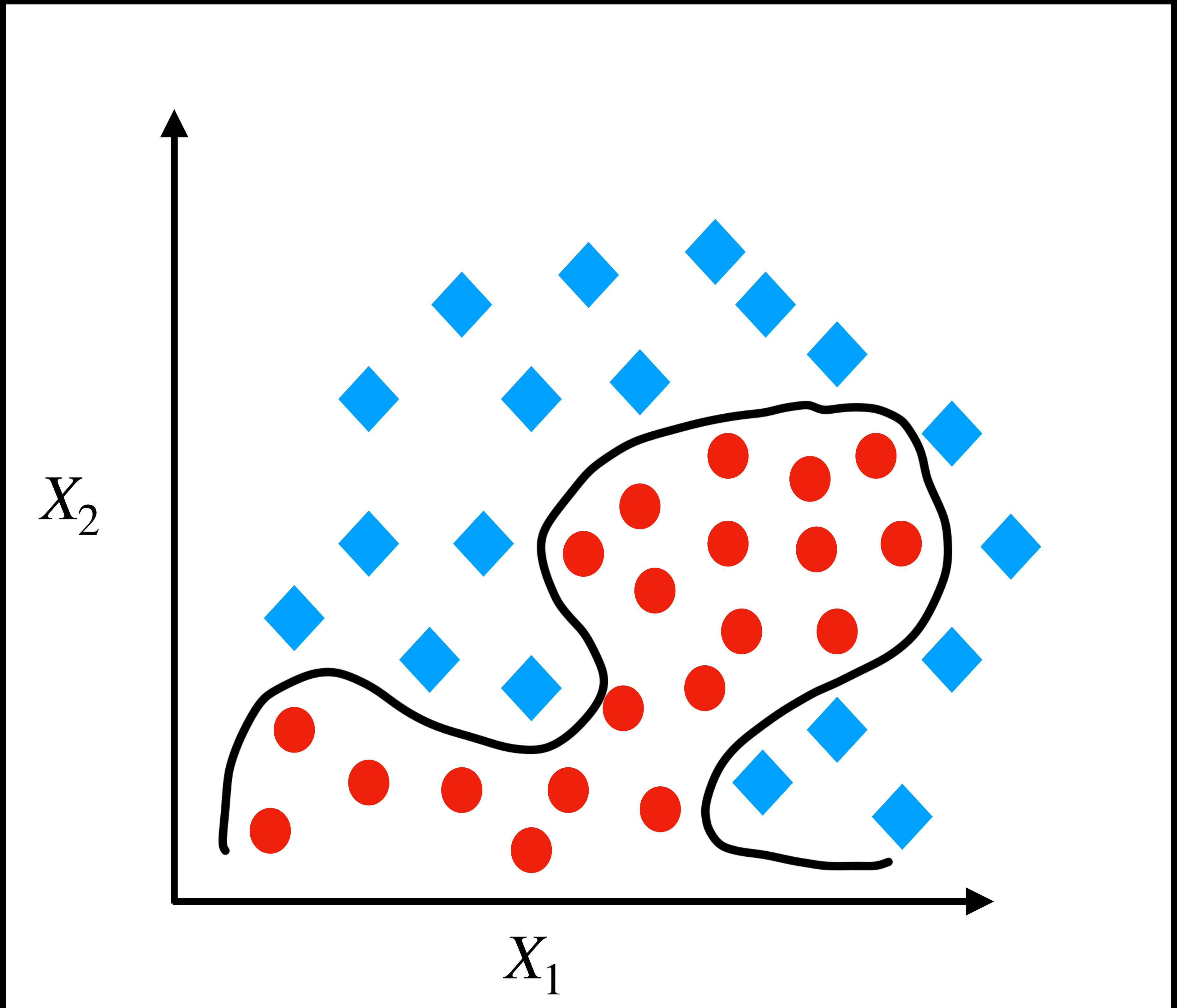
Neural Networks



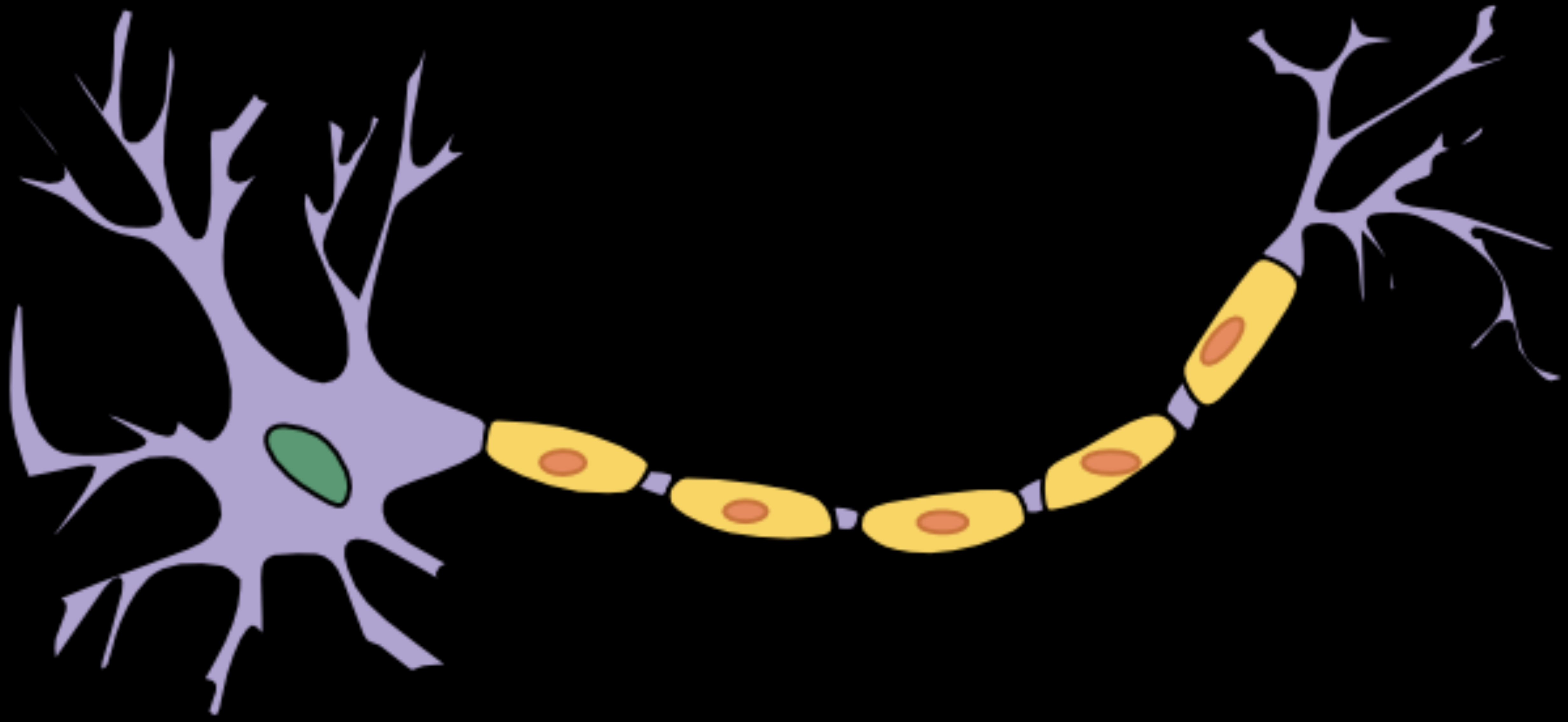
X_2
X_1

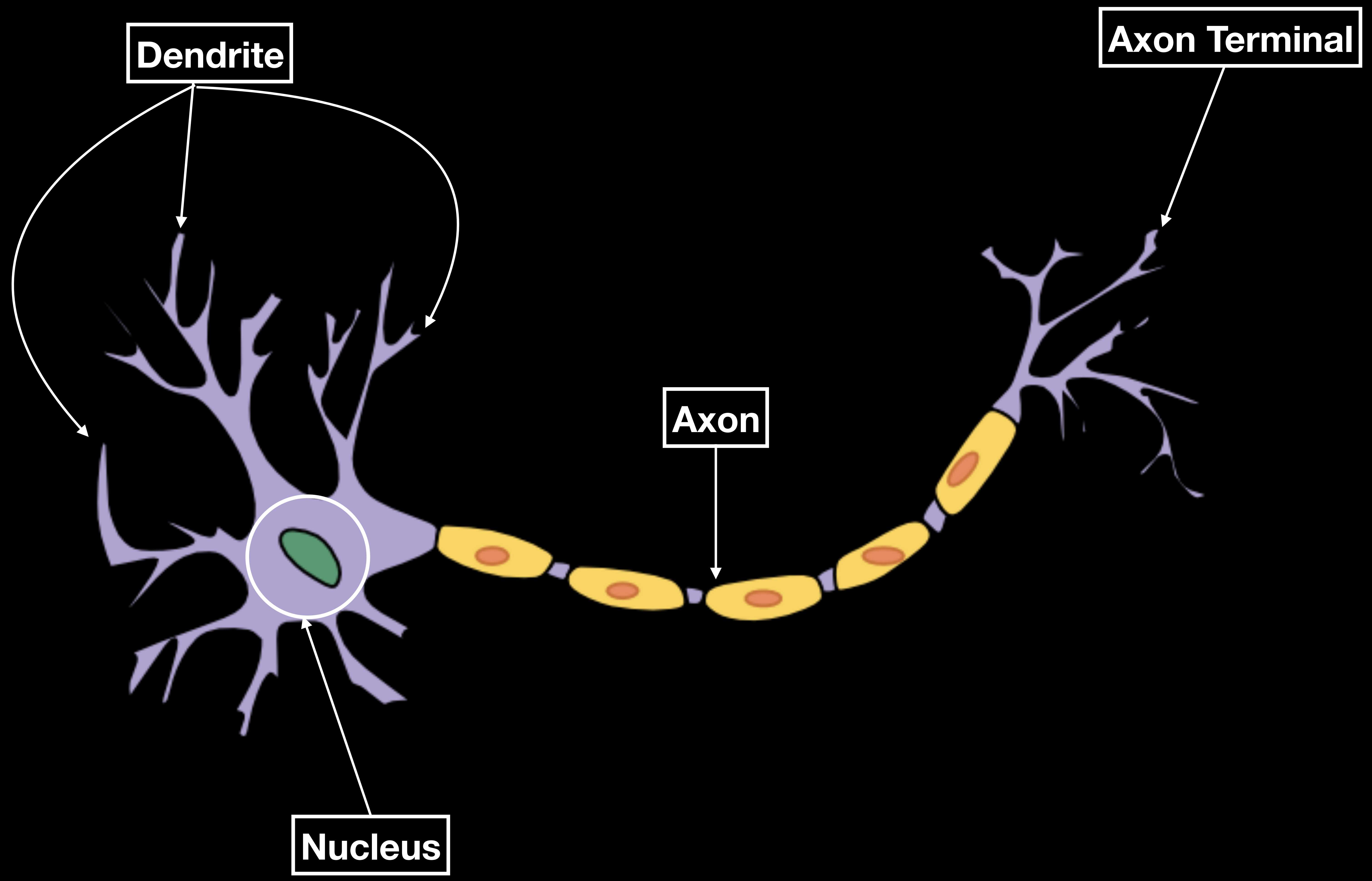


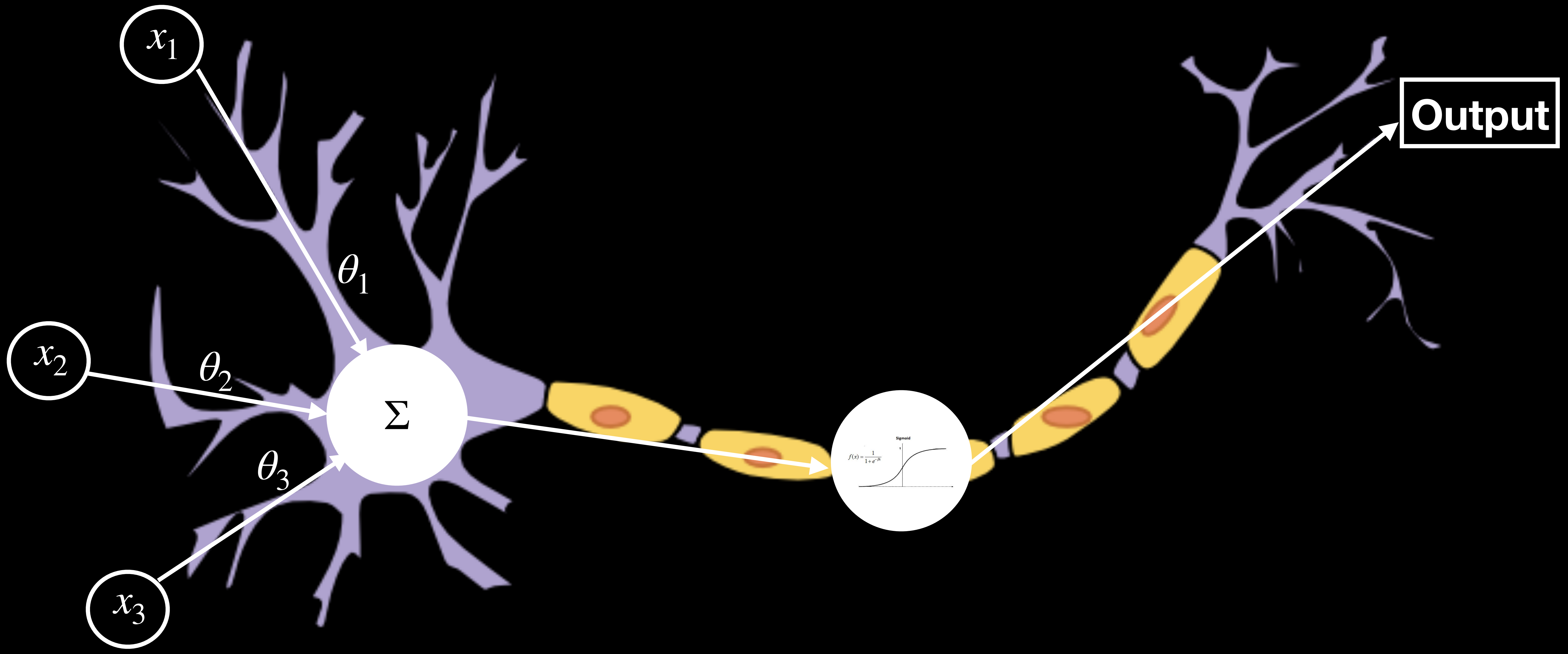
X_2
X_1

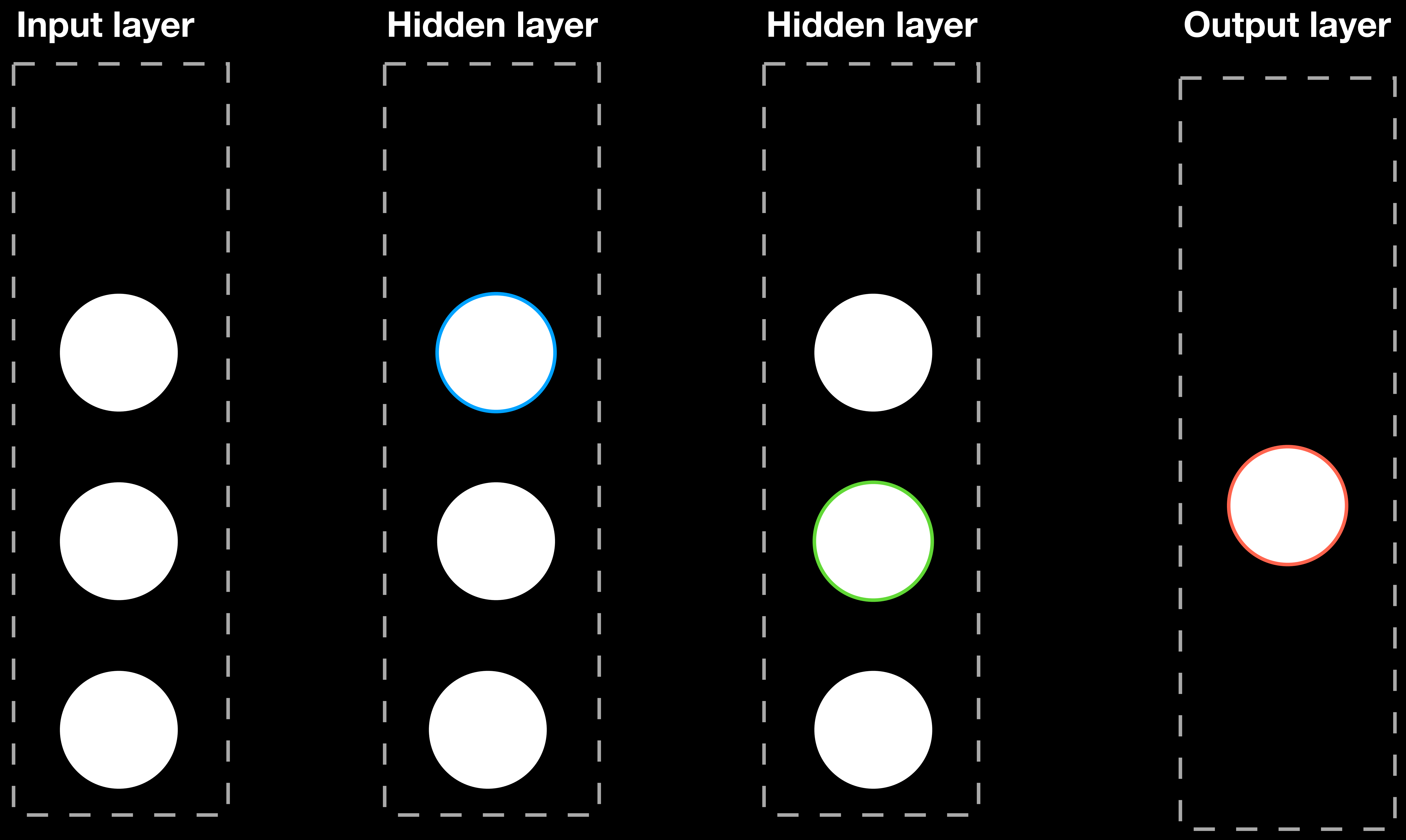


$$y = aX_1 + bX_2 + cX_1X_2 + fX_1^3X_2^4 + \ldots$$

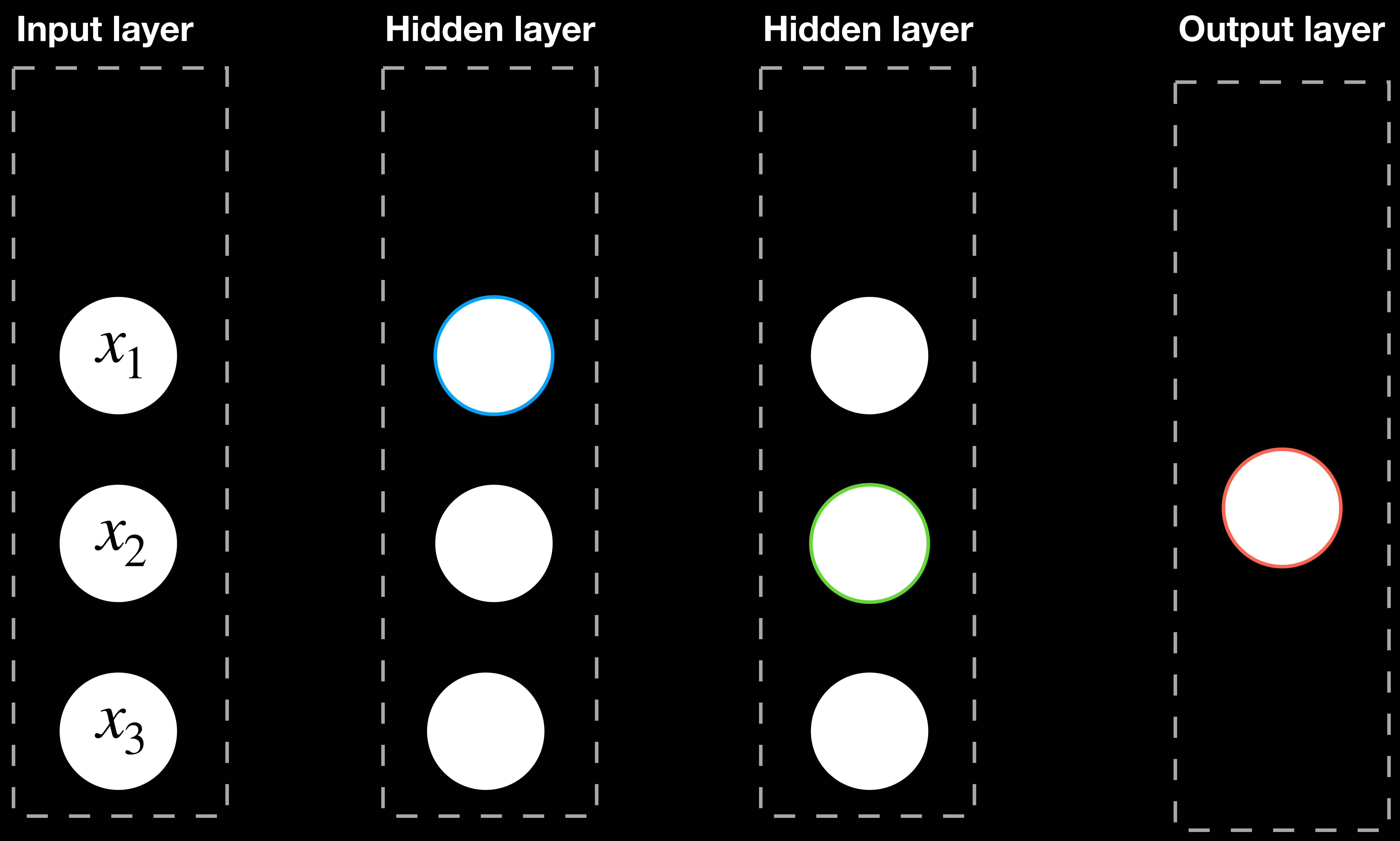



Dendrite
Axon Terminal
Axon
Nucleus

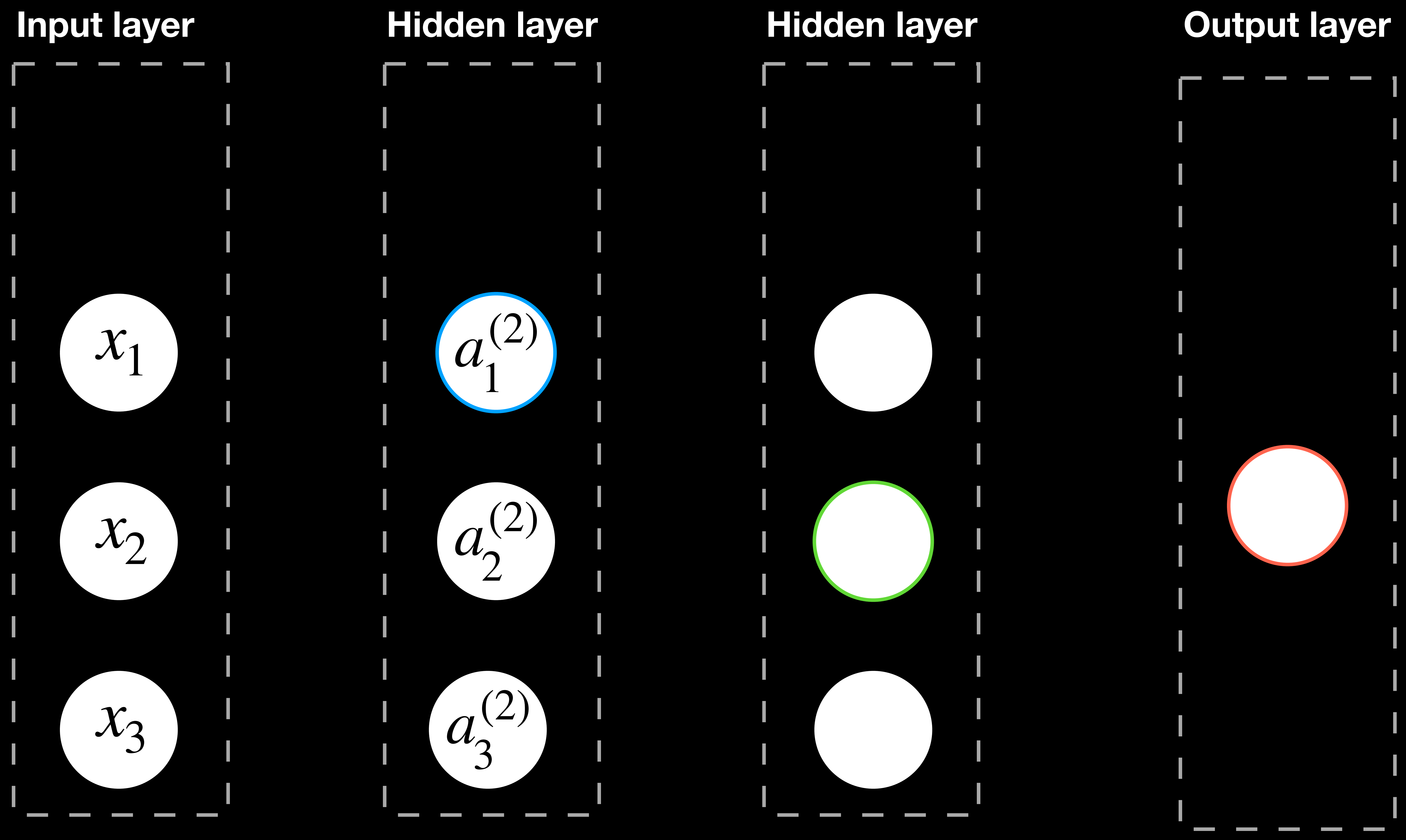

x_1
x_2
x_3
θ_1
θ_2
θ_3
Σ
Output



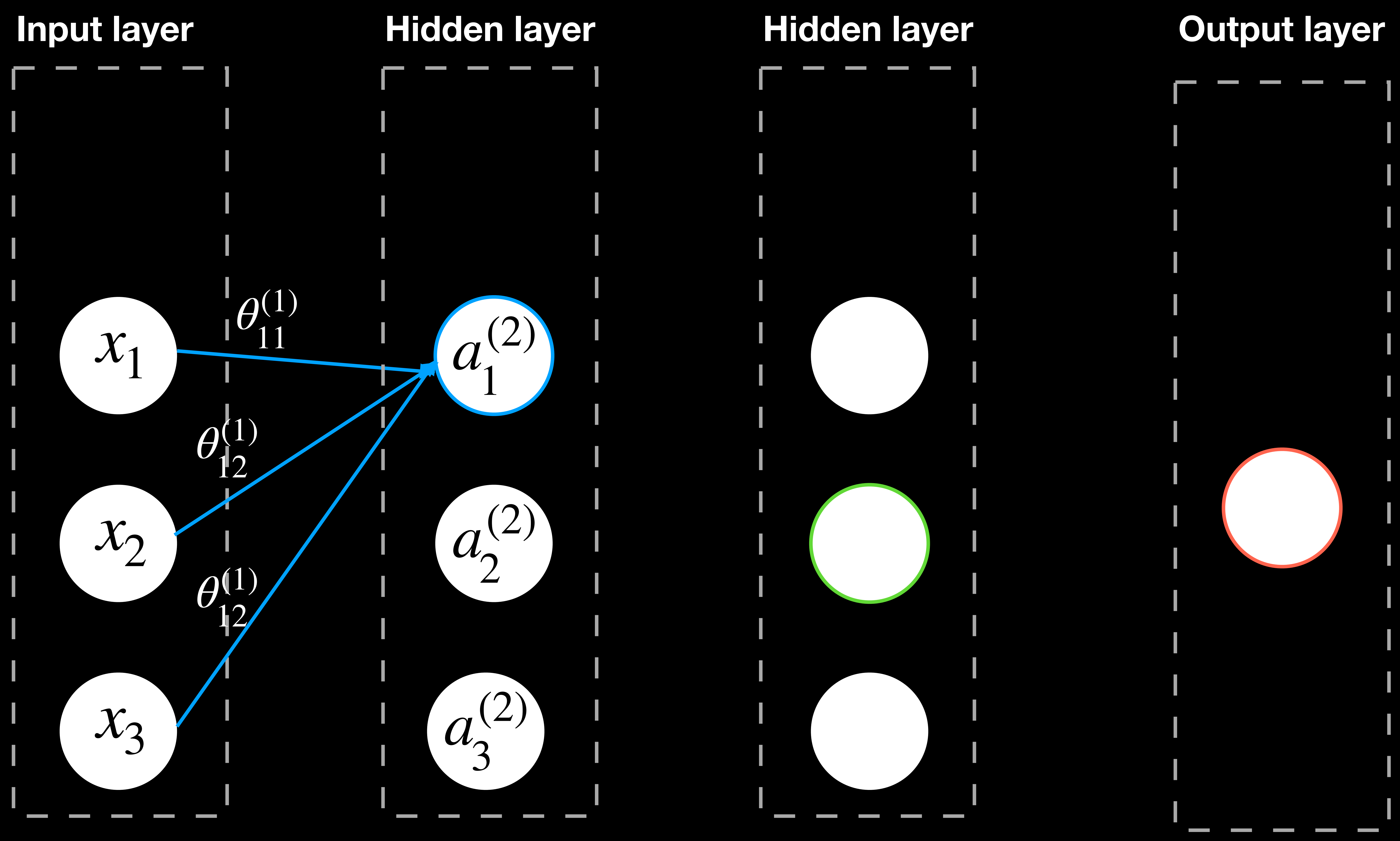
$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



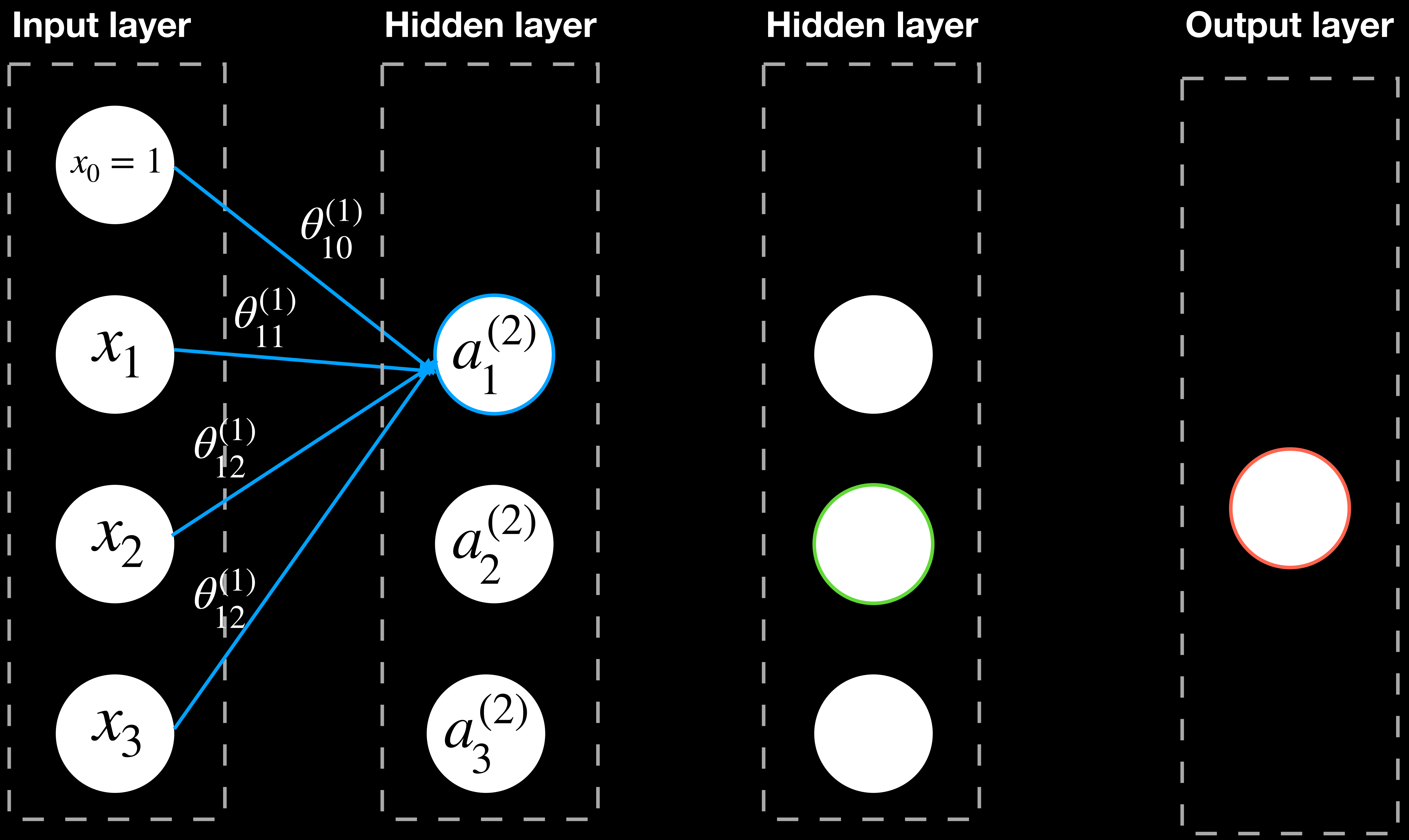
$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



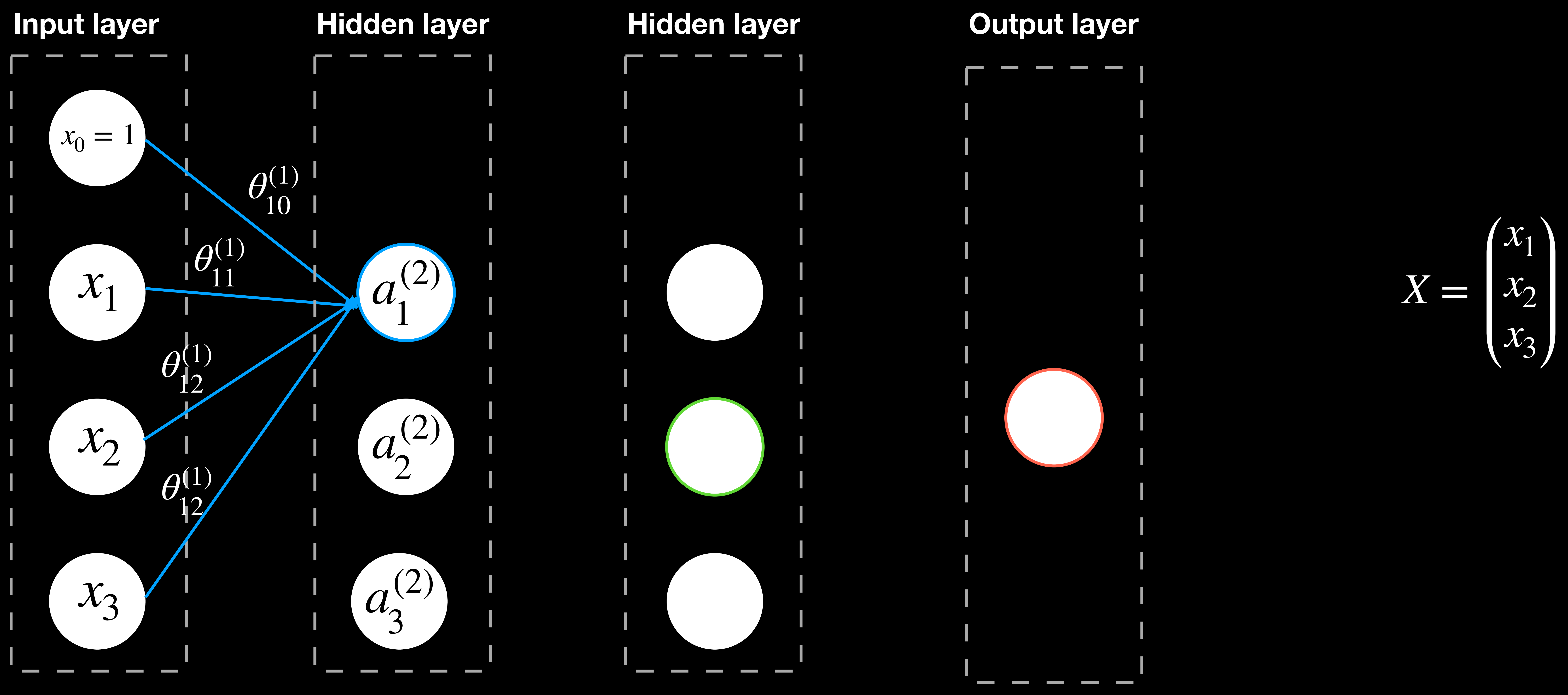
$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



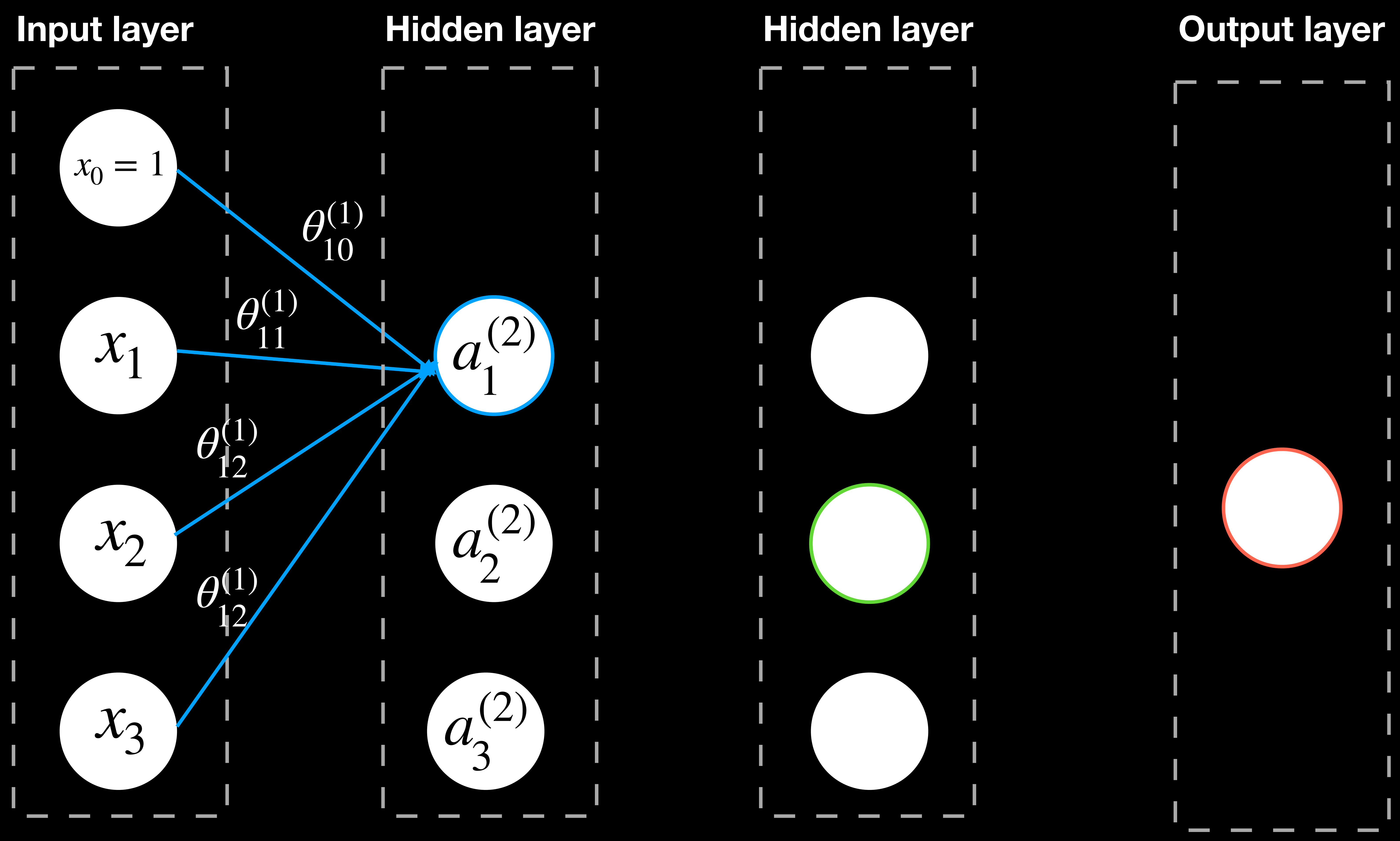
$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

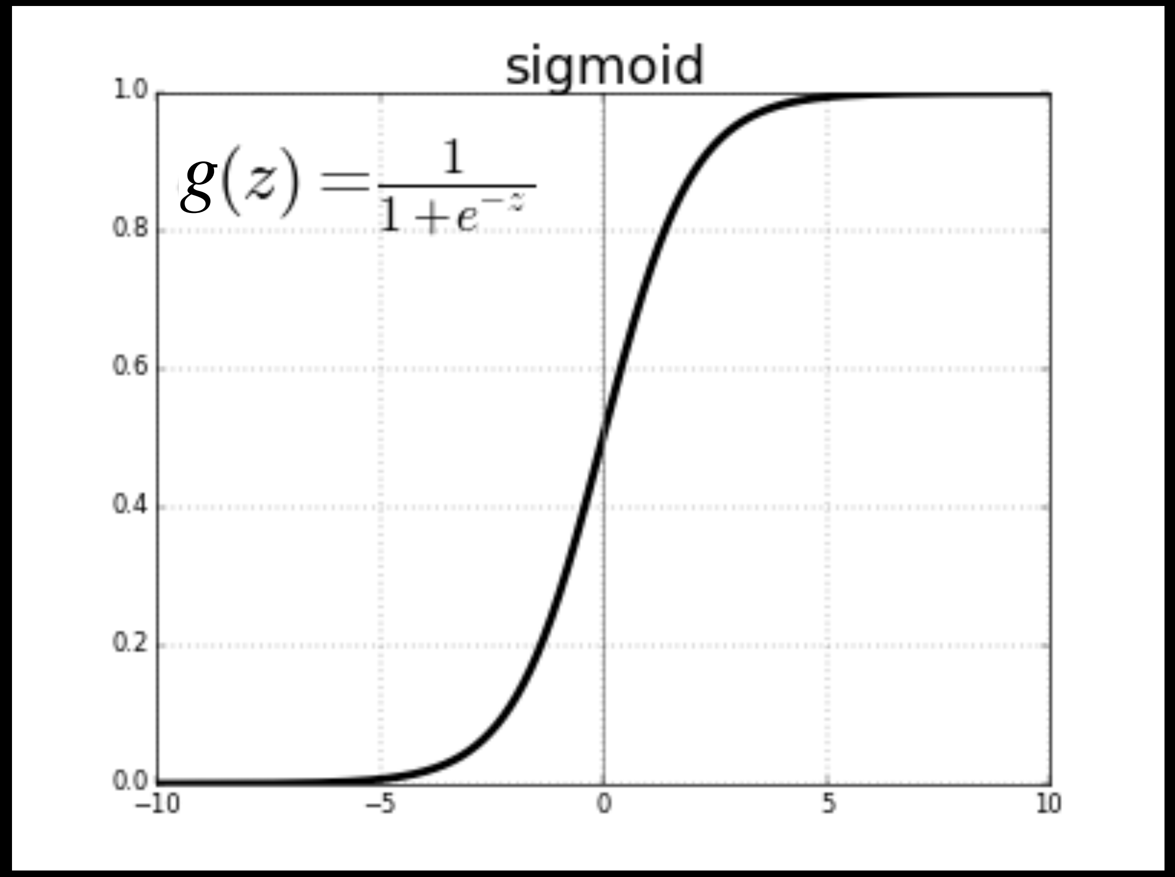


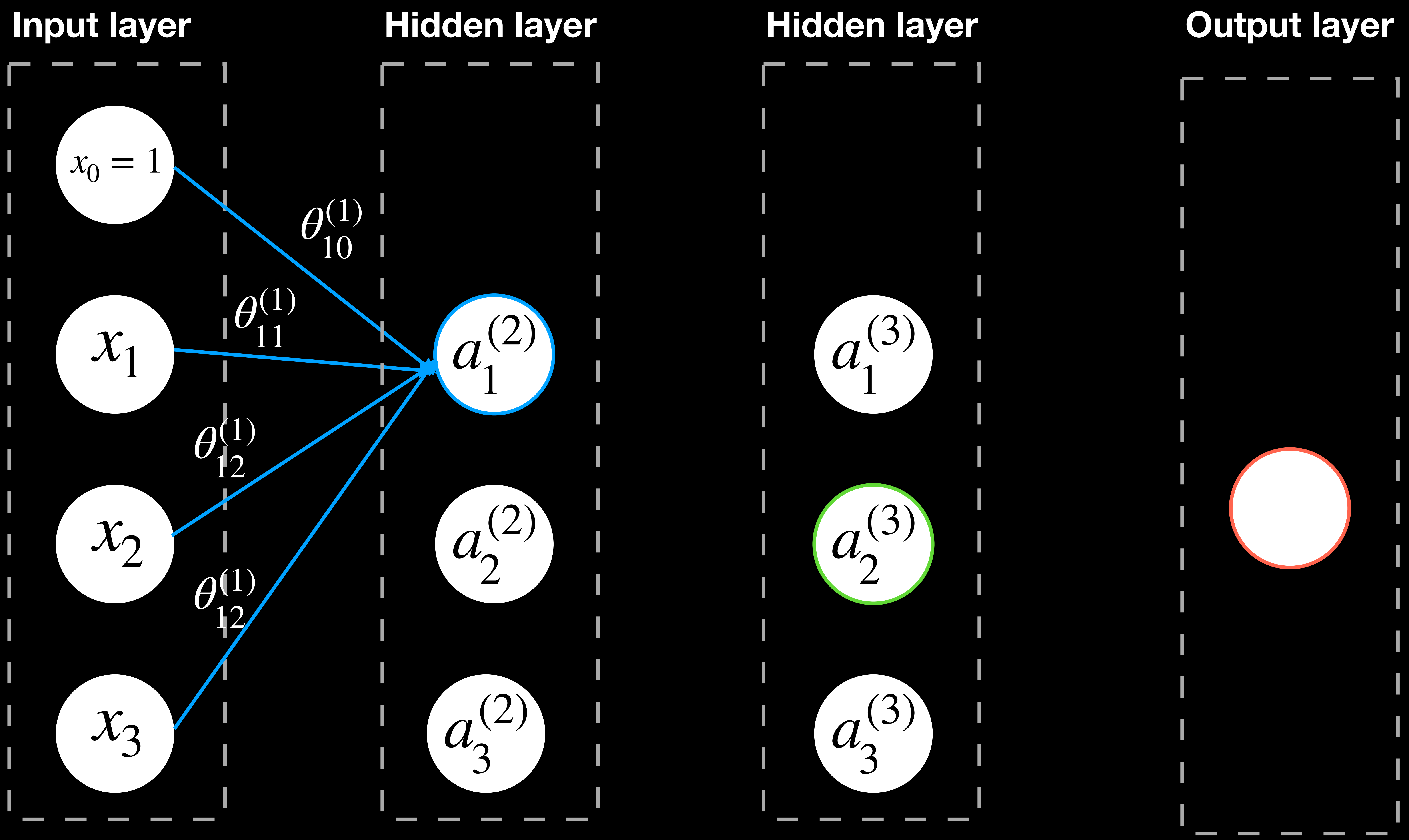
$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$



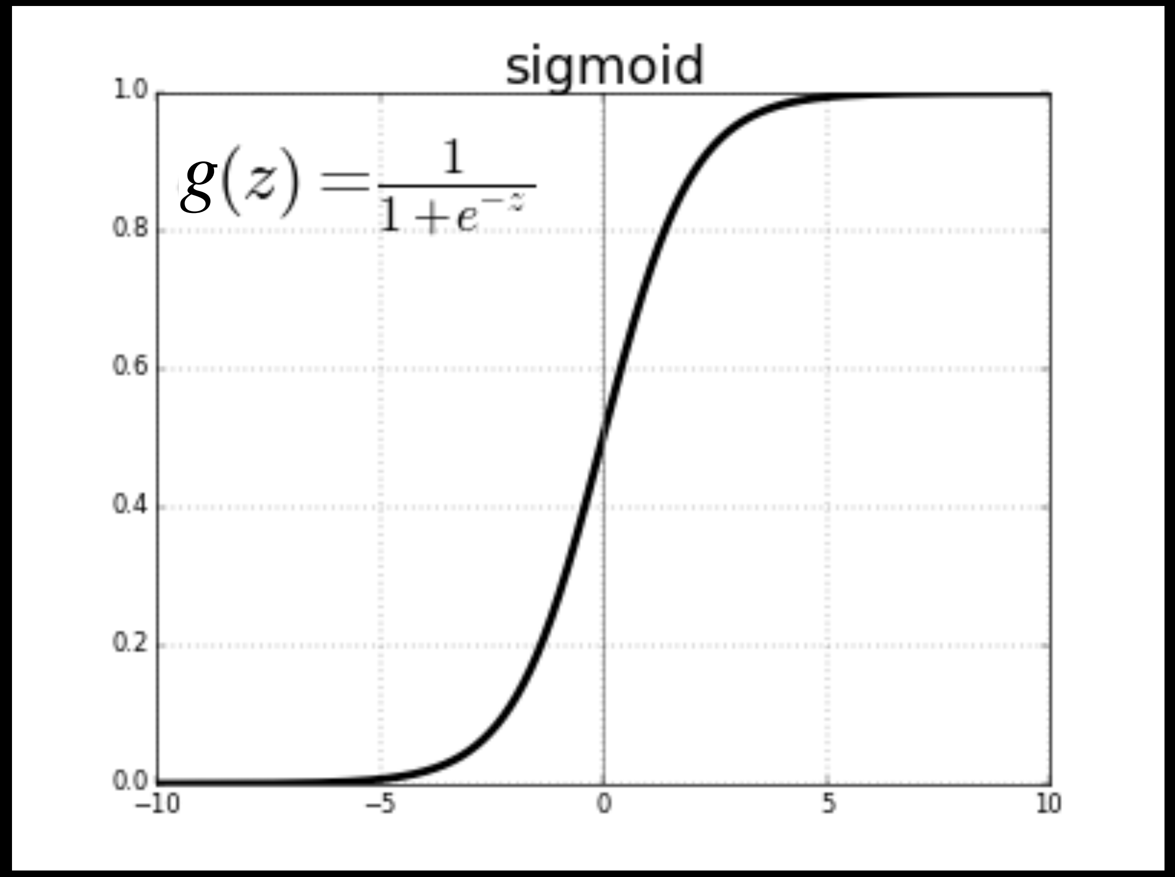
$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\boxed{a^{(2)}_1 = g\left(\theta^{(1)}_{10}x_0 + \theta^{(1)}_{11}x_1 + \theta^{(1)}_{12}x_2 + \theta^{(1)}_{13}x_3\right) = g\left(z^{(2)}_1\right)}$$

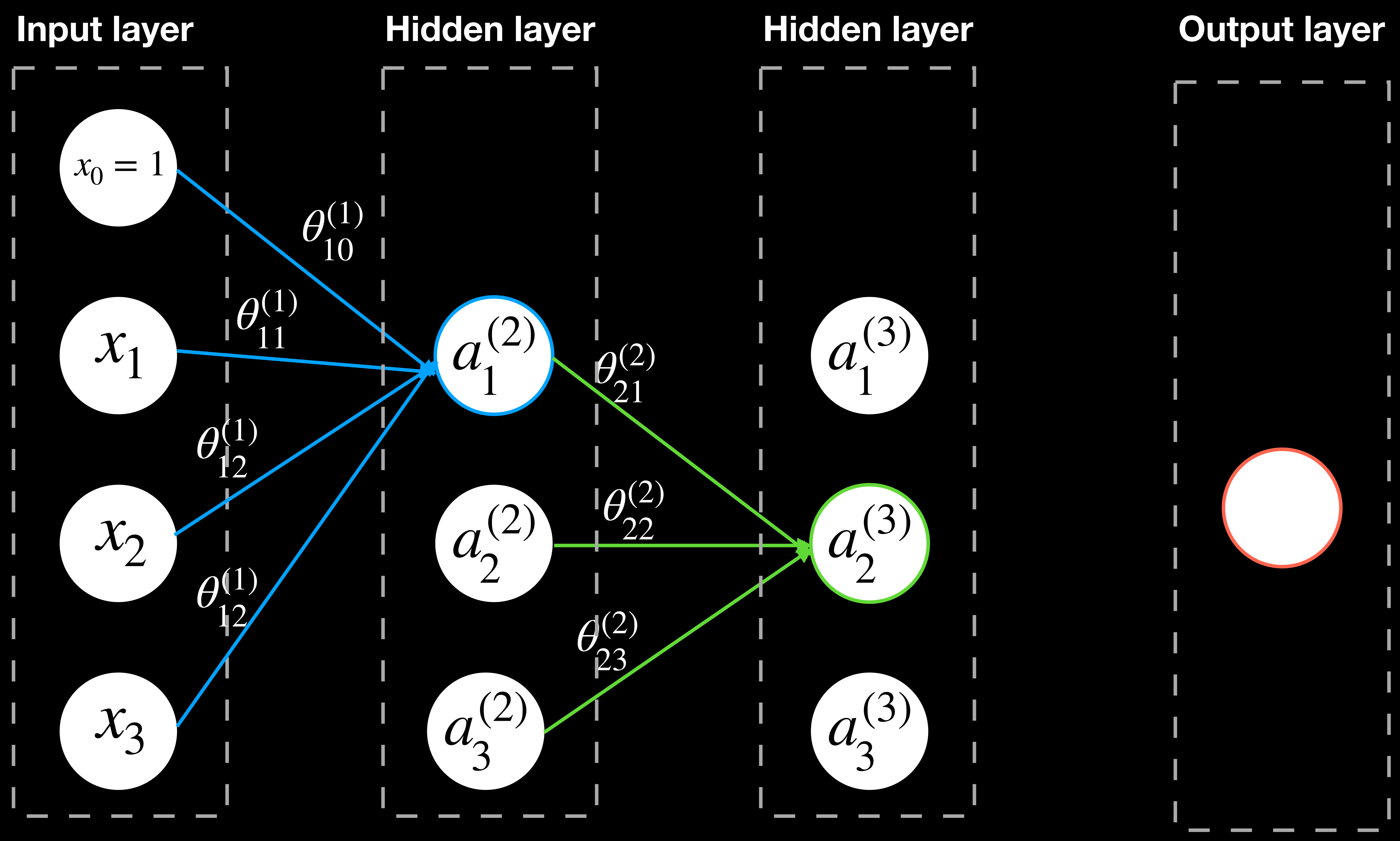




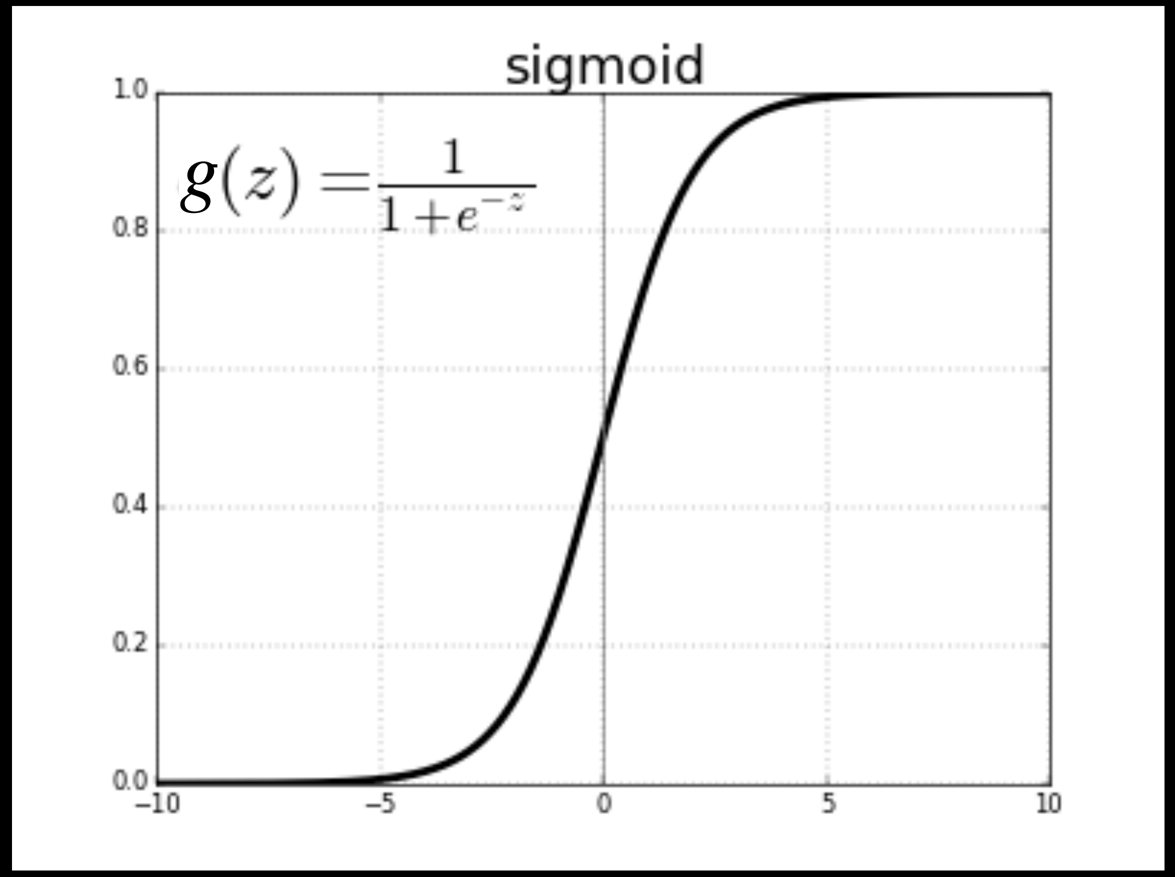
$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



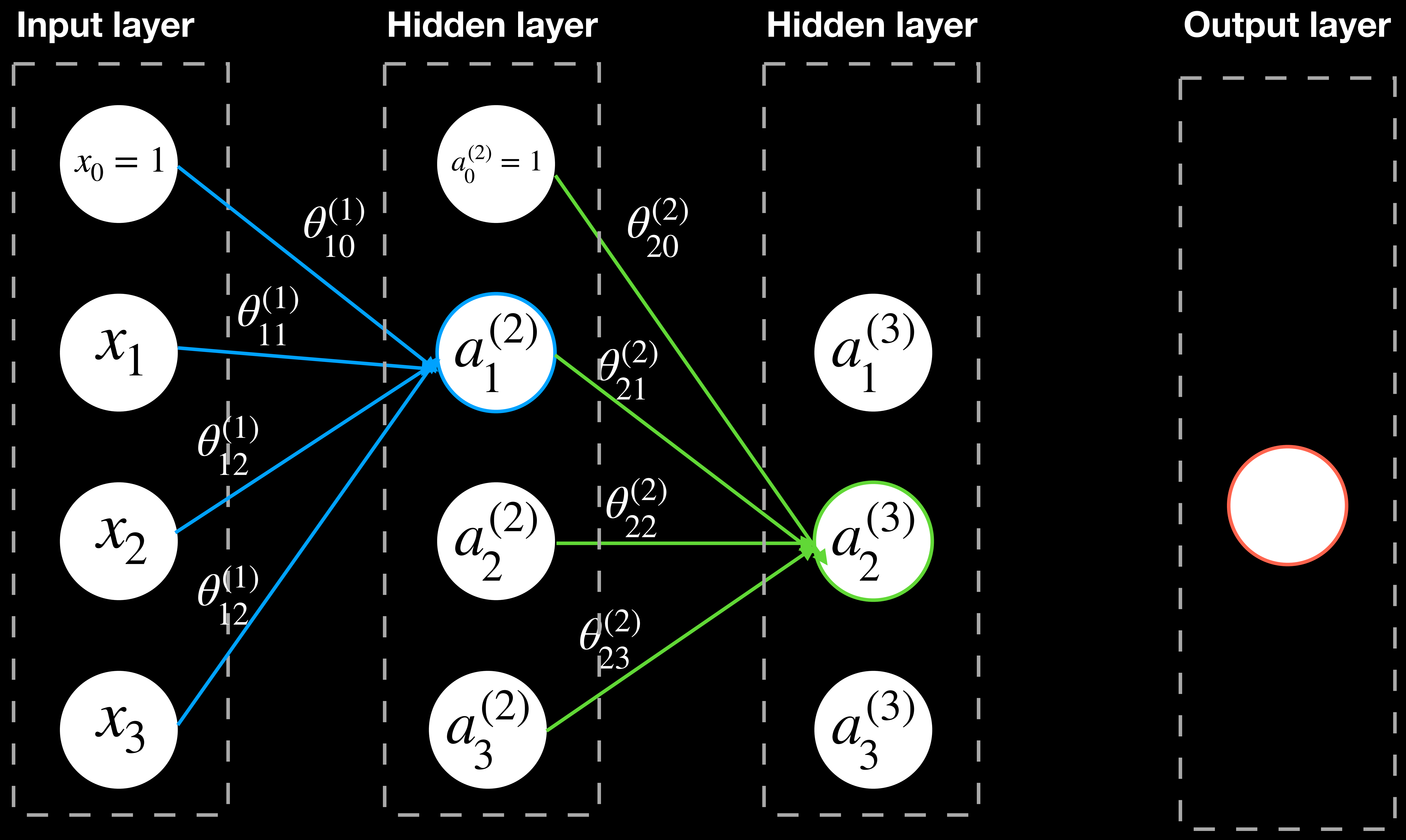
$$\boxed{a^{(2)}_1 = g\left(\theta^{(1)}_{10}x_0 + \theta^{(1)}_{11}x_1 + \theta^{(1)}_{12}x_2 + \theta^{(1)}_{13}x_3\right) = g\left(z^{(2)}_1\right)}$$



$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

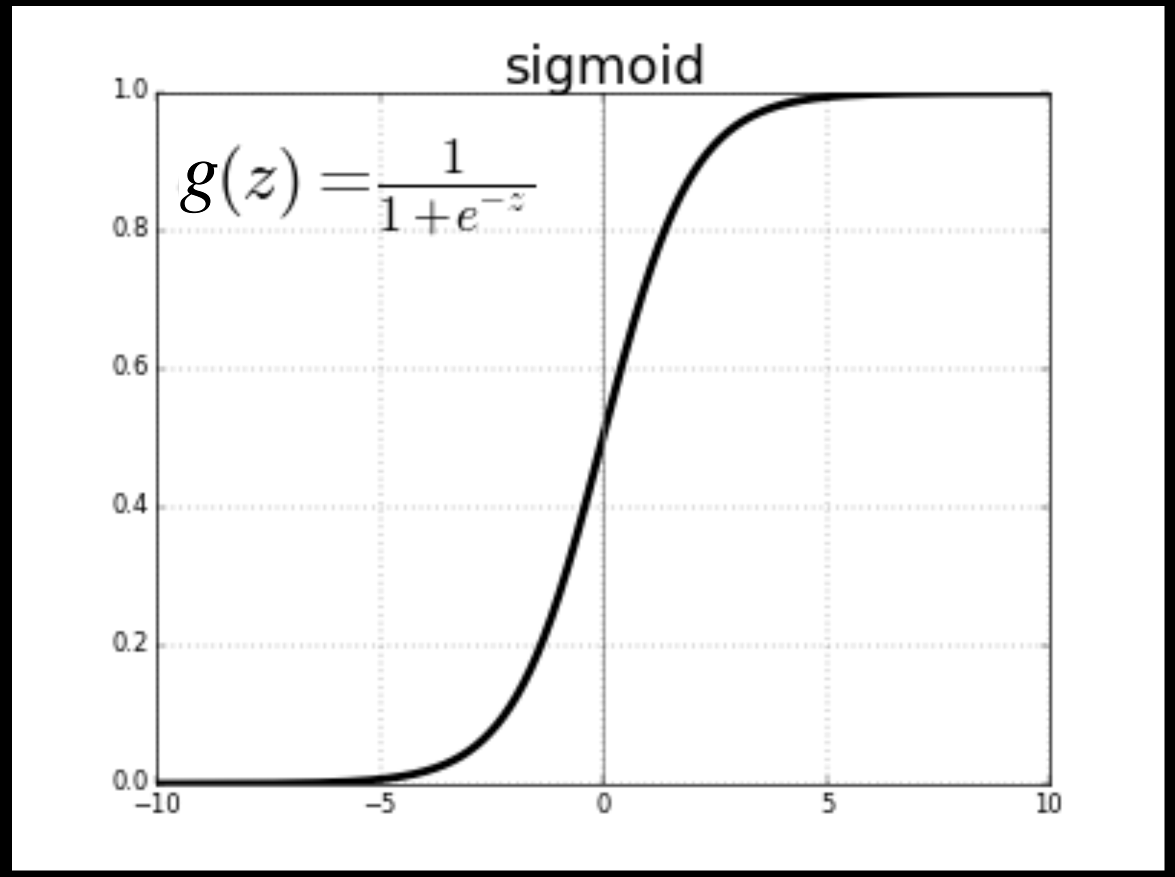


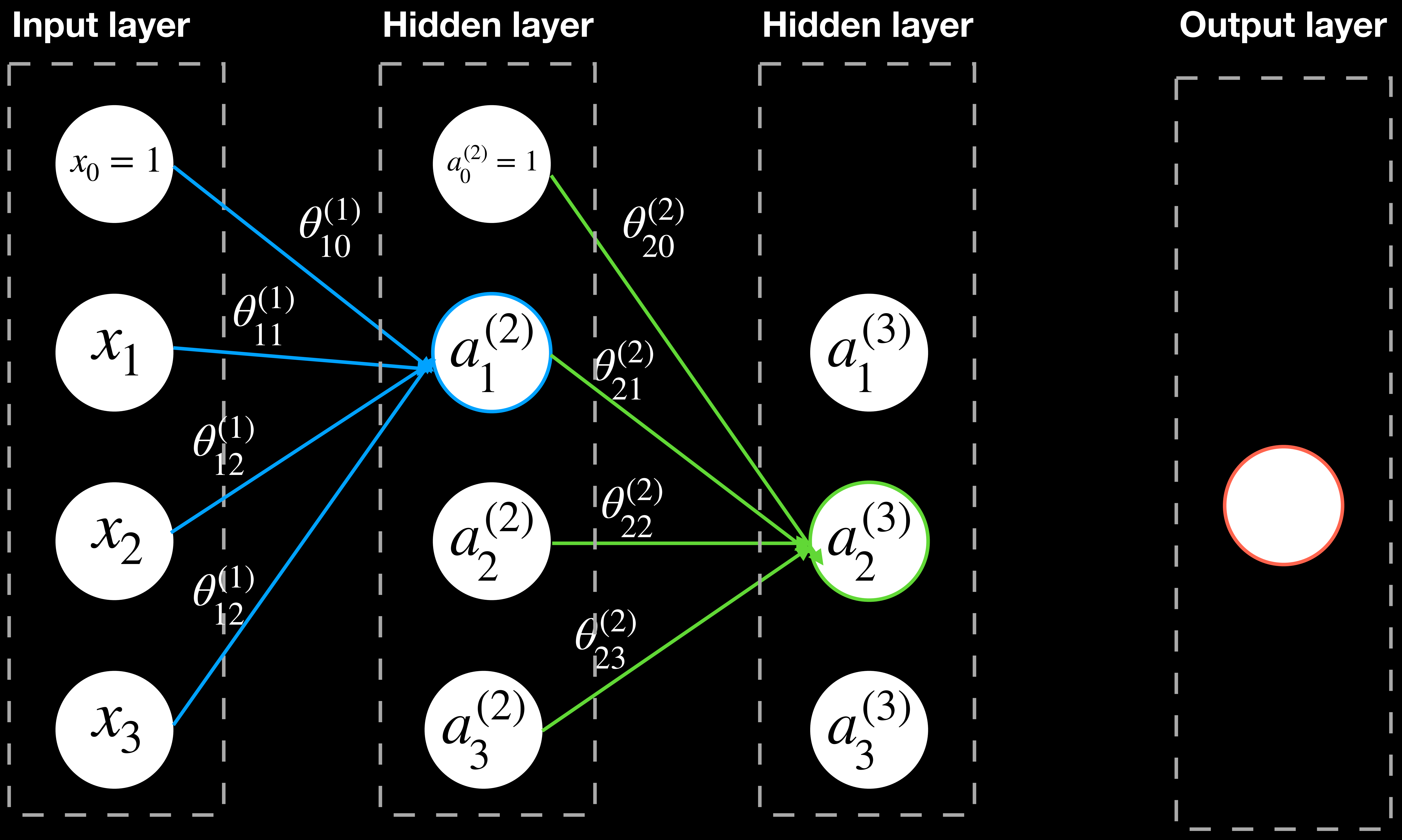
$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$



$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$

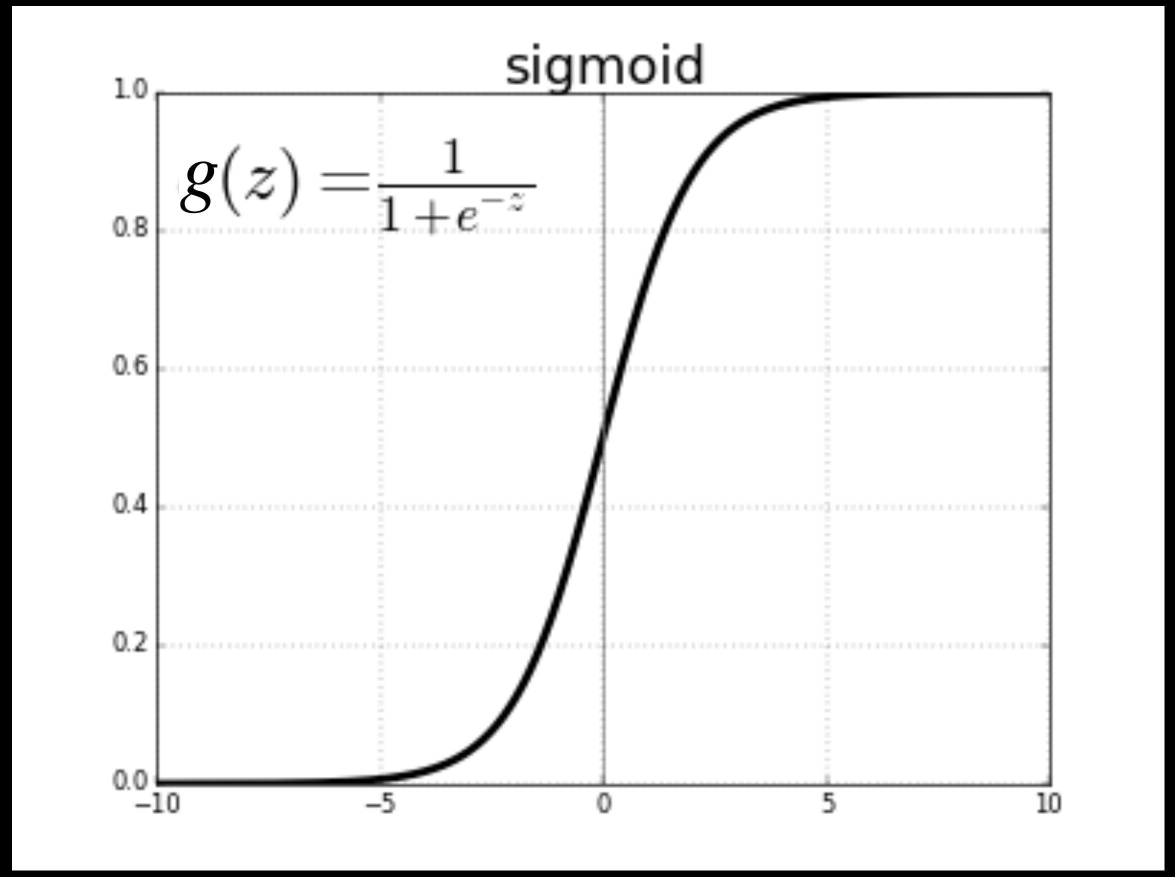


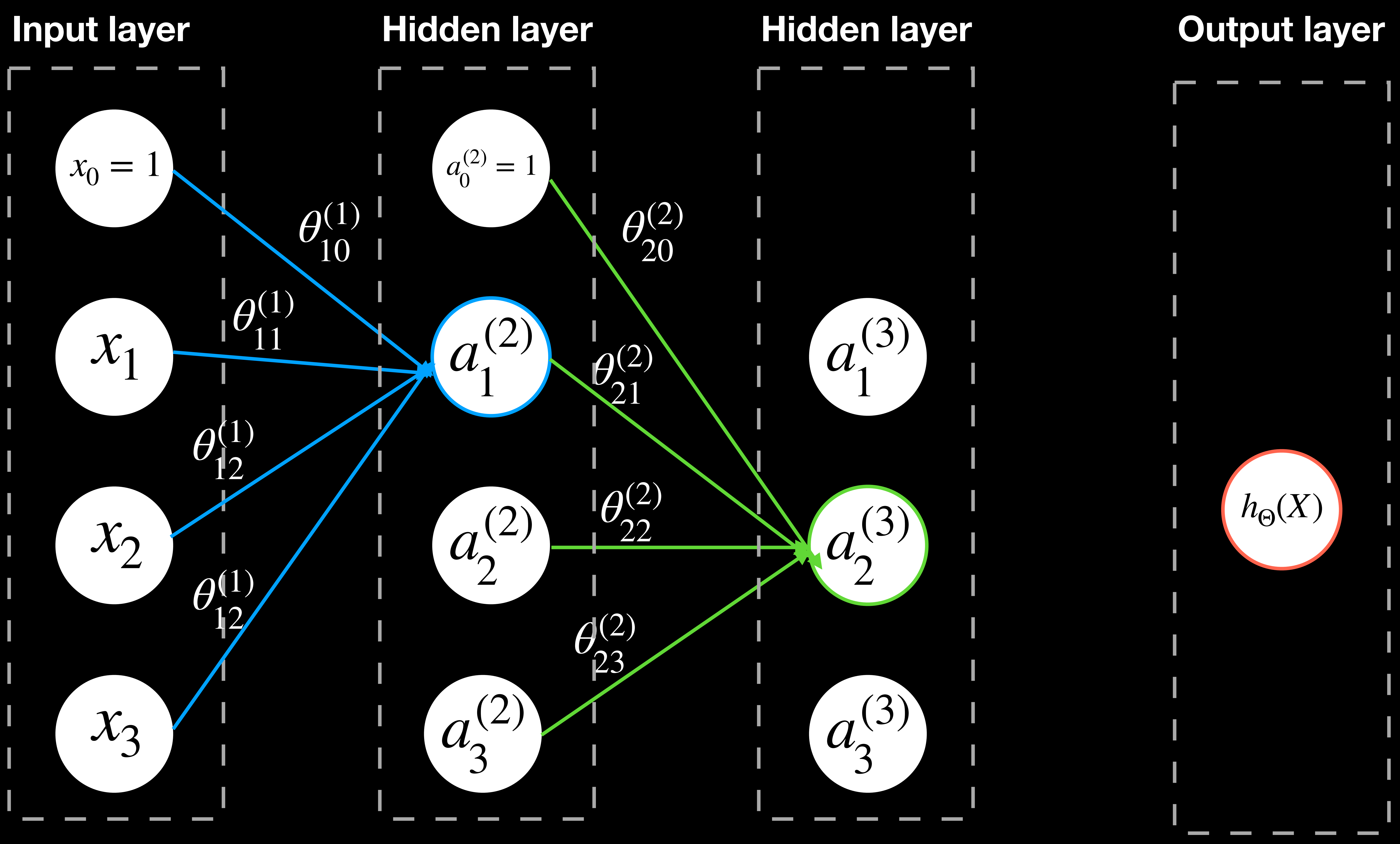


$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

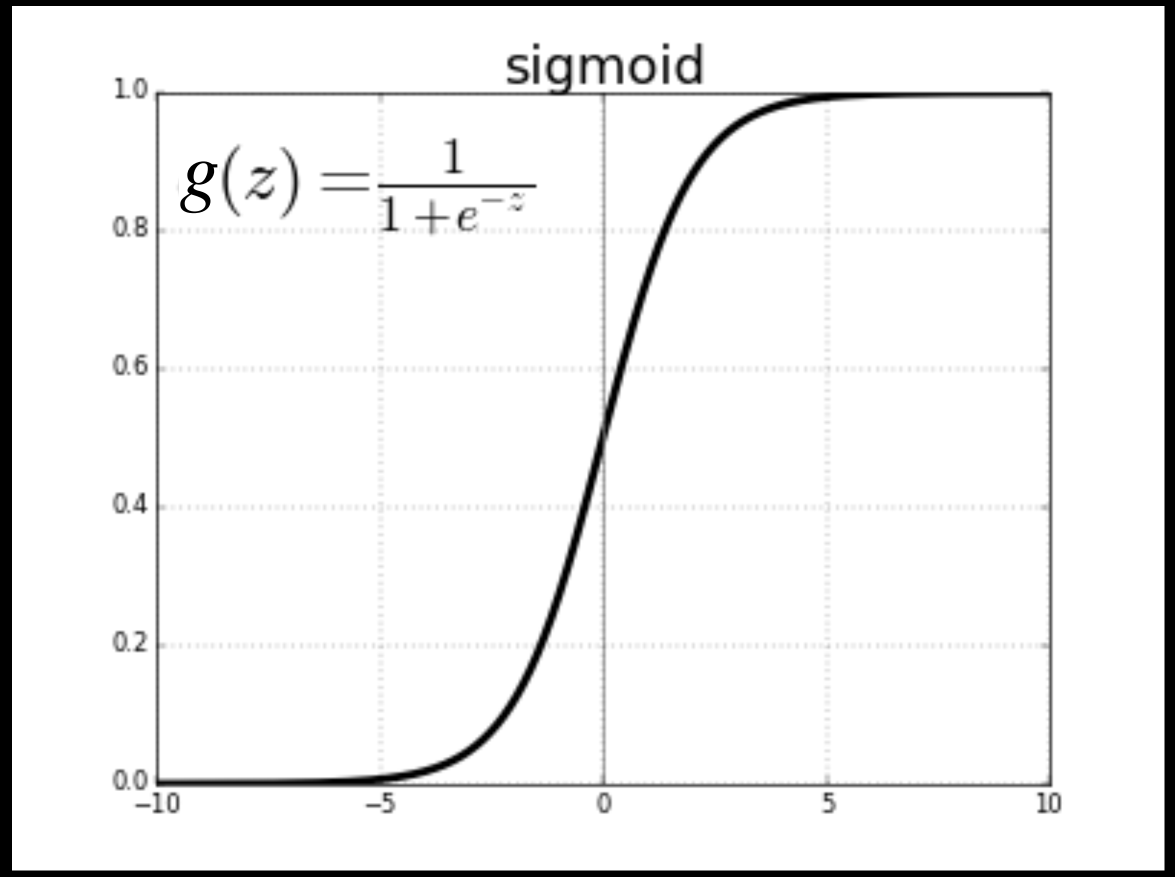
$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$

$$\boxed{a_2^{(3)} = g\left(\theta_{20}^{(2)}a_0^{(2)} + \theta_{21}^{(2)}a_1^{(2)} + \theta_{22}^{(2)}a_2^{(2)} + \theta_{23}^{(2)}a_3^{(2)}\right) = g\left(z_2^{(3)}\right)}$$



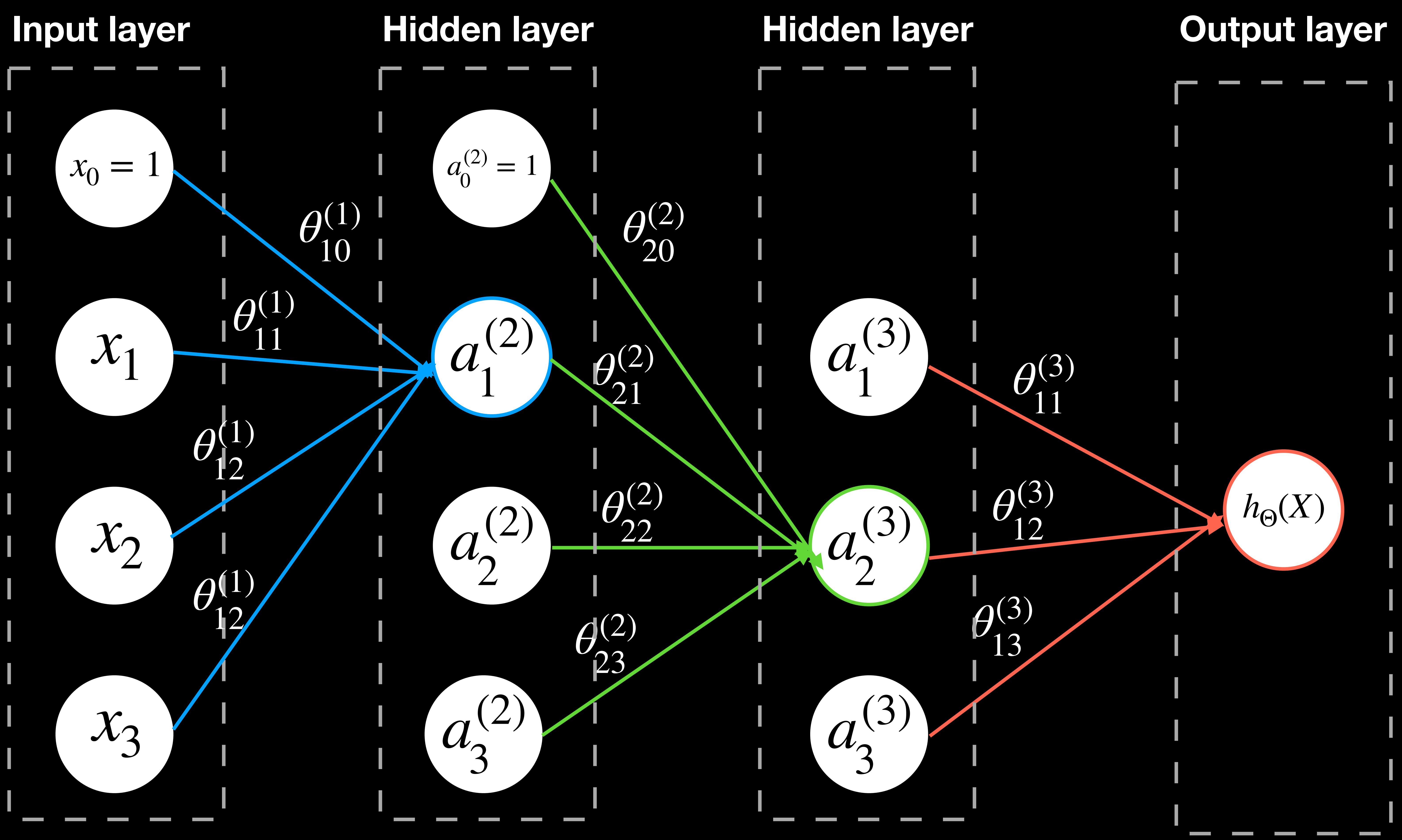


$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

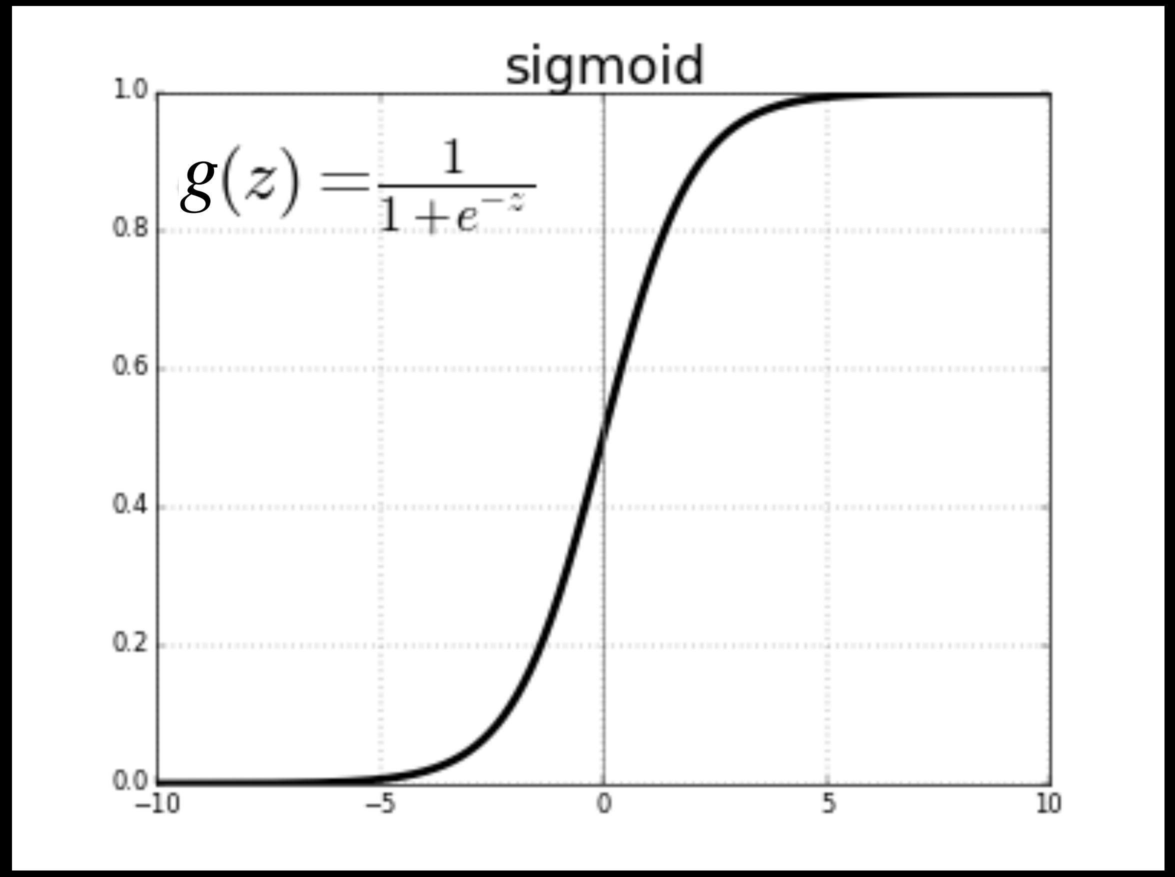


$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$

$$\boxed{a_2^{(3)} = g\left(\theta_{20}^{(2)}a_0^{(2)} + \theta_{21}^{(2)}a_1^{(2)} + \theta_{22}^{(2)}a_2^{(2)} + \theta_{23}^{(2)}a_3^{(2)}\right) = g\left(z_2^{(3)}\right)}$$

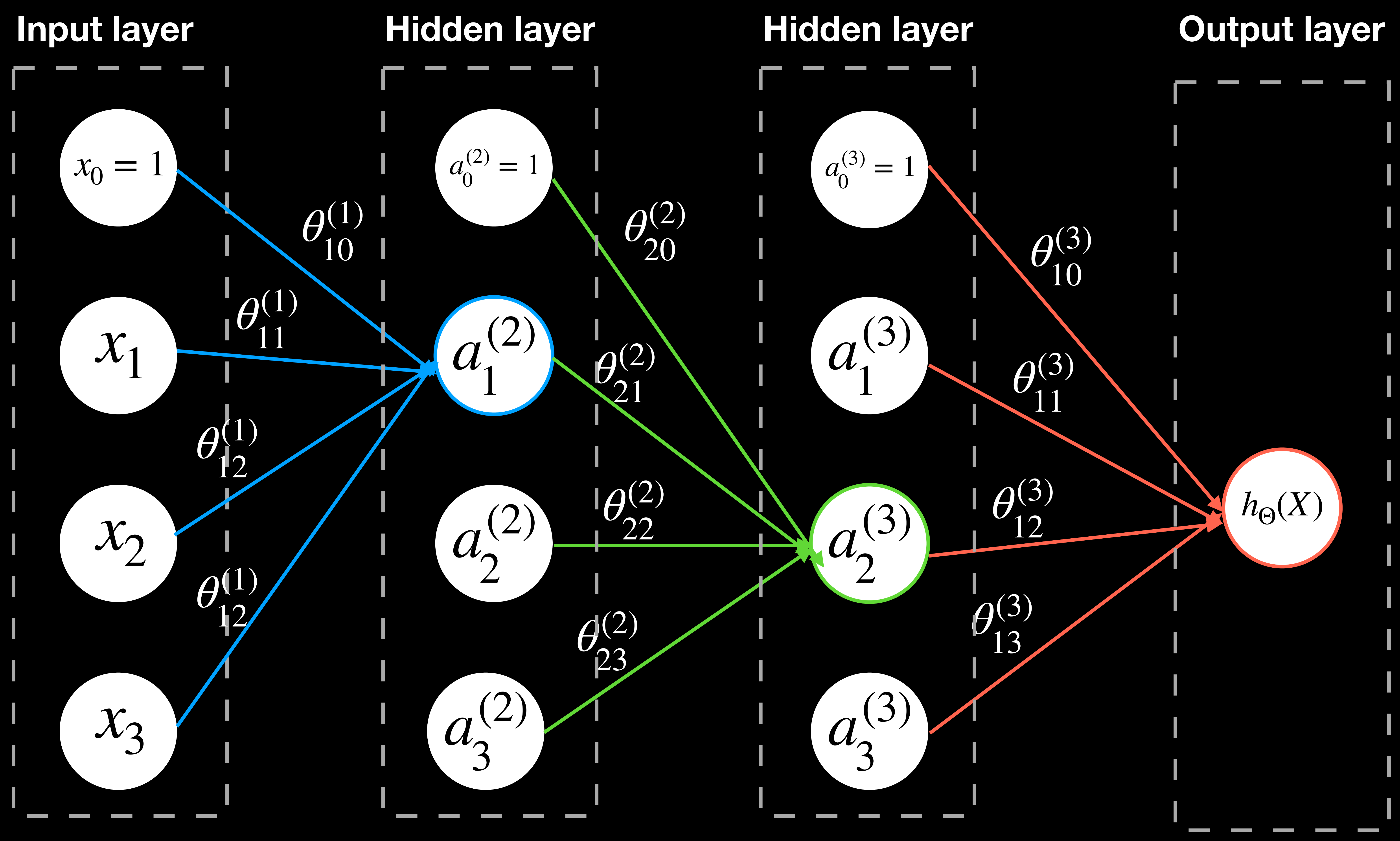


$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

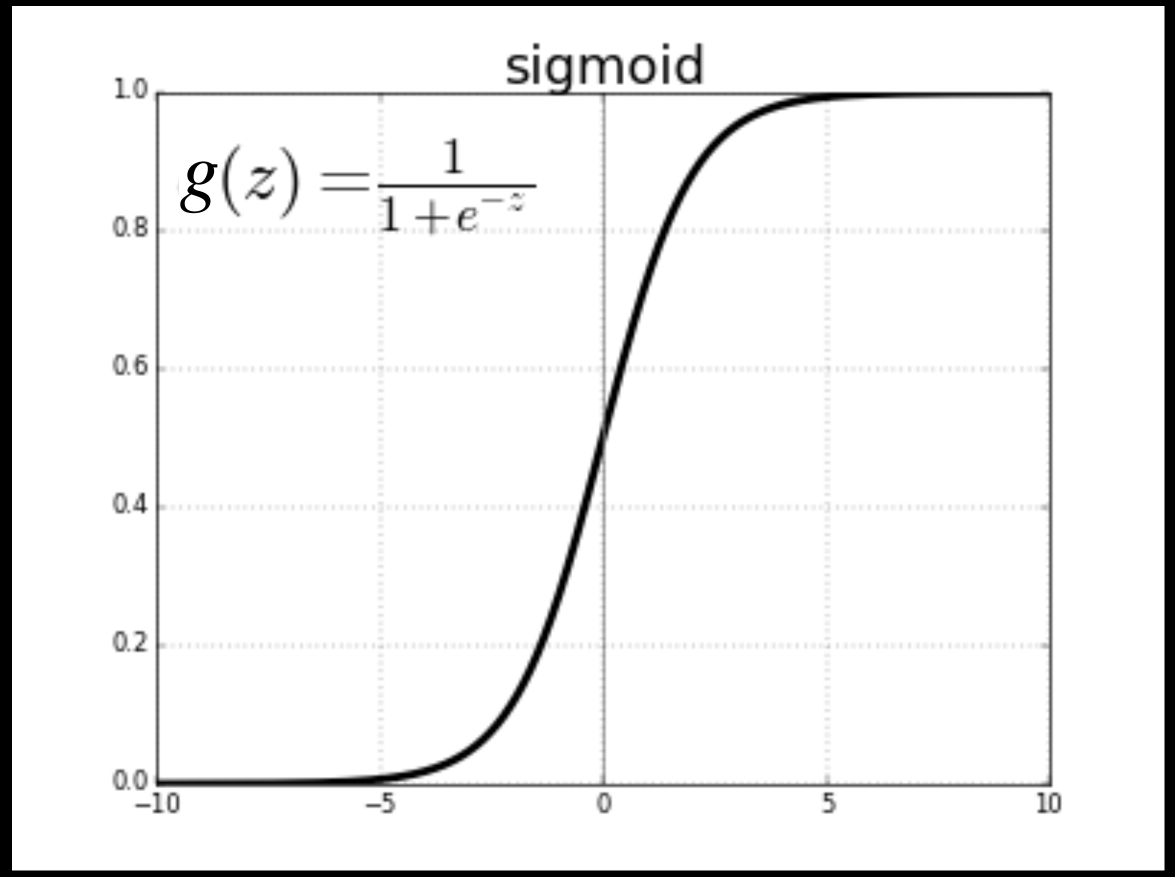


$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$

$$\boxed{a_2^{(3)} = g\left(\theta_{20}^{(2)}a_0^{(2)} + \theta_{21}^{(2)}a_1^{(2)} + \theta_{22}^{(2)}a_2^{(2)} + \theta_{23}^{(2)}a_3^{(2)}\right) = g\left(z_2^{(3)}\right)}$$

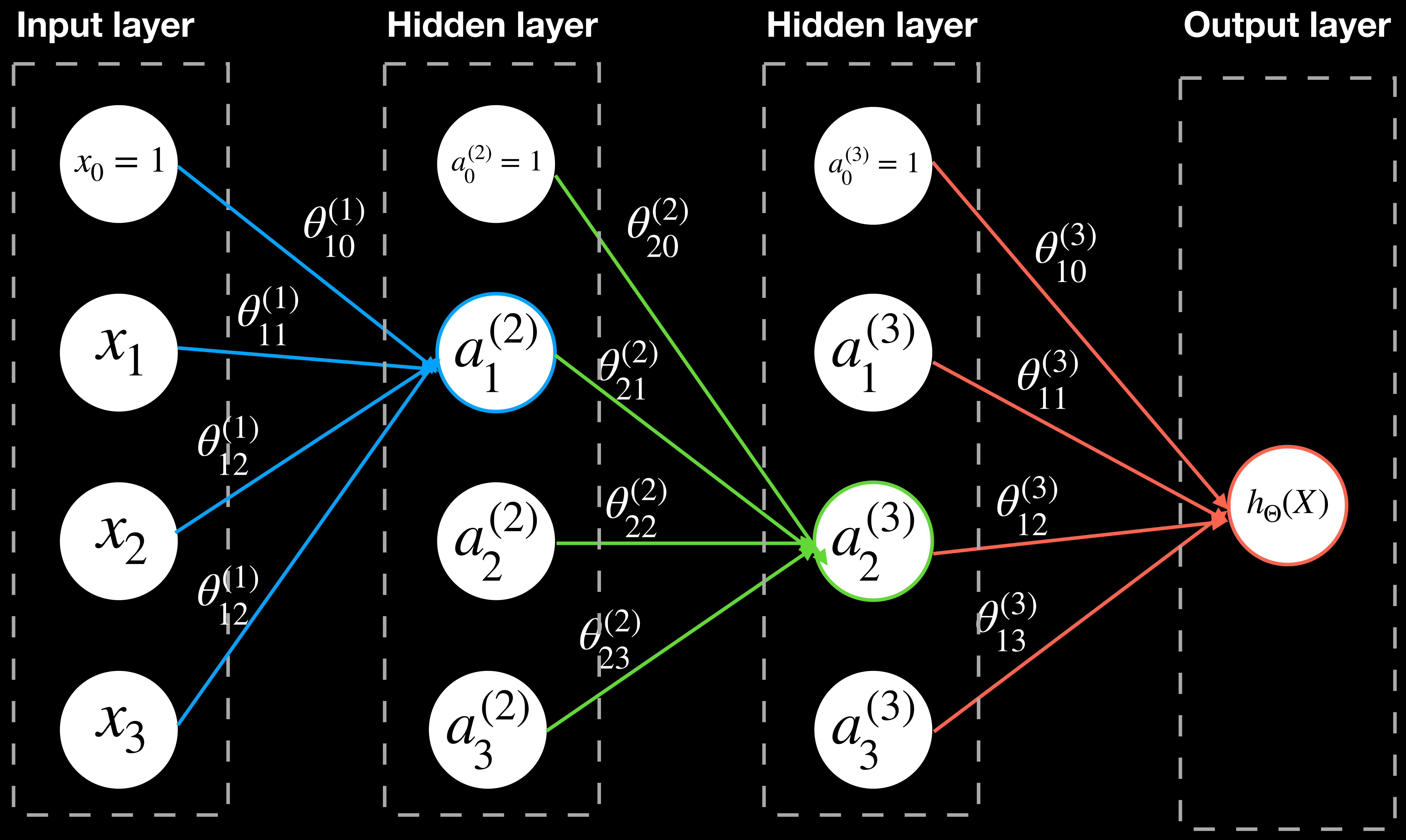


$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

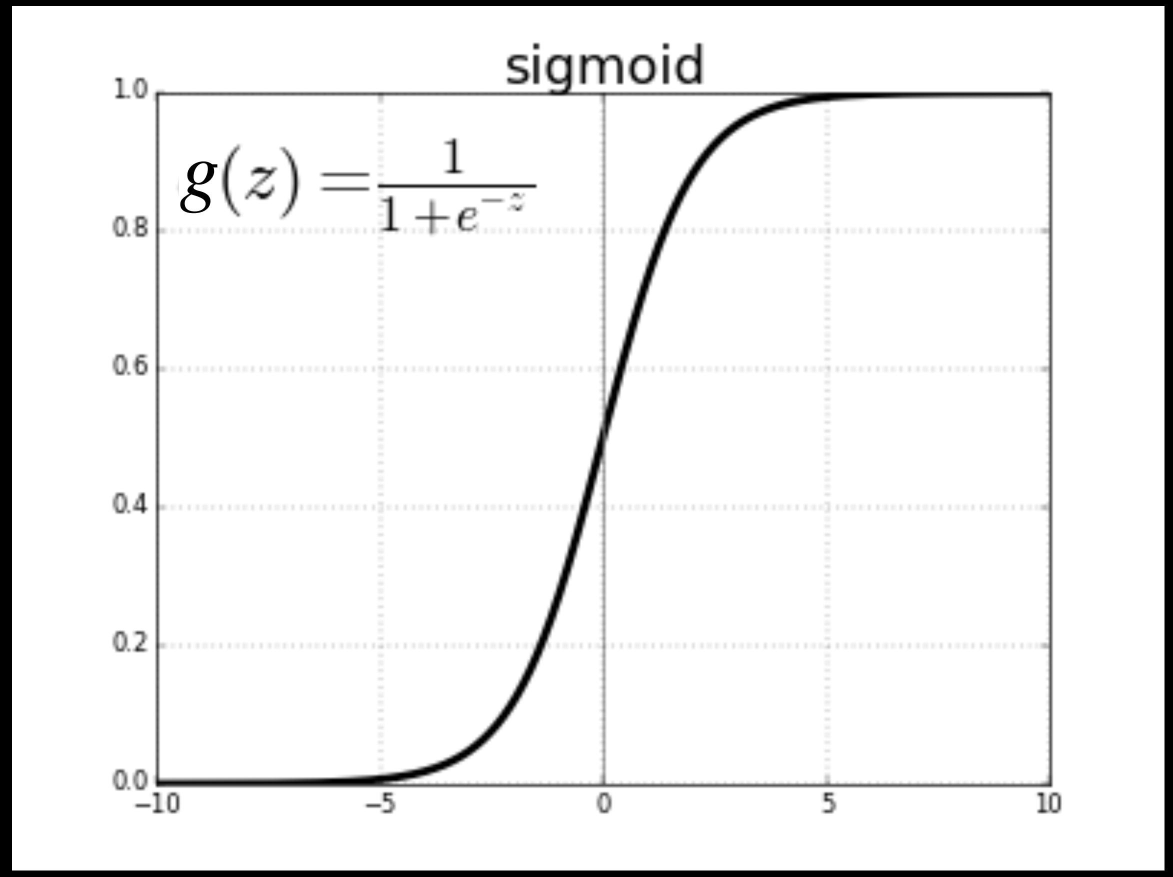


$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$

$$\boxed{a_2^{(3)} = g\left(\theta_{20}^{(2)}a_0^{(2)} + \theta_{21}^{(2)}a_1^{(2)} + \theta_{22}^{(2)}a_2^{(2)} + \theta_{23}^{(2)}a_3^{(2)}\right) = g\left(z_2^{(3)}\right)}$$



$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



$$\boxed{a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)}$$

$$\boxed{a_2^{(3)} = g\left(\theta_{20}^{(2)}a_0^{(2)} + \theta_{21}^{(2)}a_1^{(2)} + \theta_{22}^{(2)}a_2^{(2)} + \theta_{23}^{(2)}a_3^{(2)}\right) = g\left(z_2^{(3)}\right)}$$

$$\boxed{h_\Theta(X) = g\left(\theta_{10}^{(3)}a_0^{(3)} + \theta_{11}^{(3)}a_1^{(3)} + \theta_{12}^{(3)}a_2^{(3)} + \theta_{13}^{(3)}a_3^{(3)}\right) = g\left(z^{(4)}\right)}$$

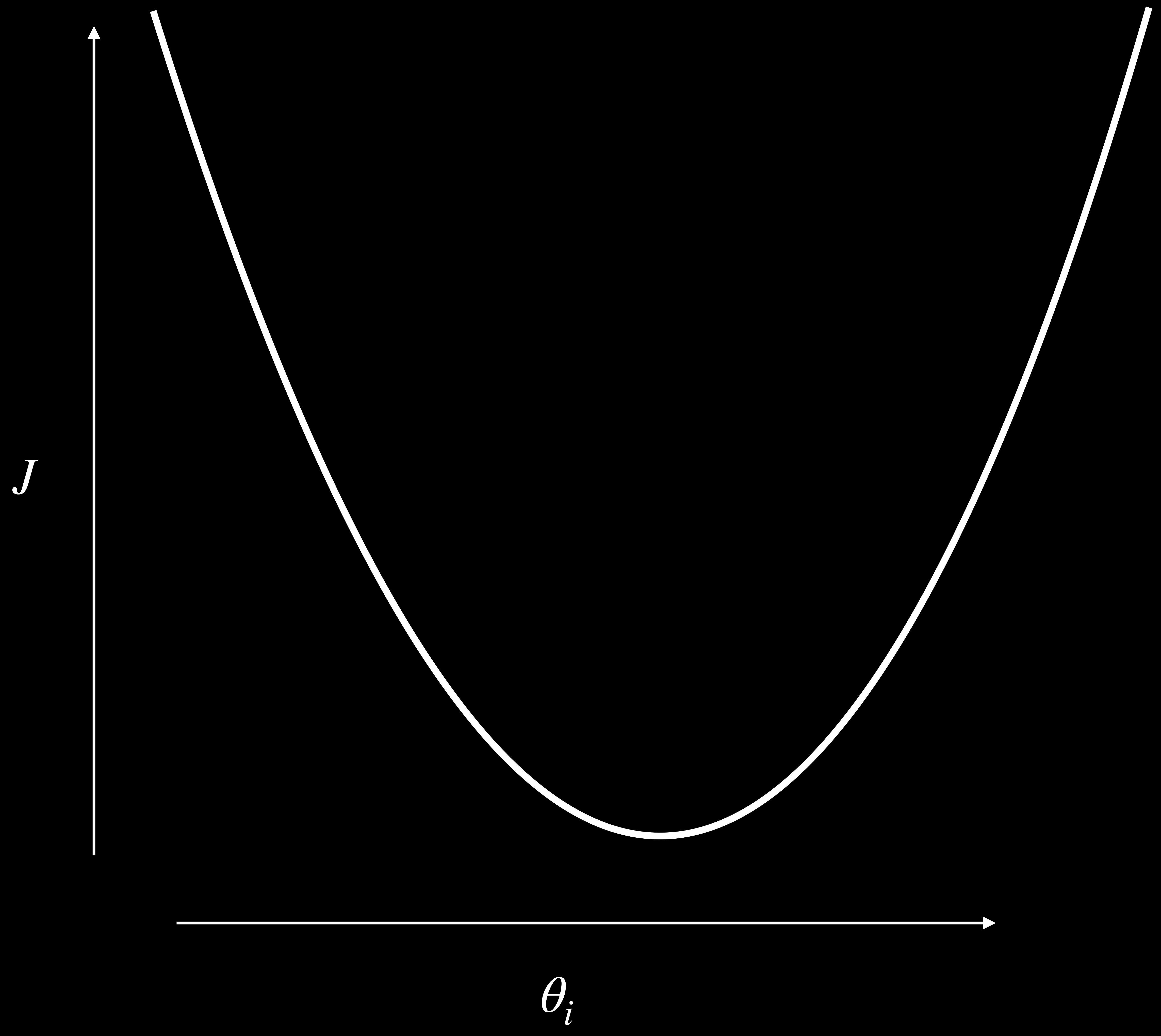
Cost Function

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K}\overbrace{\boxed{y_k^{(i)}\log\left(h_\Theta\left(x^{(i)}\right)\right)_k}}^{\textbf{y=1}} + \overbrace{\boxed{\left(1-y_k^{(i)}\right)\log\left(1-h_\Theta\left(x^{(i)}\right)\right)_k}}^{\textbf{y=0}}\right] + \overbrace{\boxed{\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\theta_{ji}^{(l)}\right)^2}}^{\textbf{Regularisation Term}}$$

- m is the number of inputs
- K is the number of classes
- L is the number of layers in network
- s_l is the number of units in the layer l (not including bias unit)
- λ is the regularisation parameter

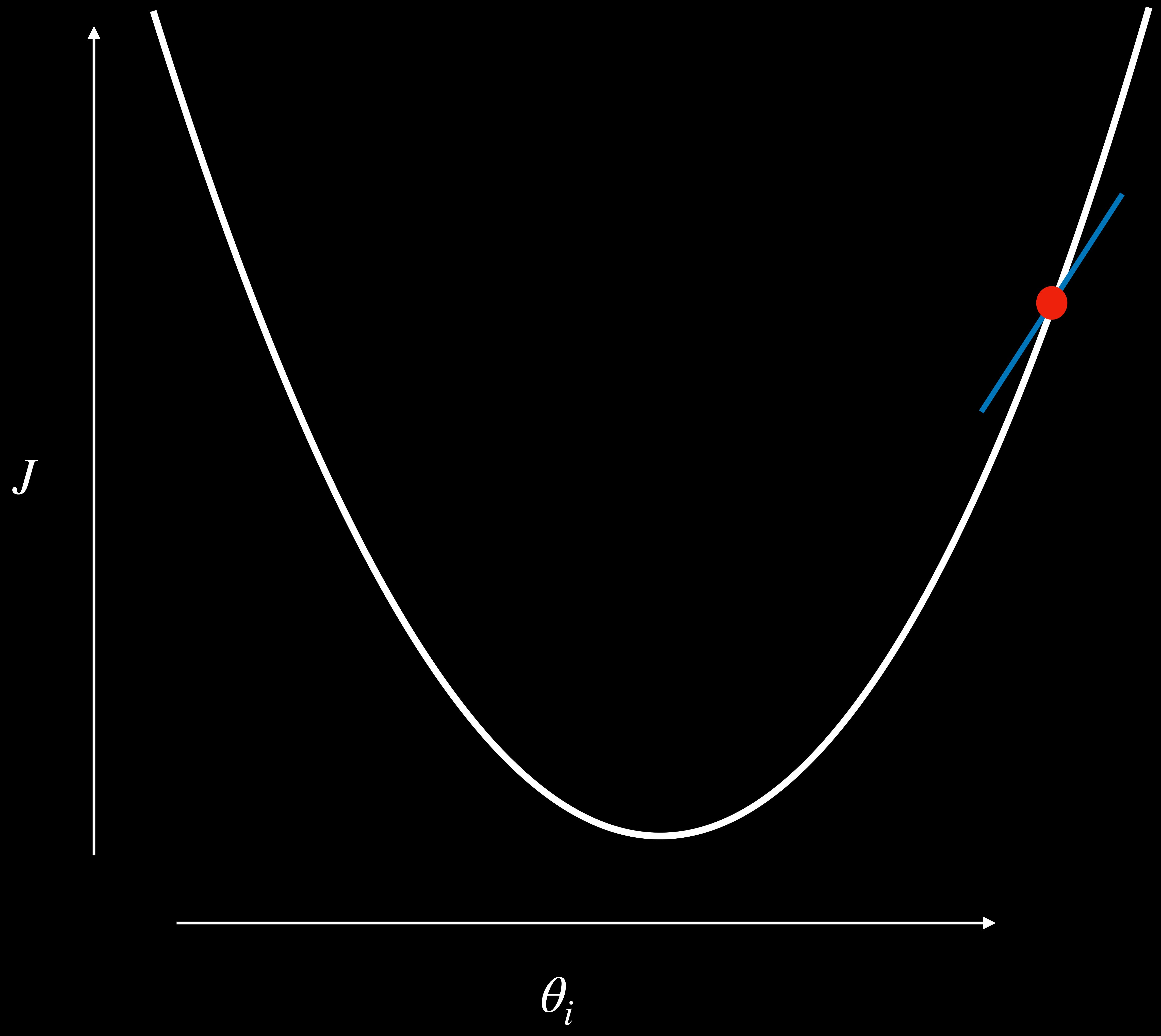
Minimising J

- Gradient Decent:
  - Calculate $\frac{\delta}{\delta\theta_i}J$
  - Update θ_i as: $\theta_i := \theta_i - \alpha\frac{\delta}{\delta\theta_i}J$
  - Recalculate J



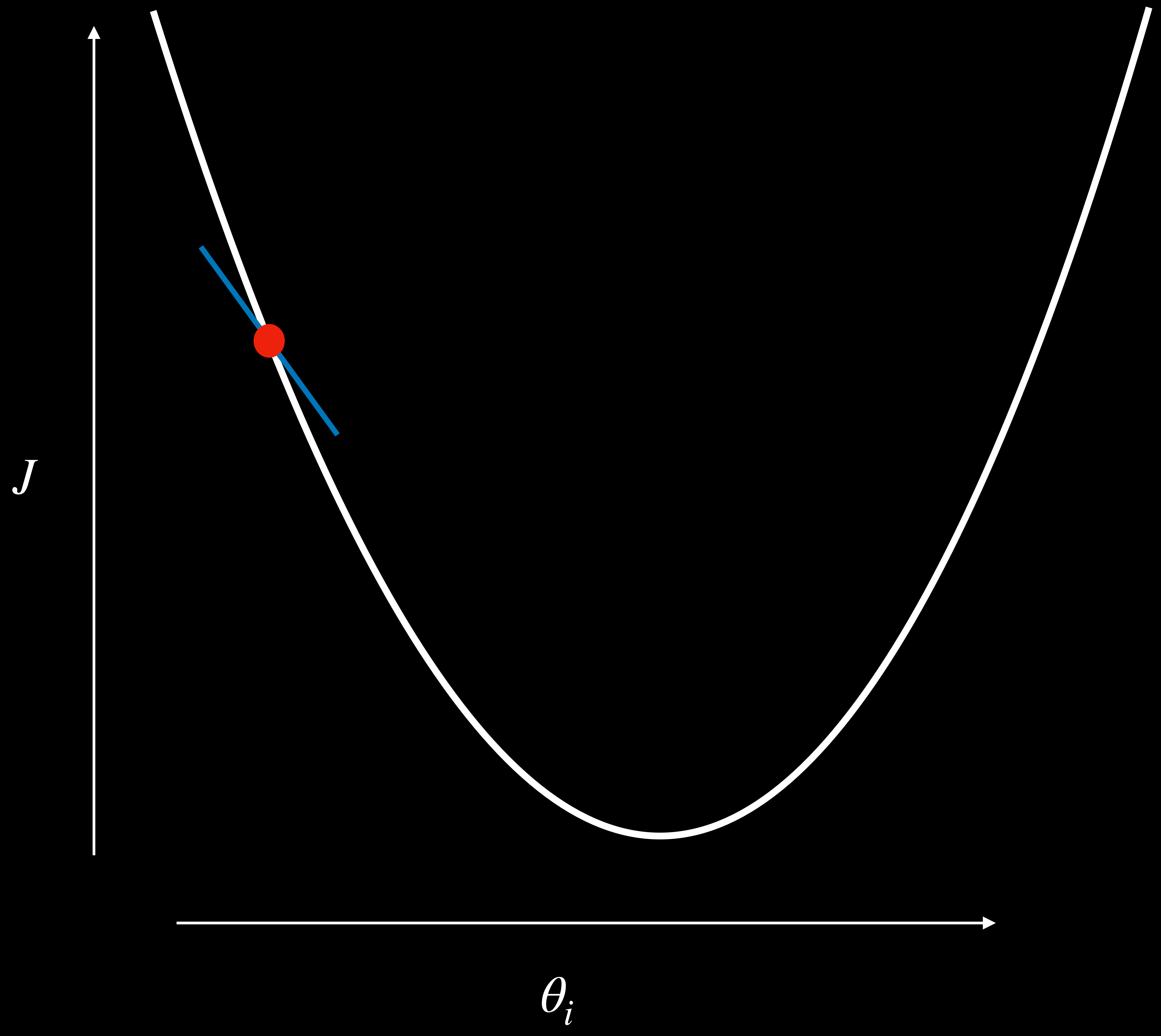
Minimising J

- Gradient Decent:
  - Calculate $\frac{\delta}{\delta\theta_i}J$
  - Update θ_i as: $\theta_i := \theta_i - \alpha\frac{\delta}{\delta\theta_i}J$
  - Recalculate J



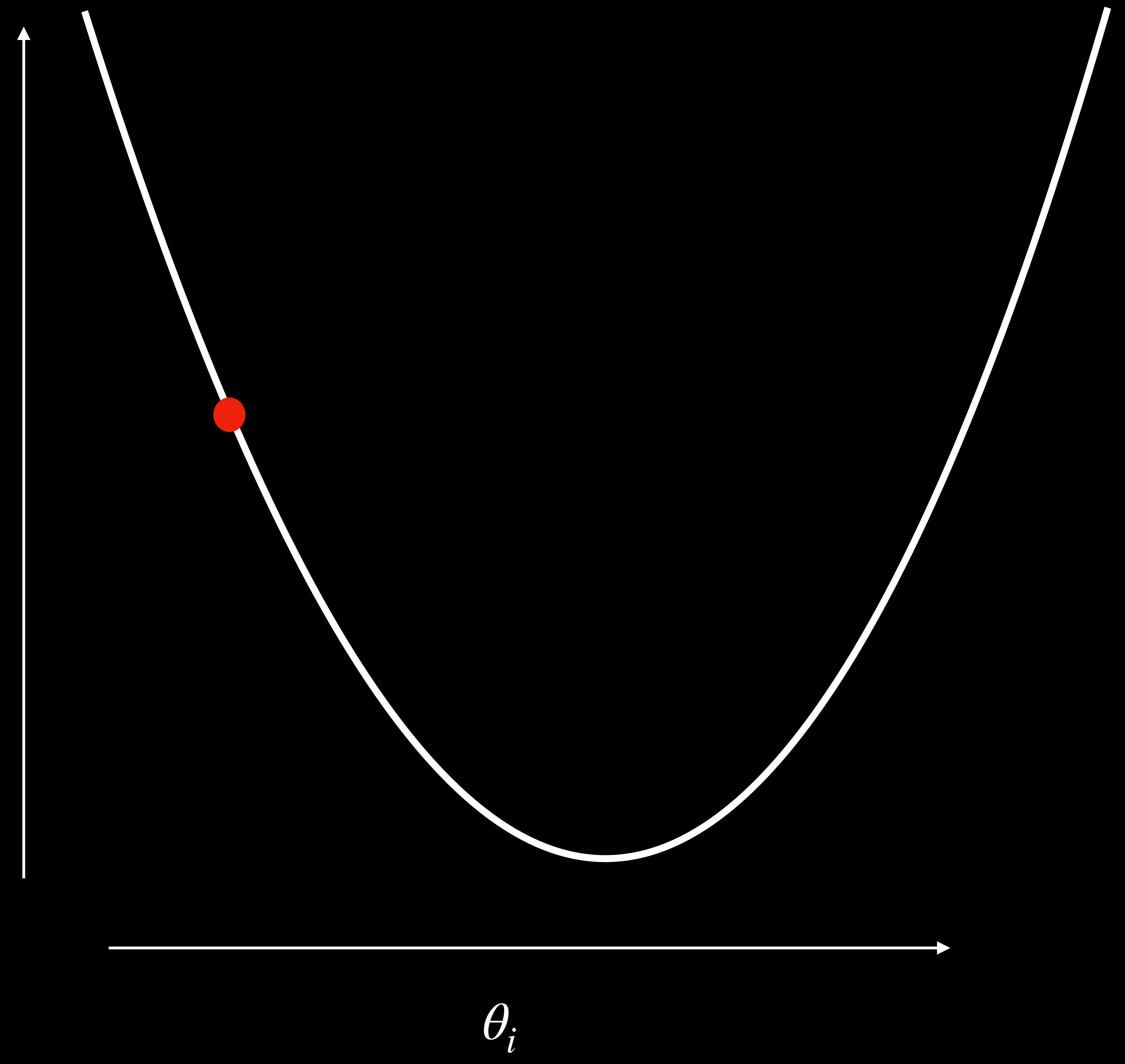
Minimising J

- Gradient Decent:
  - Calculate $\frac{\delta}{\delta\theta_i}J$
  - Update θ_i as: $\theta_i := \theta_i - \alpha\frac{\delta}{\delta\theta_i}J$
  - Recalculate J



Minimising J

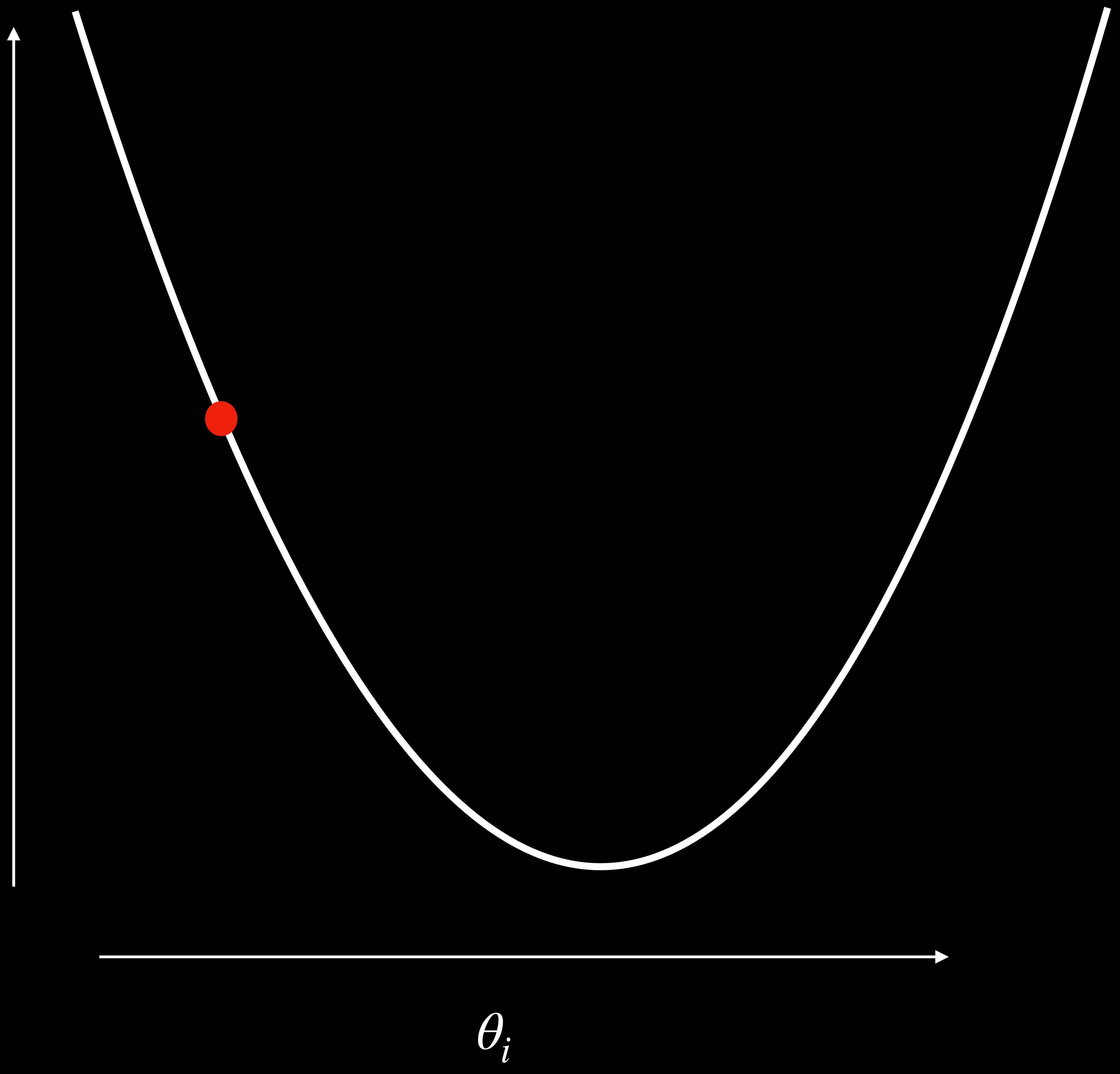
$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta\theta_i} J$$



Minimising J

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta\theta_i} J$$

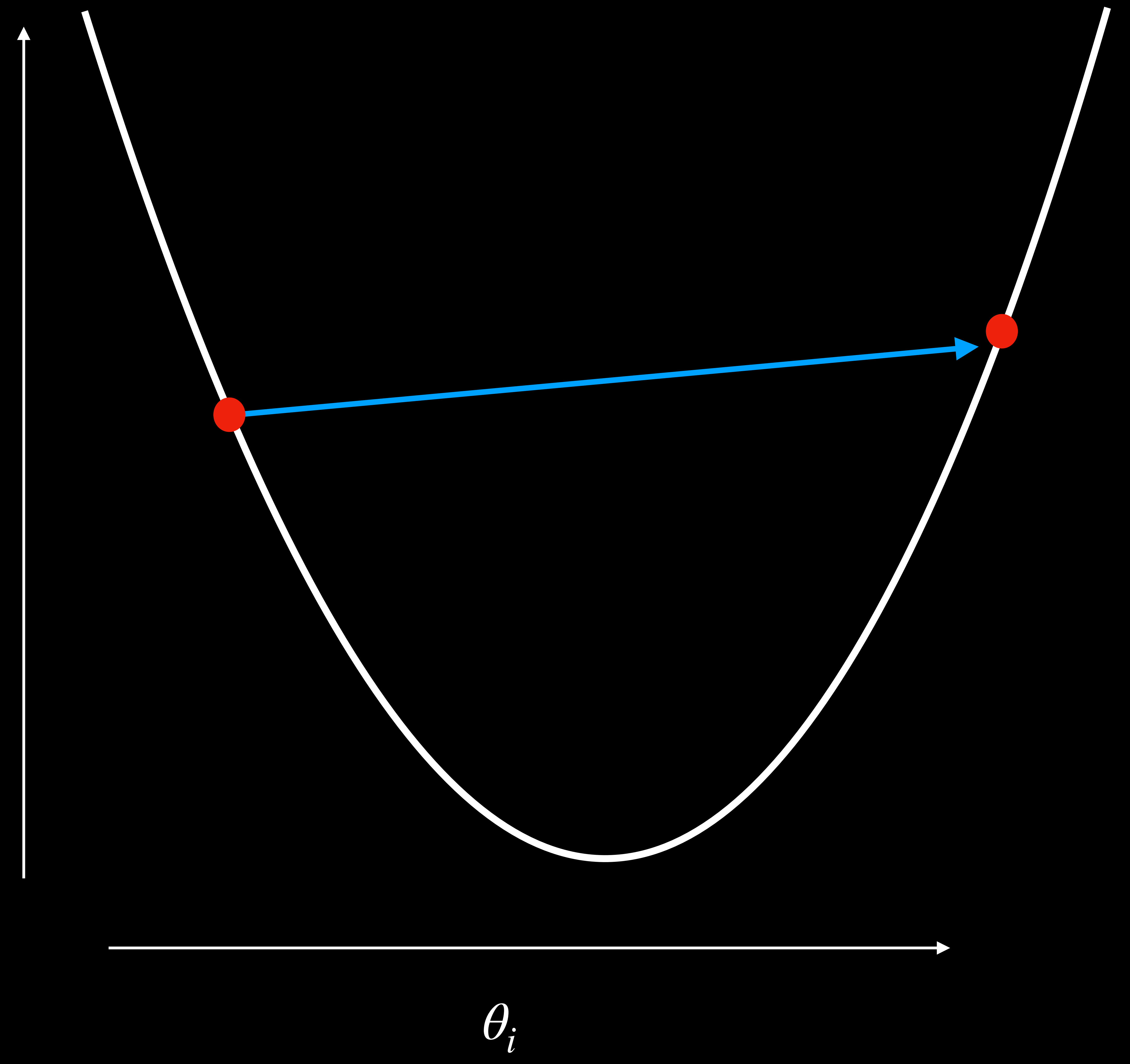
- If α too large end up diverging



Minimising J

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta \theta_i} J$$

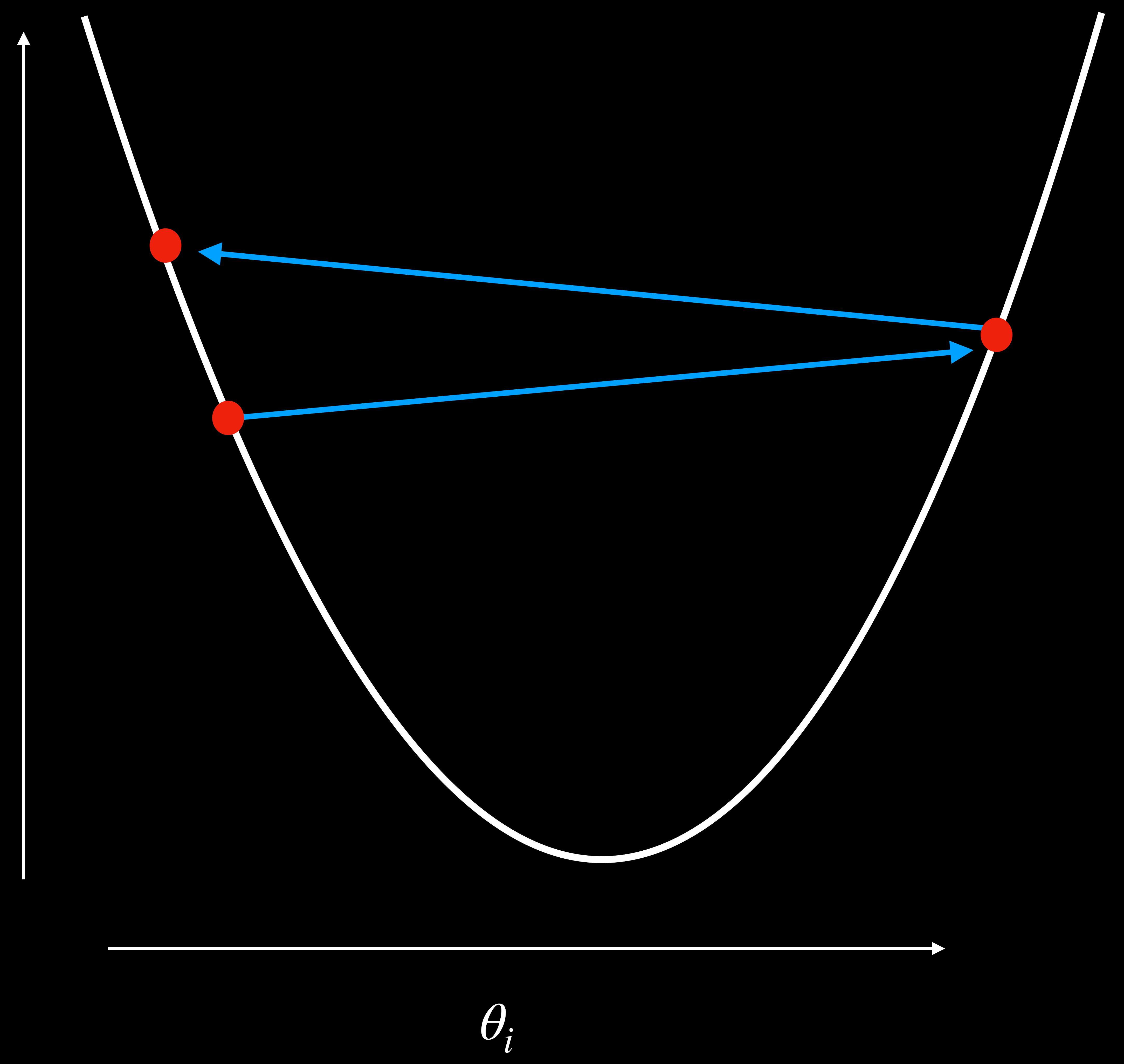
- If α too large end up diverging



Minimising J

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta\theta_i} J$$

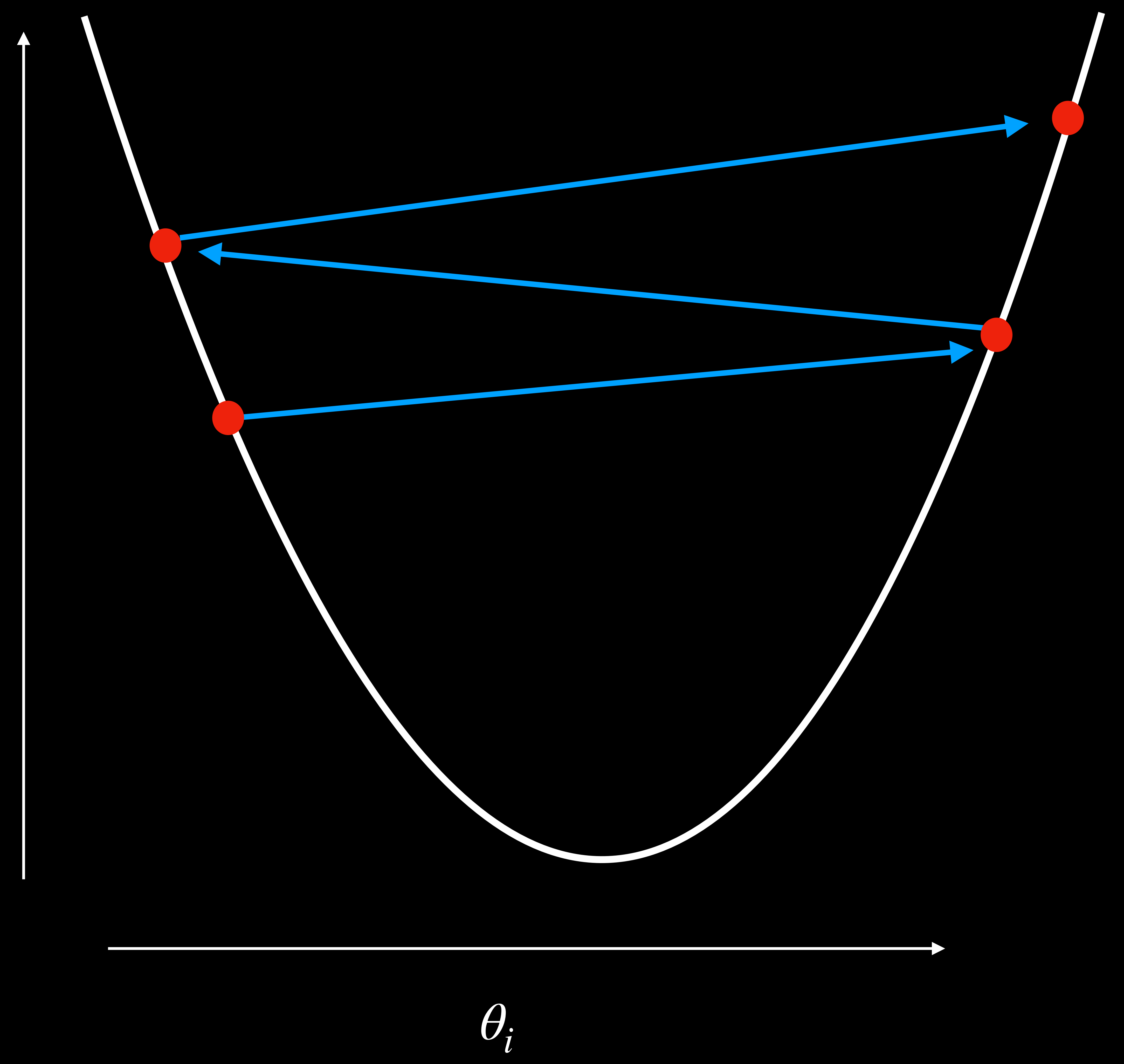
- If α too large end up diverging



Minimising J

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta\theta_i} J$$

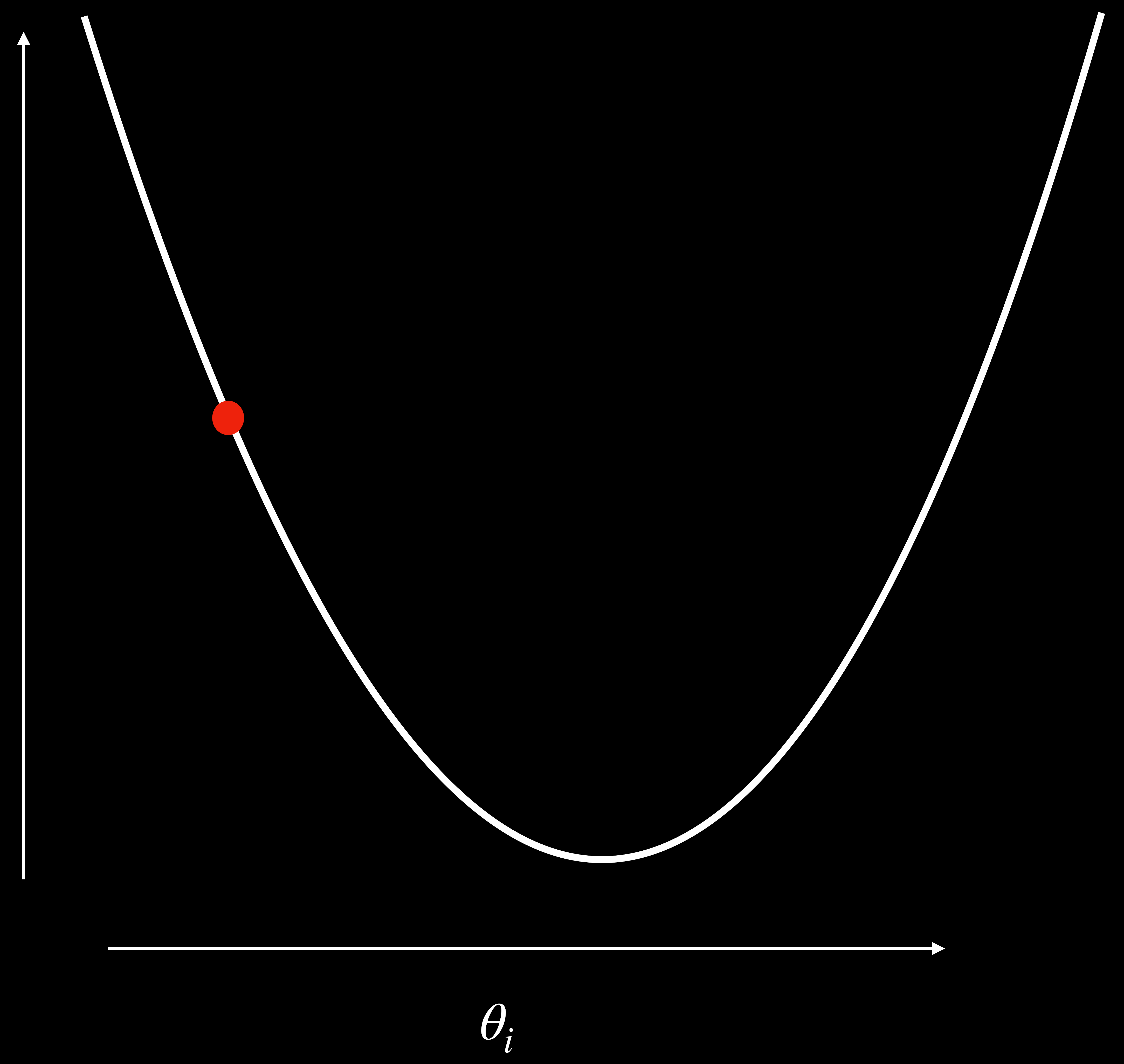
- If α too large end up diverging



Minimising J

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta\theta_i} J$$

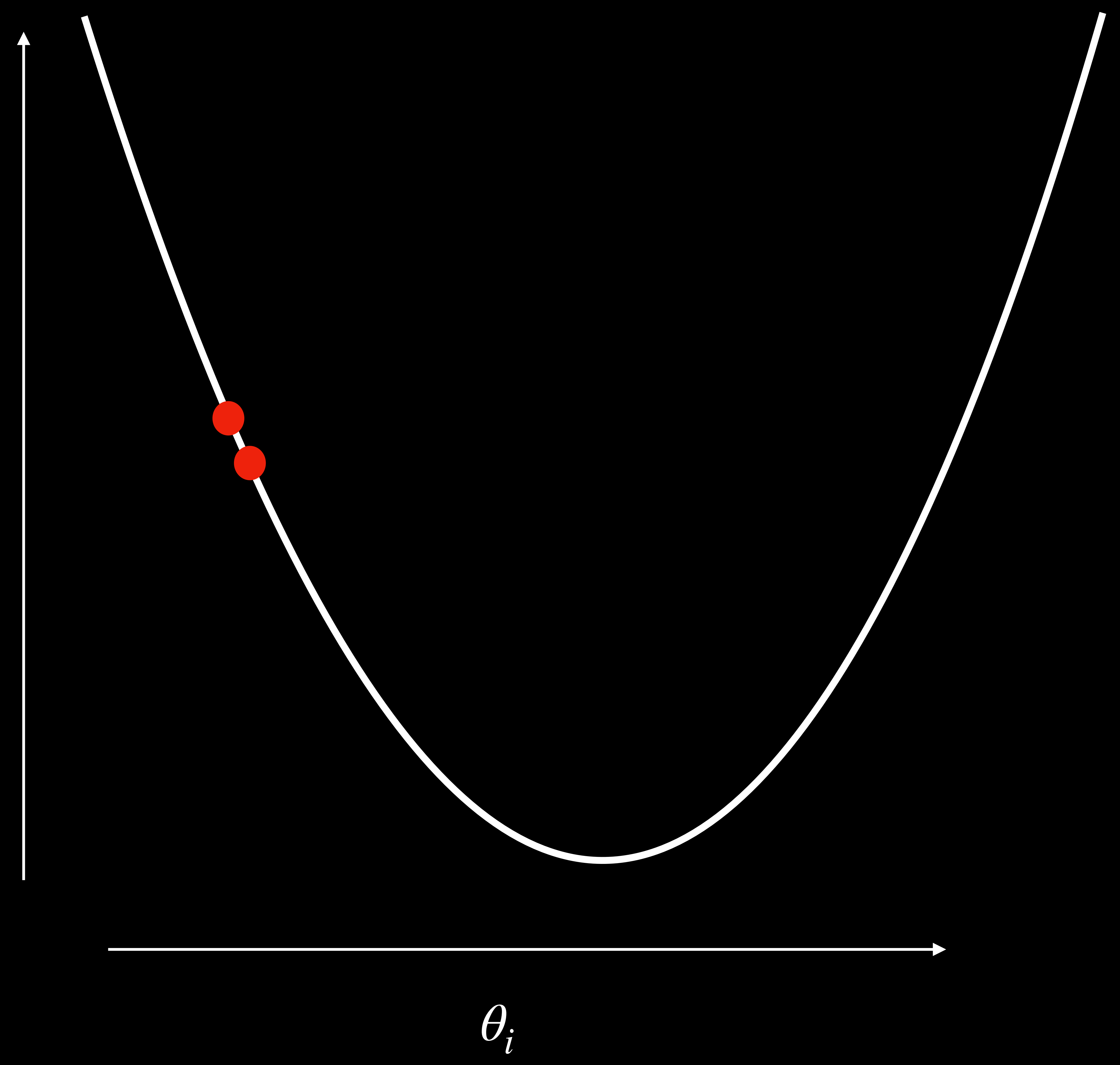
- If α too large end up diverging
- If α too small can take too long to converge



Minimising J

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta\theta_i} J$$

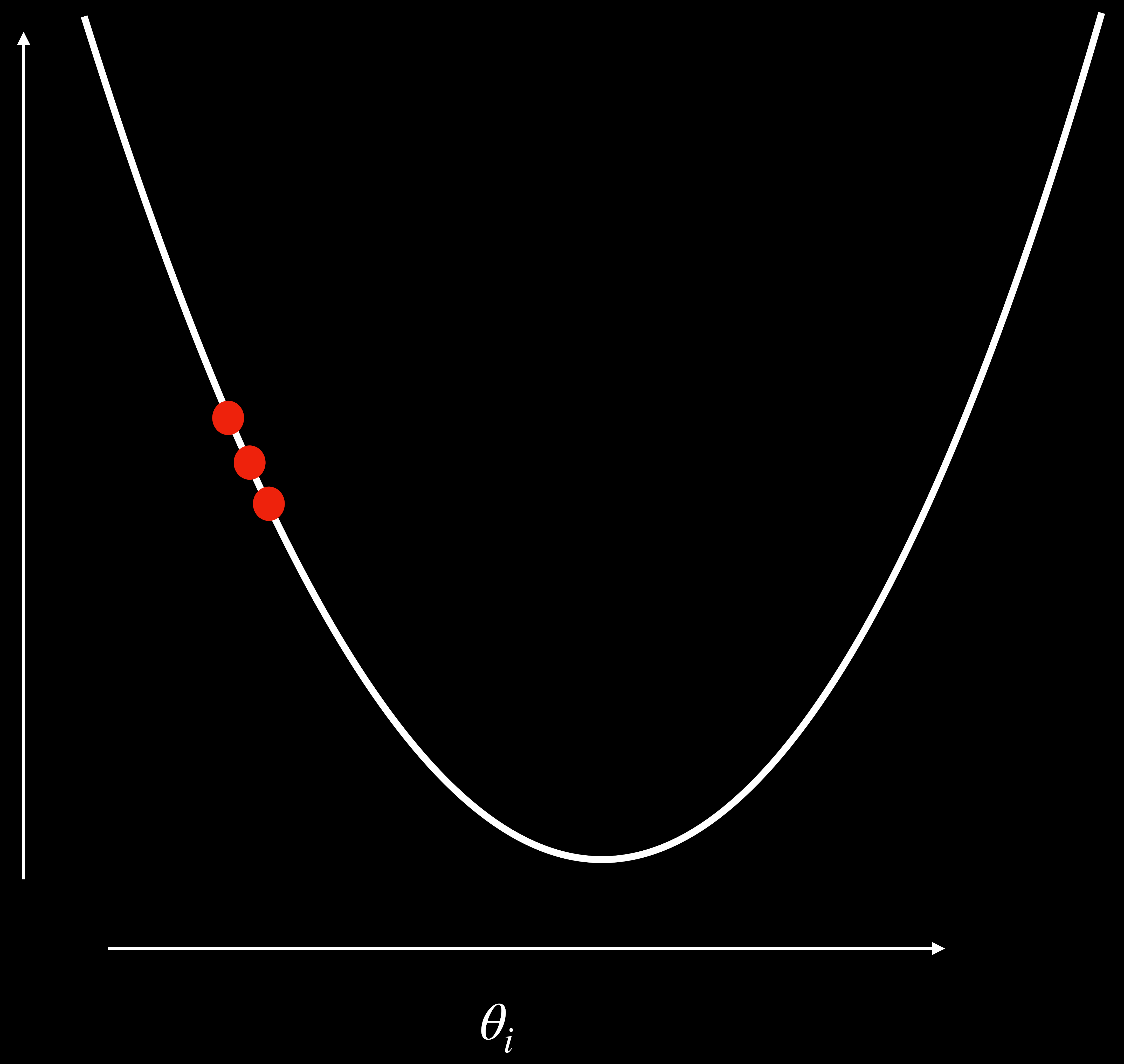
- If α too large end up diverging
- If α too small can take too long to converge



Minimising J

$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta \theta_i} J$$

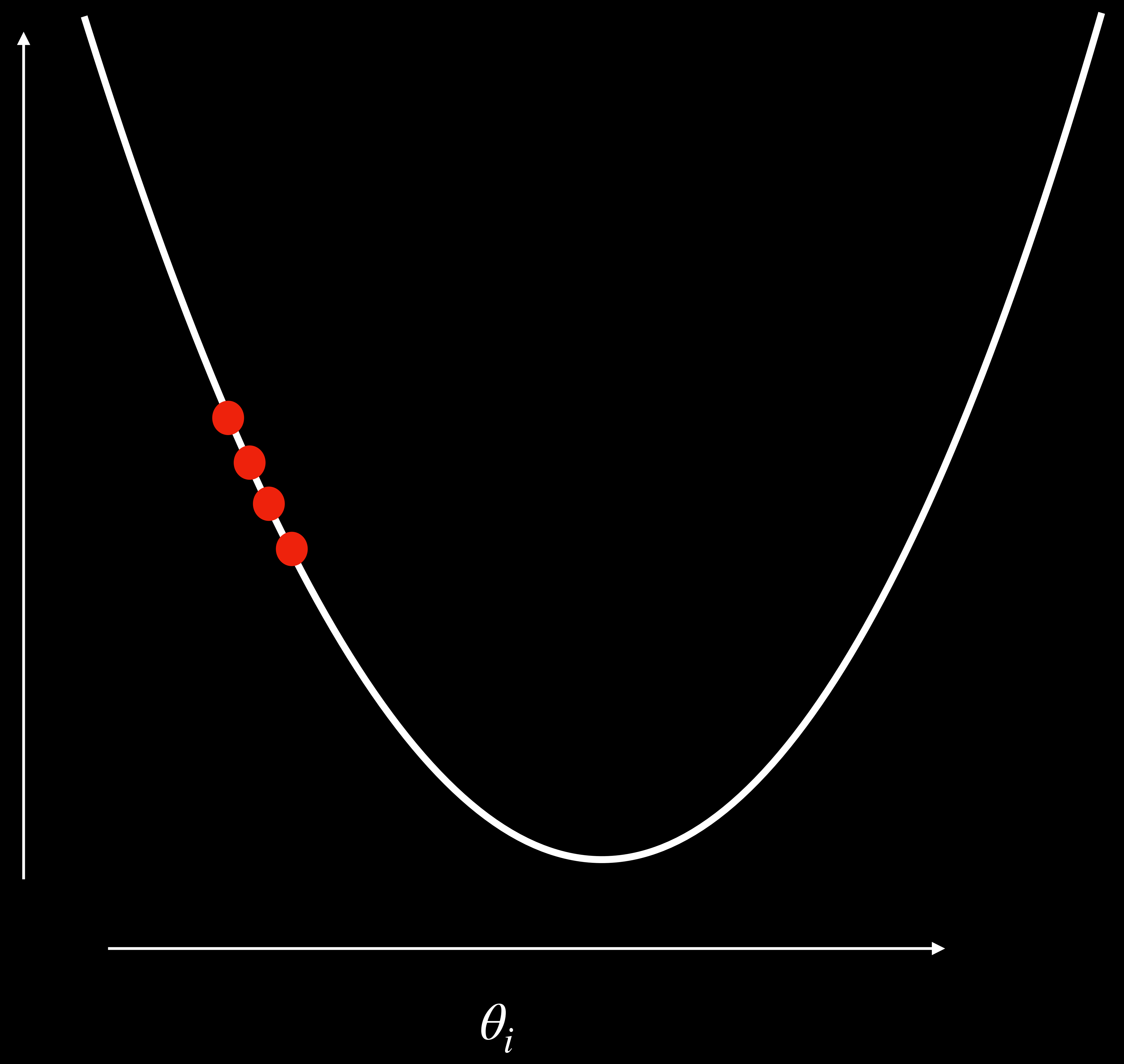
- If α too large end up diverging
- If α too small can take too long to converge



Minimising J

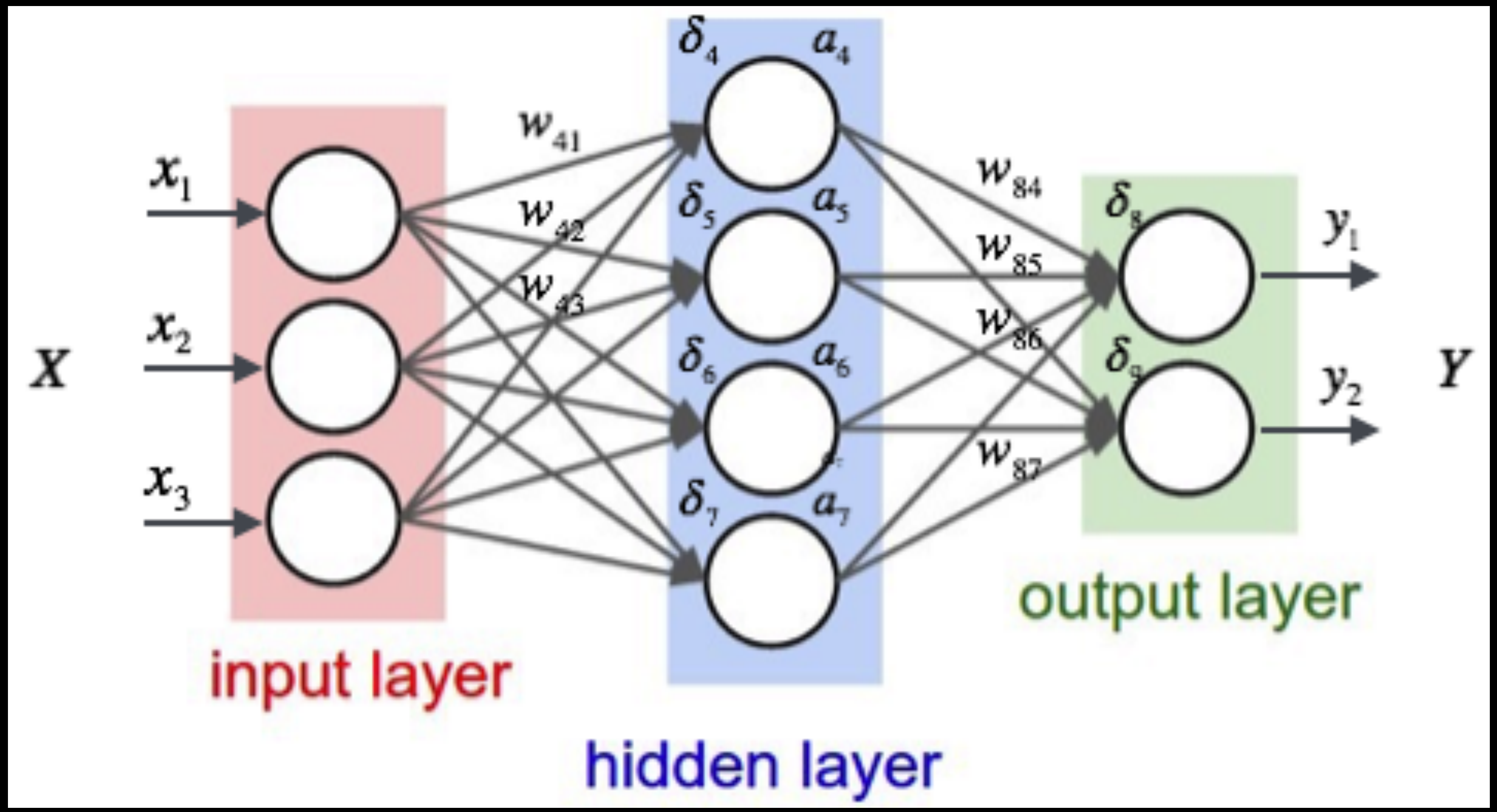
$$\theta_i := \theta_i - \alpha \frac{\delta}{\delta \theta_i} J$$

- If α too large end up diverging
- If α too small can take too long to converge



Gradient descent with NN

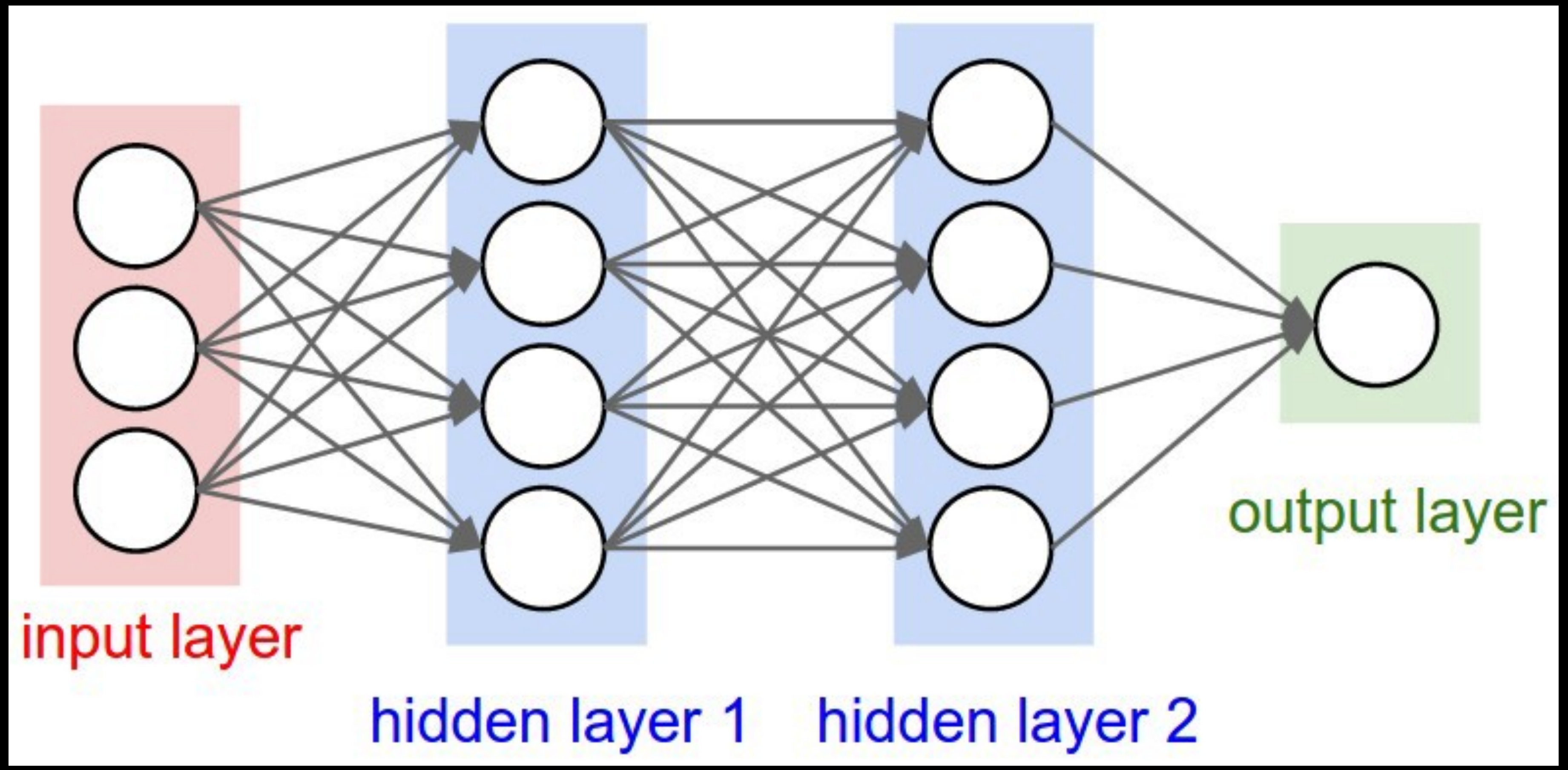
- Forward Propgation to calculate J
- Backward propagation:
  - Working backwards from output calculate the ‘error’ on each node.
  - Use this to calculate $\frac{\delta}{\delta\theta_i}J$ and update θ_i
- Repeat



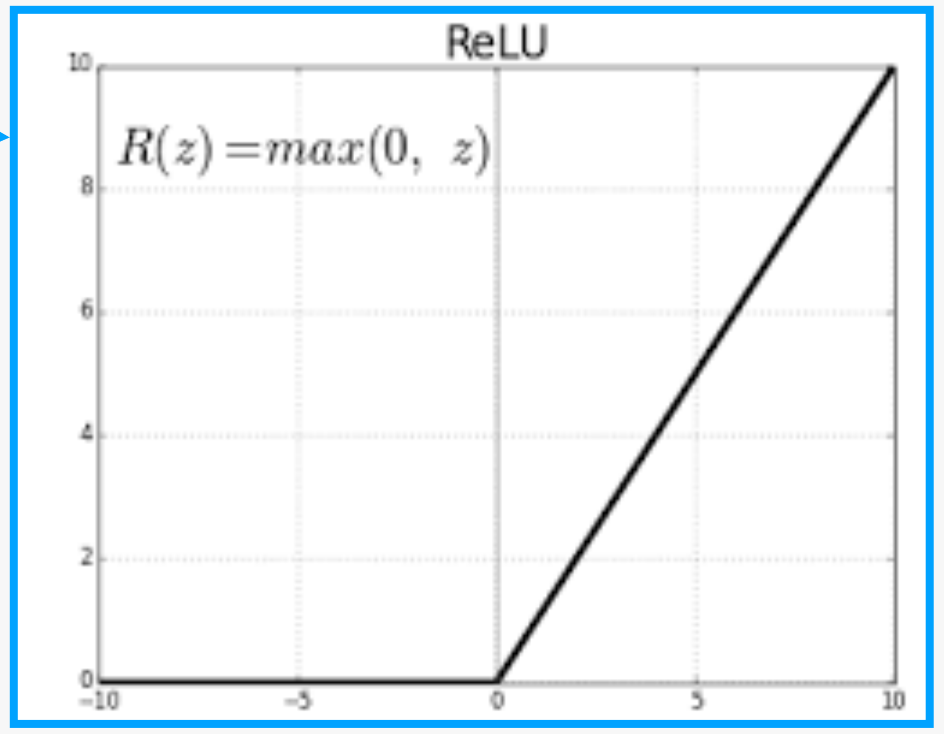
In [4]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

https://github.com/keras-team/keras/tree/master/examples

https://keras.io/

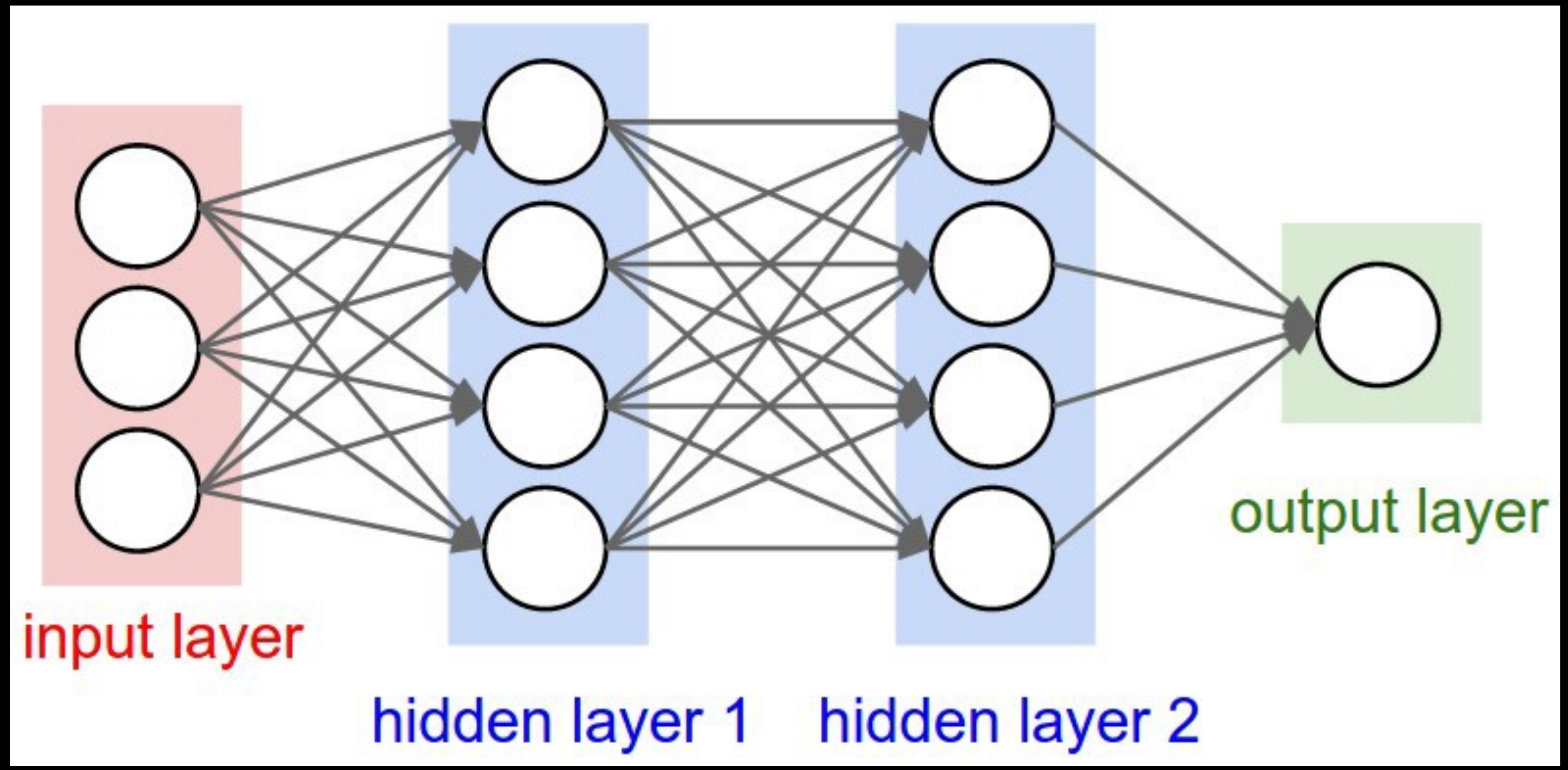


In [4]:
```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```



https://github.com/keras-team/keras/tree/master/examples

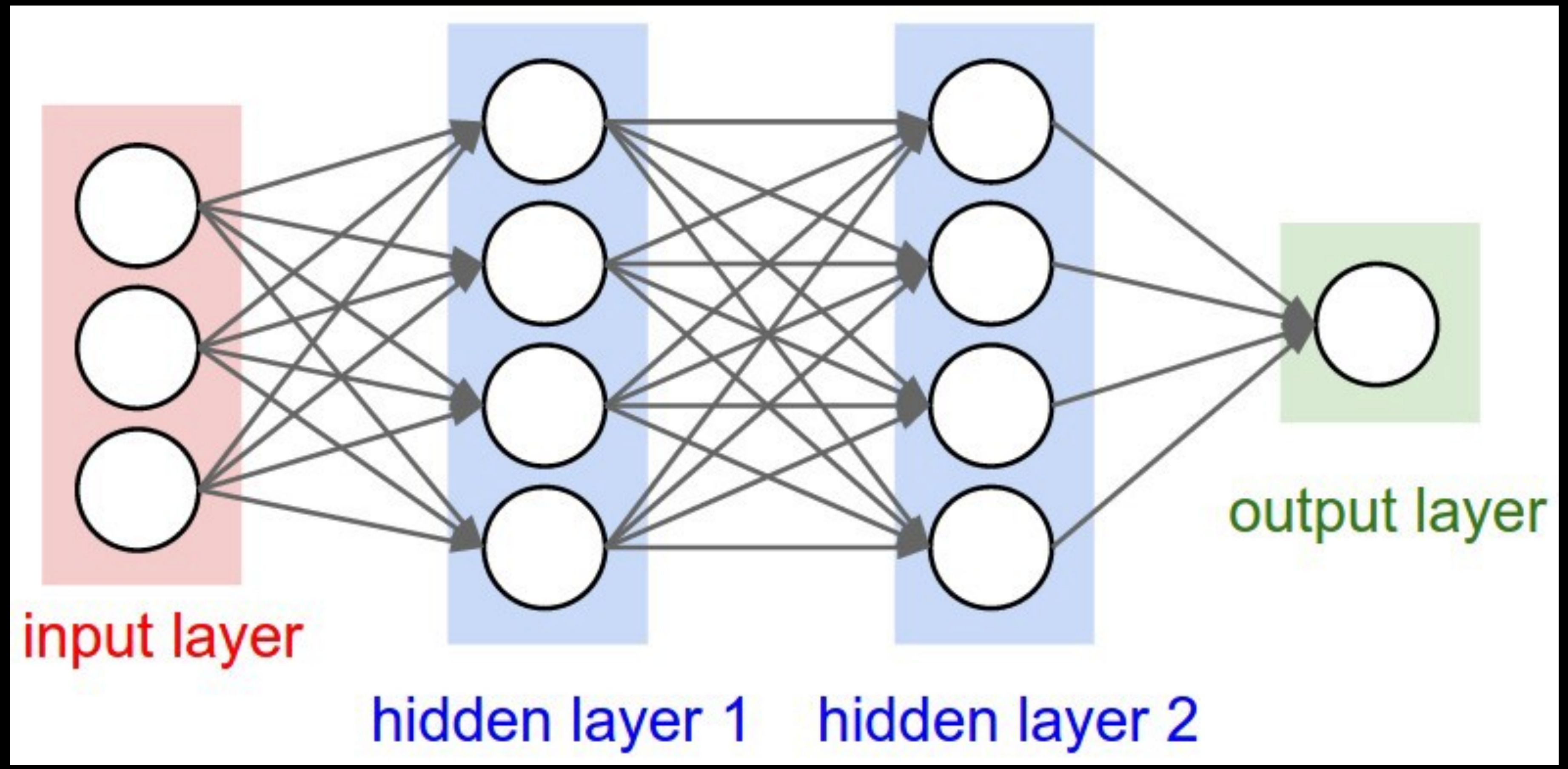
https://keras.io/



```
In [4]: import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense, Dropout
        from keras.optimizers import RMSprop

        model = Sequential()
        model.add(Dense(512, activation='relu', input_shape=(784,)))
        model.add(Dropout(0.2))
        model.add(Dense(512, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(num_classes, activation='softmax'))

        model.summary()

        model.compile(loss='categorical_crossentropy',
                      optimizer=RMSprop(),
                      metrics=['accuracy'])
```

https://github.com/keras-team/keras/tree/master/examples

https://keras.io/



In [5]:

```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.2488 - acc: 0.9238 - val_loss: 0.1489 - val_acc
: 0.9509
Epoch 2/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1031 - acc: 0.9690 - val_loss: 0.0789 - val_acc
: 0.9743
Epoch 3/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0743 - acc: 0.9772 - val_loss: 0.0800 - val_acc
: 0.9768
Epoch 4/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0590 - acc: 0.9818 - val_loss: 0.0769 - val_acc
: 0.9793
Epoch 5/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0494 - acc: 0.9848 - val_loss: 0.0887 - val_acc
: 0.9778
Epoch 6/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0440 - acc: 0.9866 - val_loss: 0.0832 - val_acc
: 0.9784
Epoch 7/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0383 - acc: 0.9884 - val_loss: 0.0834 - val_ac
c: 0.9812
Epoch 8/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.0352 - acc: 0.9900 - val_loss: 0.0793 - val_ac
c: 0.9825
```

In [5]:

```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
history = model.fit(X_train,y_train,epochs = 50,validation_split=0.10,class_weight=class_weight)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.2488 - acc: 0.9238 - val_loss: 0.1489 - val_acc
: 0.9509
Epoch 2/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1031 - acc: 0.9690 - val_loss: 0.0789 - val_acc
: 0.9743
Epoch 3/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0743 - acc: 0.9772 - val_loss: 0.0800 - val_acc
: 0.9768
Epoch 4/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0590 - acc: 0.9818 - val_loss: 0.0769 - val_acc
: 0.9793
Epoch 5/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0494 - acc: 0.9848 - val_loss: 0.0887 - val_acc
: 0.9778
Epoch 6/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0440 - acc: 0.9866 - val_loss: 0.0832 - val_acc
: 0.9784
Epoch 7/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0383 - acc: 0.9884 - val_loss: 0.0834 - val_ac
c: 0.9812
Epoch 8/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.0352 - acc: 0.9900 - val_loss: 0.0793 - val_ac
c: 0.9825
```

```
In [5]:
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
history = model.fit(X_train,y_train,epochs = 50,validation_split=0.10,class_weight=class_weight)
```

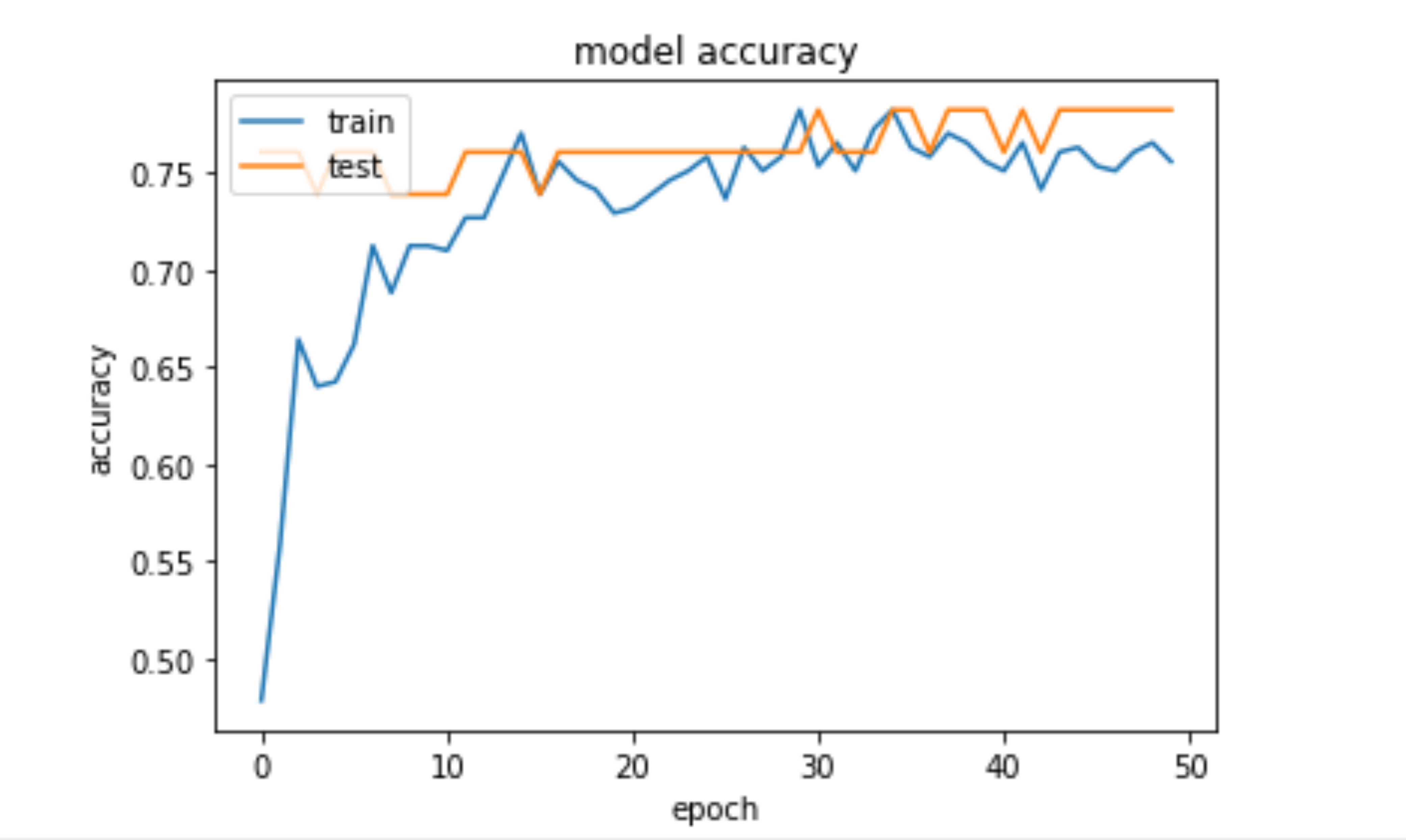
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.2488 - acc: 0.9238 - val_loss: 0.1489 - val_acc
: 0.9509
Epoch 2/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1031 - acc: 0.9690 - val_loss: 0.0789 - val_acc
: 0.9743
Epoch 3/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0743 - acc: 0.9772 - val_loss: 0.0800 - val_acc
: 0.9768
Epoch 4/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0590 - acc: 0.9818 - val_loss: 0.0769 - val_acc
: 0.9793
Epoch 5/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0494 - acc: 0.9848 - val_loss: 0.0887 - val_acc
: 0.9778
Epoch 6/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0440 - acc: 0.9866 - val_loss: 0.0832 - val_acc
: 0.9784
Epoch 7/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0383 - acc: 0.9884 - val_loss: 0.0834 - val_ac
c: 0.9812
Epoch 8/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.0352 - acc: 0.9900 - val_loss: 0.0793 - val_ac
c: 0.9825
```

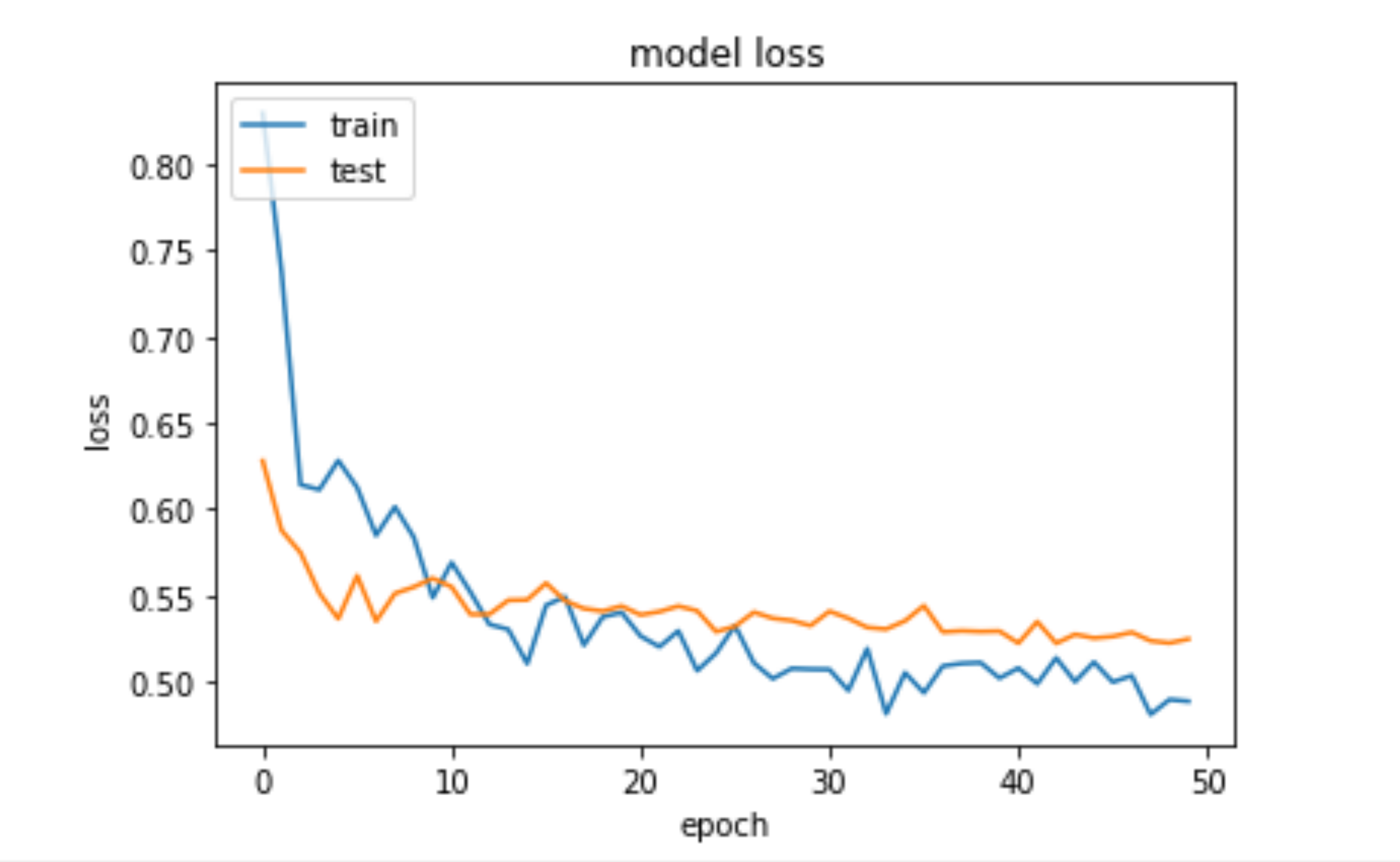
```
In [6]:
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.11100612514
Test accuracy: 0.9832
```

```
In [18]: plt.plot(history.history['acc'])
         plt.plot(history.history['val_acc'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()
```

```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





Useful references

- Free coursera course: https://www.coursera.org/learn/machine-learning
- Keras documentation: https://keras.io/
- Previous NN hack night: https://github.com/JBCA-MachineLearning/Typhoons-and-Hurricanes-Hacknight
- More info on activation functions: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0