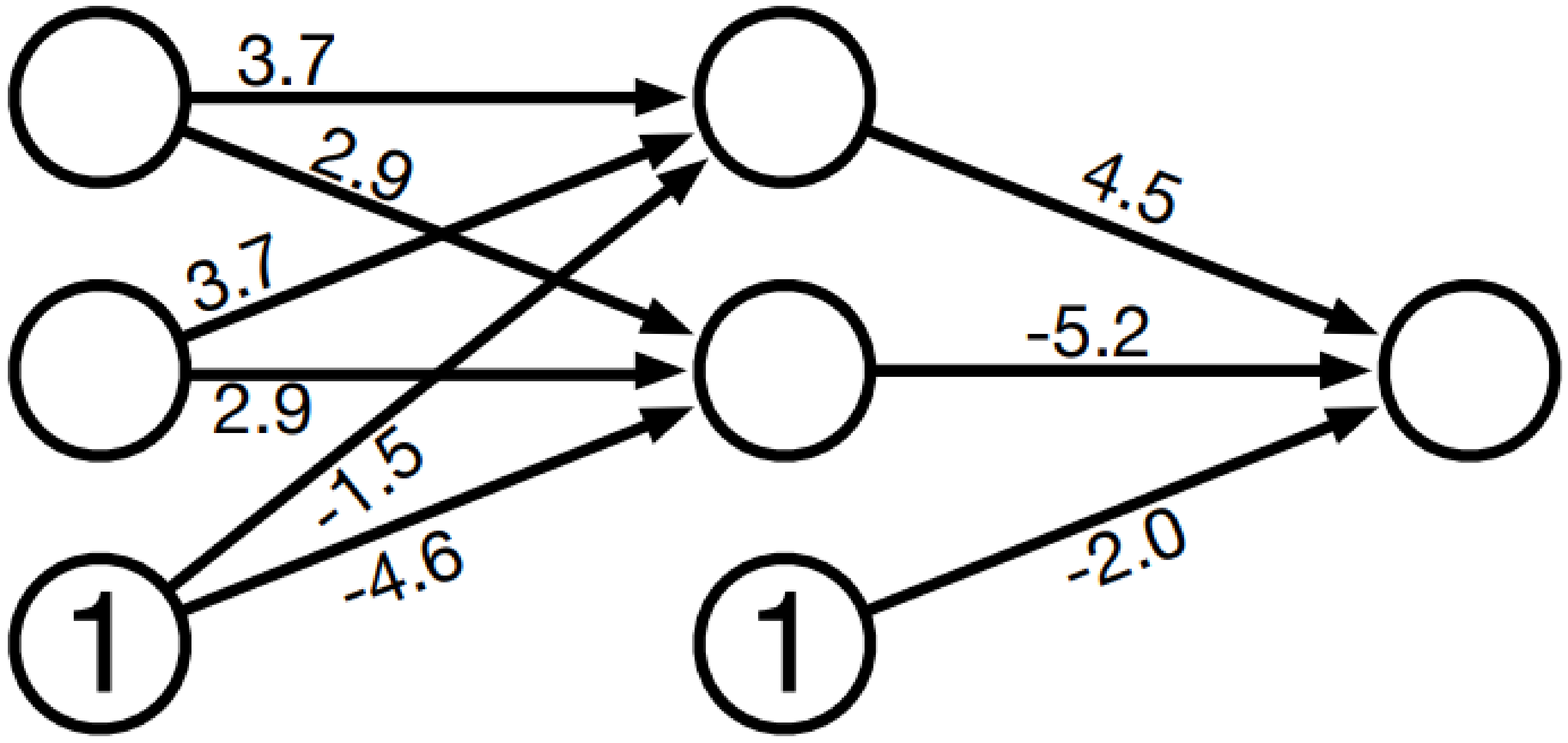
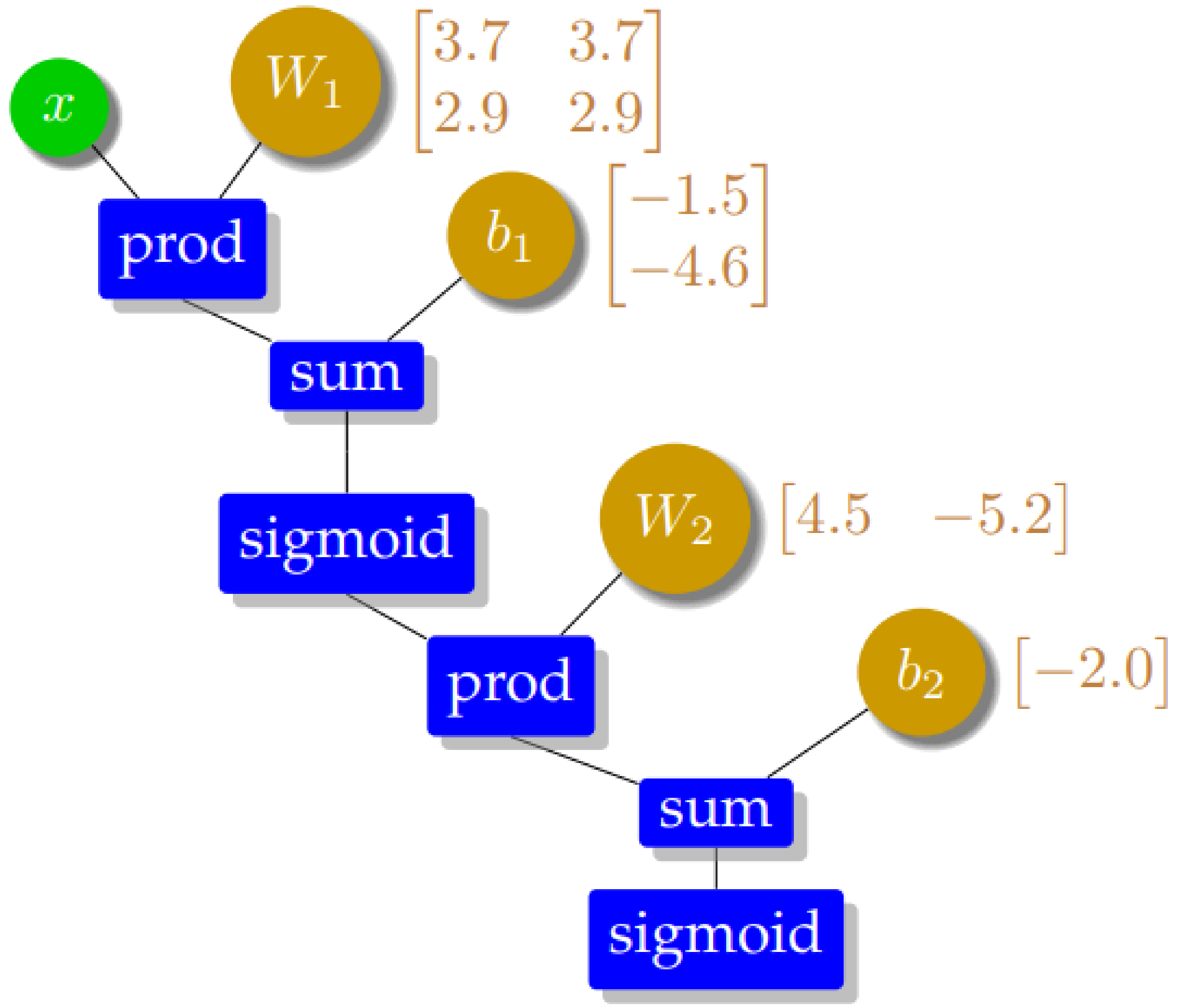
Implementing Deep Nets

Simple NN

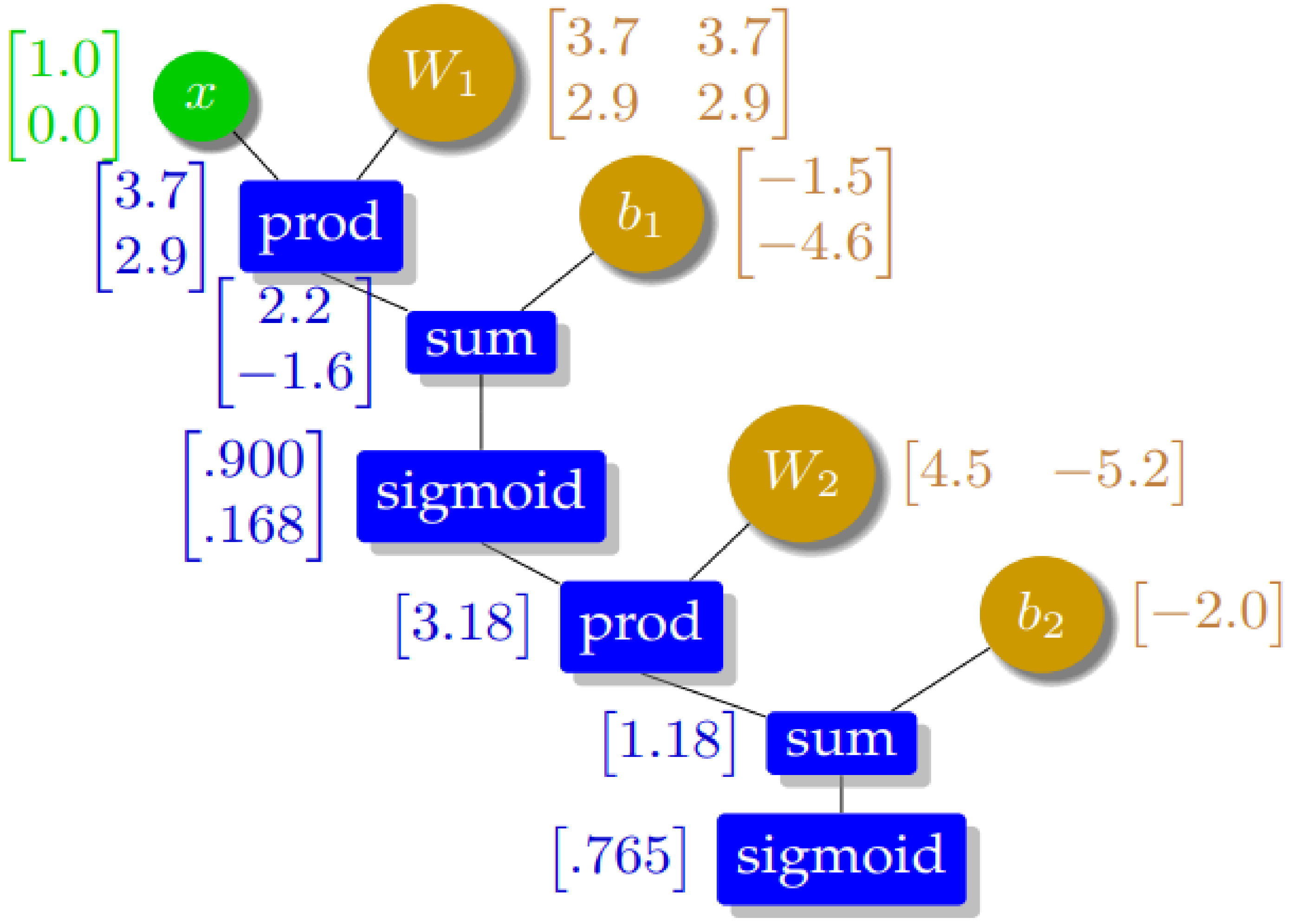


Computational Graph



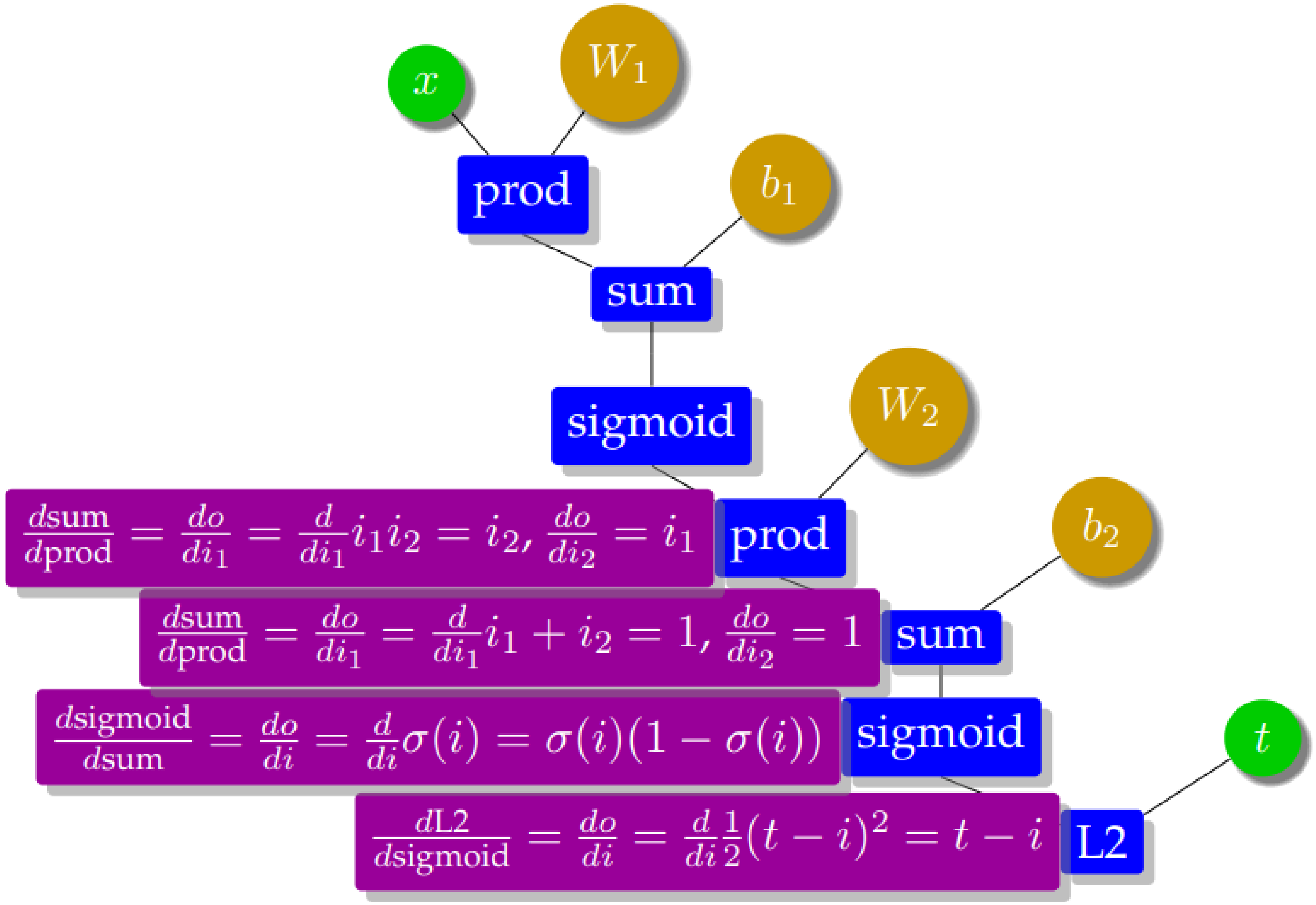
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
prod
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
sum
sigmoid
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix}$
prod
b_2
$\begin{bmatrix}-2.0\end{bmatrix}$
sum
sigmoid

Forward pass



$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
$\begin{bmatrix}3.7\\2.9\end{bmatrix}$
prod
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
$\begin{bmatrix}2.2\\-1.6\end{bmatrix}$
sum
$\begin{bmatrix}.900\\.168\end{bmatrix}$
sigmoid
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix}$
$[3.18]$
prod
b_2
$[-2.0]$
$[1.18]$
sum
$[.765]$
sigmoid

Derivatives of each node



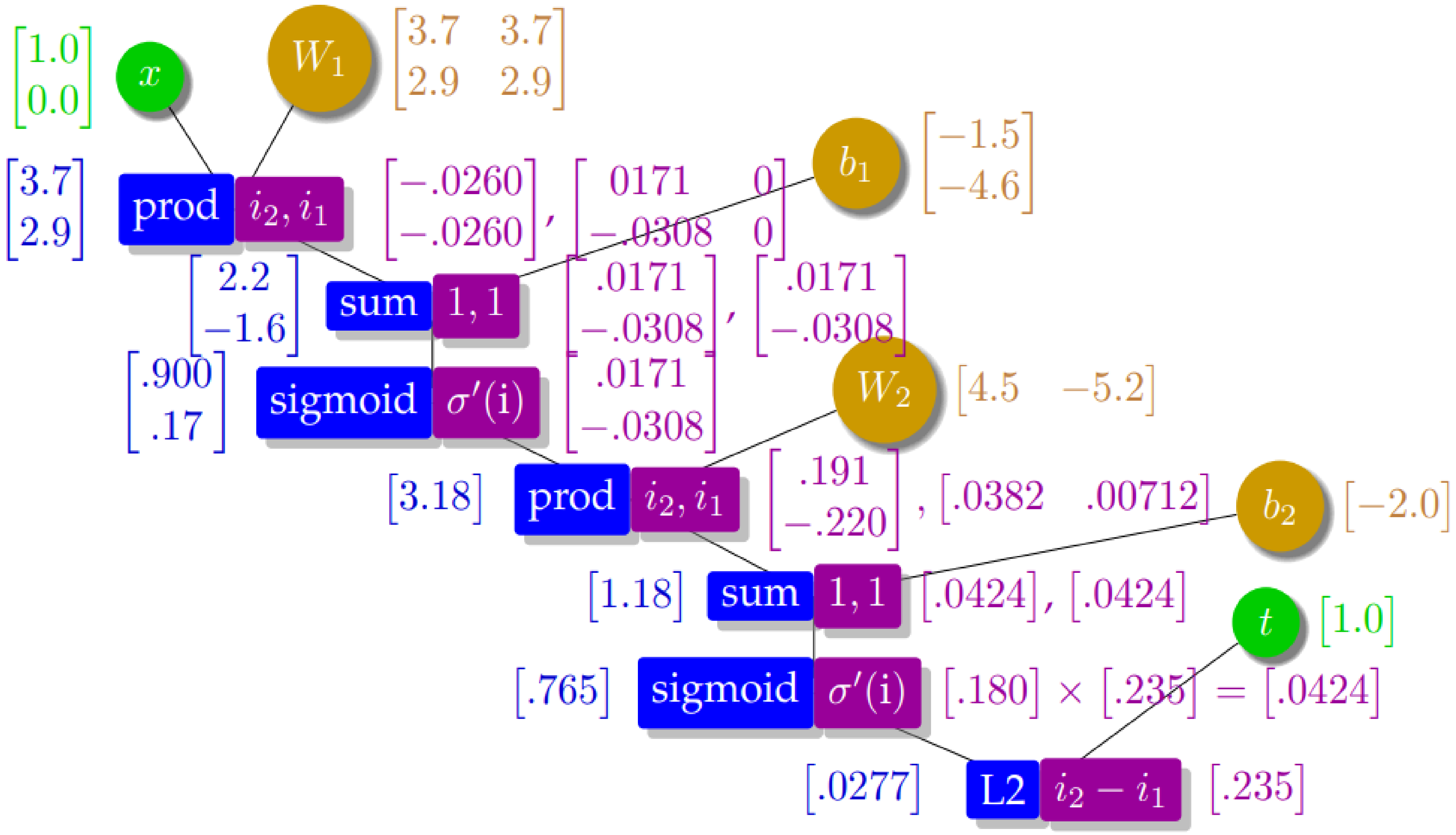
$$\frac{d\text{sum}}{d\text{prod}} = \frac{do}{di_1} = \frac{d}{di_1} i_1 i_2 = i_2,\ \frac{do}{di_2} = i_1$$

$$\frac{d\text{sum}}{d\text{prod}} = \frac{do}{di_1} = \frac{d}{di_1} i_1 + i_2 = 1,\ \frac{do}{di_2} = 1$$

$$\frac{d\text{sigmoid}}{d\text{sum}} = \frac{do}{di} = \frac{d}{di}\sigma(i) = \sigma(i)(1-\sigma(i))$$

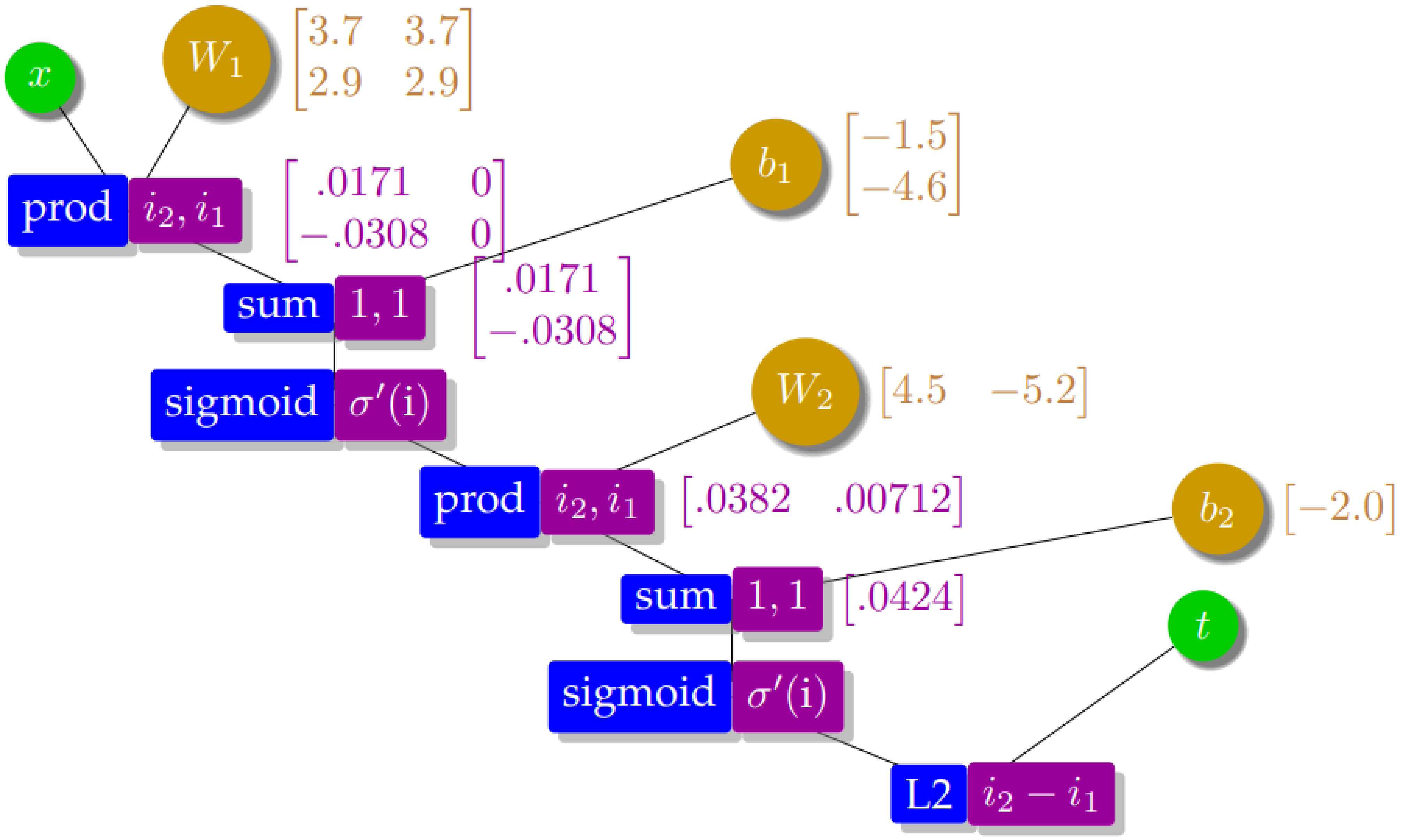
$$\frac{d\text{L2}}{d\text{sigmoid}} = \frac{do}{di} = \frac{d}{di}\frac{1}{2}(t-i)^2 = t-i$$

Backward pass: Derivative



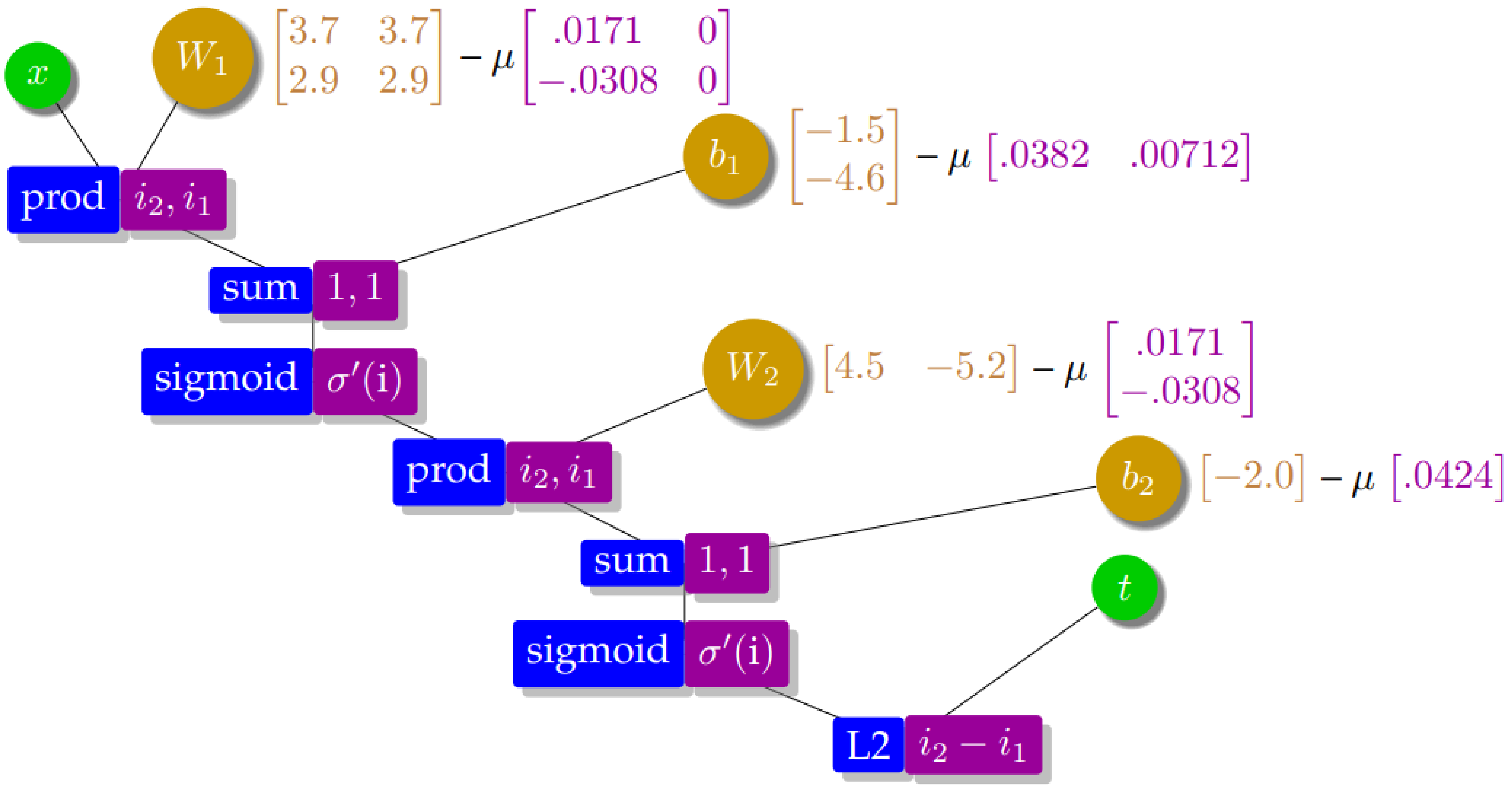
$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
$\begin{bmatrix}3.7\\2.9\end{bmatrix}$
prod
i_2, i_1
$\begin{bmatrix}-.0260\\-.0260\end{bmatrix}, \begin{bmatrix}0171 & 0\\-.0308 & 0\end{bmatrix}$
$\begin{bmatrix}2.2\\-1.6\end{bmatrix}$
sum
$1, 1$
$\begin{bmatrix}.0171\\-.0308\end{bmatrix}, \begin{bmatrix}.0171\\-.0308\end{bmatrix}$
$\begin{bmatrix}.900\\.17\end{bmatrix}$
sigmoid
$\sigma'(\text{i})$
$\begin{bmatrix}.0171\\-.0308\end{bmatrix}$
W_2
$[4.5 \quad -5.2]$
$[3.18]$
prod
i_2, i_1
$\begin{bmatrix}.191\\-.220\end{bmatrix}, [.0382 \quad .00712]$
b_2
$[-2.0]$
$[1.18]$
sum
$1, 1$
$[.0424], [.0424]$
t
$[1.0]$
$[.765]$
sigmoid
$\sigma'(\text{i})$
$[.180] \times [.235] = [.0424]$
$[.0277]$
L2
$i_2 - i_1$
$[.235]$

Gradients for parameter update



x
W_1
$\begin{bmatrix} 3.7 & 3.7 \\ 2.9 & 2.9 \end{bmatrix}$
b_1
$\begin{bmatrix} -1.5 \\ -4.6 \end{bmatrix}$
prod
i_2, i_1
$\begin{bmatrix} .0171 & 0 \\ -.0308 & 0 \end{bmatrix}$
sum
$1, 1$
$\begin{bmatrix} .0171 \\ -.0308 \end{bmatrix}$
sigmoid
$\sigma'(\mathrm{i})$
W_2
$\begin{bmatrix} 4.5 & -5.2 \end{bmatrix}$
prod
i_2, i_1
$\begin{bmatrix} .0382 & .00712 \end{bmatrix}$
b_2
$\begin{bmatrix} -2.0 \end{bmatrix}$
sum
$1, 1$
$\begin{bmatrix} .0424 \end{bmatrix}$
t
sigmoid
$\sigma'(\mathrm{i})$
L2
$i_2 - i_1$

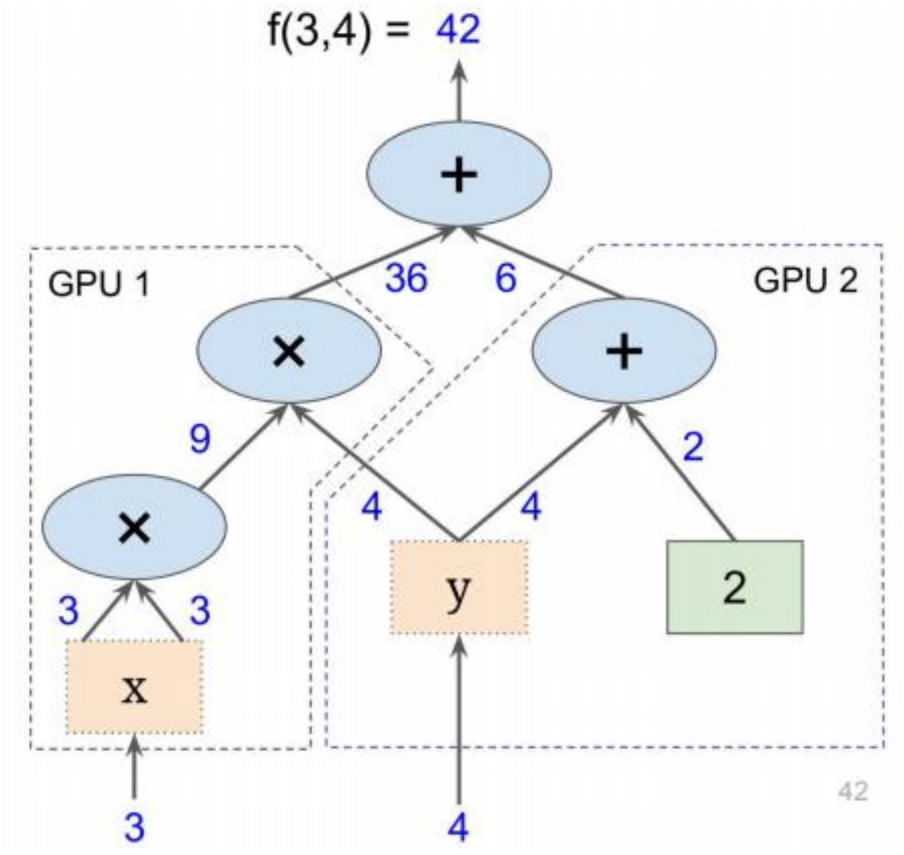
Parameter update



x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix} - \mu \begin{bmatrix}.0171 & 0\\-.0308 & 0\end{bmatrix}$
prod
i_2, i_1
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix} - \mu \begin{bmatrix}.0382 & .00712\end{bmatrix}$
sum
$1, 1$
sigmoid
$\sigma'(\text{i})$
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix} - \mu \begin{bmatrix}.0171\\-.0308\end{bmatrix}$
prod
i_2, i_1
b_2
$\begin{bmatrix}-2.0\end{bmatrix} - \mu \begin{bmatrix}.0424\end{bmatrix}$
sum
$1, 1$
t
sigmoid
$\sigma'(\text{i})$
L2
$i_2 - i_1$

Subgraphs

- Possible to break graphs into chunks and run the parallelly.



Why Graphs

- Save computation
- Break computation into small, differential pieces and facilitate auto differentiation
- Facilitate distributed computation
- Intuitive model visualization

Tensorflow

- DataFlow graph based computation model for building ML models.

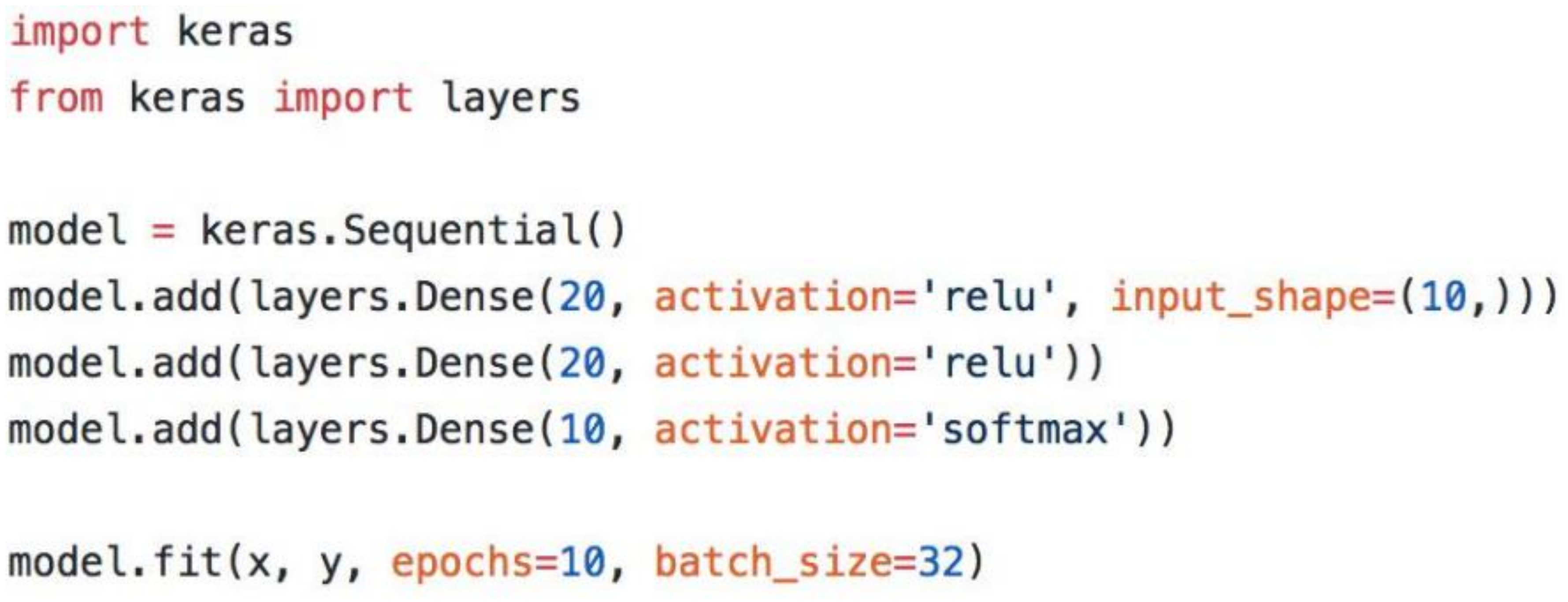
- Developed by google
- A Tensor is a n-dimensional array

- Graphs are built and run within a session which provides an environment for the graph to run

Keras

- Keras is high level API for programming models like Tensorflow
- Keras API Styles –
  - Sequential
    - Simple. Single input, output and sequential layer stacks
  - Functional API
    - Like Lego bricks
    - Multiple Input, output and arbitrary graph topology
  - Model Subclassing
    - Maximum flexibility
    - Larger potential error surface

Sequential API

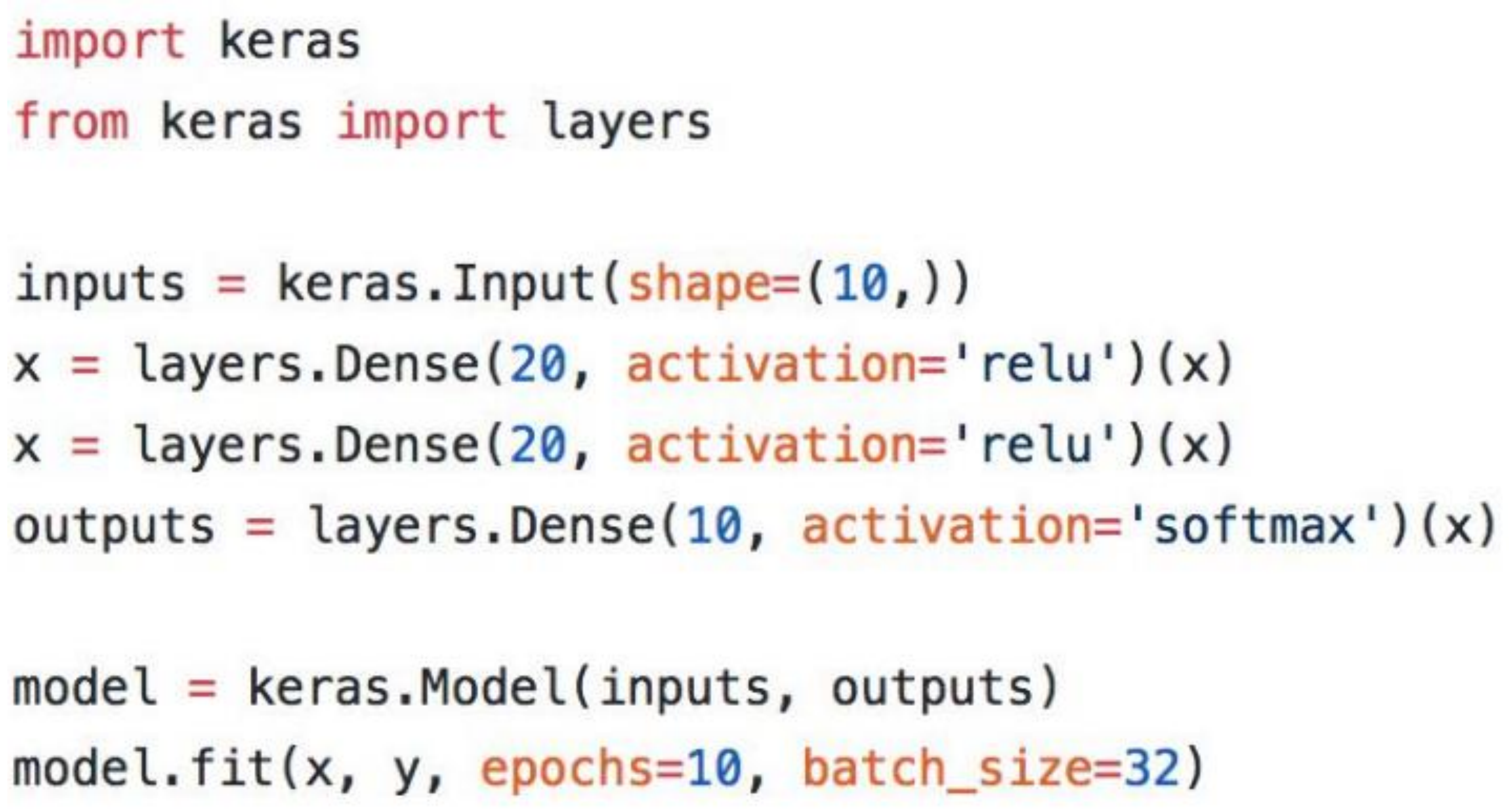


```python
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Functional API

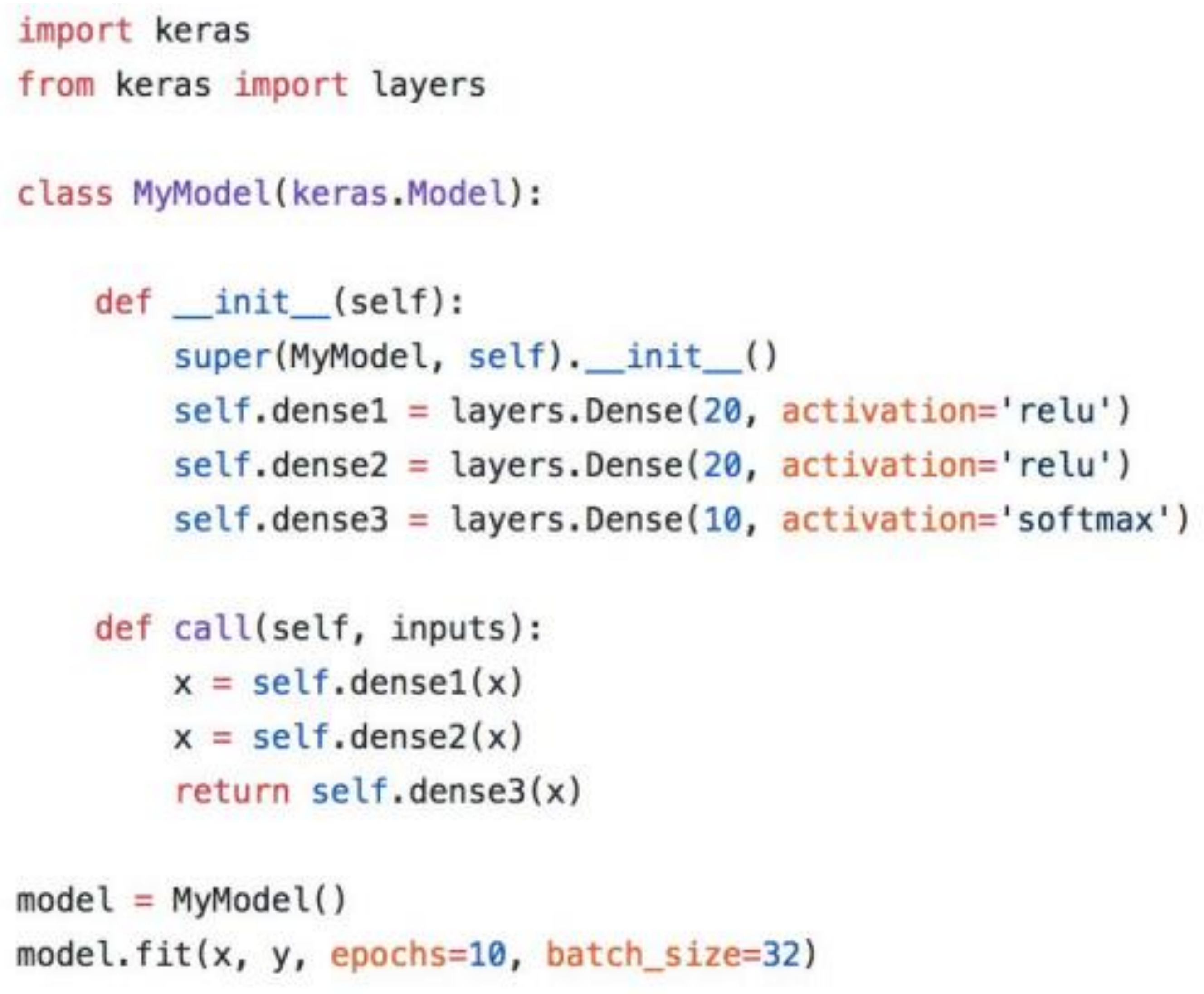


```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Model Subclassing



```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```