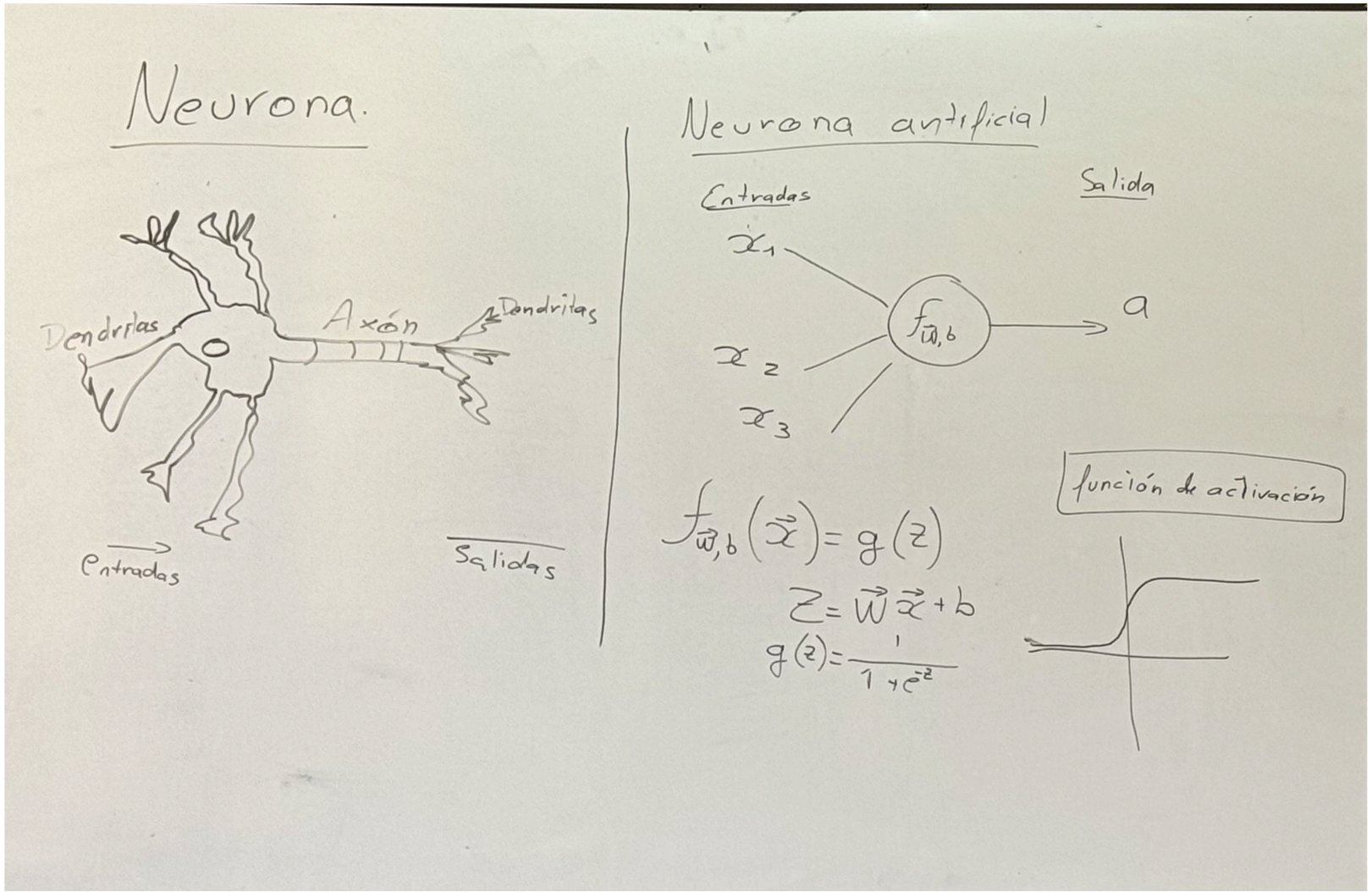
Introduction to Neural Networks

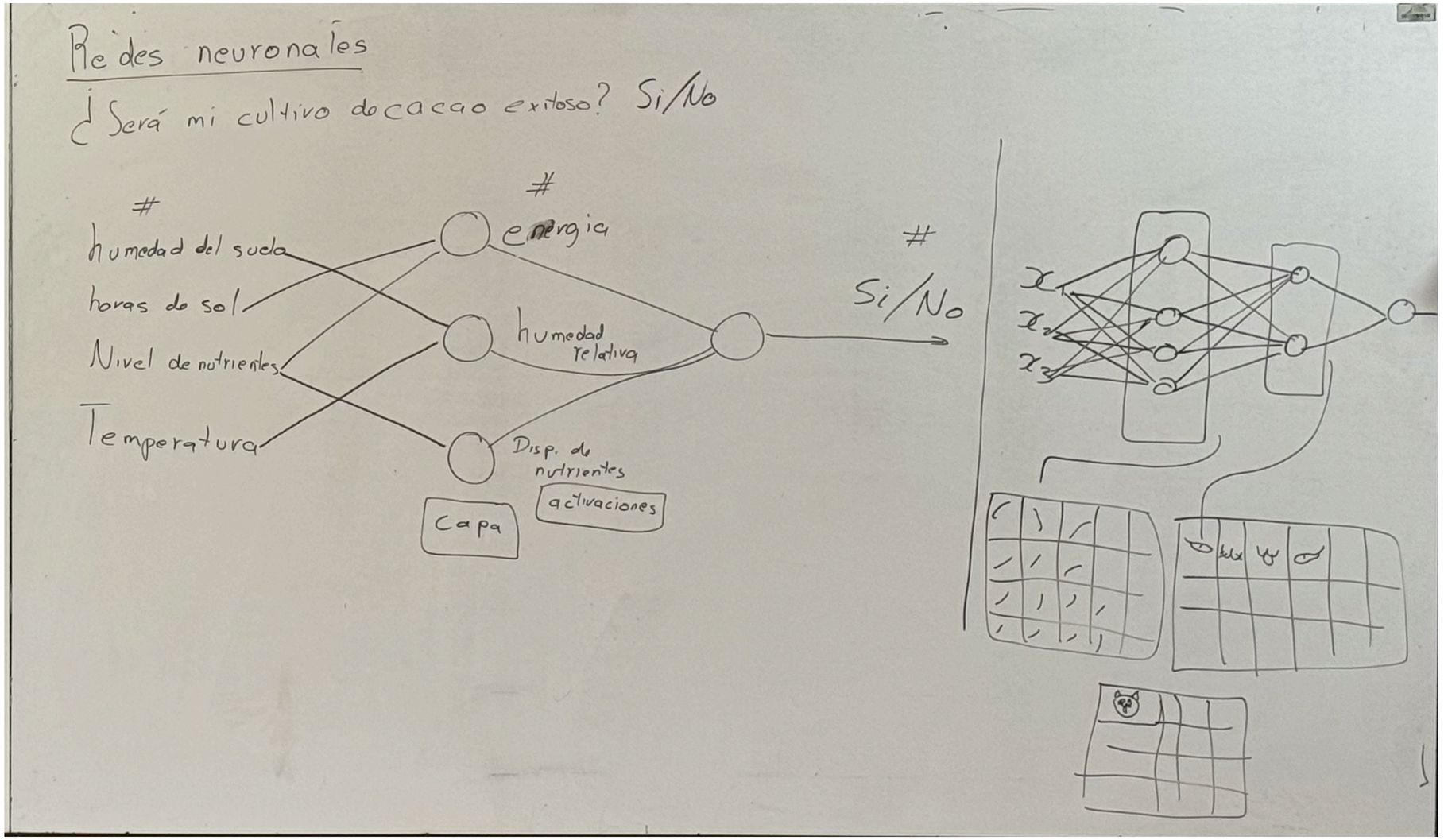
Luis Daniel Benavides Navarro, Ph.D

24-11-2025

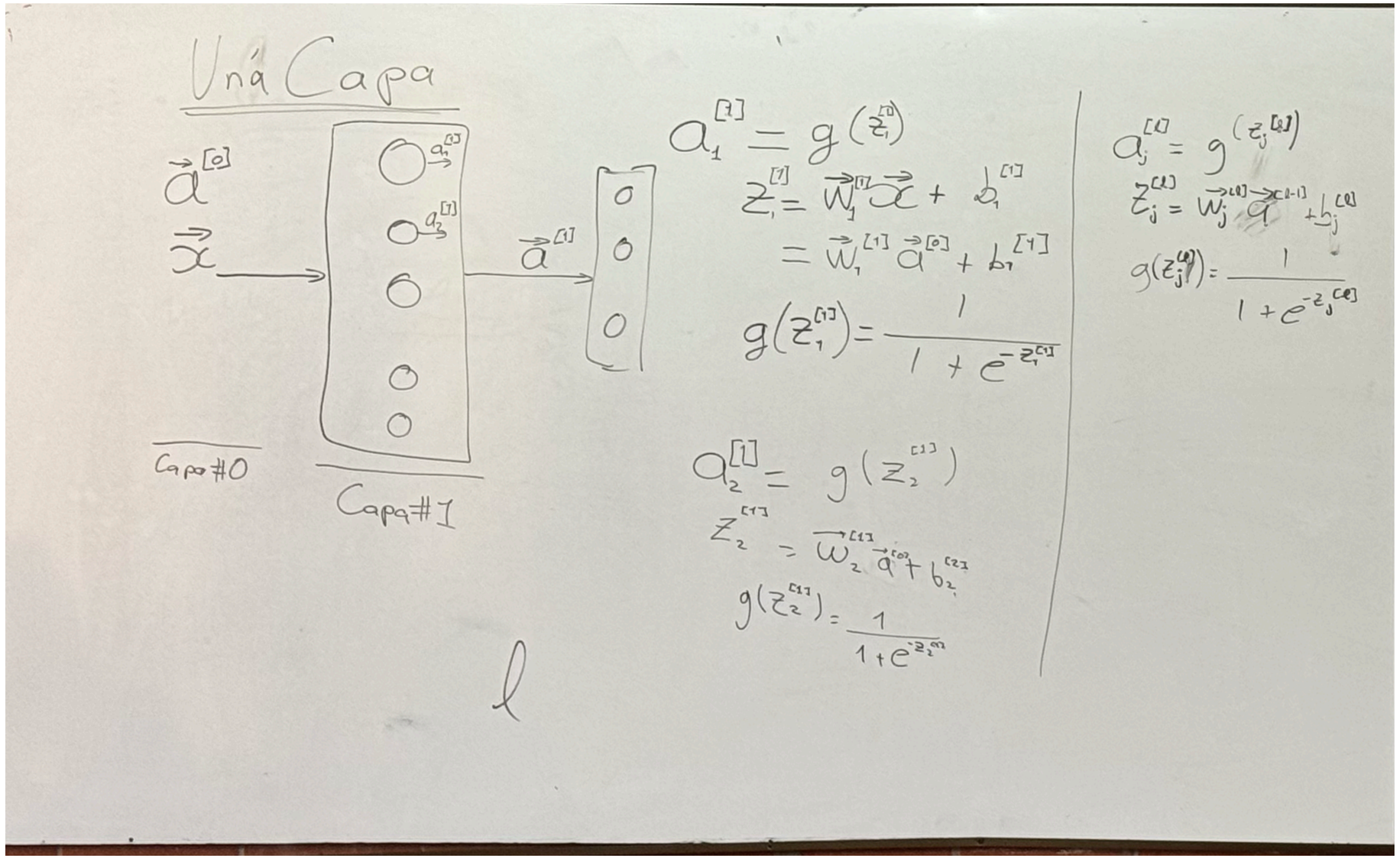
Neurons


Neurona.
Dendritas
Axón
Dendritas
Entradas
Salidas
Neurona artificial
Entradas
Salida
x_1
x_2
x_3
$f_{\vec{w},b}$
a
$f_{\vec{w},b}(\vec{x}) = g(z)$
$z = \vec{w}\vec{x} + b$
$g(z) = \frac{1}{1+e^{-z}}$
función de activación

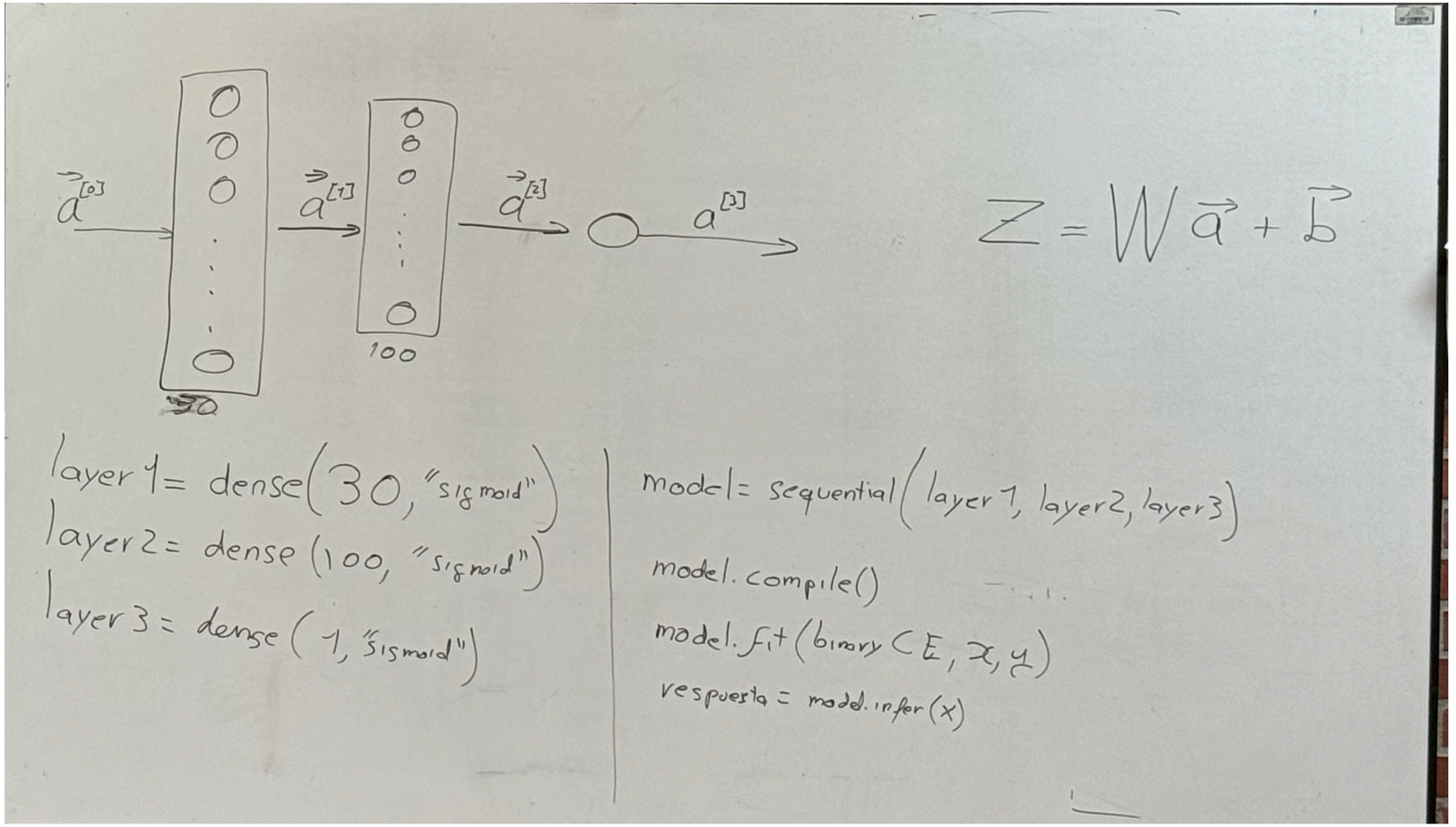
Neural Networks



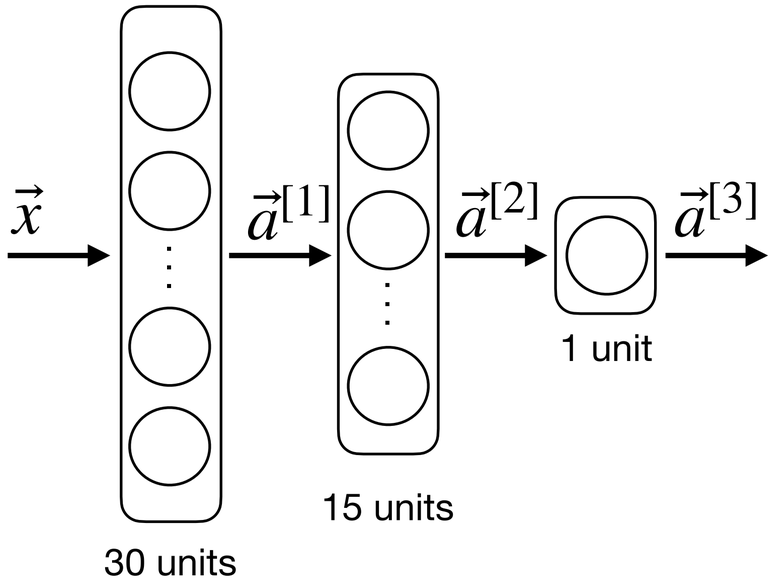
One Layer



Implementation



Implementation using Tensorflow




```
import tensorflow as tf
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model =  Sequential([
    Dense(30, activation='sigmoid'),
    Dense(15, activation='sigmoid')
    Dense(2, activation='sigmoid')
])

from tensorflow.keras.losses import BinaryCrossentropy
model.compile(loss=BinaryCrossentropy())

model.fit(X_train, y_train, epochs=100)
```

Given a set of m (x, y) samples

Model Training Steps

1. Define model: $f_{\vec{w},b}(\vec{x}) = ?$

2. Specify loss and cost functions:

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

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$

3. Train the model to minimize $J(\vec{w}, b)$.

Logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

$$L(f_{\vec{w},b}(\vec{x}), y) = -y * log(f_{\vec{w},b}(\vec{x})) - (1 - y) * log(1 - f_{\vec{w},b}(\vec{x}))$$

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$

$$w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$$

Gradien descent

$$b = b - \alpha \frac{\partial J}{\partial b}$$

Model Training Steps

1. Define model: $f_{\vec{w},b}(\vec{x}) = ?$

2. Specify lost and cost functions:

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

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$

3. Train the model to minimize $J(\vec{w}, b)$.

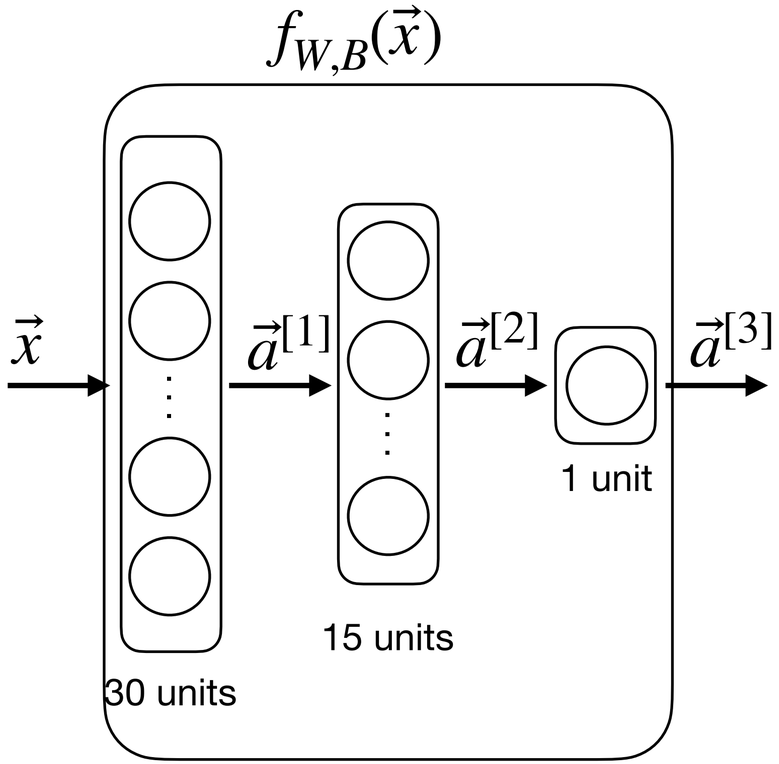
Neural Networks

```
model =  Sequential([
    Dense(30, activation='sigmoid'),
    Dense(15, activation='sigmoid')
    Dense(2, activation='sigmoid')
])
```

```
model.compile(loss=BinaryCrossentropy())
```

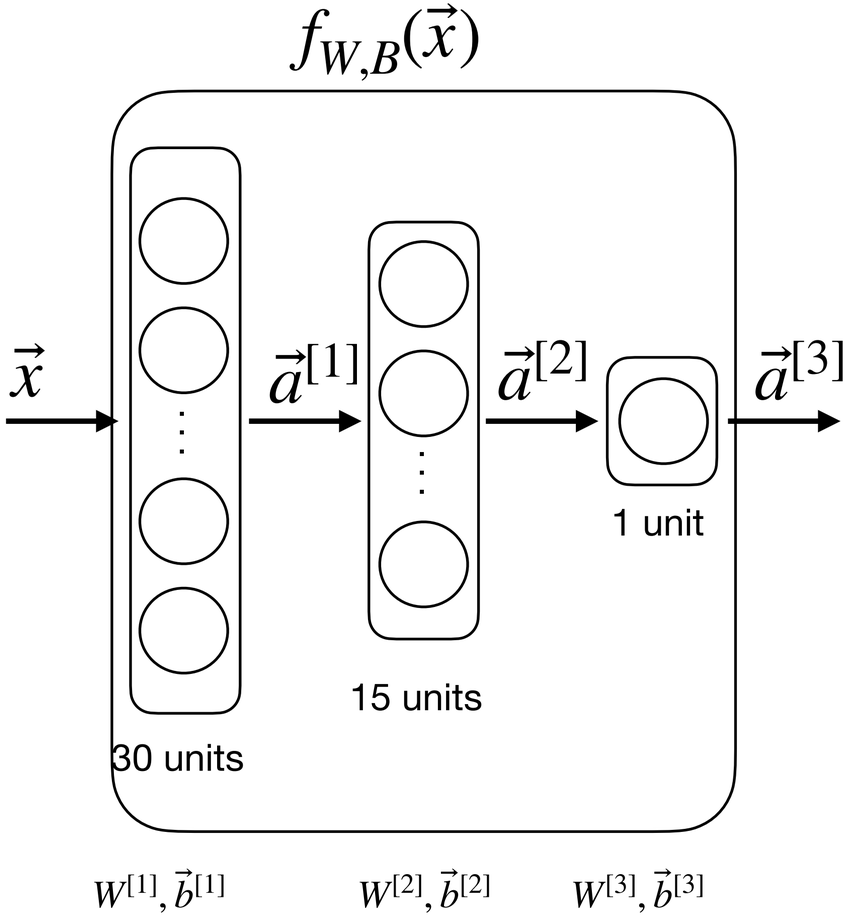
```
model.fit(X_train, y_train, epochs=100)
```

The model




```
model =  Sequential([
    Dense(30, activation='sigmoid'),
    Dense(15, activation='sigmoid')
    Dense(2, activation='sigmoid')
])
```

The cost function

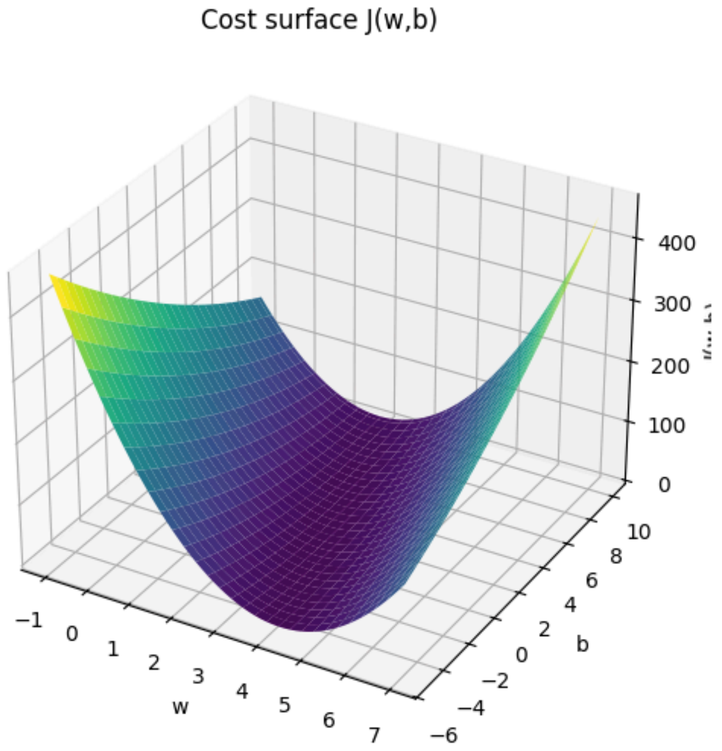


$$L(f_{W,B}(\vec{x}), y) = -y * log(f_{W,B}(\vec{x})) - (1-y) * log(1 - f_{W,B}(\vec{x}))$$

$$J(W,B) = \frac{1}{m} \sum_{i=1}^{m} L(f_{W,B}(\vec{x}^{(i)}), y^{(i)})$$

```
model.compile(loss=BinaryCrossentropy())
```

Gradient Descent



Repeat for all neurons on each layer*

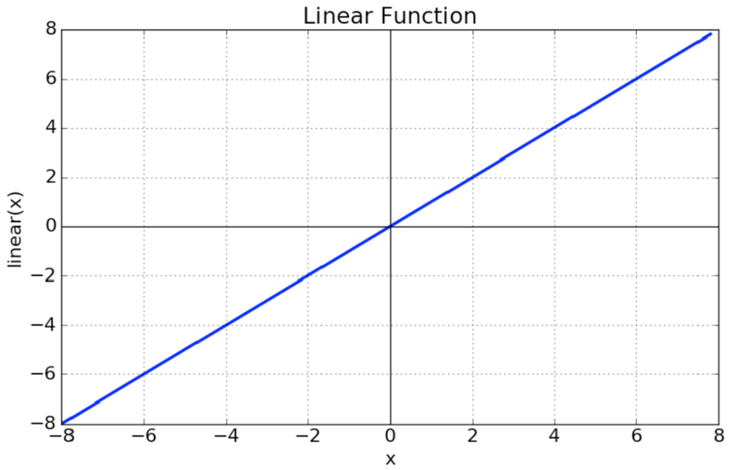
$$\overrightarrow{w}_j^{[l]} = \overrightarrow{w}_j^{[l]} - \alpha \frac{\partial J(W,B)}{\partial \overrightarrow{w}_j^{[l]}}$$

$$b_j^{[l]} = b_j^{[l]} - \alpha \frac{\partial J(W,B)}{\partial b_j^{[l]}}$$

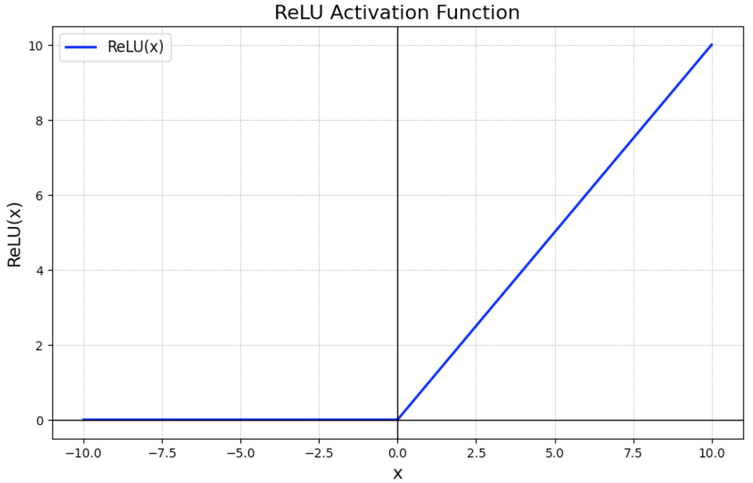
Compute derivatives using back propagation

Note: In each batch, all trainable parameters in the network are updated once.
The gradient is computed based only on the samples in that batch,
but the update is applied to every weight and bias, in every neuron, in every layer.

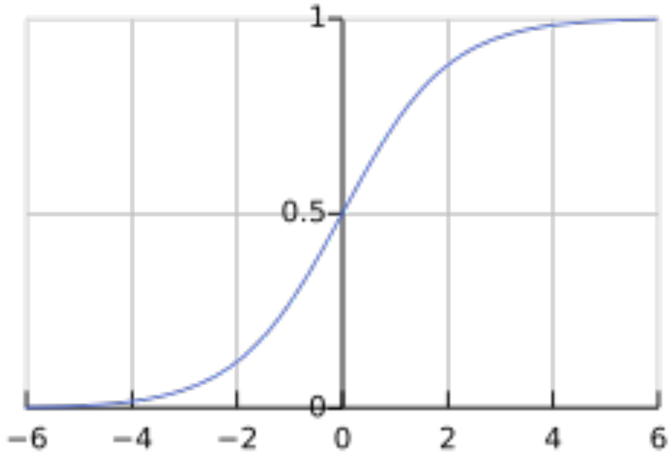
Activation functions



$$f(z) = z$$



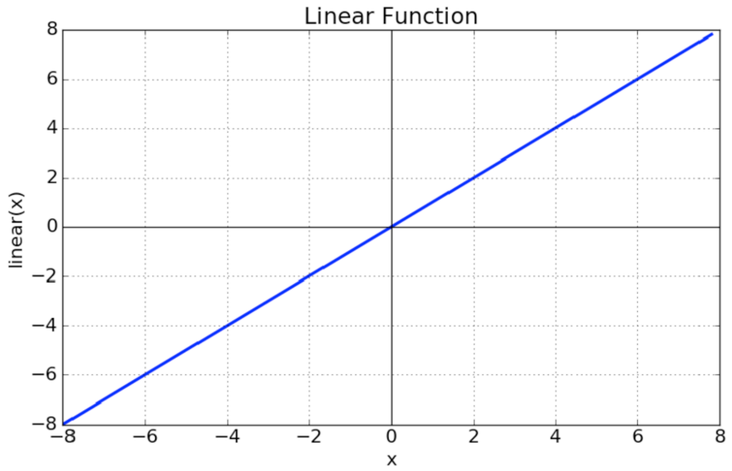
$$f(z) = max(0,z)$$



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

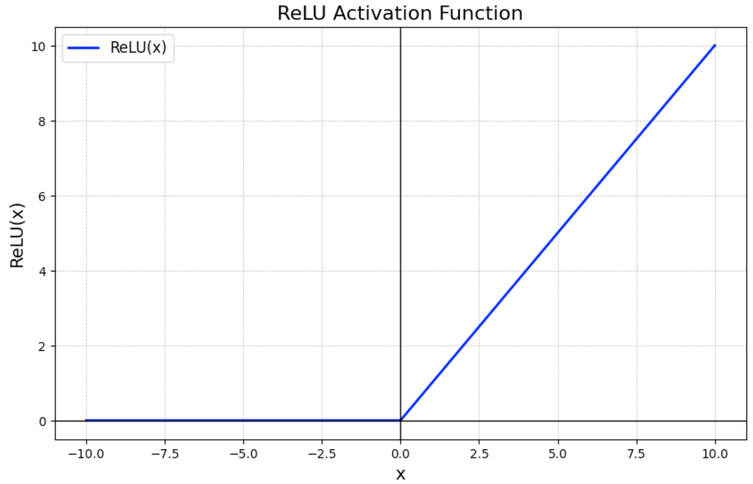
Choosing the activation function

For the **output** layer:



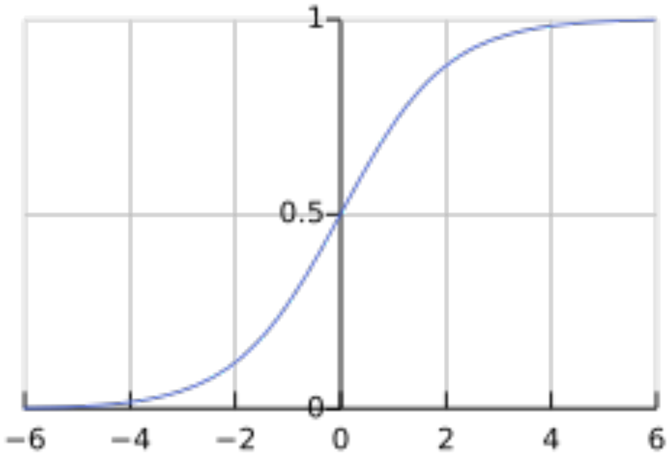
$$f(z) = z$$

Linear regression with positive and negative values



$$f(z) = max(0,z)$$

Regression with only positive values

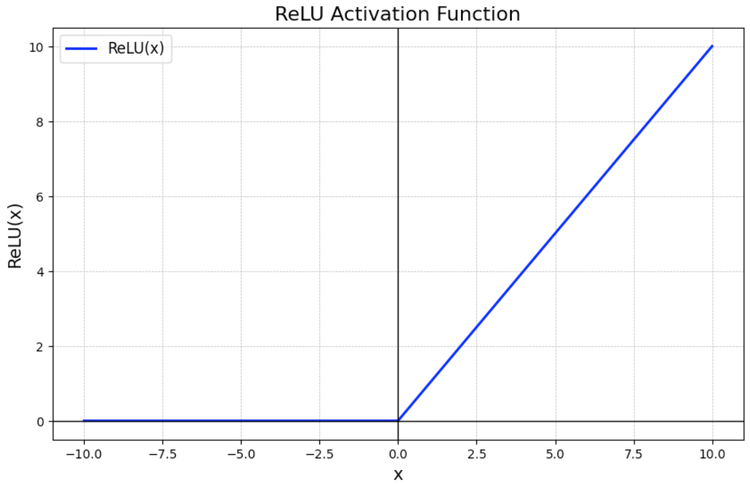


$$f(z) = \frac{1}{1 + e^{-z}}$$

Binary classification

Choosing the activation function

For the **hidden** layers:

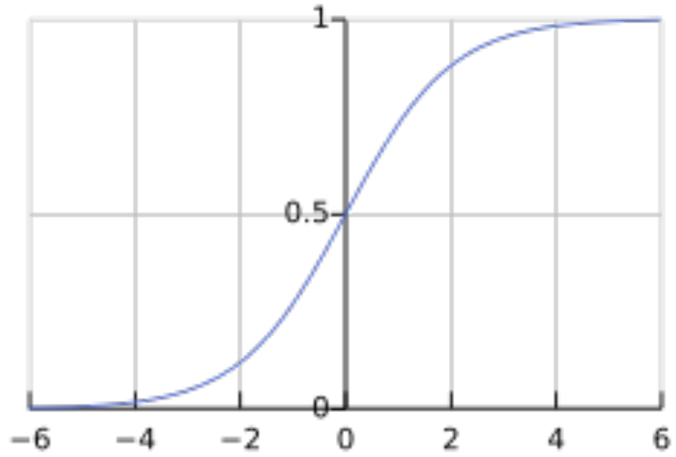


$$f(z) = max(0,z)$$

This is the most common choice

Faster to compute

The gradient descent is faster



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

Binary classification

Multiclass classification

Example:

Build a NN to detect hand written numbers in an image.

y can take more than one value

x=  y = 9

Softmax regression

Logistic regression

$$z = \vec{w} \cdot \vec{x} + b$$

$$a = g(z) = \frac{1}{1 - e^{-(\vec{w} \cdot \vec{x} + b)}} = P(y = 1 \,|\, \vec{x})$$

Example:

If $P(y = 1 \,|\, \vec{x}) = 0.75$

Then $P(y = 0 \,|\, \vec{x}) = 0.25$

Softmax regression with 4 outputs

$$z_1 = \vec{w}_1 \cdot \vec{x} + b_1 \qquad a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 1 \,|\, \vec{x})$$

$$z_2 = \vec{w}_2 \cdot \vec{x} + b_2 \qquad a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 2 \,|\, \vec{x})$$

$$z_3 = \vec{w}_3 \cdot \vec{x} + b_3 \qquad a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 3 \,|\, \vec{x})$$

$$z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \qquad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 4 \,|\, \vec{x})$$

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

Cost function

Softmax regression with 4 outputs

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \ldots + e^{z_N}} = P(y = 1 \,|\, \vec{x})$$

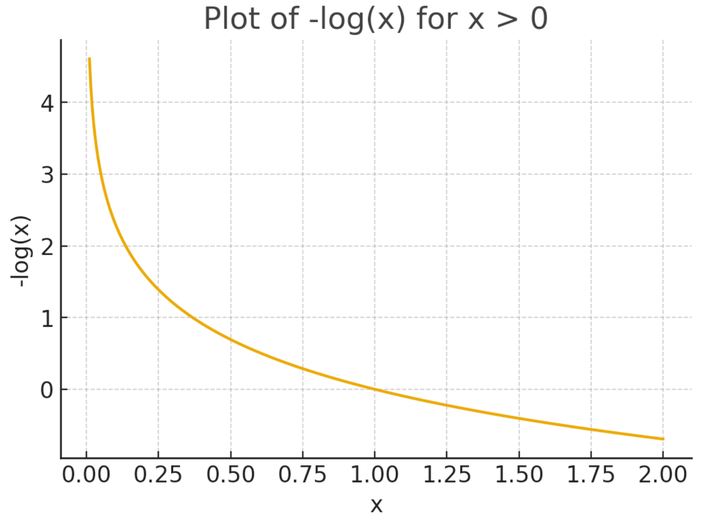
$$\vdots$$

$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \ldots + e^{z_N}} = P(y = N \,|\, \vec{x})$$

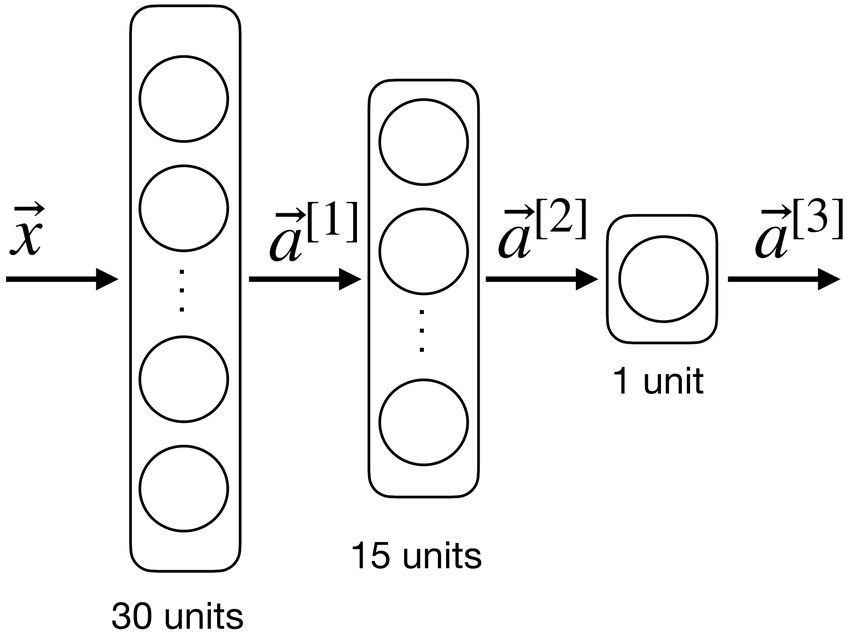
$$z_j = \vec{w}_j \cdot \vec{x} + b_j$$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y = j \,|\, \vec{x})$$

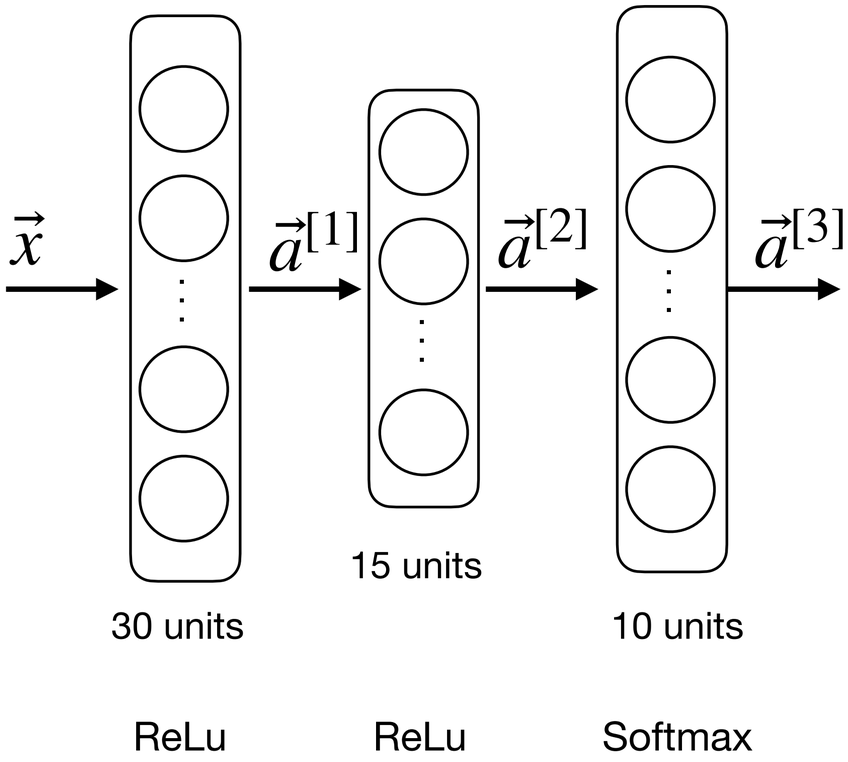
$$\text{loss}(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y = 1, \\ -\log a_2 & \text{if } y = 2, \\ \vdots & \\ -\log a_N & \text{if } y = N. \end{cases}$$



Neural Network with Softmax



Neural Network with Softmax



$$z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}$$

$$a_1^{[3]} = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + e^{z_2^{[3]}} + \ldots + e^{z_{10}^{[3]}}} = P(y = 1 \mid \vec{x})$$

$$\vdots \qquad \vdots$$

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

$$a_{10}^{[3]} = \frac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + e^{z_2^{[3]}} + \ldots + e^{z_{10}^{[3]}}} = P(y = 10 \mid \vec{x})$$

Implementation

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.losses import SparseCategoricalCrossentropy

model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation=‘softmax')
])

model.compile(loss=SparseCategoricalCrossentropy())

model.fit(X_train, y_train, epochs=100)
```

Code is very clear but is not efficient. Do not use it!

If you give a more explicit expression for the loss, Tensorflow can reorganize expressions and optimize numerical computations

Improved Implementation

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.losses import SparseCategoricalCrossentropy

model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation=‘linear')
])

model.compile(loss=SparseCategoricalCrossentropy(from_logits=true))

model.fit(X_train, y_train, epochs=100)

logit = model(X) #The model is now outputting z

f_x = tf.nn.sigmoid(logit)
```

Multiclass vs Multilabel

- Multi-Class: You're assigning one exclusive class per sample. Like categorizing an email as "Spam," "Promotion," or "Personal" — pick just one.

- Multi-Label: You're assigning multiple labels (any number, including zero or all) per sample. Like tagging a photo with "beach," "sunset," and "family" — all three can apply independently.

- Use softmax for multiclass

- Use sigmoid for multilabel

Optimizing gradient descent

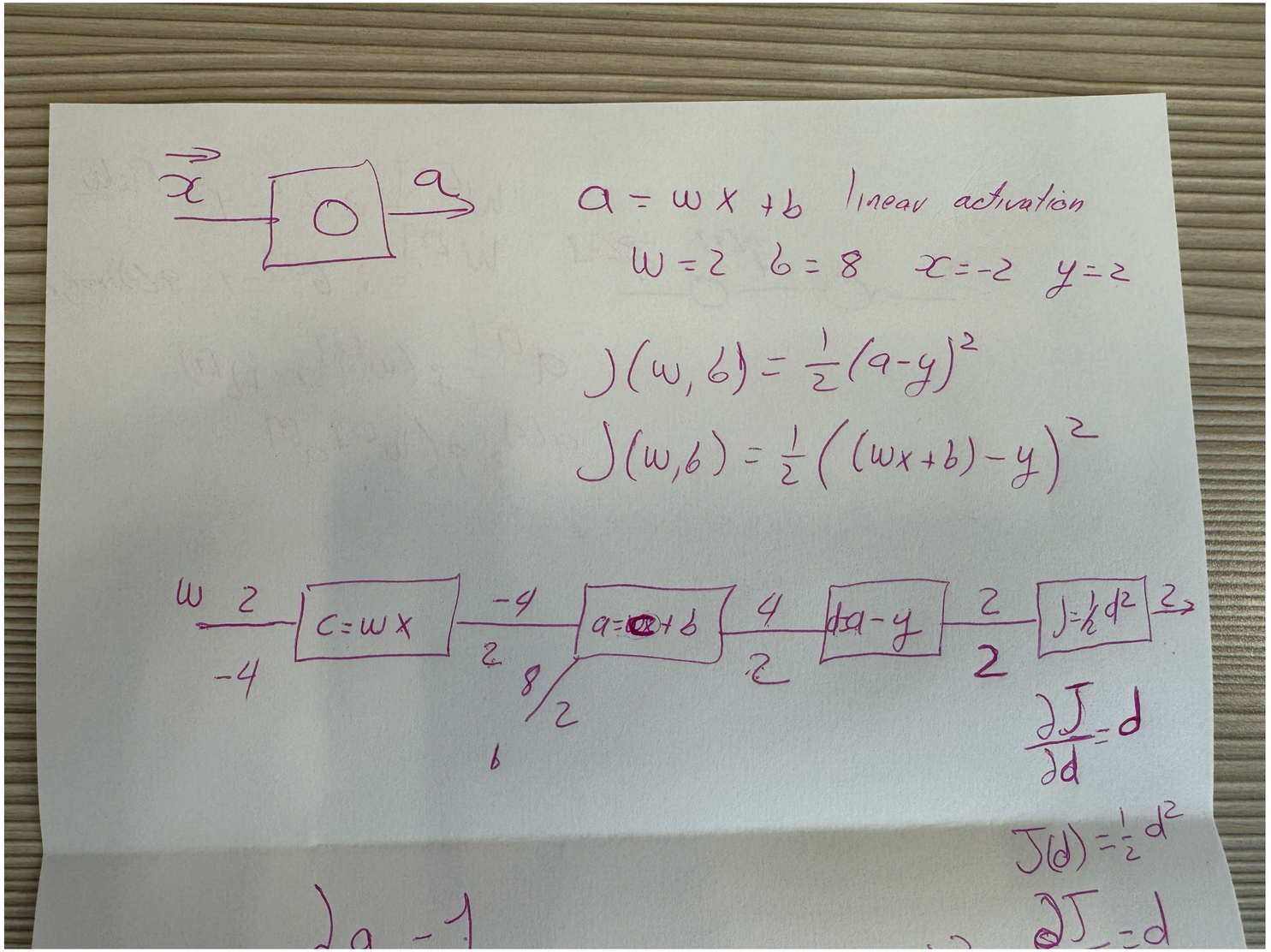
Adam Optimizer: Smart gradient descent, Auto-adjusts learning speed per parameter.

```
# Build the model
model = Sequential([
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=10, activation='linear')  # Linear for logits (with from_logits=True)
])

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=1e-3),
    loss=SparseCategoricalCrossentropy(from_logits=True)
)

# Fit the model (assuming X and Y are your data)
model.fit(X, Y, epochs=100)
```

Layer types and architectures

- Convolutional layers: Each neuron only looks at part of the previous layer output
    - Faster computation
    - Need less training data (less prone to overfitting)
- Use cases
    - Grid based data: images, videos.
- We are looking for new architectures: transformers

Backpropagation



Thank you!