# Deep Learning

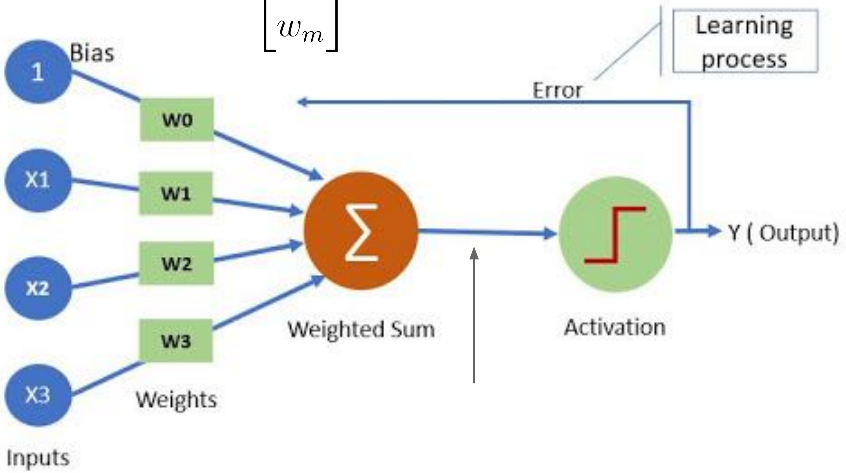
Alberto Ezpondaburu

# 1.1

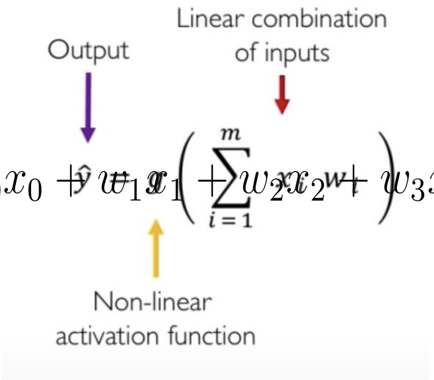
## Introduction To Deep Learning

# Perceptron: Forward Propagation

$$\mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_m \end{bmatrix}$$



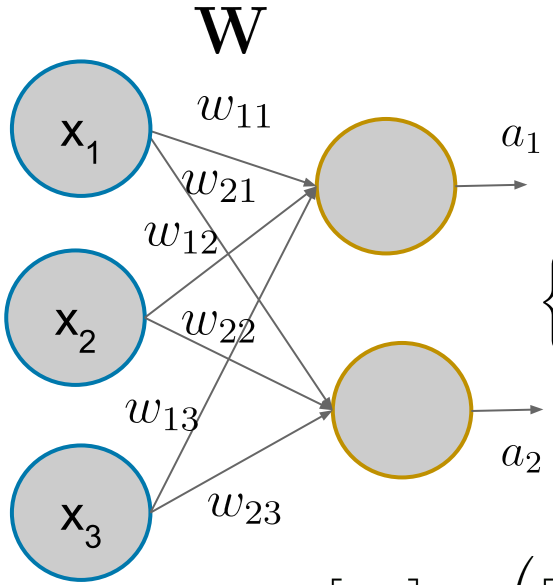

$$y = g(w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3) = g(\mathbf{w}^T\mathbf{x})$$




$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_m \end{bmatrix}$$

**Single Layer Perceptron**

# Perceptron: Multi Output



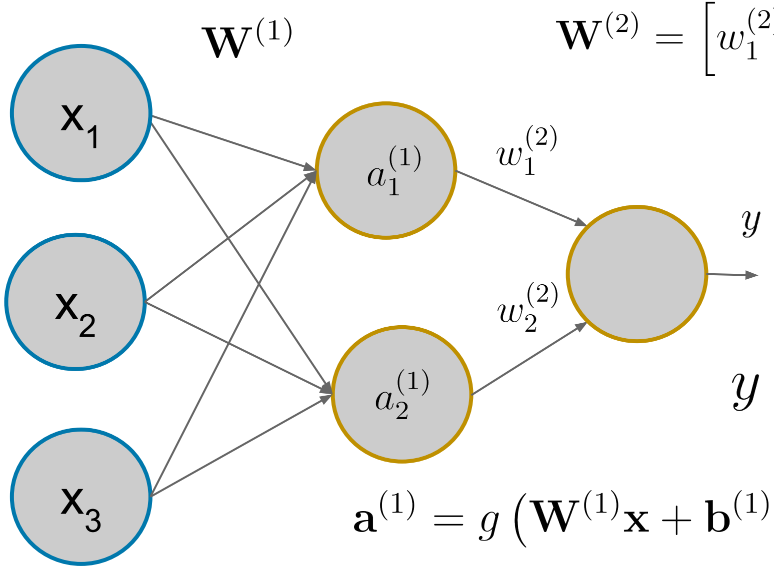

$$a_1 = g(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1)$$

$$\begin{cases} a_1 = g(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1) \\ a_2 = g(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2) \end{cases}$$

$$a_2 = g(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2)$$

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = g\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & \omega_{23} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right) = g\Big(\mathbf{Wx} + \mathbf{b}\Big)$$

# Single Layer Neural Network



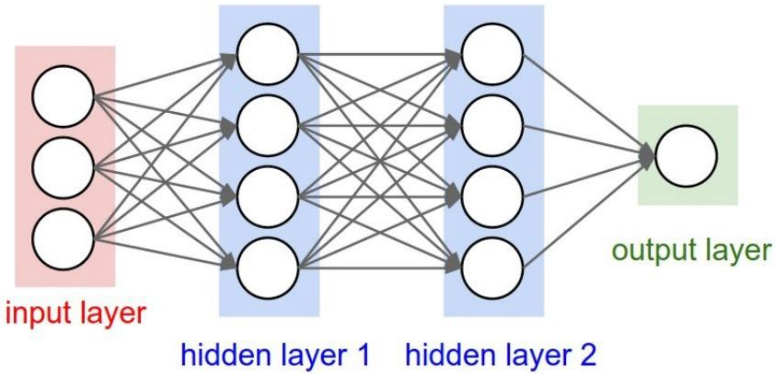

$$\mathbf{W}^{(2)} = \left[ w_1^{(2)} w_2^{(2)} \right]$$

$$y = g\left(\mathbf{W}^{(2)}\mathbf{a}^{(1)} + b^{(2)}\right) = g\left(w_1^{(2)}a_1^{(1)} + w_2^{(2)}a_2^{(1)} + b^{(2)}\right)$$

$$y = g\left(\mathbf{W}^{(2)} g\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right)$$

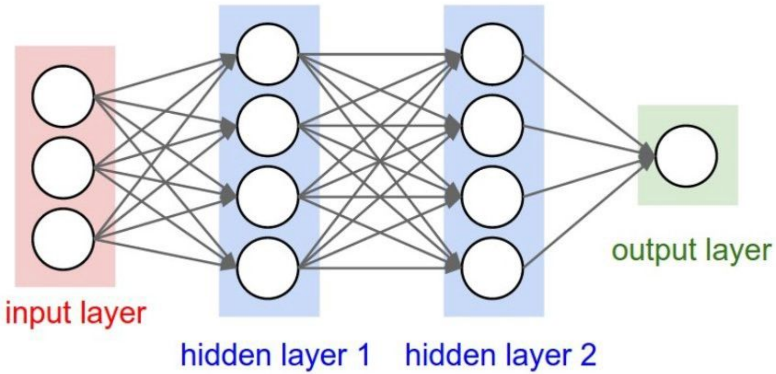
$$\mathbf{a}^{(1)} = g\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

# Multi-Layer Perceptron MLP




```
model = keras.Sequential([
    layers.Dense(4, input_shape=(3, ), activation="relu"),
    layers.Dense(4, activation="relu"),
    layers.Dense(1, activation="sigmoid"),
])
model.compile(loss="binary_crossentropy", optimizer="adam")
model.summary()
```

# Multi-Layer Perceptron MLP




```
inputs = tf.keras.Input(shape=(3, ))
l_1 = layers.Dense(4, activation='relu')(inputs)
l_2 = layers.Dense(4, activation='relu')(l_1)
outputs = layers.Dense(1, activation='sigmoid')(l_2)
model = keras.Model(inputs=inputs, outputs=outputs)
```

# Cost Functions

- Regression:
  - **Mean squared error (MSE):** $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$

```
model.compile(loss='mse')
```

  - **Mean absolute error (MAE)** $\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$

```
model.compile(loss='mae')
```

- Classification:
  - **Binary Cross-Entropy (log-loss):** $\mathcal{L}(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$

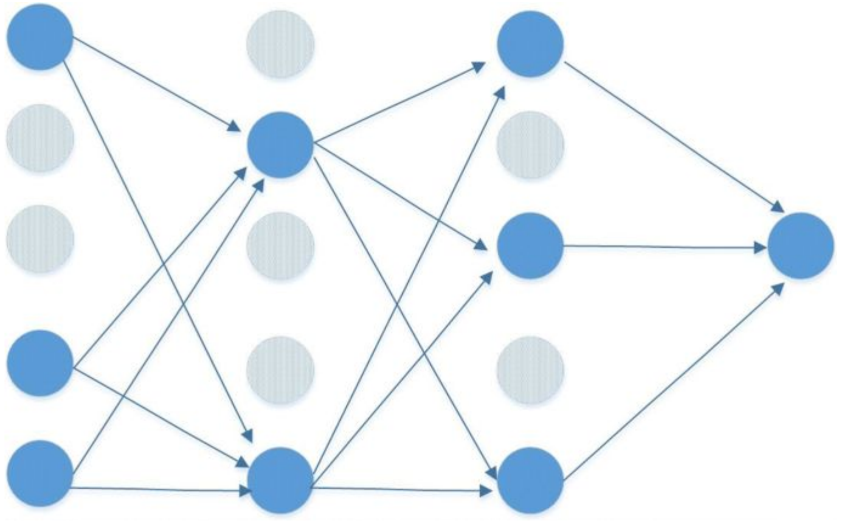
```
model.compile(loss='binary_crossentropy')
```

  - **Categorical Cross-Entropy:** $\mathcal{L}(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \cdot \log(\hat{y_{ik}})$

```
model.compile( loss='categorical_crossentropy')
```

```
model.compile( loss='sparse_categorical_crossentropy')
```

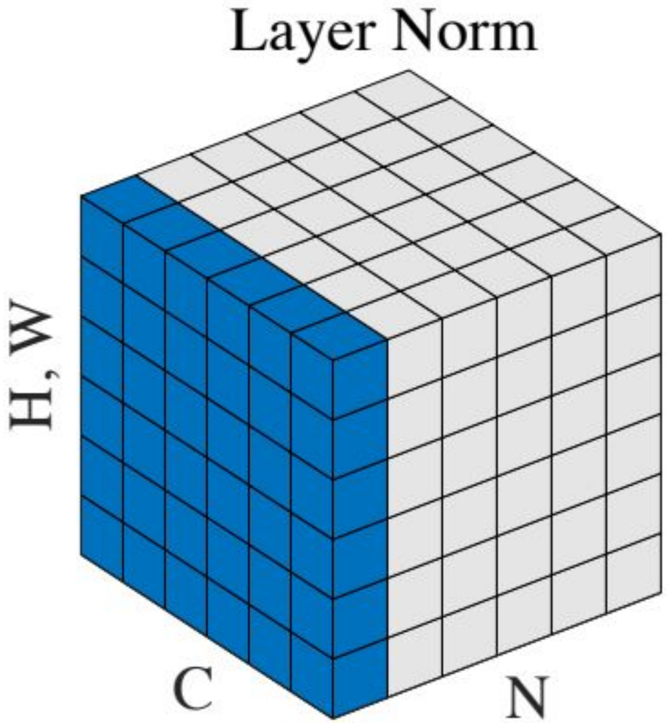
# Regularization: Dropout



- During training, randomly set some activations to 0 with probability p.
- Not in prediction.

```
from tensorflow.keras.layers import
Dropout
model = Sequential()
model.add(Dense(60, activation='relu',
input_shape=input_shape))
model.add(Dropout(0.2))
model.add(Dense(30, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

# Regularization: Layer normalization

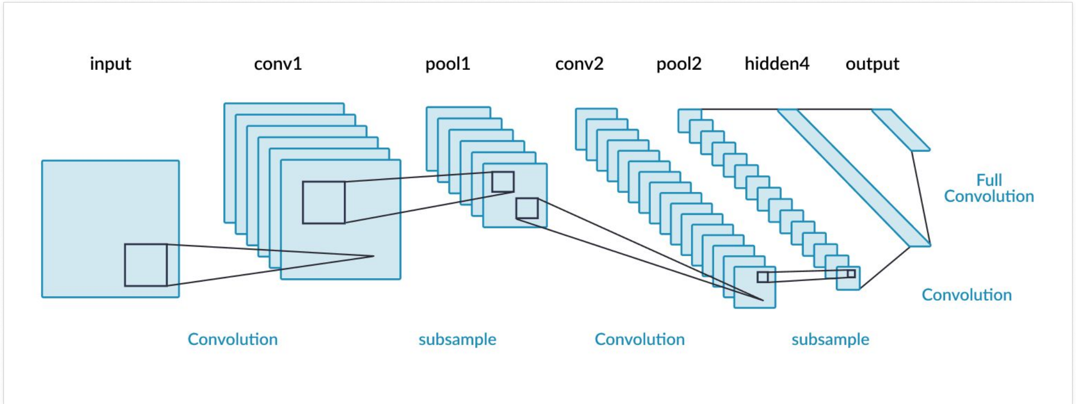



- Normalizes the activations along the feature direction instead of mini-batch direction
- BN normalizes each feature independently across the mini-batch. LN normalizes each of the inputs in the batch independently across all features.

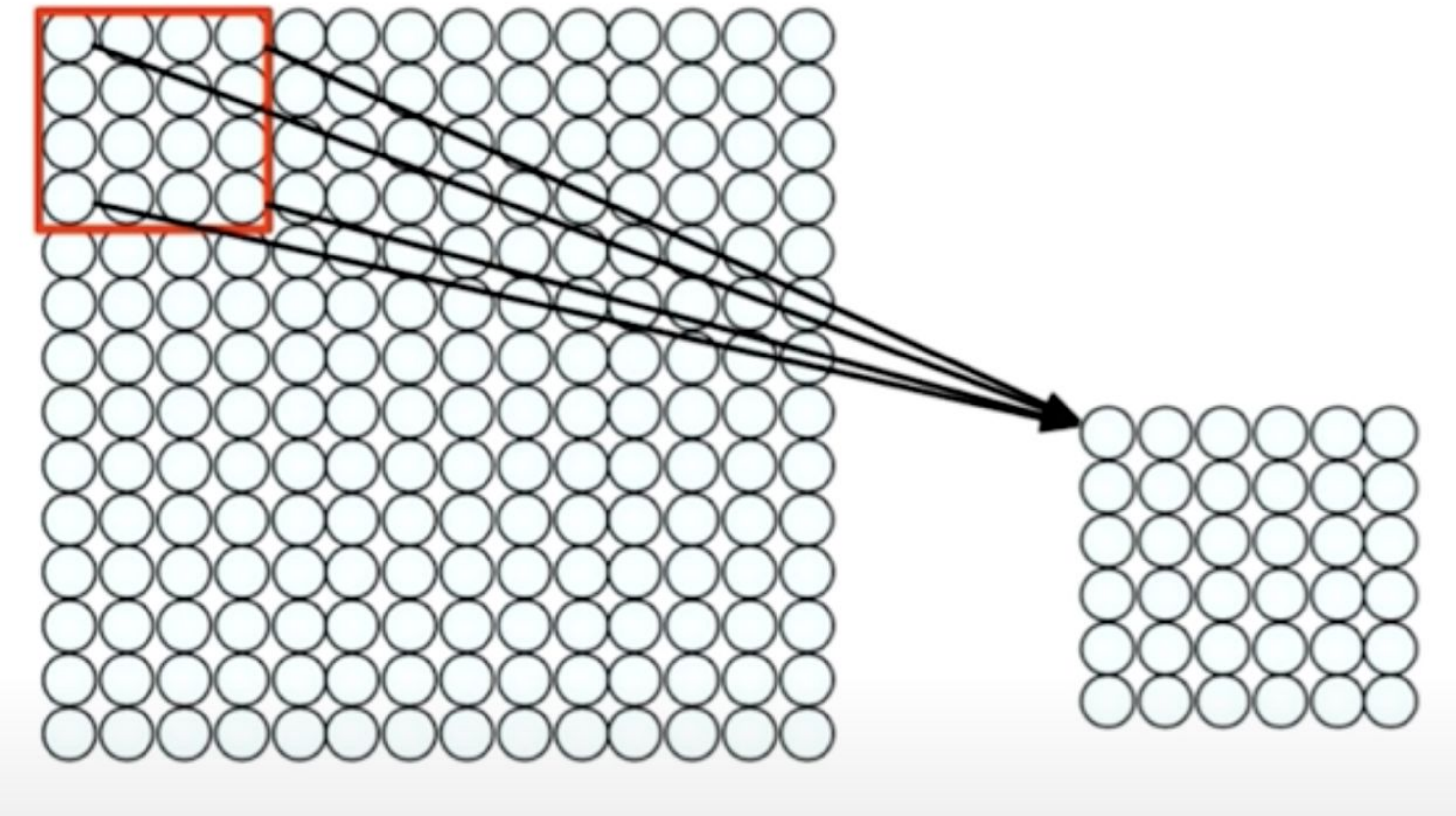
```
from keras.layers.normalization import
BatchNormalization
model = Sequential()
model.add(Dense(64, input_shape))
model.add(LayerNormalization(axis=1))
model.add(Activation('tanh'))
model.add(Dropout(0.5))
```

https://arxiv.org/abs/1607.06450

# CNN


input
conv1
pool1
conv2
pool2
hidden4
output
Full Convolution
Convolution
Convolution
subsample
Convolution
subsample

# CNN

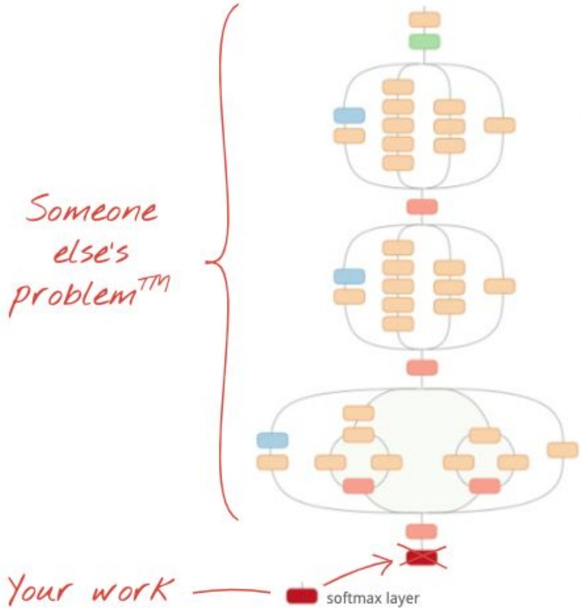


# CNN

```python
image_size = (128,128,3)
inputs = tf.keras.Input(shape=image_size)

# Conv Layer 1
conv_1 = layers.Conv2D(4, 3, padding='valid', activation='relu')(inputs)
pool_1 = layers.MaxPooling2D(pool_size=(2, 2), name='pool_1')(conv_1)

# Conv Layer 2
conv_2 = layers.Conv2D(4, 3, padding='same', activation='relu')(pool_1)
pool_2 = layers.MaxPooling2D(pool_size=(2, 2))(conv_2)

# Flattening
flat = layers.Flatten(name='flatten')(pool_2)

# Fully-connected
dense = layers.Dense(64, activation='relu')(flat)
outputs = layers.Dense(5, activation='softmax')(dense)

model = keras.Model(inputs=inputs, outputs=outputs)
```

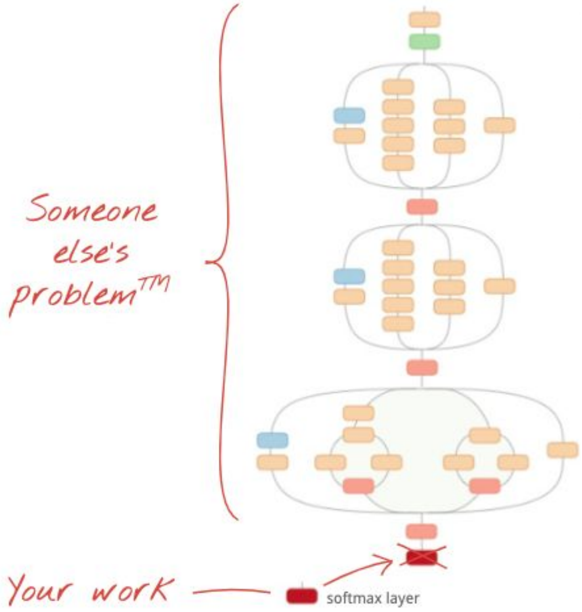
# CNN: Transfer Learning

```
from tensorflow.keras.applications import MobileNetV2
pretrained_model = MobileNetV2(input_shape=image_size,
                               include_top=False)
pretrained_model.trainable = False

model_tl = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(5, activation='softmax')
])
```



# CNN: Fine tuning

```
from tensorflow.keras.applications import MobileNetV2
pretrained_model = MobileNetV2(input_shape=image_size,
                               include_top=False)
pretrained_model.trainable = True

model_tl = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(5, activation='softmax')
])
```



ntic
master
revolucionamos la comunicación

UNIVERSIDAD
COMPLUTENSE
DE MADRID