Neural Network

Cheat Sheet

Charly Alizadeh

Most of the example comes from the documentation of keras. This is just a more convenient way to retrieve the essential informations needed for the exam. In the example I assume that keras is imported by `from tensorflow import keras`

Create a model

Basic Syntax

```
from keras import Sequential
from keras.layers import Dense

model = keras.Sequential([
    Dense(10, activation='relu'),
    Dense(5, activation='sigmoid')
])
```

```
from keras import Sequential

model = keras.Sequential()
model.add(Dense(10, activation='relu'))
model.add(Dense(5, activation='sigmoid'))
```

Layers

Layer type

```
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

dense = Dense(
    10, # Number of neurons
    acitvation='relu', # Activation function
    use_bias=True, # Whether to use bias or not
    kernel_initializer="glorot_uniform", # The statistic distribution used to initialize the weights
    bias_initializer="zeros", # The statistic distribution used to initialize the bias
    kernel_regularizer=None, # Regularization method used to apply penalties on weights
    bias_regularizer=None, # Regularization method used to apply penalties on bias
    activity_regularizer=None, # Regularization method used to apply penalties on the outputs
    kernel_constraint=None, # Constraint applied on the weights matrix
    bias_constraint=None, # Constraint applied on the bias
)
```

Dense

```
from keras.layers import Flatten

flatten = Flatten()
```

Flatten

```
from keras.layers import Conv2D

conv2D = Conv2D(
    filters=65, # Number of filters
    kernel_size=3 # Dimension of the kernels/filters. Can also be a tuple or list of 2 integers
    strides=1 # Stride of the convolution. Can also be a tuple or a list of 2 integers
    padding="valid" # Padding used in the convolution. Either "valid" or "same"
    data_format=None, # Format of the input. Either "channels_last" or "channels_first" If None it will
    dilation_rate=(1, 1), # Dilation rate of the filters.
    activation=None, # Activation function
    use_bias=True, # Whether to use bias or not
    kernel_initializer="glorot_uniform", # The statistics distribution used to initialize the weights
    bias_initializer="zeros", # The statistics distribution used to initialize the bias
    kernel_regularizer=None, # Regularization method used to apply penalties on weights
    bias_regularizer=None, # Regularization method used to apply penalties on bias
    activity_regularizer=None, # Regularization method used to apply penalties on bias
    kernel_constraint=None, # Constraint applied on the weights matrix
    bias_constraint=None # Constraint applied on the bias
)
```

Conv2D

```
from keras.layers import MaxPooling2D

max_pooling2D = MaxPooling2D(
    pool_size=2, # Pooling window
    strides=2, # Strides of the pooling window
    padding="valid", # Padding used in the convolution. Either "valid" or "same"
    data_format=None # Format of the input. Either "channels_last" or "channels_first" If None it will
)
```

MaxPooling2D

```
from keras.layers import AveragePooling2D

average_pooling2D = AveragePooling2D(
    pool_size=2, # Pooling window
    strides=2, # Strides of the pooling window
    padding="valid", # Padding used in the convolution. Either "valid" or "same"
    data_format=None # Format of the input. Either "channels_last" or "channels_first" If None it will
)
```

AveragePooling2D

```
from keras.layers import GlobalMaxPooling2D

global_max_pooling2D = GlobalMaxPooling2D(
    data_format=None # Format of the input. Either "channels_last" or "channels_first" If None it will
)
```

GlobalMaxPooling2D

```
from keras.layers import GlobalAveragePooling2D

global_average_pooling2D = GlobalAveragePooling2D(
    data_format=None # Format of the input. Either "channels_last" or "channels_first" If None it will
)
```

GlobalAveragePooling2D

Regularization

```
from keras.layers import Dense
from keras.regularizers import l1

dense = Dense(
    64,
    kernel_regularizer=l1(l1=0.01),
    activation='relu'
)
```

L1

```
from keras.layers import Dense
from keras.regularizers import l2

dense = Dense(
    64,
    kernel_regularizer=l2(l2=0.01),
    activation='relu'
)
```

L2

```
from keras.layers import Dense
from keras.regularizers import l1_l2

dense = Dense(
    64,
    kernel_regularizer=l1_l2(l1=0.01, l2=0.01),
    activation='relu'
)
```

L1_L2

```
from keras.layers import Dense
from keras.constraints import MaxNorm

dense = Dense(
    64,
    kernel_constraints=MaxNorm(max_value=2, axis=0),
    activation='relu'
    )
```

Max norm constraints

```
from keras import Sequential
from keras.layers import Dense, Dropout

model = Sequential([
    Dense(10, activation='relu'),
    Dropout(rate=0.5),
    Dense(1)
])
```

Dropout

Compile the model

The optimizers

SGD

```
from keras.optimizers import SGD

sgd = SGD(
    learning_rate=0.01, # Learning rate of the gradient descent
    momentum=0, # Momentum coefficient. If 0 no momentum is applied
    nesterov=False, # Wether to use Nesterov momentum: https://youtu.be/LdkkZglLZOQ
    )
```

RMSprop

```
from keras.optimizers import RMSprop

rmsprop = RMSprop(
    learning_rate=0.001, # Learning rate of the gradient descent
    rho=0.9, # Factor applied to the olds gradients accumulation
    momentum=0.0 # Momentum coefficient
    epsilon=1e-07 # Small value for numerical stability (avoiding diving by 0)
    centered=False # If True, gradients are normalized by the estimated variance of the gradient; if Fa
)
```

Adam

```
from keras.optimizers import Adam

adam = Adam(
    learning_rate=0.01, # Learning rate of the gradient descent
    beta_1=0.9, # Exponential decay rate for the 1st momentum estimates
    beta_2=0.999, # Exponential decay rate for the 1st momentum estimates
    epsilon=1e-07, # Small value for numerical stability (avoiding diving by 0)
    amsgrad=False # Wether to apply AMSGrad variant of the Adam optimizer: https://arxiv.org/abs/1904.0
)
```

Loss functions

- Binary Classification
  - Binay Cross-Entropy: `"binary_crossentropy"`
  - Hinge Loss: `"hinge"`
  - Squared Hinge Loss: `"squared_hinge"`
- Multi-Class Classification
  - Mutli-Class Cross-Entropy Loss: `"categorical_crossentropy"`
  - Sparse Multiclass Cross-Entropy Loss: `"sparse_categorical_crossentropy"`
  - Kulbkack Leibler Divergense Loss: `"kl_divergence"`
- Regression
  - Mean Squared Error Loss: `"mse"`
  - Mean Squared Logarithmic Error Loss: `"mean_squared_logarithmic_error"`
  - Mean Aboslute Error Loss: `"mean_aboslute_error"`

Metrics

- Accuracy: `"accuracy"`
- Binary accuracy: `"binary_accuracy"`
- Categorical accuracy : `"categorical_accuracy"`

  You can also use all the loss functions

Compilation parameters

```
model.compile(
    optimizer="rmsprop", # Optimizer used in the gradient descent
    loss=None, # Loss function used to calculate the output error
    metrics=None, # Metrics computed during the training
)
```

Train the model

```
model.fit(
    x=Xtrain, # Observations/Inputs
    y=ytrain, # True label of the inputs
    batch_size=15, # Batch size in mini batch
    epochs=10, # Number of epochs
    verbose=1, # 0 -> no output. 1 -> progress bar. 2 -> 1 line per epoch
    validation_split=0.0, # Percentage of the training set used as a validation set
    validation_data=None, # Data used as a validation set. Format (Xvalidation, yvalidation)
    shuffle=True, # Wether to shuffle the training set after each epochs
```

```
    class_weight=None, # Dictionary to map output index to a weight (in order to focus more on one outp
    sample_weight=None, # Numpy array to map the inputs to a weight (usually 1:1 map with the inputs bu
    workers=1, # Number of process used to fit the model
    use_multiprocessing=False, # Wether to use multiprocessing or not
)
```

Predict

The method used to predict the outputs depends on how you build your model and is likely to be different.

Classification

We suppose that the model output a vector corresponding to the probabilities of the input to belong to each class.

```
probs = model.predict(testX)
label = np.argmax(probs)
```

Regression

For a regression no post-process is needed to retrieve the output

```
predicted_values = model.predict(testX)
```

Analyse model

To retrieve information about the model you need to store the output of `.fit` in a variable.

```
history = model.fit(...)
```

Evaluate a test set

The metrics returned by the `.evaluate()` method are the ones passed in the `metrics` parameter you compile the model.

```
evaluation = model.evaluate(textX, testy, return_dict=True)
for (metric, val) in evaluation.items():
    print(f'{metric}: {val}')
```

Visualization

Plot Accuracy vs Validation Accuracy

```
import matplotlib.pyplot as plt

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
plt.scatter(range(len(accuracy)), accuracy, label='train')
plt.scatter(range(len(val_accuracy)), val_accuracy, label='validation')
plt.legend()
plt.grid()
```

Plot Loss vs Validation Loss

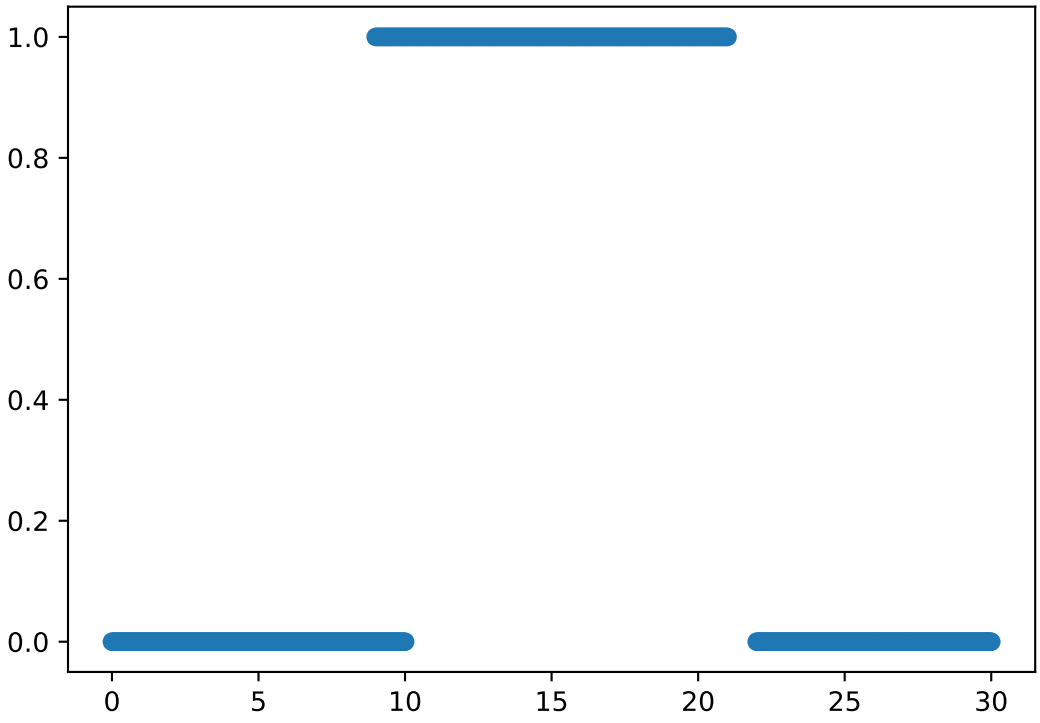
```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
plt.scatter(range(len(loss)), loss, label='loss')
plt.scatter(range(len(val_loss)), val_loss, label='val_loss')
plt.legend()
plt.grid()
```

Example

Binary Classification

Generate the data

```
import numpy as np
import matplotlib.pyplot as plt

nb_sim = 200

# Features
X = np.concatenate([np.linspace(0,10,nb_sim),
                    np.linspace(9,21,nb_sim),
                    np.linspace(22,30,nb_sim)])
# Targets
y = np.concatenate([np.repeat(0, nb_sim),
                    np.repeat(1, nb_sim),
                    np.repeat(0, nb_sim)])


plt.scatter(X, y)
plt.show()
```




```
# Scaling
X = (X - np.min(X)) / (np.max(X) - np.min(X)) # !! This work because the vector X is a 1-dimensional ar

# Shuffle
data_set = np.array(list(zip(X, y)))
np.random.shuffle(data_set)
X = data_set[:, 0]
y = data_set[:, 1]

# We actually need to change the target format. 0 -> [1, 0]; 1 -> [0, 1]
y = np.array([[0, 1] if val == 1 else [1, 0] for val in y])

# Split between training and testing
train_index = int(0.8 * X.shape[0])
X_train, X_test = X[:train_index], X[train_index:]
y_train, y_test = y[:train_index], y[train_index:]
```

Create and train the model

```
from tensorflow import keras
```

```
## WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
## WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
## WARNING:root:Limited tf.compat.v2.summary API due to missing TensorBoard installation.
## WARNING:root:Limited tf.summary API due to missing TensorBoard installation.
```

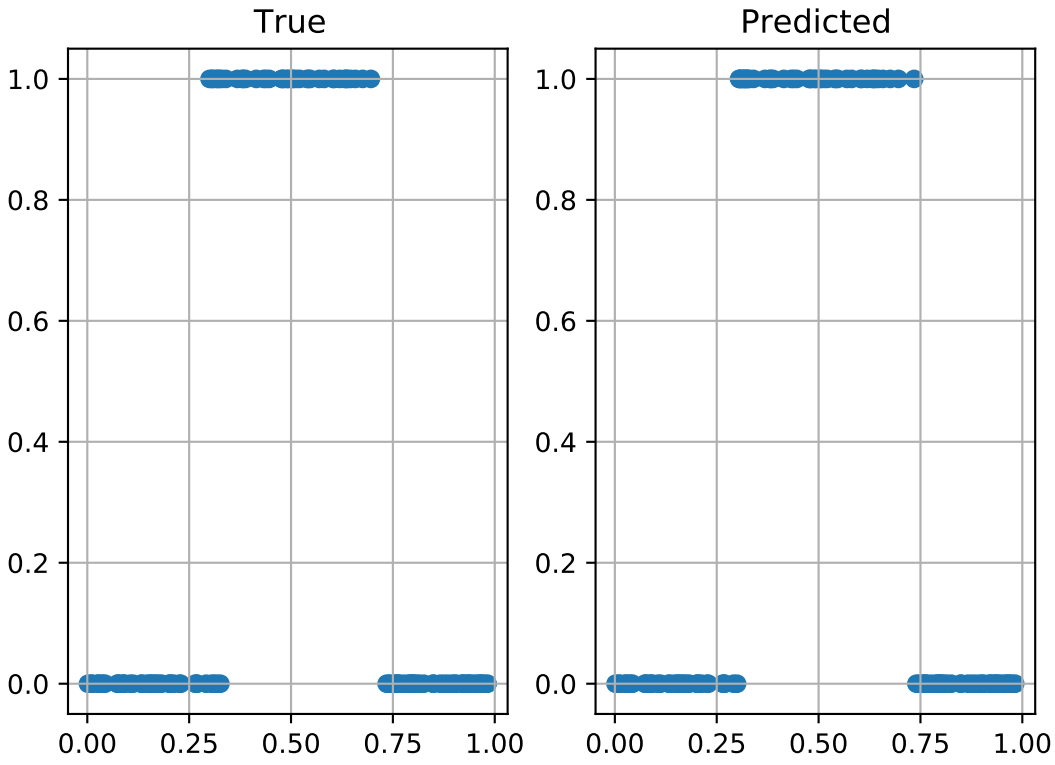
```
from keras import Sequential
from keras.layers import Dense
from keras.optimizers import SGD

model = Sequential([
    Dense(10, activation='tanh'),
    Dense(2, activation='sigmoid')
])
model.compile(
    optimizer=SGD(learning_rate=0.5),
    loss='binary_crossentropy',
    metrics=['binary_accuracy']
)
history = model.fit(
    X_train,
    y_train,
    epochs=100,
    validation_split=0.2,
    verbose=0
)
```

Evaluate and visualize

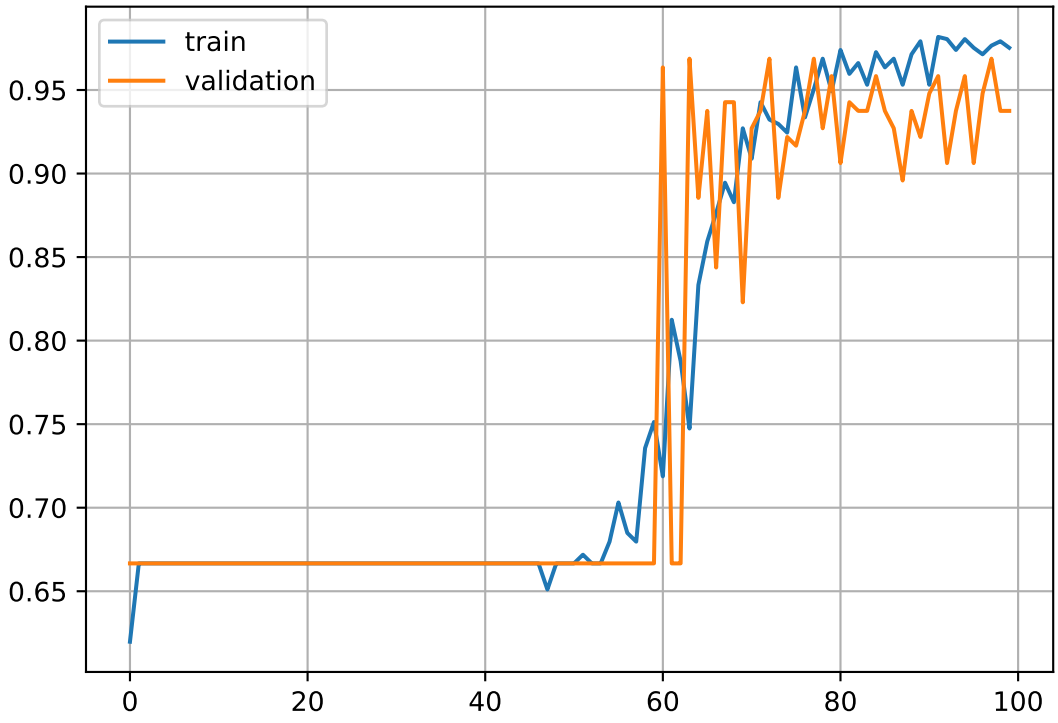
```
evaluation = model.evaluate(X_test, y_test, return_dict=True, verbose=0)
for (metric, val) in evaluation.items():
    print(f'{metric}: {val}')
```

```
## loss: 0.16790111362934113
## binary_accuracy: 0.949999988079071
```

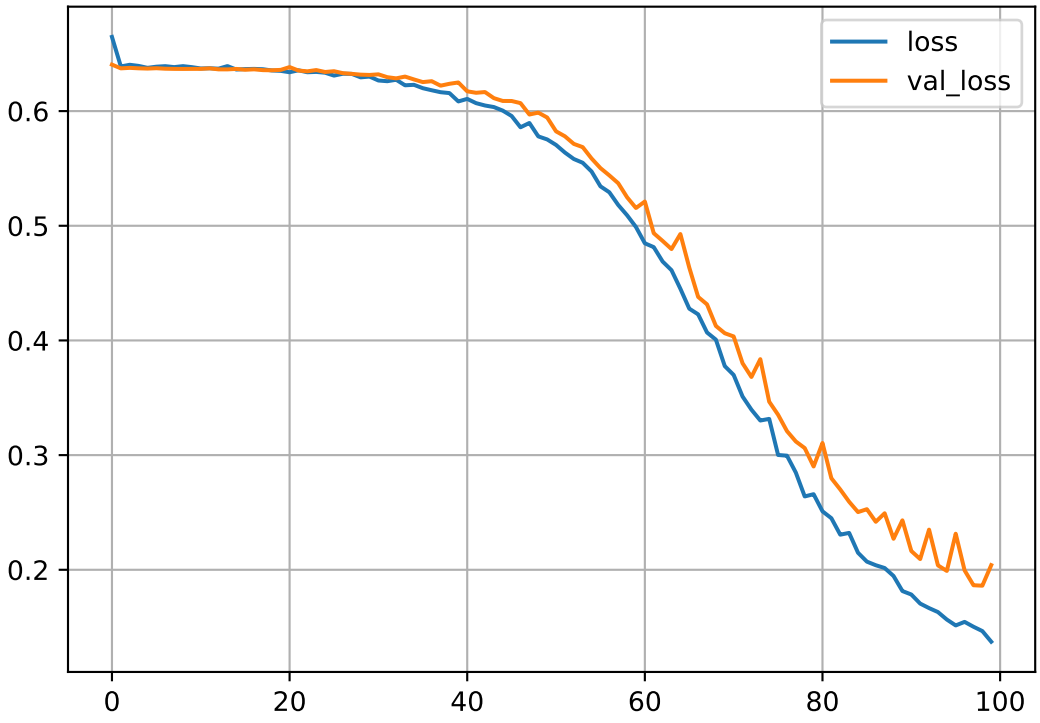
```
output_label = [np.argmax(p) for p in model.predict(X_test)]
true_label = [np.argmax(t) for t in y_test]
plt.subplot(121)
plt.scatter(X_test, true_label)
plt.title('True')
plt.grid()
plt.subplot(122)
plt.scatter(X_test, output_label)
plt.title('Predicted')
plt.grid()
plt.show()
```




```
accuracy = history.history['binary_accuracy']
val_accuracy = history.history['val_binary_accuracy']
plt.plot(range(len(accuracy)), accuracy, label='train')
plt.plot(range(len(val_accuracy)), val_accuracy, label='validation')
plt.legend()
plt.grid()
plt.show()
```



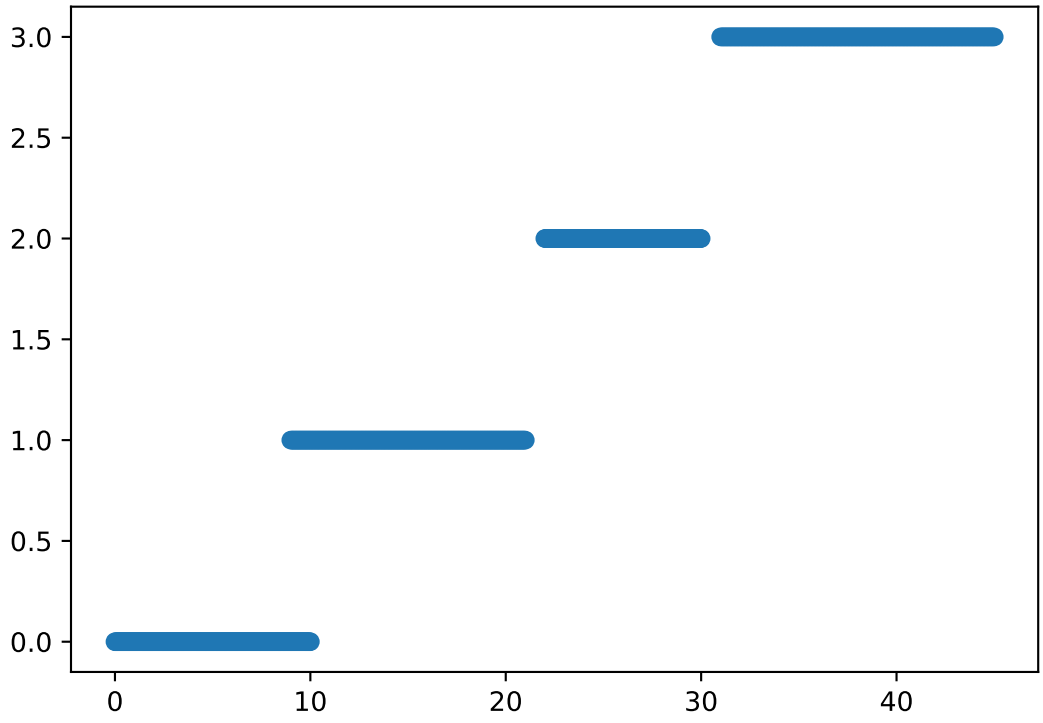

```
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(range(len(loss)), loss, label='loss')
plt.plot(range(len(val_loss)), val_loss, label='val_loss')
plt.legend()
plt.grid()
plt.show()
```



Multi Classification

Generate the data

```
import numpy as np
import matplotlib.pyplot as plt

nb_sim = 200

# Features
X = np.concatenate([np.linspace(0, 10, nb_sim),
                    np.linspace(9, 21, nb_sim),
                    np.linspace(22, 30, nb_sim),
                    np.linspace(31, 45, nb_sim)])
# Targets
y = np.concatenate([np.repeat(0, nb_sim),
                    np.repeat(1, nb_sim),
                    np.repeat(2, nb_sim),
                    np.repeat(3, nb_sim)])


plt.scatter(X, y)
plt.show()
```




```
# Scale. See also: https://www.tensorflow.org/api_docs/python/tf/keras/utils/normalize
X = (X - np.min(X)) / (np.max(X) - np.min(X)) # !! This work because the vector X is a 1-dimensional ar


# Shuffle
data_set = np.array(list(zip(X, y)))
np.random.shuffle(data_set)
X = data_set[:, 0]
y = data_set[:, 1]


# We actually need to change the target format. See also: https://www.tensorflow.org/api_docs/python/tf
y = np.array([[0] * int(val) + [1] + [0] * int(np.max(y) - val) for val in y])

# Split between training and testing
train_index = int(0.8 * X.shape[0])
X_train, X_test = X[:train_index], X[train_index:]
y_train, y_test = y[:train_index], y[train_index:]
```

Create and train the model

```
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
```

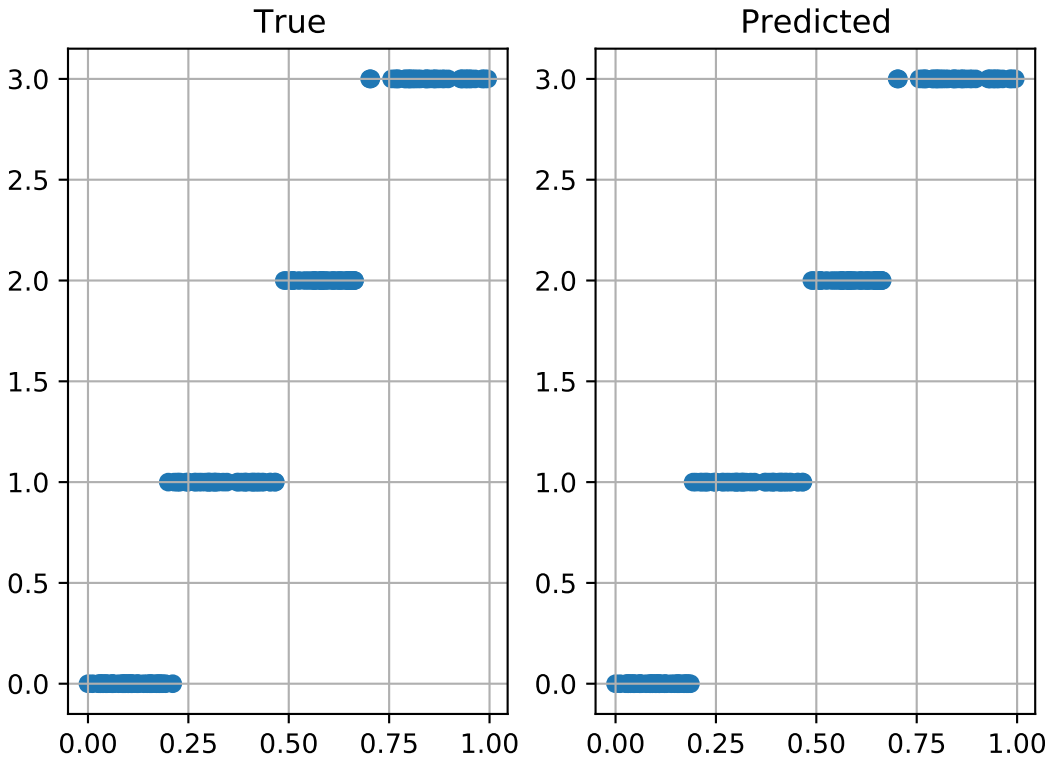
```
model = Sequential([
    Dense(10, activation='tanh'),
    Dense(4, activation='softmax')
])
model.compile(
    optimizer=SGD(learning_rate=0.5),
    loss='categorical_crossentropy',
    metrics=['categorical_accuracy']
)
history = model.fit(
    X_train,
    y_train,
    epochs=100,
    validation_split=0.2,
    verbose=0
)
```

Evaluate and visualize

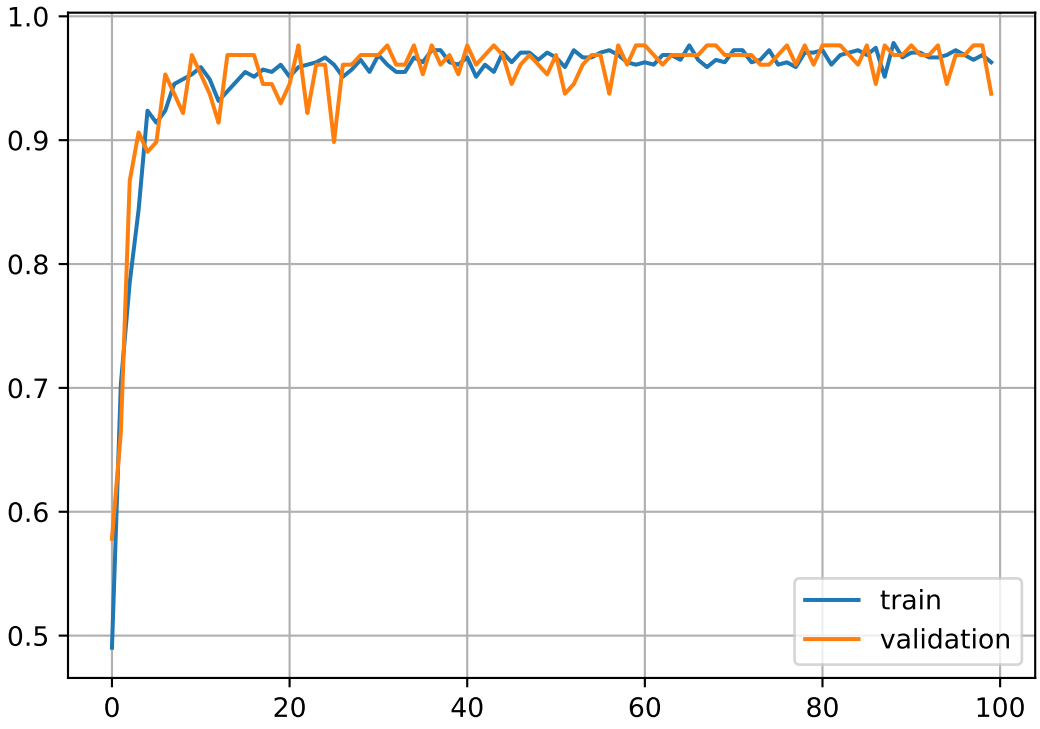
```
evaluation = model.evaluate(X_test, y_test, return_dict=True, verbose=0)
for (metric, val) in evaluation.items():
    print(f'{metric}: {val}')
```

```
## loss: 0.07038575410842896
## categorical_accuracy: 0.987500011920929
```

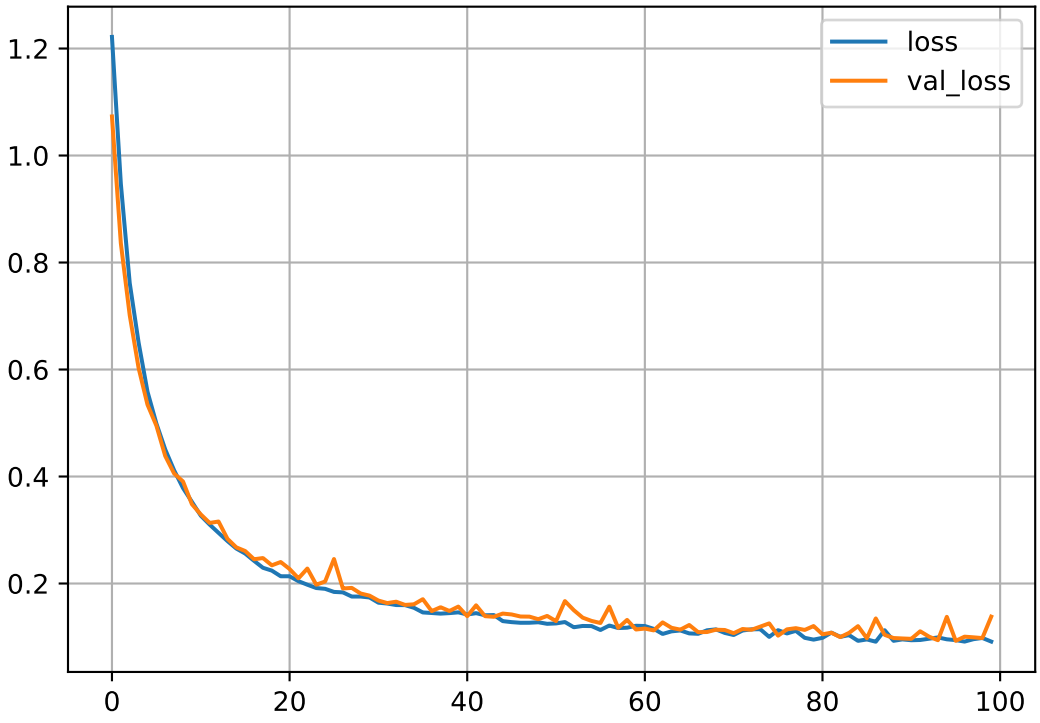
```
output_label = [np.argmax(p) for p in model.predict(X_test)]
true_label = [np.argmax(t) for t in y_test]
plt.subplot(121)
plt.scatter(X_test, true_label)
plt.title('True')
plt.grid()
plt.subplot(122)
plt.scatter(X_test, output_label)
plt.title('Predicted')
plt.grid()
plt.show()
```




```
accuracy = history.history['categorical_accuracy']
val_accuracy = history.history['val_categorical_accuracy']
plt.plot(range(len(accuracy)), accuracy, label='train')
plt.plot(range(len(val_accuracy)), val_accuracy, label='validation')
plt.legend()
plt.grid()
plt.show()
```




```
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(range(len(loss)), loss, label='loss')
plt.plot(range(len(val_loss)), val_loss, label='val_loss')
plt.legend()
plt.grid()
plt.show()
```


loss
val_loss
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100