```
from numpy.random import randint
import numpy as np
from tensorflow import keras
from keras import layers
import matplotlib.pyplot as plt
```

OPTIMIZATION 1. Take best model 2. round

1% of permutation, 5 array length, 10 max digit, 100 epochs, 16 batch, 32 32 == 5% error

1% of permutation, 5 array length, 10 max digit, 100 epochs, 16 batch, 16 16 == 6% error

1% of permutation, 5 array length, 10 max digit, 100 epochs, 16 batch, 64 64 == 3% error

1% of permutation, 5 array length, 10 max digit, 100 epochs, 16 batch, 64 64 64 == 2% error

1% of permutation, 5 array length, 10 max digit, 100 epochs, 16 batch, 64 64 64 64 == 1.5% error

1% of permutation, 5 array length, 10 max digit, 150 epochs, 16 batch, 64 64 64 64 == 1.5% error

CHANGE to 10 length array

10^-5% of permutation, 10 array length, 10 max digit, 100 epochs, 16 batch, 64 64 64 64 == 5% error

Anche se ha visto molto meno da un buon risultato

10^-4% of permutation, 10 array length, 10 max digit, 100 epochs, 16 batch, 64 64 64 64 == 3% error

```
 # parameters

 modelPath = ""
 modelName = "NNMedian.keras"

 n_arrays = 2000
 array_length = 5
 max_digit = 10
 epochs = 100
 batch_size = 8
```

```
# calculate number of permutation
print("number of permutation: ", max_digit**array_length)
print("percentage of view:", (n_arrays * 0.5) * 100 / max_digit**array_length)
```

```
number of permutation:  100000
percentage of view: 1.0
```

```
# generate input and output data

input_arrays = []
label_arrays = []
for _ in range(n_arrays):
    temp_array = randint(0, max_digit, array_length)
    temp_label = np.median(temp_array)
    input_arrays.append(temp_array)
    label_arrays.append(temp_label)
```

```
print(len(input_arrays), len(label_arrays))
```

```
2000 2000
```

```
# normalize

for i in range(len(input_arrays)):
    input_arrays[i] = input_arrays[i].astype("float32") / (max_digit - 1)
    label_arrays[i] = label_arrays[i].astype("float32") / (max_digit - 1)
```

```
# make np array
input_arrays = np.array(input_arrays)
label_arrays = np.array(label_arrays)
```

```
print(input_arrays[:10])
print(label_arrays[:10])
```

```
[[0.7777778  0.8888889  0.22222222 0.7777778  0.22222222]
```

```
 [0.7777778  0.33333334 0.8888889  0.22222222 0.7777778 ]
 [0.5555556  0.11111111 0.8888889  0.33333334 0.        ]
 [0.5555556  0.44444445 0.11111111 0.7777778  0.5555556 ]
 [0.44444445 0.6666667  0.7777778  0.11111111 0.5555556 ]
 [0.5555556  0.22222222 0.11111111 0.5555556  0.33333334]
 [0.11111111 0.         0.11111111 1.         0.8888889 ]
 [0.44444445 0.44444445 0.11111111 0.22222222 1.        ]
 [0.11111111 0.33333334 0.8888889  1.         0.5555556 ]
 [1.         0.11111111 0.11111111 0.8888889  0.7777778 ]]
[0.77777778 0.77777778 0.33333333 0.55555556 0.55555556 0.33333333
 0.11111111 0.44444444 0.55555556 0.77777778]
```

```
# split data

n_train = int(0.5 * n_arrays)
n_val = int(0.25 * n_arrays)
n_eval = n_arrays - n_train - n_val

data_test = input_arrays[:n_train]
data_validation = input_arrays[n_train:(n_train+n_val)]
data_evaluation = input_arrays[n_train + n_val:]

label_test = label_arrays[:n_train]
label_validation = label_arrays[n_train:(n_train+n_val)]
label_evaluation = label_arrays[n_train + n_val:]
```

```
print(data_test.shape, data_validation.shape, data_evaluation.shape)
print(label_test.shape, label_validation.shape, label_evaluation.shape)
```

```
(1000, 5) (500, 5) (500, 5)
(1000,) (500,) (500,)
```

```
from tensorflow import keras
from keras import layers

inputs = keras.Input(shape=(array_length))
x = layers.Dense(64, activation="relu") (inputs)
x = layers.Dense(64, activation="relu") (x)
x = layers.Dense(64, activation="relu") (x)
x = layers.Dense(64, activation="relu") (x)
outputs = layers.Dense(1, activation="sigmoid") (x)

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

model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 5)]               0

 dense_15 (Dense)            (None, 64)                384

 dense_16 (Dense)            (None, 64)                4160

 dense_17 (Dense)            (None, 64)                4160

 dense_18 (Dense)            (None, 64)                4160

 dense_19 (Dense)            (None, 1)                 65

=================================================================
Total params: 12,929
Trainable params: 12,929
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["mae"]
)
```

```
callbacks = [
keras.callbacks.ModelCheckpoint(
    filepath=modelPath + modelName,
    monitor="val_mae",
```

```
    save_best_only=True,)
```

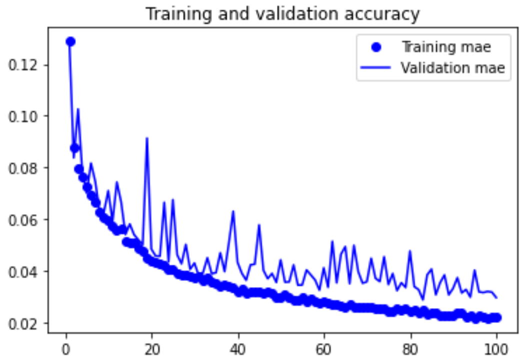
```
history = model.fit(data_test,
                    label_test,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data=(data_validation, label_validation),
                    callbacks=callbacks)
```

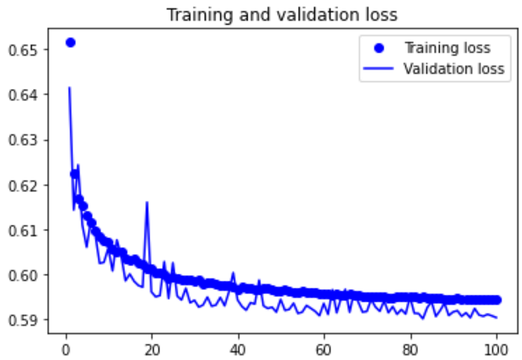
```
Epoch 72/100
125/125 [==============================] - 0s 4ms/step - loss: 0.5951 - mae: 0.0253 - val_loss: 0.5927 - val_mae:
Epoch 73/100
125/125 [==============================] - 0s 4ms/step - loss: 0.5950 - mae: 0.0251 - val_loss: 0.5917 - val_mae:
Epoch 74/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5950 - mae: 0.0253 - val_loss: 0.5941 - val_mae:
Epoch 75/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5948 - mae: 0.0245 - val_loss: 0.5914 - val_mae:
Epoch 76/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5948 - mae: 0.0242 - val_loss: 0.5928 - val_mae:
Epoch 77/100
125/125 [==============================] - 0s 4ms/step - loss: 0.5951 - mae: 0.0253 - val_loss: 0.5911 - val_mae:
Epoch 78/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5950 - mae: 0.0246 - val_loss: 0.5921 - val_mae:
Epoch 79/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5951 - mae: 0.0254 - val_loss: 0.5911 - val_mae:
Epoch 80/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5949 - mae: 0.0242 - val_loss: 0.5949 - val_mae:
Epoch 81/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5950 - mae: 0.0250 - val_loss: 0.5913 - val_mae:
Epoch 82/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5948 - mae: 0.0239 - val_loss: 0.5912 - val_mae:
Epoch 83/100
125/125 [==============================] - 0s 4ms/step - loss: 0.5950 - mae: 0.0246 - val_loss: 0.5901 - val_mae:
Epoch 84/100
125/125 [==============================] - 0s 4ms/step - loss: 0.5946 - mae: 0.0234 - val_loss: 0.5925 - val_mae:
Epoch 85/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5948 - mae: 0.0240 - val_loss: 0.5939 - val_mae:
Epoch 86/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5948 - mae: 0.0237 - val_loss: 0.5906 - val_mae:
Epoch 87/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5946 - mae: 0.0229 - val_loss: 0.5919 - val_mae:
Epoch 88/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5945 - mae: 0.0227 - val_loss: 0.5935 - val_mae:
Epoch 89/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5945 - mae: 0.0226 - val_loss: 0.5908 - val_mae:
Epoch 90/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5945 - mae: 0.0228 - val_loss: 0.5916 - val_mae:
Epoch 91/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5947 - mae: 0.0236 - val_loss: 0.5919 - val_mae:
Epoch 92/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5945 - mae: 0.0235 - val_loss: 0.5906 - val_mae:
Epoch 93/100
125/125 [==============================] - 0s 4ms/step - loss: 0.5943 - mae: 0.0221 - val_loss: 0.5914 - val_mae:
Epoch 94/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5944 - mae: 0.0227 - val_loss: 0.5903 - val_mae:
Epoch 95/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5944 - mae: 0.0218 - val_loss: 0.5924 - val_mae:
Epoch 96/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5944 - mae: 0.0226 - val_loss: 0.5909 - val_mae:
Epoch 97/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5944 - mae: 0.0222 - val_loss: 0.5906 - val_mae:
Epoch 98/100
125/125 [==============================] - 0s 4ms/step - loss: 0.5943 - mae: 0.0216 - val_loss: 0.5911 - val_mae:
Epoch 99/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5943 - mae: 0.0221 - val_loss: 0.5907 - val_mae:
Epoch 100/100
125/125 [==============================] - 0s 3ms/step - loss: 0.5944 - mae: 0.0219 - val_loss: 0.5903 - val_mae:
```

```
mae = history.history["mae"]
val_mae = history.history["val_mae"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
x_epochs = range(1, len(val_mae) + 1)
```

```
plt.plot(x_epochs, mae, "bo", label="Training mae")
plt.plot(x_epochs, val_mae, "b", label="Validation mae")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(x_epochs, loss, "bo", label="Training loss")
plt.plot(x_epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
```

```
plt.legend()
plt.show()
```






```
model_best = keras.models.load_model(modelPath + modelName)

eval_loss, eval_mae = model_best.evaluate(data_evaluation, label_evaluation)
print(f"Evaluation accuracy: {eval_mae:.3f}")
eval_mae_perc = eval_mae * (max_digit - 1)
print(f"Evaluation mae denormalized: {eval_mae_perc:.2f}")
```

```
16/16 [==============================] - 0s 2ms/step - loss: 0.6008 - mae: 0.0308
Evaluation accuracy: 0.031
Evaluation mae denormalized: 0.28
```

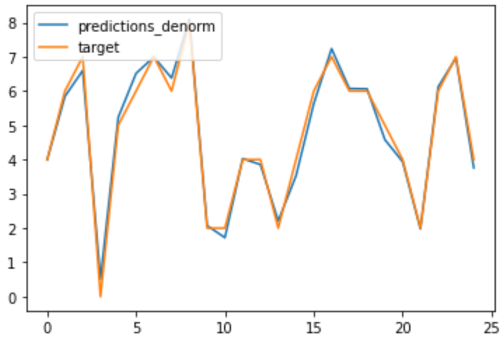
```
def denormalize(x):
    x = x * (max_digit - 1)
    return x

def rounding(x):
    x = round(x, 0)
    return x

predictions = model_best.predict(data_evaluation)
targets = label_evaluation
```

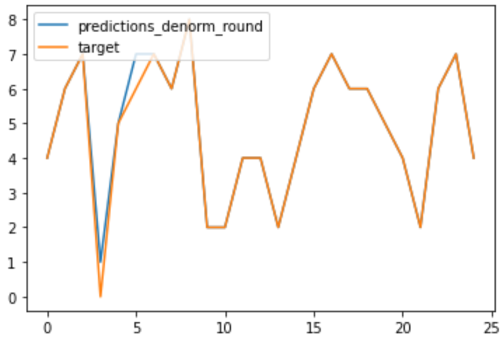
```
16/16 [==============================] - 0s 2ms/step
```

```
x = []
predictions_denorm = []
predictions_denorm_round = []
targets_denorm = []
for i in range(len(predictions)):
    x.append(i)
    predictions_denorm.append(denormalize(float(predictions[i])))
    predictions_denorm_round.append(rounding(denormalize(float(predictions[i]))))
    targets_denorm.append(denormalize(targets[i]))

print(predictions_denorm[:25])
print(predictions_denorm_round[:25])
print(targets_denorm[:25])

plt.plot(x[:25], predictions_denorm[:25], label="predictions_denorm")
plt.plot(x[:25], targets_denorm[:25], label="target")
plt.legend(loc="upper left")
plt.show()

plt.plot(x[:25], predictions_denorm_round[:25], label="predictions_denorm_round")
plt.plot(x[:25], targets_denorm[:25], label="target")
```

```
plt.legend(loc="upper left")
plt.show()
```

```
[3.99136546254158, 5.842572748661041, 6.599141299724579, 0.508658342063427, 5.233065962791443, 6.513870120048523, 
[4.0, 6.0, 7.0, 1.0, 5.0, 7.0, 7.0, 6.0, 8.0, 2.0, 2.0, 4.0, 4.0, 2.0, 4.0, 6.0, 7.0, 6.0, 6.0, 5.0, 4.0, 2.0, 6.0
[4.0, 6.0, 7.0, 0.0, 5.0, 6.0, 7.0, 6.0, 8.0, 2.0, 2.0, 4.0, 4.0, 2.0, 4.0, 6.0, 7.0, 6.0, 6.0, 5.0, 4.0, 2.0, 6.0
```






```
def mean_absolute_error(pre, tar):
    sum = 0
    n = len(pre)
    for i in range(n):
        sum += abs(tar[i] - pre[i])
    error = sum/n
    return error
```

```
print("MAE of denormalized: ", mean_absolute_error(predictions_denorm, targets_denorm))
print("MAE of denormalized rounded: ", mean_absolute_error(predictions_denorm_round, targets_denorm))
print("Percentage error: ", round(mean_absolute_error(predictions_denorm_round, targets_denorm) * 100 / max_digit,
```

```
MAE of denormalized:  0.27704644679650664
MAE of denormalized rounded:  0.154
Percentage error:  1.54 %
```

Colab paid products - Cancel contracts here

✓ 0s completed at 6:23 PM