Univerzitet u Tuzli
Fakultet elektrotehnike
Predmet: Inteligentni sistemi
Akademska godina: 2021./2022.
Student: Indir Karabegović
Datum: 13.05.2022. god.

Izvještaj zadaće broj 2

Zadatak br. 1 (1.C.X)

U LV6 je urađen primjer MLP klasifikacije tri različita dataseta (polumjeseci, koncentrični, linearno separabilni), sa različitim korakom alpha. Kao rezultat je plot sa 15 subplotova na jednom plotu za različito alfa.

c) izvršiti korekciju korištenjem **različitog learning ratea (varijacije learning_rate_init i learning_rate)** i prikazati rezultate na jednom plotu. Tabelarno prikazati 10 kombinacija.

Rješenje:

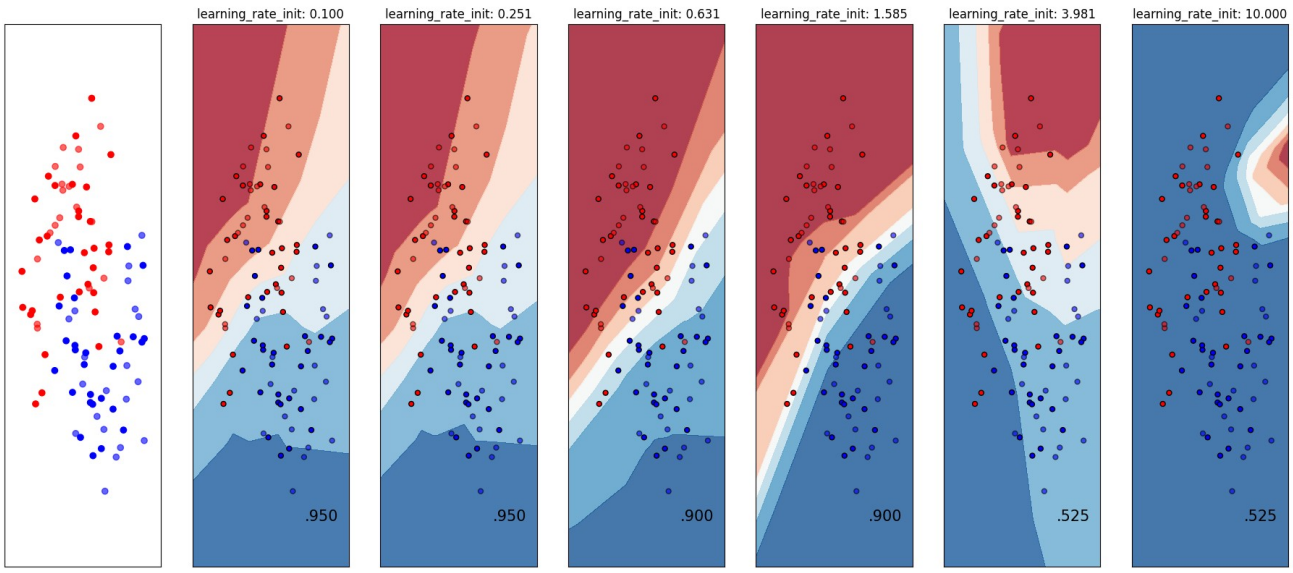
Kao što je u zadatku navedno, potrebno je mijenjati dva parametra, a to su **learning_rate** i **learning_rate_init.**

learning_rate parametar može da ima sljedeće vrijednosti: ***'constant', 'invscaling', 'adaptive',*** pri čemu vrijednost **solver**-a mora biti jednaka **sgd**

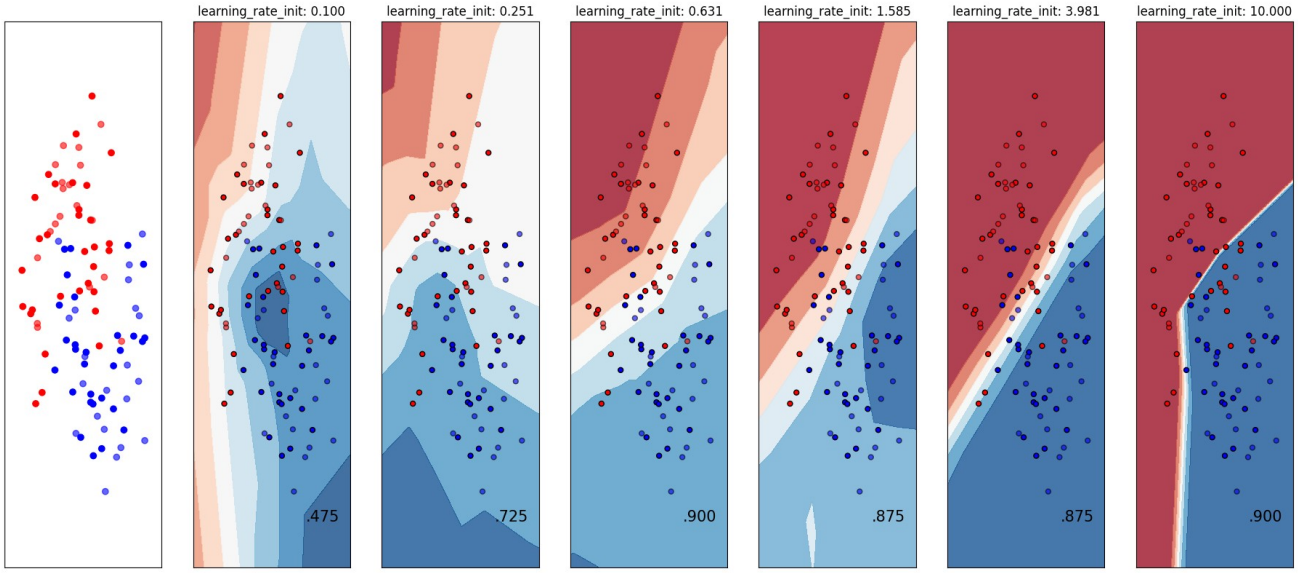
learning_rate_init parametar je parametar koji je tipa **float**, te mu je default vrijednost jednaka **0.001**

U ovom zadatku smo za svaku vrijednost learning_rate-a mijenjali nekoliko vrijednosti learning_rate_init-a što će biti predstavljeno na narednih nekoliko slika:

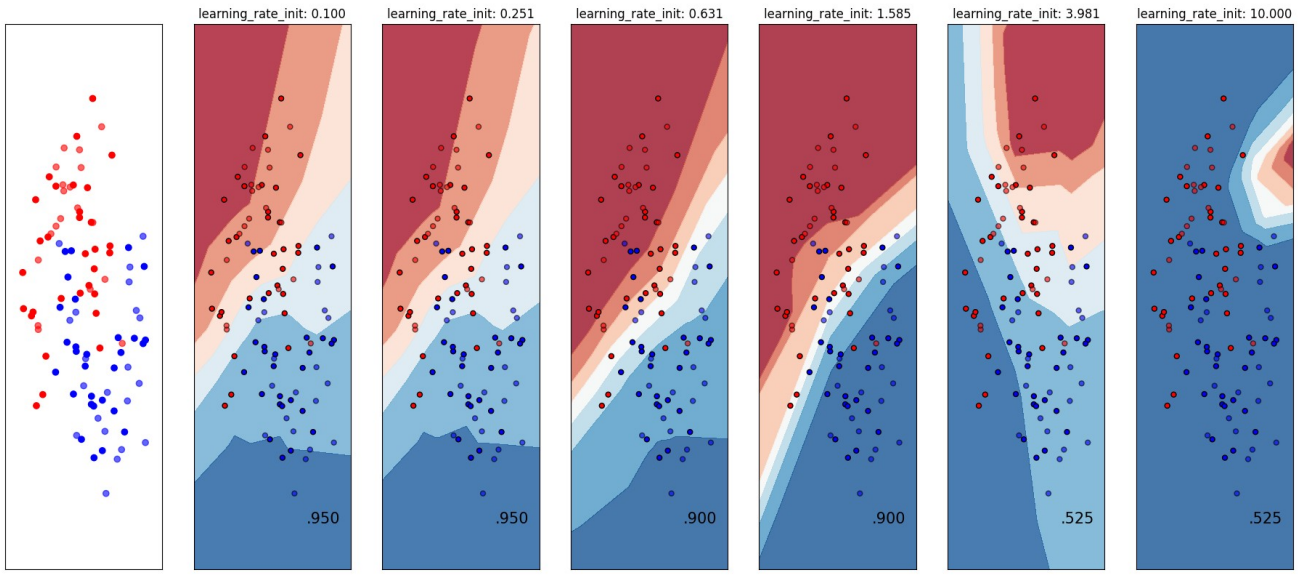
- **learning_rate = *'constant'***



- **learning_rate = *'invscaling'***



- **learning_rate = *'adaptive'***



Nakon što smo pokrenuli kod i izvršili plotanje, možemo primjetiti da kod *constant* i *adaptive* learning_rate-a, sa povećanjem learning_rate_init-a gubimo procenat tačnosti, dok kod *invscaling* learning_rate-a sa povećanjem dobijemo veću tačnost. Ako se fokusiramo na *constat learning rate* možemo primjetiti da nam je najveća vrijednost rezultata za learning rate init jednak 0.100, odnosno, tada imamo najbolju klasifikaciju. Klasifikacija je dobra sve dok nam vrijednost ne prekorači vrijednost veću od 2, tada ova vrjednost rezultata naglo pada. Isti je slučaj i kod adaptive learning rate-a, Kao što smo ranije rekli, kod invscaling learning rate-a, sa povećanjem

learning rate init-a vidimo da se konačna vrijednost (procenat efikasnosti klasifikacije) povećava. Za vrijednost od 0.100 imamo efikasnost od 47,5%, dok kod learning rate init-a od 10.000 imamo efikasnost od 90%. U narednoj tabeli ćemo prikazati neke rezultate za različite vrijednosti learning_rate_init-a i različit learning_rate.

Number	Solver	Max. iter	Learning rate	Learning rate init	Result
1.	sgd	2000	constant	0.100	0.950
2.	sgd	2000	constant	0.631	0.900
3.	sgd	2000	constant	3.981	0.525
4.	sgd	2000	invscaling	0.100	0.475
5.	sgd	2000	invscaling	0.631	0.900
6.	sgd	2000	invscaling	3.981	0.875
7.	sgd	2000	invscaling	10.000	0.900
8.	sgd	2000	adaptive	0.100	0.950
9.	sgd	2000	adaptive	0.631	0.900
10.	sgd	2000	adaptive	3.981	0.525

U nastavku je prikazan kod za navedeni zadatak:

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification, make_checkerboard
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import make_pipeline
h = 0.02  # step size in the mesh
lr_init = np.logspace(-1, 1, 6)
classifiers = []
names = []
```

```
for learning_rate_init in lr_init:
    classifiers.append(
        make_pipeline(
            StandardScaler(),
            MLPClassifier(
                solver="sgd",
                random_state=1,
                max_iter=2000,
                early_stopping=True,
                hidden_layer_sizes=[10, 10],
                # learning_rate="constant",
                # learning_rate="invscaling",
                learning_rate="adaptive",
                learning_rate_init=learning_rate_init
            ),
        )
    )
    names.append(f"learning_rate_init: {learning_rate_init:.3f}")

X, y = make_classification(
    n_features=2, n_redundant=0, n_informative=2, random_state=0, n_clusters_per_class=1
)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [
    make_moons(noise=0.3, random_state=0),
]
```

```
# print(datasets)
figure = plt.figure(figsize=(10, 5))
i = 1
# iterate over datasets
for X, y in datasets:
    # split into training and test part
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.4, random_state=42)
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(["#FF0000", "#0000FF"])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
    # and testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1
    # iterate over classifiers
    for name, clf in zip(names, classifiers):
        ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)
```

```
# Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max] x [y_min, y_max].
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(
            np.column_stack([xx.ravel(), yy.ravel()]))
    else:
        Z = clf.predict_proba(np.column_stack(
            [xx.ravel(), yy.ravel()]))[:, 1]
    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=0.8)
    # Plot also the training points
    ax.scatter(
        X_train[:, 0],
        X_train[:, 1],
        c=y_train,
        cmap=cm_bright,
        edgecolors="black",
        s=25,)
    # and testing points
    ax.scatter(
        X_test[:, 0],
        X_test[:, 1],
        c=y_test,
        cmap=cm_bright,
        alpha=0.6,
        edgecolors="black",
        s=25,  )
```

```
        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(name)
        ax.text(
            xx.max() - 0.3,
            yy.min() + 0.3,
            f"{score:.3f}".lstrip("0"),
            size=15,
            horizontalalignment="right",
        )
        i += 1
figure.subplots_adjust(left=0.02, right=0.98)
plt.show()
```

Zadatak br. 2 (2.A.X)

U LV7 je urađeno treniranje neuronske mreže na linearnu funkciju y= x+10, na opsegu 0-800, uz predikciju na intervalu 800-1000.

a) Uraditi regresiju na kosinusnu funkciju **y = cos(x)**.

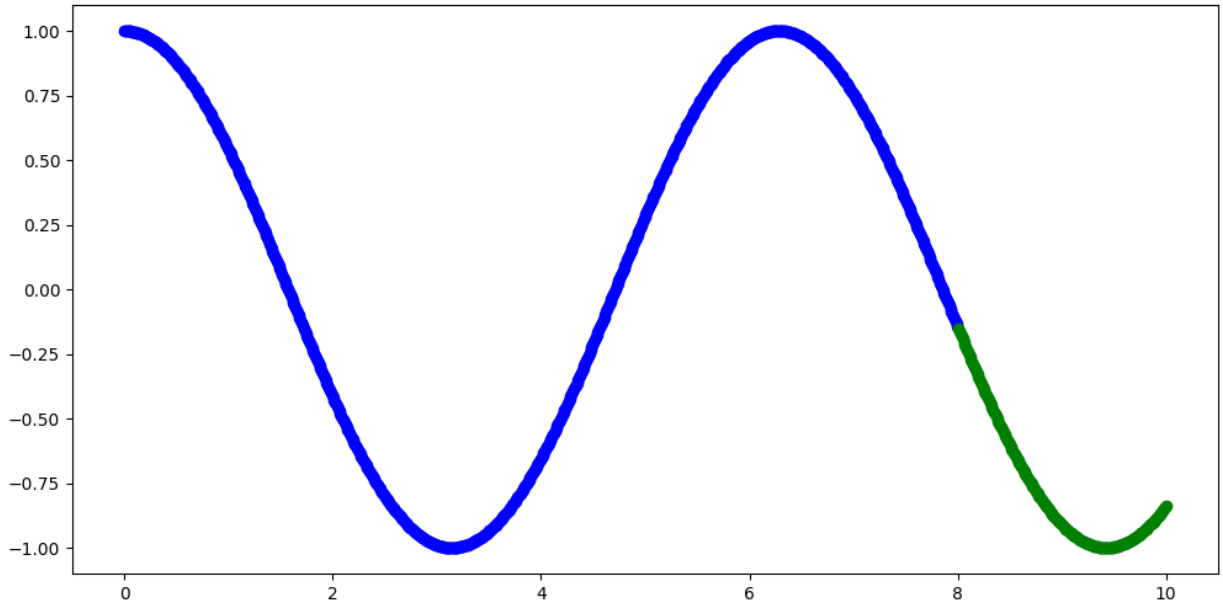
Analizirati različite kombinacije, isprobati **različite mreže s više skrivenih slojeva i više neurona po sloju**. Koristiti jednu proizvoljno odabranu aktivacijsku funkciju. U tabelu unijeti rezultate minimalno 9 kombinacija. Za rezultat koristiti metriku Mean Square error i dati analizu.

Za sve zadatke trenirati mrežu na opsegu 0…8, predikcija intervala 8…10. Minimalno 800+200 tačaka. Plotati originalni dataset na osnovu koga je izvršen trening (0…8) jednom bojom, a podatke za verifikaciju (na intervalu 8…10) drugom bojom Plotati očekivane rezultate (izlaze svih kombinacija neuronskih mreža (0…10), na istom plotu (različitom bojom ili različitim linijama).

Rješenje:

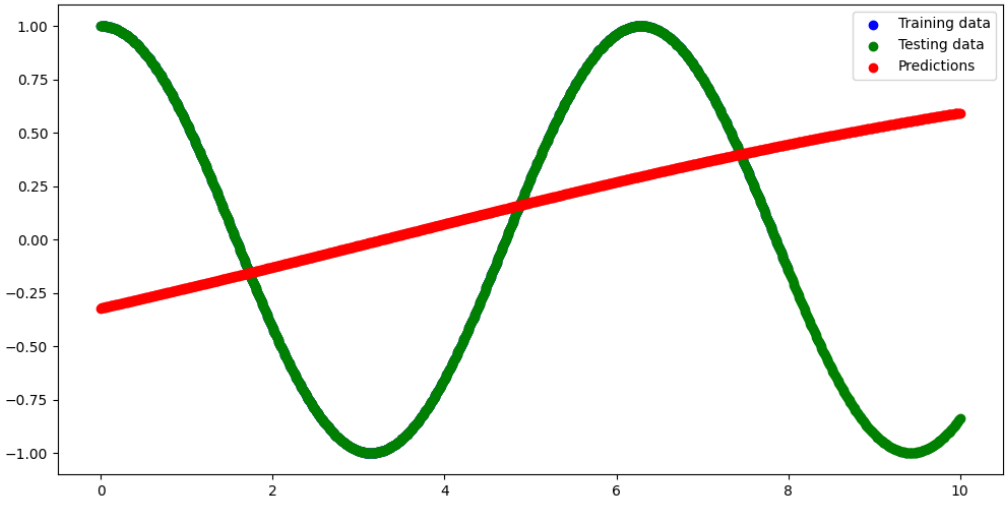
U ovom zadatku cilj nam je uraditi regresiju na kosinusnu funkciju, te isprobati različite tipove mreža, različit broj slojeva, te različit broj neurona po svakom od slojeva. Prvo što ćemo uraditi u ovom slučaju jeste plotati

traženi kosinusni signal, te na njemu različitim bojama prikazati podatke za treniranje i podatke koji su namijenjeni za učenje. Navedeni dijagram je prikazan na narednoj slici:



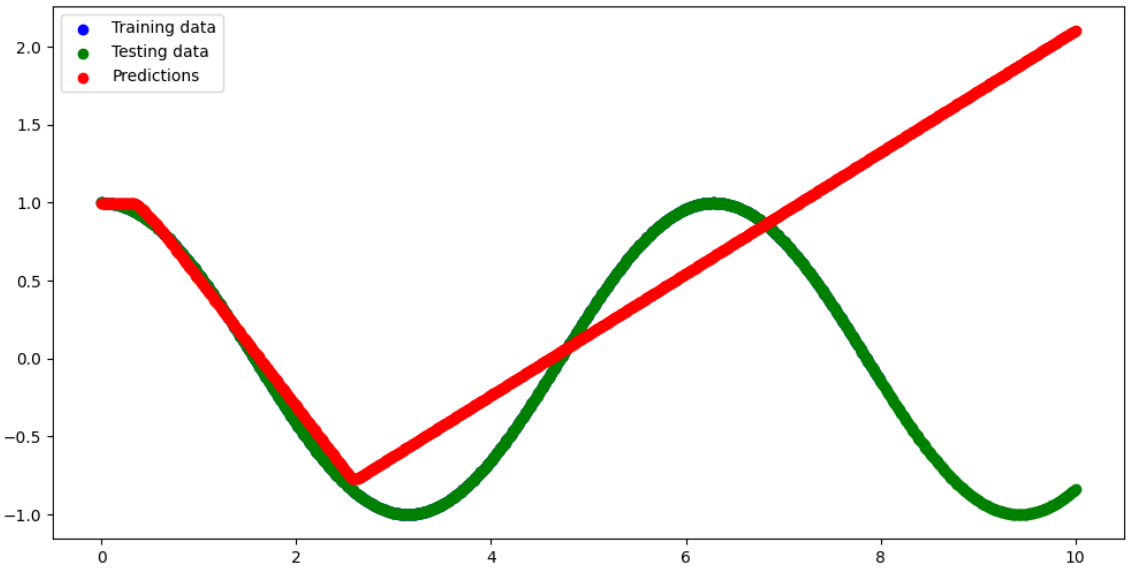
Sa slike možemo vidjeti da je 800 uzoraka (80%) korišteno za učenje mreže, dok je 200 uzoraka odnosno 20% iskorišteno za treniranje mreže. Dalje, na ovaj kosinusni signal ćemo primjeniti modele koje smo koristili na vježbama, te mijenjanjem broja slojeva, aktivacijskih funkcija i broja neurona po sloju pokušat ćemo da napravimo što bolju predikciju. Prije nego što pristupimo slučaju kada imamo 20% testnih podataka, analizirat ćemo različite slučajeve kada nam testni skup podataka obuhvata sve naše podatke (i podatke za treniranje i učenje). Analiziramo sljedeće slučajeve:

1) Broj slojeva je 2, a broj neurona je 2, mse = 0.4620



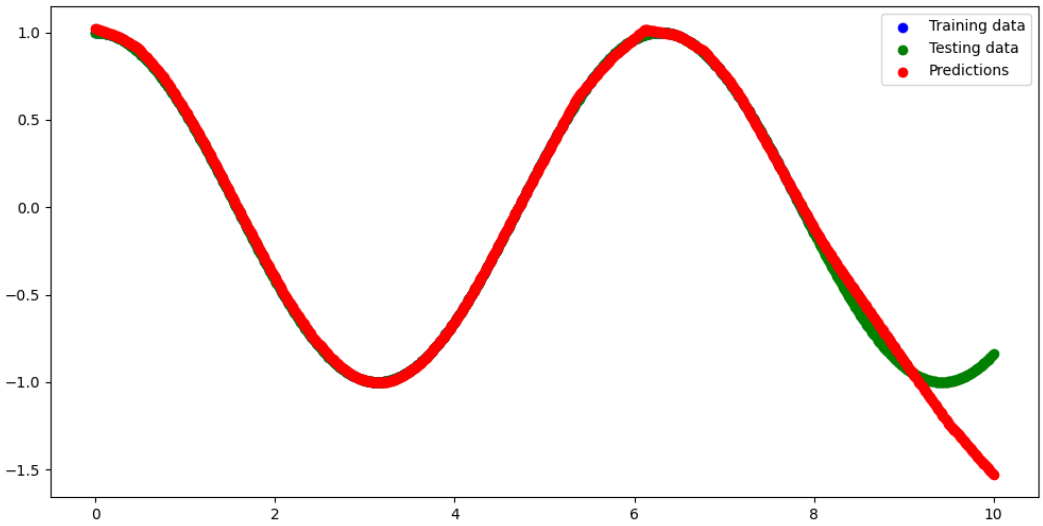
Sa slike možemo vidjeti da je predikcija izrazito loša, te ćemo zbog toga povećati broj slojeva i broj neurona koji se nalaze u tim slojevima. Kao aktivacijsku funkciju ćemo koristiti *relu* funkciju

2) Broj slojeva 4, broj neurona po slojevima (20,30,10,1), mse =0.4178



Sa slike možemo vidjeti da je sada predikcija nešto bolja nego u prethodnom slučaju, te sada možemo dodatno povećati broj neurona u svakom sloju, te dodati aktivacijsku funkciju u svaki od navedenih slojeva.

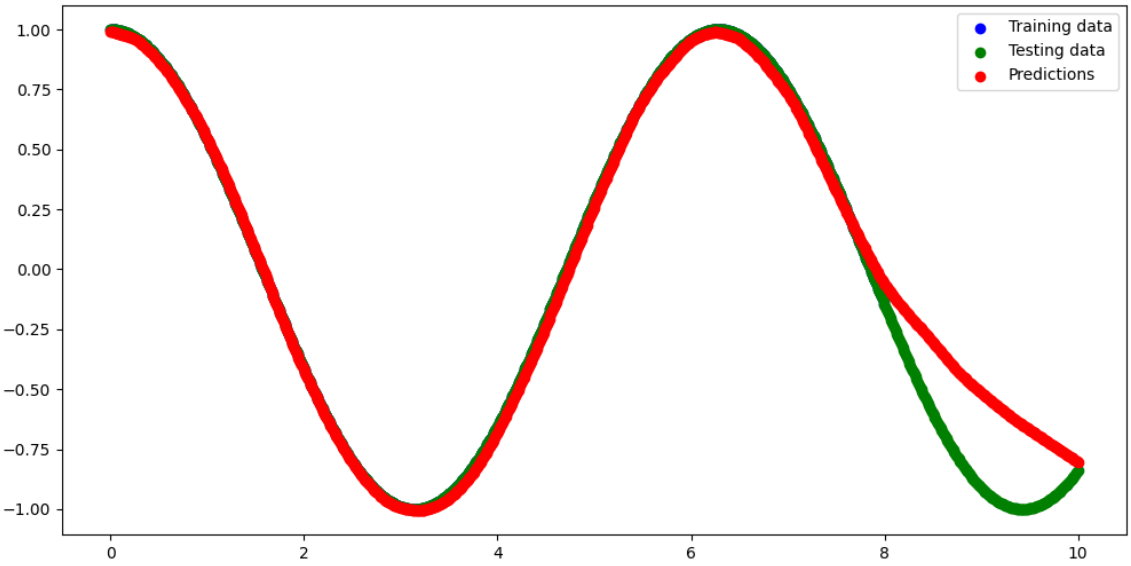
3) Broj slojeva je 4, broj neurona po slojevima je (60,80,100,1), mse = 2,9374 * 10^{-4}



Sa slike možemo vidjeti da smo ovim postavkama uspjeli postići izuzetno dobru predikciju kada su nam podaci isti kao podaci za učenje, međutim kada je riječ o podacima za treniranje, predikcija nam bježi, odnosno postaje

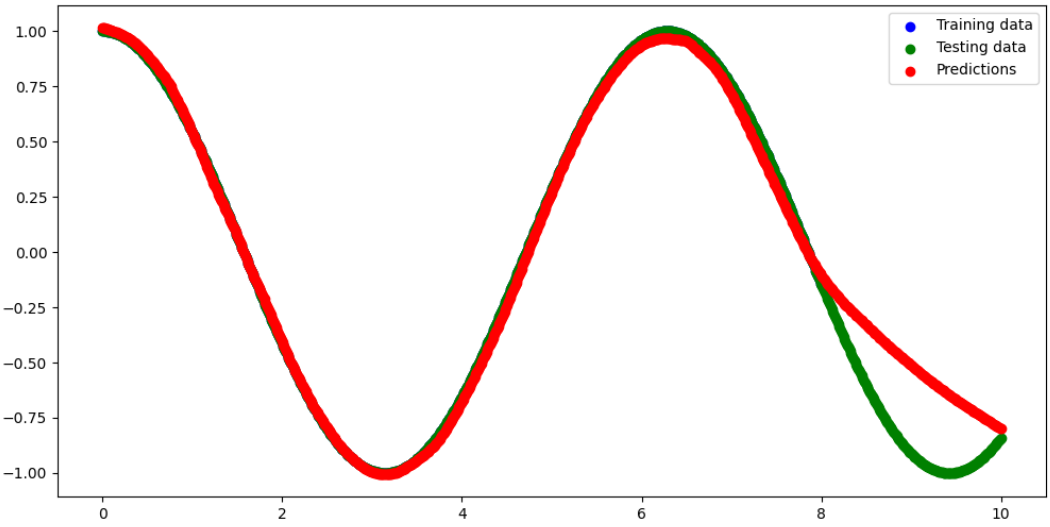
linearna i ne prati adekvatno sinusnu funkciju. Sada ćemo dodati još jedan sloj, te ćemo dodatno povećati broj neurona po svakom sloju (u svakom sloju ćemo ponovo imati *relu* aktivacijsku funkciju).

4) Broj slojeva je 5, broj neurona po slojevima je (300,400,400,300,1), mse = 0.0023



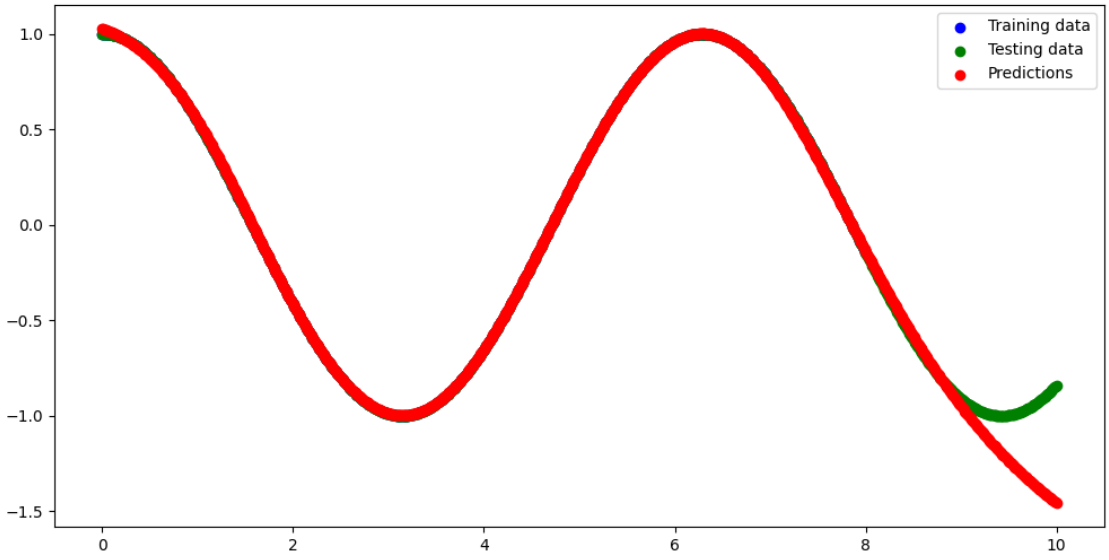
U odnosu na prethodni slučaj možemo primjetiti da dobijamo zakrivljenje na testnom dijelu podataka, međutim sama predikcija je znatno lošija. Sada ćemo pokušati dodati još jedan sloj, te vidjeti kakav je efekat dodavanja još jednog sloja.

5) Broj slojeva je 6, broj neurona po slojevima je (300,400,400,300,500,1), mse = 0.0017



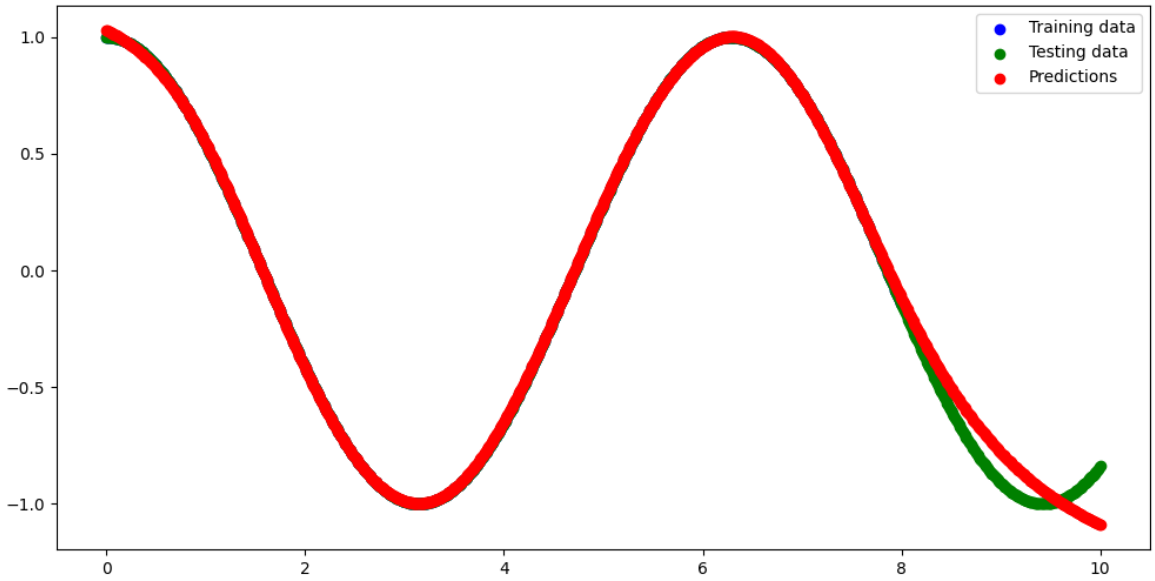
Možemo primjetiti da nam daljnje povećavanje broja neurona ili broja slojeva neće donijeti nikakvo poboljšanje, te ćemo morati primjeniti drugi pristup, odnosno, pokušat ćemo mijenjati aktivacijsku funkciju. Sada će nam aktivacijska funkcija biti tanh.

6) Broj slojeva je 4, broj neurona po sloju je (60,80,100, 1), a mse = $2.96*10^{-5}$



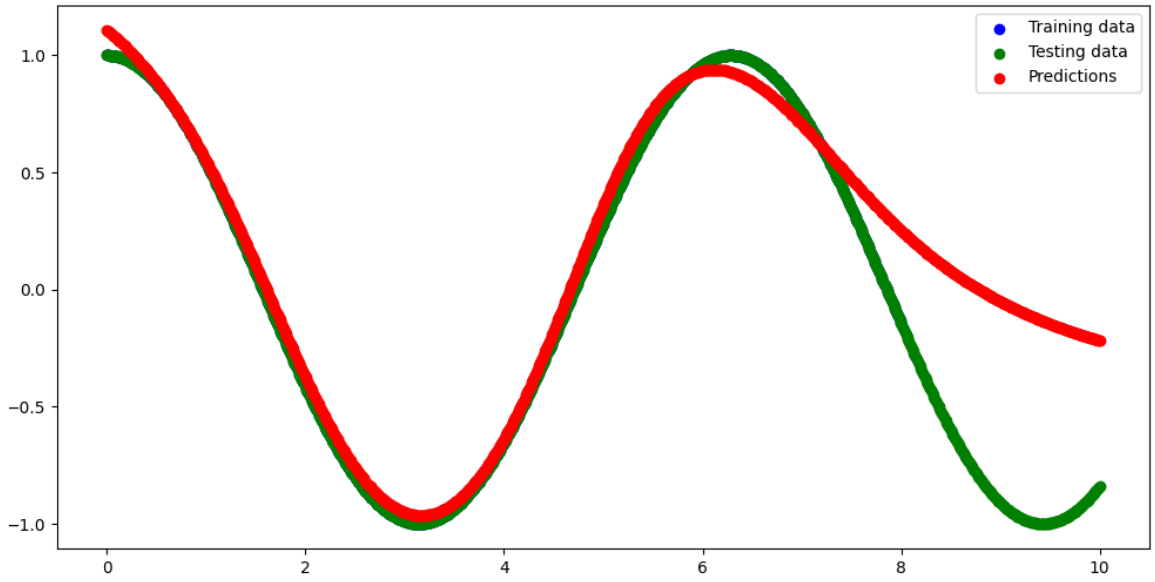
Sa slike možemo primjetiti da imamo odličnu predikciju na trening podacima, međutim predikcija na testnim podacima je dobra sve dok ne dođe do zakrivljenja kosinusne funkcije. Sada ćemo povećati broj neurona u svakom sloju, te ćemo ispratiti promjene:

7) Broj slojeva je 4, broj neurona po svakom sloju je (300,400,100,1), a mse = 7,4209 * 10^{-5}



I ovdje možemo primjetiti da nam predickija u dijelu testnih podataka nije dobra. Sada ćemo pokušati da probamo jos sigmoid aktivacijsku funkciju, te imamo sljedeći slučaj:

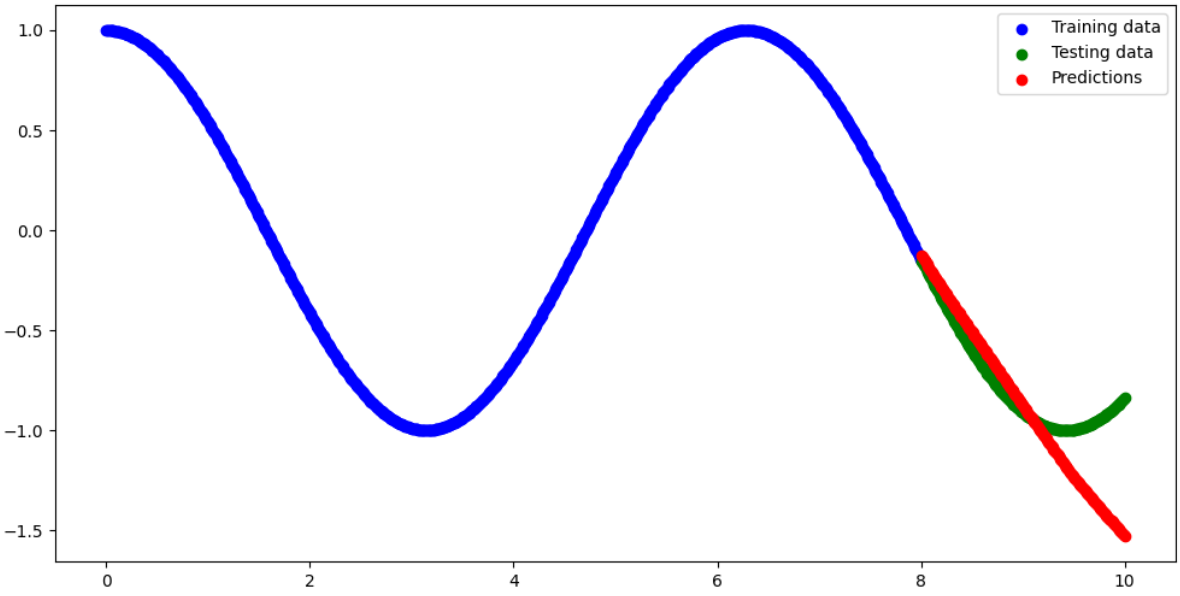
8) Broj slojeva je 4, broj neurona je (60,80,100,1), mse = 0.0068



Vidimo da nam ova funkcija daje lošije rezultate od prethodne dvije aktivacijske funkcije. Ukoliko do sada dva najbolja slučaja primjenimo na odnos podataka:

80% - trening podaci, 20% - testni podaci, dobijamo sljedeća dva slučaja:

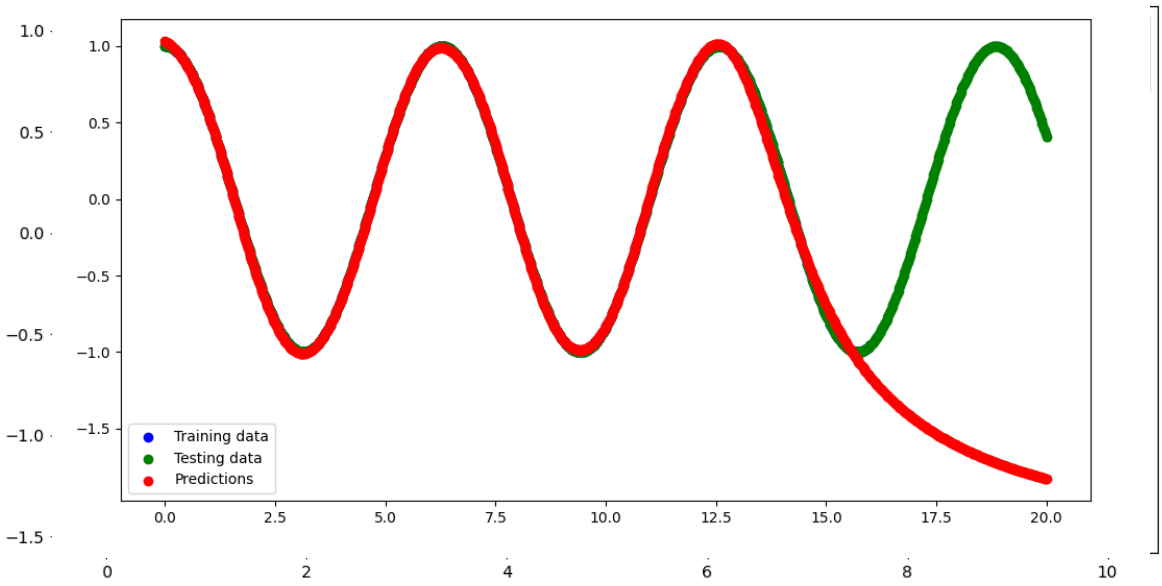
9) Broj slojeva je 4, broj neurona je (60,80,100,1), aktivacijska funkcija je: relu, mse = $2{,}9374 * 10^{-4}$



Vidimo je u ovom slučaju, predikcija izrazito loša, odnosno, da nam predviđeni podaci ne prate kosinusnu krivu.

U ovom slučaju je aktivacijska funkcija tanh, međutim opet vidimo da je predikcija izrazito loša kada dođe do zakrivljenja kosinusne krive. Postavlja se piranje kakav će slučaj biti ako proširimo ovaj grafik. Navedni grafik je predstavljen na sljedećoj slici:

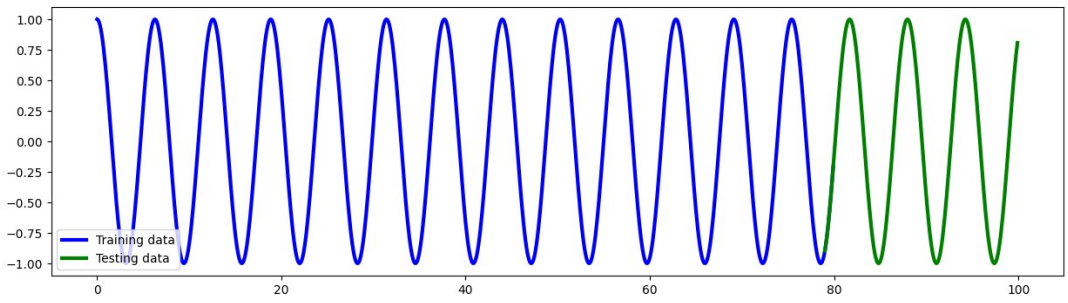
10) Broj slojeva je 4, broj neurona je (60,80,100,1), aktivacijska funkcija je: tanh, mse = 2,9661 * 10^{-5}



Ovdje možemo primjetiti da imamo ponovo dobru predikciju kada su nam testni podaci jednaki trening podacima, međutim, kada dobijemo nove testne podatke, naša mreža ne zna kako da se ponaša. Prema tome, možemo zaključiti da imamo pogrešan pristup rješavanju ovog problema. Drugi pristup koji ćemo pokušati jeste pristup sa LSTM vremenskom serijom.

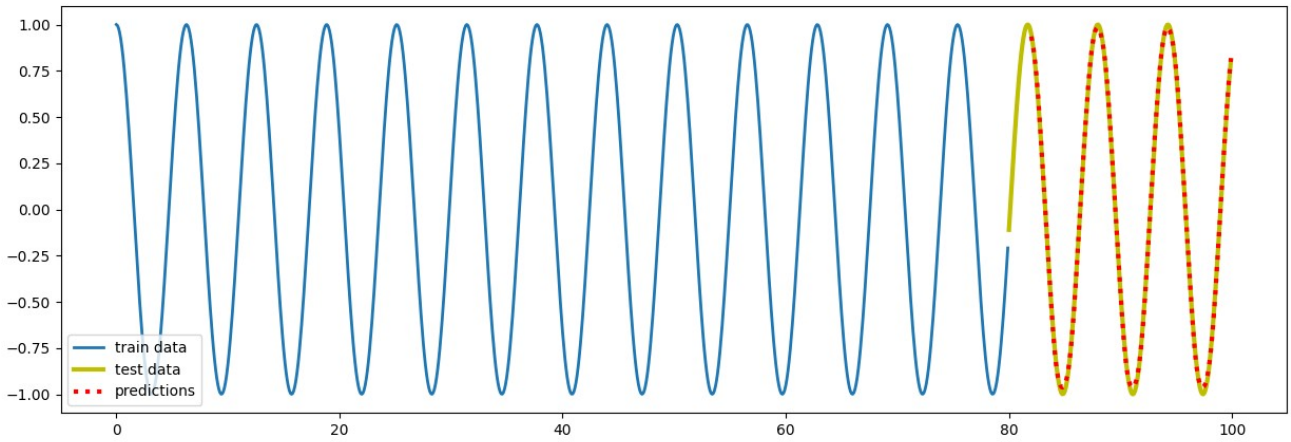
Pristup sa LSTM vremenskom serijom

(LSTM) je vještačka neuronska mreža koja se koristi u oblastima vještačke inteligencije i dubokog učenja. Za razliku od standardnih neuronskih mreža naprijed (feedforward), LSTM ima povratne veze. Takva rekurentna neuronska mreža može obraditi ne samo pojedinačne tačke podataka (kao što su slike), već i čitave nizove podataka (kao što su govor ili video). Ponovo ćemo posmatrati identičnu kosinusnu funkciju koja je prikazana na narednoj slici, te su razdvojeni podaci za učenje i podaci za treniranje.



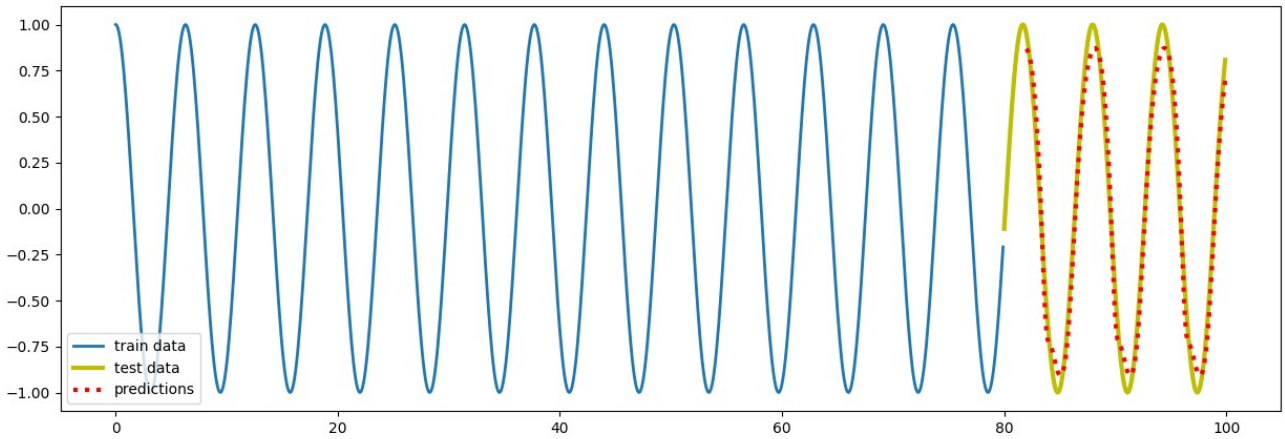
Nakon toga smo kreirali odgovarajuće treninrajuće i testirajuće serije. Te potom, kao i u prethodnim slučajevima kreirali odgovarajući model, pri čemu smo ovdje koristili rekurentnu mrežu LSTM (mreža sa povratnim vezama). Sada smo smanjili i broj epoha te broj neurona, odnosno imamo:

11) Broj slojeva je 2 (pri čemu je jedan LSTM, a drugi Dense) broj neurona po sloju je (6,1), mse = $2{,}672 * 10^{-4}$



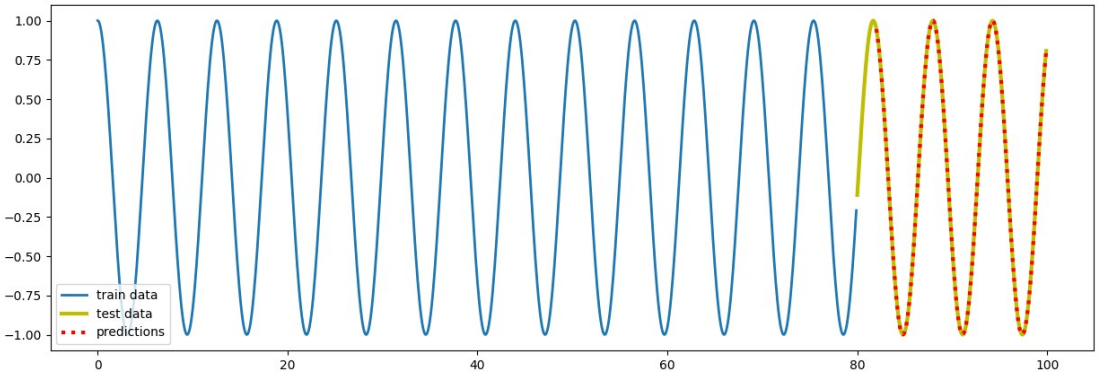
Ovdje vidimo da nam je predikcija (crvena isprekidana linija), skoro idealna, odnosno idealno pratimo kosinusnu funkciju. Sada možemo mijenjati broj neurona u LSTM sloju. Smanjivam broje neurona na 2 u LSTM sloju, dobijamo nešto lošiju predikciju, a povećanjem broja ovih neurona predkcija nam ostaje istog kvaliteta, odnosno imamo idealno praćenje kosinusne krive.

12) Broj slojeva 2 (LSTM, Dense), broj neurona po sloju (2,1), mse = 0.0096



Vidimo da nam je sa 2 neurona kod LSTM-a predikcija nešto slabija nego u slučaju sa 6 neurona.

13) Broj slojeva 2 (LSTM, Dense), broj neurona po sloju (12,1), mse = $2.0738 * 10^{-5}$



U ovom slučaju je predikcija, kao što je i bilo očekivano, skoro pa idealna. Za navedenu analizu, odnosno za sve primjere će biti prikazana tabela u kojoj ćemo sumirati broj epoha, aktivacijsku funkciju, broj slojeva te broj neurona po slojevima, i konačno vrijednost MSE-a

Redni broj	Aktiv. f-ja	Broj epoha	Broj slojeva i vrsta sloja	Broj neurona po svakom sloju	MSE (Mean Square Error)
1.	default	20	2 (Dense)	(2,1)	0.4620
2.	relu	100	4 (Dense)	(20,30.10,1)	0.4178
3.	relu	100	4 (Dense)	(60.80,100,1)	$2.96*10^{-5}$
4.	relu	100	5 (Dense)	(300,400,400,300,1)	0.0023
5.	relu	100	6 (Dense)	(300,400,400,300,500,1)	0.0017
6.	tanh	100	4 (Dense)	(60,80,100,1)	$2.96*10^{-5}$
7.	tanh	100	4 (Dense)	(300,400,100,1)	$7,4209 * 10^{-5}$
8.	sigmoid	100	4 (Dense)	(60,80,100,1)	0.0068
9.	relu/linear	100	4 (Dense)	(60.80.100,1)	$2,9374 * 10^{-4}$
10.	tanh/linear	100	4 (Dense)	(60,80,100,1)	$2,9661 * 10^{-5}$
11.	default	20	2 (LSTM, Dense)	(6,1)	$2,672 * 10^{-4}$
12.	default	20	2 (LSTM, Dense)	(2,1)	0.0096
13.	default	20	2 (LSTM, Dense)	(12,1)	$2.0738 * 10^{-5}$

U nastavku dokumenta je predstavljen kod za navednu analizu. Prvi pristup:

```
import math
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
print(tf.__version__)
X = np.linspace(0, 20, 1000)
y = np.cos(X)
X.shape, y.shape
len(X)
X_train = X[:800]
y_train = y[:800]
X_test = X[0:1000]
y_test = y[0:1000]
len(X_train), len(X_test)

plt.figure(figsize=(12, 6))
plt.scatter(X_train, y_train, c='b', label='Training data')
plt.scatter(X_test, y_test, c='g', label='Testing data')
plt.show()
tf.random.set_seed(42)
model_1 = tf.keras.Sequential([
tf.keras.layers.Dense(1, activation=tf.keras.activations.tanh),
])
model_1.compile(loss=tf.keras.losses.mse,
optimizer=tf.keras.optimizers.Adam(),
metrics=['mse'])
model_1.fit(tf.expand_dims(X_train, axis=-1), y_train, epochs=20, verbose=1)
preds = model_1.predict(X_test)
preds

def plot_preds(traindata=X_train,

trainlabels=y_train,
testdata=X_test,
testlabels=y_test,
predictions=preds):
plt.figure(figsize=(12, 6))
plt.scatter(traindata, trainlabels, c="b", label="Training data")
plt.scatter(testdata, testlabels, c="g", label="Testing data")
plt.scatter(testdata, predictions, c="r", label="Predictions")
plt.legend()
```

```
plot_preds(traindata=X_train,
trainlabels=y_train,
testdata=X_test,
testlabels=y_test,
predictions=preds)
print(model_1.summary())
print(len(model_1.layers))
tf.random.set_seed(42)
model_2 = tf.keras.Sequential([
tf.keras.layers.Dense(60, activation=tf.keras.activations.tanh),
# povecan broj neurona daje bolji model
tf.keras.layers.Dense(80, activation=tf.keras.activations.tanh),
tf.keras.layers.Dense(100, activation=tf.keras.activations.tanh),
# zadnji sloj mora biti jednak dimenzijama sistema
tf.keras.layers.Dense(1)
])
model_2.compile(loss=tf.keras.losses.mse,
optimizer=tf.keras.optimizers.Adam(),
metrics=['mse'])
model_2.fit(tf.expand_dims(X_train, axis=-1), y_train, epochs=100, verbose=1)
preds_2 = model_2.predict(X_test)
plot_preds(predictions=preds_2)
print(model_2.summary())
print(len(model_2.layers))
plt.show()
```

Drugi pristup (LSTM):

```
from keras.layers import LSTM
from keras.layers import Dense
from keras.models import Sequential
from keras.preprocessing.sequence import TimeseriesGenerator
import math
import matplotlib.pyplot as plt
import numpy as np
X_train = np.arange(0, 80, 0.1)
y_train = np.cos(X_train)

X_test = np.arange(80, 100, 0.1)
y_test = np.cos(X_test)
```

```
n_features = 1
train_series = y_train.reshape((len(y_train), n_features))
test_series = y_test.reshape((len(y_test), n_features))
fig, ax = plt.subplots(1, 1, figsize=(15, 4))
ax.plot(X_train, y_train, lw=3, c='b', label='Training data')
ax.plot(X_test, y_test, lw=3, c='g', label='Testing data')
ax.legend(loc="lower left")
plt.show()
look_back = 20
train_generator = TimeseriesGenerator(train_series, train_series,
length=look_back,
sampling_rate=1,
stride=1,
batch_size=10)
test_generator = TimeseriesGenerator(test_series, test_series,
length=look_back,
sampling_rate=1,
stride=1,
batch_size=10)
n_neurons = 12
model = Sequential()
model.add(LSTM(n_neurons, input_shape=(look_back, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(train_generator, epochs=20, verbose=1)
test_predictions = model.predict(test_generator)
print(model.summary())
print(len(model.layers))

x = np.arange(82, 100, 0.1)
fig, ax = plt.subplots(1, 1, figsize=(15, 5))
ax.plot(X_train, y_train, lw=2, label='train data')
ax.plot(X_test, y_test, lw=3, c='y', label='test data')
ax.plot(x, test_predictions, lw=3, c='r', linestyle=':', label='predictions')
ax.legend(loc="lower left")
plt.show()

extrapolation = list()

seed_batch = y_test[:look_back].reshape((1, look_back, n_features))
current_batch = seed_batch
```

```
# extrapolate the next 180 values
for i in range(180):
predicted_value = model.predict(current_batch)[0]
extrapolation.append(predicted_value)
current_batch = np.append(current_batch[:, 1:, :], [
[predicted_value]], axis=1)


x = np.arange(82, 100, 0.1)
#x = np.arange(110, 200, 0.5)
fig, ax = plt.subplots(1, 1, figsize=(15, 5))
ax.plot(X_train, y_train, lw=2, label='train data')
ax.plot(X_test, y_test, lw=3, c='y', label='test data')
ax.plot(x, extrapolation, lw=3, c='r', linestyle=':', label='extrapolation')
ax.legend(loc="lower left")
plt.show()
```