**Міністерство освіти і науки**

**України Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**
**Кафедра обчислювальної техніки**

# ЗВІТ

лабораторної роботи №1

з курсу «Програмні засоби проєктування і реалізації неромережевих систем»

Тема: «Перцептрон»

Перевірив:

Шимкович В. М.

Виконав:

Студент Гр. ІП-01 Шпилька В.С.

Київ 2023

**Завдання:** Написати програму, що реалізує нейронні мережі для моделювання функції двох змінних. Функцію двох змінних, типу f(x+y) = $x^2+y^2$, обрати самостійно. Промоделювати на невеликому відрізку, скажімо від 0 до 10.

Дослідити вплив кількості внутрішніх шарів та кількості нейронів на середню відносну помилку моделювання для різних типів мереж (feed forward backprop, cascade - forward backprop, elman backprop):

Функція тренування:

```
def train(model_name,
          version,
          hidden_neurons,
          data_path = DATA_PATH,
          train_percent = TRAIN_PERCENT,
          val_percent = VAL_PERCENT,
          test_percent = TEST_PERCENT,
          save_folder = SAVE_FOLDER,
          epochs = EPOCHS,
          batch_size = BATCH_SIZE,
          lr = DEFAULT_LR):

    data = Dataloader(data_path)
    train, val, test = data.split(train_percent, val_percent, test_percent)

    model = None
    if(model_name == 'FeedForward'):
        model = FeedForwardModel(hidden_neurons=hidden_neurons)
    elif(model_name == 'CascadeForward'):
        model = CascadeForwardModel(hidden_neurons=hidden_neurons)
    elif(model_name == 'Elman'):
        model = ElmanModel(hidden_neurons=hidden_neurons)

    #values for schedules
    initial_learning_rate = 10**(-4)
    final_learning_rate = 10**(-7)
    learning_rate_decay_factor = (final_learning_rate / initial_learning_rate)**(1/epochs)
    steps_per_epoch = int(len(train)/batch_size)
```

```
learning_rate = lr
if(lr == -1):
    learning_rate = tf.keras.optimizers.schedules.ExponentialDecay(
                    initial_learning_rate=initial_learning_rate,
                    decay_steps=steps_per_epoch,
                    decay_rate=learning_rate_decay_factor
                )


model.compile(loss = 'mean_squared_error', metrics = ['mean_absolute_error'],
              optimizer = tf.keras.optimizers.SGD(
                  learning_rate=learning_rate))

path_to_save = save_folder + model_name + '/' + version + '/'

checkpoint_dir = path_to_save + "Checkpoints/"
checkpoint_path = checkpoint_dir + "cp-{epoch:04d}.ckpt"
checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             monitor='val_loss',
                             verbose=1,
                             save_weights_only = True,
                             mode='auto')

tf_path = path_to_save + "Model/tf"
fullModelSave = ModelCheckpoint(filepath=tf_path,
                             monitor='val_loss',
                             verbose=1,
                             save_best_only=True,
                             mode='auto')
```

```
log_dir = path_to_save + "Logs/"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

callbacks_list = [checkpoint, tensorboard_callback, fullModelSave]

model.fit(
    np.reshape(train[:, :2], (-1, 2)),
    train[:, 2],
    batch_size,
    epochs = epochs,
    validation_data = (np.reshape(val[:, :2], (-1, 2)), val[:, 2]),
    callbacks = callbacks_list,
    verbose = 1)
```

В функція передається назва моделі, версію під якої зберегти та кількість нейронів в кожному шарі. Функція створює модель, компілює її, створює функції для зберігання чекпоінтів, найкращої моделі та логів. Та саме тренування за допомогою model.fit

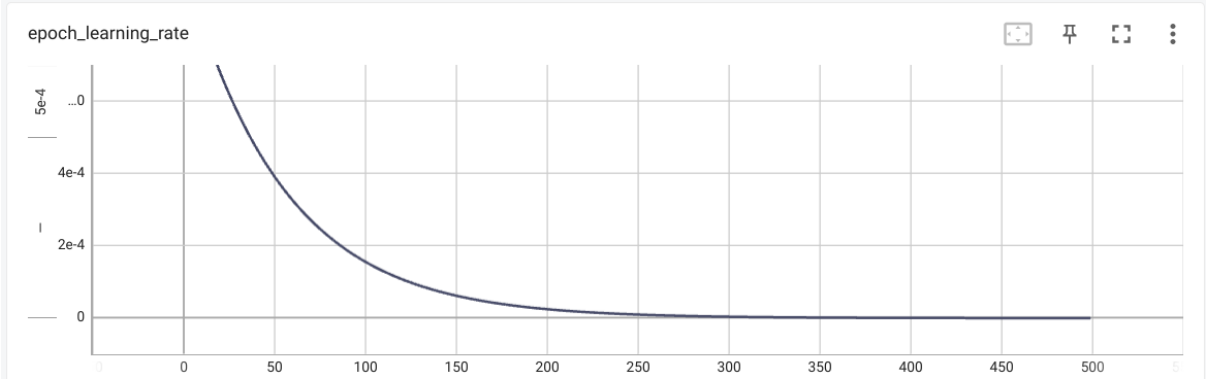
1. Тип мережі: feed forward backprop:

```
Models > feed_forward.py > FeedForwardModel > input_size
1   import tensorflow as tf
2
3   def FeedForwardModel(input_size = (2), hidden_neurons = [10]):
4       inputs = tf.keras.layers.Input(input_size)
5       current_layer = inputs
6       for neurons in hidden_neurons:
7           current_layer = tf.keras.layers.Dense(neurons, activation = 'relu')(current_layer)
8       outputs = tf.keras.layers.Dense(1, activation = 'relu')(current_layer)
9
10      model = tf.keras.Model(inputs, outputs)
11
12      return model
```

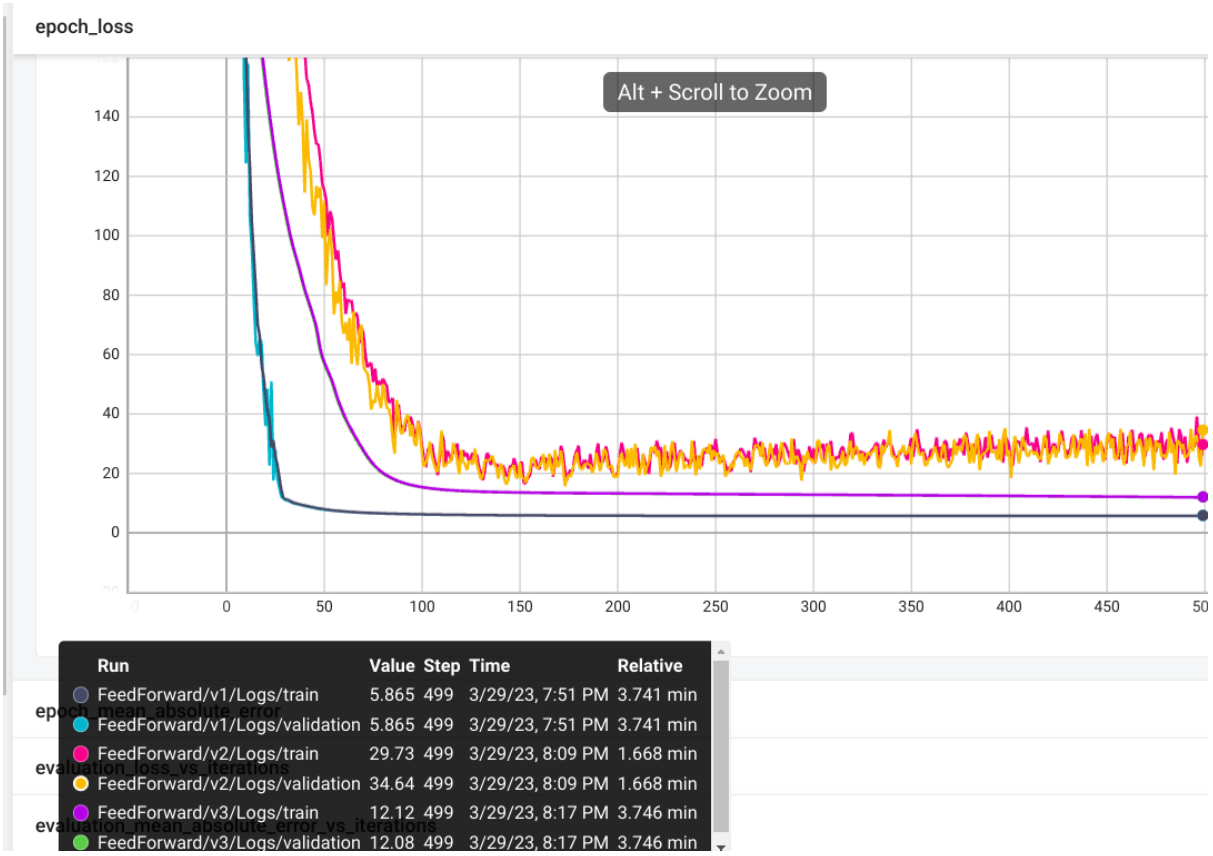
а) 1 внутрішній шар з 10 нейронами;

Одним з питанням яке було вирішено на даном кроці стало, що краще використати крок навчання константне число, чи ExponentialDecay, який зменшує learning rate під час навчання:

Функція зміни learning rate:



На наступному рисунку, показано як зменшується loss функції при різних learning rate:



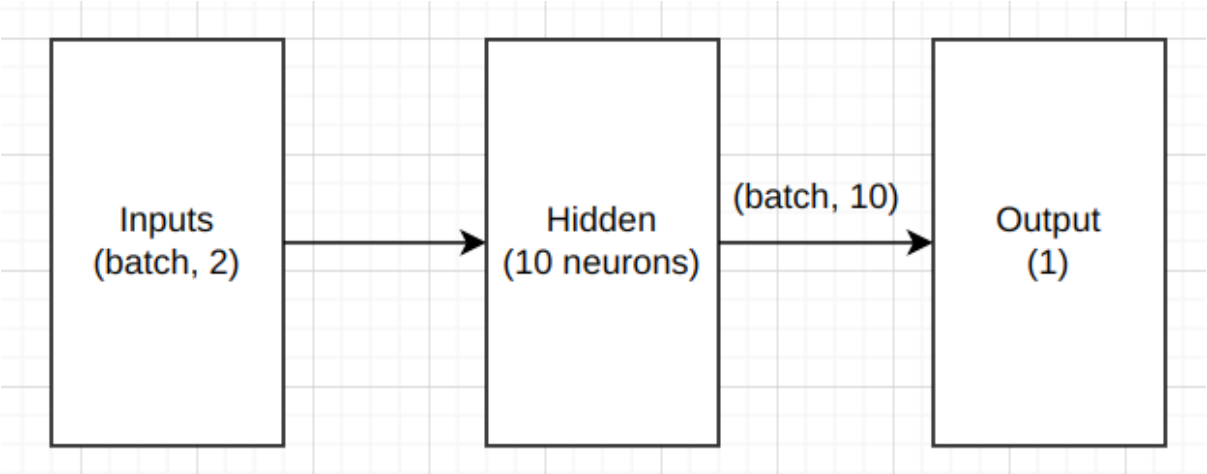

v1 – використання ExponentialDecay в межах 10e-3 до 10e-7

v2 – константне значення 10e-3

v2 – константне значення 10e-4

Як бачимо за допомогою ExponentialDecay ми швидше і ліпше отримали loss значення функції. Тому в майбутніх тренуваннях будемо використовувати його.

На наступному рисунку зображено архітектуру і приклад використання нейронної мережі:



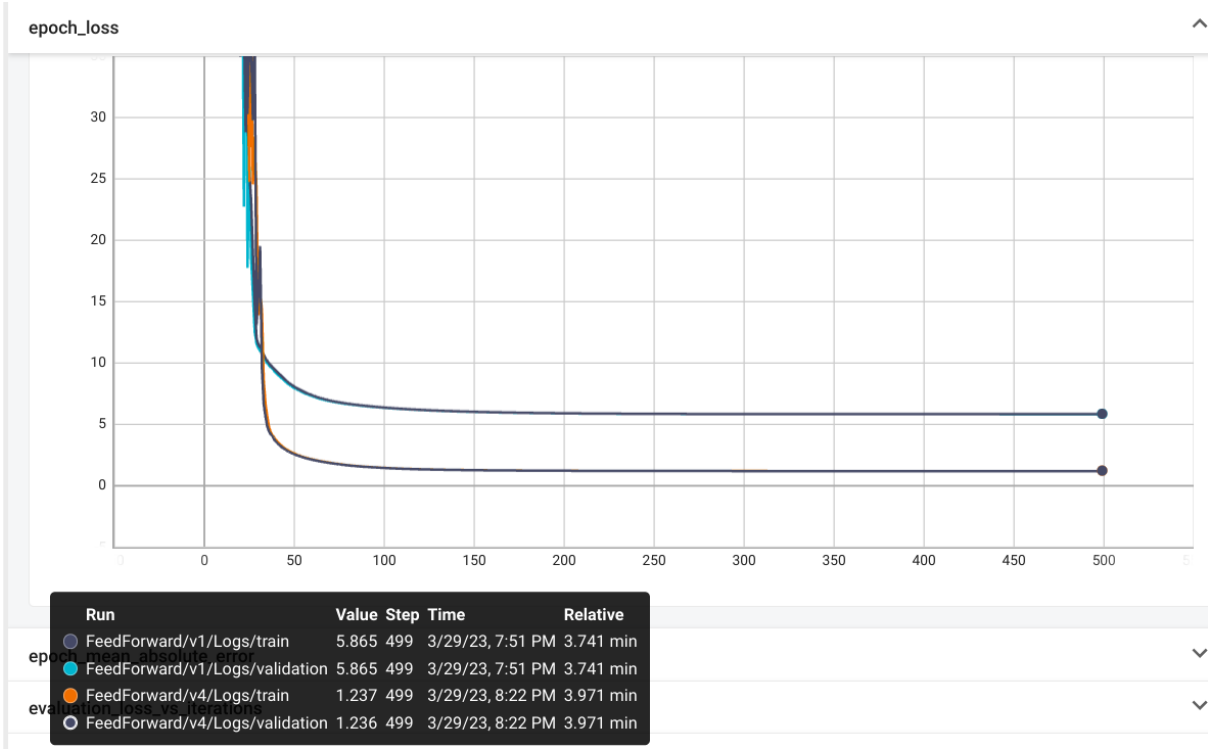

```
    model = tf.keras.models.load_model("Artifacts/Models/FeedForward/v1/Model/tf")
    model.summary()
    print("3^2 + 10^2 = ", model.predict([[3,10]], verbose=0))
```

[16] ✓ 0.2s

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 10)                30

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
3^2 + 10^2 =  [[104.78835]]
```

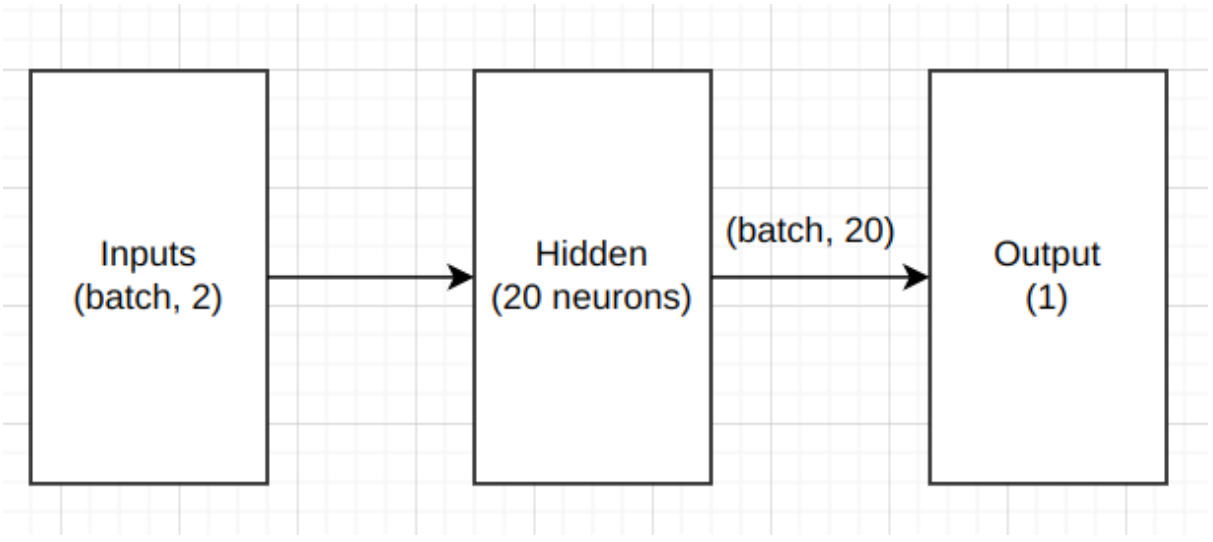
б) 1 внутрішній шар з 20 нейронами;

Різниця з попереднім випадком наведена на наступному рисунку:




Як бачимо додавання ще 10 нейронів значно покращило ситуацію.

Архітектура і приклад використання наведено нижче:



```
    model = tf.keras.models.load_model("Artifacts/Models/FeedForward/v4/Model/tf")
    model.summary()
    print("3^2 + 10^2 = ", model.predict([[3,10]], verbose=0))
```

[18] ✓ 0.2s

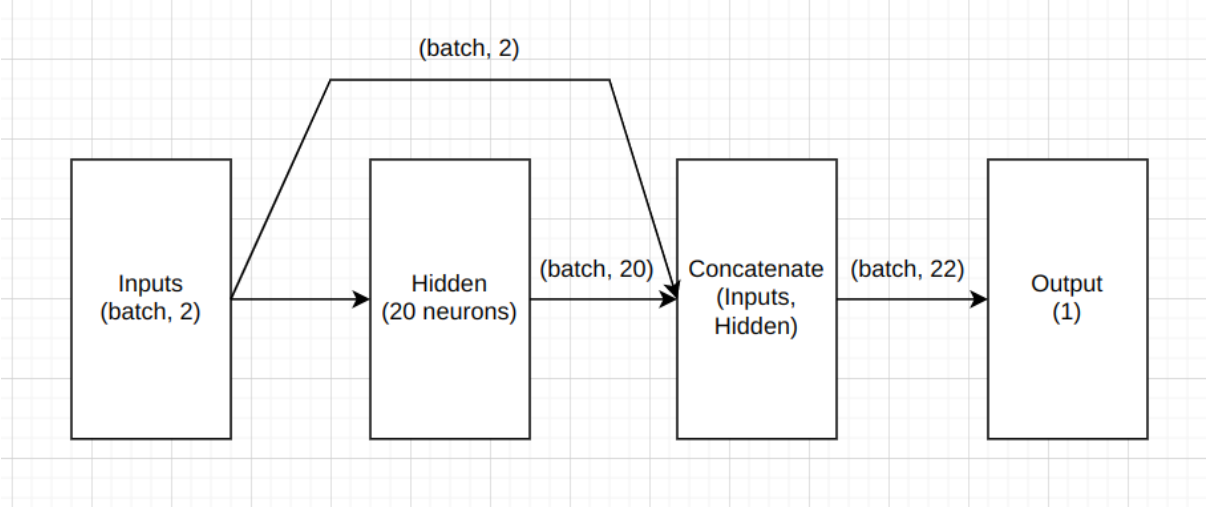
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 20)                60

 dense_1 (Dense)             (None, 1)                 21

=================================================================
Total params: 81
Trainable params: 81
Non-trainable params: 0
_________________________________________________________________
3^2 + 10^2 =  [[108.2513]]
```

2. Тип мережі CascadeForward:

```python
import tensorflow as tf

def CascadeForwardModel(input_size = (2), hidden_neurons = [10]):
    inputs = tf.keras.layers.Input(input_size)

    concat = inputs

    for neurons in hidden_neurons:
        hidden = tf.keras.layers.Dense(neurons, activation = 'relu')(concat)
        concat = tf.keras.layers.Concatenate(axis=-1)([concat, hidden])

    outputs = tf.keras.layers.Dense(1, activation = 'relu')(concat)

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

    return model
```

а) 1 внутрішній шар з 20 нейронами;

Архітектура і приклад використання:




```
model = tf.keras.models.load_model("Artifacts/Models/CascadeForward/v1/Model/tf")
model.summary()
print("3^2 + 10^2 = ", model.predict([[3,10]], verbose=0))
```

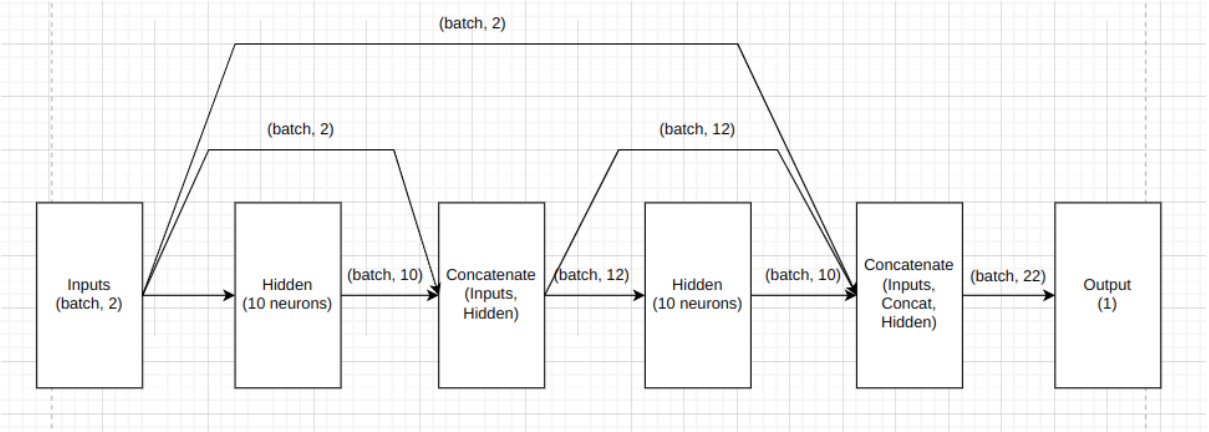
[20] ✓ 0.3s

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0         []

 dense (Dense)               (None, 20)                60        ['input_1[0][0]']

 concatenate (Concatenate)   (None, 22)                0         ['input_1[0][0]',
                                                                  'dense[0][0]']

 dense_1 (Dense)             (None, 1)                 23        ['concatenate[0][0]']

=================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
_________________________________________________________________
3^2 + 10^2 =  [[108.16865]]
```

+ Code + Markdown

б) 2 внутрішніх шари по 10 нейронів у кожному;

Архітектура і приклад використання:

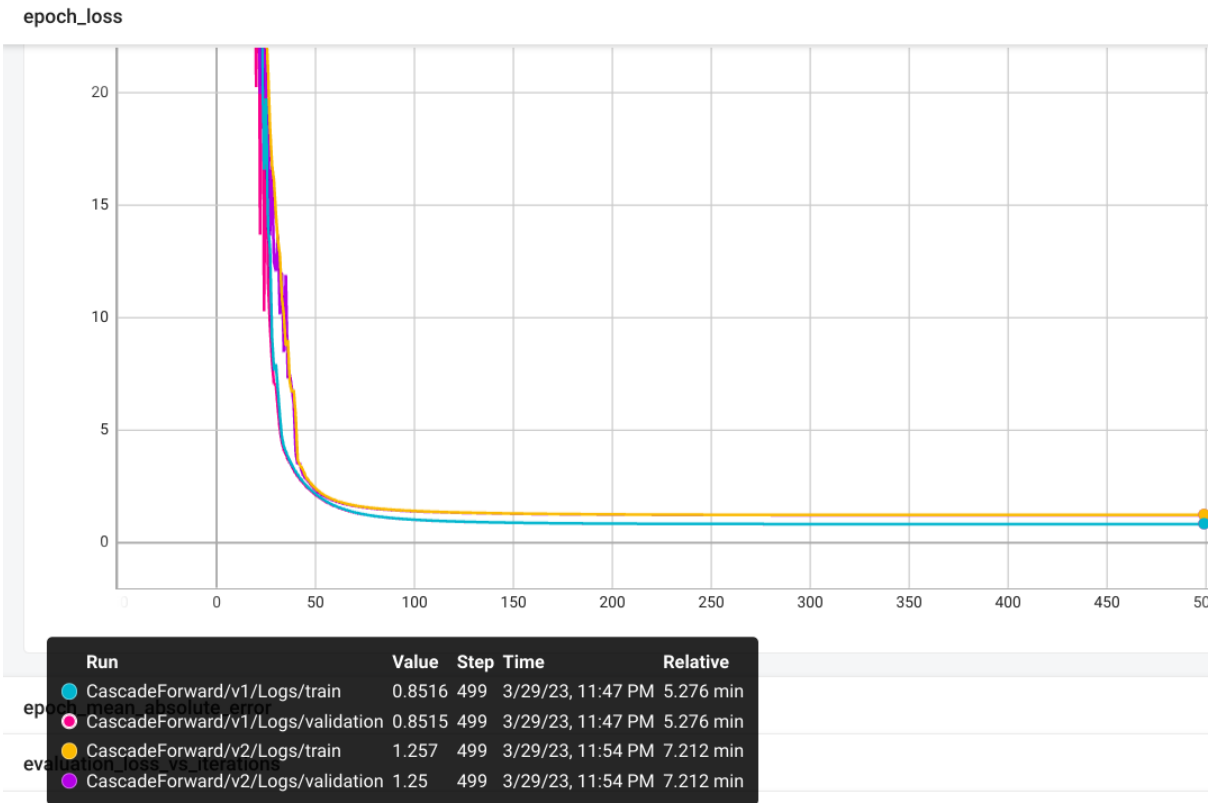



```
model = tf.keras.models.load_model("Artifacts/Models/CascadeForward/v2/Model/tf")
model.summary()
print("3^2 + 10^2 = ", model.predict([[3,10]], verbose=0))
```

[21] ✓ 0.5s

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0         []

 dense (Dense)               (None, 10)                30        ['input_1[0][0]']

 concatenate (Concatenate)   (None, 12)                0         ['input_1[0][0]',
                                                                  'dense[0][0]']

 dense_1 (Dense)             (None, 10)                130       ['concatenate[0][0]']

 concatenate_1 (Concatenate) (None, 22)                0         ['concatenate[0][0]',
                                                                  'dense_1[0][0]']

 dense_2 (Dense)             (None, 1)                 23        ['concatenate_1[0][0]']

=================================================================
Total params: 183
Trainable params: 183
Non-trainable params: 0
_________________________________________________________________
3^2 + 10^2 =  [[108.456055]]
```

Порівняння:



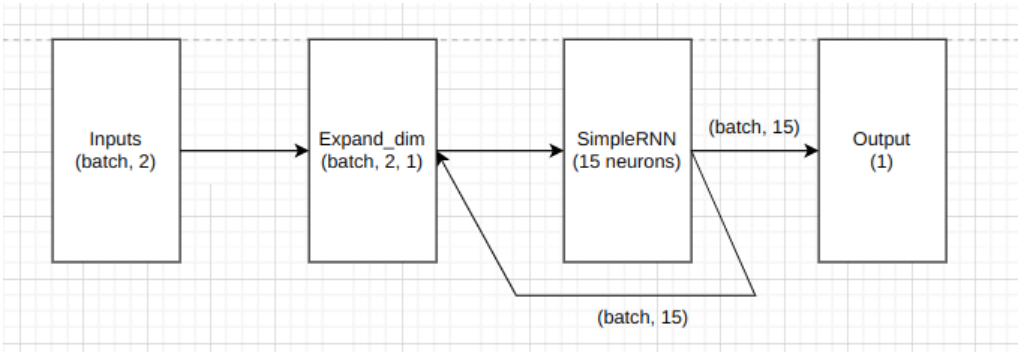

Нейрона мережа з одним прихованим шаром, але більшою кількістю нейронів показала трохи краще результати, ніж 2 шари, але менше нейронів.

3. Тип мережі Elman:

```python
import tensorflow as tf

def ElmanModel(input_size = (2), hidden_neurons = [10]):
    inputs = tf.keras.layers.Input(input_size)

    current_layer = tf.expand_dims(inputs, axis = 1)
    current_layer = tf.keras.layers.SimpleRNN(hidden_neurons[0])(current_layer)

    for neurons in hidden_neurons[1:]:
        current_layer = tf.expand_dims(current_layer, axis = 1)
        current_layer = tf.keras.layers.SimpleRNN(neurons, activation='relu')(current_layer)

    outputs = tf.keras.layers.Dense(1, activation = 'relu')(current_layer)

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

    return model
```

а) 1 внутрішній шар з 15 нейронами;

Архітектура і приклад використання:



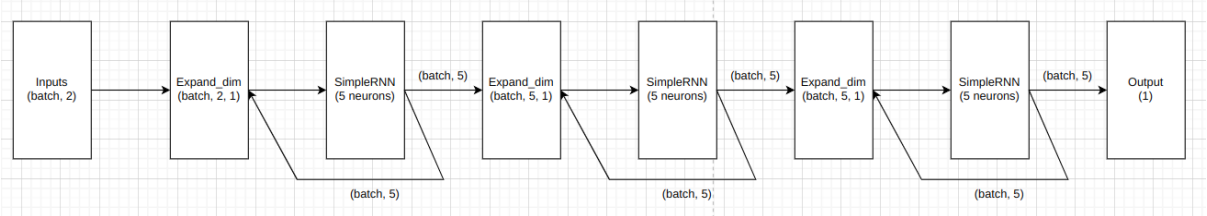

```
model = tf.keras.models.load_model("Artifacts/Models/Elman/v1/Model/tf")
model.summary()
print("3^2 + 10^2 = ", model.predict([[3,10]]))
```

[22] ✓ 0.6s

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 tf.expand_dims (TFOpLambda)  (None, 1, 2)             0

 simple_rnn (SimpleRNN)      (None, 15)                270

 dense (Dense)               (None, 1)                 16

=================================================================
Total params: 286
Trainable params: 286
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 115ms/step
3^2 + 10^2 =  [[106.150154]]
```

b) 3 внутрішніх шари по 5 нейронів у кожному;

Архітектура і приклад використання:

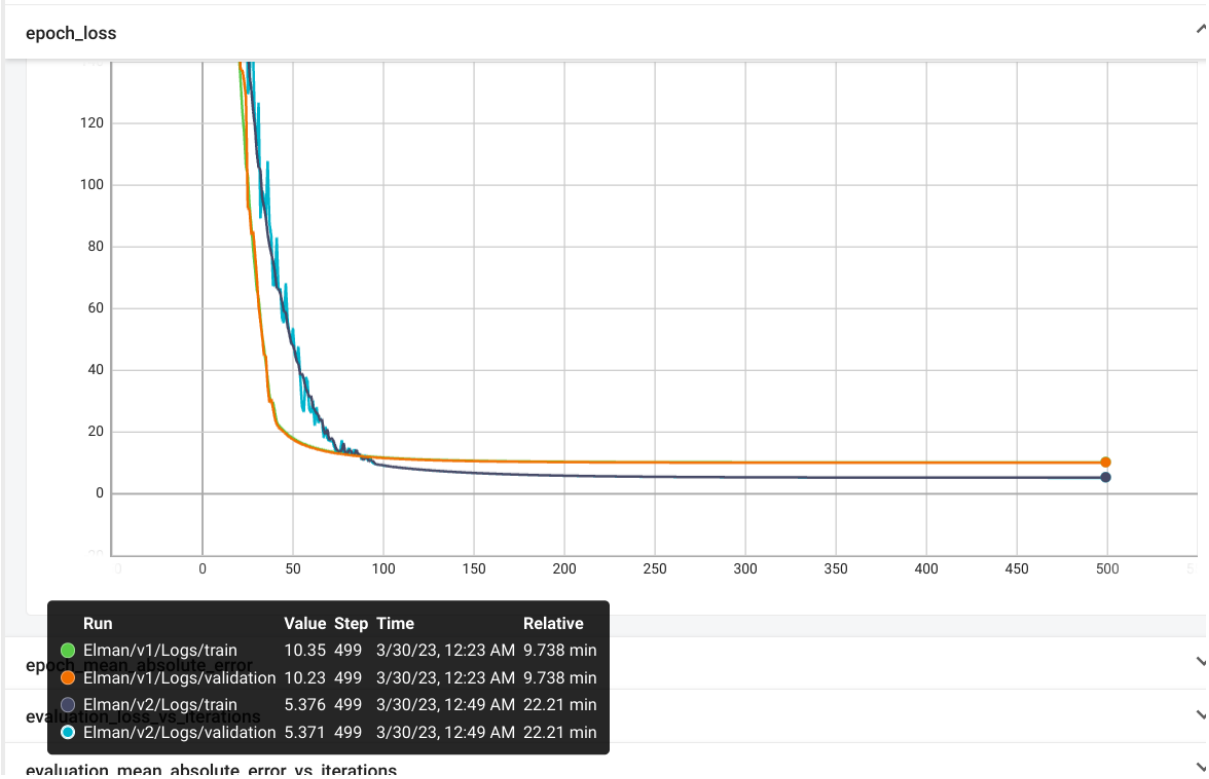



```
model = tf.keras.models.load_model("Artifacts/Models/Elman/v2/Model/tf")
model.summary()
print("3^2 + 10^2 = ", model.predict([[3,10]], verbose = 0))
```

[24] ✓ 1.5s

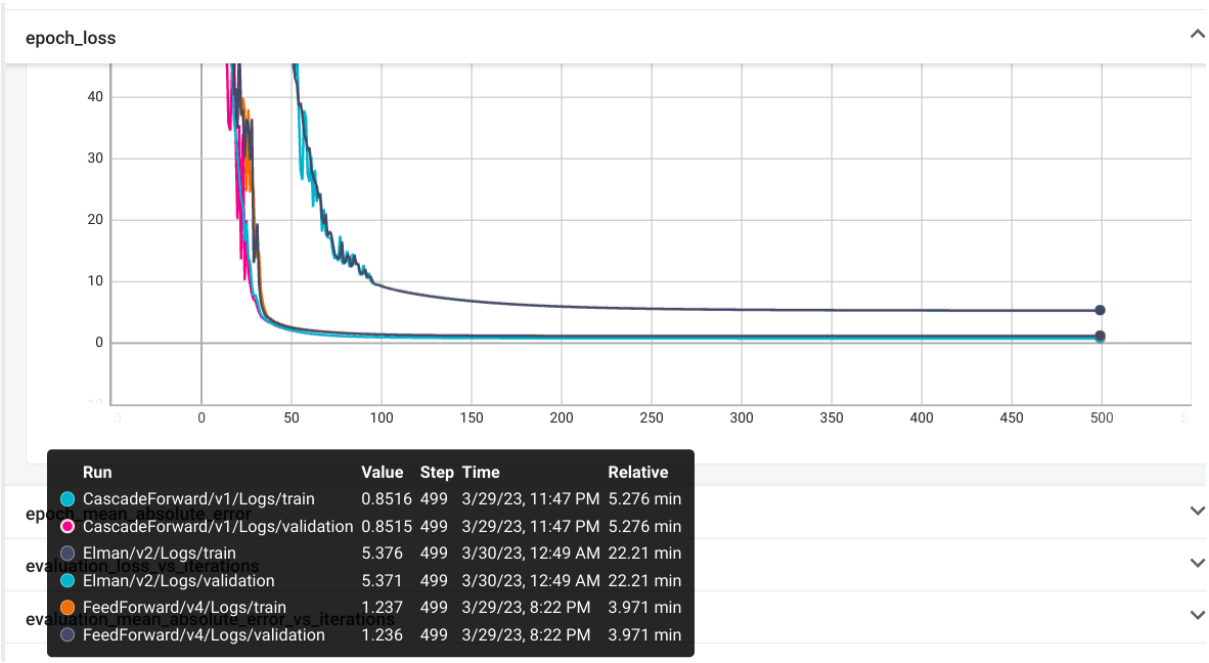
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 tf.expand_dims (TFOpLambda)  (None, 1, 2)             0

 simple_rnn (SimpleRNN)      (None, 5)                 40

 tf.expand_dims_1 (TFOpLambd  (None, 1, 5)             0
 a)

 simple_rnn_1 (SimpleRNN)    (None, 5)                 55

 tf.expand_dims_2 (TFOpLambd  (None, 1, 5)             0
 a)

 simple_rnn_2 (SimpleRNN)    (None, 5)                 55

 dense (Dense)               (None, 1)                 6

=================================================================
Total params: 156
Trainable params: 156
Non-trainable params: 0
_________________________________________________________________
3^2 + 10^2 =  [[105.95732]]
```

Порівняння:




Нейрона мережа з 3 прихованими шарами працює в загальному краще ніж з 1 шаром, але більшою кількістю нейронів.

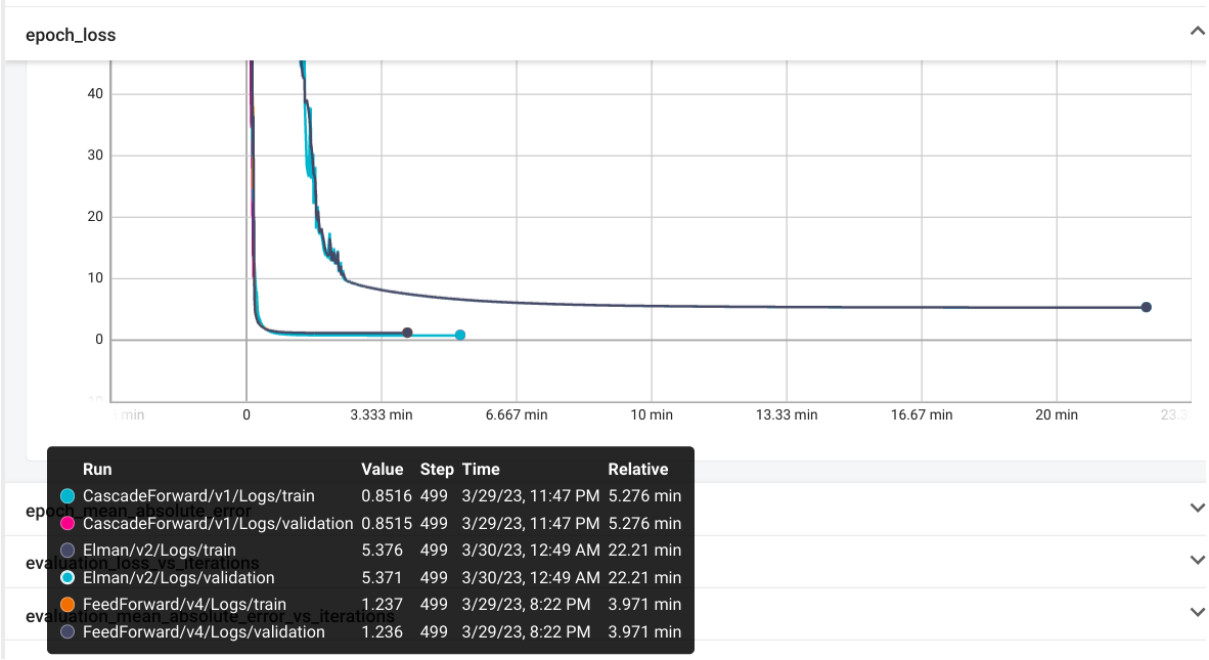
Загальне порівняння:




З графіку видно, що найкраще працює саме CascadeForward з одним прихованим шаром з 20 нейронами. Нейронна мережа Елмана показала

гірші результати, адже рекурентні нейронні мережі, скоріше потрібні для знаходження закономірностей в часових рядах, а не звичайних функціях.

Що стосується часу тренування:




FeedForward тренується найшвидше, CascadeForward трохи повільніше, Elman дуже повільно.

**Висновки:**

В даній лабораторній роботі було розглянуто 3 види нейронних мереж. Було розглянуто користь від використання ExponentionalDecay для зміни кроку навчання та показано перевагу даного підходу перед константним значенням. Було розглянуто зміну результату від зміни кількості нейронів у шару. Можна однозначно сказати, чим більше нейронів в одному шарі тим краще, але якщо збільшувати кількість шарів жертвуючи кількістю нейронів, то результат може бути неочікуваним і залежить від конкретних даних та ситуації. Також було зроблено висновок, що мережа Елмана не підходить для задачі регресії, і скоріше за все її варто використовувати для прогнозування в часових рядах.