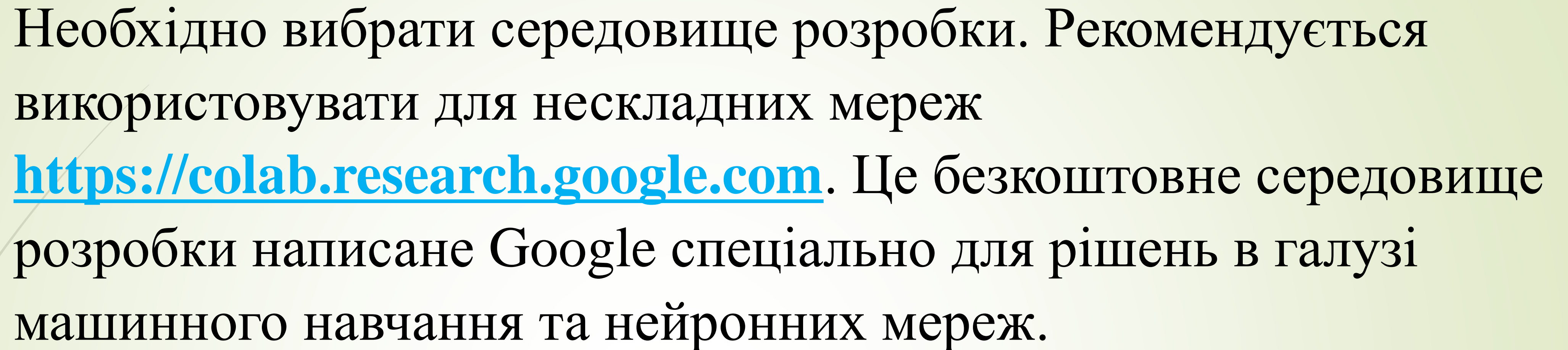
ЛЕКЦІЯ 14

Принципи створення нейронних мереж при роботі з Tensorflow



Необхідно вибрати середовище розробки. Рекомендується використовувати для нескладних мереж https://colab.research.google.com. Це безкоштовне середовище розробки написане Google спеціально для рішень в галузі машинного навчання та нейронних мереж.

Основні кроки:

1. Передобробка даних.
2. Побудова моделі.
3. Компіляція моделі.
4. Вивчення моделі.
5. Оцінка моделі.

Попередня обробка даних:

Результат роботи нейромережі наполовину залежить від того, наскільки добре підготовані дані для моделі. Пропущені дані або неправильна розмірність може звести нанівець всі зусилля. Тому важливо приділити цьому етапу час і увагу.

Насправді для обробки даних можна використовувати те, що зручніше. Можна писати свої функції на Python або використовувати бібліотеку sklearn.preprocessing.

У самому tensorflow також є засоби для передобробки в модулі tf.keras.preprocessing. Зокрема великий вибір інструментів для зображень та тексту.

На що звернути увагу насамперед:

«Провали» в даних.

Ситуацію можна виправити кількома способами. Якщо таких записів небагато, і ними можна знехтувати, то їх можна видалити. Також їх можна замінити: це може бути середнє або найчастіше використовується для даної фічі.

Нормалізація даних.

Приведе значення до певного діапазону. Наприклад [-1; 1] або z-масштабування (у sklearn це StandardScaler).

Обробка категоріальних даних.

Як правило, це створення під кожну категорію своєї фічі. Можна використовувати OneHotEncoder зі sklearn або get_dummies в pandas.

Також необхідно розбити дані на 3 вибірки:

- навчальна,
- валідаційну,
- тестову.

Навчальний набір даних - це те, на чому безпосередньо навчається модель. Валідація потрібна для перевірки вашої моделі. У процесі навчання мережа може дуже підлаштуватися під дані, на яких навчається. В результаті така модель не зможе працювати з даними, відмінними від навчального набору. Це називається перенавчанням. Щоб уникнути цього потрібен валідаційний набір, на якому контролюється, як модель передбачає. Тестовий набір використовується лише один раз, коли мережа повністю навчена для фінальної оцінки. Після цього мережа не доучується і змінюється. Зазвичай на валідаційний набір виділяється 20-30% даних та на тестовий 5-10%.

Побудова моделі

Найпростіше створити модель за допомогою класу **Sequential**, що приймає список шарів. Щоб подивитися вміст моделі, потрібно викликати метод **summary()**. Важливо визначити розмір вхідних даних. Це можна зробити за допомогою атрибута **input_shape** у першому шарі.

```
import tensorflow as tf

model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(32, activation='relu', 
input_shape=(10,), name='hidden_layer_1'),
  tf.keras.layers.Dropout(0.2, name='dropout'),
  tf.keras.layers.Dense(10, name='hidden_layer_2')
])
model.summary()
```

Вивід:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden_layer_1 (Dense)       (None, 32)                352
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
hidden_layer_2 (Dense)       (None, 10)                330
=================================================================
Total params: 682
Trainable params: 682
Non-trainable params: 0
_________________________________________________________________
```

Шари можна додавати і послідовно, використовуючи метод add(). Наприклад, додамо пару додаткових шарів і викличемо **`summary()`**.

```
model.add(tf.keras.layers.Dense(5, activation='relu',
name='hidden_layer_3'))
model.add(tf.keras.layers.Dense(2, name='output'))
model.summary()
```

Вивід:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden_layer_1 (Dense)       (None, 32)                352
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
hidden_layer_2 (Dense)       (None, 10)                330
_________________________________________________________________
hidden_layer_3 (Dense)       (None, 5)                 55
_________________________________________________________________
output (Dense)               (None, 2)                 12
=================================================================
Total params: 749
Trainable params: 749
Non-trainable params: 0
```

Зауважте, до моделі додалося ще два шари. Внизу вказується кількість параметрів, які необхідно навчити. Відповідно, чим більше параметрів, тим довше навчатиметься ваша мережа.

Sequential зручний якщо у вас невелика мережа з послідовною структурою, де шари йдуть один за одним, є тільки один вхід та один вихід. Але можуть бути складніші варіанти моделей.

Для цього в tensorflow є функціональний API. Перепишемо модель вище.

```
input = tf.keras.Input(shape=(10,), name='input')
layer_1 = tf.keras.layers.Dense(32, activation='relu',
name='hidden_layer_1')(input)
dropout = tf.keras.layers.Dropout(0.2, name='dropout')(layer_1)
layer_2 = tf.keras.layers.Dense(10, name='hidden_layer_2')(dropout)
layer_3 = tf.keras.layers.Dense(5, activation='relu',
name='hidden_layer_3')(layer_2)
output = tf.keras.layers.Dense(2, name='output')(layer_3)

model_2 = tf.keras.Model(
    inputs=input,
    outputs=output,
)

model_2.summary()
```

Вивід:

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 10)]              0
_________________________________________________________________
hidden_layer_1 (Dense)       (None, 32)                352
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
hidden_layer_2 (Dense)       (None, 10)                330
_________________________________________________________________
hidden_layer_3 (Dense)       (None, 5)                 55
_________________________________________________________________
output (Dense)               (None, 2)                 12
=================================================================
Total params: 749
Trainable params: 749
Non-trainable params: 0
_________________________________________________________________
```

Тепер кожен шар є функцією, яка приймає на вхід результат роботи попереднього шару. Структуру такої моделі можна вивести наочно з усіма залежностями шарів один від одного (Рис.1).

```
tf.keras.utils.plot_model(model_2, show_shapes=True)
```

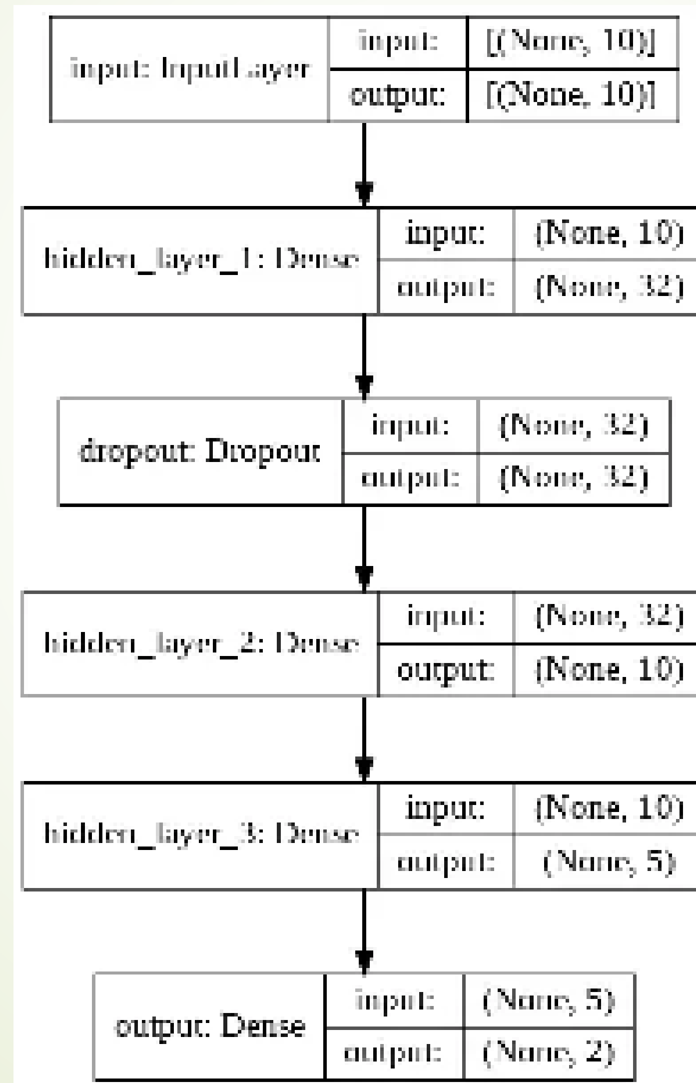


Рис.1 Структура залежності шарів.

За допомогою функціонального API можна створити структуру з кількома входами та виходами (Рис.2).

```
# Назначимо два окремих входи
input_1 = tf.keras.Input(shape=(10,), name='input_1')
input_2 = tf.keras.Input(shape=(20,), name='input_2')

# Визначимо структуру для обробки першого входу
layer_1 = tf.keras.layers.Dense(32, name='layer_1')(input_1)

# Для другого входу
layer_2 = tf.keras.layers.Dense(32, name='layer_2')(input_2)
layer_3 = tf.keras.layers.Dense(16, name='layer_3')(layer_2)

# Об’єднаємо
concatenate = tf.keras.layers.concatenate([layer_1, layer_3])

# Визначимо два виходи
output_1 = tf.keras.layers.Dense(1, name='output_1')(concatenate)
output_2 = tf.keras.layers.Dense(1, name='output_2')(concatenate)

model_3 = tf.keras.Model(
    inputs=[input_1, input_2],
    outputs=[output_1, output_2],
)

tf.keras.utils.plot_model(model_3, show_shapes=True)
```

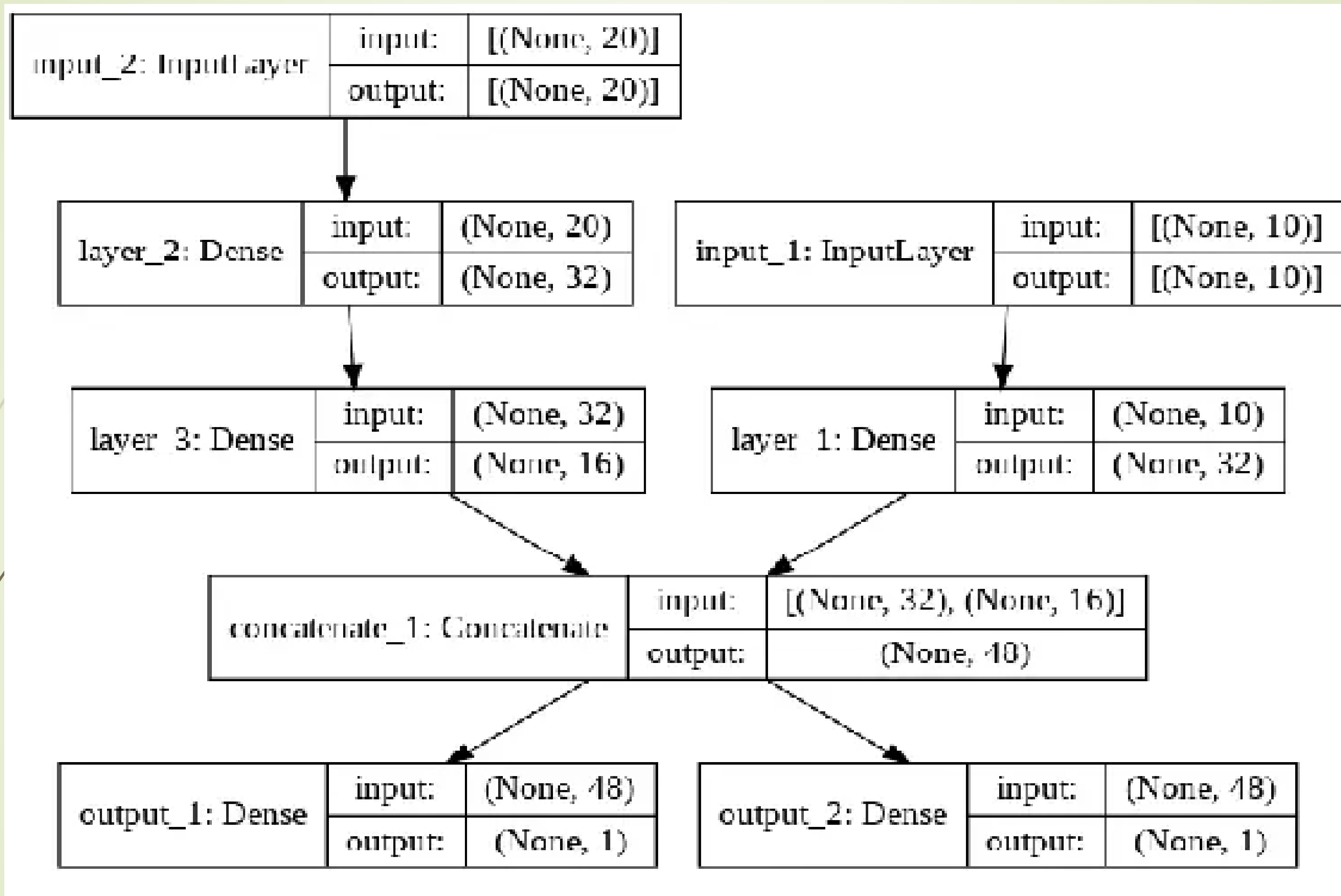


Рис.2 Структура з кількома входами та виходами.

Компіляція моделі

Після того, як структура моделі готова, її потрібно скомпілювати. Для цього модель має метод `compile()`. Головне, що можна визначити за компіляції - це функція втрат. Власне, від неї залежить результат навчання. Наприклад, середньоквадратична помилка сильніше «карає» модель за викиди, ніж середня абсолютна помилка. Що саме використовувати залежить від завдання. Якщо вам потрібно передбачити ціну житла, то швидше за все викиди погано вплинуть на результат. І тут більше підійде середня абсолютна помилка. Якщо ви пророкуєте рух цін на фондовому ринку, то занадто велике відхилення від ціни буде на шкоду, і тут краще середньоквадратична помилка.

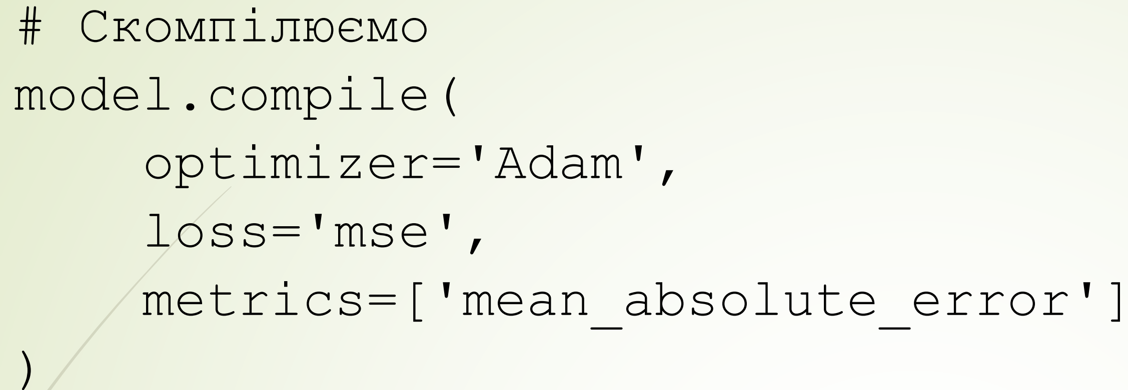
Ще один важливий параметр – це алгоритм оптимізації. Добре підібраний оптимізатор дозволить вам швидше навчити модель та по можливості уникнути локальних мінімумів.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(5, input_shape=(10,),
name='hidden_layer_1'),
  tf.keras.layers.Dense(2, name='output')
])

model.compile(

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_l
ogits=True), # Функція втрат
    optimizer='Adam', # Оптимізатор
    metrics=[ # Метрики
        'accuracy', # Якщо у об’єкта призначено ім’я, то
можна викликати об’єкт з використанням імені
        tf.keras.metrics.Precision()
    ]
)
```

Навчання моделі

Після того, як модель скомпільована, її потрібно навчити. Для цього застосовується метод **`fit()`**. Він приймає вхідні дані та очікувані відповіді мережі. Можна вказати масив з валідаційними даними, максимальну кількість епізодів та багато іншого. Простий приклад:

```
import numpy as np

# Ініціалізація набору даних випадковими числами.
X = np.array(np.random.random((100, 5))) # Матриця 100 на 5 з
діапазоном значень [0;1]
Y = np.array(np.random.random((100))) # Вектор довжини 100 с
діапазоном значень [0;1]
# Створимо модель
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(3, input_shape=(5,)),
  tf.keras.layers.Dense(1)
])
```

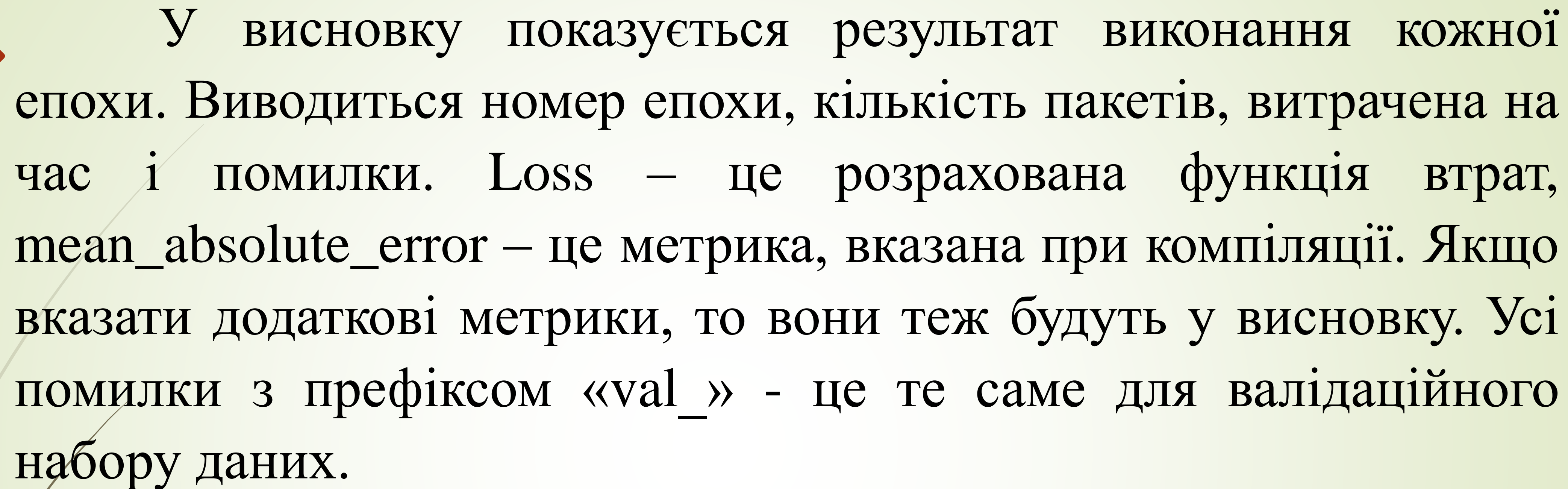


```
# Скомпілюємо
model.compile(
    optimizer='Adam',
    loss='mse',
    metrics=['mean_absolute_error']
)

# Навчимо
model.fit(
    X, # Набір вхідних даних
    Y, # Набір вірних відповідей
    validation_split=0.2, # Цей приклад автоматично виділить
частину навчального набору на валідаційні данні. В даному
випадку 20%
    epochs=10, # Процес навчання завершиться через 10 епох
    batch_size = 8 # Набір даних буде розбитий на пакети
(батчі) по 8 елементів набору в кожному.
```

Вихід:

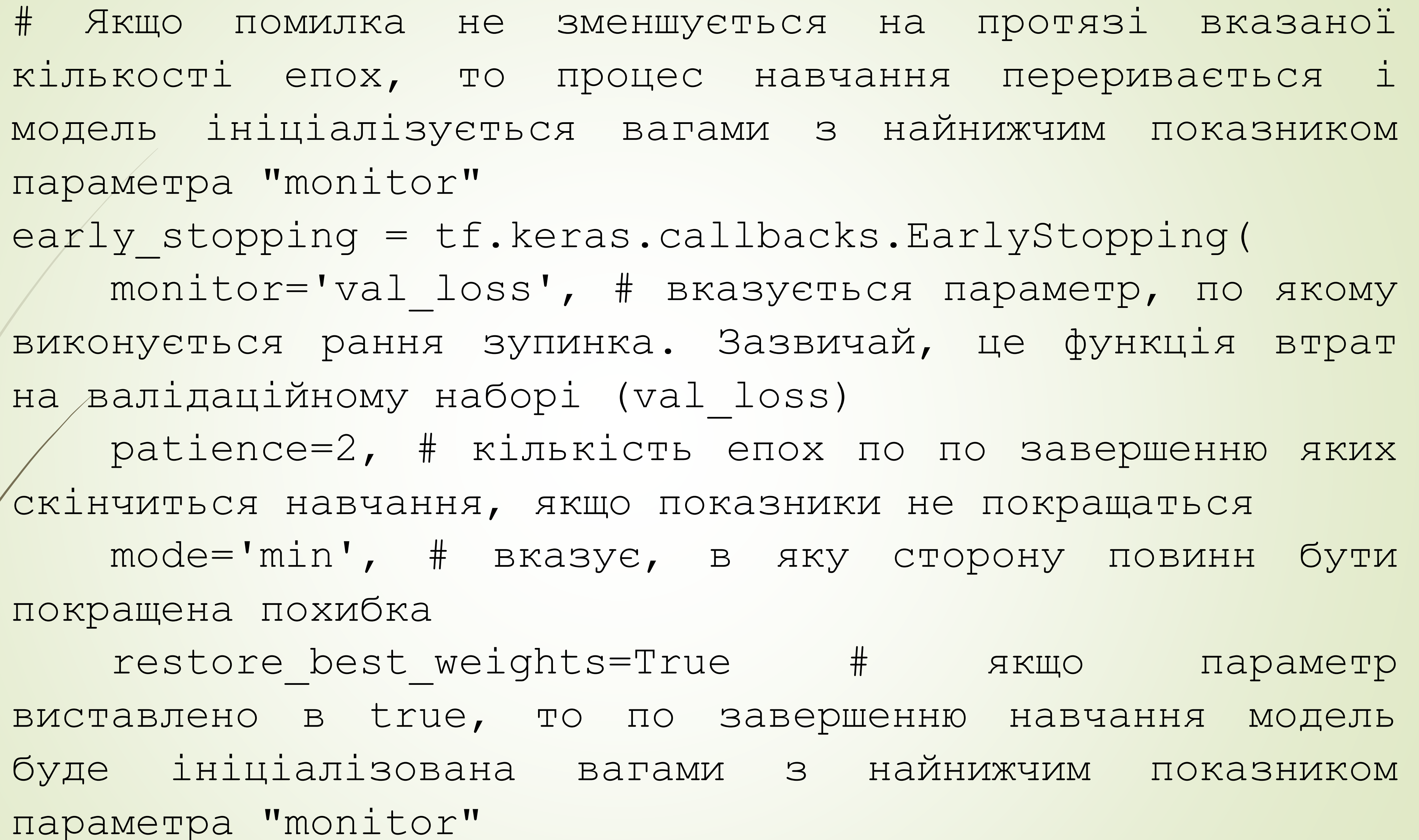
```
Epoch 1/10
10/10 [==============================] - 0s 16ms/step -
loss: 0.2524 - mean_absolute_error: 0.4273 - val_loss:
0.2713 - val_mean_absolute_error: 0.4401
Epoch 2/10
10/10 [==============================] - 0s 3ms/step -
loss: 0.2353 - mean_absolute_error: 0.4063 - val_loss:
0.2420 - val_mean_absolute_error: 0.4183
Epoch 3/10
10/10 [==============================] - 0s 5ms/step -
loss: 0.2256 - mean_absolute_error: 0.4017 - val_loss:
0.2348 - val_mean_absolute_error: 0.4122
Epoch 4/10
10/10 [==============================] - 0s 4ms/step -
loss: 0.2203 - mean_absolute_error: 0.3968 - val_loss:
0.2309 - val_mean_absolute_error: 0.4095
```

```
Epoch 5/10
10/10 [==============================] - 0s 4ms/step - loss: 0.2166 -
mean_absolute_error: 0.3937 - val_loss: 0.2251 - val_mean_absolute_error:
0.4045
Epoch 6/10
10/10 [==============================] - 0s 3ms/step - loss: 0.2094 -
mean_absolute_error: 0.3857 - val_loss: 0.2262 - val_mean_absolute_error:
0.4072
Epoch 7/10
10/10 [==============================] - 0s 3ms/step - loss: 0.2056 -
mean_absolute_error: 0.3810 - val_loss: 0.2251 - val_mean_absolute_error:
0.4072
Epoch 8/10
10/10 [==============================] - 0s 3ms/step - loss: 0.1999 -
mean_absolute_error: 0.3757 - val_loss: 0.2174 - val_mean_absolute_error:
0.4004
Epoch 9/10
10/10 [==============================] - 0s 3ms/step - loss: 0.1950 -
mean_absolute_error: 0.3710 - val_loss: 0.2134 - val_mean_absolute_error:
0.3971
Epoch 10/10
10/10 [==============================] - 0s 3ms/step - loss: 0.1909 -
mean_absolute_error: 0.3661 - val_loss: 0.2093 - val_mean_absolute_error:
0.3935
```



У висновку показується результат виконання кожної епохи. Виводиться номер епохи, кількість пакетів, витрачена на час і помилки. Loss – це розрахована функція втрат, mean_absolute_error – це метрика, вказана при компіляції. Якщо вказати додаткові метрики, то вони теж будуть у висновку. Усі помилки з префіксом «val_» - це те саме для валідаційного набору даних.

У цьому виді модель можна навчити, але набагато ефективніше це можна зробити, якщо використовувати функціонал зворотних виходів. З їхньою допомогою можна здійснити ранню зупинку навчання для боротьби з перенавчанням, візуалізувати дані та багато іншого. Ось приклад деяких із них:



```
# Якщо помилка не зменшується на протязі вказаної кількості епох, то процес навчання переривається і модель ініціалізується вагами з найнижчим показником параметра "monitor"
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', # вказується параметр, по якому виконується рання зупинка. Зазвичай, це функція втрат на валідаційному наборі (val_loss)
    patience=2, # кількість епох по по завершенню яких скінчиться навчання, якщо показники не покращаться
    mode='min', # вказує, в яку сторону повинн бути покращена похибка
    restore_best_weights=True    # якщо параметр виставлено в true, то по завершенню навчання модель буде ініціалізована вагами з найнижчим показником параметра "monitor"
```

```
)

# Зберігає модель для подальшого заванаження
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
    filepath='my_model', # шлях до папки, де буде збережено модель
    monitor='val_loss',
    save_best_only=True, # якщо параметр виставлено в true, то зберігається тільки краща модель
    mode='min'
)

# Зберігає логі виконання навчання, які можна буде подивитись в спеціальному середовищі TensorBoard
tensorboard = tf.keras.callbacks.TensorBoard(
    log_dir='log', # шлях до папки де будуть зберігатись логі.
)
```

Навчимо цю ж модель, але вже використовуючи зворотні виклики.

```
model.fit(
    X,
    Y,
    validation_split=0.2,
    epochs=50,
    batch_size = 8,
    callbacks = [
        early_stopping,
        model_checkpoint,
        tensorboard
    ]
)
```

```
Epoch 1/50
10/10 [==============================] - 0s 10ms/step - loss: 0.1864 - mean_absolute_error: 0.3616 - val_loss: 0.2080 - val_mean_absolute_error: 0.3932
INFO:tensorflow:Assets written to: my_model/assets
Epoch 2/50
10/10 [==============================] - 0s 4ms/step - loss: 0.1831 - mean_absolute_error: 0.3582 - val_loss: 0.2025 - val_mean_absolute_error: 0.3879
INFO:tensorflow:Assets written to: my_model/assets
```

```
Epoch 48/50
10/10 [==============================] - 0s 4ms/step
- loss: 0.1002 - mean_absolute_error: 0.2649 -
val_loss: 0.1149 - val_mean_absolute_error: 0.3030
INFO:tensorflow:Assets written to: my_model/assets
Epoch 49/50
10/10 [==============================] - 0s 4ms/step
- loss: 0.0997 - mean_absolute_error: 0.2641 -
val_loss: 0.1145 - val_mean_absolute_error: 0.3023
INFO:tensorflow:Assets written to: my_model/assets
Epoch 50/50
10/10 [==============================] - 0s 6ms/step
- loss: 0.0987 - mean_absolute_error: 0.2633 -
val_loss: 0.1130 - val_mean_absolute_error: 0.3007
INFO:tensorflow:Assets written to: my_model/assets
```

Як бачимо, процес навчання зупинився раніше, оскільки модель не поліпшувалася. В результаті є і збережена модель, її можна відновити за допомогою **`tf.saved_model.load()`**. Якщо не використовувати **`ModelCheckpoint`**, модель можна зберегти методом **`save()`**.

```
model.save("my_model")
model_restore = tf.saved_model.load("my_model")
INFO:tensorflow:Assets written to: my_model/assets
```

Можна подивитись і як модель навчалася. Для цього необхідно викликати TensorBoard.

```
%load_ext tensorboard
%tensorboard --logdir "log"

The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 287), started 0:01:15 ago. (Use '!kill 287' to kill it.)
```

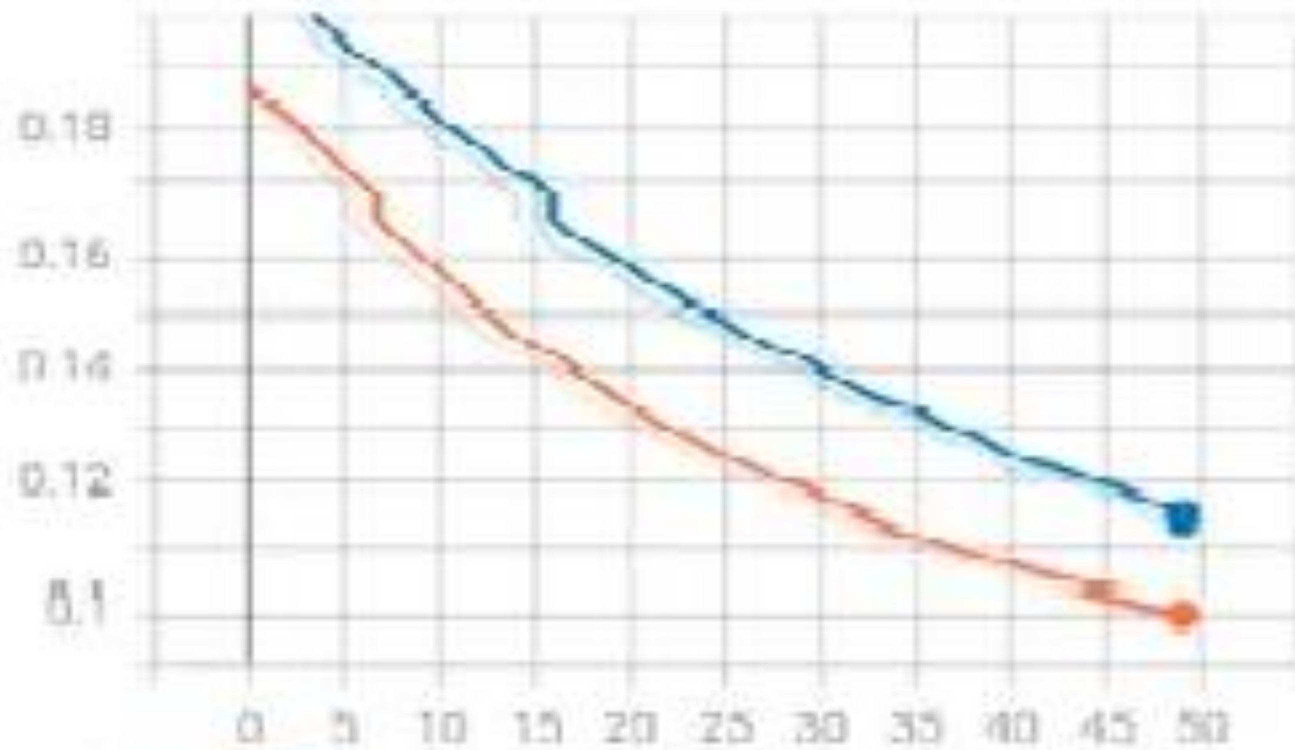


Рис.3 Динаміка навчання моделі

Оцінка моделі

Останнє, що нам залишилося, - оцінити навчену модель. І тому є метод **evaluate()**. Наприклад створимо тестові дані:

```
X_test = np.array(np.random.random((10, 5)))
Y_test = np.array(np.random.random((10)))

res = model.evaluate(X_test, Y_test)
print("loss and mean_absolute_error", res)

1/1 [==============================] - 0s 21ms/step - loss: 0.1259 - mean_absolute_error: 0.3187
loss and mean_absolute_error [0.1258658617734909, 0.3187190890312195]
```

Для отримання передбачуваних значень використовуйте **`predict()`**.

```
predictions = model.predict(X_test)
print(predictions)
```

Вивід:

```
[[0.2159217 ]
[0.2733177 ]
[0.43817037]
[0.7428887 ]
[0.2788944 ]
[0.41930375]
[0.39496648]
[0.30358872]
[0.28828645]
[0.3757842 ]]
```