МІНІСТЕРСТВООСВІТИІ НАУКИУКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут Інформаційних технологій
Кафедра штучного інтелекту

Чичкарьов Є.А.

Методичні рекомендації щодо проведення практичних занять з дисципліни «Сучасні технології програмування в системах зі штучним інтелектом» для студентів спеціальності 122 Комп'ютерні науки всіх форм навчання

Практична робота № 3. Багатокласова класифікація з використанням бібліотеки глибокого навчання Keras

Розглянуто на засіданні кафедри ШІ
Протокол № 1 від «29» серпня 2024 р

Київ
2024

УДК 004.658

Чичкарьов Є.А. Методичні рекомендації щодо проведення практичних занять з дисципліни «Сучасні технології програмування в системах зі штучним інтелектом» для студентів спеціальності 122 Комп’ютерні науки всіх форм навчання. Практична робота № 3. Багатокласова класифікація з використанням бібліотеки глибокого навчання Keras. - Київ: ДУТ, 2024. 12 с.

Методичні рекомендації призначені для ознайомлення студентів спеціальності 122 Комп’ютерні науки всіх форм навчання з різними аспектами створення додатків для вирішення задач штучного інтелекту та набуття ними практичного досвіду створення додатків, які використовують технології глибокого навчання і пакету TensorFlow, при виконанні практичних занять з дисципліни «Сучасні технології програмування в системах зі штучним інтелектом».

Рецензенти:

Рекомендовано
на засіданні кафедри штучного інтелекту,
протокол № 1 від 29 серпня 2024 р.

ЗМІСТ

Практична робота №3. Багатокласова класифікація з використанням бібліотеки глибокого навчання Keras

Мета: *ознайомитися з поняттям багатошарового персептрона, побудовою багатокласового класифікатора з використанням пакету Keras, його можливостями для побудови і навчання моделей TensorFlow, оцінкою оцінки функції втрат для багатокласової класифікації.*

1 Теоретичний розділ

1.1 Поняття багатошарового персептрону

Багатошаровий персептрон — це тип мережі, де кілька шарів групи персептронів складаються разом, щоб створити модель.

Багатошарове сприйняття також відоме як MLP. Це повністю пов'язані щільні шари, які перетворюють будь-який вхідний розмір у потрібний розмір. Багатошарове сприйняття — це нейронна мережа, яка має кілька рівнів. Щоб створити нейронну мережу, ми об'єднуємо нейрони так, щоб виходи одних нейронів були входами інших нейронів.

Багатошаровий персептрон має один вхідний рівень, і для кожного входу є один нейрон (або вузол), він має один вихідний рівень з одним вузлом для кожного виходу, і він може мати будь-яку кількість прихованих шарів, і кожен прихований шар може мати будь-яка кількість вузлів. Схематична діаграма багатошарового персептрона (MLP) зображена нижче (рис. 1).

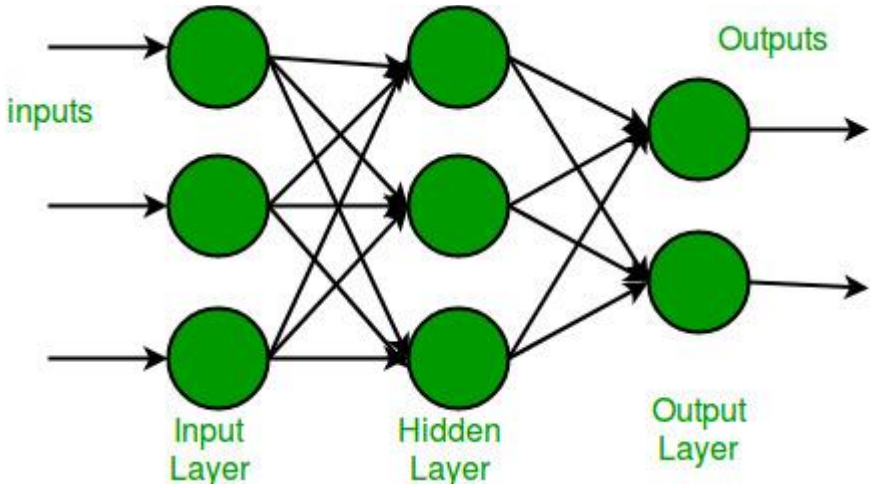

Рис. 1. Багатошаровий персептрон з оним прихованим шаром

Перш ніж перейти до концепції рівня та кількох персептронів, давайте почнемо з будівельного блоку цієї мережі, яким є персептрон. Подумайте про персептрон/нейрон як про лінійну модель, яка приймає кілька вхідних даних і створює вихід (рис. 2). У нашому випадку персептрон — це лінійна модель, яка

приймає купу вхідних даних, множить їх на вагові коефіцієнти та додає зсув, щоб створити вихід.

$$Z = \vec{w} \cdot X + b$$

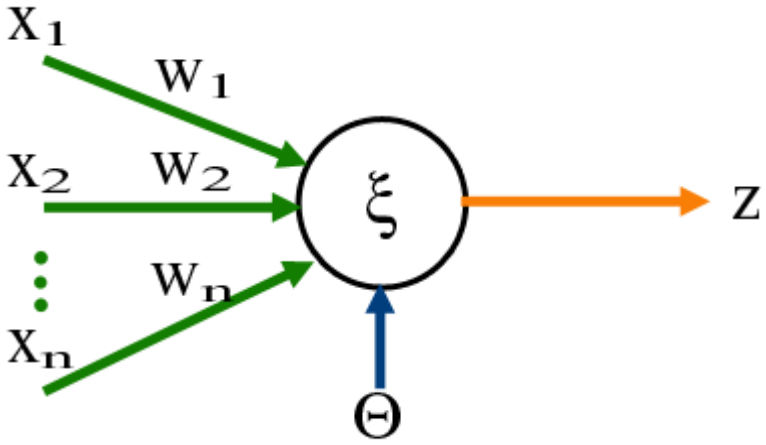

Рис. 2. Зображення персептрона

Тепер, якщо ми складемо купу цих перцептронів разом, це стане прихованим шаром, який також відомий як щільний шар у сучасній термінології глибокого навчання (рис. 3).

Щільний шар:

$$f(X)=W \cdot X + \vec{b}$$

Зауважте, що член зміщення тепер є вектором, а W – ваговою матрицею

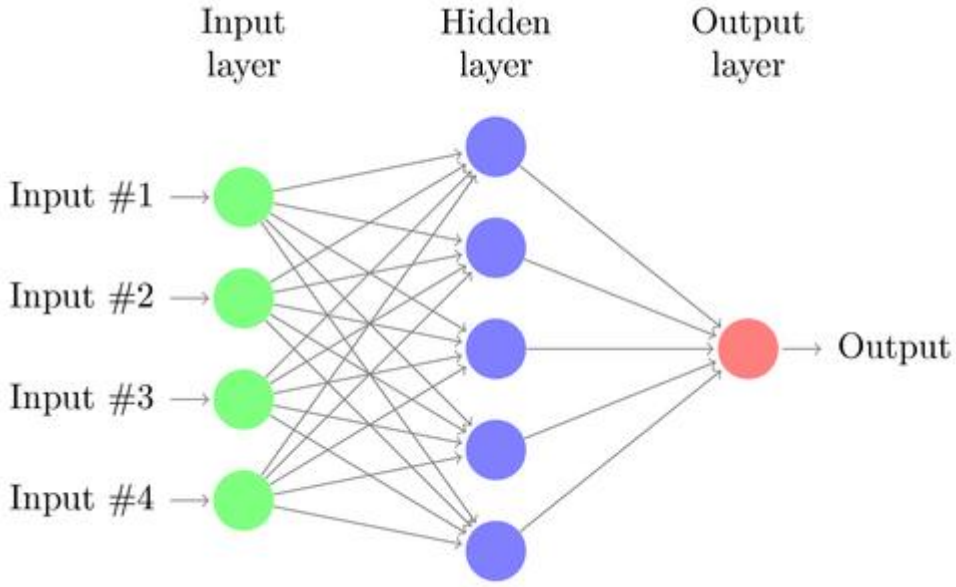

Рис. 3. Одношарова мережа персептронів

Тепер ми розуміємо щільний шар, давайте додамо їх декілька, і ця мережа стане багатошаровою мережею персептронів (рис.4).

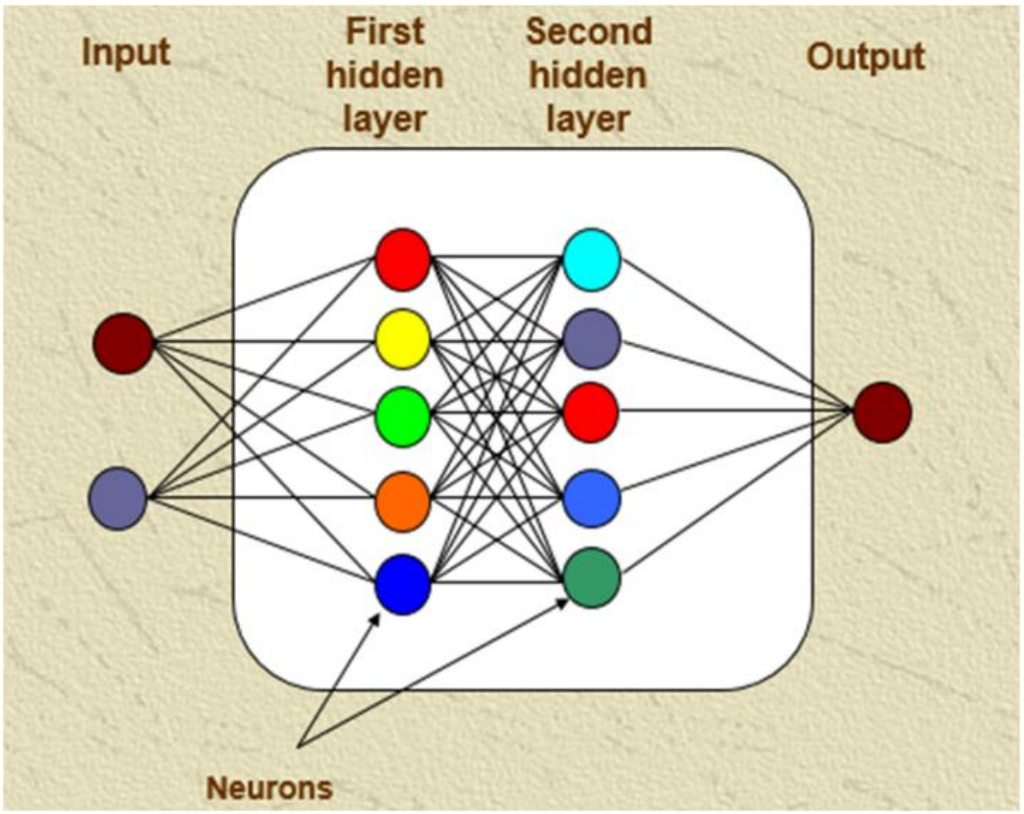

Рис. 3. Багатошарова персептронна мережа

Якщо ви помітили наш щільний шар, використовуйте лише лінійні функції, а будь-яка комбінація лінійних функцій призводить лише до лінійного результату. Оскільки ми хочемо, щоб наш MLP був гнучким і вивчав нелінійні межі рішень, нам також потрібно ввести нелінійність у мережу. Ми вирішуємо завдання введення нелінійності шляхом додавання функції активації. Існують різні типи функцій активації, які можна використовувати, але ми будемо впроваджувати Rectified Linear Units (ReLu), яка є однією з популярних функцій активації. Функція ReLU — це проста функція, яка дорівнює нулю для будь-якого вхідного значення, нижчого від нуля, і має таке саме значення для значень, більших за нуль.

Функція ReLU: $f(x) = max(0, X)$

Тепер ми розуміємо щільний шар, а також розуміємо призначення функції активації, залишилося лише навчити мережу.

Для навчання нейронної мережі нам потрібна функція втрат, і кожен рівень повинен мати **цикл прямого зв'язку** та **цикл зворотного поширення**.

Цикл прямого зв'язку отримує вхідні дані та генерує вихідні дані для створення прогнозу, а цикл зворотного поширення допомагає навчити модель, регулюючи вагові коефіцієнти в шарі, щоб зменшити вихідні втрати.

У зворотному поширенні оновлення ваги виконується за допомогою градієнтів зворотного поширення за допомогою правила ланцюга та оптимізується за допомогою алгоритму оптимізації.

Отже, щоб узагальнити нейронну мережу, потрібно кілька будівельних блоків

- **Щільний шар** — повнозв'язаний шар, $f(X) = W \cdot X + b$
- **Рівень ReLU** (або будь-яка інша функція активації для введення нелінійності)
- **Функція втрат** — (кросентропія у випадку багатокласової проблеми класифікації, див. нижче)
- **Алгоритм Backprop** — стохастичний градієнтний спуск із зворотним поширенням градієнтів

1.2 Машинне навчання і функції втрат

Нейронні мережі глибокого навчання навчаються за допомогою алгоритму оптимізації стохастичного градієнтного спуску.

В рамках алгоритму оптимізації похибка для поточного стану моделі повинна бути оцінена неодноразово. Це вимагає вибору функції похибок, яку умовно називають функцією втрат , яку можна використовувати для оцінки втрат моделі, щоб ваги можна було оновити для зменшення втрат під час наступної оцінки.

Моделі нейронних мереж вивчають відображення входів і виходів із прикладів, а вибір функції втрат має відповідати структурі конкретної проблеми прогнозного моделювання, такої як класифікація чи регресія. Крім того, конфігурація вихідного рівня також повинна відповідати обраній функції втрат. Треба знати, до якого класу віднести проблему, щоб мати можливість правильно оцінити обрану функцію втрат. Наприклад, якщо є проблема регресії, ми повинні знати, що зазвичай використовується функція mean_square_error. Звичайно, ці функції також мають різні параметри.

1.3 Стандартні функції втрат в Keras

1.3.1 Бінарна класифікація (Binary Classification)

Функція втрат бінарної класифікації використовується під час розв'язання задачі, що включає лише два класи. Наприклад, коли передбачається шахрайство в транзакціях з кредитними картками, транзакція або є шахрайською, або ні.

Іноді використовують бінарна перехресну ентропію (Binary Cross Entropy). Бінарна перехресна ентропія обчислює втрату перехресної ентропії між прогнозованими та справжніми класами. За замовчуванням використовується зменшення sum_over_batch_size. Це означає, що втрата поверне середнє значення втрат на зразок у вибірці (batch).

```
      y_true = [[0., 1.], [0.2, 0.8],[0.3, 0.7],[0.4, 0.6]]
      y_pred = [[0.6, 0.4], [0.4, 0.6],[0.6, 0.4],[0.8, 0.2]]
      bce =
tf.keras.losses.BinaryCrossentropy(reduction='sum_over_batch_size')
      bce(y_true, y_pred).numpy()
```

Зменшення суми означає, що функція втрат поверне суму втрат на вибірку в batch (партії).

```
bce = tf.keras.losses.BinaryCrossentropy(reduction='sum')
bce(y_true, y_pred).numpy()
```

Використання зменшення як none повертає повний масив втрат на вибірку.

```
bce = tf.keras.losses.BinaryCrossentropy(reduction='none')
bce(y_true, y_pred).numpy()
array([0.9162905 , 0.5919184 , 0.79465103, 1.0549198 ], dtype=float32)
```

У двійковій класифікації використовується функція активації сигмоїдної функції активації. Він обмежує вихід числом від 0 до 1.

1.3.2 Багатокласова класифікація (Multiclass classification)

У задачах, що включають передбачення більш ніж одного класу, використовуються різні функції втрат. У цьому розділі ми розглянемо пару:

Категоріальна кросентропія (Categorical Crossentropy)

CategoricalCrossentropy також обчислює перехресну втрату ентропії між справжніми та прогнозованими класами. Мітки надаються у форматі one_hot.

```
cce = tf.keras.losses.CategoricalCrossentropy()
cce(y_true, y_pred).numpy()
```

Розріджена категорична кросентропія (Sparse Categorical Crossentropy)

Якщо у вас є два або більше класів і мітки є цілими числами, слід використовувати SparseCategoricalCrossentropy.

```
y_true = [0, 1,2]
y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1],[0.1, 0.8, 0.1]]
scce = tf.keras.losses.SparseCategoricalCrossentropy()
scce(y_true, y_pred).numpy()
```

1.4 Набір даних для виконання класифікації

В цей роботі розглянемо задачу багатокласової класифікації.

Приклад класифікації розглянемо стосовно набору даних про квіти іричів. Цей набір даних добре вивчений і є хорошою проблемою для практики на нейронних мережах, оскільки всі чотири вхідні змінні є числовими й мають однаковий масштаб у сантиметрах. Кожен екземпляр описує властивості спостережуваних вимірювань квітки, а вихідною змінною є певний вид ірису.

Це проблема багатокласової класифікації, тобто існує більше двох класів, які потрібно передбачити. Насправді існує три види квітки. Це важлива проблема для практики з нейронними мережами, оскільки значення трьох класів вимагають трохи інший підхід у порівнянні до бінарної класифікації.

Набір даних квітки ірису є добре вивченою проблемою, і тому ви можете очікувати досягнення точності моделі в діапазоні від 95% до 97%. Це є хорошою ціллю для розробки ваших моделей.

Цей набір даних квітів ірису можна завантажити з репозиторію UCI Machine Learning і розмістити його в поточному робочому каталозі з іменем файлу «iris.csv».

Для роботи необхідно імпортувати потрібні модулі за зразком:

```
import pandas
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
```

Далі завантажуємо набір даних. Найпростіше завантажити дані у вигляді csv-файлу за допомогою pandas. Потім ви можете розділити атрибути (стовпці) на вхідні змінні (X) і вихідні змінні (Y). Приклад:

```
# load dataset
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
```

1.5 Побудова і навчання моделі

При моделюванні проблем багатокласової класифікації за допомогою нейронних мереж корисно змінити форму вихідного атрибута з вектора, який містить значення для кожного значення класу, на матрицю з логічним значенням для кожного значення класу та чи має даний екземпляр це значення класу або ні.

Це називається одноразовим кодуванням або створенням фіктивних змінних із категоріальної змінної. Наприклад, у цій задачі три значення класу: Iris-setosa, Iris-versicolor та Iris-virginica. Таким чином, вихідна змінна містить три варіанти рядкових значень.

Перетворення вмісту рядкової змінної Y до цілих чисел виконуємо за допомогою класу scikit-learn LabelEncoder. Потім вектор з цілими числами

перетворюємо в масив зі стовпчиками, які містять 0 або 1, за допомогою функції Keras to_categorical().

```
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = to_categorical(encoded_Y)
```

Після перекодування змінна Y містить масив наступного виду (одиниця в першому стовпчику відповідає Iris-setosa, одиниця в другому стовпчику відповідає Iris-versicolor, одиниця в третьому стовпчику відповідає Iris-virginica)

1,	0,	0
0,	1,	0
0,	0,	1

Далі створимо функцію, яка повертає сформовану модель (див. приклад нижче). Вона створить базову нейронну мережу для проблеми класифікації ірисів. Ця функція створює просту, повнозв'язну нейронну мережу з одним прихованим шаром, який (в наведеному зразку коду) містить вісім нейронів.

Вихідний рівень повинен містити три кінцевих значення, по одному для кожного класу. Вихідне значення з найбільшою оцінкою імовірності буде прийнято як клас, передбачений моделлю. Топологію мережі цієї простої однорівневої нейронної мережі можна підсумувати наступним чином:

```
4 inputs -> [8 hidden nodes] -> 3 outputs
```

Прихований рівень використовує функцію активації ReLu, що є звичайною практикою. Зауважимо, що для багатокласової класифікації на вихідному рівні використана функція активації «softmax». Це гарантує, що вихідні значення знаходяться в діапазоні від 0 до 1 і можуть використовуватися як прогнозовані ймовірності.

Нарешті, мережа використовує ефективний алгоритм оптимізації градієнтного спуску Адама з логарифмічною функцією втрат, яка в Keras називається «categorical_crossentropy».

```
# define baseline_model
def baseline_model():
   # create model
   model = Sequential()
```

```
    model.add(Dense(8, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model
```

Навчання моделі виконується за допомогою метода fit(), а оцінка якості моделі – за допомогою методу evaluate(). Приклад:

```
model = baseline_model()
# Adam optimizer with learning rate of 0.001
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

print('Neural Network Model Summary: ')
print(model.summary())

# Train the model
model.fit(train_x, train_y, verbose=2, batch_size=5, epochs=200)

# Test on unseen data

results = model.evaluate(test_x, test_y)

print('Final test set loss: {:4f}'.format(results[0]))
print('Final test set accuracy: {:4f}'.format(results[1]))
```

2 Завдання

1. Дослідити вплив на результати навчання кількість нейронів в прихованому шарі (обрати 16, 32, 34 нейрона замість 8).
2. Додати 1, 2, 4 приховані шари, спробувати змінити кількість нейронів в цих шарах.
3. Перетворити моделі з прихованими шарами у форму функціонального API.
4. Створити власний набір даних для бінарної класифікації, перевірити розроблені моделі на новому датасеті.

3 Контрольні питання

1. Що таке Keras і нащо він потрібен при роботі з Tensotflow?
2. Які шари зазвичай використовують для побудови моделей в Keras?
3. Які API побудови моделей використовуються в TensorFlow?

4. Для яких цілей використовується шар Dense?
5. Для яких цілей використовується шар Flatten?
6. Для яких цілей використовується шар Dropout?
7. Як підготувати дані для навчання моделі?
8. Як навчити модель в Tensorflow/Keras?
9. Як перевірити навчену модель?
10. Як виконати прогноз за навченою моделлю?