Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

Keras a Tensorflow
Tvorba neuronové sítě

Semestrální práce předmětu Umělá inteligence a její aplikace

Autor: Jakub David

Abstrakt

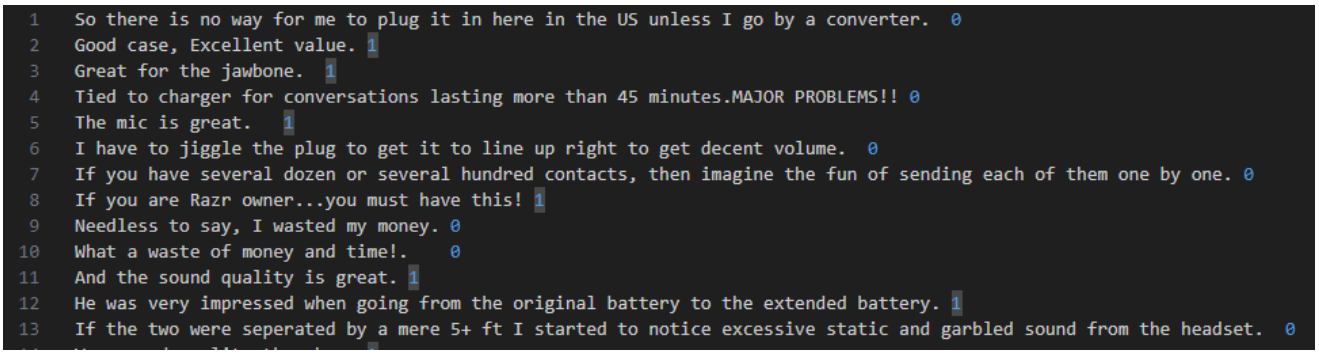
Tato práce se věnuje studiu a vývoji neuronových sítí pro klasifikaci textových dat. Jejím cílem je zkoumat, jak efektivně a účinně tyto sítě dokážou zpracovat a rozdělit data do kategorií. Zaměřuje se zejména na analýzu sentimentu v textech, což znamená určení, jestli je obsah textu pozitivní či negativní. Výzkum využíval konkrétní soubor dat obsahující recenze zákazníků Amazonu. Proces psaní a úpravy kódu probíhal v prostředí Jupiter notebook za použití Google Colab a VSCode, s využitím dovedností získaných na cvičeních a rozšířením již existujícího kódu.

Zdrojové kódy a kompletní práce je k dispozici zde: KubaDavid/VSE-AI-Keras-Tensorflow: VSE-AI (github.com)

Obsah

Úvod

V dnešní době narůstá potřeba analyzovat rozsáhlé množství textových dat, která jsou generována na internetu, v sociálních sítích, recenzích a dalších zdrojích. Proto se analýza sentimentu textů stává klíčovým úkolem pro mnohé odborníky. V této oblasti se neuronové sítě ukazují jako jeden z efektivních nástrojů pro zpracování a klasifikaci těchto dat. Tyto sítě představují soubor algoritmů, kde každý neuron funguje jako matematická funkce, která přijímá vstupy a podle daného algoritmu je rozděluje do kategorií. Neuronové sítě se skládají z několika vrstev a jsou schopné zpracovávat jak textová, tak obrazová data. V tomto případě se však zaměřujeme specificky na textová data, zejména recenze z Amazonu.



Obrázek 1 - Ukázka dat

Metody

V této studii jsme použili dva typy neuronových sítí: jednovrstvou neuronovou síť a konvoluční neuronovou síť, abychom mohli porovnat jejich výkon. Konvoluční síť pracuje se dvěma vrstvami. Pro analýzu a vývoj byl vybrán programovací jazyk Python spolu s jeho knihovnami, zejména Keras a TensorFlow v prostředí Jupiter notebooku. Nejdříve došlo k předzpracování dat, včetně tokenizace, odstraňování stop slov, normalizace a vytvoření tzv. Bag of Words reprezentace. Poté byla data rozdělena na sady pro trénování, validaci a testování. Tento postup vycházel z předem připraveného kódu používaného na cvičeních.

Předzpracování

Pro trénování a testování modelů byla využita datová sada od Kotzias et al. (2015), konkrétně "Sentiment Labelled Sentences Data Set – amazon_cells_labelled". Nejdříve byla tato data načtena pomocí knihovny pandas, přičemž byla rozdělena na jednotlivé věty a odpovídající štítky, které určují klasifikační kategorie. Poté jsem data rozdělil na trénovací, testovací a validační sady s využitím metody train_test_split z knihovny scikit-learn. Trénovací sada obsahuje 60 % všech dat, zatímco testovací a validační sady obsahují každá 20 % dat.

```
# Reading data

import pandas as pd

dfAmazon = pd.read_csv('amazon_cells_labelled.csv', names=['sentence', 'label'], sep='\t')

sentences = dfAmazon['sentence'].values

labels = dfAmazon['label'].values
labels = labels.astype(int)


# Checking the data format
print(sentences.shape)
print(labels.shape)
print("{}:{}".format(sentences[10], labels[10]))
```

Obrázek 2 - Import a úprava dat

```
# Dividing to train, test and validation datasets

from sklearn.model_selection import train_test_split

sentences_train, sentences_test, labels_train, labels_test = train_test_split(sentences, labels, test_size
sentences_train, sentences_val, labels_train, labels_val = train_test_split(sentences_train, labels_train,

# Data check
print(sentences_train.shape)
print(labels_train.shape)
print("{}:{}".format(sentences_train[9], labels_train[9]))
print(sentences_val.shape)
print(labels_val.shape)
print("{}:{}".format(sentences_val[9], labels_val[9]))
print(sentences_test.shape)
print(labels_test.shape)
print("{}:{}".format(sentences_test[9], labels_test[9]))
```

Obrázek 3 - Dělení na trénovací a testovací data

```python
# Preprocessing into Bag of Words

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(sentences_train)

# Check
print(vectorizer.vocabulary_['and'])
print(vectorizer.vocabulary_['good'])
print(vectorizer.vocabulary_['service'])
print(sentences_train[9])

X_train = vectorizer.transform(sentences_train)
X_val = vectorizer.transform(sentences_val)
X_test = vectorizer.transform(sentences_test)

# Check
print(X_train.shape)
print(X_train[9, :])
print(X_val.shape)
print(X_val[9, :])
print(X_test.shape)
print(X_test[9, :])

y_train = labels_train
y_val = labels_val
y_test = labels_test
```

Obrázek 4 - Preprocessing

Jednovrstvá síť

První metodou použitou pro klasifikaci textu byla jednovrstvá neuronová síť. Tato síť se skládá pouze z jedné plně propojené vrstvy. Stejně jako u konvoluční sítě byla i zde provedena příprava vstupních dat pomocí techniky CountVectorizer. Poté byla vytvořena jednovrstvá síť s jednou plně propojenou vrstvou a byla v ní použita sigmoidní aktivační funkce. Stejná ztrátová funkce a optimalizační algoritmus, které byly použity u konvoluční sítě, byly aplikovány i zde. Metrikou použitou pro hodnocení výkonnosti sítě byla opět přesnost.

```
# Preprocessing into Bag of Words

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(sentences_train)

# Check
print(vectorizer.vocabulary_['and'])
print(vectorizer.vocabulary_['good'])
print(vectorizer.vocabulary_['service'])
print(sentences_train[9])

X_train = vectorizer.transform(sentences_train)
X_val = vectorizer.transform(sentences_val)
X_test = vectorizer.transform(sentences_test)

# Check
print(X_train.shape)
print(X_train[9, :])
print(X_val.shape)
print(X_val[9, :])
print(X_test.shape)
print(X_test[9, :])

y_train = labels_train
y_val = labels_val
y_test = labels_test
```

Obrázek 5 - Nastavení jednovrstvé sítě

Konvoluční síť

V této práci byla pro klasifikaci textu využita konvoluční neuronová síť. Tento typ architektury obsahuje jednu nebo více konvolučních vrstev, které slouží k extrakci klíčových rysů ze vstupních dat. Použitá konvoluční síť obsahovala jednu konvoluční vrstvu a jednu vrstvu pro výstup. Vstupní data byla zpracována do formátu Bag of Words pomocí techniky CountVectorizer z knihovny scikit-learn, což převedlo textové vstupy na binární vektory.

Pro definici konvoluční sítě byla použita knihovna Keras. V konvoluční vrstvě byla implementována aktivační funkce "relu", zatímco pro výstupní vrstvu byla použita sigmoidní aktivační funkce. Zmínka: Při testování s aktivační funkcí "softmax" jsem dosahoval pouze nízké přesnosti okolo 0,0x, což naznačuje pravděpodobnou chybu v implementaci. Pro trénink sítě byla zvolena ztrátová funkce binary_crossentropy a optimalizační algoritmus Adam. Jako metrika byla použita přesnost.

```python
# Convolution network
import tensorflow as tf

from keras.models import Sequential
from keras import layers

# Number of attributes
input_dim = X_train.shape[1]
print(input_dim)

# Convolution network
convolutionModel = Sequential()
convolutionModel.add(layers.Dense(10, input_dim=input_dim, activation='relu'))
convolutionModel.add(layers.Dense(1, activation='sigmoid'))
convolutionModel.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
convolutionModel.summary()

# Single layer network
singleModel = Sequential()
singleModel.add(layers.Dense(1, input_dim=input_dim, activation='sigmoid'))
singleModel.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
singleModel.summary()
```

Obrázek 6 - Nastavení konvoluční sítě

Trénování a učení

Oba modely, konvoluční a jednovrstvá neuronová síť, byly trénovány na trénovací souboru dat s využitím metody fit z knihovny Keras. Pro každý z modelů byl stanoven specifický počet epoch a velikost dávek (batch size), což bylo určeno na základě provedených experimentů a validace na validační množině dat. Hlavním cílem tréninku bylo minimalizovat ztrátovou funkci a maximalizovat přesnost klasifikace. Nakonec byly pro oba modely zvoleny stejné parametry - počet epoch 10 a velikost dávek 10 - aby bylo možné lépe porovnat a ilustrovat rozdíly ve výsledcích obou metod.

Vyhodnocení

Po dokončení učení byly oba modely, konvoluční i jednovrstvá neuronová síť, vyhodnoceny na testovací množině dat. K tomuto účelu byla použita metoda evaluate z knihovny Keras. Jako hlavní metriky pro hodnocení kvality obou modelů byly zvoleny přesnost (accuracy) a ztrátová funkce (loss). Tyto metriky poskytují důležitý přehled o výkonu modelů v praktických podmínkách.

```python
# Learning

convolutionHistory = convolutionModel.fit(X_train, y_train, epochs=10, batch_size=10, validation_data=(X_val, y_val), verbose=True)

# Evaluation of Convolution on training data
convolutionTrainLoss, convolutionTrainAccuracy = convolutionModel.evaluate(X_train, y_train, verbose=False)
print("Accuracy of convolution model on training data: {:.4f}".format(convolutionTrainAccuracy))

# Evaluation of Convolution on validation data
convolutionValLoss, convolutionValAccuracy = convolutionModel.evaluate(X_val, y_val, verbose=False)
print("Correctness of the convolutional model on validation data: {:.4f}".format(convolutionValAccuracy))
print("Convolutional model loss function on validation data: {:.4f}".format(convolutionValLoss))

# Single layer network trainging
singleHistory = singleModel.fit(X_train, y_train, epochs=10, batch_size=10, validation_data=(X_val, y_val), verbose=True)

# Evaluation of the single-layer model on the training set
singleTrainLoss, singleTrainAccuracy = singleModel.evaluate(X_train, y_train, verbose=False)
print("Correctness of the single-layer model on training data: {:.4f}".format(singleTrainAccuracy))

# Evaluation of a single-layer model on a validation set
singleValLoss, singleValAccuracy = singleModel.evaluate(X_val, y_val, verbose=False)
print("Correctness of the single-layer model on validation data: {:.4f}".format(singleValAccuracy))
print("Loss function of a single-layer model on validation data: {:.4f}".format(singleValLoss))

# Evaluation of the convolutional model on the test set
convolutionTestLoss, convolutionTestAccuracy = convolutionModel.evaluate(X_test, y_test, verbose=False)
print("Correctness of the convolution model on test data: {:.4f}".format(convolutionTestAccuracy))
print("Loss function of the convolution model on test data: {:.4f}".format(convolutionTestLoss))

# Evaluation of the single-layer model on the test set
singleTestLoss, singleTestAccuracy = singleModel.evaluate(X_test, y_test, verbose=False)
print("Správnost jednovrstvého modelu na testovacích datech: {:.4f}".format(singleTrainAccuracy))
print("Ztrátová funkce jednovrstvého modelu na testovacích datech: {:.4f}".format(singleTestLoss))
```

Obrázek 7 - Učení

Výsledky

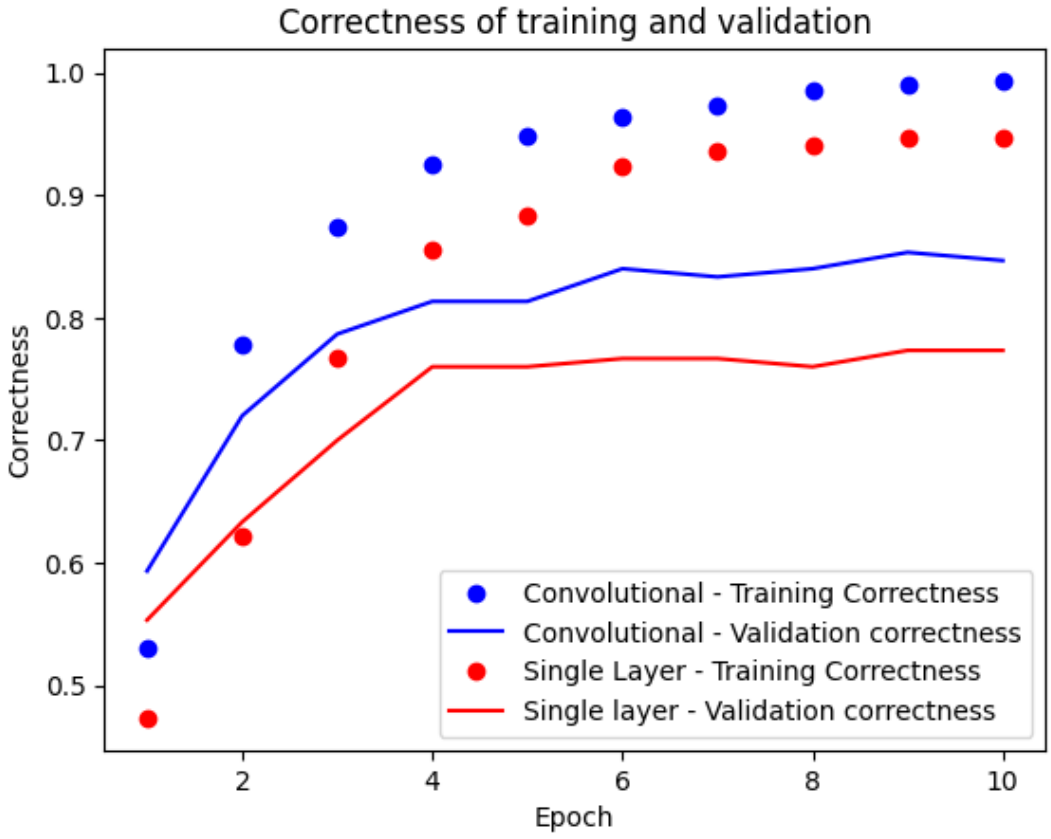
Na základě provedených experimentů byly trénovány oba modely, jak konvoluční, tak jednovrstvý, na trénovací množině dat. Po tréninku byly modely následně vyhodnoceny na validačních a testovacích množinách dat. Výsledky tohoto vyhodnocení byly shrnuty v následující tabulce, která poskytuje přehled o výkonech obou modelů v různých scénářích testování.

Model	Přesnost na TRD	Přesnost na VD	Přesnost na TD	Ztrátová funkce na VD	Ztrátová funkce na TD
Konvoluční	0,997	0,847	0,808	0,420	0,462
Jednovrstvá	0,953	0,773	0,973	0,596	0,533

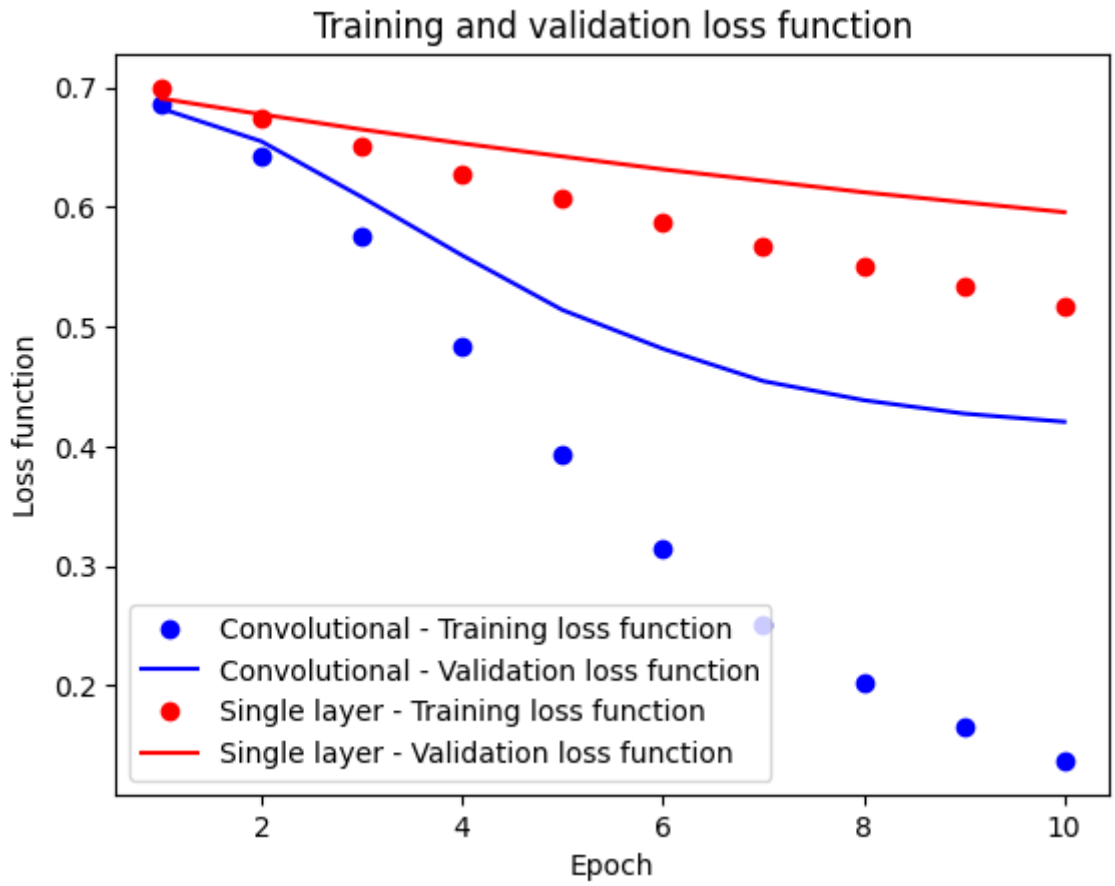
Na základě prezentovaných výsledků je patrné, že konvoluční síť dosáhla lepší přesnosti na validačních datech než jednovrstvý model, s přesností 84.7 % oproti 77.3 %. Na testovacích datech však jednovrstvý model překonal konvoluční síť, dosahující přesnosti 97.3 % proti 80.8 % u konvoluční sítě.

Co se týče ztrátové funkce, konvoluční síť dosáhla hodnoty 0.420 na validačních datech a 0.462 na testovacích datech. Nižší hodnota ztrátové funkce naznačuje lepší přizpůsobení modelu datům a vyšší přesnost předpovědí. Jednovrstvý model dosáhl hodnoty ztrátové funkce 0.596 na validačních datech a 0.533 na testovacích datech, což značí větší chybu v předpovědích a nižší přesnost v porovnání s konvoluční sítí.

Grafické zobrazení výsledků obou modelů ukazuje, že konvoluční síť má lepší výsledky na validačních datech, zatímco jednovrstvá síť exceluje na testovacích datech. Zajímavé je, že přes velký rozdíl v přesnosti na testovacích datech, oba modely dosahují podobného výkonu na validačních datech.



Obrázek 8 - Správnost obou modelů



Obrázek 9 - Ztrátová funkce obou modelů

Na základě dosažených výsledků lze konstatovat, že konvoluční síť obecně dosahuje nižších hodnot ztrátové funkce a vyšší přesnosti v porovnání s jednovrstvým modelem. Toto ukazuje, že konvoluční síť je efektivnějším modelem pro danou úlohu klasifikace dat. Díky své komplexnější struktuře a schopnosti efektivněji rozpoznávat vzory v datech, což byl i původní předpoklad, se konvoluční síť jeví jako vhodnější volba pro tuto specifickou úlohu.

Závěr

V této studii jsme se věnovali analýze a implementaci dvou typů neuronových sítí - jednovrstvé a konvoluční - pro účely klasifikace sentimentu textu. Po implementaci byla provedena série experimentů, jejichž cílem bylo vyhodnotit přesnost obou modelů za použití specifických nastavení, která jsou popsána v předchozím textu. Získané výsledky naznačují, že konvoluční síť s dvěma vrstvami dosahuje lepších výsledků ve srovnání s jednovrstvým modelem, což ukazuje na její větší efektivitu v této konkrétní úloze klasifikace sentimentu.

Zdroje

1. **Umělá inteligence a její aplikace.** ***Materiály z hodin 4IZ232.*** **[Online] [Citace: 28. 12 2023.]**

2. Wikipedia. ***Umělá neuronová síť.*** **[Online] [Citace: 28. 12 2023.] https://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%A1_neuronov%C3%A1_s%C3%AD%C5%A5.**

3. TensorFlow. ***Convolutional Neural Network (CNN).*** **[Online] [Citace: 28. 12 2023.] https://www.tensorflow.org/tutorials/images/cnn.**

4. Datacamp. ***Convolutional Neural Networks (CNN) with TensorFlow Tutorial.*** **[Online] [Citace: 28. 12 2023.] https://www.datacamp.com/tutorial/cnn-tensorflow-python.**