Assignment 3 – Practical Deep Learning Workshop

1)

a. For the preprocess of the search phrase and the item description to sequences of single characters we used:

```
def text_to_char_sequence(text):
    return ' '.join(list(str(text)))
```

And got the following:

	search_term_chars
0	anglebracket
1	lbracket
2	deckover
3	rainshowerhead
4	showeronlyfaucet
...	...
74062	tvriserglass
74063	r20halogenlight
74064	schlagelocksienahalf...
74065	zengardendecor
74066	finesheercurtain63i...

74067 rows × 1 columns

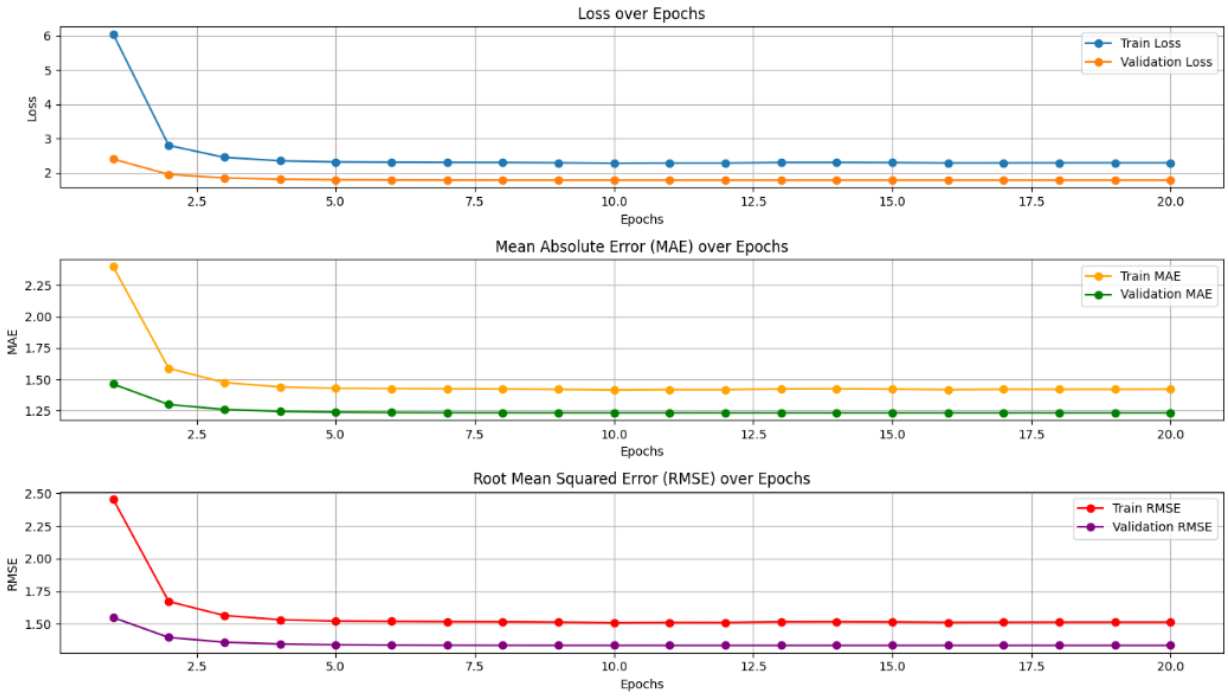
	product_title_chars
0	SimpsonStrong-Tie12-G...
1	SimpsonStrong-Tie12-G...
2	BEHRPremiumTexturedD...
3	DeltaVero1-HandleSho...
4	DeltaVero1-HandleSho...
...	...
74062	AtlanticWindowpane576...
74063	Philips40-WattHalogen...
74064	SchlageCamelotIn-Acti...
74065	Plastec11in.x24in....
74066	LICHTENBERGPoolBlueN...

74067 rows × 1 columns

b. We began with the classic Siamese network from the article.

```
inputs = Input(shape=input_shape)
x = layers.Conv1D(64, 10, activation='relu')(inputs)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(128, 7, activation='relu')(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(128, 4, activation='relu')(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(256, 4, activation='relu')(x)
x = layers.Flatten()(x)
x = layers.Dense(4096, activation='sigmoid')(x)
return Model(inputs, x)
```

And got the following results:



Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Character-level Siamese Network	156 seconds	1.5137	1.3349	1.48	1.4195	1.2319	1.38

As we can see these are very bad results, so we decided to change direction. We believe that CNN is not the best solution here, so we tried some different things.

Now we have built a Siamese network in a bit different way,

Input Shape: Two inputs, input1 and input2(the search input and the title input), both with the same shape.

Shared Layers:

- Embedding: Embeds input sequences.
- LSTM: Processes the sequences and outputs the last hidden state.

Distance Calculation:

- The absolute difference between the two processed inputs is computed using a Lambda layer.

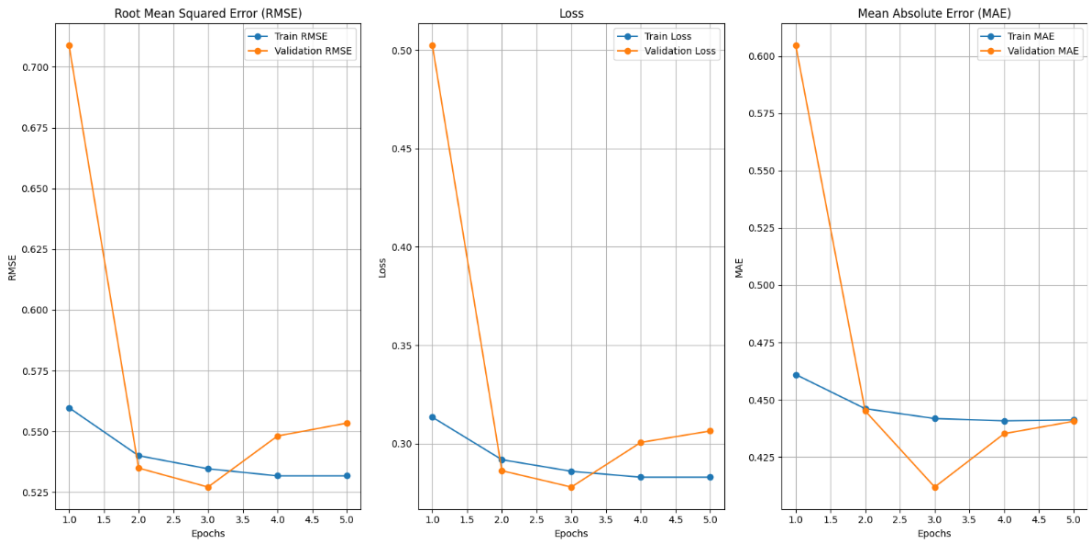
Output Layer:

- A fully connected layer with a linear activation is used to predict the relevance score.

Compilation:

- Loss function: Mean Squared Error (MSE).
- Metrics: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

This architecture took a long time to run on the computer, so we reduced the number of epochs to 5 and obtained the following results:



Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Character-level Siamese LSTM and Embedding	14258.4 seconds = 3.96 hours	0.5318	0.5272	0.613	0.4408	0.4406	0.553

As we can see these are much better results than the first architecture.

c. For the benchmark we used random forest and linear regression and got the following results:

Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Character-level Linear regression	11.83	0.53	0.5288	7.73	0.435	0.433	7.703
Character-level Random Forest	40.51	0.467	0.518	2.07	0.381	0.423	1.46

As we can see, the linear regression was very fast and achieving good results on the validation but did not perform well on the test set.

Pretty like the linear regression, the random forest was relatively quick and achieved good results on the validation but did not perform well on the test set.

So, we decided to try different benchmark now for shared network with the following architecture:

```
inputs = Input(shape=input_shape)
x = Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=embedding_dim)(inputs)
x = Conv1D(128, kernel_size=3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(64, activation='relu')(x)
```

Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Character-level Shared Network	358 seconds	1.4813	1.4811	1.48	1.3815	1.382	1.3805

These are also not very good performances.

For the last try for bench mark we noticed that most of the relevance is 3 or close to that, so we decided to do naïve model that predict always 2.5.

Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Naive	1 second	0.547	0.545	0.548	0.447	0.444	0.446

As we can see, it is much better.

2)

a.

```
train_text = train['search_term'] + ' ' + train['product_title']
test_text = test['search_term'] + ' ' + test['product_title']

# function for cleaning and normalizing text
def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    return text

# Extract complex tokens using regex
def extract_tokens(text):
    # Regex to find "90°", "%", "#SC", and similar patterns
    special_tokens = re.findall(r'[#\w\d°%]+', text)
    return special_tokens
```

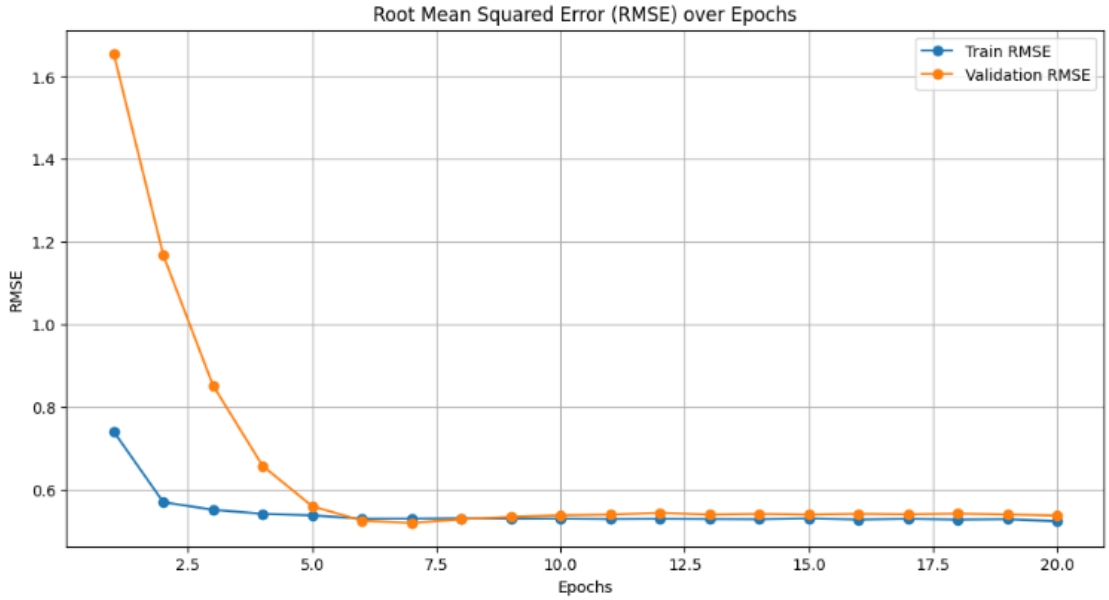
b. We used genism Word2Vec and used all the tokens we got from the train and the test and got Vocabulary size of 15999.

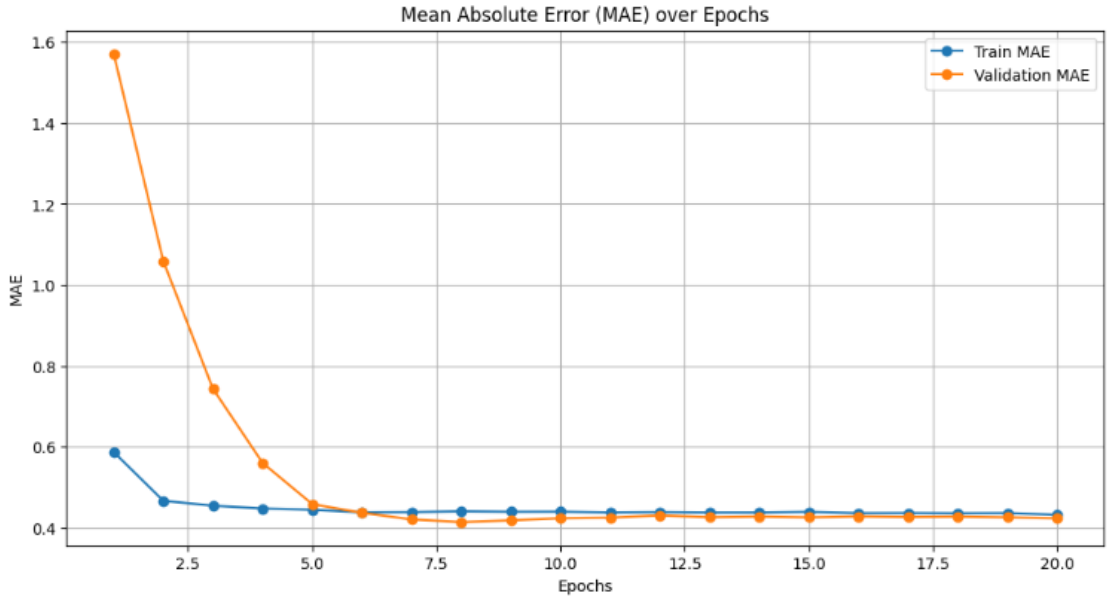
c. In this part we built a new Siamese network in the following way:

Layer (type)	Output Shape	Param #	Connected to
search_input (InputLayer)	(None, 50)	0	-
title_input (InputLayer)	(None, 50)	0	-
word2vec_embedding (Embedding)	(None, 50, 100)	1,600,000	search_input[0][0], title_input[0][0]
shared_lstm (LSTM)	(None, 64)	42,240	word2vec_embedding[0]… word2vec_embedding[1]…
dropout (Dropout)	(None, 64)	0	shared_lstm[0][0]
dropout_1 (Dropout)	(None, 64)	0	shared_lstm[1][0]
lambda_1 (Lambda)	(None, 64)	0	dropout[0][0], dropout_1[0][0]
output_layer (Dense)	(None, 1)	65	lambda_1[0][0]

And got the following results:

Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Word-level Siamese LSTM and Embedding	12611.2 second ~ 3.5 hours	0.524	0.538	0.547	0.432	0.4234	0.445

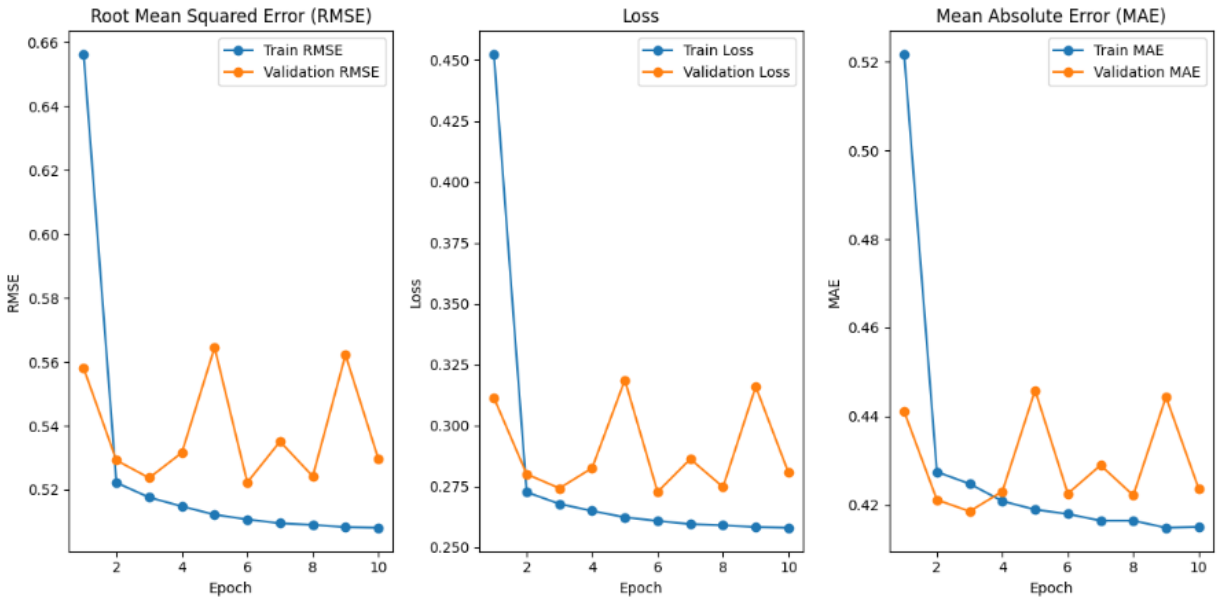




d. In this part we used "bert-base-uncased" and build the model in the following way:

Layer (type)	Output Shape	Param #	Connected to
search_input_ids (InputLayer)	(None, 50)	0	-
title_input_ids (InputLayer)	(None, 50)	0	-
bert_embedding1 (Lambda)	(None, 768)	0	search_input_ids[0][0]
bert_embedding2 (Lambda)	(None, 768)	0	title_input_ids[0][0]
lambda_2 (Lambda)	(None, 768)	0	bert_embedding1[0][0], bert_embedding2[0][0]
dense_2 (Dense)	(None, 64)	49,216	lambda_2[0][0]
dense_3 (Dense)	(None, 32)	2,080	dense_2[0][0]
output_layer (Dense)	(None, 1)	33	dense_3[0][0]

And got the following results:



Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Siamese bert model	4455 seconds ~ 1.23 hours	0.508	0.5298	0.5278	0.415	0.4237	0.436

3)

Model Type	Runtime	Train RMSE	Val RMSE	Test RMSE	Train MAE	Val MAE	Test MAE
Character-level Siamese Network	156 seconds	1.5137	1.3349	1.48	1.4195	1.2319	1.38
Character-level Siamese LSTM and Embedding	14258.4 seconds = 3.96 hours	0.5318	0.5272	0.613	0.4408	0.4406	0.553
Character-level Linear regression	11.83	0.53	0.5288	7.73	0.435	0.433	7.703
Character-level Random Forest	40.51	0.467	0.518	2.07	0.381	0.423	1.46
Character-level Shared Network	358 seconds	1.4813	1.4811	1.48	1.3815	1.382	1.3805
Naive	1 second	0.547	0.545	0.548	0.447	0.444	0.446
Word-level Siamese LSTM and Embedding	12611.2 second ~ 3.5 hours	0.524	0.538	0.547	0.432	0.4234	0.445
Siamese bert model	4455 seconds ~ 1.23 hours	0.508	0.5298	0.5278	0.415	0.4237	0.436
Word-level Linear regression	3.66 seconds	0.238	0.28	0.32	0.397	0.43	0.44
Word-level Random Forest	58.17 seconds	0.27	0.28	0.33	0.428	0.432	0.445

Summary and Analysis of Results

1. Overview of Results

- The models span different architectures, including Siamese Networks, LSTMs, Linear Regression, Random Forest, and BERT, applied at both character and word levels.
- Metrics evaluated include Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) for Train, Validation, and Test sets.
- Runtime varies significantly across models, from 1 second (Naive approach) to over 3.96 hours (Character-level Siamese LSTM and Embedding).

2. Performance Analysis

a. Naive Baseline

- **Runtime**: Fastest (1 second).
- **Performance**: Relatively high errors (Val RMSE = 0.545, Test RMSE = 0.548). This model serves as a simple benchmark but is outperformed by nearly all other models.

b. Linear Regression

- **Character-Level**:
  - **Runtime**: 11.83 seconds (very efficient).
  - **Performance**: Comparable to LSTMs on training data (Train RMSE = 0.53), but significantly worse on Test RMSE (7.73), indicating **overfitting** or potential issues with scaling/normalization.
- **Word-Level**:
  - **Runtime**: Extremely fast (3.66 seconds).
  - **Performance**: Best overall results among models (Test RMSE = 0.32, Test MAE = 0.44), making it an excellent **low-complexity and effective option**.

c. Random Forest

- **Character-Level**:
  - **Runtime**: Moderate (40.51 seconds).
  - **Performance**: Strong generalization (Test RMSE = 2.07, Test MAE = 1.46). However, performance lags compared to word-level models.
- **Word-Level**:
  - **Runtime**: 58.17 seconds.
  - **Performance**: Competitive results (Test RMSE = 0.33, Test MAE = 0.445). Slightly worse than word-level linear regression but better than many deep learning models.

d. Siamese Networks

- **Character-Level**:
    - Simple Siamese Network: Decent performance (Test RMSE = 1.48, Test MAE = 1.38) with reasonable runtime (156 seconds).
    - Siamese LSTM + Embedding: Extremely long runtime (3.96 hours), best character-level performance (Test RMSE = 0.613), but underperforms compared to simpler word-level models.
- **Word-Level**:
    - Siamese LSTM + Embedding: Runtime is still high (~3.5 hours) with competitive performance (Test RMSE = 0.547, Test MAE = 0.445), but it does not significantly outperform other methods.
    - BERT: Relatively not efficient (1.23 hours), delivering competitive results (Test RMSE = 0.5278, Test MAE = 0.436).

e. Shared Network

- **Runtime**: 358 seconds.
- **Performance**: Moderate errors (Test RMSE = 1.48, Test MAE = 1.38), but does not outperform simpler methods.

3. Insights

1. **Word-Level Models Dominate**:
    - Word-level approaches (e.g., Linear Regression, Random Forest) achieve the best performance across all metrics with significantly lower runtime compared to complex deep learning methods.
2. **Deep Learning Trade-offs**:
    - Models like Character-level Siamese LSTM + Embedding and Word-level BERT require substantial runtime, but their performance gains are marginal compared to simpler methods.
    - For projects requiring rapid development and efficiency, deep learning methods may not justify their computational cost.
3. **Linear Regression**:
    - Word-level Linear Regression offers the best balance of runtime (3.66 seconds) and performance (Test RMSE = 0.32), emerging as the **optimal choice for this task**.
4. **Random Forest**:
    - Provides robust results with reasonable runtimes, making it a strong alternative to linear regression when interpretability is less critical.