HYPERPARAMETER OPTIMIZATION

CNN Hyperparameter optimization

Hyperparameter optimization

Both the CNN and RNN notebooks showed improvements over the 50% random baseline, however, this does not represent a particularly high level of performance for this dataset. Can you improve on the performance by adjusting the hyperparameters?

Report which hyperparameters you modified and how these modifications affected the results. What is the highest accuracy you were able to achieve?

You can experiment with these hyperparameters (or any other parameters you find interesting): max_length, embedding_dim, filter_size (CNN), num_filters (CNN), hidden_size (RNN), nonlinearity (RNN), learning_rate, per_device_train_batch_size, max_steps

All done like in the provided notebook, so baseline is with:

```
### CONFIG

config = BasicConfig(
    vocab_size = tokenizer.vocab_size,
    num_labels = len(set(dataset['train']['label'])),
    embedding_dim = 64,
    filter_size = 3,
    num_filters = 10,
)

### ARGS

trainer_args = transformers.TrainingArguments(
    "checkpoints",
    evaluation_strategy="steps",
    logging_strategy="steps",
    load_best_model_at_end=True,
    eval_steps=500,
    logging_steps=500,
    learning_rate=0.001,
    per_device_train_batch_size=8,
    max_steps=2500,
)
```

Evaluate and print out results:

```
[17] eval_results = trainer.evaluate(dataset["test"])
     pprint(eval_results)
     print('Accuracy:', eval_results['eval_accuracy'])
```

```
print("Accuracy:", eval_results["eval_accuracy"])
```

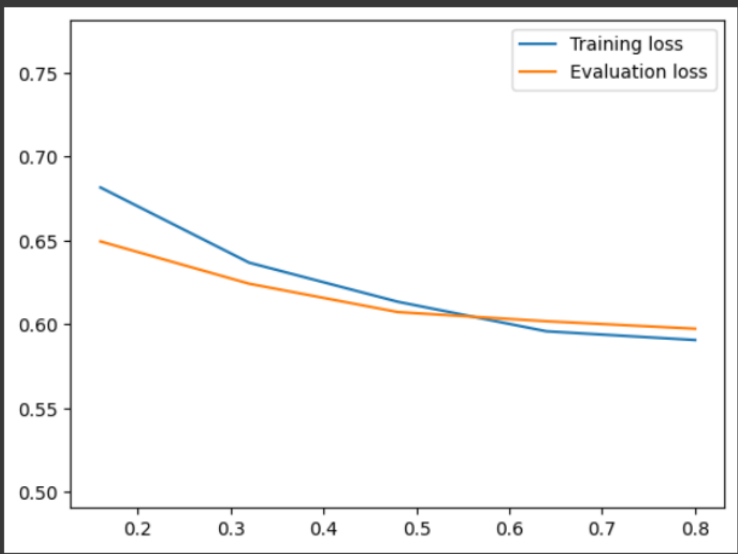
```
[3125/3125 00:08]
{'epoch': 0.8,
 'eval_accuracy': 0.67064,
 'eval_loss': 0.5973329544067383,
 'eval_runtime': 8.8535,
 'eval_samples_per_second': 2823.726,
 'eval_steps_per_second': 352.966}
Accuracy: 0.67064
```

Let's also have a look at training and evaluation loss and evaluation accuracy progression. (The code here is only for visualization and you do not need to understand it, but you should aim to be able to interpret the plots.)

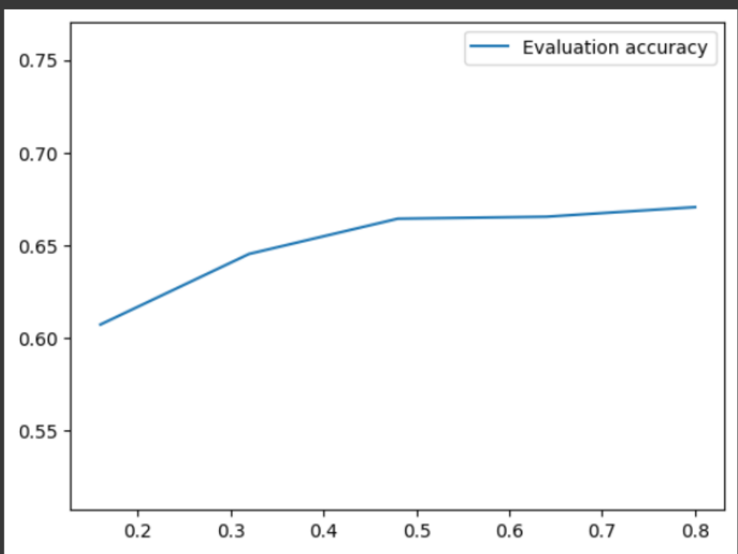
```
[18] %matplotlib inline
     import matplotlib.pyplot as plt

     def plot(logs, keys, labels):
         values = sum([logs[k] for k in keys], [])
         plt.ylim(max(min(values)-0.1, 0.0), min(max(values)+0.1, 1.0))
         for key, label in zip(keys, labels):
             plt.plot(logs["epoch"], logs[key], label=label)
         plt.legend()
         plt.show()

     plot(training_logs.logs, ["loss", "eval_loss"], ["Training loss", "Evaluation loss"])
```



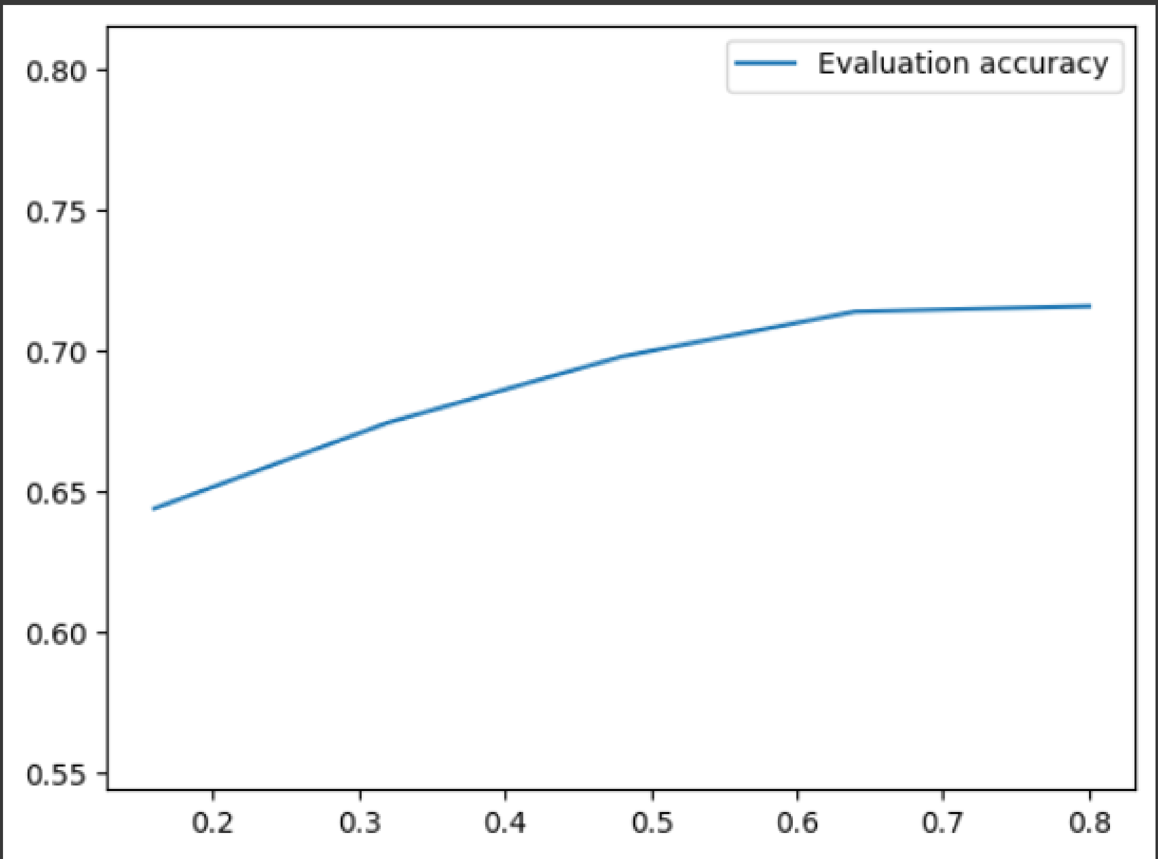

```
plot(training_logs.logs, ["eval_accuracy"], ["Evaluation accuracy"])
```



ARGS

- `embedding_dim` 64 => 128 increased Accuracy: 0.68996
- with keeping `embedding_dim` as 128 and increased `filter_size` 3 ==> 6 Accuracy: 0.69264
- keeping above as is and doubling the `num_filters`10 ==> 20 Accuracy: 0.71584

```
plot(training_logs.logs, ["eval_accuracy"], ["Evaluation accuracy"])
```



After that i run optuna study like presented in intro to HLT exercise 6 with minor adjustments to parameters:

```
import optuna

def objective(trial):
    # Define the search space for hyperparameters
    learning_rate = trial.suggest_float("learning_rate", 1e-6, 1e-3,
log=True)
    batch_size = trial.suggest_categorical("batch_size", [16, 64, 128,
256])


    trainer_args = transformers.TrainingArguments(
        "cnn_checkpoints", #save checkpoints here
        evaluation_strategy="steps",
        logging_strategy="steps",
        eval_steps=500,
        logging_steps=500,
```

```
        learning_rate=learning_rate, #learning rate of the gradient descent
        max_steps=10000,
        load_best_model_at_end=True,
        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=batch_size
    )

    model = SimpleCNN(config)

    trainer = transformers.Trainer(
        model=model,
        args=trainer_args,
        train_dataset=dataset["train"],
        eval_dataset=dataset["test"].select(range(1000)), #make a smaller subset to evaluate on
        compute_metrics=compute_accuracy,
        data_collator=data_collator,
        callbacks=[early_stopping]
    )

    # Train the model and get the best validation loss
    trainer.train()
    eval_results = trainer.evaluate()
    return eval_results["eval_accuracy"] #let's try to maximize accuracy

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=7)
```

```
Best trial (number 0):
  Value: 0.642
  Params: {'learning_rate': 2.291937346839778e-05, 'batch_size': 128}

All trials:
  Trial 0:
    Value: 0.642
    Params: {'learning_rate': 2.291937346839778e-05, 'batch_size': 128}
  Trial 1:
    Value: 0.527
    Params: {'learning_rate': 2.5116312839219326e-06, 'batch_size': 64}
  Trial 2:
    Value: 0.526
    Params: {'learning_rate': 1.0454350693142235e-05, 'batch_size': 16}
  Trial 3:
    Value: 0.594
    Params: {'learning_rate': 1.761113500582794e-05, 'batch_size': 16}
  Trial 4:
    Value: 0.542
    Params: {'learning_rate': 3.121692066775901e-06, 'batch_size': 128}
  Trial 5:
    Value: 0.586
    Params: {'learning_rate': 1.6648533076980423e-05, 'batch_size': 16}
  Trial 6:
    Value: 0.537
    Params: {'learning_rate': 1.6671241807138018e-06, 'batch_size': 256}
```

So with adjusting trainer parameters it actually got worse and the best results were 71.5% from config.

RNN

baseline achieved was:

```
{'epoch': 0.8,
 'eval_accuracy': 0.50812,
 'eval_loss': 0.6931626796722412,
 'eval_runtime': 10.1816,
 'eval_samples_per_second': 2455.4,
 'eval_steps_per_second': 306.925}
Accuracy: 0.50812
```

RNN ARGS

- with change of `nonlinearity` from "tanh" to "relu"

```
    {'epoch': 0.8,
 'eval_accuracy': 0.51412,
 'eval_loss': 0.6923446655273438,
 'eval_runtime': 10.0994,
 'eval_samples_per_second': 2475.395,
```

```
 'eval_steps_per_second': 309.424}
Accuracy: 0.51412
```

- After doubling the `hidden_size` from 96 to 192 i saw virtually nonexistent change

```
{'epoch': 0.8,
 'eval_accuracy': 0.51228,
 'eval_loss': 0.6925438046455383,
 'eval_runtime': 10.0167,
 'eval_samples_per_second': 2495.83,
 'eval_steps_per_second': 311.979}
Accuracy: 0.51228
```

- Stacking the layers `num_layers` from 1 ==> 5 also did nothing

```
{'epoch': 0.8,
 'eval_accuracy': 0.5112,
 'eval_loss': 0.6929822564125061,
 'eval_runtime': 13.9058,
 'eval_samples_per_second': 1797.809,
 'eval_steps_per_second': 224.726}
Accuracy: 0.5112
```

so i left the parameters as they now were and conducted optuna study

```
    import optuna

def objective(trial):
    # Define the search space for hyperparameters
    learning_rate = trial.suggest_float("learning_rate", 1e-6, 1e-3,
log=True)
    batch_size = trial.suggest_categorical("batch_size", [16, 64, 128,
256])


    trainer_args = transformers.TrainingArguments(
        "rnn_checkpoints", #save checkpoints here
        evaluation_strategy="steps",
        logging_strategy="steps",
        eval_steps=500,
        logging_steps=500,
        learning_rate=learning_rate, #learning rate of the gradient descent
        max_steps=10000,
        load_best_model_at_end=True,
        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=batch_size
    )
```

```
    model = SimpleRNN(config)

    trainer = transformers.Trainer(
        model=model,
        args=trainer_args,
        train_dataset=dataset["train"],
        eval_dataset=dataset["test"].select(range(1000)), #make a smaller
subset to evaluate on
        compute_metrics=compute_accuracy,
        data_collator=data_collator,
        callbacks=[early_stopping]
    )

    # Train the model and get the best validation loss
    trainer.train()
    eval_results = trainer.evaluate()
    return eval_results["eval_accuracy"] #let's try to maximize accuracy

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=7)
```

```
Best trial (number 6):
  Value: 0.512
  Params: {'learning_rate': 7.646822188369781e-06, 'batch_size': 64}

All trials:
  Trial 0:
    Value: 0.505
    Params: {'learning_rate': 1.0487080041045669e-06, 'batch_size': 128}
  Trial 1:
    Value: 0.506
    Params: {'learning_rate': 0.0004283601162876318, 'batch_size': 64}
  Trial 2:
    Value: 0.505
    Params: {'learning_rate': 1.5897145858773748e-06, 'batch_size': 16}
  Trial 3:
    Value: 0.51
    Params: {'learning_rate': 5.285245803533564e-06, 'batch_size': 256}
  Trial 4:
    Value: 0.507
    Params: {'learning_rate': 1.9399734875958204e-05, 'batch_size': 16}
  Trial 5:
    Value: 0.511
    Params: {'learning_rate': 2.39018153459405e-06, 'batch_size': 256}
  Trial 6:
    Value: 0.512
    Params: {'learning_rate': 7.646822188369781e-06, 'batch_size': 64}
```

I actually managed not to get any increase in the accuracy