Nowadays, NLP applications have become more and more popular. Applications such as IBM Watson Assistant[1], Salesforce Einstein Chatbot[2] significantly changed the way we work and live. Most of the NLP applications are built from some NLP toolkits which largely decrease the amount of work we need to devote to building NLP layers and optimizers. Two of the most famous NLP toolkits are Pytorch and Keras. Pytorch is developed by Facebook and contains libraries for various usages such as computer vision and natural language processing. Keras provides high-level APIs and is built based on TensorFlow. In this tech review, we will analyze and compare these two NLP toolkits.

Firstly, compared with Pytorch, Keras is more lightweight and more user-friendly. Pytorch includes a more comprehensive machine learning ecosystem and includes low-level APIs[3], so you may expect to interact with array expressions more often than using Keras. Keras includes mostly high-level APIs. These features make Keras a better choice for fast-paced development in the real industry and Pytorch a better choice for academic research because researchers can gain more control of the low-level executions. A specific example can be constructing a simple convolutional network in Keras and in Pytorch[5]:

Keras

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPool2D())
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(MaxPool2D())
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

PyTorch

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.conv2 = nn.Conv2d(32, 16, 3)
        self.fc1 = nn.Linear(16 * 6 * 6, 10)
        self.pool = nn.MaxPool2d(2, 2)
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 6 * 6)
        x = F.log_softmax(self.fc1(x), dim=-1)
        return x
model = Net()
```

Here we can find that Keras requires less effort and less code understanding requirements to construct a layer because we do not need to extend an existing class. Additionally, Keras requires fewer lines of code to construct a model and seems more concise and readable, so it is user-friendly to beginners while PyTorch is a better choice if you want to better explore the inner workings of your model and manage it. Keras also implicitly enables GPU acceleration but PyTorch requires the user to specify when to transfer data between CPU and GPU.

Secondly, from the perspective of debugging, debugging codes in PyTorch will be easier than debugging codes in Keras because Keras add lots of abstractions which increases the complexity of the codes. Although these abstractions make it more convenient for users to call high-level APIs, it exerts extra burdens for users to debug their codes. Since PyTorch gives easy access to its code and does not contain many wrappers around its code, it is easier for debugging. PyTorch also supports some debugging tools to help identify the error, for example, pdb, ipdb because PyTorch is developed in python. It also gives opportunities to create dynamic computational graphs because it is built as a dynamic library which is more convenient than referring to a static dynamic computational graph because it does not require the input sequence length to be fixed[3]. Besides, Since TensorFlow is not built purely on python, the debugging of Keras might also require referring to some low-level codes that are in other languages which also makes it harder to debug.

Thirdly, PyTorch usually has better performance but is not as portable as Keras does[7]. PyTorch has better performance because it does not have too many abstractions and focuses more on the low-level functional codes. Keras will have more overhead for different interfaces and abstractions. Other factors affecting the performance differences can be different implementations and different code complexity. But since PyTorch saves its models in python, it is not as portable as Keras, which can save its models in JSON+H5 files. So sharing models on different platforms by Keras seems easier than it does by PyTorch. But PyTorch also has larger community support due to the larger number of users which helps model implementation and code debugging. And Pytorch also has a better active development process which attaches better potentials to this toolkit.

Another difference I find is that models in PyTorch usually have fewer parameters but have easier access to source codes. That is because PyTorch permits users to reuse its codes from low-level implementations but Keras usually gives more parameters and capsulations to encourage users to use its high-level APIs. One example can be the LSTM class in PyTorch and the LSTM class in Keras[8][9]. PyTorch supports parameters such as input_size, hidden_size, num_layers, bias, batch_first, etc) but does not support parameters such as activation, unroll, stateful, etc. This gives more convenience for beginners to control their model by simply modifying some of the parameters instead of extending the existing class. This seems also true for the text processing part of the two toolkits.

```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
```

For Keras, a general tokenizer is provided with various arguments such as oov_token which provides a replacement for words missing in the corpus.

```
loaded_data = ["now this ain't funny", "so don't you dare laugh"]
encoder = StaticTokenizerEncoder(loaded_data, tokenize=lambda s: s.split())
encoded_data = [encoder.encode(example) for example in loaded_data]
```

But for PyTorch[10], you might need to refer to several TokenizerEncoders to do the same thing. And they might not have a convenient interface for you to customize your corpus or how you want to do the replacements. Overall, Keras gives more convenient interfaces for users to directly specify their models or tokenizers. PyTorch provides more low-level prototypes for users to build their codes on top of these prototypes. We should also admit that PyTorch provides comprehensive text preprocessing libraries such as TorchText and TorchNLP, but from my point of view, these libraries seem much more complicated and not as well encapsulated as Keras does. The preprocessing functionality of PyTorch is more powerful than Keras has but it requires longer learning curves and is not that convenient to use.

In conclusion, as a beginner, I will prefer Keras for doing NLP-related projects such as text categorization used in this course because for users not working in academic research or industrial development, they may not need to train a large number of data sets. That is to say, the performance advantage of PyTorch may not be as useful as the quick development advantage that Keras has. But it also depends, for some models mentioned in the lecture such as K-NN that have a larger speed and memory requirement, we may need to refer to PyTorch to make more configurations to constrain the memory usage and to tune our model train faster. In most cases, Keras, acting as the lightweight version of TensorFlow, is sufficient to produce a great natural language understanding or a natural language generation model. So my conclusion is that Keras is more suitable for projects that require fast prototyping and will be a good toolkit for beginners. But in order to take a step forward to produce a more complex NLP model or to pursue a better training performance, PyTorch will be a better choice.

References:

1. https://www.ibm.com/products/watson-assistant
2. https://www.salesforce.com/products/einstein/overview/
3. https://analyticsindiamag.com/tensorflow-vs-keras-which-one-should-you-choose/
4. https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/
5. https://deepsense.ai/keras-or-pytorch/
6. https://analyticsindiamag.com/pytorch-vs-keras-who-suits-you-the-best/
7. https://www.geeksforgeeks.org/keras-vs-pytorch/
8. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html
9. https://keras.io/api/layers/recurrent_layers/lstm/
10. https://stackoverflow.com/questions/57767854/keras-preprocessing-text-tokenizer-equivalent-in-pytorch