Advanced Keras: motivation

In introducing Deep Learning, we have asserted that

> *It's all about the Loss function*

That is: the key to solving many Deep Learning problems

- Is not in devising a complex network architecture
- But in writing a Loss function that captures the semantics of the problem

Up until now

- We have been using pre-defined Loss functions (e.g., `binary_crossentropy`)
- Specifying the Loss function in the compile statement

```
model.compile(loss='binary_crossentropy')
```

- Using the built-in "training loop" (cycling through epochs, using Gradient Descent to update weights)
    - `model.fit( ... )`

You can write your own loss functions (https://keras.io/api/losses/)

In Keras, a Loss function has the signature

```
loss_fn(y_true, y_pred, sample_weight=None)
```

But what if your Loss function needs access to values that are not part of the signature ?

In that case, you might need to write your own training loop (https://keras.io/guides/writing_a_training_loop_from_scratch/)

- Cycle through epochs
- Within each epoch, cycle through mini-batches of examples
- For each mini-batch of examples: compute the custom loss
- Compute the gradient of the loss with respect to the weights
- Update the weights

Rather than writing the entire training loop, it sometimes suffices to just write the [train step (https://keras.io/guides/customizing_what_happens_in_fit/)](https://keras.io/guides/customizing_what_happens_in_fit/)

- The "body" of the doubly-nested loop (for each epoch, for each mini-batch)
- The part that
    - Computes the Loss
    - Computes gradients
    - Updates the weights

In this module we will

- Illustrate the Functional model
    - allows more complex network architectures than the Keras Sequential model
- Show how to write complex Loss functions
- Show how to write custom training loops

In addition, we will show you how to create new Layer types (e.g., Transformer)

Functional model

[Autoencoder (Autoencoder_example.ipynb#Library-for-Vanilla-Autoencoder-demo)](Autoencoder_example.ipynb#Library-for-Vanilla-Autoencoder-demo)

- Functional model

Issues

- We could use a Sequential model with initial Encoder layers and final Decoder layers
    - But we would not be able to independently access the Encoder nor the Decoder as isolated models

VAE: Complex Loss; Manual Gradient updates

[Variational Autoencoder (VAE) (https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/generative/ipynb/vae.ipynb#scrollTo=DEU05Oe0vJrY)](https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/generative/ipynb/vae.ipynb#scrollTo=DEU05Oe0vJrY)

- Functional model
- [VAE: Custom train step (https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/generative/ipynb/vae.ipynb#scrollTo=0EHkZ1WCHw9E)](https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/generative/ipynb/vae.ipynb#scrollTo=0EHkZ1WCHw9E)
    - Complex loss

Issues

Transformer: Custom layers, Skip connections, Layer Norm

[Transformer layer (https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/nlp/ipynb/neural_machine_translation_with_transformer.ipynb#s lMkSs)](https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/nlp/ipynb/neural_machine_translation_with_transformer.ipynb)

- Functional model
- Custom layers
- Layer Norm
- Skip connections

The following diagram shows the architecture, which we can compare to the code

- [Full architecture diagram (compare with code) (Transformer.ipynb#Full-Encoder-Decoder-Transformer-architecture)](Transformer.ipynb#Full-Encoder-Decoder-Transformer-architecture)

We can dig deeper to examine how the Attention layers are implemented in code:

- [Scaled dot-product attention (https://www.tensorflow.org/text/tutorials/transformer#scaled_dot_product_atten](https://www.tensorflow.org/text/tutorials/transformer#scaled_dot_product_attention)
- [Multi-head attention (https://www.tensorflow.org/text/tutorials/transformer#mult head_attention)](https://www.tensorflow.org/text/tutorials/transformer#multi-head_attention)

Issues

- Build a new layer type
- Why are the components layers (e.g., Dense, MultiHeadAttention, LayerNormalization) instantiated in the class constructor
    - As opposed to being defined in the "call" method
    - Because we **need** one instance of the layer
        - Not a new instance each time the class is "called" per batch
            - This would result in brand new weights for each example batch
    - The "call" method accesses the shared layer instances and performs the computation using them

Neural Style Transfer

[Neural Style Transfer (https://keras.io/examples/generative/neural_style_transfer/)](https://keras.io/examples/generative/neural_style_transfer/)

- Complex Loss
- Custom training loop

[Here (https://www.tensorflow.org/tutorials/generative/style_transfer)](https://www.tensorflow.org/tutorials/generative/style_transfer) is a tutorial view of the notebook.

In [2]:

```
print("Done")
```

```
Done
```