Music Composer

Fateme Tavakoli | Amirreza Amouie

Introduction to AI Project, Winter 2019

Dr. Pilehvar

“What I cannot create,
I cannot understand.”

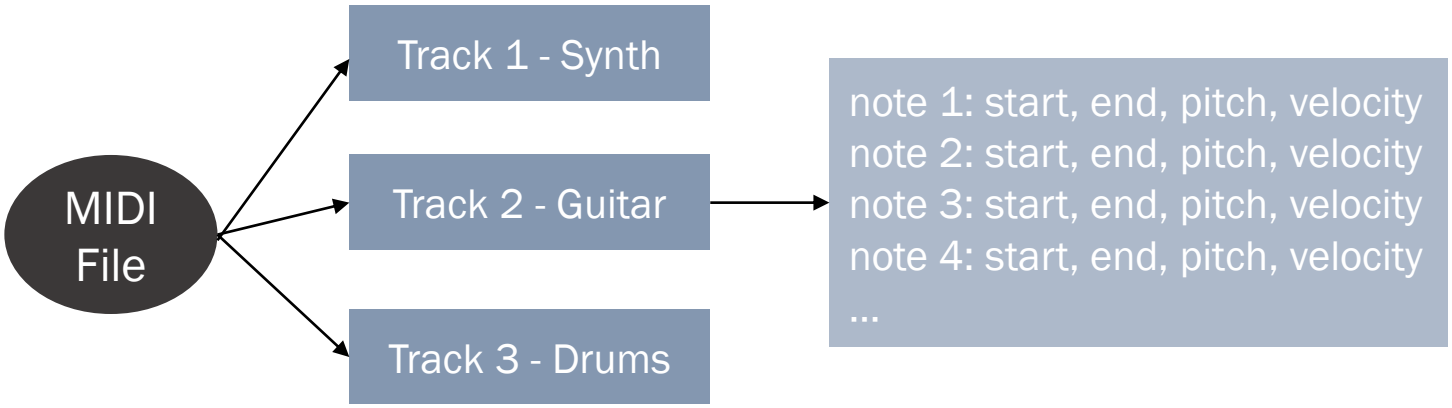
- Richard Feynman

Data

- Project's dataset is a collection of about 175,000 MIDI files *(we used The Lakh MIDI Dataset v0.1)*
- We used `pretty midi` to help us with reading and writing.

Data

- MIDI file structure looks like this:



Data

- As mentioned in the chart in previous slides, we see that each song has multiple instruments
- But we chose the instrument with longest bars (so in each song, the instrument with greatest number of notes were kept)
- Drum based instruments were left out because they have different annotation system than other instruments like piano, violin, etc.

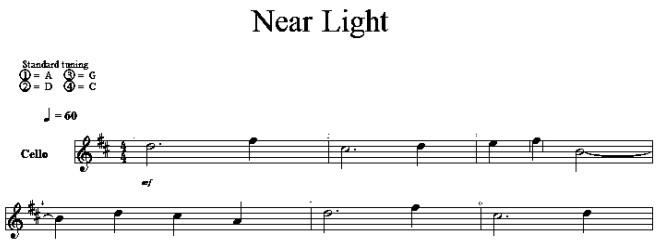
Data

- Each note has this values:
  - Start time
  - End time
  - Pitch
  - Velocity
- We categorized each note due to its values.
- Then we hot-encoded notes due to categories

Data - hot-encode

- Instead of start time and end time we used the difference between them (duration) and we already had velocities and pitches.
- We calculated max value for each of the three variables mentioned above and splitted the notes according to their distribution of all notes in that song.

Data



Near Light.mid

```
4d54 6864 0000 0006 0001 0004 01e0 4d54
726b 0000 0072 00ff 0308 5669 6f6c 696e
2031 00c0 0000 ff51 030f 4240 00ff 5804
0402 1808 0090 4e48 0090 4a48 8360 804e
4000 804a 4000 904a 4883 6080 4a40 0090
4a48 8360 804a 4000 904a 4883 6080 4a40
0090 4e48 8360 804e 4000 904a 4883 6080
4a40 0090 4a48 8360 804a 4000 904a 4883
6080 4a40 00ff 2f00 4d54 726b 0000 00d0
00ff 0308 5669 6f6c 696e 2032 00c0 0000
9042 4800 903b 488f 0080 4240 0080 3b40
0090 4248 0090 3948 8f00 8042 4000 8039
4000 9042 4800 903e 4887 4080 4240 0080
3e40 0090 4248 0090 3748 8740 8042 4000
8037 4087 4090 4248 0090 3948 8740 8042
```

preprocess →

```
[102, 324, 122, 403, …]
```

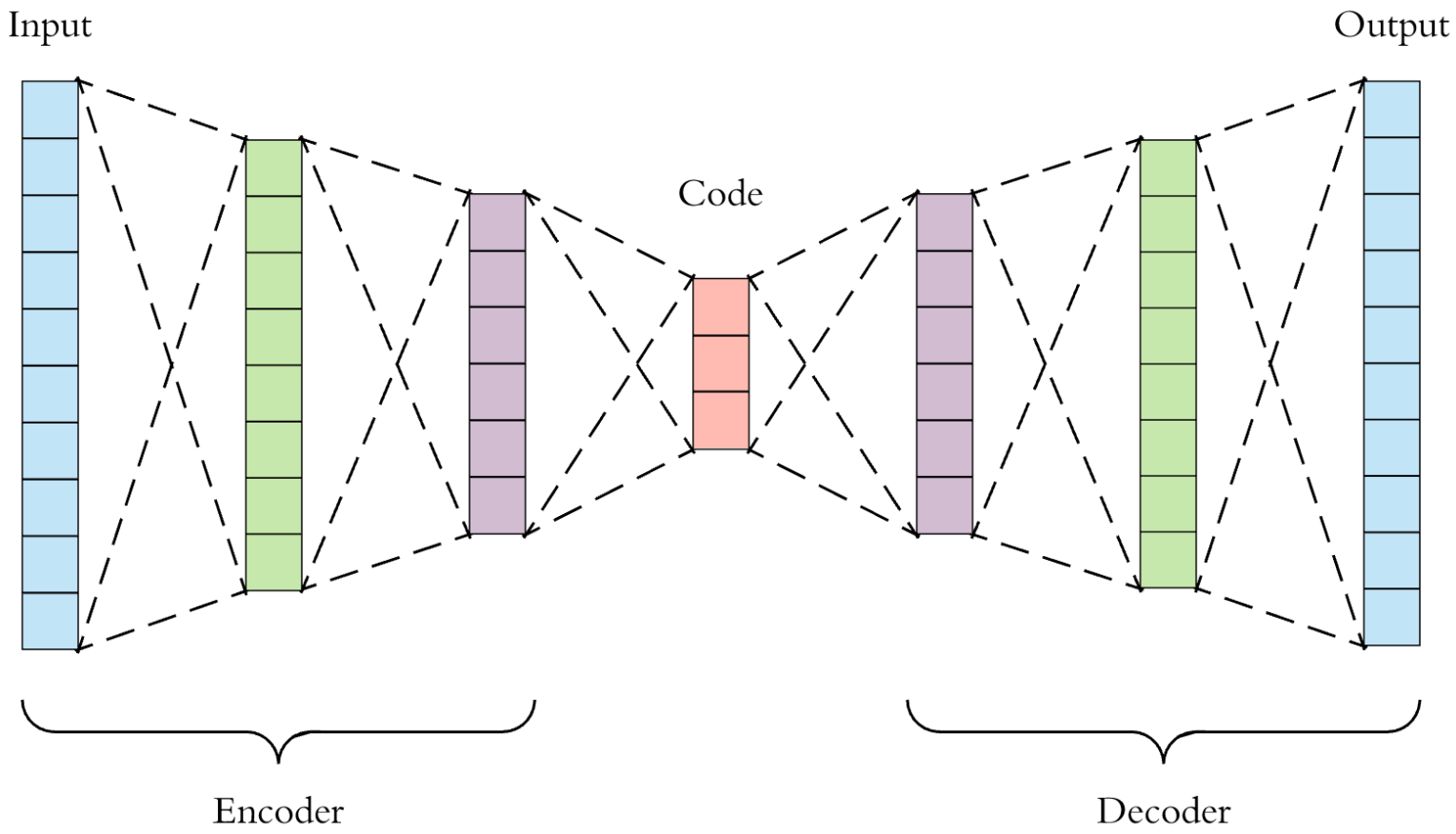
Batches

- Instead of loading all the training dataset at once, training data are loaded in sets of specific size (definition of batch)
- We have a `BatchProcessor` class that generates batches for each iteration.
- Each batch's notes are one-hot-encoded together.

Model

- We needed an architecture that ***generates*** new ***songs***.
- First of all, because we needed to *generate* new songs, we used encoder-decoder architecture.
- Second of all, because music and songs are sequential, we needed to use recurrent networks; so we chose LSTM.

Model – What is encoder and decoder?



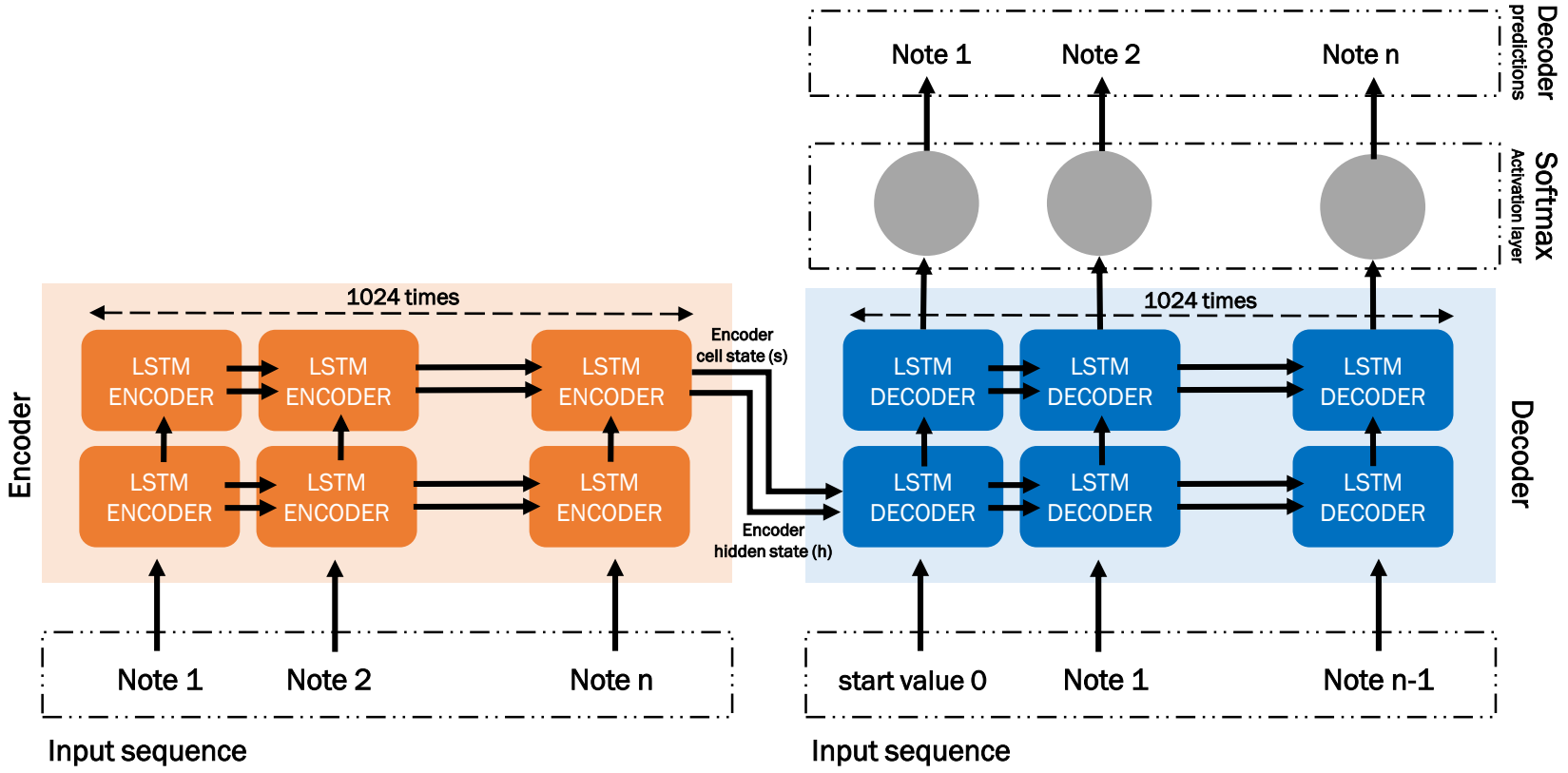
Model – Good things about LSTM

- It can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies because of gradient vanishing
- LSTM has memory; it can maintain information in its memory for long periods of time.
- A set of gates is used to control when information enters the memory, when it's output, and when it's forgotten.

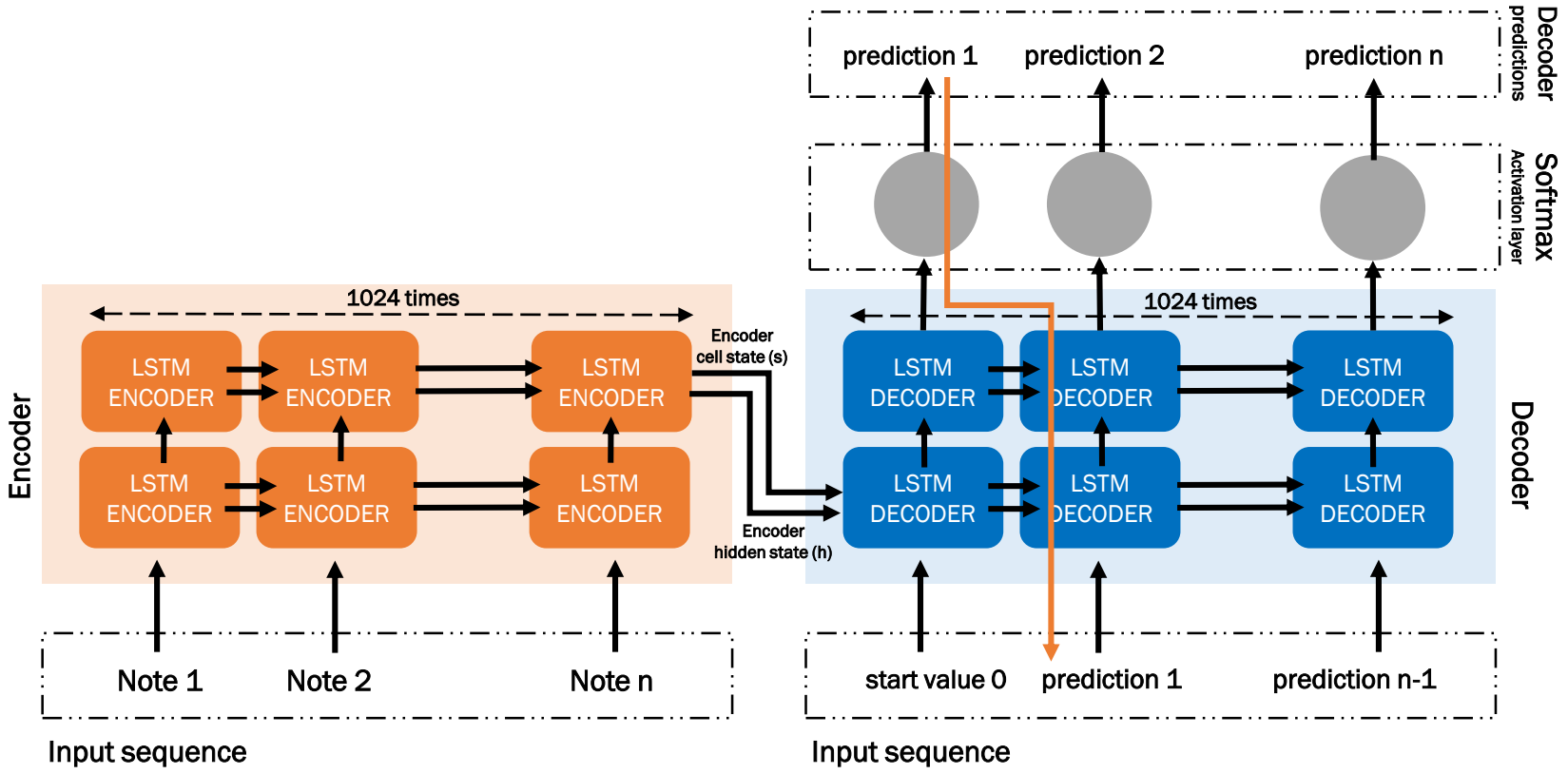
Network implementation - TF

- We used TensorFlow.
- We needed to have control over our network and TensorFlow gave us more control than Keras
- For example, in TF, we have a computation graph that we can do any computation with any input that we want.
- Also TF is more used in researches and building new networks as well.
- TF's syntax is harder than Keras and might be more confusing.

Training Model



Testing/Inference Model



CODE SNIPPETS ->

Code Snippets – One-Hot Encode

```
def one_hot_encode(self, song):
    indices = self.encode_song(song)[:60]
    n_labels = len(indices)
    n_unique_labels = 444
    one_hot_encoded_song = np.zeros((n_labels, n_unique_labels))
    one_hot_encoded_song[np.arange(n_labels), indices] = 1

    return one_hot_encoded_song
```

Code Snippets – Label Encoder

```
def label_encoder(self, note, song):
    d = song.max_duration / 4
    DurationLabel = int((note.end - note.start) / d)
    v = song.max_vel / 4
    VelocityLabel = int(note.velocity / v)
    p = song.max_pitch / 4
    PitchLabel = int(note.pitch / p)
    return DurationLabel * 10 + VelocityLabel * PitchLabel*100
```

Code Snippets – Batch

```
def get_next_batch(self):
    self.midi_processor.read_files(
          self.last_iteration * self.batch_size,
         (self.last_iteration + 1) * self.batch_size)

    batch = self.get_batch(self.last_iteration)
    encoded_batch = self.hot_encode_batch(batch)

    self.last_iteration += 1
    return encoded_batch
```

Code Snippets – Loss

```
with tf.name_scope("optimization"):
    # Loss function - weighted softmax cross entropy
    cost = tf.contrib.seq2seq.sequence_loss(
           training_logits,
           targets,
           masks)
```

Code Snippets – Gradient Clipping and Optimizer

```
# Optimizer
optimizer = tf.train.AdamOptimizer(lr)


# Gradient Clipping
gradients = optimizer.compute_gradients(cost)
capped_gradients = [(tf.clip_by_value(grad, -1., 1.), var) for grad,
                    var in gradients if grad is not None]
train_op = optimizer.apply_gradients(capped_gradients)
```

Code Snippets – Graph Run

```
with tf.Session(graph=train_graph) as sess:
        sess.run(tf.global_variables_initializer())

 _, loss = sess.run(...)

batch_train_logits = sess.run(...)

batch_valid_logits = sess.run(...)
```

Code Snippets – Accurancy

```
# accuracy
train_acc = get_accuracy(target_batch, batch_train_logits)
valid_acc = get_accuracy(valid_targets_batch, batch_valid_logits)
```

Code Snippets – Saving Model

```
#Save model
saver = tf.train.Saver()
saver.save(sess, save_path)
```

Code Snippets – Model

```
train_logits, inference_logits = seq2seq_model
enc_outputs, enc_states = encoding_layer(input_data, ...)
train_output, infer_output = decoding_layer(dec_input, ...)
```

Code Snippets – Optimizer

```
train_output, infer_output = decoding_layer(dec_input, ...)
hamuntrain_output = decoding_layer_train(...)
```