<table>
<thead>
<tr>
<th>Tuesday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5  OCT 22ND</strong></td>
<td><strong>OCT 24TH</strong></td>
</tr>
<tr>
<td>9: Generative Grading II</td>
<td>10: Neural Inference</td>
</tr>
<tr>
<td>Really zero shot?</td>
<td>Best tools for a hard problem</td>
</tr>
<tr>
<td>Tonight: Feedback on ideas from teaching team</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>OCT 31ST</strong></td>
</tr>
<tr>
<td><strong>6  OCT 29TH</strong></td>
<td></td>
</tr>
<tr>
<td>11: Research in Generative Grading</td>
<td>12: Teachable Agents</td>
</tr>
<tr>
<td>Make progress in a brand new field</td>
<td>Students thinking generatively</td>
</tr>
<tr>
<td>Have a teaching-team 1 on 1 meeting before wed</td>
<td><strong>Due: Assn 3</strong></td>
</tr>
<tr>
<td></td>
<td>1 min presentations in class</td>
</tr>
</tbody>
</table>

Submit 2 proposals with team
The Code.org Dataset

- Students learning **nested loops**
- 50k students with **1.5 million submissions** to a curriculum of **8 exercises**.
- **800 human labels** across 2 of the exercises.
How would you modify what you’ve learned to draw these squares? They start at 15 pixels long, the largest is 300 pixels long, and each square is 15 pixels larger than the last.

```blocks
when run
  for counter from 15 to 300 count 15
```
Edit distance is meaningless, Code is Zifian

Exponential combination of decisions. Super fat tailed. Everything looks unique
Traditional Deep Learning Doesn’t Work

Label student code

Last Problem (P8)

Feedback F1 Score

Old Gaurd

Humans
Inaccurate, Uninterpretable, and Data Hungry

Label student code

Last Problem (P8)

Feedback F1 Score

Old Guard

Deep Learning

Humans

Piech et al., ICML 2014
Inaccurate, Uninterpretable, and Data Hungry

Label student code

Last Problem (P8)

Feedback F1 Score

<table>
<thead>
<tr>
<th></th>
<th>Old Guard</th>
<th>Deep Learning</th>
<th>Humans</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Label student code

Feedback F1 Score

Last Problem (P8)

- Old Gaurd
- Deep Learning
- Humans

We need one shot learning

We need verifiability
ideaToText
How do you know if a grammar is good?
Testing "distribution" similarity

a) One way to know if you are doing a good job

b) Could be important for automating pieces of this process
Can you quantify a good grammar?
Can you quantify a good grammar?

Better Grammar

Worse Grammar
Quantify this difference

Divergences
• L1 norm
• L2 norm
• KL divergence (entropy)

\[ D_{KL}(P \parallel Q) = - \sum_{x \in X} P(x) \log \left( \frac{Q(x)}{P(x)} \right) \]
Motivation for Earth Movers Distance

Requires that you have an understanding of distance in the “input” space
Motivation for Earth Movers Distance
Motivation for Earth Movers Distance

Requires that you have an understanding of distance in the “input” space
Can we think of it a little bit more like this?

(a) Code.org  
(b) University classroom
Is there a concept of “program distance”? 

(a) $D_{\text{unlabeled}}$  
(b) $D_{\text{syn}}$: Turn/Move  
(c) $D_{\text{syn}}$: No Repeat
At the end of the day, what matters is: Can you predict labels on unseen programs!
Should we scale prediction accuracy by counts?

How do we balance the insight that common programs are more important than rare ones?

Our evaluation will be balanced between head and tail.
At this point we have a ton of synthetic data
How do you make predictions?
You can’t just use nearest neighbors 😞

1 million unique solutions to programming Linear Regression

WWW 2014
For(15, 300, 15) {
  Repeat(4) {
    Move(Counter)
    TurnRight(90)
  }
}

The Prediction Engine
For(15, 300, 15) {
  Repeat(3) {
    Move(Counter)
    TurnRight(90)
  }
}
Recurrent Neural Networks
Recurrent Neural Networks

A vector which represents what we have seen so far

Vector encoding the next piece of information

At each time point you can optionally make a prediction

*Note: this was coded pre-tensor flow
Recurrent Neural Networks

*Note: this was coded pre-tensor flow*
Recurrent Neural Networks

$h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow \cdots \rightarrow h_T$

$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_T$

$y_T$
For (15, 300, 15) { Repeat (3) { Move (Counter) TurnRight(90) } }
For (15, 300, 15) { Repeat (3) { Move (Counter) TurnRight(90) } }

predictions

hidden

inputs

raw text
some open problems

• How to capture decomposition invariance
• How to simulate *process* of writing a program
• Can you learn these generative grammars from data?
• what makes a grammar useful?
• What is the best way to know what your grammar is missing?
Can you make it?

Synthetize some programs and use them to train an RNN
What you need to do for part 2

1. Add “labels” to your ideaToText
2. Simulate some data and save it
3. Write your neural network model
4. Run your model:
   1. preprocess the data
   2. call train
   3. call test

Submit: your trained model via the *checkpoints* directory
1. Add “labels” to your ideaToText
2. Simulate some data and save it
3. Write your neural network model
4. Run your model:
   1. preprocess the data
   2. call train
   3. call test

shapeLoop-none
square-none
side-none
shapeLoopHeader-missingValue
shapeLoopHeader-wrongOrder
shapeLoopHeader-wrongDelta
shapeLoopHeader-wrongEnd
shapeLoopHeader-wrongStart
square-armsLength
square-unrolled
square-wrongNumSides
side-forgotLeft
side-forgotMove
side-wrongMoveLeftOrder
side-armsLength
turn-wrongAmount
turn-rightLeftConfusion
move-wrongAmount
What you need to do for part 2

1. Add “labels” to your ideaToText
2. Simulate some data and save it
3. Write your neural network model
4. Run your model:
   1. preprocess the data
   2. call train
   3. call test

```python
[{'code': '', 'labels': []},
 {'code': '', 'labels': []},
 {'code': '', 'labels': []},
]
```

Submit: your trained model via the checkpoints directory
1. Add “labels” to your ideaToText
2. Simulate some data and save it
3. **Write your neural network model**
4. Run your model:
   1. preprocess the data
   2. call train
   3. call test

---

**models.py**

```python
class FeedbackNN(nn.Module):
    
    Neural network responsible for ingesting a tokenized student program, and spitting out a categorical prediction.

    We give you the following information:
    vocab_size: number of unique tokens
    num_labels: number of output feedback labels

    
```

Submit: your trained model via the *checkpoints* directory
1. Add “labels” to your ideaToText
2. Simulate some data and save it
3. Write your neural network model
4. Run your model:
   1. preprocess your data
   2. call train
   3. call test

Submit: your trained model via the checkpoints directory
1) How to make an RNN
# Gated recurrent unit: see blog – https://colah.github.io/posts/2015-08-Understanding-LSTMs/
self.rnn = nn.RNN(INPUT_DIM, HIDDEN_DIM, batch_first=True)

2) How to call forward on an RNN
_, hidden = self.rnn(packed)

3) How to pack data for an RNN forward call
# this is an important function ... if you have sentences that are not the
# same "length", then we cant store them in a tensor w/o pad tokens. But
# when learning, we should not take into account pad tokens (since they
# are not truly signal). This util: "pack_padded_sequence" flattens everything
# and keeps track of sentence lengths for you.
# See https://gist.github.com/HarshTrivedi/f4e7293e941b17d19058f6fb90ab0fec
# for a tutorial.
packed = rnn-utils.pack_padded_sequence(
    embed_seq,
    sorted_lengths.data.tolist() if batch_size > 1 else length.data.tolist(),
    batch_first = True
)
nn.RNN(INPUT_DIM, HIDDEN_DIM, batch_first=True)