Innovative Digital Learning Environments

CS398
CS398 Short Pitches

Tuesday Nov 5th (everyone gets 2 to 3 mins)
Challenge

HOUR OF CODE
Challenge

The Hour of Deep Learning
Challenge

How can you teach **Logistic Regression** and/or **Deep Learning** to a non-tech 30 year old in 1 hour using **online tools**
Claim:
Programming (code -> run -> repeat) was the **first digital pedagogy**

People were learning:
logic / design / how to learn
What circuit is in the black box?
How have digital pedagogies led to new learning experiences?
Novel Pedagogy #1: Virtual Science Lab
Novel Pedagogy #1: Virtual Science Lab

Engin Bumbacher, Carl Wieman & Paulo Blikstein
Four Prototypical Trajectories
Novel Pedagogy #2: Teachable Agents

This is Betty

Students build Betty’s brain (is that weird?)
Novel Pedagogy #2: Teachable Agents
Novel Pedagogy #3: Coding

3: Hello World App

```
function start() {
  // Change the part inside the quotes to see the text
  var text = new Text("Hello, World");

  var x = getWidth() / 2 - text.getWidth() / 2;
  var y = getHeight() / 2 + text.getHeight() / 2;
  text.setPosition(x, y);
  add(text);
}
```

Hello, World
Novel Pedagogy #3: Coding

Archivo: LíneaDeSonido.java

```java
import stanford.karel.*;
public class LíneaDeSonido extends Karel {
    public void run() {
        /* Se repite hasta que Karel se enfrenta a una pared. */
        while(frenteDespejado()) {
            /* colocar un cono en la plaza actual */
            ponerConeo();
            /* moverse a la siguiente plaza */
            moverse();
        }
        /* resuelve el error del poste de la cerca */
        ponerConeo();
    }
}
```

Cambiara mundo

```
7 + + + + + + +
6 + + + + + + +
5 + + + + + + +
4 + + + + + + +
3 + + + + + + +
2 + + + + + + +
1 + + + + + + +
    1 2 3 4 5 6 7
```
Novel Pedagogy #3: Coding

```python
1 a = 5
2
3 while(true):
   4     a += 1
```

Frames

- Global frame
  - `a` = 5

Objects

Step 2 of 2
Honorable Mentions
YouTube for Guitar
Dragon Box
It's a doggy-dog world out there.

The phrase doggy-dog world appears to be a miswritten version of a more common phrase.
First, the human-computer interaction (HCI) project, led by Prof. Michael Bernstein at Stanford University, set out to create a new paid crowdsourcing marketplace a la Amazon Mechanical Turk. In current crowdsourcing marketplaces, workers feel disrespected, and requesters do not trust the results they receive [32, 51]. The HCI project works on designing,

Udacity: Crowd sourced teaching?
General Takeaways
General Patterns

• Immediate, objective feedback on your thinking
• Make it feel real
• Neither too easy nor too hard
• Demonstrate long term value
• Be engaging / fun
• New opportunities for collaboration

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>(Manipulative/Observant)</td>
</tr>
<tr>
<td></td>
<td>Students are dynamic. They are not passive listeners but play active roles in learning activities, actively manipulating objects and information, and observing results from the learning activities.</td>
</tr>
<tr>
<td>Constructive</td>
<td>(Articulative/Reflective)</td>
</tr>
<tr>
<td></td>
<td>Students construct their own understanding and knowledge, reflect and articulate their personal understandings of phenomena/activities observed.</td>
</tr>
<tr>
<td>Authentic</td>
<td>(Complex/Contextual)</td>
</tr>
<tr>
<td></td>
<td>Students engage in authentic tasks and problems rather than memorizing abstract concepts and ideas; solving real-life problems.</td>
</tr>
<tr>
<td>Intentional</td>
<td>(Goal-directed/Regulatory)</td>
</tr>
<tr>
<td></td>
<td>Students set their learning goals and plan their learning pathways.</td>
</tr>
<tr>
<td>Cooperative</td>
<td>(Collaborative/Conversational)</td>
</tr>
<tr>
<td></td>
<td>Students work with peers to solve problems/tasks through collaborative activities and discussions to better learn and apply their knowledge.</td>
</tr>
</tbody>
</table>

Characteristics of Meaningful Learning Attributes (Jonassen et al 2003)
Challenge

How can you teach **Logistic Regression** and/or **Deep Learning** to a non-tech 30 year old in 1 hour using **online tools**?
\[ P(y^{(i)} = 1) = \sigma\left(\sum_j \theta_j \cdot x_j^{(i)}\right) \]
Weights

\[ P(y^{(i)} = 1) = \sigma\left(\sum_j \theta_j \cdot x_j^{(i)}\right) \]
Weighed Sum

\[ P(y^{(i)} = 1) = \sigma \left( \sum_j \theta_j \cdot x_j^{(i)} \right) \]
\[
P(y^{(i)} = 1) = \sigma \left( \sum_j \theta_j \cdot x_j^{(i)} \right)
\]
Prediction

\[ P(y^{(i)} = 1) = \sigma\left(\sum_j \theta_j \cdot x_j^{(i)}\right) \]
Parameters Affect Prediction

\[ P(y^{(i)} = 1) = \sigma\left(\sum_{j} \theta_j \cdot x_j^{(i)}\right) \]
Parameters Affect Prediction

\[ P(y^{(i)} = 1) = \sigma \left( \sum_j \theta_j \cdot x_j^{(i)} \right) \]
Parameters Affect Prediction

\[ z = 2.1 \]
\[ \sigma(z) = 0.7 \]

\[ P(y^{(i)} = 1) = \sigma\left( \sum_j \theta_j \cdot x_j^{(i)} \right) \]
Logistic Regression

\[ P(Y = 1 | x) = \sigma(\theta^T x) \]
Gradient Ascent / Descent

Walk uphill and you will find a local maxima
(if your step size is small enough)

Note: Logistic regression LL function is convex
Logistic Regression

1. Make a prediction

\[ P(Y = 1 | X = x) = \sigma(\theta^T x) \]

2. Calculate the prediction loss (-log likelihood)

\[ \text{Loss}(\theta) = \sum_i \text{binaryCrossEntropy}(P(y^{(i)} = 1), y^{(i)}) \]

3. Get derivative of log likelihood with respect to thetas

\[ \frac{\partial \text{Loss}}{\partial \theta_j} = \text{loss.backward()} \]

Auto grad
We Can Put Neurons Together
Learning Goals

Learners should be able to:

• Know supervised learning paradigm (train/test)
• Understand when supervised classification is appropriate
• Understand the components of logistic regression

• Reach: can learners implement logistic regression?
How this works
def hacMergeIdeas(studentList):
    # originally each student is their own "group"
    groups = studentList

    # continue until we converge on one idea
    while len(groups) > 1:
        # each group should match with a group
        # as similar as possible to themselves
        pairings = findPairings(60)

        # each pair should merge their ideas!!!!
        newGroups = []
        for pair in pairings:
            newGroup = mergeIdeas(pair)
            newGroups.append(newGroup)

        # there should be half as many groups.
        groups = newGroups
def hacMergeIdeas(studentList):
    # originally each student is their own "group"
    groups = studentList

    # continue until we converge on one idea
    while len(groups) > 1:
        # each group should match with a group
        # as similar as possible to themselves
        pairings = findPairings(60)

        # each pair should merge their ideas!!!!
        newGroups = []
        for pair in pairings:
            newGroup = mergeIdeas(pair)
            newGroups.append(newGroup)

        # there should be half as many groups.
        groups = newGroups
def hacMergeIdeas(studentList):
    # originally each student is their own "group"
    groups = studentList

    # continue until we converge on one idea
    while len(groups) > 1:
        # each group should match with a group
        # as similar as possible to themselves
        pairings = findPairings(60)

        # each pair should merge their ideas!!!!
        newGroups = []
        for pair in pairings:
            newGroup = mergeIdeas(pair)
            newGroups.append(newGroup)

        # there should be half as many groups.
        groups = newGroups
def hacMergeIdeas(studentList):
    # originally each student is their own "group"
    groups = studentList

    # continue until we converge on one idea
    while len(groups) > 1:
        # each group should match with a group
        # as similar as possible to themselves
        pairings = findPairings(60)

        # each pair should merge their ideas!!!!
        newGroups = []
        for pair in pairings:
            newGroup = mergeIdeas(pair)
            newGroups.append(newGroup)

        # there should be half as many groups.
        groups = newGroups
def hacMergeIdeas(studentList):
    # originally each student is their own "group"
    groups = studentList

    # continue until we converge on one idea
    while len(groups) > 1:
        # each group should match with a group
        # as similar as possible to themselves
        pairings = findPairings(60)

        # each pair should merge their ideas!!!!
        newGroups = []
        for pair in pairings:
            newGroup = mergeIdeas(pair)
            newGroups.append(newGroup)

        # there should be half as many groups.
        groups = newGroups
def hacMergeIdeas(studentList):
    # originally each student is their own "group"
    groups = studentList
    
    # continue until we converge on one idea
    while len(groups) > 1:
        # each group should match with a group
        # as similar as possible to themselves
        pairings = findPairings(60)

        # each pair should merge their ideas!!!!
        newGroups = []
        for pair in pairings:
            newGroup = mergeIdeas(pair)
            newGroups.append(newGroup)

        # there should be half as many groups.
        groups = newGroups
Failed merge?

Rock paper scissors to decide who goes on
Generate a prototype

(10 mins)
Find a match

(1 mins)

Everyone in the group should participate
Four Prototypical Trajectories

Merge

(~5 mins)

Everyone in the group should participate