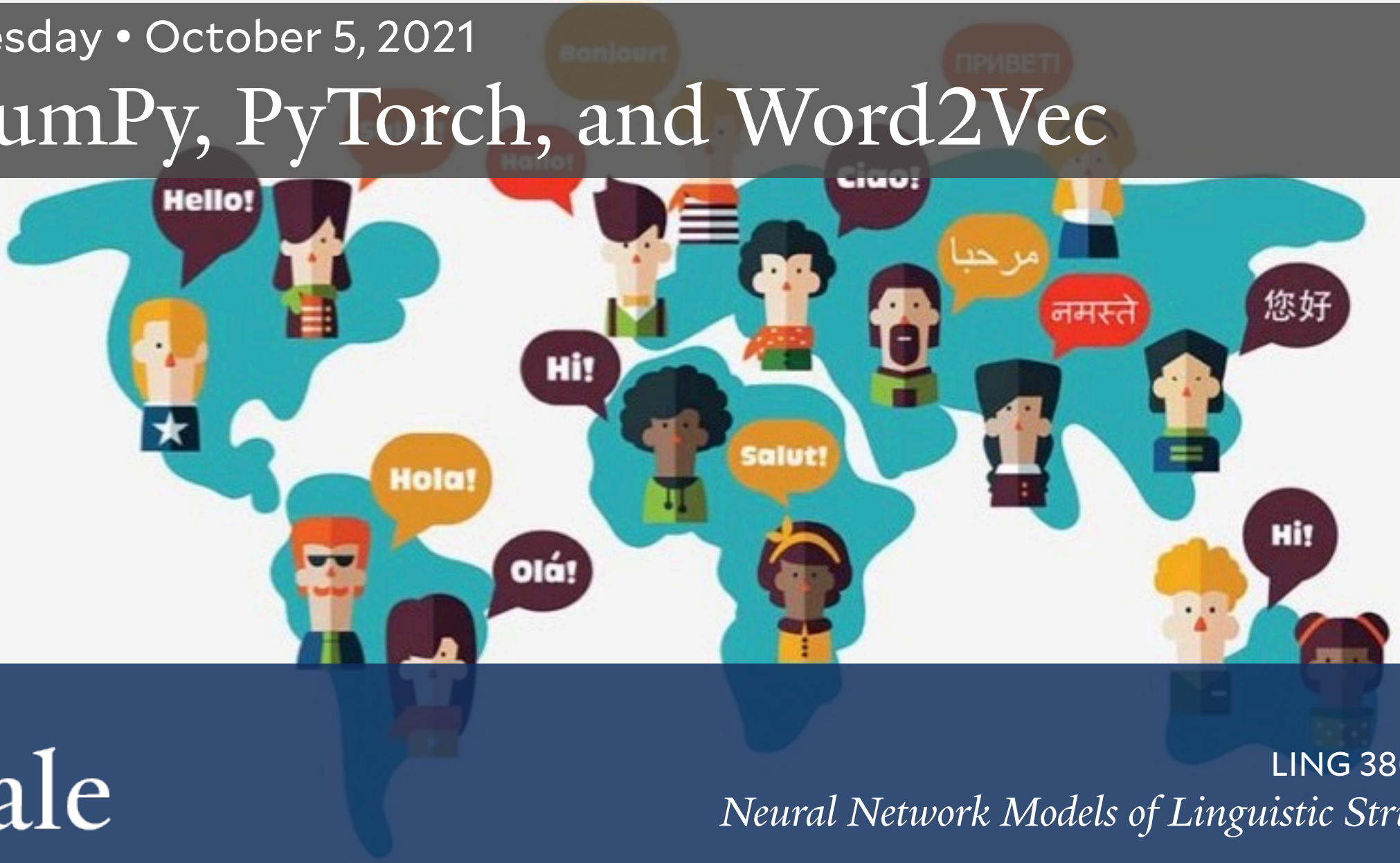
Tuesday • October 5, 2021

NumPy, PyTorch, and Word2Vec



Yale

LING 380/780

Neural Network Models of Linguistic Structure

Assignment 1 Review

Assignment 1

- Solutions are available on Canvas.
- Grading is in progress.

Sigmoid

Sigmoid Definition

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Use broadcasting!

```
def sigmoid(x):
  return 1. / (1. + np.exp(-x))
```

Zero Centering

```
def zero_center(x):
  return x - x.mean(axis=-1, keepdims=True)
```

- Use `axis=-1` to take the mean of rows
- Use `keepdims=True` to facilitate broadcasting

Zero Centering

```
def zero_center(x):
  return x - x.mean(axis=-1, keepdims=True)
```

- Example:
  - `x.shape == (2, 3, 4)`
  - `x.mean(axis=-1, keepdims=True).shape == (2, 3, 1)`
  - Mean broadcasts to shape `(2, 3, 4)`

Zero Centering

Incorrect Solution:

```
def zero_center(x):
  return x - x.mean()
```

Takes the mean of *all* items of `x` instead of rows of `x`

Zero Centering

Incorrect Solution:

```
def zero_center(x):
  return x - x.mean(axis=-1)
```

The mean cannot broadcast since it is the wrong shape: `(2, 3)` instead of `(2, 3, 1)`

Even Rows

```
def even_rows(x):
    return x[::2]
```

Indexing Syntax

- `x[i:j:k]` means “every `k`th row from row `i` to row `j`”

Even Rows

Alternative Solution

```
def even_rows(x):
    return x[np.arange(len(x)) % 2 == 0]
```

Even Rows

Alternative Solution

```
def even_rows(x):
    return x[np.arange(len(x)) % 2 == 0]
```

```
array([0, 1, 2, 3, …])
```

Even Rows

Alternative Solution

```
def even_rows(x):
    return x[np.arange(len(x)) % 2 == 0]
```

```
array([0, 1, 0, 1, …])
```

Even Rows

Alternative Solution

```
def even_rows(x):
    return x[np.arange(len(x)) % 2 == 0]
```

```
array([True, False, True, False, …])
```

Mask

```
def mask(x, mask_val, replace_with=-1):
    x[x == mask_val] = replace_with
```

- Indexing by boolean array
- Indexing returns references whose can be modified using =

Mask

Why doesn't this work?

```
def mask(x, mask_val, replace_with=-1):
    y = x[x == mask_val]
    y = replace_with
```

- Sets the **name** y to `replace_with`, but doesn't affect the entries of `x`

Mask

Why does this work?

```
def mask(x, mask_val, replace_with=-1):
    y = x[x == mask_val]
    y[:] = replace_with
```

- The left-hand side of the = in line 2 is the references to the items in y

Accuracy

```
def accuracy(logits, labels):
  return (logits.argmax(axis=-1) == labels).sum() / len(logits)
```

Example

```
logits = array([[1.5, -.75, .25],    labels = array([0, 2, 1, 2])
                [-2.1, -1.3, -.5],
                [-.1, 2.5, 1.4],
                [.3, -.01, .15]]
```

Accuracy

```
def accuracy(logits, labels):
  return (logits.argmax(axis=-1) == labels).sum() / len(logits)
```

```
array([0, 2, 1, 0])
```

Predicted Labels

Example

```
logits = array([[1.5, -.75, .25],
                [-2.1, -1.3, -.5],
                [-.1, 2.5, 1.4],
                [.3, -.01, .15]]
```

```
labels = array([0, 2, 1, 2])
```

Accuracy

```
def accuracy(logits, labels):
  return (logits.argmax(axis=-1) == labels).sum() / len(logits)
```

```
array([True, True, True, False])
```

Which predictions are correct?

Example

```
logits = array([[1.5, -.75, .25],
                [-2.1, -1.3, -.5],
                [-.1, 2.5, 1.4],
                [.3, -.01, .15]]
```

```
labels = array([0, 2, 1, 2])
```

Accuracy

```
def accuracy(logits, labels):
  return (logits.argmax(axis=-1) == labels).sum() / len(logits)
```

$$\frac{3}{\text{Number of Correct Predictions}}$$

Example

```
logits = array([[1.5, -.75, .25],    labels = array([0, 2, 1, 2])
                [-2.1, -1.3, -.5],
                [-.1, 2.5, 1.4],
                [.3, -.01, .15]]
```

Accuracy

```
def accuracy(logits, labels):
  return (logits.argmax(axis=-1) == labels).sum() / len(logits)
```

.75

Averaging over Examples

Example

```
logits = array([[1.5, -.75, .25],    labels = array([0, 2, 1, 2])
                [-2.1, -1.3, -.5],
                [-.1, 2.5, 1.4],
                [.3, -.01, .15]]
```

Get Embedding

```
def get_embedding(w, vocab, all_embeds):
  return all_embeds[vocab.index(w)]
```

Cosine Similarity

Cosine Similarity Definition

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}$$

Matrix of Cosine Similarities

$$\cos(\boldsymbol{X}, \boldsymbol{Y}) = ?$$

Cosine Similarity

$$\cos\left(\boldsymbol{X}_{i,:}, \boldsymbol{Y}_{j,:}\right) = \frac{\boldsymbol{X}_{i,:}\left(\boldsymbol{Y}_{j,:}\right)^{\top}}{\left\|\boldsymbol{X}_{i,:}\right\|\left\|\boldsymbol{Y}_{j,:}\right\|}$$

$$= \frac{\left(\boldsymbol{X}\boldsymbol{Y}^{\top}\right)_{i,j}}{\left\|\boldsymbol{X}_{i,:}\right\|\left\|\boldsymbol{Y}_{j,:}\right\|}$$

$$= \left(\left(\frac{\boldsymbol{X}}{\left\|\boldsymbol{X}_{i,:}\right\|}\right)\left(\frac{\boldsymbol{Y}}{\left\|\boldsymbol{Y}_{j,:}\right\|}\right)^{\top}\right)_{i,j}$$

Cosine Similarity

Matrix of Cosine Similarities

$$\cos(\boldsymbol{X}_{i,:}, \boldsymbol{Y}_{j,:}) = \left(\left(\frac{\boldsymbol{X}}{\|\boldsymbol{X}_{i,:}\|} \right) \left(\frac{\boldsymbol{Y}}{\|\boldsymbol{Y}_{j,:}\|} \right)^{\top} \right)_{i,j}$$

```
def cosine_sim(x, y):
  x_norm = x / np.linalg.norm(x, axis=-1, keepdims=True)
  y_norm = y / np.linalg.norm(y, axis=-1, keepdims=True)
  return x_norm @ y_norm.T
```

Get Neighbors

```
def get_neighbors_of_embedding(embedding, k, vocab, all_embeds):
  cosine_similarities = cosine_sim(all_embeds, embedding)
  top_k_indices = np.argpartition(-cosine_similarities, k)[:k]
  return [vocab[i] for i in top_k_indices]
```

```
def get_neighbors_of_word(word, k, vocab, all_embeds):
  embedding = get_embedding(word, vocab, all_embeds)
  neighbors = get_neighbors_of_embedding(embedding, k + 1, vocab,
                                         all_embeds)
  return [w for w in neighbors if w != word]
```

Analogy

```
def analogy(w1, w2, w3, vocab, all_embeds, k=1):
  e1 = get_embedding(w1, vocab, all_embeds)
  e2 = get_embedding(w2, vocab, all_embeds)
  e3 = get_embedding(w3, vocab, all_embeds)
  e = e2 - e1 + e3
  return get_neighbors_of_embedding(e, k, vocab,
                                    all_embeds)
```

The PyTorch Library

- PyTorch is a Python library for implementing neural networks.
- It has many different packages.
  - `torch`: Computation graphs and backpropagation
  - `torch.nn` (`nn` for short): Neural network architectures, loss functions
  - `torch.nn.functional` (`F` for short): Activation functions
  - `torch.optim` (`optim` for short): Optimization algorithms