

Recurrent Neural Network

Austin Bean

06/20/2019

1 nn notes

- Word embeddings from word2vec
- Train RNN before this or after?
- Which features, in other words

1.1 import some labeled data

- Word embeddings come from word2vec
- Embeddings are organized in the file train_embedding.jl
- Labels come from a Stata file
- Import data, pull out

First load the data

```
using Flux, CSVFiles, DataFrames, Word2Vec
```

```
Error: ArgumentError: Package Word2Vec not found in current path:  
- Run `import Pkg; Pkg.add("Word2Vec")` to install the Word2Vec package.
```

```
dat1 = DataFrame(load("embedded_data.csv"));
```

```
Error: SystemError: opening file "embedded_data.csv": No such file or directory
```

Now separate the labels:

```
target = convert(Array{Float64,1}, dat1[:,1])
```

```
Error: UndefVarError: dat1 not defined
```

And load the inputs - in this case, the longest diet sentence is 22 words long, so there are at most 22 columns in this data.

```
input = convert(Array{Float64,2}, dat1[:,2:22])
```

Error: UndefinedVarError: dat1 not defined

Load the vector of word embeddings:

```
embed_1 = wordvectors("./diet_embed");
```

Error: UndefinedVarError: wordvectors not defined

```
get_vector(embed_1, "NEOSURE")
```

Error: UndefinedVarError: get_vector not defined

- Now setup the RNN.

1.2 Convolutional NN

- Per Goldberg, the CNN layer provides an embedding which helps model nonlinear relationships among features.
- Bi-directional RNN is best, I think.
- This should take each (word, state) as an input and output a (state) to be fed back in.

1.3 Recurrent RNN from Goldberg:

- We are given an input x_1, \dots, x_n . Each input is represented by a vector $x_i \in \mathbb{R}^{d_{in}}$
- In my case the inputs are the words. Each word has a single-dimensional input \mathbb{R} from the embeddings

given by word2vec, though this need not be the case more generally. Single sentences are stored in the `input` table above.

- The RNN maps the vector of inputs x_1, \dots, x_n to the output $y_n \in \mathbb{R}^{d_{out}}$

$$y_{1:n} = RNN^*(x_{1:n})$$

$$y_i = RNN(x_{1:i})$$

$$x_i \in \mathbb{R}^{d_{in}}, y_i \in \mathbb{R}^{d_{out}}$$

- Think of $y_{1:i}$ as a different output for each possible $i = 1, \dots, n$.
-

$$y_{1:n} = RNN^*(x_{1:n})$$

represents the entire sequence of outputs $y_{1:1}, y_{1:2}, y_{1:3}, \dots$

1.4 Preserving a state over time

We can take the output of the system at some stage $i = 1, \dots, n$ and re-input that as an input to the next step of the system. This is a recursive version of the RNN.

The RNN is defined by two functions $O(s)$ and $R(s, x)$ where $O : \mathbb{R}^{f(d_{out})} \rightarrow \mathbb{R}^{d_{out}}$ maps the current state into the output and $R(s, x) : \mathbb{R}^{f(d_{out})} \times \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{f(d_{out})}$ maps the input x and the current state s to the output. These functions are stable over time and take inputs of constant dimension.

In other words, at each step $i = 1, \dots, n$ we have a state vector s_{i-1} and we use this state vector as the input to the function R defining the RNN.

$$\begin{aligned}s_i &= R(s_{i-1}, x_i) \\ y_i &= O(s_i)\end{aligned}$$

In this way, the state vector s_i preserves information about what happened in the preceding sequence of inputs all the way back to the first input (s_0, x_1) . This can be seen by substituting in the functions $R()$ for s at any time period t :

$$\begin{aligned}s_t &= R(s_{t-1}, x_t) \\ s_t &= R(\overbrace{R(s_{t-2}, x_{t-1})}^{=s_{t-1}}, x_t) \\ s_t &= R(R(R(s_{t-3}, x_{t-2}), x_{t-1}), x_t) \\ &\vdots \\ s_t &= R(R(R(\dots R(s_0, x_1), \dots), x_{t-1}), x_t)\end{aligned}$$

The first s_0 vector can be reasonably set to 0 - presumably we have no useful information which would cause us to set it to something else. At each stage we will want to concatenate the current state s_{i-1} with the current input x_i to feed this to the function $R(s, x)$. This can be used as an acceptor, so only the final output state will be used for prediction.

Features in this case are given very easily by the row of the `input` table, but this does not capture the fact that the set of features input to the model might be growing.

2 Implementation

Now define some functions.

```
"""`inp_vec(x, i; bi=false)`this function takes a set of features and returns input
vector.Can do bidirectional as well w/ `bi = true`.Returns a one-dimensional vector
of embeddings of words either up to `i`or from `1:i` and then `i:end` (including `i`
twice)."""
function inp_vec(x, i; bi=false)
    if bi
        return vcat(copy(x[1:i]), reverse(x[i:end]))
    else
        return copy(x[1:i])
    end
end
```

Main.WeaveSandBox0.inp_vec