The ”Predict the Next” task

Many tasks involving sequences start off as many to one:

- takes a short sequence of words (the "seed")
- outputs a prediction for a probable next word

In order to train a model to "predict the next" element we construct a training set

- The features are sequences
- The target is the next element in the sequence

Let

$$[\ \mathbf{s}_{(t)} | 1 \le t \le T\ ]$$

be the elements of sequence $\mathbf{s}$.

We will prepare $(T-1)$ training examples from this single sequence.

$\langle \mathbf{X}, \mathbf{y} \rangle =$

i	$\mathbf{x^{(i)}}$	$\mathbf{y^{(i)}}$
1	$\mathbf{s}_{(1)}$	$\mathbf{s}_{(2)}$
2	$\mathbf{s}_{(1),(2)}$	$\mathbf{s}_{(3)}$
$\vdots$		
i	$\mathbf{s}_{(1),\ldots,(i)}$	$\mathbf{s}_{(i+1)}$
$\vdots$		
$(T-1)$	$\mathbf{s}_{(1),\ldots,(T-1)}$	$\mathbf{s}_{(T)}$

For example, given a sequence of words

$\mathbf{s}$ = "I am taking a class in Machine Learning"

i	$\mathbf{x^{(i)}}$	$\mathbf{y^{(i)}}$
1	[I]	am
2	[I, am]	taking
3	[I, am, taking]	a

Being able to predict the next element may be key to understanding the "logic" underlying a sequence

- You have to understand context and domain
- You have to understand how earlier elements influence latter elements

Semi Supervised Learning

Where do the raw sequences $\mathbf{s}$ come from ?

They may be all around us ! This is especially true for sequences of words

- Each article (news, Wikipedia, etc.) is a collection of sequences (sentences)

Note that the raw sequences are not pairs of Feature/Target.

- You have to transform it into that form

Thus, we refer to this type of learning as *Semi-Superivised*

- transforming unlabeled data into labeled examples

The transformation of raw text examples into structured Feature/Target examples comprise the "Prepare the Data" step in our Recipe.

Recipe for Machine Learning

It is usually the case that Sequence data involves substantial Data Preparation.

Suppose our task is to predict the next word in a sentence.

We are given (or must obtain) a collection of sentences (e.g., one or more documents) as our raw data.

But a sentence is not the format required for the training set of the "Predict the next word" task.

Data preparation is usually a substantial prerequisite for solving tasks involving sequences.

To be precises, the "Predict the next word" task involves

- Training a many to one RNN with examples created from a sequence.
- The elements of a single example are the prefix of a sentence
- The target of the example is the next word in the sentence

Predict the next: data shape

We had warned earlier about the explosion of the number of dimensions of our data. Now is a good time to take stock

- $\mathbf{X}$, the training set, is a matrix with m rows
- Each row is an example $\mathbf{x^{(i)}}$
- Each example is a sequence $[\ \mathbf{x}^\mathbf{(i)}_{(t)} \mid 1 \le t \le ||\mathbf{x^{(i)}}||\ ]$
- Each element $\mathbf{x}^\mathbf{(i)}_{(t)}$ of the sequence encodes a word
- A word is encoded as a One Hot Encoded binary vector of length $||V||$ where V is the set of words in the vocabulary

Target $\mathbf{y^{(i)}}$ is also a word (so is vector of length $||V||$).

- Many to one: target is *not* a sequence

Predict the next: training

Just like training any other type of layer, but more expensive

- Each example involves multiple time steps: forward pass is time consuming
- The derivatives (needed for Gradient Descent) are more complex; backward pass complex and time consuming

Remember:

- the target $\mathbf{y}_{(t)}$ for step t should be $\mathbf{x}_{(t+1)}$ the next input

$$\mathbf{y}_{(t)} = \mathbf{x}_{(t+1)}$$

Predict the next: Training

RNN as a generative model (fun with RNN's)

The "Predict the next" word task is interesting on its own.

But a slight twist will make it extremely interesting

- Suppose we append the prediction to the input sequence
- And feed the extended sequence back into the model
- Repeat ! We can extend the initial short "seed" sequence to arbitrary length.

This is how the example of generating a story from a seed idea works

Since the latent state summarizes the entire prefix of a sequence

- We don't have to feed the entire extended sequence in as input
- We just need to feed the newly generated "next" element into the RNN whose latent state has already encoded the prefix

Once the model has been trained, we can generate increasingly longer sequences at test time as follows:

At test time, we feed a short "seed" sentence

$$\mathbf{x}_{(0)}, \ldots, \mathbf{x}_{(t)}$$

into the model and have it generate output.

But we then feed the output back into the model as input !

$$\mathbf{x}_{(t'+1)} = \mathbf{y}_{(t')} \text{ for } t' \geq t$$

- as in the Decoder in our Language Translation example

Test time: no forcing

The model would generate new text ad infinitum

- The next word generated would be based on what the model has learned from training
- To be the most probable word to follow the prefix

The extended sequences would thus be highly probable sequences from the same domain as the training examples.

Generating strange things

Generating stories from seeds was very popular a few years back.

Let's look at some examples.

But first, a surprise:

- Rather than solving a "predict the next word" task
- All of the following examples were generated by a "predict the next **character**" task !

The choice was motivated by practical considerations

- Both characters and words are categorical values and need to be One Hot Encoded
- The number of possible text characters is much smaller than the number of words in a vocabulary

So One Hot Encoding characters involves vectors that are much shorter than those of words.

This practical consideration would seem to make the Generative task much harder.

It is somewhat amazing that what is generated

- Has correctly spelled words/keywords
- Is Syntactically correct (sentences end with a ".", parentheses/brackets are balanced)
- Is meaningful: the elements/words are arranged in a logical order

Even though

- We have not explicilty identified any of these concepts
- Nor forced training to respect them (via a loss function)

Remember

- All of this behavior was "learned" by identifying the correct next **character**

- Fake Shakespeare (http://karpathy.github.io/2015/05/21/rnn-effectiveness/#shakespeare), or fake politician-speak
- Fake code
- Fake math textbooks (http://karpathy.github.io/2015/05/21/rnn-effectiveness/#algebraic-geometry-latex)
- Click bait headline generator (http://clickotron.com/about)

Training the generative model

Feeding the predicted $\mathbf{y}_{(t)}$ back into the model in order to generate $\mathbf{y}_{(t+1)}$ will require a subtle change during training.

At test time, the picture is straightforward:

Test time: no forcing

But during training, we may make an incorrect prediction for $\hat{\mathbf{y}}_{(t)}$!

This error would cascade, resulting in mis-prediction for all subsequent elements $\hat{\mathbf{y}}_{(t')}$ for $t' > t$

To prevent this, during training

- We **don't** feed predicted $\hat{\mathbf{y}}_{(t)}$ back into the model
- We feed the true $\mathbf{y}_{(t)}$ into the model instead

This is called *teacher forcing*

$$\mathbf{x}_{(t)}^{(\mathbf{i})} = \mathbf{y}_{(t-1)}$$

rather than

$$\mathbf{x}_{(t)}^{(\mathbf{i})} = \hat{\mathbf{y}}_{(t-1)}$$

for $t > t'$.

- When extending the sequence
- A teacher forces the student (model) to continue with the *correct* answer
- Rather than the student's answer
- If it didn't do so, once the student (model) predicted incorrectly, it's errors would compound

Training, with Teacher Forcing

Sampling from the generative model

Remember that a Classifier (the output stage of our model)

- generates a *probability distribution* (over the elements of the vocabulary V)

For the prediction, we usually *deterministically* choose the element of V with highest probability

$$\hat{\mathbf{y}} = \underset{v \in V}{\operatorname{argmax}}\, p(v)$$

Deterministic choice might not be best for the generative process

- One wrong choice propagates to all successive elements of the sequence
- The output is always the same ! Boring !

So what is usually done is that our prediction is a *sample* from the probability distribution.

Summary

Here is the process in pictures

- The training inputs are given in red
- The test (inference) time inputs are given in black

Teacher forcing is indicated in red

- Predictions $[\ \hat{\mathbf{y}}_{(t)} \mid 1 \le t \le T\ ]$ **are not** used as input (lower right)
- Only correct targets $[\ \mathbf{y}_{(t)} \mid 1 \le t \le T\ ]$ are used

Sequence to Sequence: training (teacher forcing)

The input sequence to the Decoder is modified by

- prepending a special "start of output" symbol
$$\mathbf{x}_{(-1)} = \langle \text{START} \rangle$$
- appending a special "end of output" symbol $\langle \text{END} \rangle$ to training examples
    - The Decoder stops when it generates the end of output symbol

The *Encoder* is a many to one RNN

- Takes the variable length "seed" sequence
- Outputs a fixed length representation of the seed
    - This is one of the strengths of an RNN

The *Decoder* is a one to many RNN

- Takes the fixed length representation of the seed produced by the Encoder
    - Used to initialized the Decoder's latent state $\mathbf{h}_{(0)}$
- Outputs a variable length sequence

Generative text: limitations

The model we described (and will explore in a code example) is absolutely primitive

- Predict next *character* rather than next *word*
- Simple: one RNN layer
- Trained on very small number of examples

The results of our toy model may not be as impressive as we hoped.

Understanding why is important

- It will help you to understand what is needed to improve models

- Predicting the next character is a limitation compared to predicting the next word
    - have to learn syntax as well as semantic concepts
    - motivated by limited memory available (need short OHE vectors)
- We have limited data on which to train
- We have limited time to perform the training
    - Back propagation over long sequences is time consuming
- We will use very short sequences in our training data
    - Because of the limited memory and limited time available
    - Can't capture long range dependencies across words
        - e..g, matching gender of word to subject

Here (https://app.inferkit.com/demo) is a link to a state of the art model that is not subject to the same limitations

We will learn about its architecture in a later module.

- 3 billion weights !
- Trained on 500 billion tokens
    - Used $ 42K of electricity

In [2]:

```
print("Done")
```

```
Done
```