Hierarchical Generalisation without Hierarchical Bias: A case of *seq2seq* networks

Saujas Vaduguru and Ujwal Narayan

Argument from the Poverty of the Stimulus

- Children usually don't hear sentences like "*The cat that can eat food will drink milk*"

- But they still know the correct rule for such sentences (more on this later)

- Since children face a 'poverty' of the stimulus of sentences with multiple auxiliaries, but learn the rule anyway, they must have an innate bias towards hierarchical rules

But, is it necessary?

- Some have questioned the need of a hierarchical bias to learn a hierarchical rule
- What if the hierarchical rule can be inferred without bias from sequential data?

Some approaches

- *Perfors, Tenenbaum, and Regier (2011)* showed that a learner whose task is to choose between an innately available hierarchical representation and an innately available linear representation will choose the hierarchical one

- *Fitz and Chang (2017)* studied the interaction between meaning and linguistic constraints, and concluded that no syntactic bias is required to learn the hierarchical rule

Direct predecessors

- Lewis and Elman (2001) trained an RNN to predict the well formedness of questions, but these results were called into question

- Frank and Mathis (2007) used RNNs to transform sentences into questions by creating a semantic representation of the declarative sentence and generating the question

Since then...

- We have had major improvements in RNNs, including the notion of seq2seq networks introduced by *Botvinick and Plaut (2006)* and *Sutskever, Vinyals, and Le (2014)*
- Can these models do better?

The task

- The same task of question formation in English has been used as the prime example of hierarchical rules for grammars for a long time

- We present the same task to our models as well

The task

- Given a declarative sentence, transform it to a question
  - The network is presented with both two tasks – identity and question
  - Identity is a cue to generate the same sentence again
  - Question is to transform the sentence into a question

The task

- Two different languages are used
  - No-agreement language: Auxiliaries are can, could, will, etc.
  - Agreement language: Auxiliaries are do, does, don't, etc.

The task

- We wrote grammars for each language, and generated sentences using that grammar

```
├── no_agreement_grammars
│   ├── intrans
│   │   ├── no_rc
│   │   │   ├── no_pp
│   │   │   └── pp_on_subject
│   │   └── rc_on_subject
│   │       └── no_pp
│   └── trans
│       ├── no_rc
│       │   ├── no_pp
│       │   ├── pp_on_both
│       │   ├── pp_on_object
│       │   └── pp_on_subject
│       ├── rc_on_both
│       │   └── no_pp
│       ├── rc_on_object
│       │   ├── no_pp
│       │   └── pp_on_subject
│       └── rc_on_subject
│           ├── no_pp
│           └── pp_on_object
│
```

```
├── agreement_grammars
│   ├── intrans
│   │   ├── no_rc
│   │   │   ├── no_pp
│   │   │   └── pp_on_subject
│   │   └── rc_on_subject
│   │       └── no_pp
│   └── trans
│       ├── no_rc
│       │   ├── no_pp
│       │   ├── pp_on_both
│       │   ├── pp_on_object
│       │   └── pp_on_subject
│       ├── rc_on_both
│       │   └── no_pp
│       ├── rc_on_object
│       │   ├── no_pp
│       │   └── pp_on_subject
│       └── rc_on_subject
│           ├── no_pp
│           └── pp_on_object
│
```

Generating sentences

- Each grammar was a CFG, without recursion
- The number of sentences we used is only a fraction of what the grammar can generate
- We changed the vocabulary randomly, and sampled subsets of the vocabulary to generate sentences faster

Neural Networks: What?

- Network of small computing units
- Takes a vector of input values and produces one output vector
- Learns based on the given examples and automatically infers distributions.

Recurrent Neural Network

- Networks with loops in them, thus allowing information to persist

- Hence with RNN's we can preserve the context.

seq2seq Networks

- A RNN which converts sentences from one domain to another
- An RNN encoding layer
  - We get the internal states for this sentence *i.e.* the context
- A RNN decoding layer
  - Using Teacher Forcing we train the network to predict the next character given the previous characters

Implementation

1. Encode the input sentence to state vectors.
2. Start with the target size of 1 i.e the Start of sentence token.
3. Feed the state vectors and 1-word target sequence to the decoder to produce predictions for the next word.
4. Sample the next word using these predictions.
5. Append the next word to the target sequence.
6. Repeat steps 3-5 till you get the End of Sentence or you reach the max length of the sentence.

Gated Recurrent Unit

- It is a variation of an RNN
- Does not suffer from the *"Vanishing Gradient"* problem of vanilla RNNs
- Two gates – Update Gate and Reset Gate
- Update gate: A vector which decides what information to pass to the next stage
- Reset Gate: A vector which decides what information to forget

GRU with Attention

- Lots of information
  - The required information may be lost
- Attention solves this problem
  - A vector which tells you which part of the information to focus on

Experiment

	IDENT	QUEST
RC on Subject	Train, Test	Generalisation
RC on Object	Train, Test	Train, Test
No RC	Train, Test	Train, Test

Experiment

- Randomly initialise hidden layers
- Train on the training set
- Measure accuracy on test and generalisation set

Experiment

	No-agreement	Agreement
Training	118k	117k
Test	9k	9k
Generalisation	10k	10k

Results

Test accuracy

- We evaluated each model, and measured the number of sentences that match exactly with the correct output

No-agreement

Model #	Accuracy (%)
1	84.5
2	86.9
3	91.3
4	84.7

Agreement

Model #	Accuracy
1	87.0
2	95.4
3	98.8
4	83.8

Test accuracy

- But, if we relax the constraint that the words have to match, to requiring that the POS matches, we see the accuracy improves

No-agreement

Model #	Accuracy (%)
1	93.2
2	94.7
3	97.5
4	92.7

Agreement

Model #	Accuracy
1	89.6
2	98.7
3	99.9
4	87.4

Generalisation accuracy

- We see that the model doesn't perform very well on the generalisation tasks

No-agreement

Model #	Word Match Accuracy	POS Match Accuracy
1	0.02	3.6
2	0.08	1.9
3	0	3.1
4	0.13	6.0

Agreement

Model #	Word Match Accuracy	POS Match Accuracy
1	0.02	17.5
2	0	14.0
3	1.45	10.4
4	0	11.4

But that's not all...

- The amount of data is pretty small
- The generalisation set contains kinds of sentences that the model has never seen before
- The model may have acquired grammatical structure even if it can't reproduce the entire sentence correctly

- The real question is whether the model has learnt the hierarchical rule
- To see how it does, we just need to know which auxiliary it moves to the front

Generalisation accuracy

No-agreement

Model #	First Word Accuracy
1	1.8
2	1.6
3	0.7
4	3.3

Agreement

Model #	First Word Accuracy
1	1.8
2	0.01
3	11.9
4	0.01

Generalisation accuracy

- We still note that our models don't perform anywhere near as well as those of McCoy et. al.
- In fact, we see that the model consistently learns the wrong rule by examining some of the predictions it makes

Examples

$=$ Doesn't her monkey who does live call the elephants?

$>$ Does her monkey who doesn't call the elephants?

$=$ Does our elephant who doesn't giggle impress our dogs?

$>$ Doesn't our elephant who does giggle does impress our dogs?

$=$ Will the seal who can live impress her seals below her dogs?

$>$ Can the seal who will live will impress her seals?

$=$ Would your dogs that the yaks could read irritate the dogs?

$>$ Could your dogs that the dogs could would irritate the dogs?

Explanations

- There could be many reasons for the differences between out results and those of McCoy et al

  - We're using a slightly different vocabulary, and generating sentences using a different mechanism

  - We did not hit upon good initialisations

  - Our model isn't exactly the same as theirs (we don't have their code, so we can't tell)

Future Work

McCoy et. al.

- Test the final encoder states for different features using a linear classifier
    - This allows us to understand what the network is actually learning
- Analyse errors in our model and compare with findings of *Crain and Nakayama*

...and beyond

- Try this on other, more complex, hierarchical tasks
- Try this on real world, not generated data
- Try this on other languages

Conclusions

Apparently not

- We were not able to get the same kind of accuracies that the original paper got
- This could be due to a multitude of reasons, including sheer bad luck
- Our models consistently seemed to learn the wrong rules, like moving the first auxiliary to the front

...but that's not all

- McCoy et al found that only one architecture – GRU with attention – was able to perform well, but we didn't observe that
- Maybe there are some architectural improvements that will allow the model to perform better on the task

Questions?

Find our code, and more information at

https://github.com/saujasv/hierarchical-rnn