Natural Language Generation: Traditional Approaches and Research Directions

Lecture 3: Deep Learning Approaches

Pablo Ariel Duboue, PhD.

Textualization Software Ltd.
Vancouver, Canada

17th Estonian Summer School on Computer and Systems Science

Outline

Review from Lecture 1

Subtasks:

- Content Planning.
  - Content Selection.
  - Document Structuring.
- Sentence Planning.
  - Aggregation.
  - Referring Expression Generation.
  - Lexicalization.
- Surface Realization.
  - Linearization.

Review from Lecture 2

- Concepts.
  - Language Models.
  - Generate-and-Rank.
- Evaluation.
  - BLEU.
  - ROUGE.

Today

- Deep Learning basics.
- Recurrent-neural network generation.

Outline

1. Deep Learning
   - Basic Concepts
   - Recurrent Neural Networks
2. Deep Learning for NLG
   - Methods
   - Advanced Topics
3. Examples
   - Selected Papers
   - Keywords 4 Bytecodes

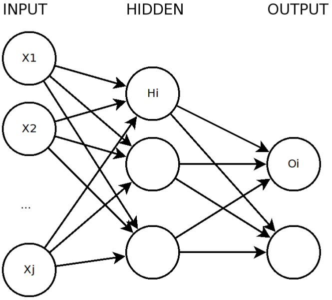
What is Deep Learning

- Deep Learning is a series of techniques enabling the training of deep (multiple hidden layers) neural networks with backpropagation.
  - Better activation functions: rectified linear units (ReLUs).
  - Better initialization of weights: Xavier initialization.
  - Better training scheduling: mini-batches.
  - Better objective functions: SoftMax layers.

Artificial Neural Networks

- Artificial Neural Networks are biologically inspired models of computation started in early 1940s.
  - They precede the von Neumann Architecture by two years.

- Each neuron computes the same function:
  - $h_i = f\left(\sum_j H_{ij}x_j + B_i\right)$
- The output is passed to the next layers.
- Architectures with loops are also possible.
  - We will see them shortly, as they are used in NLG.



Training ANNs: Backpropagation

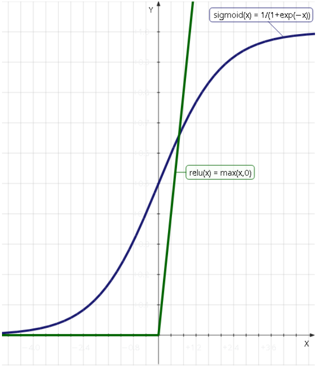
- Given
  - a trainset of inputs and expected outputs,
  - an error metric $\mathcal{L}$,
  - a given configuration of weights H.
- The network can be evaluated and its gradients with respect to the errors computed.
- The gradients can then be employed to change the weights:

$$H_{ij}^{t+1} = H_{ij}^{t} - \alpha \frac{\partial \mathcal{L}}{\partial H_{ij}}$$

- This technique, gradient decent, trains the network.
- In reality, not only fully connected feed-forward networks can be trained by this technique.
  - If the neuronal graph is derivable, its weights can be trained.

ReLUs

- In the past, neural networks used a sigmoid function:
  - $\sigma(z) = \frac{1}{1+e^{-z}}$
  - Better biological analog.
  - More computationally expensive.
- Instead of using a sigmoid as done in traditional neural networks, use a linear unit with a non-linearity at zero:
  - $\mathrm{relu}\,(x) = \max\,(x, 0)$
  - Faster to compute, particularly on GPUs.
  - Better overall.



Xavier Initialization

- From Glorot et al. (2010) JMLR paper, traditional weight initialization used in neural networks before deep learning many time exhibit higher variability than the data variability.
    - The initialization procedure dominated the training results.
    - Therefore, many repetitions were needed, many did not succeeded training the network.
- Instead, focus on having initial weights with a constant variance, given the training data.
    - That is achieving by setting the weights from a Gaussian distribution with zero mean, and a variance equal to $^1/_N$ where N is the number of input neurons.

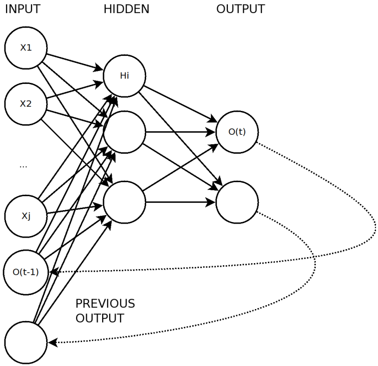
Other Improvements

- SoftMax Layers:
  - Have a differentiable layer that can learn a pattern of activation where only one neuron is active (the maximum) and the rest are as close to zero as possible.
  - A differentiable (thus "soft") version of the max function.
  - $O_j = \sigma(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{N} e^{x_k}}$, assuming there are N output neurons, x_j is the weighted sum of the outputs of the hidden layer.
- Minibatches: perform the weight updates over small number of training instances.
  - Started as a practical need for using GPUs.
  - Have certain implication regarding overfitting and getting stuck in local minima.

Outline

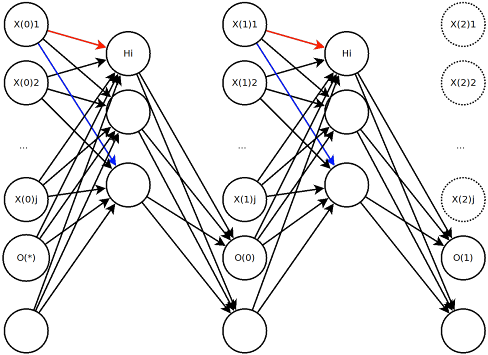
RNNs

- Recurrent Neural Networks
  - Feed back the output into the network



Backpropagation Through Time

- RNNs can be trained by “unrolling” the network
- Problem: Vanishing gradients
  - Inputs too far away in time from the output will impact the learning process too little



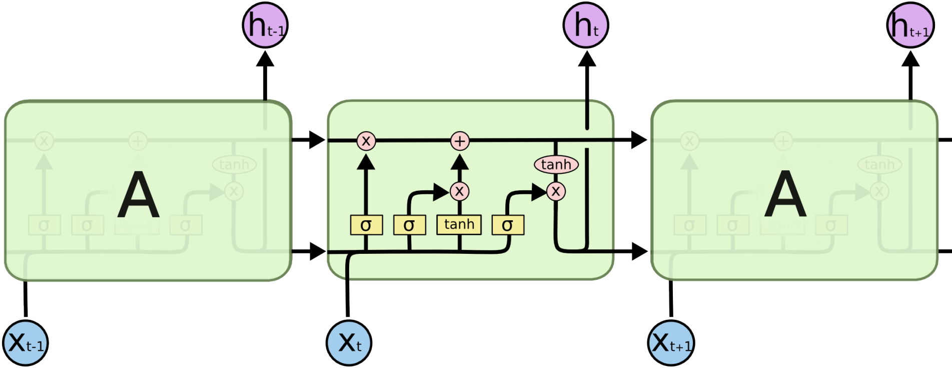
Computing Paradigm

- Training in backpropagation-through-time is to compute update for the weights and apply it only once.
  - Usually as the average of the updates.
  - All these updates are tied to the same weight.
- Gradient descent, weight-tying and differentiable updates define a computing paradigm.
  - Any graph of units such that the path from input to output is differentiable can thus be trained.
  - This graph is sometimes made explicit in some NN frameworks, like TensorFlow.
- Programming in this new paradigm takes the form of master equations or flow diagrams.

Deep Learning RNNs

- To escape the problem of the vanishing gradients, it is possible to confine the recurrence to a complex set of neurons that hold a small local memory.
  - This memory is thus passed as output to be reused next time.
  - The set of neurons can operate in this memory:
    - by accessing parts of it (by multiplication with a soft bit mask).
    - by deleting it (by subtracting after applying a soft bit mask).
    - by updating it (by addition after applying a soft bit mask).
- Based on this idea, two such units are in use:
  - Gated Recurrent Units (GRUs), simpler.
  - Long-term Short-Term Memories (LSTM), more complex.

LSTM



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Outline

Methods

- Language Embeddings.
- Encoder-Decoder Architectures.
  - seq2seq tasks (MT).
- Attention Models.
- Pointer networks.
- Encoding Structured Knowledge.

Language Embeddings

- NNs operate in this "soft" world of differentiable processes.
  - Good for sensor data like images.
  - Language is not very easy to represent for them.
- Representing words with "one hot encoding":
  - Define an input vector of the size of the vocabulary.
  - Set one neuron to 1.0 (the one for that word), the rest to 0.0.
    - If the vocabulary is 60,000 words, is a vector of size 60,000 with 59,999 zeros.
- An alternative is to use a dimensionality reduction technique using embeddings:
  - Embedding: any function mapping a (usually large) set of discrete objects into vectors of a given dimension.
  - Relationships between vectors are expected to follow the discrete objects themselves.
    - For words, we expect that "related" words (in the semantic sense) will have "close" vectors in the Euclidean sense.

Distributional Hypothesis

- Embeddings are related to the **Distributional Hypothesis.**
  - The meaning of a word lies not on the word itself but in the places where the word is used.
  - For example: *"the meaning of a ¿? lies not on the ¿? itself but in the places where the ¿? is used"*
- We can then use large collections of texts ("corpora") to obtain representations for words based on this hypothesis.
  - If two words are used in similar contexts, then they might have a representation that is close to each other.
    - The vector for "dog" might be close to the vector for "cat" (using Euclidean distance).

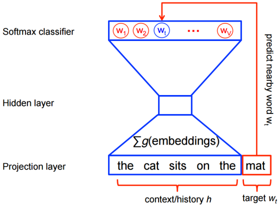
Global Embeddings

- The concept of embeddings is not new.
  - They scaled poorly before.
- An embedding is defined by:
  - A metric in the object space.
  - An optimization problem using that metric.
- For words, we use the prediction of next words using previous words as the metric.
- Global embeddings require that a system is able to predict the correct embedding over all the embeddings in the vocabulary
  - Very time consuming
  - Need to be done in every step of the search for optimal embeddings.

Global Embeddings

https://www.tensorflow.org/tutorials/word2vec

$$P(w_t|h) = \text{softmax}(\text{score}(w_t, h)) = \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word w' in Vocab}} \exp\{\text{score}(w', h)\}}$$

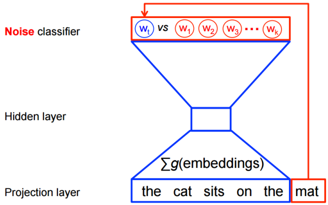


$$\mathcal{L}_{\text{global}} = \log P(w_t|h) = \text{score}(w_t, h) - \log\left(\sum_{\text{Word w' in Vocab}} \exp\{\text{score}(w', h)\}\right)$$

Local Embeddings: Word2Vec

- Instead of using global embeddings, word2vec (Mikolov et al 2013) uses as objective function the score of a trained classifier that discriminates the target word from noise words.

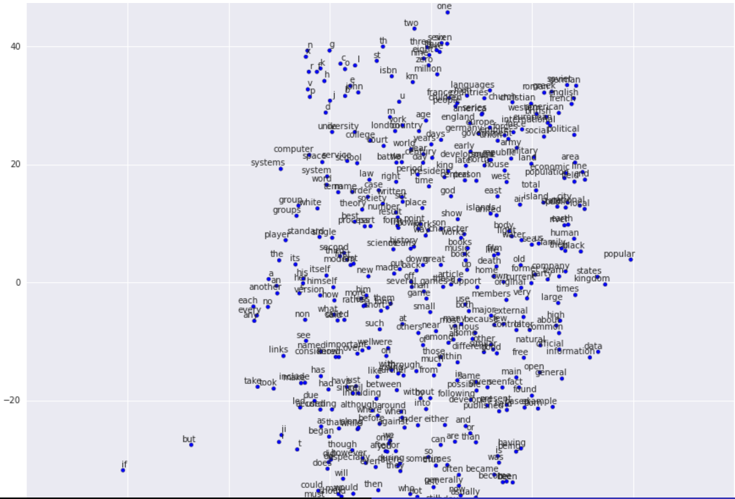
$$\begin{aligned}\mathcal{L}_{\text{local}} &= \log Q_\theta(D=1|w_t,h) + \\ &+k \mathop{\mathbb{E}}_{\tilde{w}\sim P_{\text{noise}}} [\log Q_\theta(D=0|\tilde{w},h)]\end{aligned}$$



- $Q_\theta(D=1|w,h)$, binary logistic regression classifier output of seeing word w in context h using embeddings θ.

Word2Vec: Example

- 96M of text (1.7m tokens) from Wikipedia:



Available Vectors

- Build your own:
  - Word2Vec, a C++ library: https://code.google.com/archive/p/word2vec/
  - fastText, a C++ library: https://github.com/facebookresearch/fastText
  - Gensim, a Python library: https://://radimrehurek.com/gensim/models/word2vec.html
  - Tensorflow, a Python DL environment: https://www.tensorflow.org/tutorials/word2vec
- Get them from the Web:
  - 300-dimensional vectors over 100 billion words Google News dataset: https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing)
  - CommonCrawl and others: https://nlp.stanford.edu/projects/glove/
  - All the rest: http://ahogrammer.com/2017/01/20/the-list-of-pretrained-word-embeddings/

Beyond Word2Vec

- Word2vec has been around by many years now.
- There are plenty of new alternatives.
- GloVe seems to work better in practice.
- Re-computing the embeddings as part of an end-to-end training is also recommended.
- Work has started in ontological embeddings, see RDF2Vec.

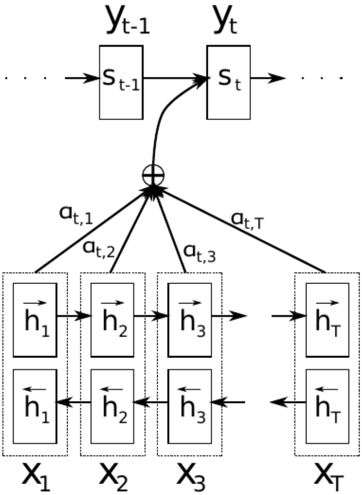
Encoder-Decoder Systems

- An encoder-decoder framework solves sequence-to-sequence (seq2seq) problems by training end-to-end two RNNs:
  - An encoder RNN that consumes the input.
  - A decoder RNN that generates the output.
    - It uses the last hidden layer as extra information for generation.
    - Or all the hidden layers in some systems.
- Popularized by their success in Machine Translation.
  - The encoder RNN will consume a sentence in say, French.
  - Its last hidden layer constitutes the "meaning" of the sentence.
  - The decoder RNN will use that "meaning" to generate a translation in say, English.
  - The system is trained end-to-end using examples of pairs of translated French/English sentences.
    - Both RNNs learn to work together to have a last hidden layer with enough information for decoding a correct translation.

Attention Models

- Starting the decoding process from the last hidden layer is problematic, as it needs to summarize a lot of information.
- Attention models perform more processing between the encoder and the decoder:
  - It uses the hidden layer of all steps, not just the last.
  - It weights the hidden layers of different steps at different times to feed into the decoder RNNs.
    - This weighting scheme is the “attention” model.
- It obviates the need for explicit, word-by-word source-language/target-language correspondence, as in Statistical Machine Translation.

Attention Models



Outline

Pointer Networks

- A common issue with data-driven systems is how to deal with Out-Of-Vocabulary (OOV) words.
  - Particularly numbers and proper nouns.
- Pointer networks (Vinayls et al, 2015) are an expansion of the encoder/decoder framework to have the ability to copy sections of the input directly into the output.
  - Instead of using attention as a weighted average of the hidden states of the encoder, it replaces attention with probabilistic pointers.
  - The same as attention, these pointers are learned.

Encoding Structured Knowledge

- When generating from data, there is the question of how to encode the input data.
- In the simpler case, it can be fed into the encoder-decoder as triples $\langle \text{entity}, \text{relation}, \text{value} \rangle$
  - The issue remains how to order the triples.
  - Different orderings produce different results.
- An extension of Pointer Networks (presented by Vinayls et al. in "Order Matters: Sequence to sequence for sets", 2015) can be used to learn from sets.

Outline

Wen et al, 2015

Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems, EMNLP.

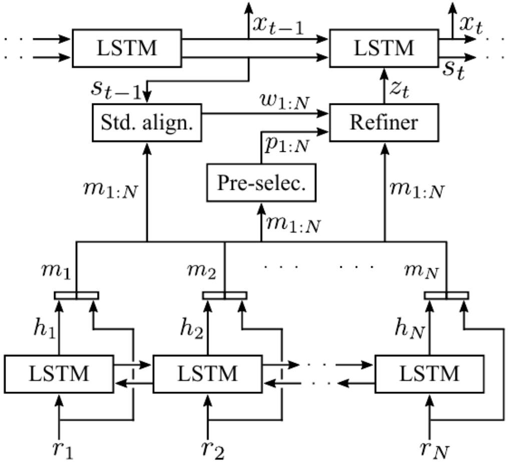
- Propose the Semantically Conditioned LSTM (SC-LSTM) to avoid repetitions in the generated output
  - *Good Eats is a great **British** restaurant that serves **British**.*
  - *Good Eats is a **child friendly** restaurant in the cheap price range. They also allow **kids**.*
- Add another cell to the LSTM that contains a vector (passed from time t to time $t+1$)
  - The DA cell (dialog act).
  - The hope is that it keeps track of what information has been output.
  - It is initialized with all dialog acts one-hot encoded.
  - It is update from the previous output and current input and it affects the output of the LSTM.

Mei et al, 2016

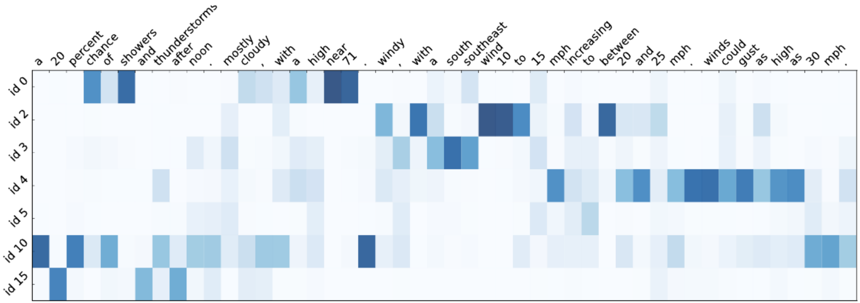
What to talk about and how? Selective generation using LSTMs with coarse-to-fine alignment, NAACL-HLT.

- They perform joint content selection and realization (they call it "selective generation")
    1. Bidirectional LSTM-RNN to encode all records.
    2. Coarse-to-fine to select records (special attention mechanism).
    3. Decoder to generate the text.
- Same domains (WeatherGov and RoboCup) as others.
    - But their system struggles with RoboCup ("knowledge starved") at 1,000 training instances.

Mei et al, 2016: Architecture



Mei et al, 2016: Alignments



Record details:
id-0: temperature(time=06−21, min=52, mean=63, max=71); id-2: windSpeed(time=06−21, min=8, mean=17, max=23);
id-3: windDir(time=06−21, mode=SSE); id-4: gust(time=06−21, min=0, mean=10, max=30);
id-5: skyCover(time=6−21, mode=50−75); id-10: precipChance(time=06−21, min=19, mean=32, max=73);
id-15: thunderChance(time=13−21, mode=SChc)

Mei et al, 2016: Results

Method	F-1	sBLEU
KL12	–	33.70
KL13	–	36.54
ALK10	65.40	38.40
Our model	**73.21**	**61.01**

- KL = Konstas and Lapata (PCFGs)
- ALK = Angeli et al., 2010 (second lecture), appears as 28.8 in the paper).

Mei et al, 2016: Details

- Input set needs to be linearized.
  - Or given in random order.
- Decoder runs from previously generated word, context vector and LSTM hidden state.
- They generate numbers as any other token.
  - Which is a little baffling.

Mei et al, 2016: Embeddings

Word	Nearest neighbor
gusts	gust
clear	sunny
isolated	scattered
southeast	northeast
storms	winds
decreasing	falling

Gardent et al. 2017

The WebNLG Challenge: Generating Text from RDF Data, INLG.

- Evaluation challenge, generating text from DBpedia subsets.
    - Text was written by crowd workers.
- Example:
    - Data:
        - `(JOHN_E_BLAHA BIRTH_DATE 1942_08_26)`
        - `(JOHN_E_BLAHA BIRTH_PLACE SAN_ANTONIO)`
        - `(JOHN_E_BLAHA OCCUPATION FIGHTER_PILOT)`
    - Text:

        John E Blaha, born in San Antonio on 1942-08-26, worked as a fighter pilot.
- 25,298 data-text pairs.

Gardent et al. 2017: Details

- Organizers provided a competitive baseline system using the OpenNMT toolkit.
- 9 submissions from 6 sites.
- Three approaches:
  - Neural-based (5 submissions).
  - Statistical-based (1 submissions).
  - Symbolic (3 submissions).

Gardent et al. 2017: Results

BLEU		
1	MELBOURNE	45.13
2	TILB-SMT	44.28
3–4	PKUWRITER	39.88
3–4	UPF-FORGE	38.65
5–6	TILB-PIPELINE	35.29
5–6	TILB-NMT	34.60
7	BASELINE	33.24
8	ADAPT	31.06
9	UIT-VNU	7.07

TER		
1	MELBOURNE	0.47
2	TILB-SMT	0.53
3–4	PKUWRITER	0.55
3–5	UPF-FORGE	0.55
4–5	TILB-PIPELINE	0.56
6–7	TILB-NMT	0.60
6–7	BASELINE	0.61
8–9	UIT-VNU	0.82
8–9	ADAPT	0.84

METEOR		
1	UPF-FORGE	0.39
2	TILB-SMT	0.38
3	MELBOURNE	0.37
4	TILB-NMT	0.34
5–6	ADAPT	0.31
5–7	PKUWRITER	0.31
6–7	TILB-PIPELINE	0.30
8	BASELINE	0.23
9	UIT-VNU	0.09

- red is neural, gray is statistical, blue is symbolic

Gardent et al. 2017: Results (Seen)

BLEU		
1	ADAPT	60.59
2–3	MELBOURNE	54.52
2–4	TILB-SMT	54.29
3–4	BASELINE	52.39
5	PKUWRITER	51.23
6	TILB-PIPELINE	44.34
7	TILB-NMT	43.28
8	UPF-FORGE	40.88
9	UIT-VNU	19.87

TER		
1	ADAPT	0.37
2	MELBOURNE	0.40
3–4	BASELINE	0.44
3–4	PKUWRITER	0.45
5	TILB-SMT	0.47
6	TILB-PIPELINE	0.48
7	TILB-NMT	0.51
8	UPF-FORGE	0.55
9	UIT-VNU	0.78

METEOR		
1	ADAPT	0.44
2	TILB-SMT	0.42
3–4	MELBOURNE	0.41
3–4	UPF-FORGE	0.40
5–6	TILB-NMT	0.38
5–8	TILB-PIPELINE	0.38
6–8	PKUWRITER	0.37
6–8	BASELINE	0.37
9	UIT-VNU	0.15

- red is neural, gray is statistical, blue is symbolic
- Systems evaluated only on categories provided at training time.

Gardent et al. 2017: Results (Unseen)

BLEU		
1	UPF-FORGE	35.70
2	MELBOURNE	33.27
3	TILB-SMT	29.88
4–5	PKUWRITER	25.36
4–5	TILB-NMT	25.12
6	TILB-PIPELINE	20.65
7	ADAPT	10.53
8	BASELINE	06.13
9	UIT-VNU	0.11

TER		
1	UPF-FORGE	0.55
2	MELBOURNE	0.55
3	TILB-SMT	0.61
4–5	TILB-PIPELINE	0.65
4–5	PKUWRITER	0.67
6	TILB-NMT	0.72
7	BASELINE	0.80
8	UIT-VNU	0.87
9	ADAPT	1.4

METEOR		
1	UPF-FORGE	0.37
2	TILB-SMT	0.33
3	MELBOURNE	0.33
4	TILB-NMT	0.31
5	PKUWRITER	0.24
6	TILB-PIPELINE	0.21
7	ADAPT	0.19
8	BASELINE	0.07
9	UIT-VNU	0.03

- red is neural, gray is statistical, blue is symbolic
- Systems evaluated on new categories, not used during training.

Lebret et al. 2016

Neural Text Generation from Structured Data with Application to the Biography Domain, EMNLP

- Introduces the WIKIBIO corpus.
  - Freely available.
  - 700k data-text pairs, first sentence of biographical information plus infobox, from Wikipedia.
- Conditional Neural Model for generation.
- Need to handle very large vocabulary.
  - Fixed vocabulary plus copy actions.

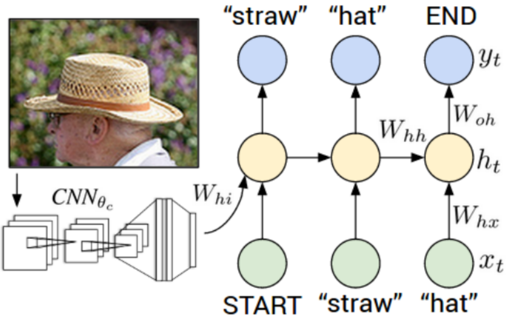
Tanti et al 2017

Where to put the Image in an Image Caption Generator, Natural Language Engineering

- Two ways to add the image data:
  - The image features can go to the input layer of the language model.
    - Conditioning it by injecting the features.
  - They can go to a layer after it.
    - Conditioning it by merging the features.
- They show that late binding is superior to early binding on a number of metrics.
- The multimodal representation should be delayed to a later stage.
  - Value of a "multimodal" layer.
  - RNNs do not *generate* text, but encode it to be used by a later layer.

Tanti et al 2017: Details

- NLG using RNNs: predict next word based on “history”.
  - Caption generation: “history” + image features.

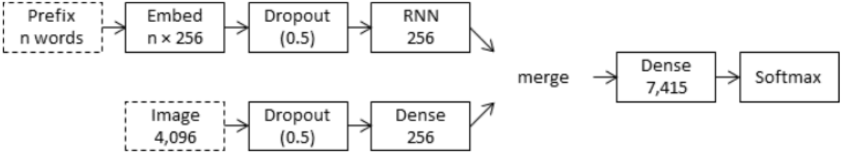


from Karpathy and Li, 2015

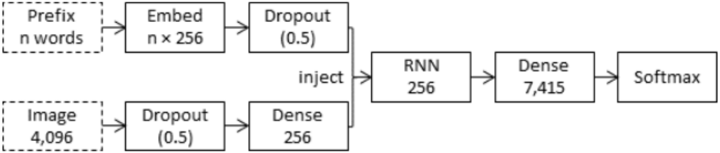
Tanti et al 2017: Details

- The key question is whether the image influences the RNN.
  - If yes then it is injected.
  - If not, it is merged.
- Deeper question about how language should be grounded on vision.
  - In the merging case, the RNN does not "generate" text.
    - just encodes the prefix for use by later (generating) stages.
- Image-injected RNNs generated words one-at-a-time for a given time-step (continuous view).
- Merging approach is more encoder-decoder (discontinuous view).
  - the state of the RNN is re-initialized at every time step.
  - the generated-prefix-so-far is feed back into the network.

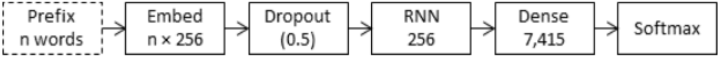
Tanti et al 2017: Networks



(a) The merge architecture.



(b) The inject architecture.



(c) The imageless language model architecture.

Tanti et al 2017: Results

	CIDEr	METEOR	ROUGE-L
merge-add-srnn	**0.337 (0.009)**	0.157 (0.002)	0.397 (0.003)
merge-mult-lstm	0.337 (0.004)	**0.158 (0.002)**	**0.399 (0.004)**
inject-post-srnn	0.333 (0.008)	0.156 (0.002)	0.392 (0.003)
merge-add-lstm	0.331 (0.011)	0.156 (0.002)	0.394 (0.002)
mao	0.325 (0.012)	0.156 (0.003)	0.395 (0.006)
merge-concat-lstm	0.320 (0.013)	0.155 (0.001)	0.393 (0.003)
inject-post-lstm	0.320 (0.007)	0.152 (0.001)	0.386 (0.003)
merge-mult-srnn	0.319 (0.009)	0.155 (0.001)	0.393 (0.004)
inject-par-lstm	0.318 (0.006)	0.152 (0.001)	0.388 (0.003)
merge-concat-srnn	0.316 (0.006)	0.152 (0.001)	0.388 (0.001)
inject-par-srnn	0.297 (0.007)	0.148 (0.001)	0.381 (0.004)
inject-pre-lstm	0.291 (0.009)	0.150 (0.004)	0.383 (0.008)
vinyals	0.290 (0.005)	0.148 (0.002)	0.379 (0.002)
inject-init-lstm	0.281 (0.003)	0.146 (0.000)	0.379 (0.003)
inject-init-srnn	0.256 (0.007)	0.147 (0.001)	0.381 (0.003)
inject-pre-srnn	0.238 (0.005)	0.144 (0.003)	0.371 (0.008)
langmodel-srnn	0.085 (0.001)	0.097 (0.011)	0.260 (0.027)
langmodel-lstm	0.070 (0.010)	0.090 (0.001)	0.260 (0.028)

Outline

Reverse Engineering

- From Wikipedia

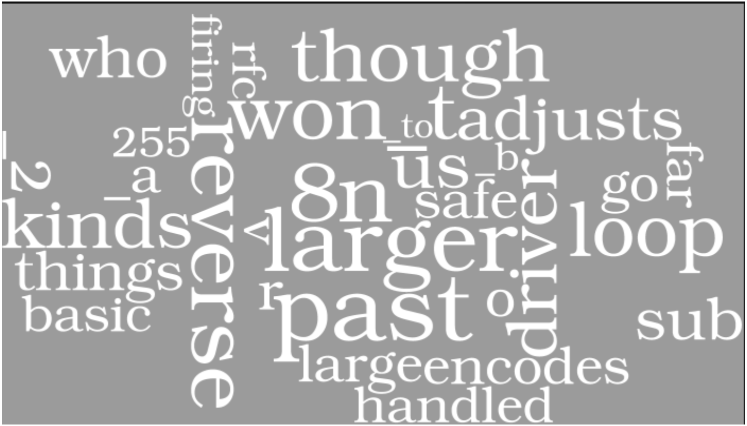
  Reverse engineering is the process of discovering the technological principles of a device, object, or system through analysis of its structure, function, and operation. (...) The same techniques are subsequently being researched for application to legacy software systems (...) ***to replace incorrect, incomplete, or otherwise unavailable documentation***.

- REcon: the premier reverse engineering conference, held yearly at Montreal and Belgium.

Reverse Engineering: Example

```
private final int c(int) {
    0 aload_0
    1 getfield org.jpc.emulator.f.v
    4 invokeinterface org.jpc.support.j.e()
    9 aload_0
    10 getfield org.jpc.emulator.f.i
    13 invokevirtual org.jpc.emulator.motherboard.q.e()
    16 aload_0
    17 getfield org.jpc.emulator.f.j
    20 invokevirtual org.jpc.emulator.motherboard.q.e()
    23 iconst_0
    24 istore_2
    25 iload_1
    26 ifle 128
    29 aload_0
    30 getfield org.jpc.emulator.f.b
    33 invokevirtual org.jpc.emulator.processor.t.w()
```

Keywords for org.jpc.emulator.motherboard.q.e()



Reverse Engineering: Example

```
private final int c(int) {
    0 aload_0
    1 getfield org.jpc.emulator.f.v
    4 invokeinterface org.jpc.support.j.e()
    9 aload_0
    10 getfield org.jpc.emulator.f.i
    13 invokevirtual org.jpc.emulator.motherboard.q.e()
    16 aload_0
    17 getfield org.jpc.emulator.f.j
    20 invokevirtual org.jpc.emulator.motherboard.q.e()
    23 iconst_0
    24 istore_2
    25 iload_1
    26 ifle 128
    29 aload_0
    30 getfield org.jpc.emulator.f.b
    33 invokevirtual org.jpc.emulator.processor.t.w()
```

Keywords for org.jpc.emulator.processor.t.w()

Java Bytecodes

- JVM is a stack machine.
- The set of opcodes (~200) is small to simplify porting to new architectures.
- The opcodes fall into six categories:
  - Load/store (e.g. aaload, bastore).
  - Arithmetic/logic (e.g. iadd, fcmpg).
  - Type conversion (e.g. i2b, f2d).
  - Object construction and manipulation (new, putfield).
  - Operand stack manipulation (e.g. swap, dup2_x1).
  - Control flow (e.g. if_icmpgt,goto).
  - Method invocation and return (e.g. invokedynamic, lreturn).

Keywords 4 Bytecodes

- The keywords4bytecodes started as a collaboration with the people in Les Laboratoires Foulab, a hackerspace in Montreal, Canada. `http://keywords4bytecodes.org`
- Motivation (Schrittwieser, 2016):

  > *Identifier names are often critical to human understanding of a program but cannot be fully restored with the help of automated code analysis techniques.*

- Machine Learning for Natural Language Generation
  - Finding good semantic representations “in the wild” is rare.
    - Detailed semantic representations vs. natural language.
    - Similarities with binary code and associated text.
  - Reverse Engineering practitioners could tolerate noisy text.

K4B: Data

Data	# classes	# methods	# instructions
Apache (dev-train)	67,217	574,620	11,027,500
Eclipse (dev-test)	93,865	721,153	13,352,704
Apache+Eclipse (train)	161,082	1,295,773	24,380,204
Rest (test)	519,541	4,318,079	89,353,021
Rest sample (obf)	111,562	1,032,290	23,974,818
TOTAL	680,623	5,613,852	113,733,225

- Using the Debian archive
  - `apt-file search --package-only .jar`
    - 1,400+ packages
  - `dpkg-query -p` *package name*
    - Look for `Source` field
  - `dpkg-source -x` *source .dsc*
    - Search for Java source files.
  - `dpkg -x` *binary .deb*
    - Search for jars, disassemble the methods.

K4B: Results using Random Forests

To appear, Duboue, 2018, *Deobfuscating Name Scrambling as a Natural Language Generation Task*, ASAI

Token	Count	(%)	Baseline	F1	Prec.	Rec
OTHER	2,449,084	(56.7)	0.2835	0.74	0.70	0.78
WRAPPER	544,621	(12.6)	0.0630	0.37	0.50	0.29
add	66,778	(1.5)	0.0075	0.22	0.48	0.14
clone	7,968	(0.1)	0.0005	0.35	0.72	0.23
compare	9,517	(0.2)	0.0010	0.39	0.70	0.27
contains	16,811	(0.3)	0.0015	0.08	0.50	0.04
equals	17,749	(0.4)	0.0020	0.72	0.83	0.64
get	706,435	(16.3)	0.0815	0.54	0.47	0.64
hash	16,479	(0.3)	0.0015	0.49	0.81	0.35
is	117,638	(2.7)	0.0135	0.39	0.44	0.36
jj	12,431	(0.2)	0.0010	0.97	0.97	0.98
next	15,086	(0.3)	0.0015	0.18	0.56	0.10
set	249,094	(5.7)	0.0285	0.57	0.76	0.45
to	63,070	(1.4)	0.0070	0.35	0.61	0.24
value	14,247	(0.3)	0.0015	0.67	0.90	0.53

K4B: Deep Learning

- Encoder/decoder Architecture
  - Sequence of bytecode types as input
  - Sequence of letters as output

Source Code → Compiled

```
String getTitle() {
  if(this.title == null)
    return ";
  else
    return this.title; }
```

→

```
getTitle:
(opcode 25) aload_0
(opcode 180) getfield title
(opcode 199) ifnonnull 10
(opcode 18) ldc ""
(opcode 176) areturn
(opcode 25) aload_0
(opcode 180) getfield title
(opcode 176) areturn
```

K4B: Obfuscation

Compiled		Obfuscated
getTitle:		a:
(opcode 25) aload_0		(opcode 25) aload_0
(opcode 180) getfield title		(opcode 180) getfield b
(opcode 199) ifnonnull 10		(opcode 199) ifnonnull 10
(opcode 18) ldc ""	→	(opcode 18) ldc ""
(opcode 176) areturn		(opcode 176) areturn
(opcode 25) aload_0		(opcode 25) aload_0
(opcode 180) getfield title		(opcode 180) getfield b
(opcode 176) areturn		(opcode 176) areturn

K4B: Deep Learning

- Encoder/decoder Architecture
  - Sequence of bytecode types as input
  - Sequence of letters as output

Obfuscated		OpenNMT
a: (opcode 25) aload_0 (opcode 180) getfield b (opcode 199) ifnonnull 10 (opcode 18) ldc "" (opcode 176) areturn (opcode 25) aload_0 (opcode 180) getfield b (opcode 176) areturn	→	Op25 Op180 Op199 Op18 Op176 Op25 Op180 Op176 ↓ get Ch103 Ch101 Ch116 Ch84 Ch105 Ch116 Ch108 Ch101

Not Covered

- Different styles and personality.
- Creative and entertaining text.

Summary

- NLG at its core deals with representing and handling large number of principled decisions.
  - A process of enriching the input representation culminating into a full fledged text.
- Growing field with plenty of interesting problems to address.
  - Evaluation is still an issue.
- Data-driven approaches focus on domains with available data.
  - These methods haven't found their way yet to commercial counterparts.
- Outlook:
  - Commercial transfer of data-driven methods.
  - Intent beyond information.

For Further Reading

Barnard, K.
Computational Methods for Integrating Vision and Language.
Morgan and Claypool Publishers, 2016.

Tsung-Hsien Wen.
Tutorial: Deep Learning and NLG.
INLG, 2016. http://www.macs.hw.ac.uk/InteractionLab/INLG2016/docs/DL4NLG_final.pptx