Deep Learning
Recurrent Networks:
Modelling Language
Sequence-to-Sequence models

Spring 2024
Attendance: @1389
Sequence-to-sequence modelling

• Problem:
  – A sequence $X_1 \ldots X_N$ goes in
  – A different sequence $Y_1 \ldots Y_M$ comes out

• E.g.
  – Speech recognition: Speech goes in, a word sequence comes out
    • Alternately output may be phoneme or character sequence
  – Machine translation: Word sequence goes in, word sequence comes out
  – Dialog : User statement goes in, system response comes out
  – Question answering : Question comes in, answer goes out

• In general $N \neq M$
  – No synchrony between $X$ and $Y$. 
Sequence to sequence

- Sequence goes in, sequence comes out
- No notion of “time synchrony” between input and output
  - May even not even maintain order of symbols
    - E.g. “I ate an apple” → “Ich habe einen apfel gegessen”
  - Or even seem related to the input
    - E.g. “My screen is blank” → “Please check if your computer is plugged in.”
Sequence to sequence

- Sequence goes in, sequence comes out
- No notion of “time synchrony” between input and output
  - May even not even maintain order of symbols
    - E.g. “I ate an apple” $\rightarrow$ “Ich habe einen apfel gegessen”
  - Or even seem related to the input
    - E.g. “My screen is blank” $\rightarrow$ “Can you check if your computer is plugged in?”
Sequence to sequence

- Sequence goes in, sequence comes out
- No notion of “time synchrony” between input and output
  - May even not even maintain order of symbols
    - E.g. “I ate an apple” → “Ich habe einen apfel gegessen”
  - Or even seem related to the input
    - E.g. “My screen is blank” → “Can you check if your computer is plugged in?”
A different kind of problem

• Input and output may not have correspondence...
But first – a brief detour...
Which open source project?

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    rw->name = "Getjbbregs";
    bprm_self_clear1(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac) | PFMR_CLOBATHINC_SECON]
    return segtable;
}
```
Related math. What is it talking about?

Proof. Omitted.

**Lemma 0.1.** Let $\mathcal{C}$ be a set of the construction.
Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

\[ \mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L}) \]

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\text{etale}}$ we have

\[ \mathcal{O}_X(\mathcal{F}) = \{ \text{morph}_{\text{etale}} \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F}) \} \]

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of $\mathcal{O}$-modules.

**Lemma 0.2.** This is an integer $\mathcal{Z}$ is injective.

Proof. See Spaces, Lemma 22.

**Lemma 0.3.** Let $S$ be a scheme. Let $X$ be a scheme and $Y$ is an affine open covering. Let $\mathcal{U} \subseteq X$ be a canonical and locally of finite type. Let $X$ be a scheme. Let $X$ be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let $X$ be a scheme. Let $X$ be a scheme covering. Let

\[ b : X \rightarrow Y' \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X. \]

be a morphism of algebraic spaces over $S$ and $Y$.

Proof. Let $X$ be a nonzero scheme of $X$. Let $X$ be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

1. $\mathcal{F}$ is an algebraic space over $S$.
2. If $X$ is an affine open covering.

Consider a common structure on $X$ and $X$ the functor $\mathcal{O}_X(U)$ which is locally of finite type.

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

\[ \begin{array}{c}
\Spec(K_0) \\
\downarrow \text{Mor}_{\text{etale}} \\
\downarrow \\
\text{d}(\mathcal{O}_{X_{\text{etale}}}, \mathcal{G})
\end{array} \]

is a limit. Then $\mathcal{G}$ is a finite type and assume $S$ is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_*$. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

Proof. We have see that $X = \Spec(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of $X$ is an open neighbourhood of $U$.

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemma 22. A reduced above we conclude that $U$ is an open covering of $C$. The functor $\mathcal{F}$ is a “field

\[ \mathcal{O}_{X,A} \rightarrow \mathcal{F} \rightarrow \mathcal{-1(\mathcal{O}_{X_{\text{etale}}})} \rightarrow \mathcal{O}_{X_2}(\mathcal{O}_{X_1}) \]

is an isomorphism of covering of $\mathcal{O}_X$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that $X$ is an isomorphism.

The property $\mathcal{F}$ is a disjoint union of Proposition 22 and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over $S$.

If $\mathcal{F}$ is a scheme theoretic image points.

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_1}$ is a closed immersion, see Lemma 22. This is a sequence of $\mathcal{F}$ is a similar morphism.
Naturalism and decision for the majority of Arab countries' capitalise was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Imminences]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]
The unreasonable effectiveness of recurrent neural networks..

• All previous examples were *generated* blindly by a *recurrent* neural network..
  – With simple architectures

• http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Modern text generation is a lot more sophisticated than that

• ChatGPT...
Brief detour: Language models

• Modelling language using recurrent nets

• More generally language models and embeddings..
Language Models

• LMs model the probability distribution of token sequences in the language
  – Word sequences, if words are the tokens

• Can be used to
  – Compute the probability of a given token sequence
  – Generate sequences from the distribution of the language
Language Models

\[ P(w_1 w_2 w_3 w_4 \ldots) = P(w_1) P(w_2|w_1) \]
\[ P(w_3|w_1 w_2) P(w_4|w_1 w_2 w_3) \ldots \]

• The actual target is to model the probabilities of entire word sequences
• However, we typically use Bayes rule to compute this incrementally
  – Language models generally perform *next symbol prediction*
  – Always predicting the next symbol, given all previous symbols
• However, never forget, they *actually* model the probability of entire sequences
  – Sentences, paragraphs, books
  – They model *language*
Language modelling through next-word prediction using RNNs

Four score and seven years ???

ABRAHAMLINCOL??

• Problem: Given a sequence of words (or characters) predict the next one
Language modelling: Representing words

• Represent words as one-hot vectors
  – Pre-specify a vocabulary of N words in fixed (e.g. lexical) order
    • *E.g.* [ A  AARDVARK  AARON  ABACK  ABACUS... ZZYP]
  – Represent each word by an N-dimensional vector with N-1 zeros
    and a single 1 (in the position of the word in the ordered list of
    words)
    • *E.g.* “AARDVARK” → [0 1 0 0 0 ...]
    • *E.g.* “AARON” → [0 0 1 0 0 0 ...]

• Characters can be similarly represented
  – English will require about 100 characters, to include both cases,
    special characters such as commas, hyphens, apostrophes, etc.,
    and the space character
Predicting words

Four score and seven years ???

\[ W_n = f(W_0, ..., W_{n-1}) \]

\( W_0 \)

\( W_1 \)

\( W_{n-1} \)

\( f() \)

\[ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \]

\[ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \]

• Given one-hot representations of \( W_0 \ldots W_{n-1} \), predict \( W_n \)
Predicting words

Four score and seven years ???

\[ W_n = f(W_0, \ldots, W_{n-1}) \]

- Given one-hot representations of \( W_0 \ldots W_{n-1} \), predict \( W_n \)
- **Dimensionality problem**: All inputs \( W_0 \ldots W_{n-1} \) are both very high-dimensional and very sparse
Why one-hot representation

- The one-hot representation makes no assumptions about the relative importance of words
  - All word vectors are the same length
- It makes no assumptions about the relationships between words
  - The distance between every pair of words is the same
The one-hot representation

- The one hot representation uses only $N$ corners of the $2^N$ corners of a unit cube
  - Actual volume of space used = 0
    - $(1, \varepsilon, \delta)$ has no meaning except for $\varepsilon = \delta = 0$
  - Density of points: $\mathcal{O} \left( \frac{N}{\sqrt{N}} \right)$
- This is a tremendously inefficient use of dimensions
Solution to dimensionality problem

- Project the points onto a lower-dimensional subspace
  - Or more generally, a linear transform into a lower-dimensional subspace
  - The volume used is still 0, but density can go up by many orders of magnitude
    - Density of points: $O \left( \frac{N}{\sqrt{M}} \right)$
Solution to dimensionality problem

• Project the points onto a lower-dimensional subspace
  – Or more generally, a linear transform into a lower-dimensional subspace
  – The volume used is still 0, but density can go up by many orders of magnitude
    • Density of points: $O \left( \frac{N}{\sqrt{M}} \right)$
  – If properly learned, the distances between projected points will capture semantic relations between the words
The *Projected* word vectors

Four score and seven years ???

\[ W_n = f(PW_0, PW_2, \ldots, PW_{n-1}) \]

- **Project** the N-dimensional one-hot word vectors into a lower-dimensional space
  - Replace every one-hot vector \( W_i \) by \( PW_i \)
  - \( P \) is an \( M \times N \) matrix
  - \( PW_i \) is now an \( M \)-dimensional vector
  - **Learn** \( P \) using an appropriate objective
    - Distances in the projected space will reflect relationships imposed by the objective
"Projection"

\[ W_n = f(PW_1, PW_2, ..., PW_{n-1}) \]

- \( P \) is a simple linear transform
- A single transform can be implemented as a linear layer with \( M \) outputs
  - The same linear layer applies to all inputs
  - Also viewable as a network with identical subnets (shared parameter network)
Predicting words: The TDNN model

- Predict each word based on the past N words
  - “A neural probabilistic language model”, Bengio et al. 2003
  - Hidden layer has Tanh() activation, output is softmax

- One of the outcomes of learning this model is that we also learn low-dimensional representations $PW$ of words
Alternative models to learn projections

- **Soft bag of words**: Predict word based on words in immediate context
  - Without considering specific position
- **Skip-grams**: Predict adjacent words based on current word

\[
\begin{align*}
W_1 &\quad W_2 &\quad W_3 &\quad W_4 &\quad W_5 &\quad W_6 &\quad W_7 &\quad W_8 &\quad W_9 &\quad W_{10} \\
\hline
\end{align*}
\]

Color indicates shared parameters
• From Mikolov et al., 2013, “Distributed Representations of Words and Phrases and their Compositionality”
Select all that are true

- The distance between any two non-identical one-hot vectors is the same
- Words are represented as one-hot embeddings because these do not impose any a priori assumption about which words are closer than others
- Word embeddings derived from language models are lower-dimensional real-valued representations where the distance between words is a meaningful representation of their closeness
- Low dimensional word embeddings enable you to find representations for words that were not part of your training vocabulary
Select all that are true

- The distance between any two non-identical one-hot vectors is the same
- Words are represented as one-hot embeddings because these do not impose any a priori assumption about which words are closer than others
- Word embeddings derived from language models are lower-dimensional real-valued representations where the distance between words is a meaningful representation of their closeness
- Low dimensional word embeddings enable you to find representations for words that were not part of your training vocabulary
Modelling language

- The hidden units are (one or more layers of) LSTM units
- Trained via backpropagation from a lot of text
  - No explicit labels in the training data: at each time the next word is the label.
Generating Language: Synthesis

- On trained model: Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector

\[ y_t^i = P(W_t = V_i | W_1 \ldots W_{t-1}) \]

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words
Generating Language: Synthesis

- On trained model: Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector
- Draw a word from the distribution
  - And set it as the next word in the series

\[ y_t^i = P(W_t = V_i | W_1 \ldots W_{t-1}) \]

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words.
Generating Language: Synthesis

- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution

\[ y_t^i = P(W_t = V_i | W_1 \ldots W_{t-1}) \]

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words
Generating Language: Synthesis

- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution

\[ y_t^i = P(W_t = V_i | W_1 \ldots W_{t-1}) \]

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words
Generating Language: Synthesis

- Feed the drawn word as the next word in the series
  - And draw the next word from the output probability distribution
- Continue this process until we terminate generation
  - In some cases, e.g. generating programs, there may be a natural termination
Which open source project?

Trained on linux source code

Actually, uses a character-level model (predicts character sequences)
Composing music with RNN

• Feed the drawn word as the next word in the series
  – And draw the next word by sampling from the output probability distribution

• More generally: When do we stop?
A note on beginnings and ends

• A sequence of words by itself does not indicate if it is a complete sentence or not
  
  … four score and eight …
  
  – Unclear if this is the start of a sentence, the end of a sentence, or both (i.e. a complete sentence)

• To make it explicit, we will add two additional symbols (in addition to the words) to the base vocabulary
  
  – \texttt{<sos>} : Indicates start of a sentence
  
  – \texttt{<eos>} : Indicates end of a sentence
A note on beginnings and ends

• Some examples:

  four score and eight
  – This is clearly the middle of sentence

  <sos> four score and eight
  – This is a fragment from the start of a sentence

  four score and eight <eos>
  – This is the end of a sentence

  <sos> four score and eight <eos>
  – This is a full sentence

• In situations where the start of sequence is obvious, the <sos> may not be needed, but <eos> is required to terminate sequences

• Sometimes we will use a single symbol to represent both start and end of sentence, e.g. just <eos>, or even a separate symbol, e.g. <s>
Generating Language: Synthesis

- Feed the drawn word as the next word in the series
  - And draw the next word by sampling from the output probability distribution

- Continue this process until we draw an <eos>
  - Or we decide to terminate generation based on some other criterion
Poll 2 (@1391)

Which of the following is a complete sentence

<sos> Hello World <eos>
<sos> Hello World
Hello World <eos>
Hello World
Poll 2

Which of the following is a complete sentence

<sos> Hello World <eos>

<sos> Hello World

Hello World <eos>

Hello World
Returning to our problem: Sequence to sequence modelling

• Sequence $X_1 \ldots X_N$ goes in, sequence $Y_1 \ldots Y_M$ comes out

• Cases
  – 1 : order correspondence between input and output
    • The nth output corresponds to the nth segment of the input
  – 2 : No correspondence between input and output
    • May even not even maintain order of symbols
      – E.g. “I ate an apple” → “Ich habe einen apfel gegessen”
    • Or may even even seem unrelated to the input
    • E.g. “My screen is blank” → “Please check if your computer is plugged in.”
Returning to our problem: Sequence to sequence modelling

- Sequence $X_1 \ldots X_N$ goes in, sequence $Y_1 \ldots Y_M$ comes out
- Cases
  - 1: order correspondence between input and output
    - The nth output corresponds to the nth segment of the input
  - 2: No correspondence between input and output
    - May even not even maintain order of symbols
      - E.g. “I ate an apple” → “Ich habe einen apfel gegessen”
    - Or may even even seem unrelated to the input
    - E.g. “My screen is blank” → “Please check if your computer is plugged in.”
Returning to our problem: Sequence to sequence modelling

- Sequence $X_1 \ldots X_N$ goes in, sequence $Y_1 \ldots Y_M$ comes out

Cases

1. Order correspondence between input and output
   - The nth output corresponds to the nth segment of the input

2. No correspondence between input and output
   - May even not maintain order of symbols
     - E.g. “I ate an apple” → “Ich habe einen apfel gegessen”
   - Or may even seem unrelated to the input
     - E.g. “My screen is blank” → “Please check if your computer is plugged in.”
Modelling the problem

• *Delayed* sequence to sequence
Modelling the problem

- Delayed sequence to sequence

First process the input and generate a hidden representation for it

• *Delayed* sequence to sequence
# First run the inputs through the network
# Assuming h(-1,1) is available for all layers

for t = 0:T-1  # Including both ends of the index
    \[ [h(t),..] = \text{RNN\_input\_step}(x(t),h(t-1),...) \]

\[ H = h(T-1) \]

"RNN\_input" may be a multi-layer RNN of any kind
Modelling the problem

First process the input and generate a hidden representation for it

Then use it to generate an output

• *Delayed* sequence to sequence
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
for t = 0:T-1  # Including both ends of the index
    [h(t),...] = RNN_input_step(x(t),h(t-1),...)
H = h(T-1)

# Now generate the output y_{out}(1),y_{out}(2),...
t = 0
h_{out}(0) = H
do
dt = t+1
[y(t),h_{out}(t)] = RNN_output_step(h_{out}(t-1))
y_{out}(t) = draw_word_from(y(t))
until y_{out}(t) == <eos>
Pseudocode

# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
for t = 0:T-1  # Including both ends of the index
    [h(t),...] = RNN_input_step(x(t),h(t-1),...)
H = h(T-1)
The output at each time is a probability distribution over symbols.
We draw a word from this distribution.

# Now generate the output y_{out}(1),y_{out}(2),...
t = 0
h_{out}(0) = H
do
    t = t+1
    [y(t),h_{out}(t)] = RNN_output_step(h_{out}(t-1))
    y_{out}(t) = draw_word_from(y(t))
until y_{out}(t) == <eos>
• **Problem**: Each word that is output depends only on current hidden state, and not on previous outputs

First process the input and generate a hidden representation for it.

Then use it to generate an output.
Pseudocode

# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
for t = 0:T-1  # Including both ends of the index
    \[ h(t), \ldots \] = RNN_input_step(x(t), h(t-1), ...)

H = h(T-1)

# Now generate the output y_{out}(1), y_{out}(2), ...

\# Now generate the output y_{out}(1), y_{out}(2), ...

\# Now generate the output y_{out}(1), y_{out}(2), ...

t = 0
h_{out}(0) = H
do
t = t+1
[y(t), h_{out}(t)] = RNN_output_step(h_{out}(t-1))
y_{out}(t) = draw_word_from(y(t))
until y_{out}(t) == \langle eos \rangle

Changing this output at time t does not affect the output at t+1

E.g. If we have drawn “It was a” vs “It was an”, the probability
that the next word is “dark” remains the same (dark must ideally
not follow “an”)

This is because the output at time t does not influence the
computation at t+1

The RNN recursion only considers the hidden
state h(t-1) from the previous time and not the
actual output word y_{out}(t-1)
Modelling the problem

- *Delayed* sequence to sequence
  - Delayed *self-referencing* sequence-to-sequence
The “simple” translation model

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

I ate an apple <eos>
The “simple” translation model

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

- Subsequently a second RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced
The “simple” translation model

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a second RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced
The “simple” translation model

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a second RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

Ich habe

I ate an apple <eos><sos> Ich habe
The “simple” translation model

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a second RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced
The “simple” translation model

• The input sequence feeds into a recurrent structure
• The input sequence is terminated by an explicit <eos> symbol
  – The hidden activation at the <eos> “stores” all information about the sentence

• Subsequently a second RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  – The output at each time becomes the input at the next time
  – Output production continues until an <eos> is produced
The “simple” translation model

Note that drawing a different word here

Would result in a different word being input here, and as a result the output here and subsequent outputs would all change
• We will illustrate with a single hidden layer, but the discussion generalizes to more layers.
Pseudocode

# First run the inputs through the network
# Assuming h(-1,1) is available for all layers

\( t = 0 \)
do
    \[ h(t), \ldots \] = RNN_input_step(x(t), h(t-1), \ldots)
until x(t) == "<eos>"

\( H = h(T-1) \)

# Now generate the output \( y_{out}(1), y_{out}(2), \ldots \)

\( t = 0 \)

\( h_{out}(0) = H \)

# Note: begins with a "start of sentence" symbol
#       \( <sos> \) and \( <eos> \) may be identical

\( y_{out}(0) = <sos> \)
do
    \( t = t+1 \)
    \[ y(t), h_{out}(t) \] = RNN_output_step(h_{out}(t-1), y_{out}(t-1))
    \( y_{out}(t) = \text{draw_word_from}(y(t)) \)
until \( y_{out}(t) == <eos> \)
Pseudocode

# First run the inputs through the network
# Assuming h(-1,1) is available for all layers

\[ t = 0 \]

\[
\text{do}
\]

\[
[h(t),...] = \text{RNN\_input\_step}(x(t), h(t-1), ...)
\]

\[
\text{until } x(t) == "<eos>"
\]

\[ H = h(T-1) \]

# Now generate the output y_{out}(1), y_{out}(2), ...

\[ t = 0 \]

\[ h_{out}(0) = H \]

# Note: begins with a “start of sentence” symbol
# <sos> and <eos> may be identical

\[ y_{out}(0) = <sos> \]

\[
\text{do}
\]

\[ t = t+1 \]

\[
[y(t), h_{out}(t)] = \text{RNN\_output\_step}(h_{out}(t-1), y_{out}(t-1))
\]

\[ y_{out}(t) = \text{draw\_word\_from}(y(t)) \]

\[
\text{until } y_{out}(t) == <eos> \]

**Drawing a different word at t will change the next output since y_{out}(t) is fed back as input**
• The recurrent structure that extracts the hidden representation from the input sequence is the encoder.

• The recurrent structure that utilizes this representation to produce the output sequence is the decoder.

The “simple” translation model

ENCODER

DECODER

Ich habe einen apfel gegessen
I ate an apple
A more detailed look: The one-hot word representations may be compressed via embeddings

- Embeddings will be learned along with the rest of the net
- In the following slides we will not represent the projection matrices
What the network actually produces

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  \[ y^w_k = P(O_k = w | O_{k-1}, ..., O_1, I_1, ..., I_N) \]
  - The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

I ate an apple <eos><sos>
What the network actually produces

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, ..., O_1, I_1, ..., I_N)$
  - The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$
- At each time a word is drawn from the output distribution
- The drawn word is provided as input to the next time

Ich

$y_0^{ich}$
$y_0^{apfel}$
$y_0^{bier}$
$...$
$y_0^{<eos>}$

I ate an apple <sos> <eos>
What the network actually produces

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, ..., O_1, I_1, ..., I_N)$
  - The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$
- At each time a word is \textit{drawn} from the output distribution
- The drawn word is provided as input to the next time
What the network actually produces

At each time $k$ the network actually produces a probability distribution over the output vocabulary

- $y_k^w = P(O_k = w| O_{k-1}, ..., O_1, I_1, ..., I_N)$
  - The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$
- At each time a word is \textit{drawn} from the output distribution
- The drawn word is provided as input to the next time
What the network actually produces

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  
  $y_k^w = P(O_k = w|O_{k-1}, ..., O_1, I_1, ..., I_N)$
  
  - The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$

- At each time a word is drawn from the output distribution

- The drawn word is provided as input to the next time
At each time $k$ the network actually produces a probability distribution over the output vocabulary

$$y_k^w = P(O_k = w|O_{k-1}, ..., O_1, I_1, ..., I_N)$$

- The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$

At each time a word is *drawn* from the output distribution

The drawn word is provided as input to the next time
What the network actually produces

• At each time $k$ the network actually produces a probability distribution over the output vocabulary

  $y_k^w = P(O_k = w | O_{k-1}, ..., O_1, I_1, ..., I_N)$

  The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$

• At each time a word is *drawn* from the output distribution

• The drawn word is provided as input to the next time
What the network actually produces

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w|O_{k-1}, ..., O_1, I_1, ..., I_N)$
  - The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$
- At each time a word is drawn from the output distribution
- The drawn word is provided as input to the next time
What the network actually produces

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  \[ y_k^w = P(O_k = w|O_{k-1}, ..., O_1, I_1, ..., I_N) \]
  - The probability given the entire input sequence $I_1, ..., I_N$ and the partial output sequence $O_1, ..., O_{k-1}$ until $k$
- At each time a word is drawn from the output distribution
- The drawn word is provided as input to the next time

---

Ich habe einen apfel gegessen <eos>

I ate an apple <eos>
Generating an output from the net

- At each time the network produces a probability distribution over words, given the entire input and entire output sequence so far.
- At each time a word is *drawn* from the output distribution.
- The drawn word is provided as input to the next time.
- **The process continues until an <eos> is drawn**
# First run the inputs through the network
# Assuming h(-1,1) is available for all layers

\[ t = 0 \]

\[ \text{do} \]

\[ \quad [h(t), ..] = \text{RNN_input_step}(x(t), h(t-1), ..) \]

\[ \text{until } x(t) == \text{"<eos>"} \]

\[ H = h(T-1) \]

# Now generate the output \( y_{out}(1), y_{out}(2), .. \)

\[ t = 0 \]

\[ h_{out}(0) = H \]

# Note: begins with a “start of sentence” symbol
# \(<\text{sos}>\) and \(<\text{eos}>\) may be identical

\[ y_{out}(0) = <\text{sos}> \]

\[ \text{do} \]

\[ \quad t = t+1 \]

\[ \quad [y(t), h_{out}(t)] = \text{RNN_output_step}(h_{out}(t-1), y_{out}(t-1)) \]

\[ \quad y_{out}(t) = \text{draw_word_from}(y(t)) \]

\[ \text{until } y_{out}(t) == <\text{eos}> \]

What is this magic operation?
The probability of the output

\[
\begin{align*}
P(O_1, \ldots, O_L | I_1, \ldots, I_N) &= P(O_1 | I_1, \ldots, I_N)P(O_2 | O_1, I_1, \ldots, I_N)P(O_3 | O_1, O_2, I_1, \ldots, I_N) \ldots P(O_L | O_1, \ldots, O_{L-1}, I_1, \ldots, I_N) \\
&= y_1^{O_1} y_2^{O_2} \ldots y_L^{O_L}
\end{align*}
\]
The probability of the output

• **The objective of drawing:** Produce the most likely output (that ends in an <eos>)

\[
\text{argmax}_{O_1, \ldots, O_L} P(O_1, \ldots, O_L | W_1^{in}, \ldots, W_N^{in})
\]

\[
= \text{argmax}_{O_1, \ldots, O_L} y_1^{O_1} y_2^{O_2} \ldots y_L^{O_L}
\]
Greedy drawing

Objective:
\[ \text{argmax}_{O_1, ..., O_L} y_1^{O_1} y_2^{O_2} \ldots y_L^{O_L} \]

- So how do we draw words at each time to get the most likely word sequence?
- **Greedy** answer – select the most probable word at each time

I ate an apple <eos> <sos>
# First run the inputs through the network
# Assuming h(-1,1) is available for all layers

t = 0
do
    \[ h(t), \ldots \] = RNN_input_step(x(t), h(t-1), \ldots) 
until \( x(t) = \langle eos \rangle \)

H = h(T-1)

# Now generate the output \( y_{out}(1), y_{out}(2), \ldots \)

t = 0

h_{out}(0) = H

# Note: begins with a “start of sentence” symbol
# \( \langle sos \rangle \) and \( \langle eos \rangle \) may be identical

y_{out}(0) = \langle sos \rangle
do
    t = t+1
    \[ y(t), h_{out}(t) \] = RNN_output_step(h_{out}(t-1), y_{out}(t-1))
    y_{out}(t) = \text{argmax}_i(y(t,i))
until y_{out}(t) = \langle eos \rangle

Select the most likely output at each time
Greedy drawing

**Objective:**

\[
\arg\max_{y_1^{O_1}y_2^{O_2}...y_L^{O_L}} y_1^{o_1}y_2^{o_2}...y_L^{o_L}
\]

- Cannot just pick the most likely symbol at each time
  - That may cause the distribution to be more “confused” at the next time
  - Choosing a different, less likely word could cause the distribution at the next time to be more peaky, resulting in a more likely output overall
Greedy is not good

- Hypothetical example (from English speech recognition: Input is speech, output must be text)
- “Nose” has highest probability at t=2 and is selected
  - The model is very confused at t=3 and assigns low probabilities to many words at the next time
  - Selecting any of these will result in low probability for the entire 3-word sequence
- “Knows” has slightly lower probability than “nose”, but is still high and is selected
  - “he knows” is a reasonable beginning and the model assigns high probabilities to words such as “something”
  - Selecting one of these results in higher overall probability for the 3-word sequence
Greedy is not good

What should we have chosen at $t=2$??

Will selecting “nose” continue to have a bad effect into the distant future?

- Problem: Impossible to know a priori which word leads to the more promising future
  - Should we draw “nose” or “knows”?
  - Effect may not be obvious until several words down the line
  - Or the choice of the wrong word early may cumulatively lead to a poorer overall score over time
Greedy is not good

What should we have chosen at \( t=1 \)??
Choose “the” or “he”? 

- Problem: Impossible to know a priori which word leads to the more promising future
  - Even earlier: Choosing the lower probability “the” instead of “he” at \( T=0 \) may have made a choice of “nose” more reasonable at \( T=1 \).

- In general, making a poor choice at any time commits us to a poor future
  - But we cannot know at that time the choice was poor
Drawing by random sampling

Objective:
\[
\text{argmax } y_1^{o_1} y_1^{o_2} \ldots y_1^{o_L} \\
_{o_1, \ldots, o_L}
\]

• Alternate option: Randomly draw a word at each time according to the output probability distribution.
Pseudocode

# First run the inputs through the network
# Assuming h(-1,l) is available for all layers

\[ t = 0 \]

\[ \text{do} \]

\[ [h(t),..] = \text{RNN\_input\_step}(x(t), h(t-1),...) \]

\[ \text{until } x(t) == \text{"<eos>"} \]

\[ H = h(T-1) \]

# Now generate the output \( y(t) \_\text{out}(1), y(t)\_\text{out}(2),... \)

\[ t = 0 \]

\[ h(t)\_\text{out}(0) = H \]

# Note: begins with a "start of sentence" symbol
# \(<sos>\) and \(<eos>\) may be identical

\[ y(t)\_\text{out}(0) = <sos> \]

\[ \text{do} \]

\[ t = t+1 \]

\[ [y(t), h(t)\_\text{out}(t)] = \text{RNN\_output\_step}(h(t)\_\text{out}(t-1), y(t)\_\text{out}(t-1)) \]

\[ y(t)\_\text{out}(t) = \text{sample}(y(t)) \]

\[ \text{until } y(t)\_\text{out}(t) == \text{<eos>} \]

Randomly sample from the output distribution.
Drawing by random sampling

Objective:
\[
\arg\max_{y_{o_1}, \ldots, y_{o_L}} y_{o_1}^{y_1} y_{o_2}^{y_2} \ldots y_{o_L}^{y_L}
\]

- Alternate option: Randomly draw a word at each time according to the output probability distribution
  - Unfortunately, not guaranteed to give you the most likely output
  - May sometimes give you more likely outputs than greedy drawing though
For greedy decoding, we choose the word that has been assigned the highest probability at each time (T/F)
• True
• False

In decoding through random sampling we randomly choose the next word according to the probability assigned to it by the decoder (T/F)
• True
• False

The procedure used for randomly sampling a word from a distribution has been presented in today’s class (T/F)
• True
• False
For greedy decoding, we choose the word that has been assigned the highest probability at each time (T/F)

- True
- False

In decoding through random sampling we randomly choose the next word according to the probability assigned to it by the decoder (T/F)

- True
- False

The procedure used for randomly sampling a word from a distribution has been presented in today’s class (T/F)

- True
- False
Your choices can get you stuck

Problem: making a poor choice at any time commits us to a poor future

– But we cannot know at that time the choice was poor

Solution: Don’t choose..

What should we have chosen at t=1??
Choose “the” or “he”? 
Optimal Solution: Multiple choices

• Retain all choices and *fork* the network
  – With every possible word as input
**Problem: Multiple choices**

- **Problem:** This will blow up very quickly
  - For an output vocabulary of size $V$, after $T$ output steps we’d have forked out $V^T$ branches
Optimal Solution: Multiple choices

• Retain all choices and *fork* the network
  – With every possible word as input
Solution: Prune

- At each time, retain only the top $K$ scoring forks
Solution: Prune

- At each time, retain only the top K scoring forks

\[ \text{Top}_K P(O_1|I_1, ..., I_N) \]
Solution: Prune

- At each time, retain only the top K scoring forks

$$\text{Top}_K P(O_1|I_1, ..., I_N)$$
Solution: Prune

- At each time, retain only the top K scoring forks

Top$_K$ $P(O_1|I_1, ..., I_N)$
Solution: Prune

- At each time, retain only the top K scoring forks
Solution: Prune

- At each time, retain only the top K scoring forks

Note: based on product

\[ Top_K P(O_2O_1|I_1, ..., I_N) = Top_K P(O_2|O_1, I_1, ..., I_N)P(O_1|I_1, ..., I_N) \]
• **Solution: Prune**

  – At each time, retain only the top K scoring forks
Solution: Prune

- At each time, retain only the top K scoring forks

\[
\text{He knows Nose} = \text{Top}_K P(O_3 | O_1, O_2, I_1, ..., I_N) \times P(O_2 | O_1, I_1, ..., I_N) \times P(O_1 | I_1, ..., I_N)
\]
Solution: Prune

- At each time, retain only the top $K$ scoring forks

$$= \text{Top}_K P(O_3|O_1, O_2, I_1, ..., I_N) \times P(O_2|O_1, I_1, ..., I_N) \times P(O_1|I_1, ..., I_N)$$
Solution: Prune

- At each time, retain only the top K scoring forks
• Terminate
  – When the current most likely path overall ends in <eos>
    • Or continue producing more outputs (each of which terminates in <eos>) to get N-best outputs
• **Terminate**
  
  – Paths cannot continue once the output an `<eos>`
    
    • So paths may be different lengths
      
      – Select the most likely sequence ending in `<eos>` across *all* terminating sequences

Example has $K = 2$
Pseudocode: Beam search

# Assuming encoder output H is available
path = <sos>
beam = {path}
pathscore = [path] = 1
state[path] = h[0]  # Output of encoder

do  # Step forward
    nextbeam = {}
    nextpathscore = []
    nextstate = {}
    for path in beam:
        cfin = path[end]
        hpath = state[path]
        [y, h] = RNN_output_step(hpath, cfin)
        for c in Symbolset
            newpath = path + c
            nextstate[newpath] = h
            nextpathscore[newpath] = pathscore[path]*y[c]
            nextbeam += newpath # Set addition
        end
    end
    beam, pathscore, state, bestpath = prune(nextstate, nextpathscore, nextbeam, bw)
end

until bestpath[end] = <eos>
# Pseudocode: Prune

# Note, there are smarter ways to implement this

```python
function prune (state, score, beam, beamwidth)
    sortedscore = sort(score)
    threshold = sortedscore[beamwidth]

    prunedstate = {}
    prunedscore = []
    prunedbeam = {}

    bestscore = -inf
    bestpath = none
    for path in beam:
        if score[path] > threshold:
            prunedbeam += path  # set addition
            prunedstate[path] = state[path]
            prunedscore[path] = score[path]
            if score[path] > bestscore
                bestscore = score[path]
                bestpath = path
        end
    end
    return prunedbeam, prunedscore, prunedstate, bestpath
```
“Theoretically correct” decoding requires you to evaluate the *entire tree* representing *every possible word sequence* to select the best one (T/F)

- True
- False

Beam search is theoretically correct decoding (T/F)

- True
- False
“Theoretically correct” decoding requires you to evaluate the entire tree representing every possible word sequence to select the best one (T/F)

- True
- False

Beam search is theoretically correct decoding (T/F)

- True
- False
Training the system

• Must learn to make predictions appropriately
  – Given “I ate an apple <eos>”, produce “Ich habe einen apfel gegessen <eos>”.

I ate an apple <eos> <sos> Ich habe einen apfel gegessen <eos>
Training: Forward pass

- Forward pass: Input the source and target sequences, sequentially
  - Output will be a probability distribution over target symbol set (vocabulary)
Training: Backward pass

- Backward pass: Compute the divergence between the output distribution and target word sequence.
Training: Backward pass

- Backward pass: Compute the divergence between the output distribution and target word sequence
- Backpropagate the derivatives of the divergence through the network to learn the net
Training: Backward pass

- In practice, if we apply SGD, we may randomly sample words from the output to actually use for the backprop and update
  - Typical usage: Randomly select one word from each input training instance (comprising an input-output pair)
    - For each iteration
      - Randomly select training instance: (input, output)
      - Forward pass
      - Randomly select a single output $y(t)$ and corresponding desired output $d(t)$ for backprop
Overall training

• Given several training instance \((X, D)\)
• For each training instance
  – Forward pass: Compute the output of the network for \((X, D)\)
    • Note, both \(X\) and \(D\) are used in the forward pass
  – Backward pass: Compute the divergence between selected words of the desired target \(D\) and the actual output \(Y\)
    • Propagate derivatives of divergence for updates
• Update parameters
Trick of the trade: Reversing the input

- **Standard trick of the trade:** The input sequence is fed *in reverse order*
  - Things work better this way
Trick of the trade: Reversing the input

- Standard trick of the trade: The input sequence is fed *in reverse order*
  - Things work better this way
Trick of the trade: Reversing the input

- Standard trick of the trade: The input sequence is fed in reverse order
  - Things work better this way

- This happens both for training and during inference on test data
Overall training

• Given several training instance \((X, D)\)

• Forward pass: Compute the output of the network for \((X, D)\) *with input in reverse order*
  
  – Note, both \(X\) and \(D\) are used in the forward pass

• Backward pass: Compute the divergence between the desired target \(D\) and the actual output \(Y\)
  
  – Propagate derivatives of divergence for updates
Applications

• Machine Translation
  – My name is Tom ➔ Ich heisse Tom/Mein name ist Tom

• Automatic speech recognition
  – Speech recording ➔ “My name is Tom”

• Dialog
  – “I have a problem” ➔ “How may I help you”

• Image to text
  – Picture ➔ Caption for picture
Machine Translation Example

- Hidden state clusters by meaning!
  - From “Sequence-to-sequence learning with neural networks”, Sutskever, Vinyals and Le
Machine Translation Example

<table>
<thead>
<tr>
<th>Type</th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our model</strong></td>
<td>Ulrich UNK, membre du conseil d’administration du constructeur automobile Audi, affirme qu’il s’agit d’une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d’administration afin qu’ils ne soient pas utilisés comme appareils d’écoute à distance.</td>
</tr>
<tr>
<td><strong>Truth</strong></td>
<td>Ulrich Hackenberg, membre du conseil d’administration du constructeur automobile Audi, déclare que la collecte des téléphones portables avant les réunions du conseil, afin qu’ils ne puissent pas être utilisés comme appareils d’écoute à distance, est une pratique courante depuis des années.</td>
</tr>
<tr>
<td><strong>Our model</strong></td>
<td>“Les téléphones cellulaires, qui sont vraiment une question, non seulement parce qu’ils pourraient potentiellement causer des interférences avec les appareils de navigation, mais nous savons, selon la FCC, qu’ils pourraient interférer avec les tours de téléphone cellulaire lorsqu’ils sont dans l’air”, dit UNK.</td>
</tr>
<tr>
<td><strong>Truth</strong></td>
<td>“Les téléphones portables sont véritablement un problème, non seulement parce qu’ils pourraient éventuellement créer des interférences avec les instruments de navigation, mais parce que nous savons, d’après la FCC, qu’ils pourraient perturber les antennes-relais de téléphonie mobile s’ils sont utilisés à bord”, a déclaré Rosenker.</td>
</tr>
<tr>
<td><strong>Our model</strong></td>
<td>Avec la créamation, il y a un “sentiment de violence contre le corps d’un être cher”, qui sera “réduit à une pile de cendres” en très peu de temps au lieu d’un processus de décomposition “qui accompagnera les étapes du deuil”.</td>
</tr>
<tr>
<td><strong>Truth</strong></td>
<td>Il y a, avec la créamation, “une violence faite au corps aimé”, qui va être “réduit à un tas de cendres” en très peu de temps, et non après un processus de décomposition, qui “accompagnerait les phases du deuil”.</td>
</tr>
</tbody>
</table>

- Examples of translation
  - From “Sequence-to-sequence learning with neural networks”, Sutskever, Vinyals and Le
Human Machine Conversation: Example

Machine: what is the error that you are running please
Human: i am seeing an error related to vpn 
Machine: what is the error message that you are getting when connecting to vpn using network connect?
Human: connection refused or something like that
Machine: may i know the version of network connect you connect?
Human: i am not sure i know that
Machine: is the network connect prompting for any user certificate

• From “A neural conversational model”, Orin Vinyals and Quoc Le
• Trained on human-human conversatons
• Task: Human text in, machine response out
Generating Image Captions

- Not really a seq-to-seq problem, more an image-to-sequence problem
- Initial state is produced by a state-of-art CNN-based image classification system
  - Subsequent model is just the decoder end of a seq-to-seq model
    - “Show and Tell: A Neural Image Caption Generator”, O. Vinyals, A. Toshev, S. Bengio, D. Erhan
Generating Image Captions

- Decoding: Given image
  - Process it with CNN to get output of classification layer
Generating Image Captions

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1...W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier
Generating Image Captions

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1...W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier
Generating Image Captions

- **Decoding:** Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier
Generating Image Captions

- **Decoding**: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, \text{Image})$
  - In practice, we can perform the beam search explained earlier
Generating Image Captions

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier
Generating Image Captions

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t | W_0 W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier
Training

- **Training**: Given several (Image, Caption) pairs
  - The image network is pretrained on a large corpus, e.g. image net
• **Training**: Given several (Image, Caption) pairs
  – The image network is pretrained on a large corpus, e.g. image net
• **Forward pass**: Produce output distributions given the image and caption
• **Training**: Given several (Image, Caption) pairs
  – The image network is pretrained on a large corpus, e.g. image net

• **Forward pass**: Produce output distributions given the image and caption

• **Backward pass**: Compute the divergence w.r.t. training caption, and backpropagate derivatives
  – All components of the network, including final classification layer of the image classification net are updated
  – The CNN portions of the image classifier are not modified (transfer learning)
Examples from Vinyals et al.

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."
Variants

A better model: Encoded input embedding is input to all output timesteps

Ich habe einen apfel gegessen <eos>

A boy on a surfboard <eos>

apple an ate I

A boy on a surfboard
Translating Videos to Natural Language Using Deep Recurrent Neural Networks

Subhashini Venugopalan, Huijun Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, Kate Saenko
# Assuming encoded input $H$ (from text, image, video)
# is available

# Now generate the output $y_{out}(1), y_{out}(2), ...$

$t = 0$

$h_{out}(0) = H$  # Encoder embedding

# Note: begins with a “start of sentence” symbol
#       <sos> and <eos> may be identical

$y_{out}(0) = <sos>$

do

  $t = t + 1$

  [$y(t), h_{out}(t)] = \text{RNN\_output\_step}(h_{out}(t-1), y_{out}(t-1), H)$

  $y_{out}(t) = \text{generate}(y(t))$  # Beam search, random, or greedy

until $y_{out}(t) == <eos>$
# Assuming encoded input $H$ (from text, image, video) is available

# Now generate the output $y_{out}(1), y_{out}(2), \ldots$

$t = 0$

$h_{out}(0) = H$  # Encoder embedding

# Note: begins with a “start of sentence” symbol
#       <sos> and <eos> may be identical

$y_{out}(0) = <sos>$

do
    $t = t + 1$
    $[y(t), h_{out}(t)] = RNN\_output\_step(h_{out}(t-1), y_{out}(t-1), H)$
    $y_{out}(t) = generate(y(t))$  # Beam search, random, or greedy
until $y_{out}(t) == <eos>$
• We are overloading this guy
• How can we do better?
• Next class