University of Massachusetts Lowell

Massachusetts Institute of Technology

Modeling Long-Distance Dependencies and Modular Representations for Natural Language Processing

Anna Rumshisky

Conversational Intelligence Summer School
@ Moscow Institute of Physics and Technology

July 2, 2018

Talk Outline

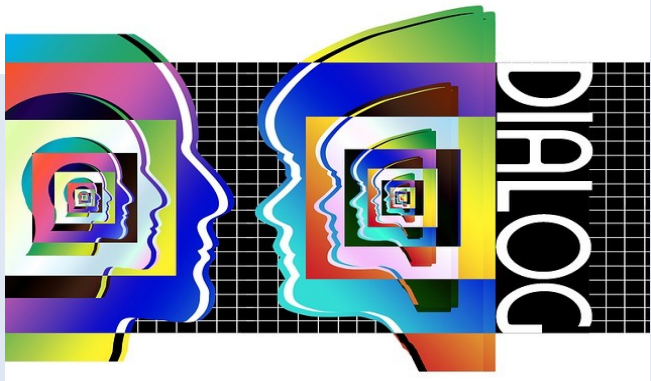
- Current state of NLP. A bit of history and a quick primer on current techniques.
- What's next – a vision for the near future and two projects:

  1. Learning a modular representation that dissociates meaning from form (adversarial learning)

  2. Extracting event timelines from text – what happened, when and in what order? (external memory for long-term dependencies)

What is our goal in NLP?

- We want to develop computational models of human language, which would be able to interpret and generate free text.
- We want our models to be as good as humans or better at the myriad of language-related tasks
  - Comprehension, reasoning, inference; fluent dialog.
  - Extracting information, summarizing content, answering questions; translation between languages.
  - Humor, irony, metaphors, poetry, story telling.
- Put simply, no general AI is possible without language understanding.



A Bit of History

- NLP started with several prominent mathematicians
  - Claude Shannon's work on estimating the entropy of human language
  - Alan Turing and Jack Good's work on statistical language modeling – estimating the probability of a given sequence of words – to break the German Enigma code during WW2.
  - Warren Weaver's take on machine translation ("I am going to pretend that [the Russian text] is really written in in English and that it has been coded in some strange symbols. All I need to do is strip off the code in order to retrieve the information contained in the text")
- But the early history was wrought with broken promises – starting with the promise of fluent machine translation in 1954

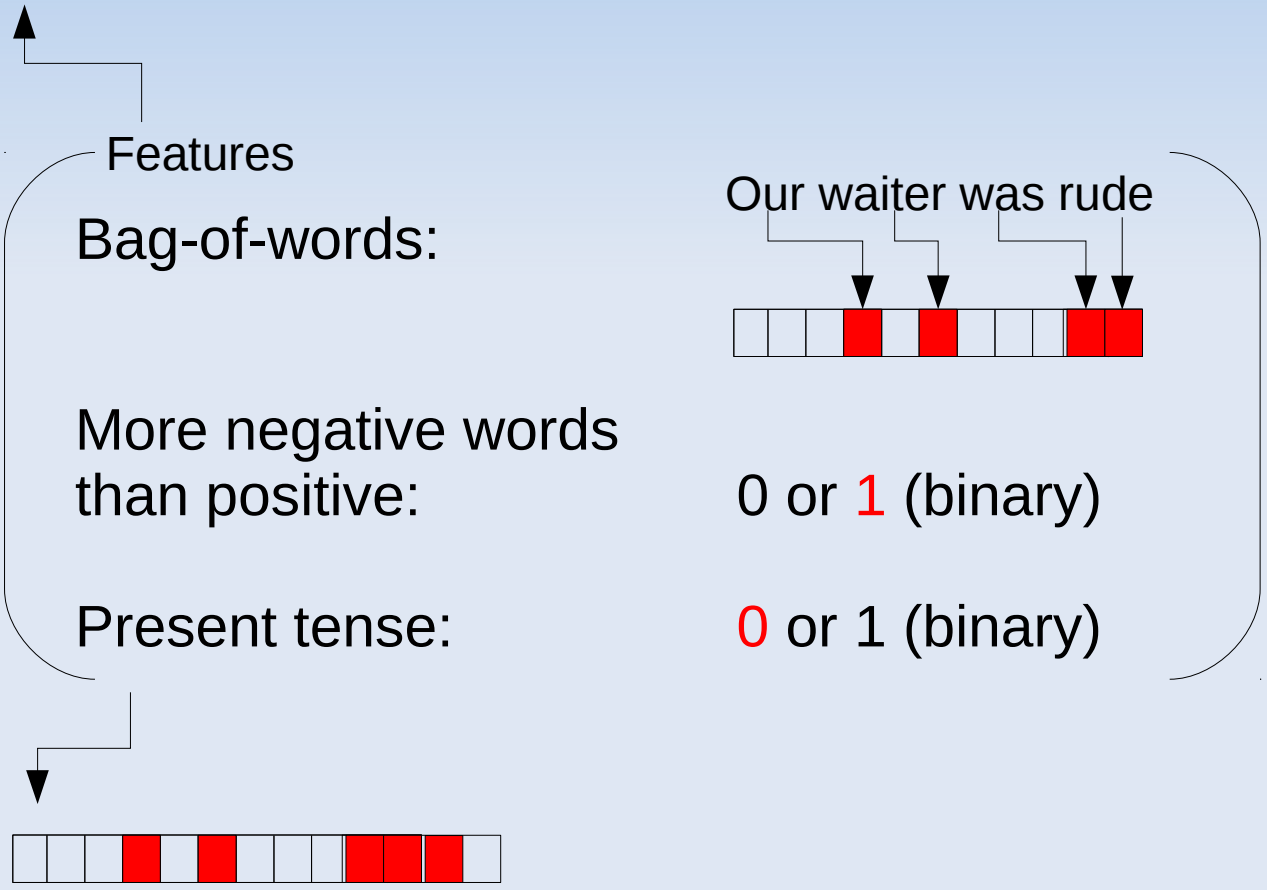
Later NLP

- **Rule-based systems** [1960s – 1980s]: directly encoding human knowledge about linguistic structure; patterns encoded by linguists.
  - Finite state methods, rule-based context-free grammars, etc.
- **Statistical machine learning** [1990s – 2000s]: knowledge about linguistic structure encoded in the form of features and patterns learned automatically from annotated corpora.
  - Sequence tagging with HMMs, CRFs; multinomial logistic regression, SVMs for classification; topic modeling with LDA; ngram language models for generation, etc.
- **The last 5-7 years (deep learning)**: data representations themselves no longer engineered by humans, but learned. Continuous-valued vectors serve as representations for input words
  - Multi-layer neural models: recurrent neural networks (RNNs) for sequence-to-sequence models: for classification, generation, etc. Attention models, external memory models.

Example – Feature Representations: Sentiment Analysis

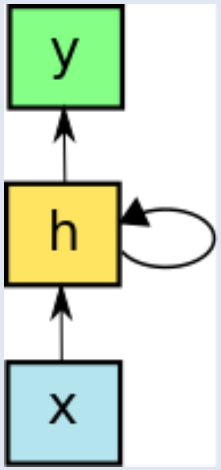
<s> Our waiter was rude . </s>

LABEL: NEGATIVE



Where are we now?

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



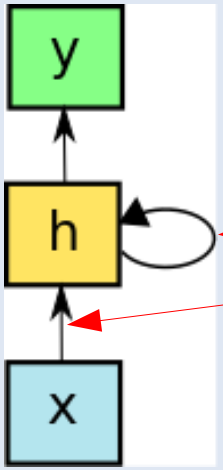
$$y_t = softmax(\mathrm{W}_{out}\, h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t\ ;\ h_{t\text{-}1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



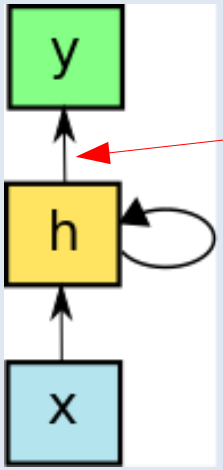
$$y_t = softmax(\mathrm{W}_{out}\, h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t\,;\, h_{t\text{-}1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



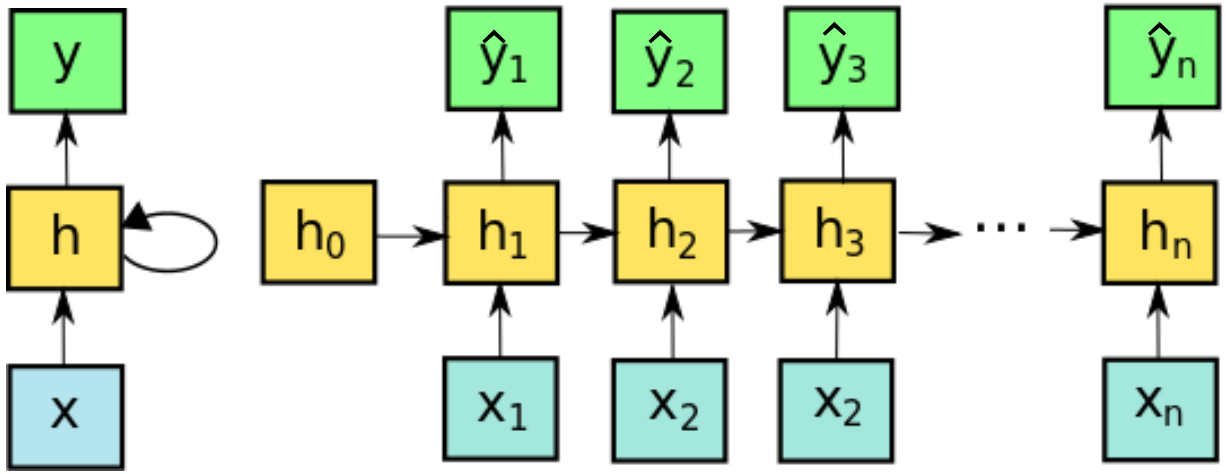
$$y_t = softmax(\mathrm{W}_{out}\, h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t\ ;\ h_{t\text{-}1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



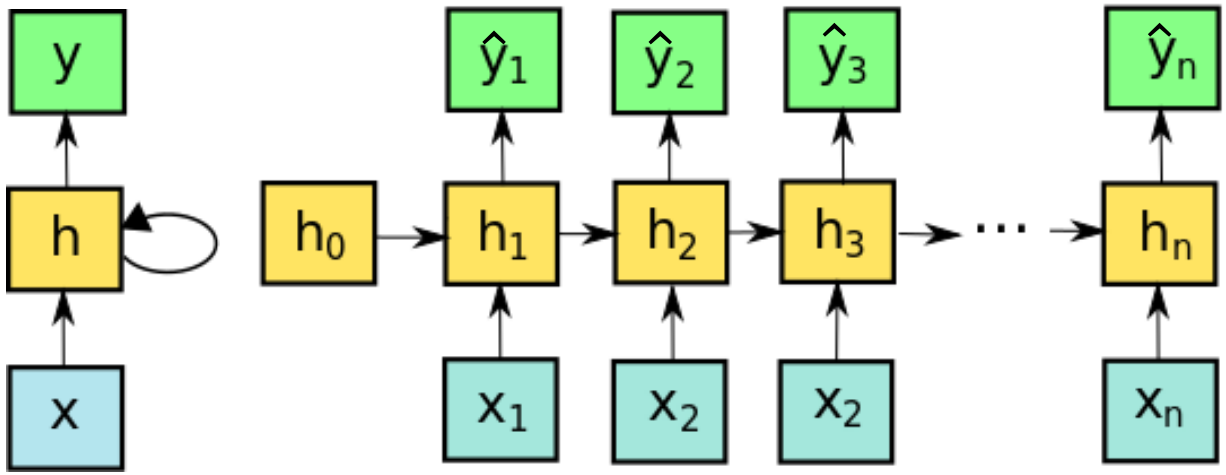
$$y_t = softmax(\mathrm{W}_{out}\, h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t\,;\, h_{t\text{-}1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



$x_i =$

Our

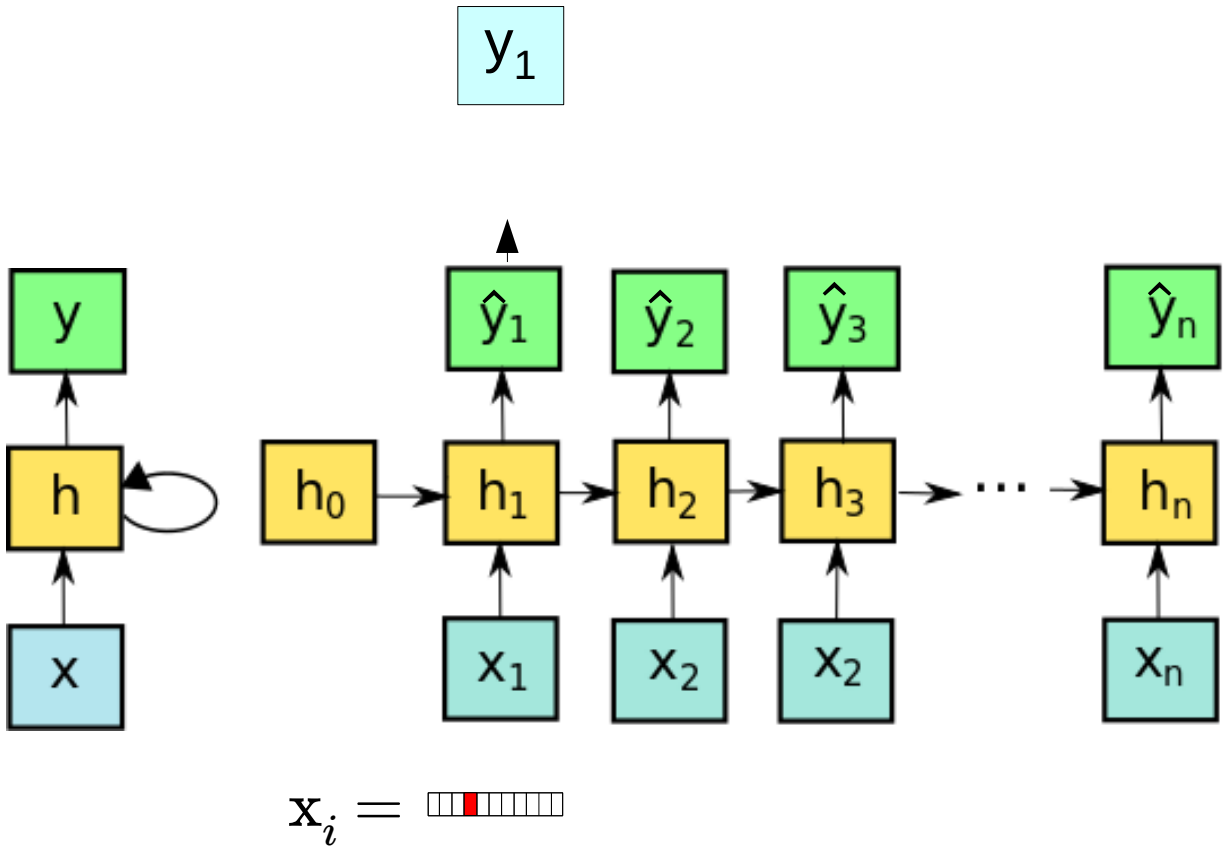
$$y_t = softmax(W_{out} h_t + b_{out})$$

$$h_t = g(W_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



Our

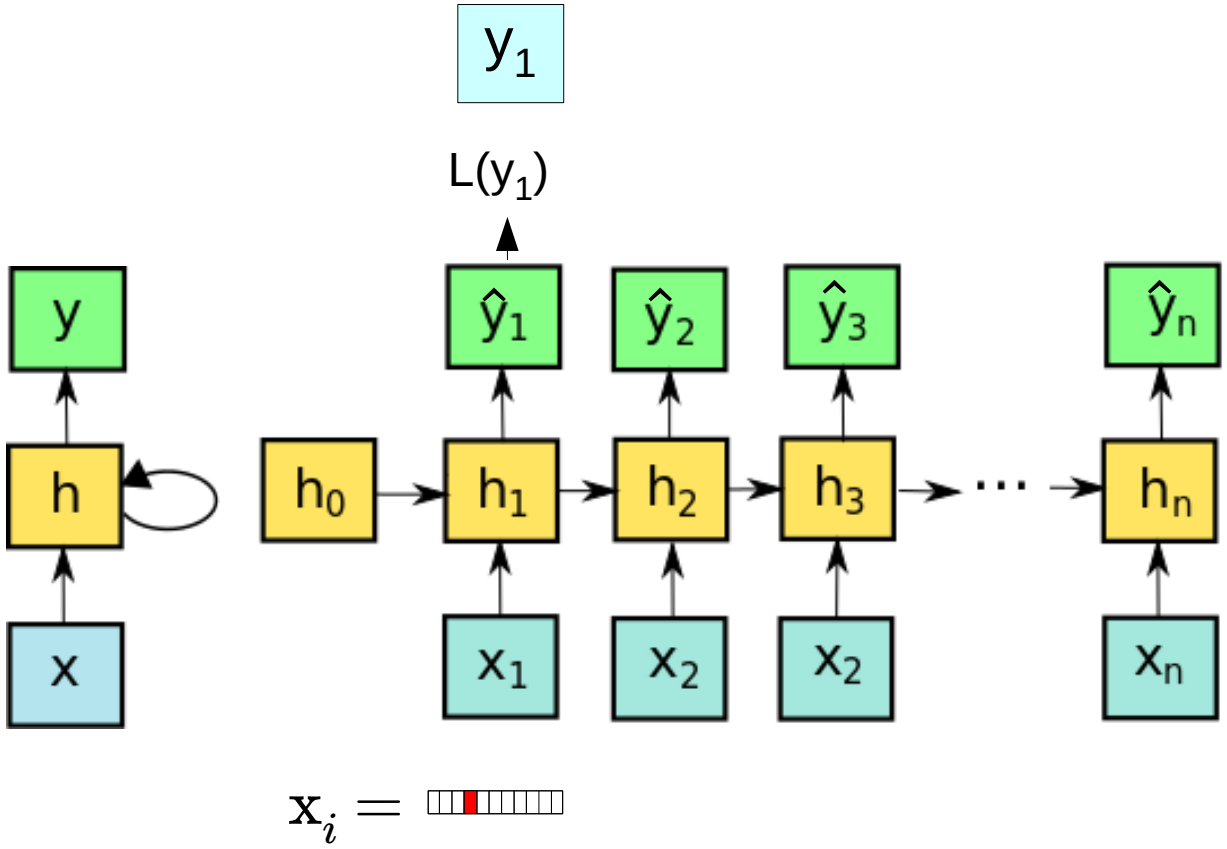
$$y_t = softmax(W_{out} h_t + b_{out})$$

$$h_t = g(W_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



Our

$L(y_1, \hat{y}_1)$ is the loss function

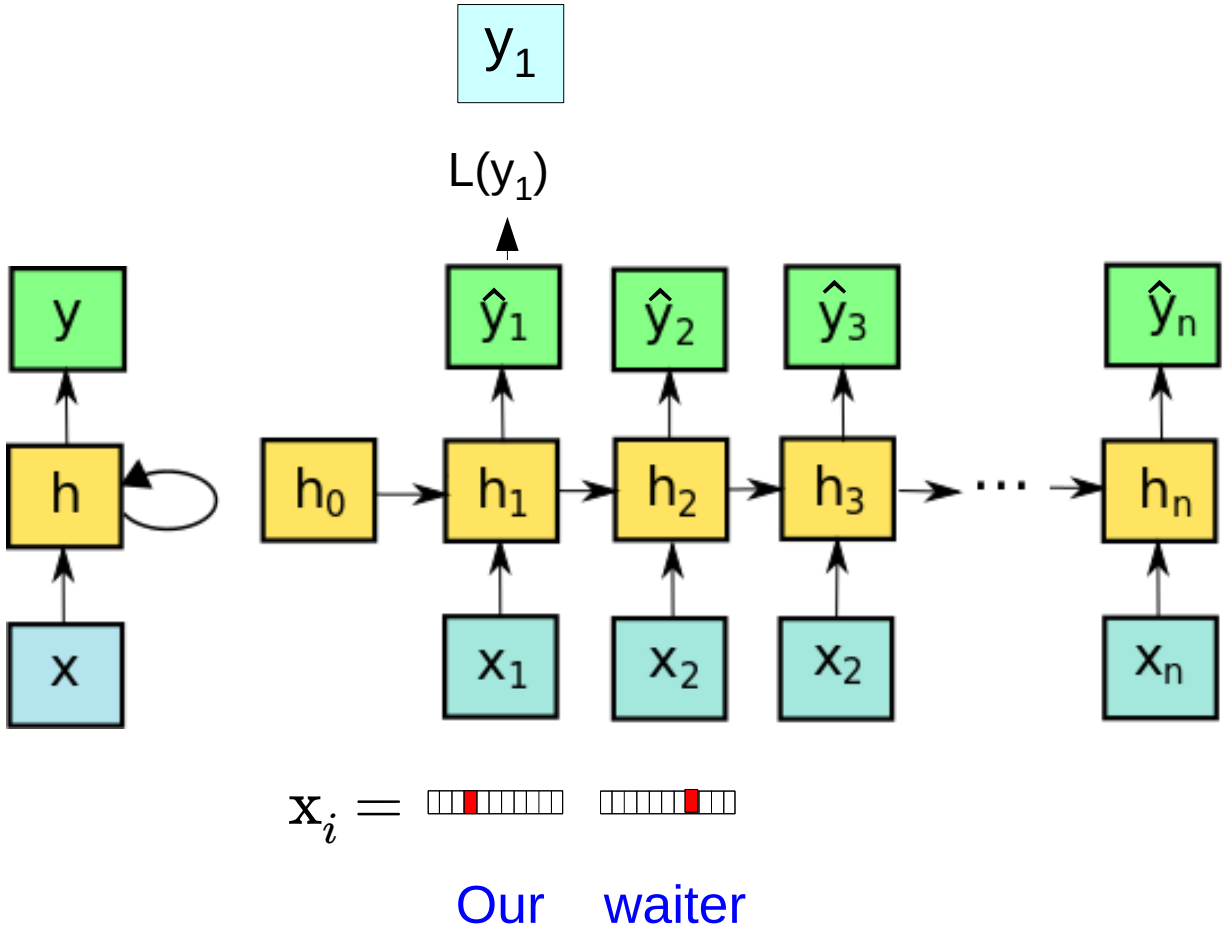
$$y_t = softmax(\mathrm{W}_{out} h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



$L(y_i, \hat{y}_i)$ is the loss function

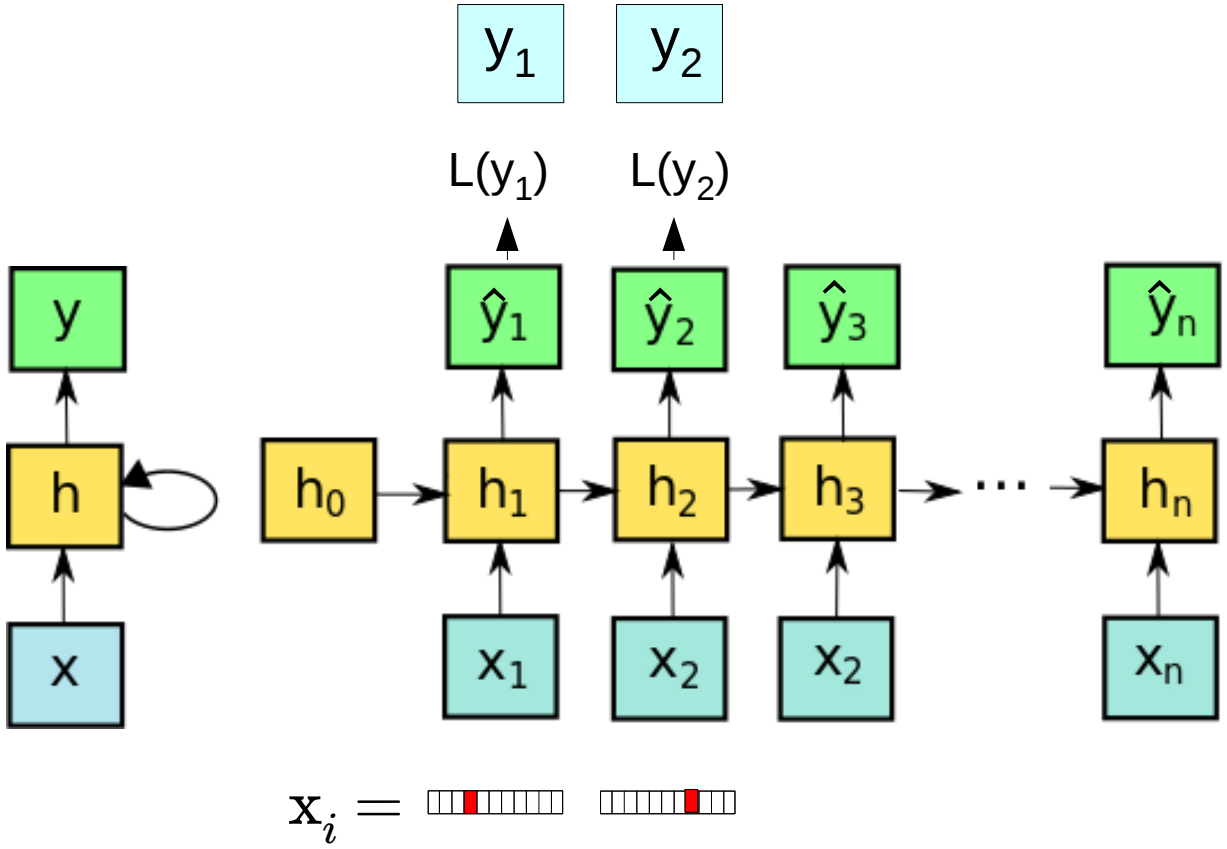
$$y_t = softmax(\mathrm{W}_{out} h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



$L(y_i, \hat{y}_i)$ is the loss function

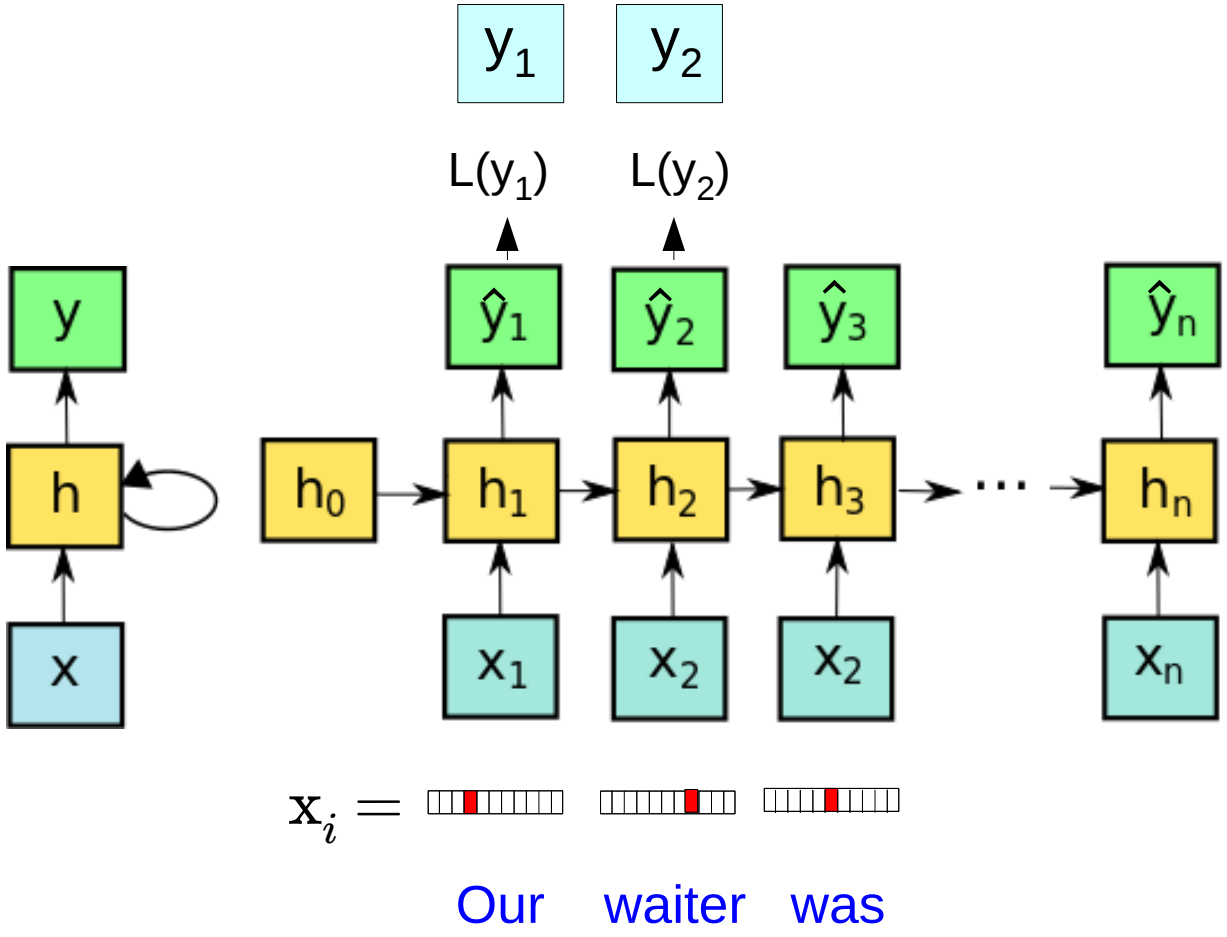
$$y_t = softmax(\mathrm{W}_{out} h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



$L(y_i, \hat{y}_i)$ is the loss function

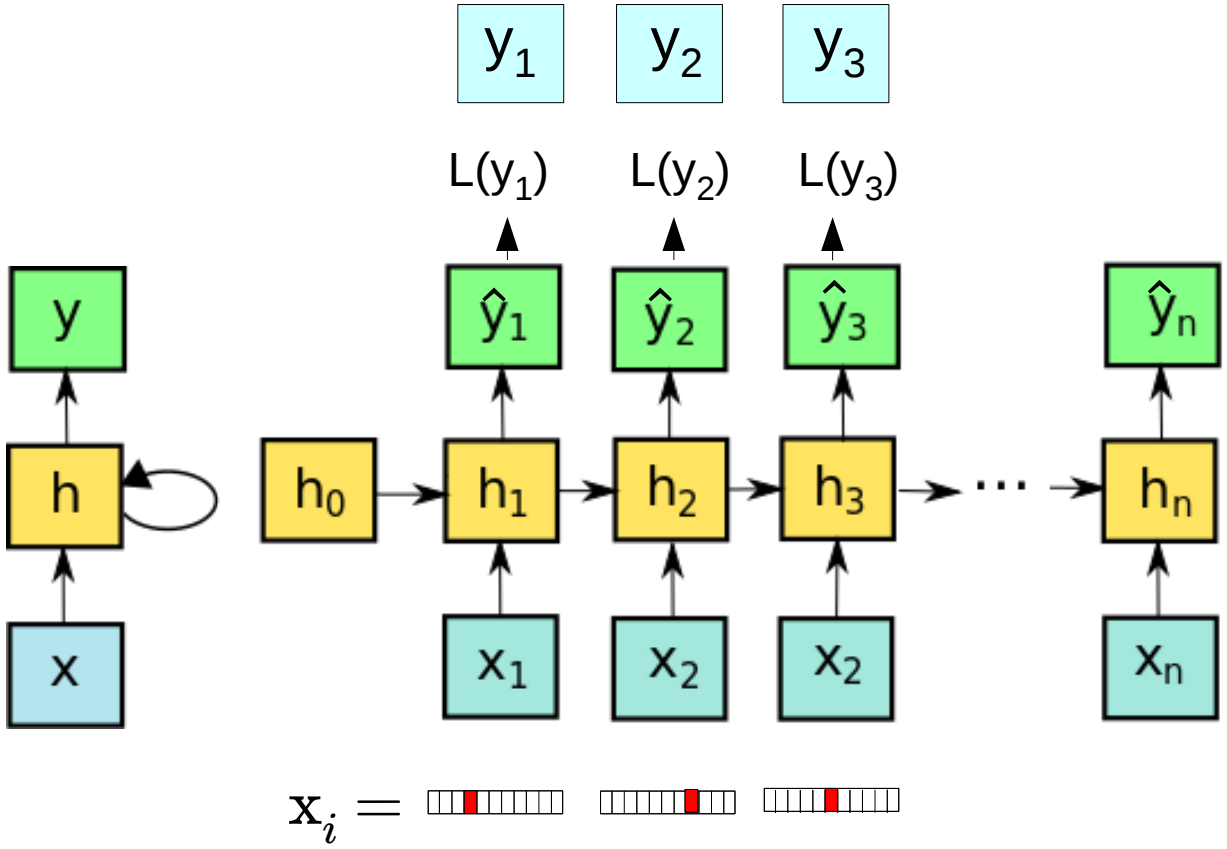
$$y_t = softmax(\mathrm{W}_{out} h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



$L(y_i, \hat{y}_i)$ is the loss function

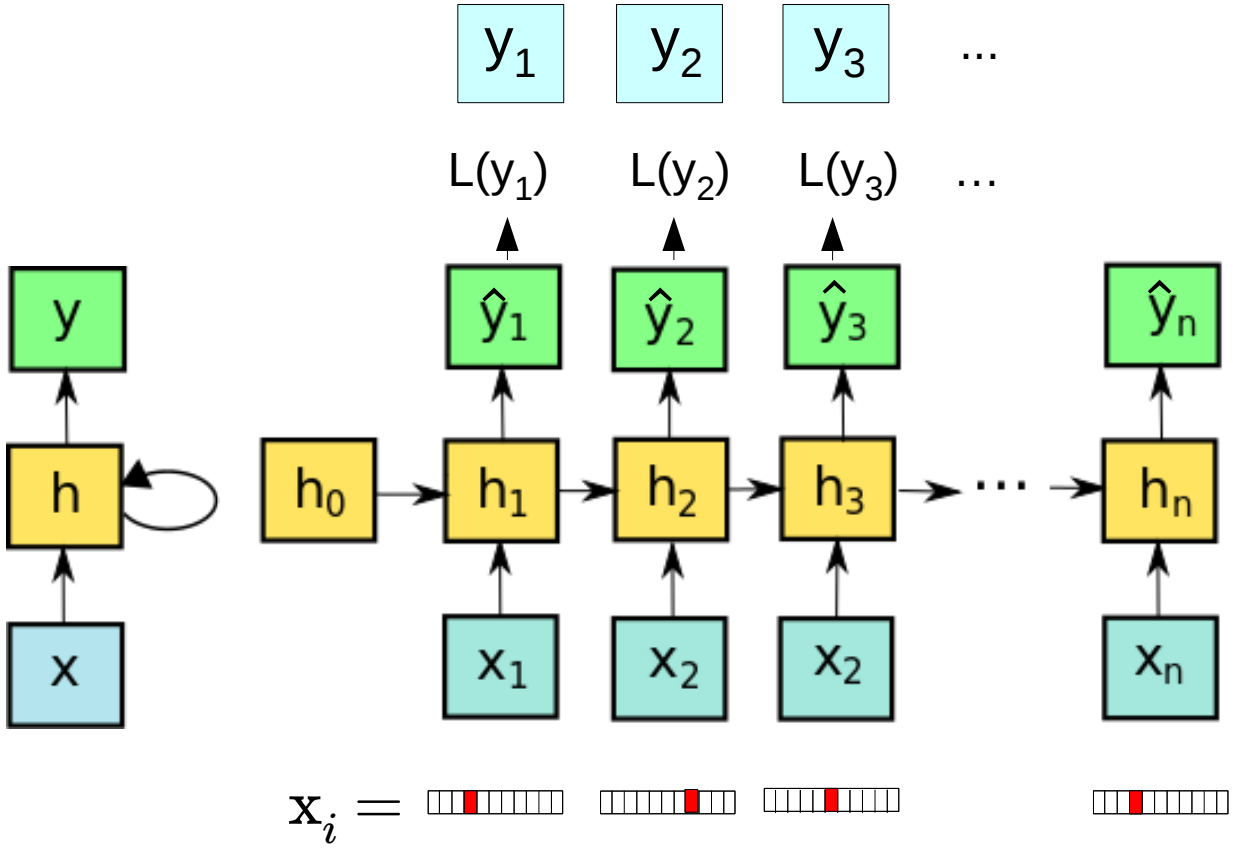
$$y_t = softmax(\mathrm{W}_{out} h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



$L(y_i, \hat{y}_i)$ is the loss function

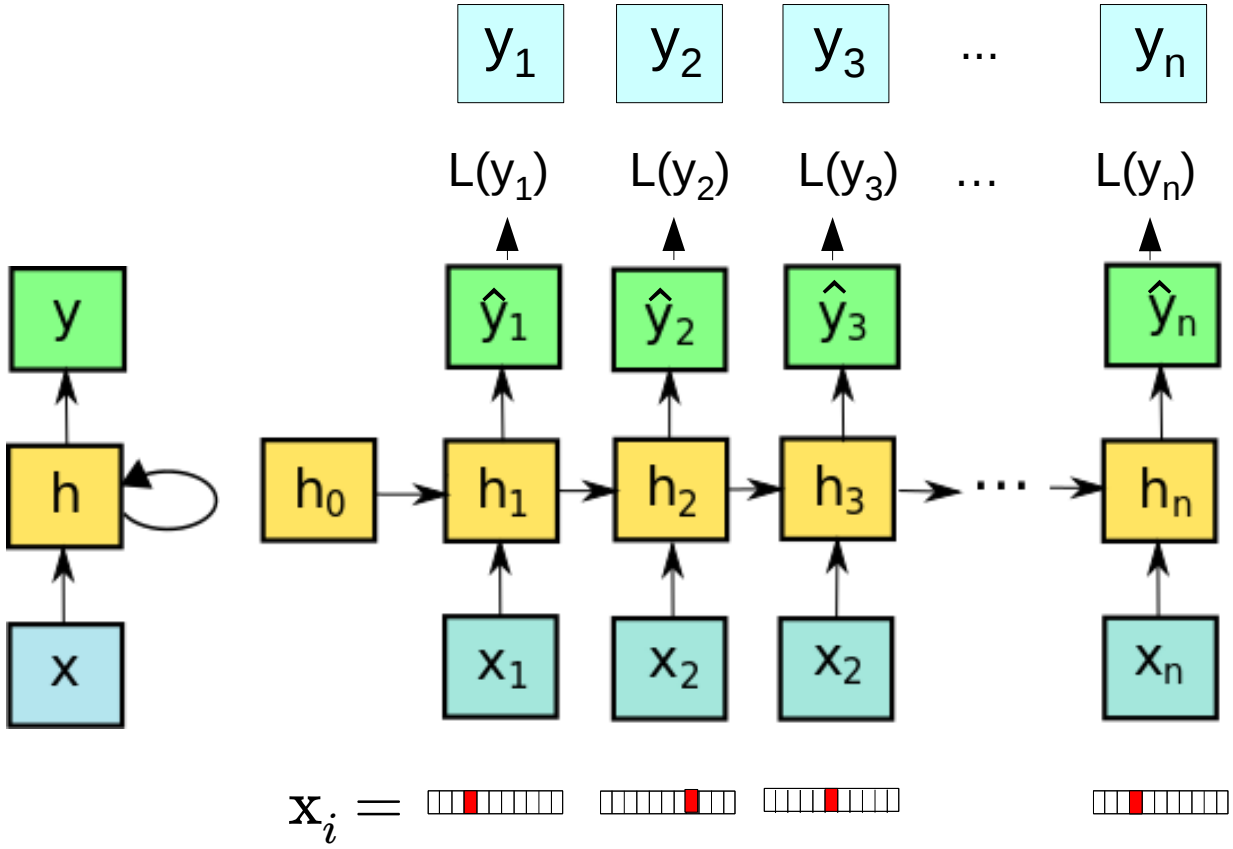
$$y_t = softmax(\mathrm{W}_{out} h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Recurrent Neural Networks

- Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



$L(y_i, \hat{y}_i)$ is the loss function

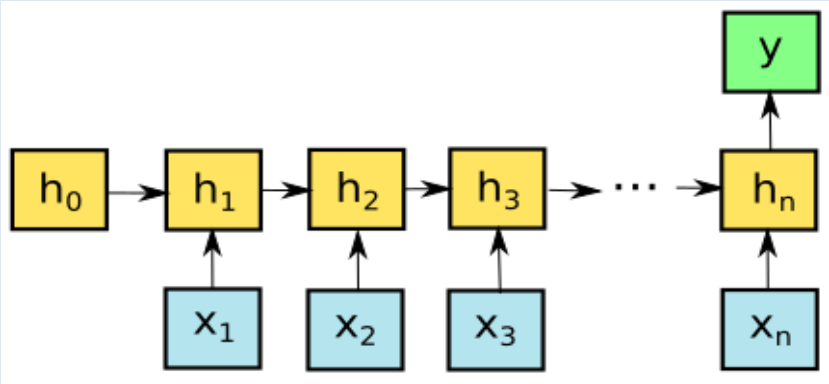
$$y_t = softmax(\mathrm{W}_{out} h_t + b_{out})$$

$$h_t = g(\mathrm{W}_{in}[x_t ; h_{t-1}] + b_{in})$$

- the same matrix W_{in} used at every timestep
- $g(.)$ is an activation function, such as σ, tanh or relu

Where are we now?

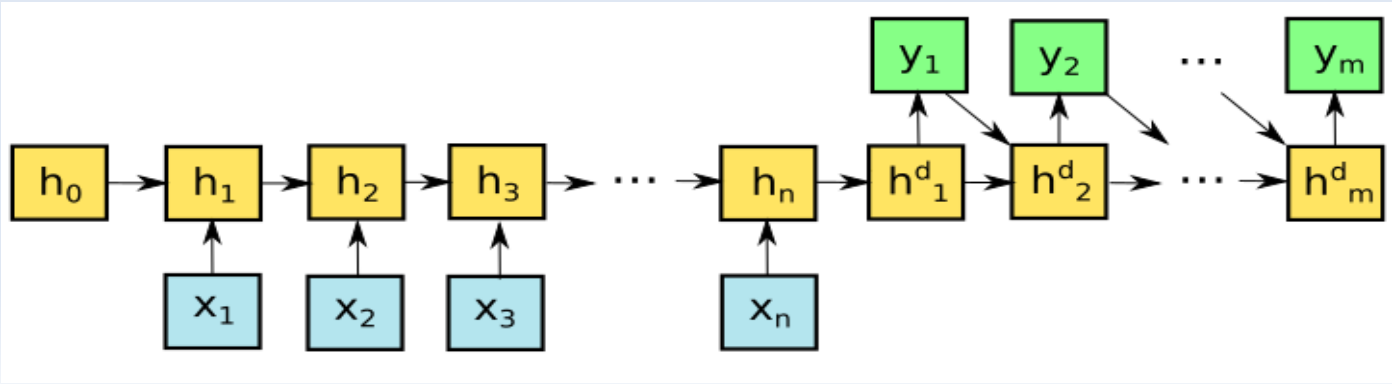
- Common architectures look like this:
  - Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.



← **classification**

← **regression**

input sequence encoded into a single vector

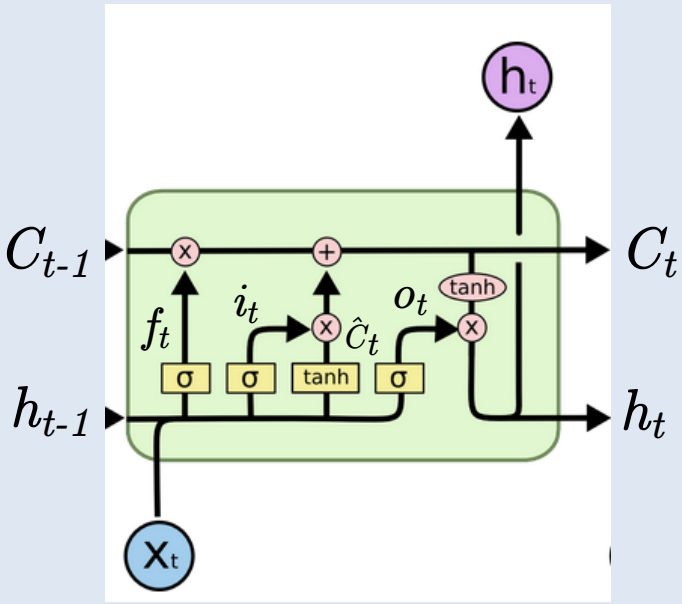


← **generation**

output sequence generated from a single vector

Where are we now?

- Common architectures look like this:
  - Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.
  - Typically, gated cell memory that can be addidively updated at each time step to learn long-distance dependencies
    - Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU).



Gates that regulate hidden state update

$$f_t = \sigma(W_f [x_t ; h_{t-1}] + b_f)$$
$$i_t = \sigma(W_i [x_t ; h_{t-1}] + b_i)$$
$$o_t = \sigma(W_o [x_t ; h_{t-1}] + b_o)$$

Cell state and output layer updates:

$$\hat{C}_t = \tanh(W_C [x_t ; h_{t-1}] + b_C)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$
$$h_t = o_t \odot \tanh(C_{t-1})$$

Diagram credit: Chris Olah

LSTMs and the like

- *Addition* is used instead of *multiplication* to update the LSTM **cell state**, which is used in place of the hidden layer.
- Since we're doing similar matrix multiplication for each gate, can optimize it by concatenating parameter matrices (and data vectors)
- LSTMs are able to better learn long-distance dependencies, but have an increased number of parameters → reduced memory per parameter.
- Number of parameters matters more than architecture. But: architecture improves learnability.

Where are we now?

- Common architectures look like this:
  - Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.
  - Typically, gated cell memory that can be updated at each time step – Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks to handle longer dependencies.
  - Using a pre-trained dense vector-space embedding as input for each word; usually also updated during training.

$$\log \mathrm{p}(\boldsymbol{w}) \approx \sum_{m=1}^{M} \sum_{n=1}^{h_m} \log \mathrm{p}(w_{m-n} \mid w_m) + \log \mathrm{p}(w_{m+n} \mid w_m)$$

$$= \sum_{m=1}^{M} \sum_{n=1}^{h_m} \log \frac{\exp(\boldsymbol{u}_{w_{m-n}} \cdot \boldsymbol{v}_{w_m})}{\sum_{j=1}^{V} \exp(\boldsymbol{u}_j \cdot \boldsymbol{v}_{w_m})} + \log \frac{\exp(\boldsymbol{u}_{w_{m+n}} \cdot \boldsymbol{v}_{w_m})}{\sum_{j=1}^{V} \exp(\boldsymbol{u}_j \cdot \boldsymbol{v}_{w_m})}$$

  - e.g. trained to maximize the probability of a given word expressed in terms of surrounding word vectors (SG-W2V: a dot product with the context vector)

Where are we now?

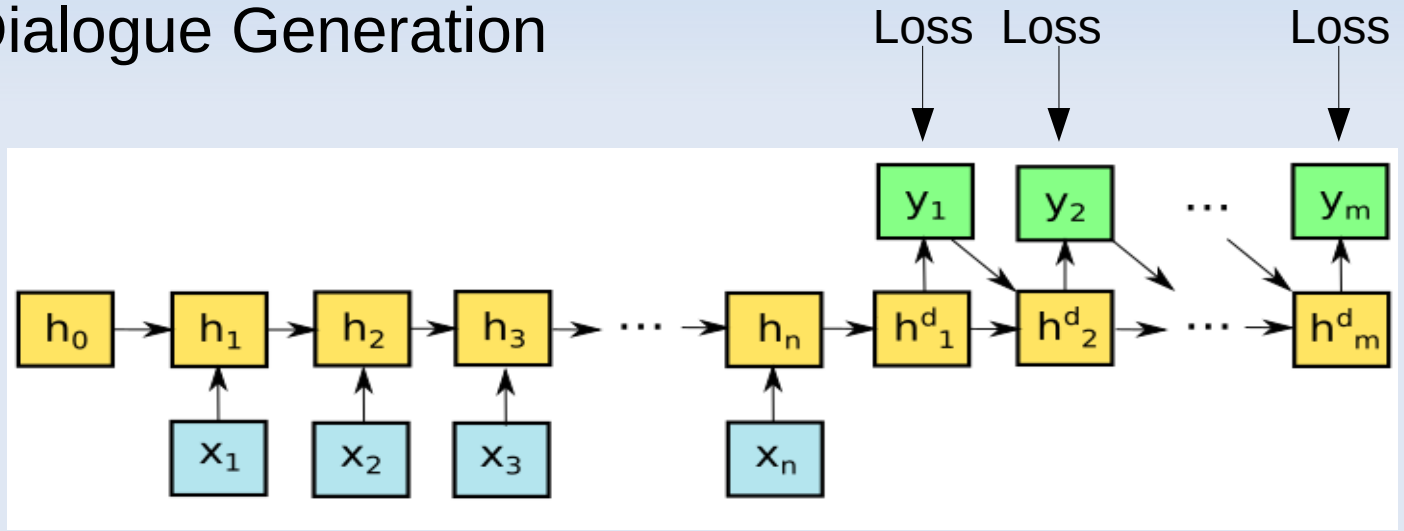
- Common architectures look like this:
  - Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.
  - Typically, gated cell memory that can be updated at each time step – Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks to handle longer dependencies.
  - Using a pre-trained dense vector-space embedding as input for each word; usually also updated during training.
  - For generation, sequence-to-sequence models with encoder-decoder architecture are the bread-and-butter of NN models
    - an input word is read in by the LSTM encoder at each timestep
    - after the input sequence is processed, the resulting hidden representation is used to generate a probability distribution over vocabulary, generating an output word at each timestep

Where are we now?

- Common architectures look like this:
  - Recurrent neural networks (RNNs) – handle variable length inputs/outputs, unrolled and trained with backpropagation to compute gradient.
  - For generation, sequence-to-sequence models with encoder-decoder architecture are used
    - an input word is read in by an RNN encoder at each timestep
    - after the input sequence is processed, the resulting hidden representation is used to generate a probability distribution over vocabulary, generating an output word at each timestep
    - Can be used in an unsupervised manner in an autoencoder architecture, where the hidden representation is used to reconstruct the input sequence

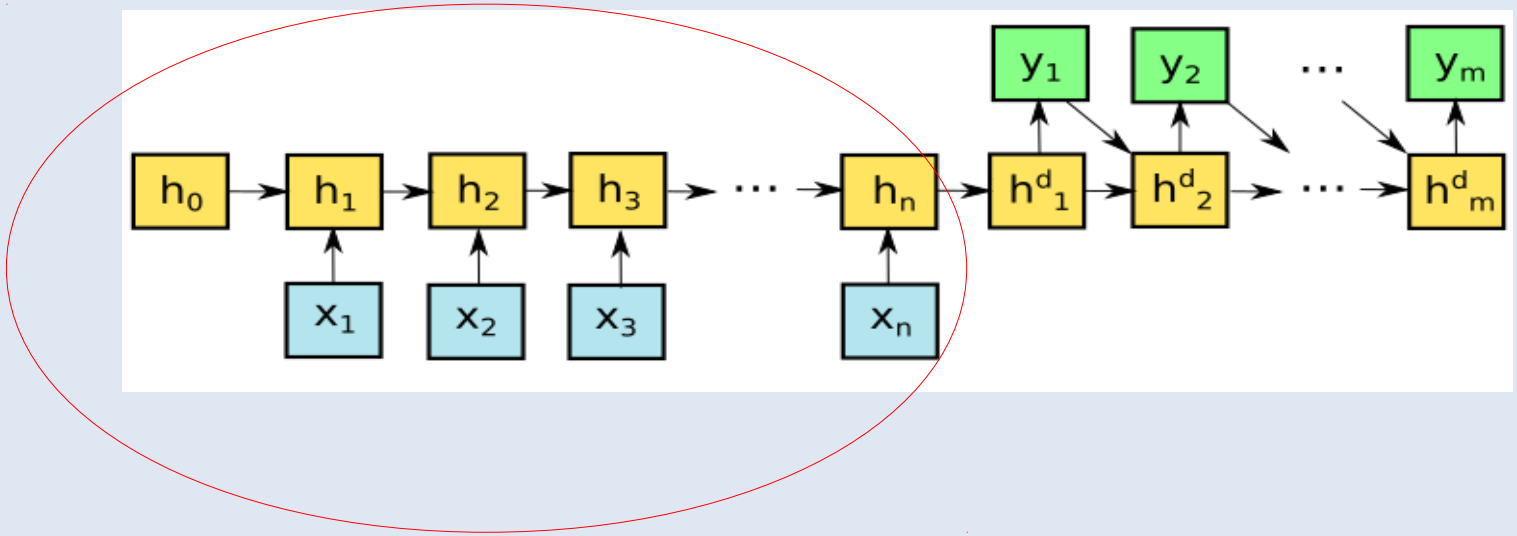
Sequence to Sequence RNNs

- Seq2seq models for generation – applications:
  - Question Answering
  - Machine Translation
  - Dialogue Generation



Sequence to Sequence RNNs

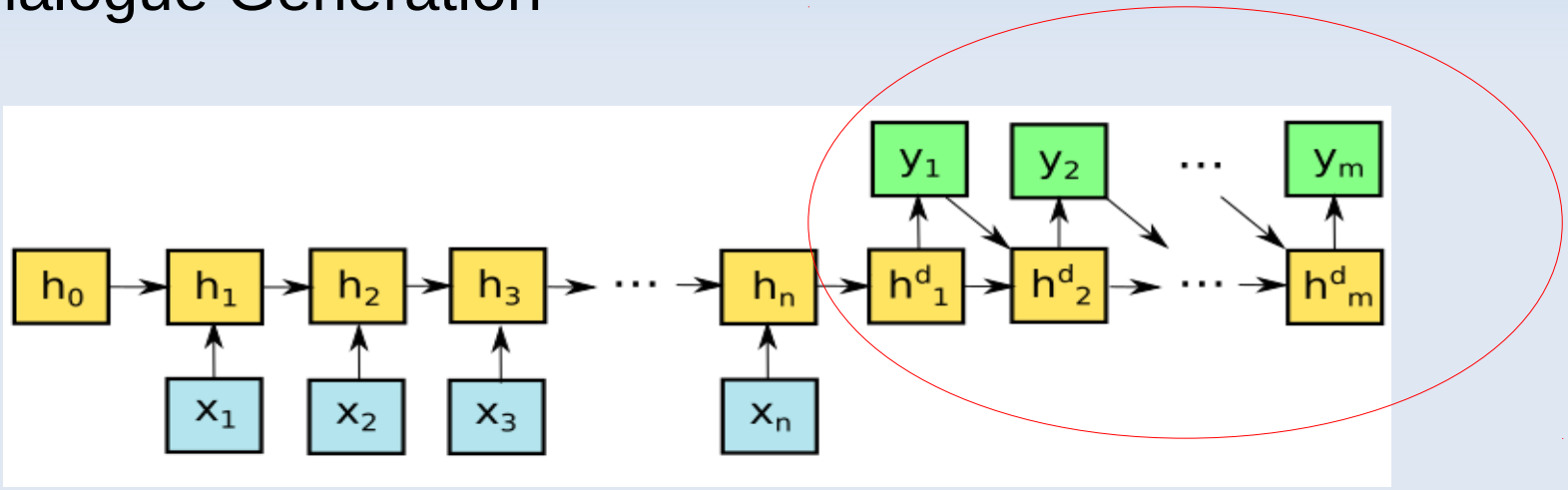
- Seq2seq models for generation – applications:
  - Question Answering
  - Machine Translation
  - Dialogue Generation



Encoder

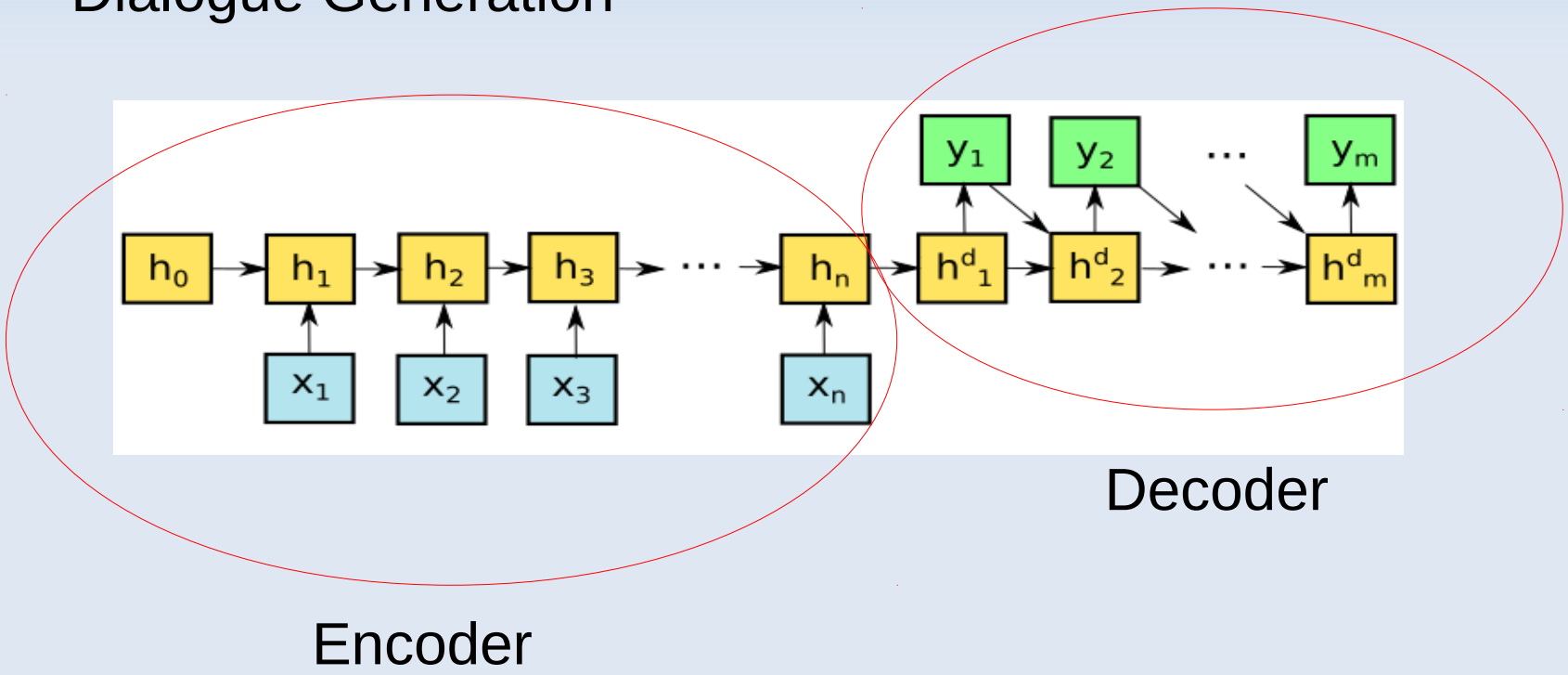
Sequence to Sequence RNNs

- Seq2seq models for generation – applications:
  - Question Answering
  - Machine Translation
  - Dialogue Generation



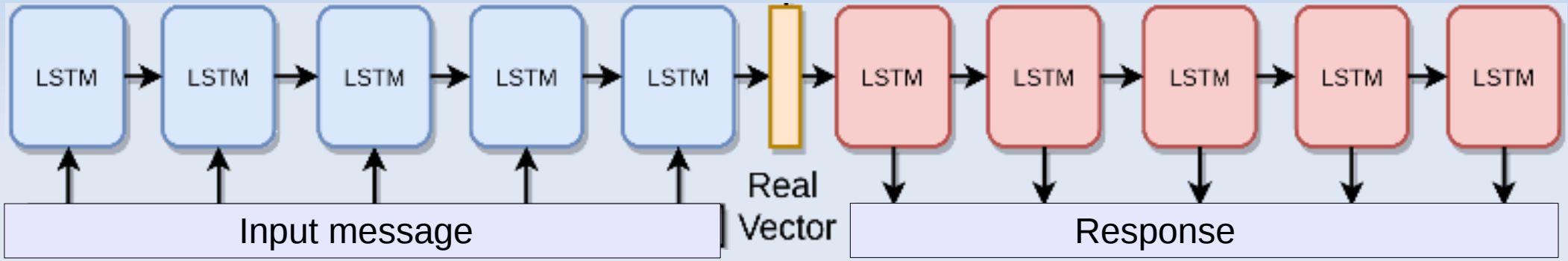
Sequence to Sequence RNNs

- Seq2seq models for generation – applications:
  - Question Answering
  - Machine Translation
  - Dialogue Generation



Where are we now?

- **(very simple) Dialogue generation**



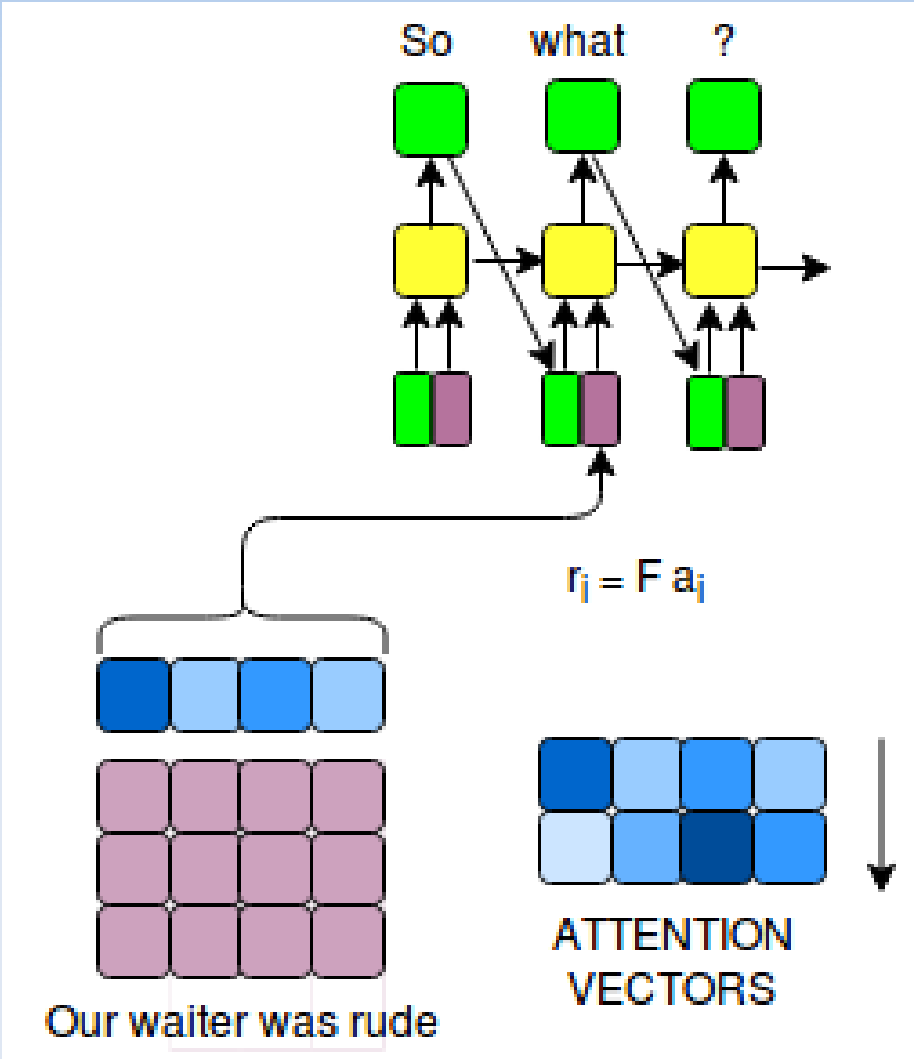
- A sequence to sequence model with encoder LSTM RNN and decoder LSTM. Each has a cell state size of 200.
- Outputs a probability distribution at each timestep, from which a word is sampled and fed back into the model at the next time step.
- Trained on Cornell Movie Dialogues dataset with about 60k conversational (message, response) pairs.
- Learns word embeddings as input instead of using pretrained ones
- 50% dropout between the encoder and the decoder.

Dialogue Generation

Text Machine NLP Lab – Chatbot demo 2017

Attention Models

- Instead of reducing the full input to a fixed-length vector prior to decoding, use the full input sequence at each generation step
- Weighted at each timestep by an ”attention” vector computed as a function of the current hidden state.



Attention Only – Transformer Models

- Input is no longer read in as a sequence
  - Instead, several rounds of attention over the full input sequence are applied at encoding time.
  - For each word w, self-attention combines information from surrounding words, creating a new representation for w
  - Repeated several times, producing more complex representations at higher layers.
  - Positional encodings are appended to word embeddings to track position in a sequence.
- Decoder also generates several layers of representation, at each point attending (left-to-right) to previously generated representations for decoded words and the representation generated by the encoder.
  - Illustration

Desiderata for NLP Models, or "What's Still Missing?"

- **Long-distance dependencies** still not handled well; e.g. enforcing topic/discourse coherence, where things mentioned earlier in the text can be referred to again.
  - LSTM/GRU RNNs with additive update hidden state help
- **Knowledge integration** – Knowledge storage, integration and update mechanisms.
  - Not solved; external knowledge integrated via regularization, task-specific external memory components
- **Modeling language processing in humans** – We're missing an integrated ability for realistic language processing; support for higher-level functions such as reasoning.
  - Not solved; memory models; used on simple subsets, synthetic data (e.g. BabI);
  - No current attempts to model either the neurobiology or the functionality of language processing in the brain.

Text Machine Lab's Work

Long-distance dependencies

- How do humans structure arguments? (Potash, Romanov, Rumshisky EMNLP 2017)
- Debate winner prediction (Potash & Rumshisky EMNLP 2017)
- What is the order and timing of events described in text? (Meng, Romanov, Rumshisky EMNLP 2017, Meng & Rumshisky, ACL 2018)

Knowledge integration

- What makes an argument convincing? (Potash, Bhattacharya, Rumshisky IJCNLP 2017)

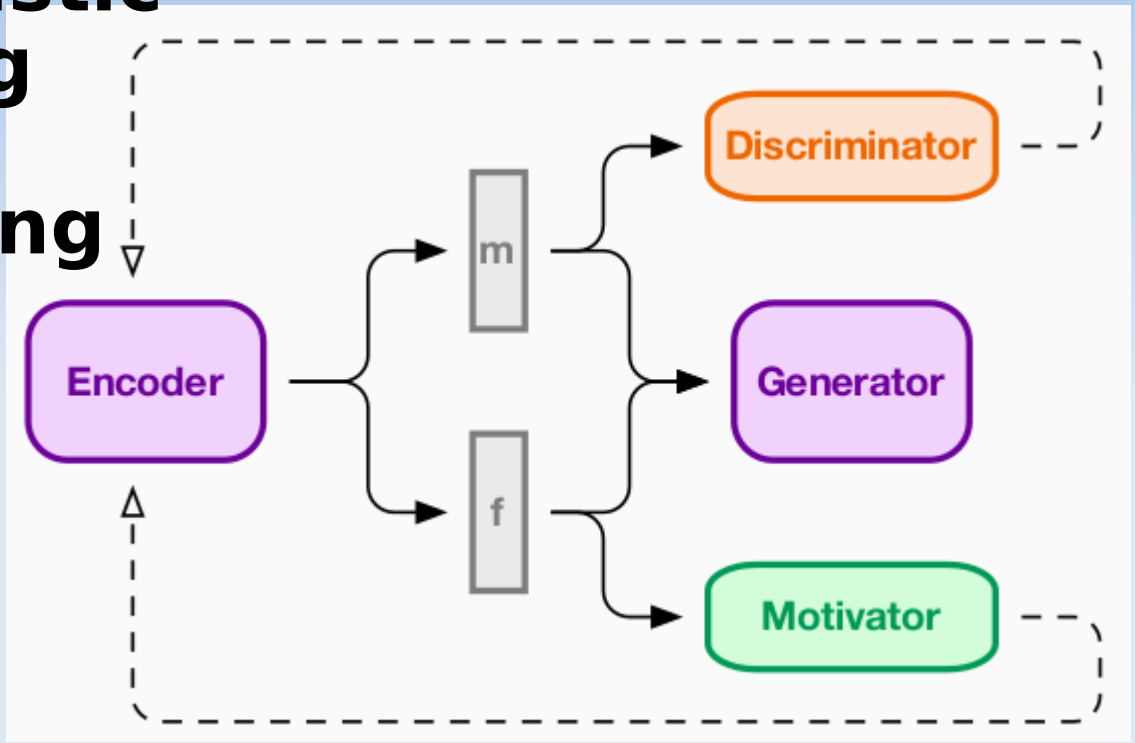
Modeling the neurobiology & the functionality of language processing in humans

- Can we learn **a modular representation of linguistic input** that mimics the functional specialization of language processing in the brain? (Romanov, Rumshisky et al under review)

This Talk

- Topic of the day #1. Learning a modular representation of linguistic input that mimics the functional specialization of language-processing regions in the brain.
  - Our solution ’18: *DissoNet* model with Adversarial / Motivational training (Romanov, Rumshisky et al, under review)
- Topic of the day #2. Extracting event timelines from text – what happened, when and in what order?
  - Our solution ’17: A uniform set of dependency-based LSTMs (Meng, Romanov & Rumshisky, EMNLP 2017)
  - Our solution ’18: *Global Context Layer* model with updateable memory to capture context (Meng & Rumshisky, ACL 2018)

Dissociating Linguistic Form from Meaning with Adversarial-Motivational Training



- GAN-like architecture to encourage the encoder/generator to produce dissociated representations.
- Adversarial behavior of the discriminator mitigated by the motivator

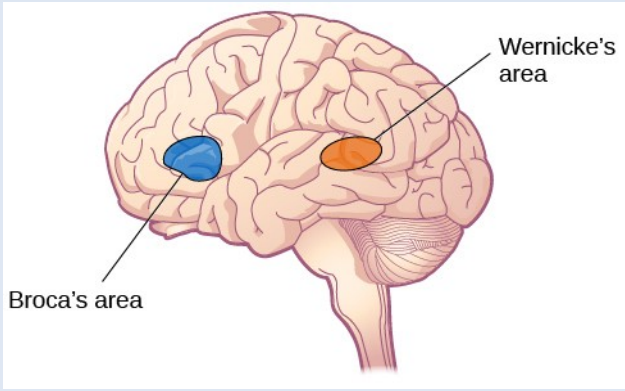
Romanov, Rumshisky, et al (under review)

Motivation

- Modeling human-like learning at the micro-level of human neurobiology (*capsules, equilibrium propagation*) and at the macro-level of general principles of intelligent cognition seem to be the pathway to general AI (*imitation learning*, *planning, prediction,* etc.)
- In NLP, few, if any, attempts to that align computational models with how humans use language.
- One property of language processing widely investigated in cognitive neuroscience is the **functional specialization of brain regions related to language processing**
  - particularly “grammatic” and “semantic” aphasias in patients with lesions in the Broca and Wernicke’s areas

Motivation

- Broca (expressive) aphasics can not form grammatical sentences, but are able to comprehend.
- Wernicke (receptive) aphasics can produce connected speech, but struggle with comprehension.
- Linguists have long modeled form and meaning as separate – although the exact mechanisms behind these impairments are hotly debated (Fedorenko 2016, Grodzinsky 2010)



- Can computational models support a representational modularity that mimics the functional specialization of language-processing regions in the brain?

Can We Dissociate Meaning from Form?

- For example, can separate out the representation of a different **linguistic register**? (a subset of language appropriate in specific circumstances)

Informal	**Formal**
Yo, what's up?	Hello, how are you?

  - formal and informal language, the language used in different genres (e.g., fiction vs. newspapers vs. academic texts), etc.
- OUR CASE: **Newspaper headlines vs. Scientific article titles**

Can We Dissociate Meaning from Form?

- For example, can we separate out the representation of the **specific diachronic slice of language**?
  - Early Modern English of Shakespeare vs. contemporary English – can our representation reflect the same meaning, but a different form?

Early Modern English (Original Shakespeare)	**Contemporary English (Translated Shakespeare)**
How long hath she been thus?	How long has she been like this?
I'll be with you straight.	I'll be there in a minute.
What would she have?	What does she want?

Can We Separate Out Specific Aspects of Meaning?

- Can we factor out specific aspects of meaning (e.g. **sentiment polarity**)?

Positive sentiment	**Negative Sentiment**
The service was horrible.	We loved the food.

- OUR CASE: Yelp reviews, Amazon reviews.

Our Task

- The goal of is to learn
  - Two encoders

    $E_m : \mathcal{X} \rightarrow \mathcal{M}$

    $E_f : \mathcal{X} \rightarrow \mathcal{F}$

    for the meaning and for the form, respectively
  - Generator $G : \mathcal{M}, \mathcal{F} \rightarrow \mathcal{X}$
- Basically, we want to be able to swap in a different form for the same meaning and generate text in a different style.

Inspirations

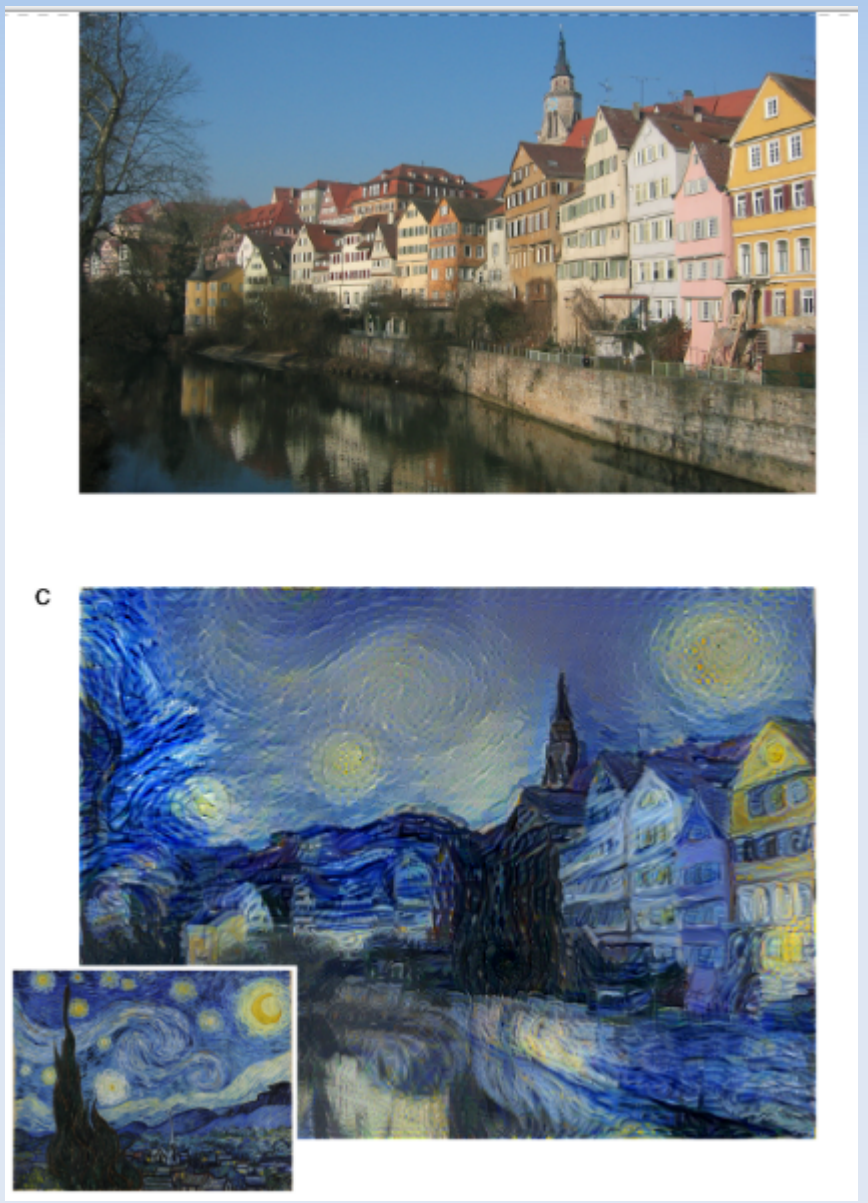
- Style transfer work on images
- Adversarial learning for Generative Adversarial Networks (GANs)

Inspirations

- Style transfer work on images
- Adversarial learning for Generative Adversarial Networks (GANs)

Related Work – Style Transfer

- One way to test whether a separate representation is learned for the form is to decode from a combination of meaning and form from different sentences.
- Our work here is most similar to the work on style transfer in computer vision:
  - *Gatys et al. 2015, Ulyanov et al. 2016, Johnson et al. 2016*
  - *Fu et al. 2017, Shen et al 2017 (text)*



Prior Work on Text Style Transfer

- Limited work on style transfer for text to date.
- Most approaches require aligned parallel corpora – based on machine translation (MT)
  - Statistical MT (Xu et al 2012), deep learning MT systems (Carlson et al 2017, Jhamtani et al 2017)
- Approaches that do not require parallel corpora
  - VAE-based approaches (Hu et al 2017, Mueller et al 2017), GAN-based approaches (Shen et al 2017, Fu et al 2017)
- **No prior work attempts to learn separate embeddings for the form (style) of each input – no learning of separate representations.**
  - Form (style) is treated as categorical or a single embedding is learned per style.

Inspirations

- Style transfer work on images
- Adversarial learning for Generative Adversarial Networks (GANs)

Related Work – Generative Adversarial Networks

- Proposed by Goodfellow in 2014, similar ideas in Schmidhuber's 1992 work on *predictability minimization.*

- Two neural networks, generator G and discriminator D.
- G's objective is to generate realistic-looking examples.
- D's objective is to discriminate between real and fake examples.

Related Work – Generative Adversarial Networks (GANs)

Intuition [Kristiadi, 2016]

- A money counterfeiting criminal
  - Produces fake money
- A cop
  - Tries to distinguish between fake and real money

- The criminal's goal:
  - Maximize the similarity between the fake money and the real money
- The cop's goal:
  - Detect counterfeit the money!

Related Work – Generative Adversarial Networks (GANs)

- Both the cop and the criminal are neural deep networks
- The cop is called “discriminator”
- The criminal is “generator”

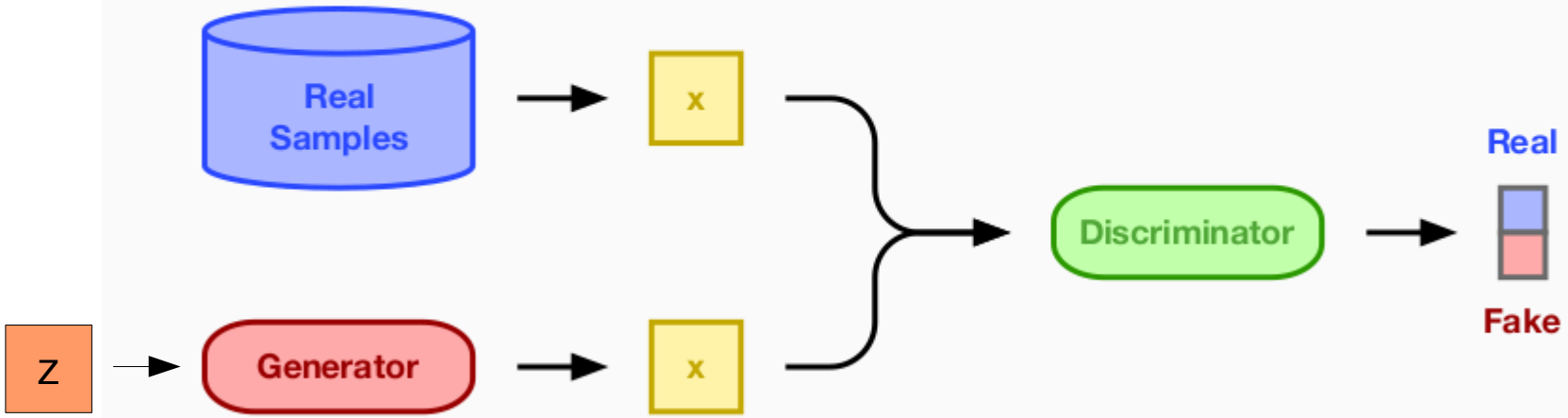


Figure 1: GAN architecture

Generative Adversarial Networks

- Two networks, generator G and discriminator D.
- Generator
  - Takes as input a random vector $z \sim P(z)$
  - Learns a mapping $\mathrm{G}(\mathrm{z}, \theta_g)$ into the data space.
- Discriminator
  - Takes as input either a real sample $\mathrm{x} \sim P_{real}$ or a sample $\hat{\mathrm{x}} \sim P_{gen}$ geneated by the generator G.
  - Outputs probability of x coming from the real data $\mathrm{D}(\mathrm{x}, \theta_d)$

Training GANs

- D is trained to discriminate between examples drawn from the real data $\mathbf{x} \sim P_{real}$ and examples $\hat{\mathbf{x}} \sim P_{gen}$ generated by G based on input noise from a (usually Gaussian) prior $z \sim P(z)$.
- G is trained to improve its generated examples to nudge D towards misclassifying generated examples as real.
- The goal is to reach an equilibrium where P_{real} and P_{gen} are close.
- Gradients from output of D are used to train G!

Training GANs

- Discriminator D is trained to
  - maximize the probability of classifying real data as real and
  - minimize the probability of classifying generated data as real

$$\max_D \; E_{x \sim P_{real}} \log D(x) + E_{x \sim P_{gen}} \log \left(1 - D(x) \right)$$

  - $D(x)$ is the probability of x being the real sample, so this maximizes the probability of real samples and minimizes the probability of generated samples.
- Generator G is trained to maximize the probability that D will assign an incorrect ”real” label to the generated samples:

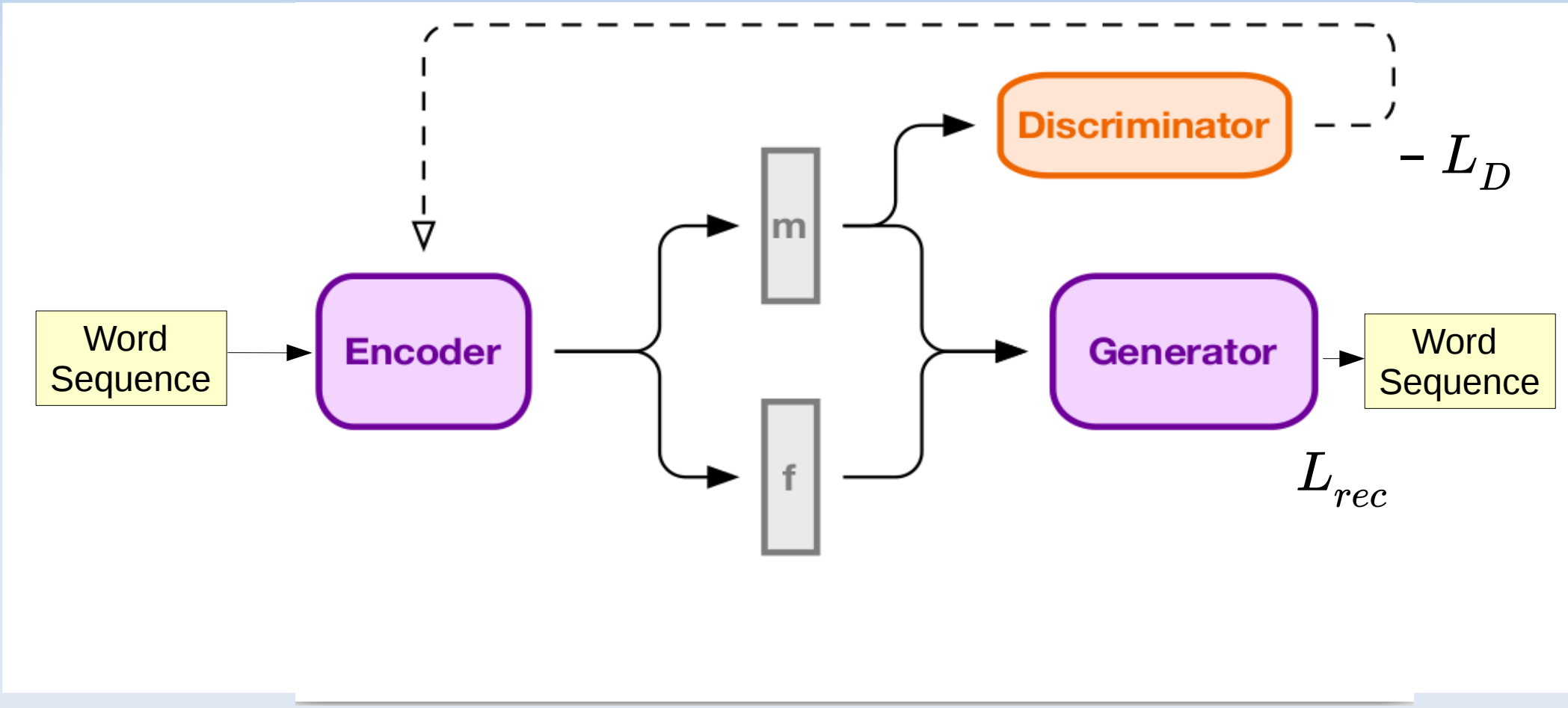
$$\max_G \; E_{z \sim P(z)} \log D((G(x))$$

- Trained with *alternating gradient update.*

DissoNet's Adversarial Training

- Want to learn a separate representation for the meaning and for the form (style) of every sentence
- Adversarial learning used to encourage the encoder/generator to produce dissociated representations.
- Supplemented by motivational learning!

DissoNet Architecture



Encoder

- Encoder is a recurrent neural network (RNN)
  - encodes an input sequence $\mathbf{x}$ into a hidden vector $\mathbf{h}$.

    $$\mathbf{h} = \mathbf{Encoder}(\mathbf{x})$$
- Two dense (fully connected) layers are used to compute the latent vectors for the form and the meaning:

  $$\mathbf{m} = \mathbf{tanh}(\mathbf{W}_m\mathbf{h} + \mathbf{b}_m)$$

  $$\mathbf{f} = \mathbf{tanh}(\mathbf{W}_f\mathbf{h} + \mathbf{b}_f)$$

Generator (Decoder)

- Generator (aka Decoder) G is also an RNN.
- Takes as input $\mathbf{m}$ and $\mathbf{f}$, concatenates them, and produces a vector $\mathbf{z}$ that captures both meaning and form:

$$\mathbf{z} = \mathbf{tanh}(\mathbf{W}_z[\mathbf{m}; \mathbf{f}] + \mathbf{b}_z)$$

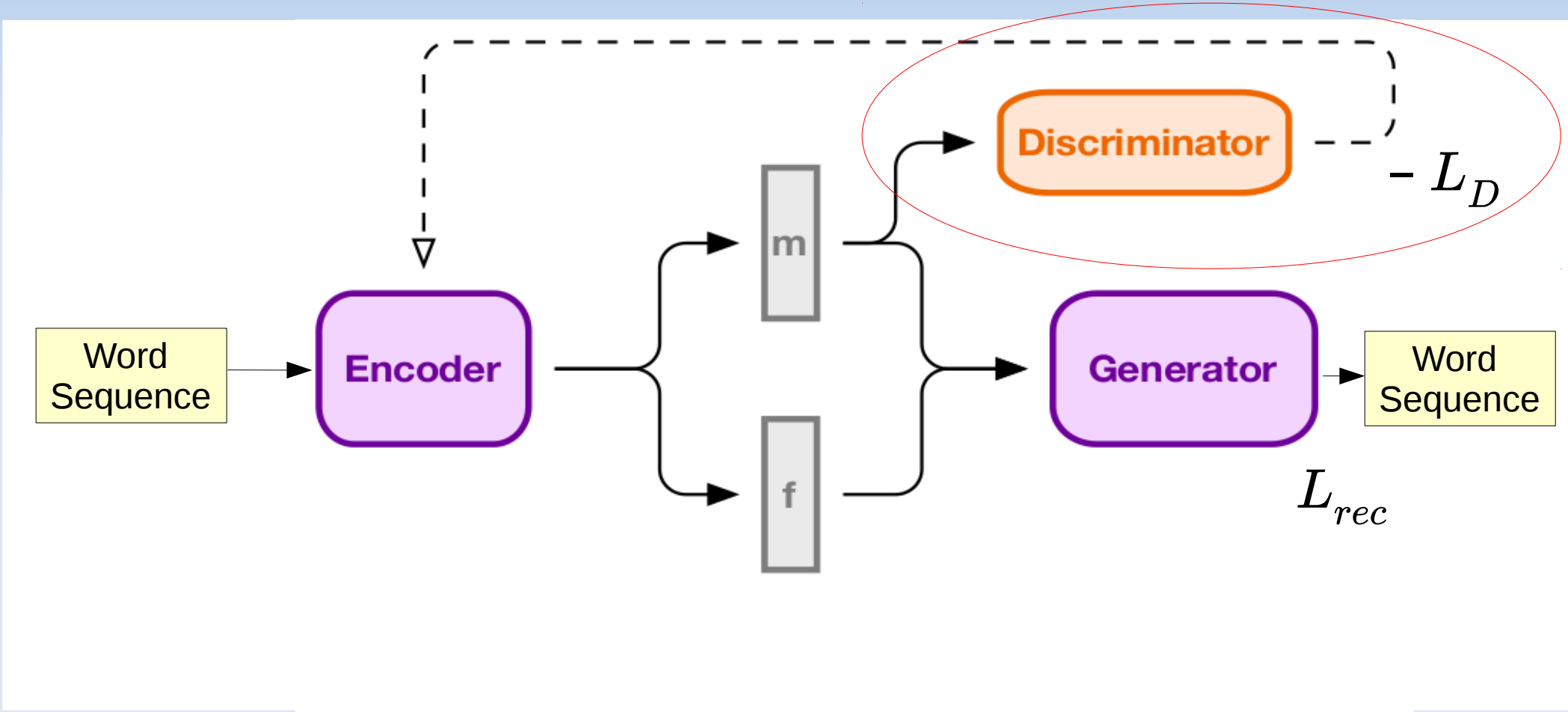
- An RNN is used to generate an output sequence based on $\mathbf{z}$, as in the standard decoder model.

Encoder and Generator Training

- The encoder and the generator are trained using a standard reconstruction loss
- $\mathcal{L}_{\text{rec}}(\boldsymbol{\theta}_E, \boldsymbol{\theta}_G) = \mathbb{E}_{x \sim X^a}[-\log p(\hat{x}|x)] + \mathbb{E}_{x \sim X^b}[-\log p(\hat{x}|x)]$
  - $\boldsymbol{\theta}_E$ are the parameters of the encoder
  - $\boldsymbol{\theta}_G$ are the parameters of the generator

– maximize the probability of correct answer for each example in the training data

DissoNet Architecture



Discriminator

- Representation of the meaning $\mathbf{m}$ should not contain any information about the form $\mathbf{f}$.
- We achieve this with an adversarial approach.
- Discriminator $\mathbf{D}$ (**a neural network consisting of several fully connected layers**) is trained to predict the $\mathbf{form}$ of the sentence based on the $\mathbf{meaning}$ vector for the sentence that it receives as input.

Discriminator

- Discriminator tries to predict the form from the meaning vector.
- Loss function is motivated by the Wasserstein modification to the GAN loss:

$$\mathcal{L}_D = E_{x \sim p(F1)}[D(E_m(x))] - E_{x \sim p(F2)}[D(E_m(x))]$$

- Wasserstein GAN loss:
  - Non-linearity (sigmoid) removed from the output layer, so we're outputting real numbers, rather than probabilities.
  - Discriminator is encouraged to output large negative values for form f_1 large positive values for form f_2.
- This loss is applied to the parameters of the encoder.

Training Procedure

- Two losses combined
  - Discriminator loss $\mathcal{L}_D$
  - Encoder-Generator Loss $\mathcal{L}_{EG} = \mathcal{L}_{rec} - \mathcal{L}_D$
- Training procedure for each batch of the data:
  - Train the discriminator with $\mathcal{L}_D$
  - Train the encoder and the generator with $\mathcal{L}_{EG}$

Motivator

- Our experiments showed that it is enough to have just the discriminator and the adversarial loss to force the model to learn to dissociate the form and the meaning.
- Albanie et al, SIGBOVIK 2017: the prevalence of the adversarial network-on-network violence is excessive and uncalled for….

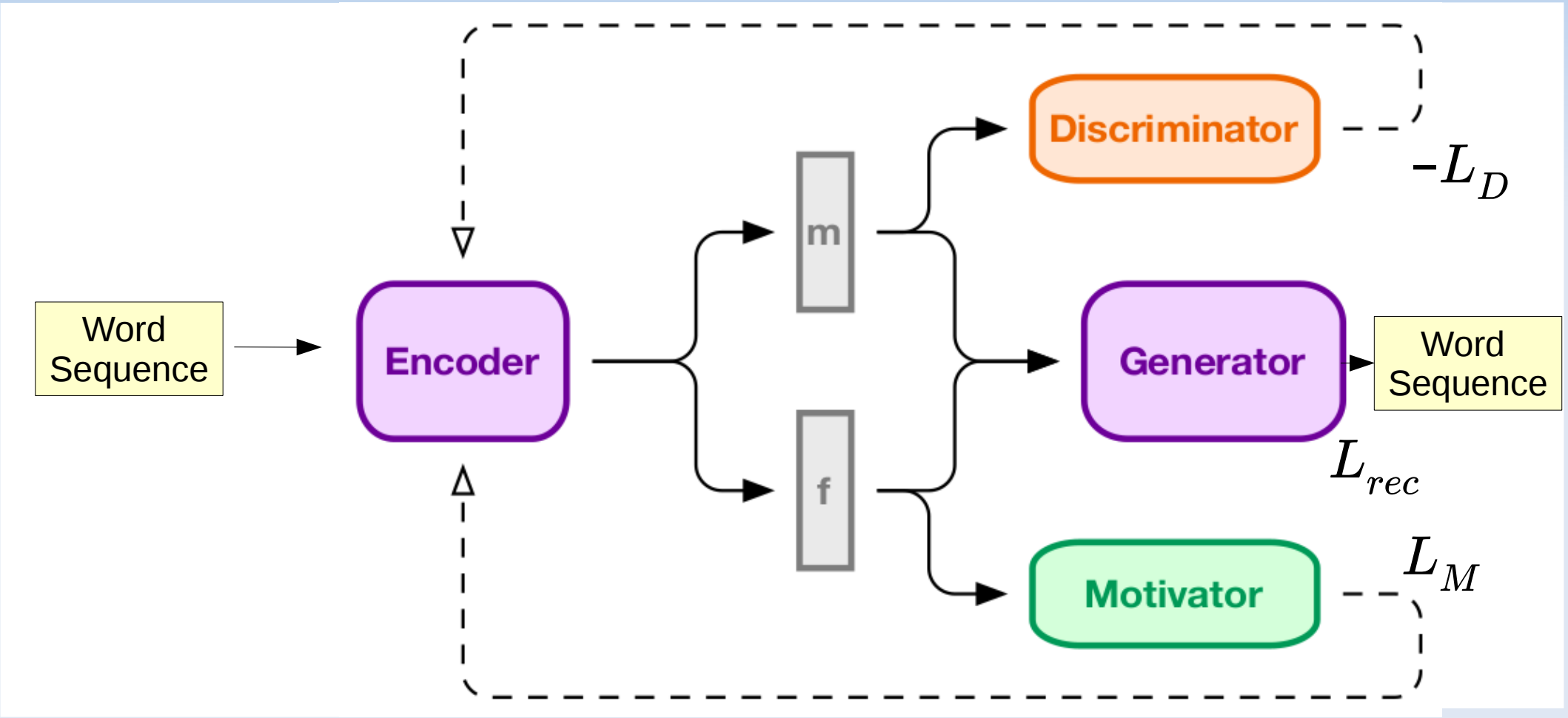
  ”Generative Unadversarial Networks (GUNs): train two models: a generator G that does its best to capture whichever data distribution it feels it can manage, and a motivator M that helps G to achieve its dream...”

Motivator

- We took this seriously!

- Perhaps, using the principle of carrot and stick will achieve better results?
- To test this, we introduce a motivator and a motivational loss to achieve a better dissociation.

DissoNet Architecture



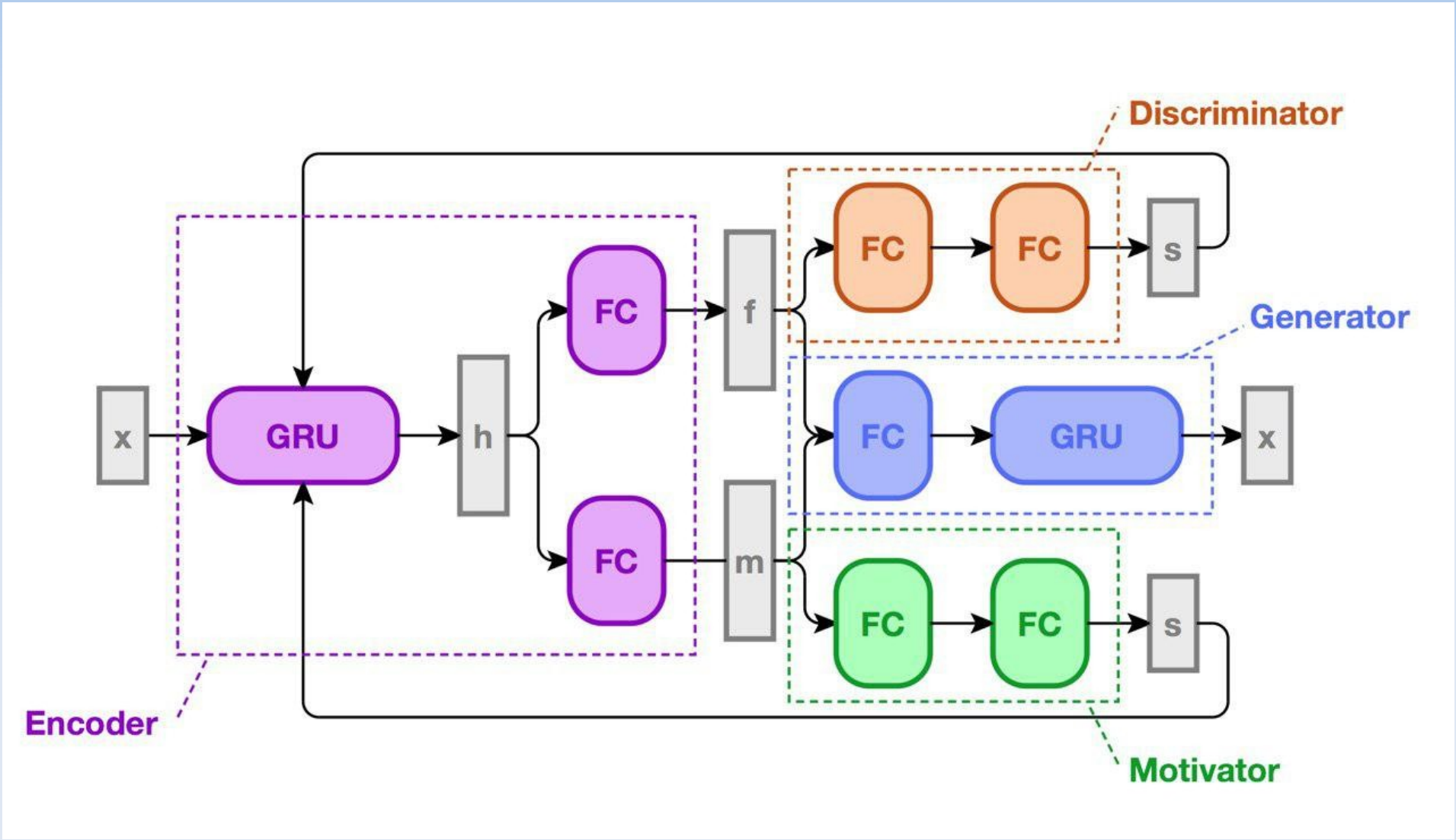
Motivator

- As the discriminator, motivator M is a classifier which learns to classify **form** $\mathbf{f}$ of the input sentence.
- Its input is not the meaning vector, but **the form vector** $\mathbf{f}$
- The discriminator *discourages* the encoder from putting any information about the form (style) into the meaning vector
- The motivational loss *encourages* the encorder to put the information about the form **into the form vector** by minimizing $\mathcal{L}_M$
- This loss is applied to the parameters of the encoder

Training Procedure

- Three losses combined
  - Discriminator loss $\mathcal{L}_D$
  - Discriminator loss $\mathcal{L}_M$
  - Encoder-Generator Loss $\mathcal{L}_{EG} = \mathcal{L}_{rec} - \mathcal{L}_D + \mathcal{L}_M$
- Training procedure for each batch of the data:
  - Train the discriminator with $\mathcal{L}_D$
  - Train the motivator with $\mathcal{L}_M$
  - Train the encoder and the generator with $\mathcal{L}_{EG}$

Detailed Architecture



Evaluation

- We compare out work to the most recent published model by Fu et al 2017
  - Does not treat style as binary, but trains a style representation vector (similar to our goal)
  - However, trains **a single representation for each style.**
- Prior studies experimented on different datasets. We reimplement the model from Fu et al 2017 and apply it to 4 different datasets used in literature.
  - Original Shakespeare vs. Contemporary English version
  - Newspaper headlines vs. Scientific article titles
  - Negative sentiment vs. Positive sentment – in Amazon reviews, in Yelp reviews.

Evaluation

Dataset set	Training			Validation			Test		
	Class 1	Class 2	Total	Class 1	Class 2	Total	Class 1	Class 2	Total
Shakespeare	13130	13400	26530	1454	1546	3000	1511	1489	3000
Headlines	93497	93199	186696	5048	4952	10000	5041	4959	10000
Yelp	173057	242815	415872	4194	5806	10000	4043	5957	10000
Amazon	276422	278146	554568	4970	5030	10000	4954	5046	10000

Table 1: Datasets size statistics.

Evaluation
(always tricky for generation)

- Fu et al., 2017 proposed to use “transfer strength” and “content preservation”
- **Transfer strength**
  - Train a classifier to detect the form.
  - Encode a sentence, decode with a different form.
  - Pass the generated sentence trough the classifier.
  - Count how often the classifier says that it is the target form.
- **Content preservation**
  - A cosine similarity between concatenated max,min,mean pooling of word embeddings in the output.
  - Reflects the *similarity of meaning* between two sentences.
  - Per Fu et al 2017, consistent with human judgment.

Evaluation

- Fu et al 2017 found that **transfer strength** and **content presevation** were in an inverse relationship
  - This makes sense: if the meaning and style are not well separated, the more you change the style, the more you would change the content (i.e. meaning).
- Transfer strength should to be high
- Content preservation should be high, but **very high cosine values likely reflect the same, rather than semantically similar words** being used.
  - Unrealistic when you change, e.g., from formal to informal style, etc.

Results

Dataset	Model	Form/Meaning Classifier	Transfer Strength	Content Preservation
Shakespeare	[Fu et al., 2017]	0.802	0.381	0.963
	DissoNet		0.590	0.838
	DissoNet + Motivator		0.587	0.850
Headlines	[Fu et al., 2017]	0.991	0.623	0.881
	DissoNet		0.999	0.783
	DissoNet + Motivator		0.997	0.787
Yelp	[Fu et al., 2017]	0.971	0.166	0.989
	DissoNet		0.906	0.815
	DissoNet + Motivator		0.935	0.813
Amazon	[Fu et al., 2017]	0.807	0.252	0.988
	DissoNet		0.603	0.865
	DissoNet + Motivator		0.729	0.852

Table 2: The results of the evaluation on different datasets and the corresponding accuracy of the form/meaning classifier

Motivational Training (t-SNE)

Visualization of of 1000 random sentences

Green points – news headlines, red – titles of scientific articles

Figure 9: Without motivator

Motivational Training (t-SNE)

Visualization of of 1000 random sentences

Green points – news headlines, red – titles of scientific articles

Figure 10: With motivator

Early Modern English vs. Contemporary English

Decoding from Early Modern English to contemporary English

Aye, sir	→	Yes, sir
Fare thee well, my lord	→	Fare you well, my lord

Decoding from contemporary to Early Modern English

This guy will tell us everything	→	This man will tell us everything.
I've done no more to Caesar than you will do to me .	→	I have done no more to Caesar than you shall do to me.

Swapping Form and Meaning

sentence 1	how can i reset a motherboard 's bios password ?
sentence 2	what should i do to think positive ?
m_1, s_2	what should i do if i forgot bios password ?
m_2, s_1	how can i get self-esteem and confidence ?

Additional Evaluation: Paraphrase

- Microsoft Research Paraphrase Corpus (MSRP);
- Task: given a pair of sentences, classify them as paraphrases or not paraphrases.
- Classification accuracy:
  - Autoencoder baseline – 74.68
  - DissoNet – 81.38
  - InferSent – 83.17

  (supervised SOTA system, trained on NLI)

DissoNet Contributions

- Competitive performance on several transfer datasets.
- Treats the differences between the linguistic form (style) as continuous, rather than categorical.
  - Each sentence has a separate meaning representation and form (style) representation – unlike the models that treat style as binary variable or create a single representation for each style.
  - In line with the reality of language use – there are different degrees of overlap between the language in different social registers or different diachronic slices.
  - Decodes directly from the form and meaning representations of a sentence in the embedding space – unlike models that supply style information to the decoder at every step.
- Learns style representations without parallel data – unlike for MT-based models.

Other Separable Representations

- Can we learn a separate representation for a personality of a chatbot?
- Personality is a learned function that defines how you integrate experience in memory and synthesize reactions.
- You need to learn this function *in addition* to learning how to work with long-term (and short-term) memory
  - Access, use, and acquire knowledge
  - Resolve long-distance dependencies for text understanding and enforcing coherence.

This Talk

- Topic of the day #1. Learning a modular representation of linguistic input that mimics the functional specialization of language-processing regions in the brain.
  - Our solution ’18: *DissoNet* model with Adversarial / Motivational training (Romanov, Rumshisky et al, under review)
- Topic of the day #2. Extracting event timelines from text – what happened, when and in what order?
  - Our solution ’17: A uniform set of dependency-based LSTMs (Meng, Romanov & Rumshisky, EMNLP 2017)
  - Our solution ’18: *Global Context Layer* model with updateable memory to capture context (Meng & Rumshisky, ACL 2018)

This Talk

- Topic of the day #1. Learning a modular representation of linguistic input that mimics the functional specialization of language-processing regions in the brain.
  - Our solution ’18: *DissoNet* model with Adversarial / Motivational training (Romanov, Rumshisky et al, under review)
- Topic of the day #2. Extracting event timelines from text – what happened, when and in what order?
  - Our solution ’17: A uniform set of dependency-based LSTMs (Meng, Romanov & Rumshisky, EMNLP 2017)
  - Our solution ’18: *Global Context Layer* model with updateable memory to capture context (Meng & Rumshisky, ACL 2018)

Temporal Information Extraction for Question Answering

- Meng, Romanov and Rumshisky, EMNLP 2017
- Meng & Rumshisky, ACL 2018

Temporal Information Extraction

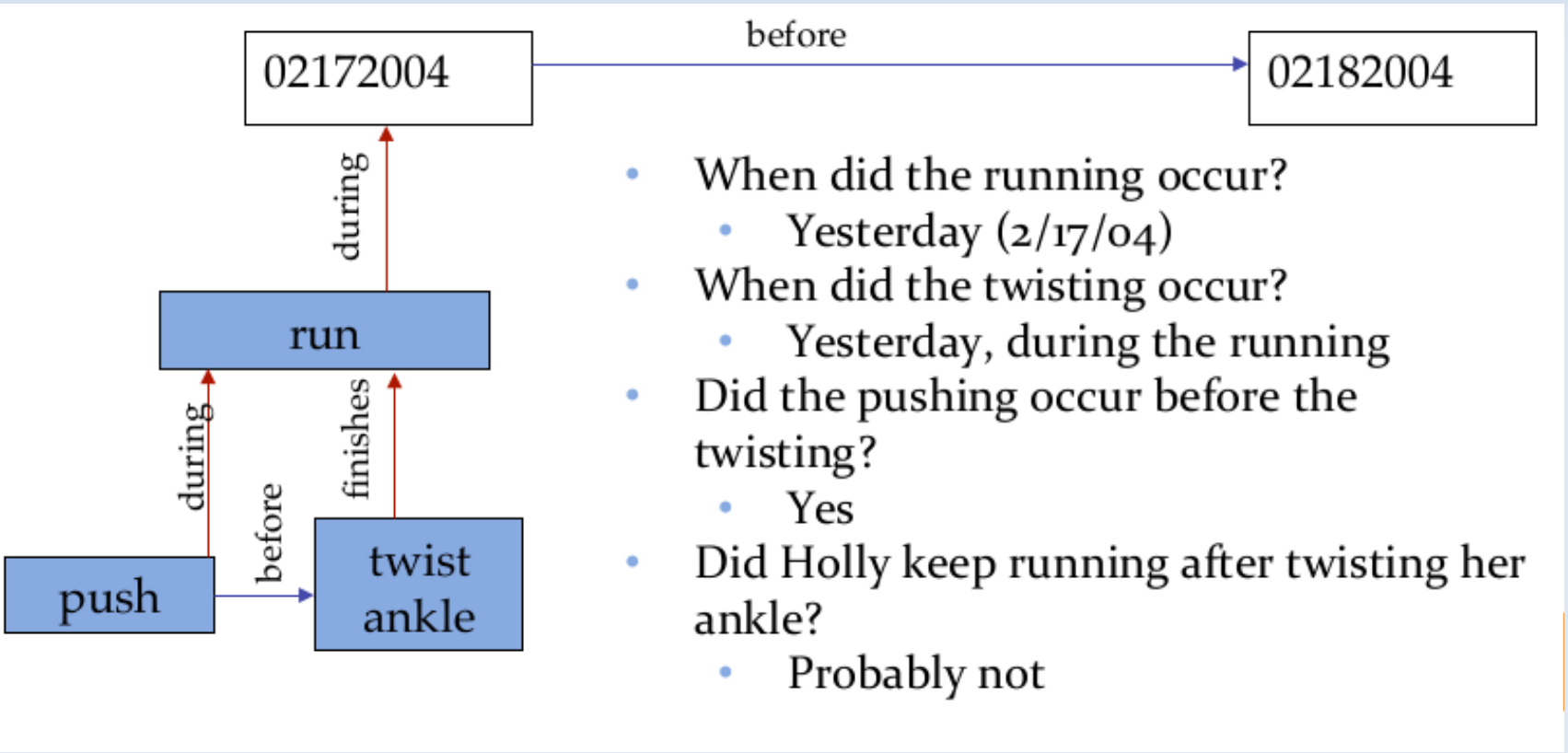
- Extracting event timelines from text – what is the order and timing of events described in this text?
  - what happened, when, and in what order
- Need this information to reason about events, to answer questions, to summarize what happened (*temporal reasoning, question answering, summarization*)

Motivation: Question Answering

- QA systems need to resolve temporal expressions in questions and answers:
  - Was there a surgical site infection post-surgery?
  - Did the patient stop vomiting after Thorazine?
  - How many Iraqi civilian casualties were there in the first week of the U.S. invasion of Iraq?
- Q: When did the Berlin Wall fall?
- A: Thursday
  - East German border workers began dismantling the Berlin Wall at the historic Brandenburg gate on Thursday night to make a new crossing.

Events, Times, TLINKs

- February 18, 2004
  - Yesterday, Holly was running a marathon when she twisted her ankle. David had pushed her.



- When did the running occur?
  - Yesterday (2/17/04)
- When did the twisting occur?
  - Yesterday, during the running
- Did the pushing occur before the twisting?
  - Yes
- Did Holly keep running after twisting her ankle?
  - Probably not

Prior Art

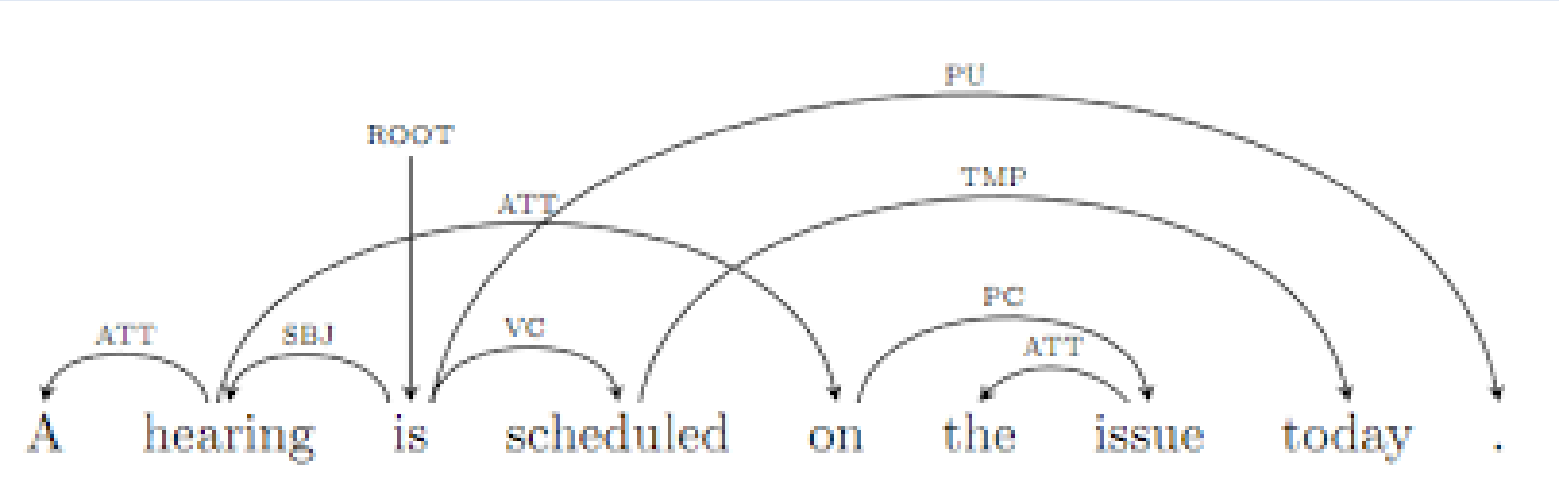
- Common solutions involve statistical or rule-based event and time detection, followed by pairwise labeling of entities with different temporal relations.
- This potentially creates conflicts in the resulting temporal graph, so transitive closure is run over the graph of temporal relations, then cycles / conflicts detected and eliminated.
- Conflict resolution relies on e.g. confidence scores.

Prior Art

- Temporal relations are often non-local, i.e. two events may be mentioned in two different parts of text
- Because temporal relations are often non-local, until recently there have been no deep learning solutions to this problem.

Our Solution – EMNLP 2017

- A relatively simple neural network architecture to capture long-distance dependencies needed to identify temporal relations.
- We used dependency path sequences, rather than sequences of tokens as input to several nearly-identical RNN-based sequence models.

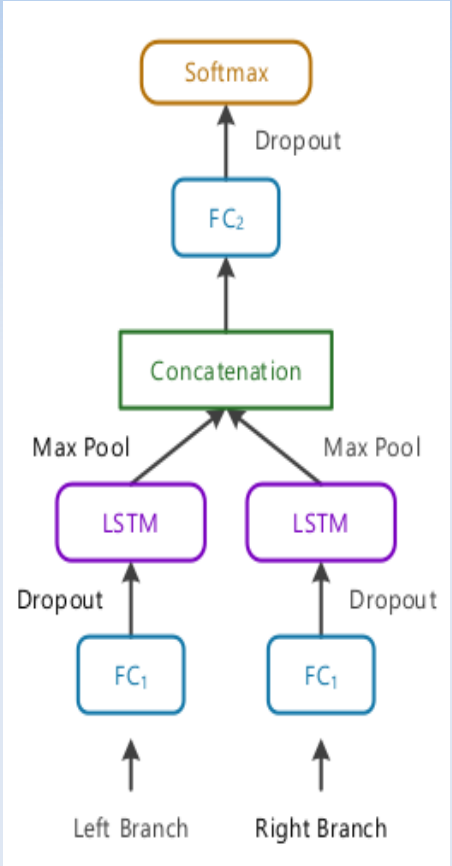


Hearing, is → RNN #1
scheduled, is → RNN #2

- Used path to the root for non-local dependencies

TLINK Model Architecture

- The shortest path has left and right branches, divided by the least common ancestor.
- Word embeddings of each branch are fed into the corresponding branch of the RNN.
- RNN outputs a probability distibuion over TLINK categories (including no-link)
- Completely uniform treatment of different entity pairs (event-event, event-timex, event-DCT)



Three Model Clones

- **Intra-sentence model**
  - Two branches: each branch is a dependency path from an entity to the least common ancestor of the entity pair.
- **Cross-sentence model (consecutive sentences)**
  - Branches are paths from each entity to root of its sentence.
- **DCT model**
  - Two branches are the paths from entity to root, and from the root to the entity.

Evaluation on Question Answering

- Used QA-TempEval (SemEval 2015 Task 5) data and evaluation methods.
  - The training set of 276 annotated news articles
- The test set contains unannotated files in three genres:
  - news articles (10 docs), Wikipedia articles (10 docs), and blogs entries (8 docs).
- Evaluation is done with a QA toolkit which has yes/no questions about event pairs, or event-time pairs.

Results on QA-TempEval

System	Prec	Rec	F1	% answered	# correct
News, 99 questions total					
HLT-FBK	0.43	0.29	0.35	69	29
TEA	0.61	0.44	0.51	73	44
Wikipedia, 130 questions total					
HLT-FBK	0.62	0.36	0.46	58	47
TEA	0.62	0.44	0.51	71	57
Blogs, 65 questions total					
HLT-FBK	0.34	0.2	0.25	58	13
TEA	0.43	0.2	0.27	46	13

- HLT-FBK was the overall winner of TempEval shared task in 2015.

$$\text{coverage} = \frac{\#\text{answered}}{\#\text{questions}}, \text{precision} = \frac{\#\text{correct}}{\#\text{answered}}$$

$$\text{recall} = \frac{\#\text{correct}}{\#\text{questions}}, \text{f1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Results on TimeBank Dense

- ”Intrinsic” evaluation on a standard benchmark corpus containing human annotation a subset of the documents in QA-TempEval
- **No modifications to the models!**
  - Optimized for the QA setup, with over-detection of events, etc.

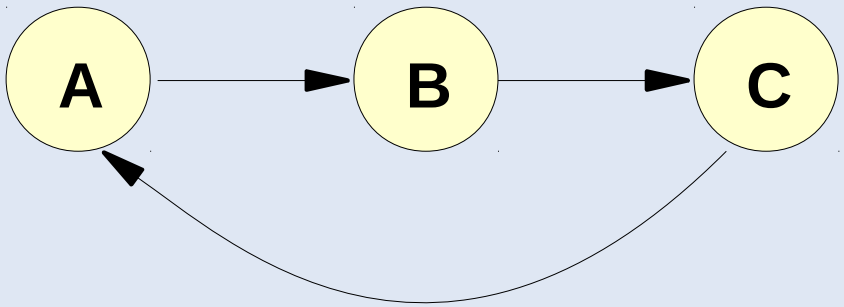
system	ClearTK	NavyT	CAEVO	CATENA	TEA-Dense	
					uniform	tuned
F1	0.447	0.453	0.507	0.511	0.505	0.519

Summary

- A relatively simple streamlined LSTM-based architecture for extraction of temporal relations
- Optimized for the extrinsic evaluation via QA, performed equally well on intrinsic evaluation.
- Uniform treatment of all types of entity pairs.
- Uniform treatment of entity pairs regardless of their location in text.

PROBLEM: Pairwise Decisions Introduce Conflicts

- Results of the individual models are combined, transitive closure applied to infer relations, then pruning to resolve conflicts / loops.
- Relations TLINKs with low probability scores are eliminated until no conflicts exist.



A before B
B before C
C before D

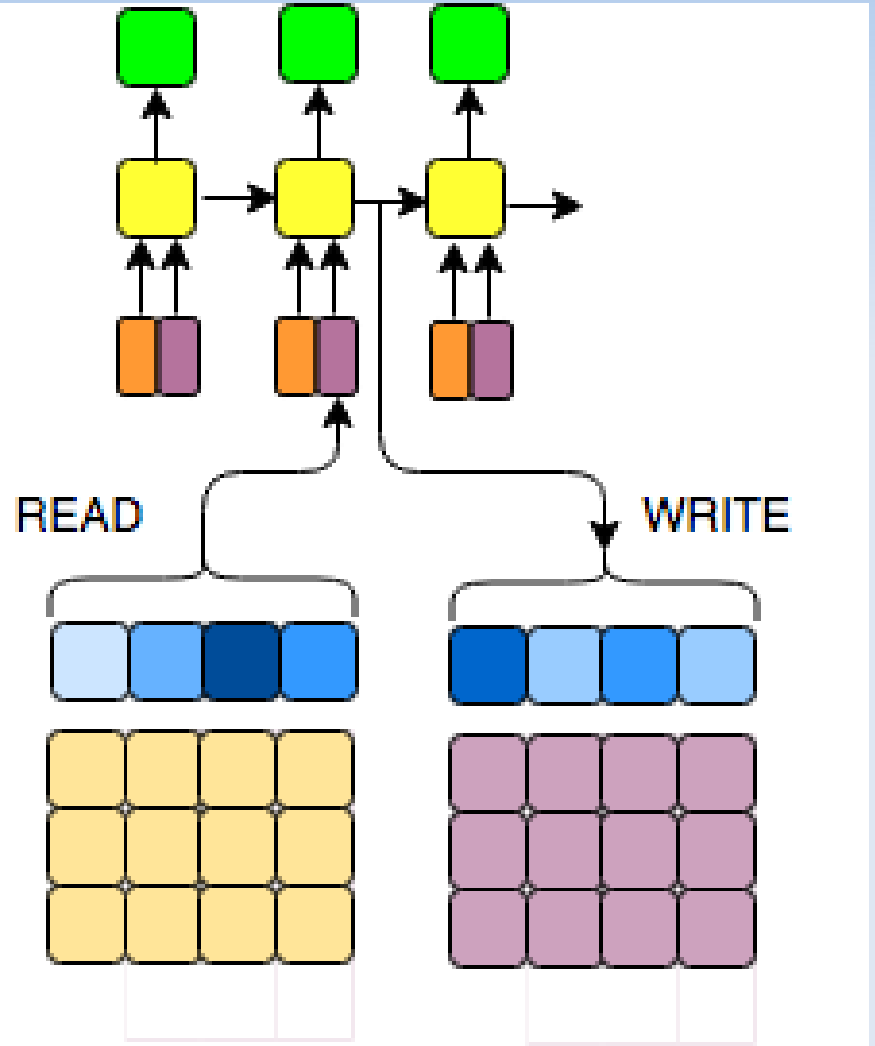
How Good is This?

- When humans read text, they do not locally order event pairs and then form a global picture of the timeline that reconciles conflicts.
- Instead, we understand relations as the relevant pieces of text are processed, possibly looking in the broader context to resolve ambiguities.
- The resolved relations are stored in our memory as “context” for further processing and later evidence suggests our early interpretation is wrong, we can correct it.
- Can we emulate this process and get rid of postprocessing the timegraph?

Memory Augmented Neural Networks with Updatable Memory

- NTM: Neural Turing Machine
- DNC: Differentiable Neural Computers
- D-NTM: Dynamic Neural Turing Machines
- TARDIS: Temporal Automatic Relation Discovery in Sequences
- **Our solution:**

  GCL: Global Context Layer



Our Solution (2018)

- Remove post-processing steps in conflict resolution, but integrate global information about existing links into the model.
- Solution: Global Context Layer (GCL).
  - Uses an external memory component and an attention-driven addressing mechanism (inspired by the Neural Turing Machine model introduced by Graves et al 2014)
- GCL model keeps an updated representation of full context, as pairs of entities are read in for pairwise classification.

System Overview

- The system processes batches of entity pairs in narrative order.
- The output is batches of labels.

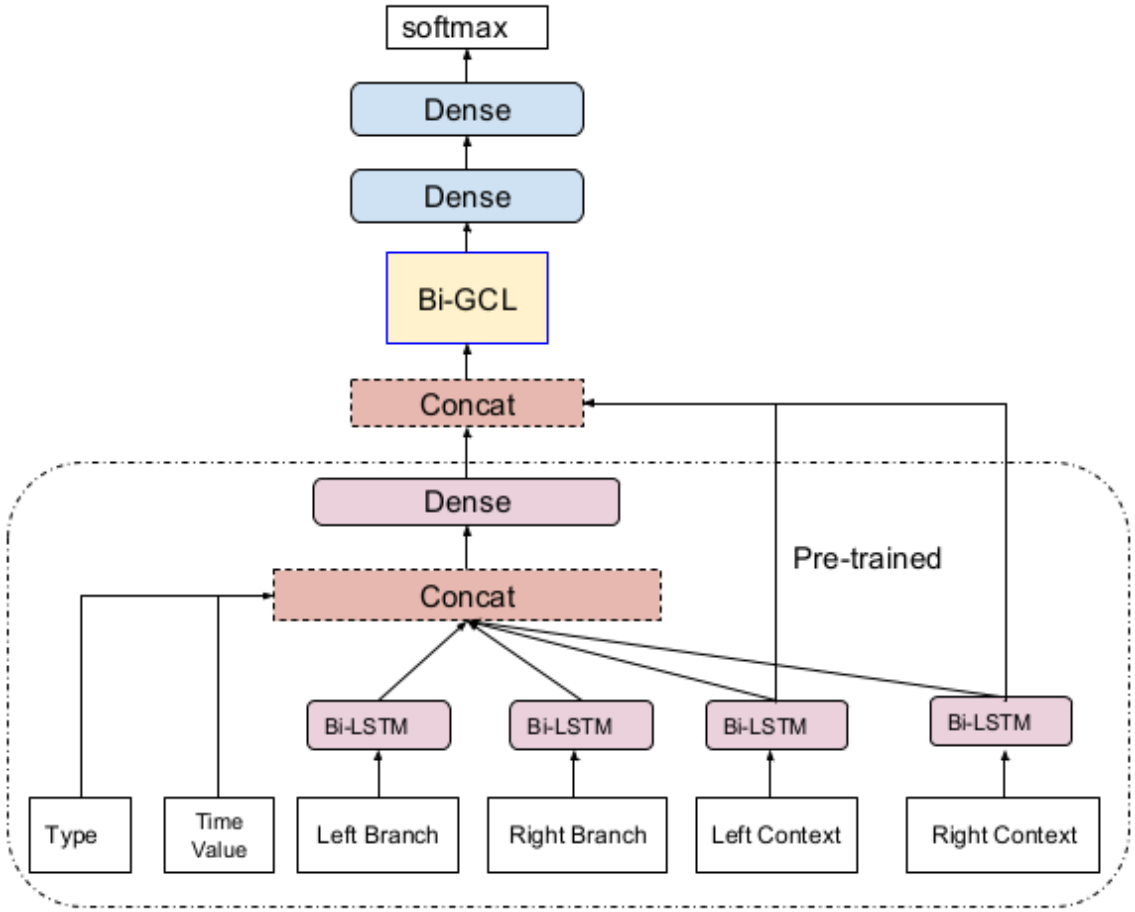


Figure 1: System overview. Originally, the pre-trained system has one more dense layer and an output layer, but they are truncated before combination. The max pooling layers on top of each Bi-LSTM layers are omitted here.

Model Input & Workflow

- There are two sets of inputs for each entity pair
  - Words in a context window around entities
  - Shortest dependency path between entities on syntactic trees

Model Input & Workflow

- There are two sets of inputs for each entity pair
  - Words in a context window around entities
  - Shortest dependency path between entities on syntactic trees
- Context for each entity is processed by RNN (Bi-LSTM) layers to get a unique representation for each entity mention (”keys”)

Model Input & Workflow

- There are two sets of inputs for each entity pair
  - Words in a context window around entities
  - Shortest dependency path between entities on syntactic trees
- Context for each entity is processed by RNN (Bi-LSTM) layers to get a unique representation for each entity mention (”keys”)
- Dependency paths are fed into the pairwise classifier, producing **an embedding that characterizes their temporal relationship**.

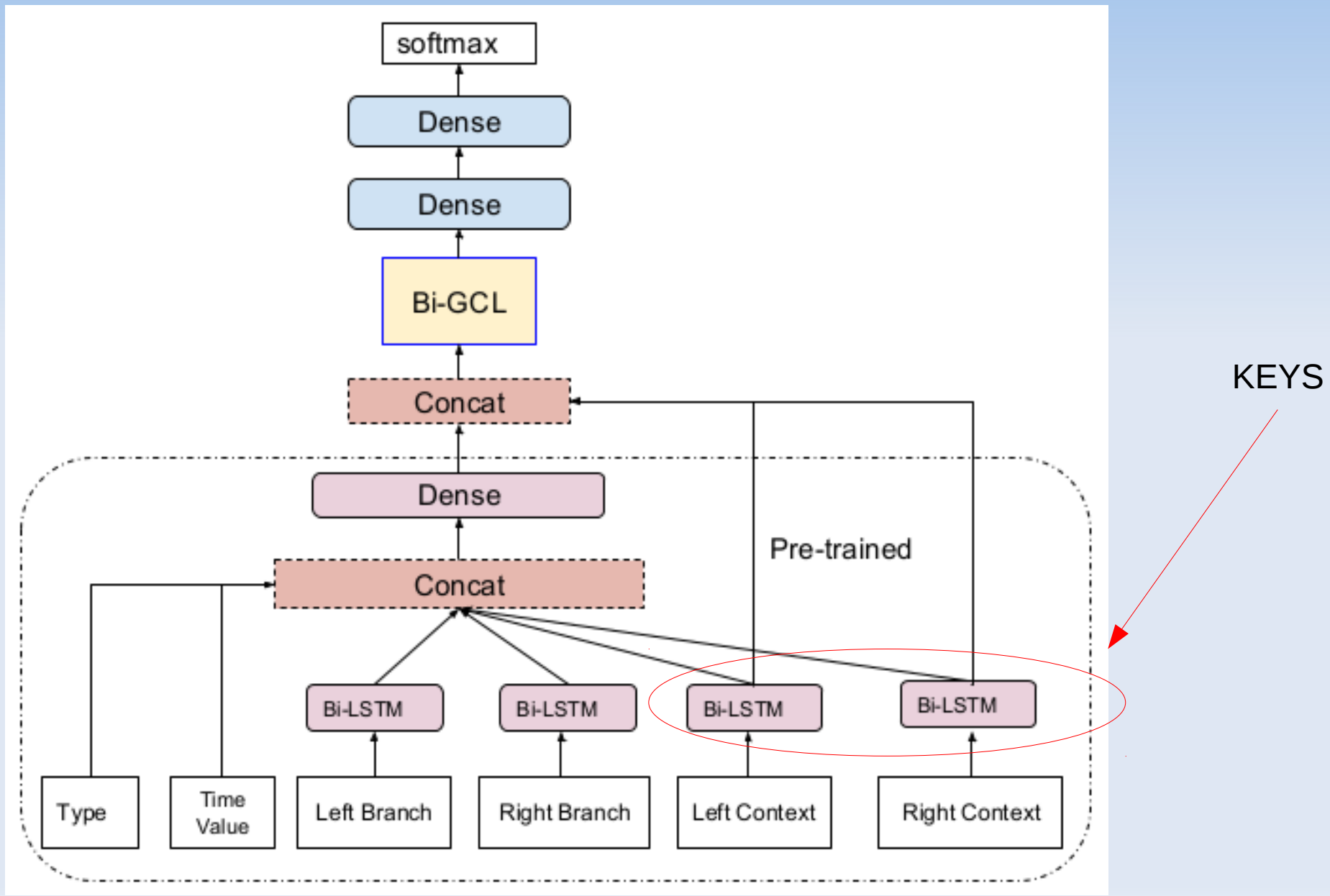
Model Input & Workflow

- There are two sets of inputs for each entity pair
  - Words in a context window around entities
  - Shortest dependency path between entities on syntactic trees
- Context for each entity is processed by RNN (Bi-LSTM) layers to get a unique representation for each entity mention (”keys”)
- Dependency paths are fed into the pairwise classifier, producing **an embedding that characterizes their temporal relationship**.
- This embedding, along with **the hidden GCL state** at previous timestep, and **retrieved memory contents** is used as input to the controller network, which produces the output layer.

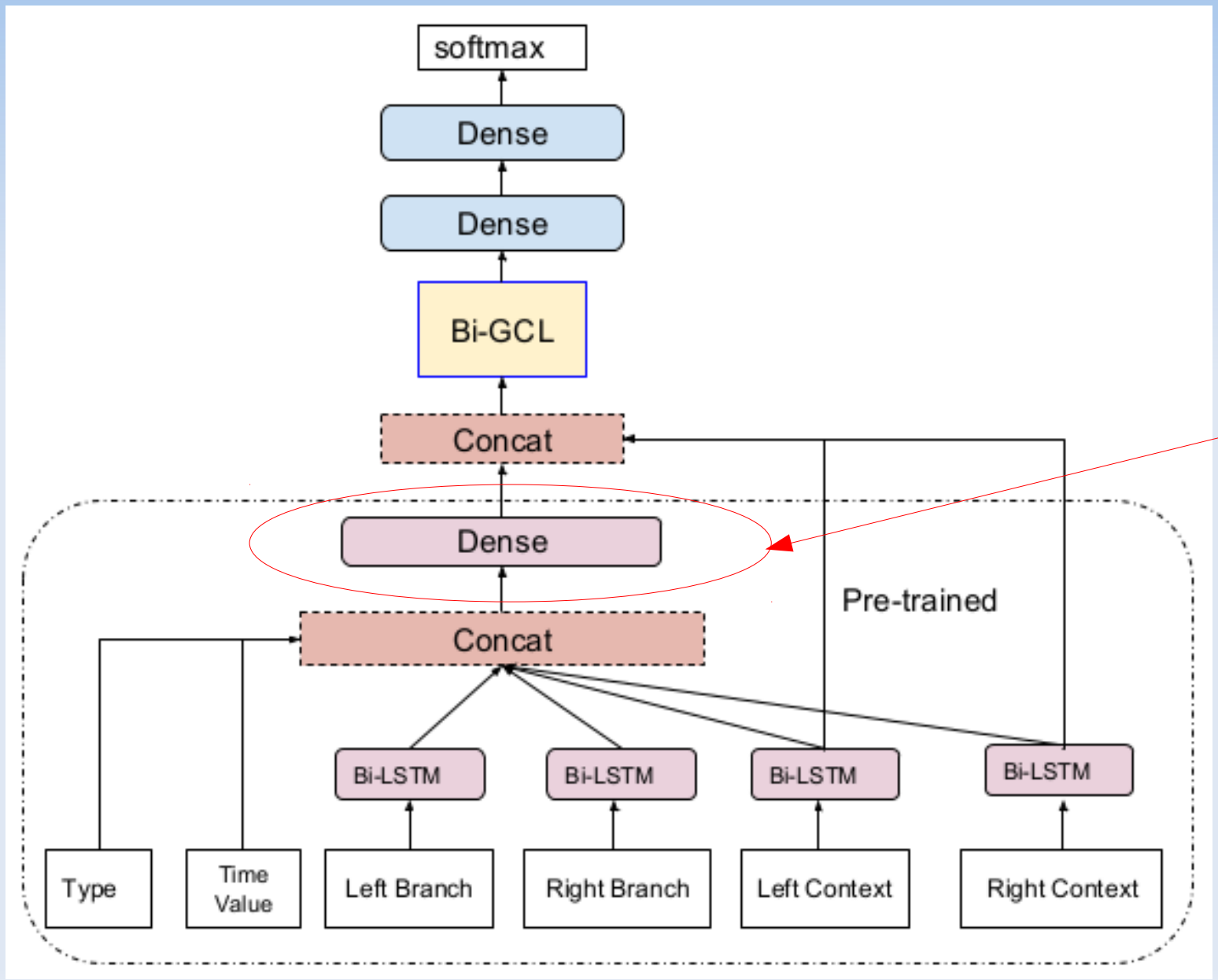
Model Input & Workflow

- There are two sets of inputs for each entity pair
  - Words in a context window around entities
  - Shortest dependency path between entities on syntactic trees
- Context for each entity is processed by RNN (Bi-LSTM) layers to get a unique representation for each entity mention (”keys”)
- Dependency paths are fed into the pairwise classifier, producing **an embedding that characterizes their temporal relationship**.
- This embedding, along with **the hidden GCL state** at previous timestep, and **retrieved memory contents** is used as input to the controller network, which produces the output layer.
- The hidden layer is written to memory and at the same time softmax is applied to it to produce classification.

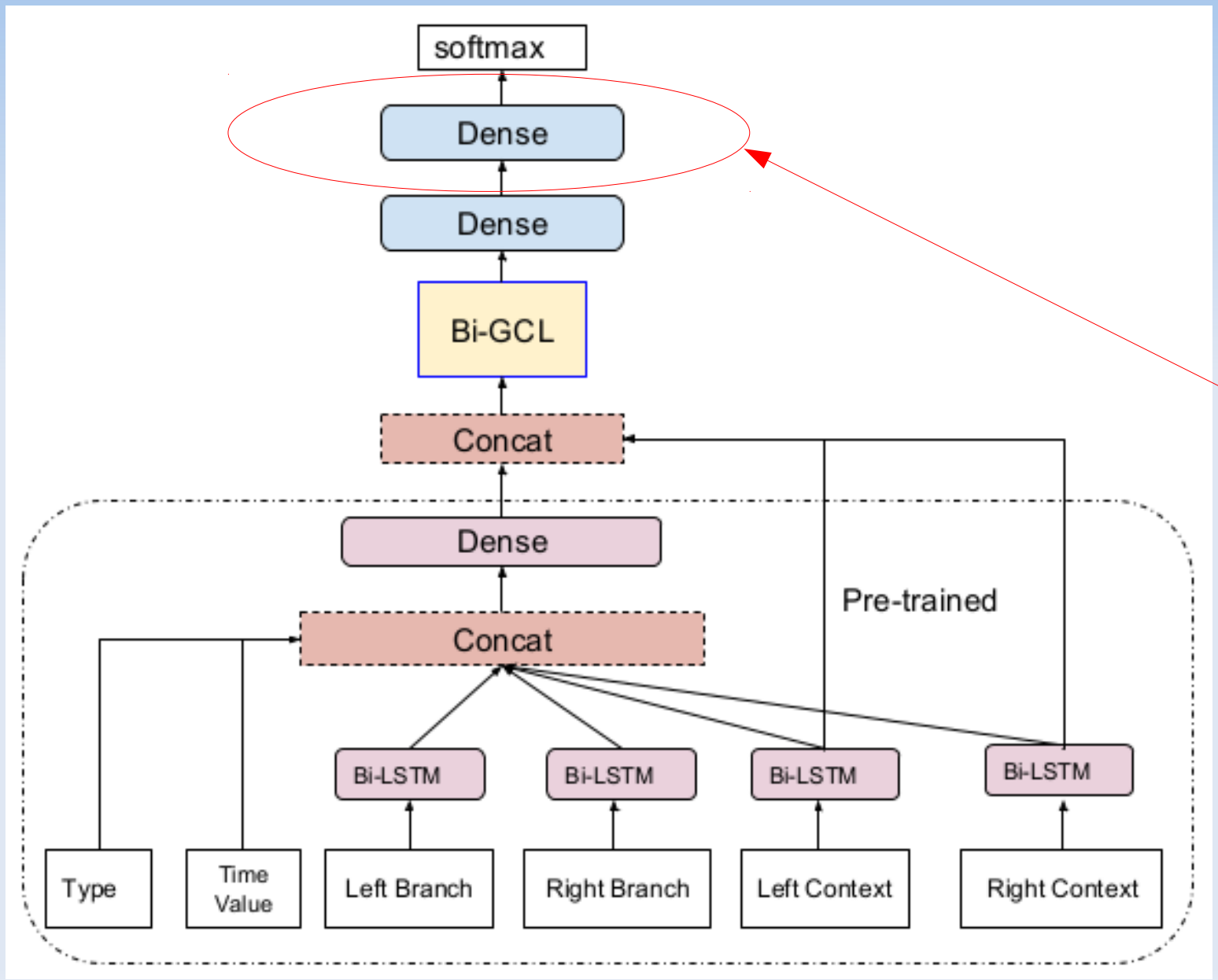
GCL Model


softmax
Dense
Dense
Bi-GCL
Concat
Dense
Concat
Pre-trained
Bi-LSTM
Bi-LSTM
Bi-LSTM
Bi-LSTM
Type
Time Value
Left Branch
Right Branch
Left Context
Right Context
KEYS

GCL Model


softmax
Dense
Dense
Bi-GCL
Concat
Dense
Concat
Pre-trained
Bi-LSTM
Bi-LSTM
Bi-LSTM
Bi-LSTM
Type
Time Value
Left Branch
Right Branch
Left Context
Right Context
RELATION Embedding

GCL Model

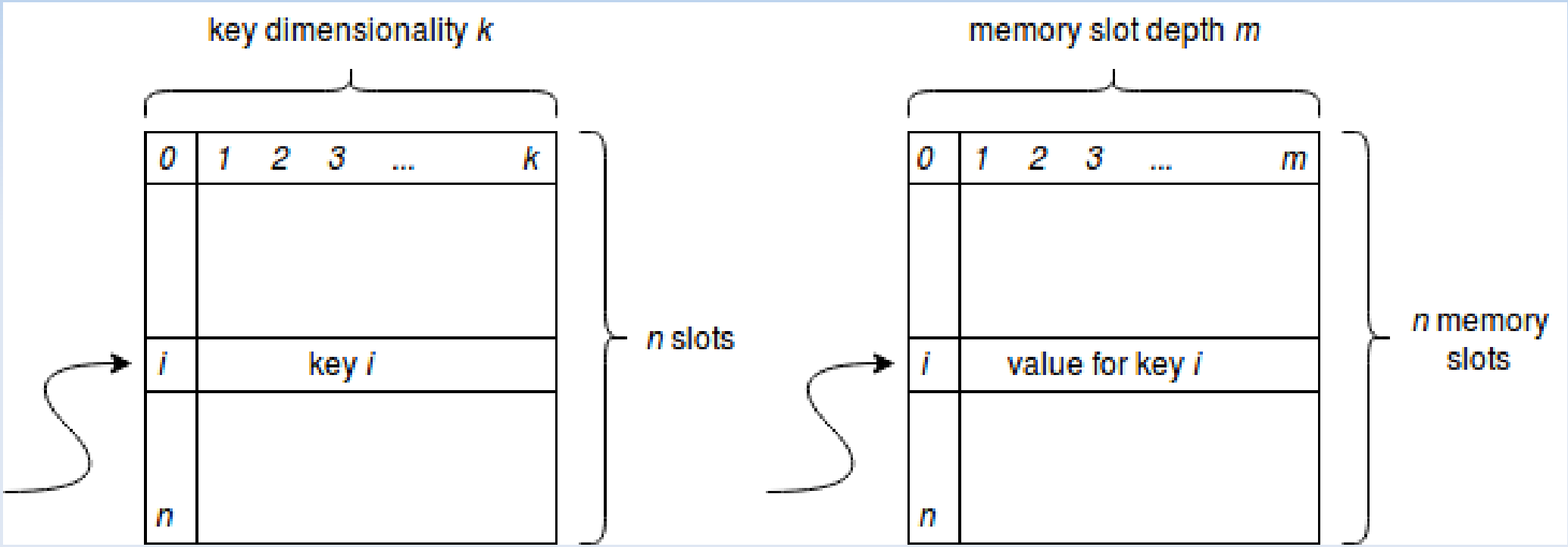


Written to memory for the current pair

Workflow

- Entity representations serve as ”keys” into the memory.
  - Keys are used to access GCL, where the old content (’value”) is retrieved and the new content stored.
- Memory location(s) with most similar key(s) are accessed and **their contents provided to the output layer along with the current input for the pair** being processed.
- The output layer produced by the model at the current timestep is saved as the content (”value”) for the current pair – and is written to the corresponding memory locations.

GCL Memory Structure



GCL Model

Addressing

Representations of input entity pairs ($\mathbf{e_1}$, $\mathbf{e_2}$) are compared with keys in GCL memory [($\mathbf{e_{M1}}$, $\mathbf{e_{M2}}$) is one key – iteration over memory slots containing pair representations:

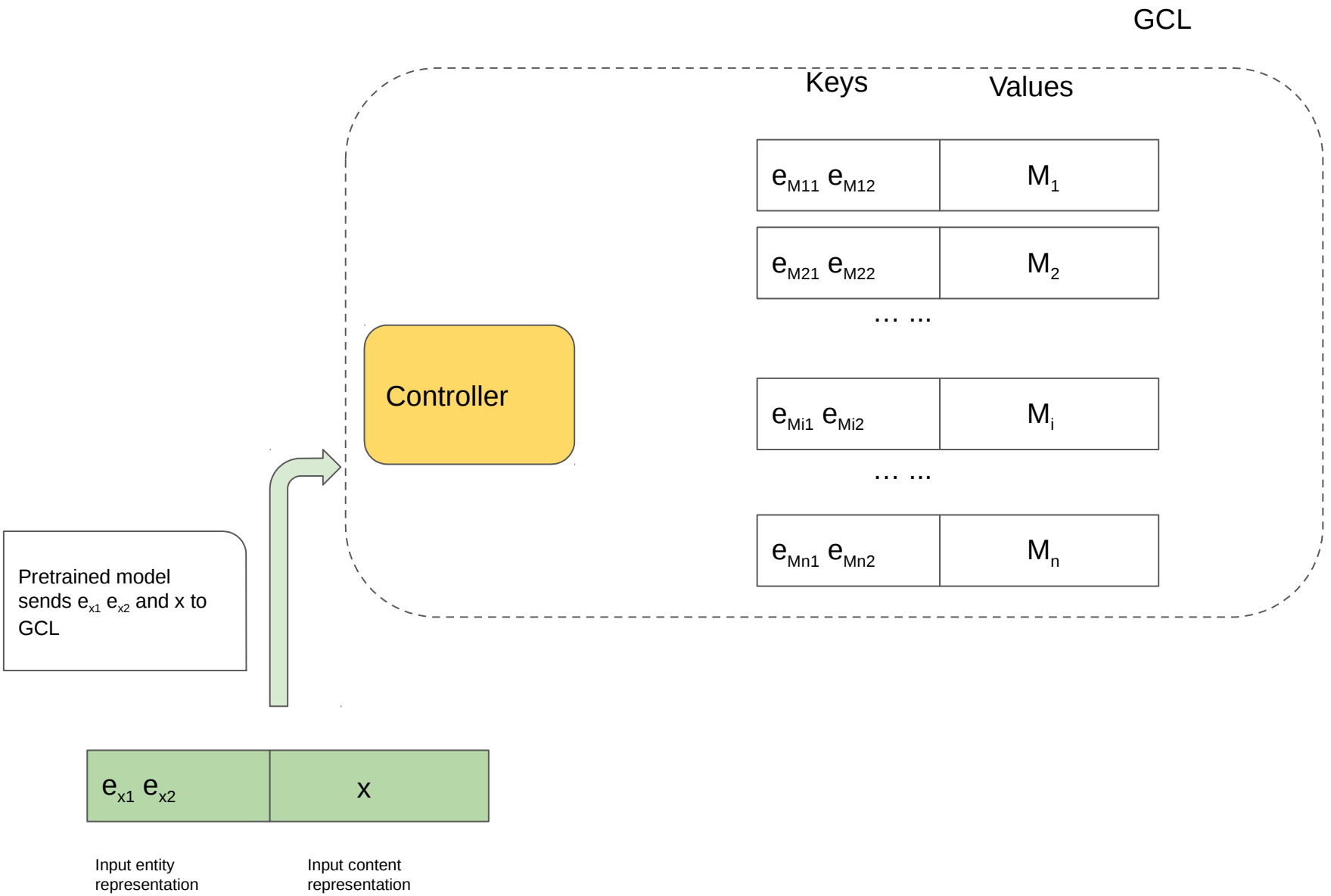
$$D[i] = \frac{1}{Z} \| \mathbf{e_1} \oplus \mathbf{e_2} - \mathbf{e_{M1}}[i] \oplus \mathbf{e_{M2}}[i] \|_2^2$$

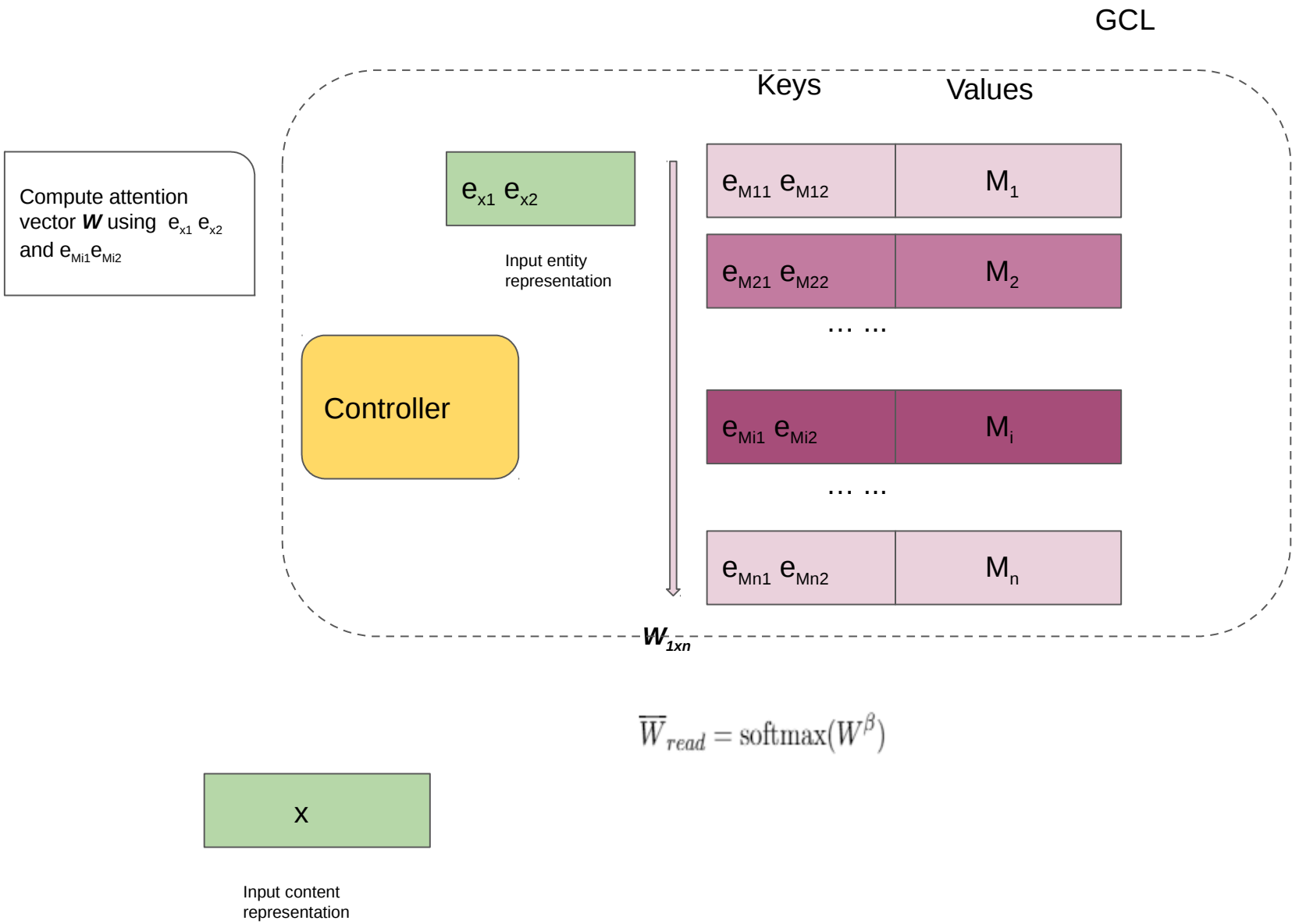
$$D'[i] = \frac{1}{Z'} \| \mathbf{e_2} \oplus \mathbf{e_1} - \mathbf{e_{M1}}[i] \oplus \mathbf{e_{M2}}[i] \|_2^2$$

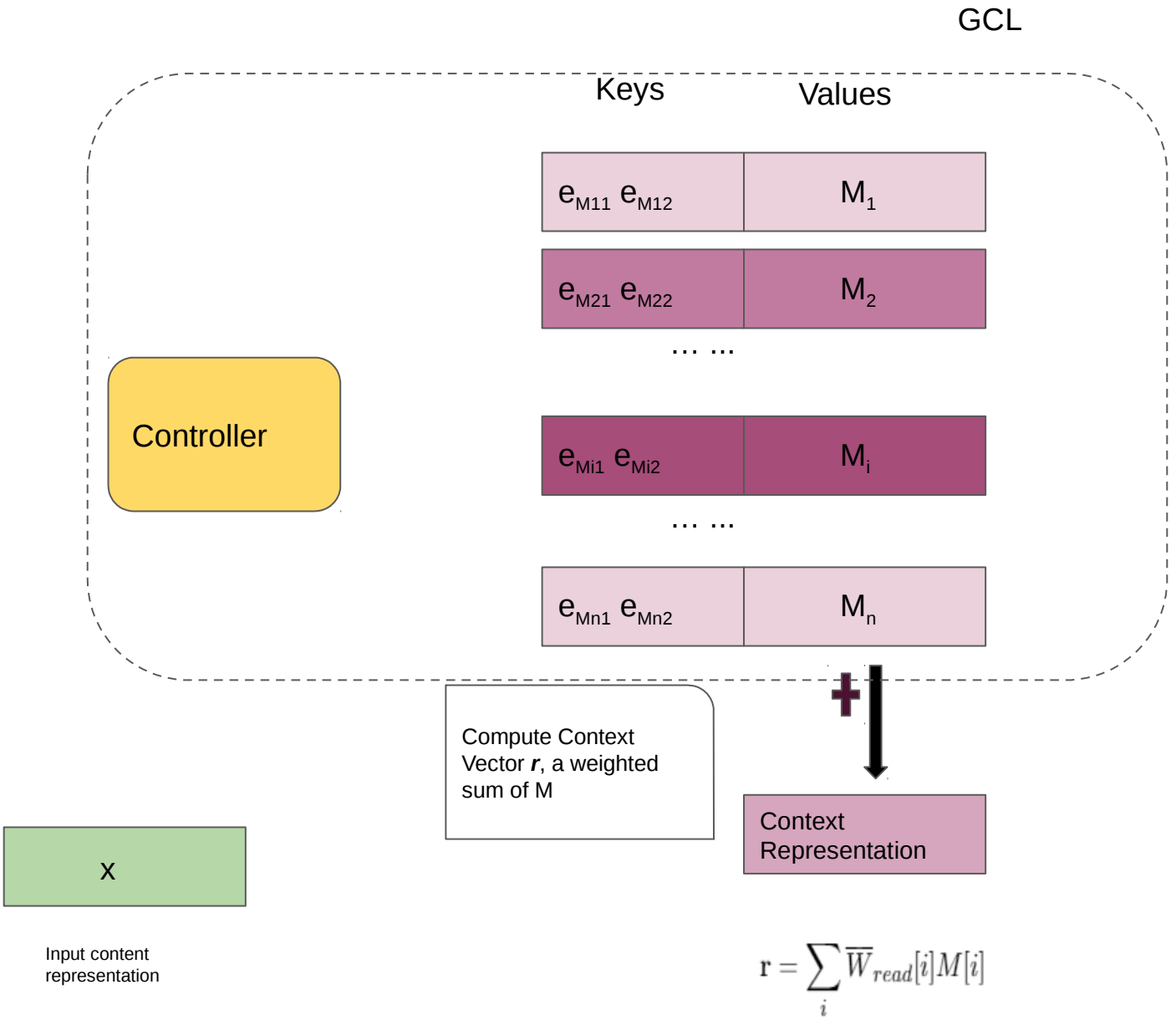
$\mathbf{W}$ is the attention vector representing the relevance of each memory location. $\mathbf{W}[i]=0$ means location i in memory has nothing to do with the input pair. Pairs are flipped on lookup because the order of entities should not affect relevance.

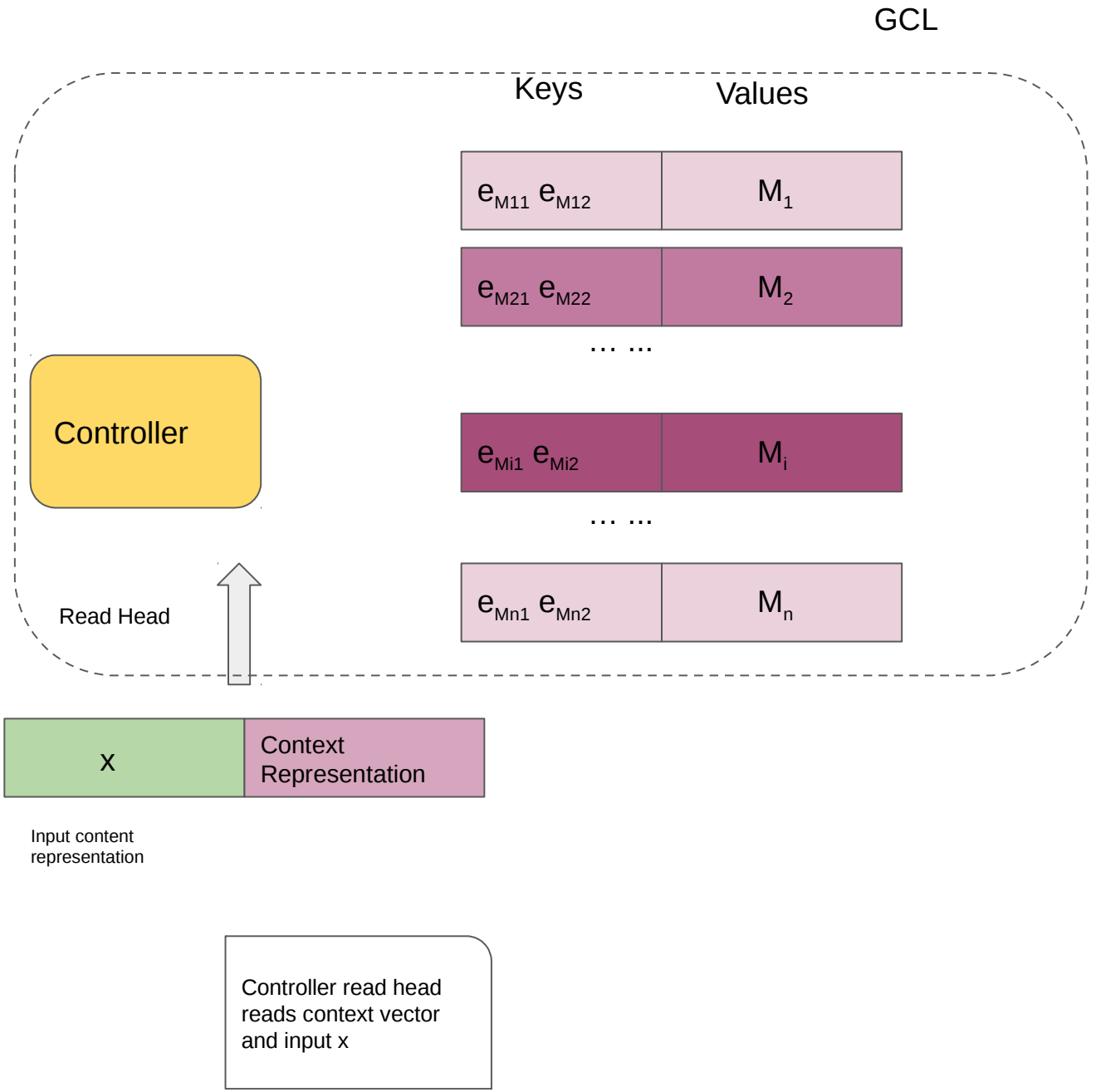
$$W[i] = \max(\text{softmax}(1 - \mathbf{D})[i], \text{softmax}(1 - \mathbf{D}')[i])$$

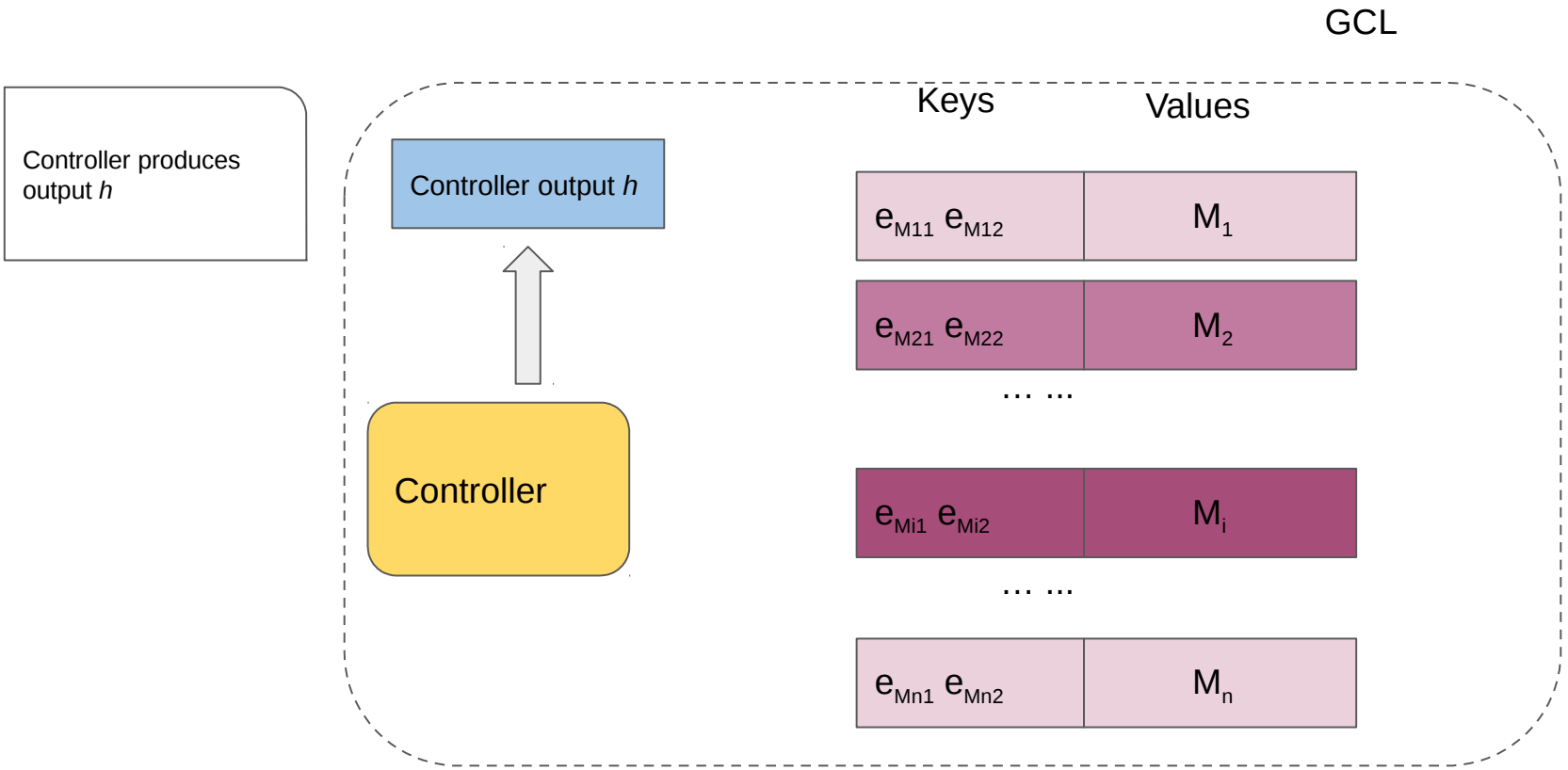
Pay more attention to locations with smaller distances to our key. Apply softmax to both vectors, Pick the maximum

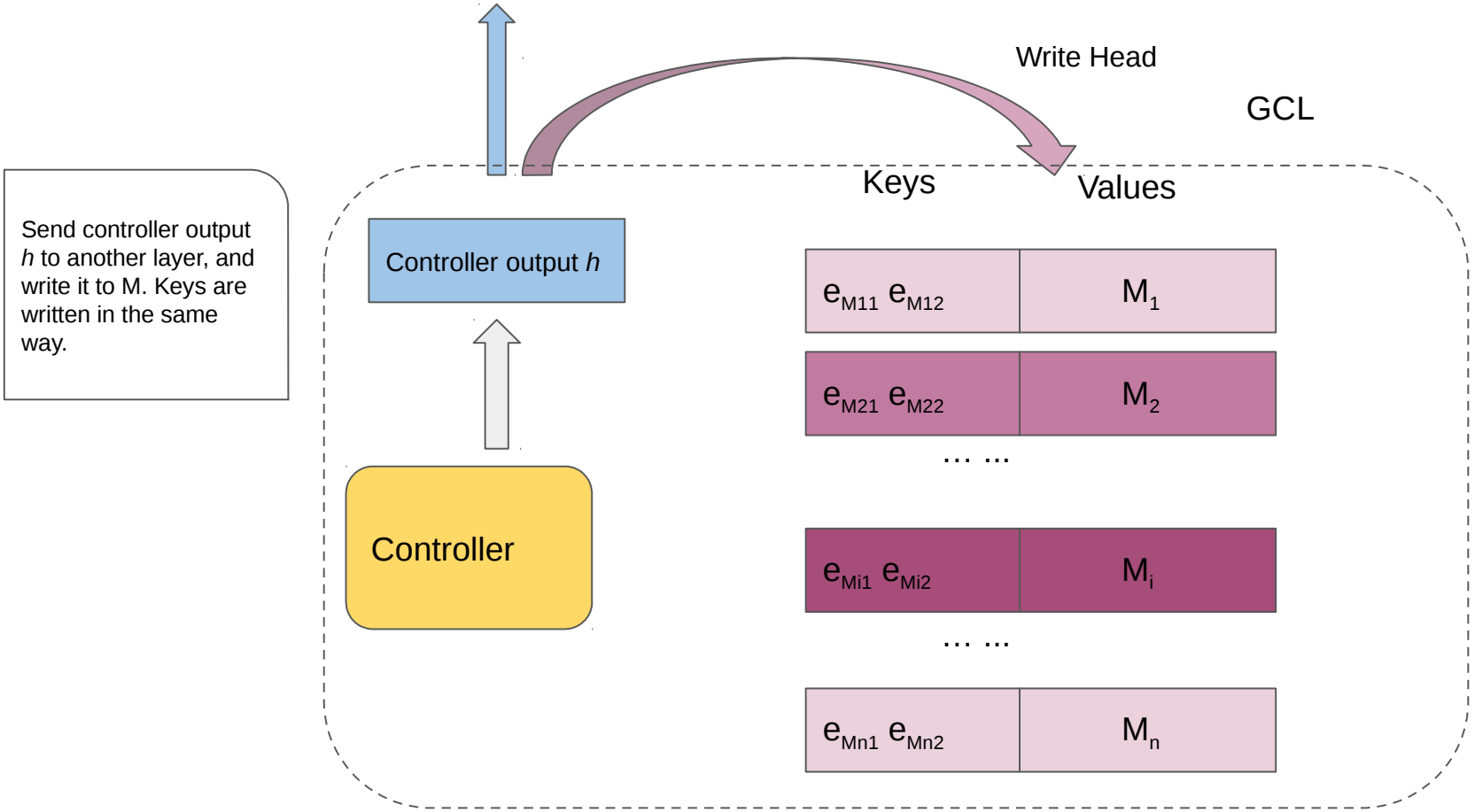

GCL
Keys
Values
e_{M11} e_{M12}
M_1
e_{M21} e_{M22}
M_2
... ...
e_{Mi1} e_{Mi2}
M_i
... ...
e_{Mn1} e_{Mn2}
M_n
Controller
Pretrained model sends e_{x1} e_{x2} and x to GCL
e_{x1} e_{x2}
x
Input entity representation
Input content representation


GCL
Keys
Values
Compute attention vector W using e_{x1} e_{x2} and $e_{Mi1}e_{Mi2}$
e_{x1} e_{x2}
Input entity representation
Controller
e_{M11} e_{M12}
M_1
e_{M21} e_{M22}
M_2
… …
e_{Mi1} e_{Mi2}
M_i
… …
e_{Mn1} e_{Mn2}
M_n
W_{1xn}
$\overline{W}_{read} = \mathrm{softmax}(W^{\beta})$
x
Input content representation


GCL
Keys
Values
e_{M11} e_{M12}
M_1
e_{M21} e_{M22}
M_2
... ...
Controller
e_{Mi1} e_{Mi2}
M_i
... ...
e_{Mn1} e_{Mn2}
M_n
Compute Context Vector *r*, a weighted sum of M
Context Representation
x
Input content representation
$\mathbf{r} = \sum_i \overline{W}_{read}[i] M[i]$


GCL
Keys
Values
e_{M11} e_{M12}
M_1
e_{M21} e_{M22}
M_2
... ...
Controller
e_{Mi1} e_{Mi2}
M_i
... ...
Read Head
e_{Mn1} e_{Mn2}
M_n
x
Context Representation
Input content representation
Controller read head reads context vector and input x

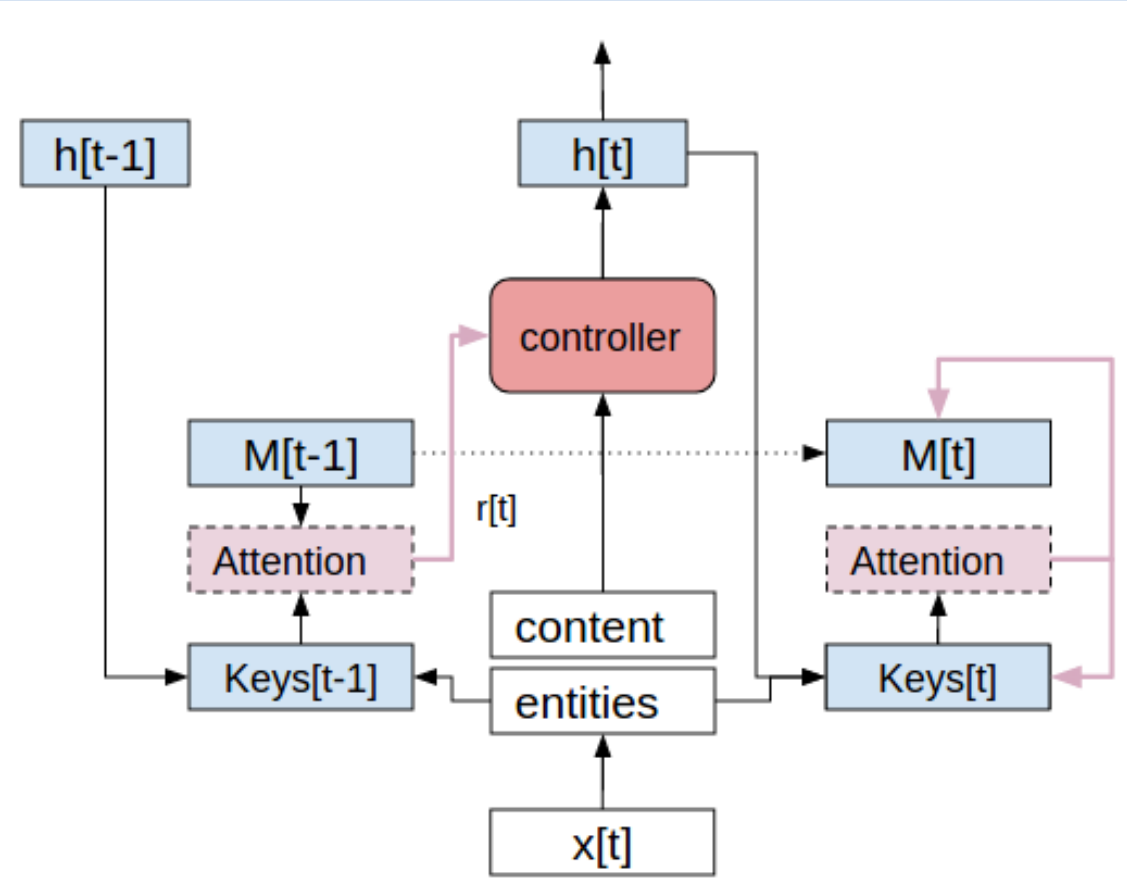

GCL
Controller produces output h
Controller output h
Controller
Keys
Values
e_{M11} e_{M12}
M_1
e_{M21} e_{M22}
M_2
... ...
e_{Mi1} e_{Mi2}
M_i
... ...
e_{Mn1} e_{Mn2}
M_n



$$M_t[i] = M_{t-1}[i] + \overline{W}_{write}(\mathbf{h}_t - M_{t-1}[i])$$

Global Context Layer

- GCL is recurrent, bidirectional:
  - Entity pairs are read in both directions, and representations combined in the output layer.



Training Procedure

- A model with GCL layer can be trained from scratch, but it can be slow to converge.
- In practice it is more efficient to train a model without GCL first, then truncate its top layers and assemble it with a model GCL layer. Continue to train the combined model until it converges.
-

Results

Model	Micro-F1	Macro-F1
CAEVO (not NN model)	.507	
CATENA (not NN model)	.511	
Cheng et al. 2017	.520[3]	
Meng et al. 2017		.519
pairwise	.535	.528
Two more hidden layers	.539	.532
GCL w/ state-tracking controller	.545	**.538**
GCL w/ stateless controller	**.546**	**.538**
GCL w/ pre-trained output layer	.541	.536

Table 1: Evaluation results on test set. The GCL models use the same hyperparameters, if possible. The two models on the top do not use neural networks. The results in the two lower blocks all use double-check. "Two more hidden layers" means adding two dense layers on top of the pre-trained model without using GCL. The last row corresponds to connecting the output layer of a pre-trained model to GCL layers with stateless controller.

Summary

- External memory component is able to eliminate the need to resolve conflicts in post-processing!
- Indexing into memory with keys derived from representations derived for entity contexts.
- This is how you really want to represent entities in text, when you want to maintain coherence in terms of things that are mentioned in text (people, objects, ideas).

What are the next steps?

- We really want a global, updatable memory that can accumulate information across tasks
  - Rather than trained locally for specific tasks
  - NOT read in and use existing sources (e.g. Wikipedia)
- This memory should allow storage and retrieval of long-term memories.
  - Conversational agents: must be able to amend, accumulate, and access knowedge base.
- Should allow short-term activations in context that help to interpret the input, keep the discourse coherence locally, and resolve long-distance dependencies correctly.
- Only then, will we have something that remotely resembles AGI from the POV of human language!

Thank you!

Text Machine Lab for NLP @ UMass Lowell

- Anna Rumshisky (PI)
- Students / Postdocs
  - Anna Rogers
  - Peter Potash
  - Alexey Romanov
  - Yuanliang Meng
  - Yen-Fu Luo
  - Olga Kovaleva
  - Saurabh Kulshreshtha
  - David Donahue

Thank you!

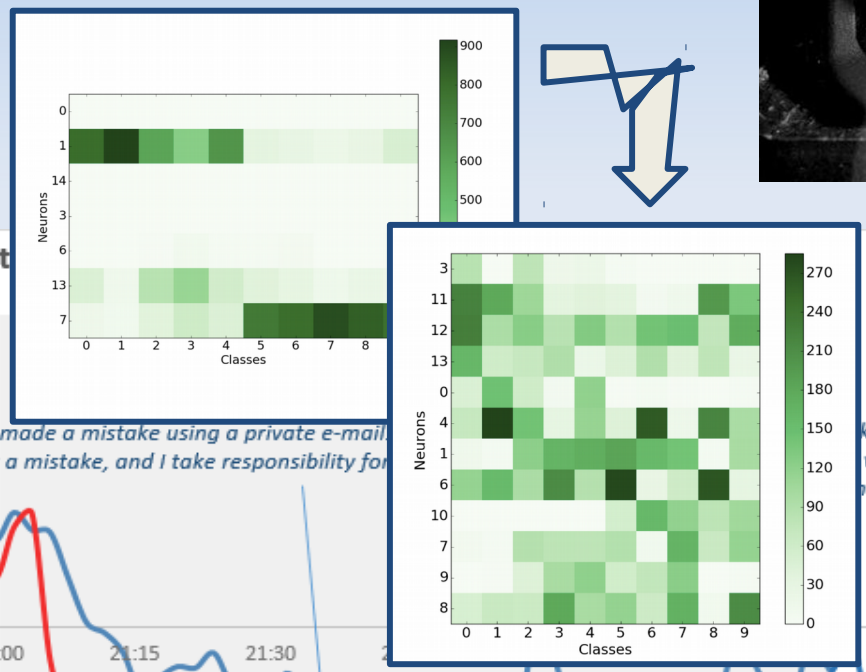
Text Machine Lab for NLP @ UMass Lowell

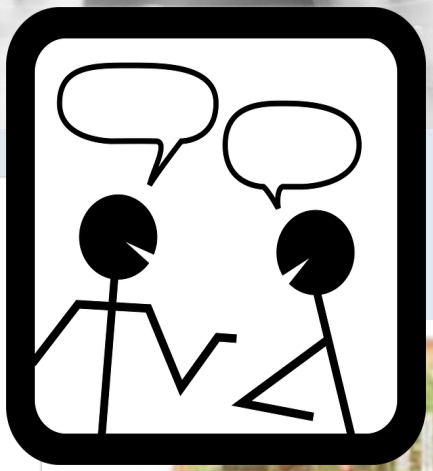
- Anna Rumshisky (PI)
- Students / Postdocs
  - Anna Rogers
  - Peter Potash
  - Alexey Romanov
  - Yuanliang Meng
  - Yen-Fu Luo
  - Olga Kovaleva
  - Saurabh Kulshreshtha
  - David Donahue

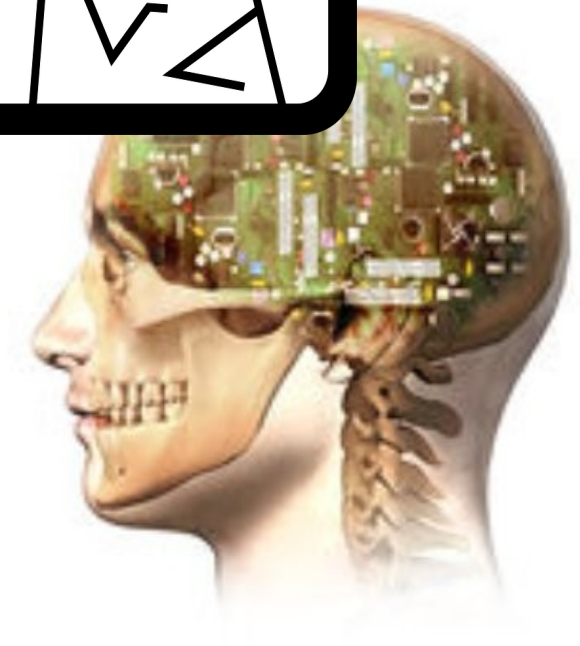
PhD position open @ Text Machine Lab:
Artificial Intelligence / Machine Learning / Deep Learning

- The Text Machine Lab for Natural Language Processing at the Computer Science Department at the University of Massachusetts Lowell invites applications for two fully-funded PhD positions. The lab conducts research in deep learning applications to a variety of problems in NLP.
- The Text Machine lab is currently being supported by over $1.5M research grants. The current externally funded research is in the areas of text-based temporal reasoning, computational analysis of civil conflict and informational biases in social media, and predictive modeling of patient outcomes and disease trajectories using clinical text from the electronic health records. Students have many opportunities to collaborate with other high profile computer science and biomedical researchers in the Boston Metro areas, including MIT, Dartmouth College, Boston University, Brandeis University, and others. More information about the lab can be found here: http://text-machine.cs.uml.edu.
- MINIMUM QUALIFICATIONS: A Master's in Computer Science or related field. A strong background in mathematics and solid coding skills. To apply, please send your CV, research statement, a copy of your academic transcripts to Anna Rumshisky at arum@cs.uml.edu.

A Peek at Our Other Projects



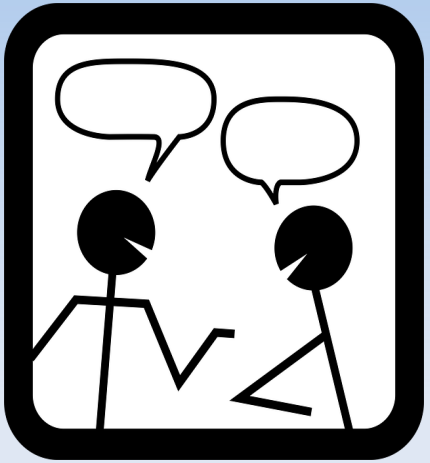




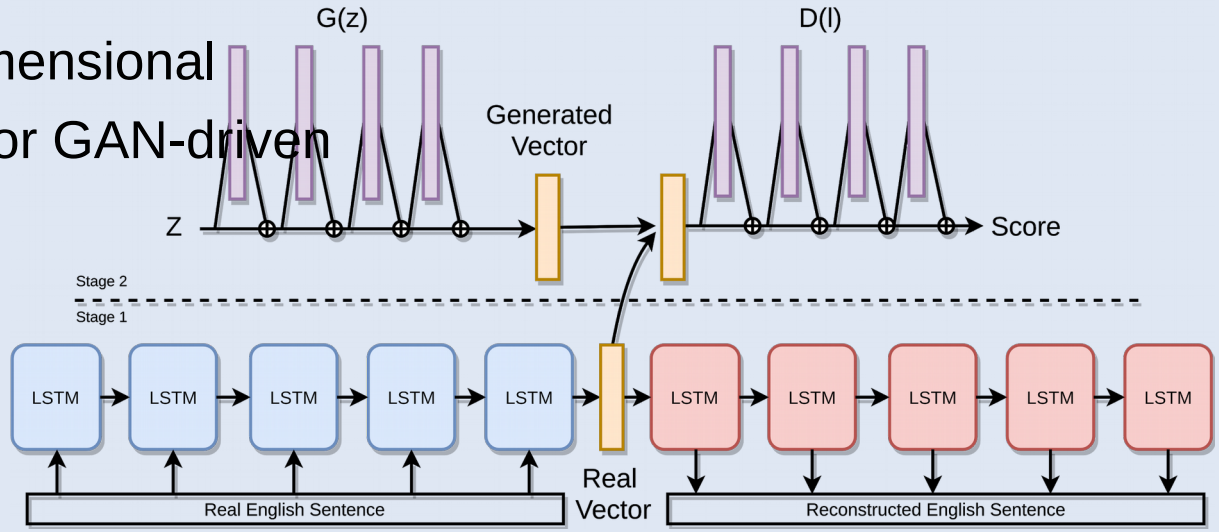
A Peek at Our Projects

- **Clinical informatics:**
  - Modeling patient state for clinical outcomes and disease trajectories (Rumshisky et al 2016 J. Trans. Psyc/Nature, Ghassemi et al KDD 2014, Luo & Rumshisky AMIA 2016, Romanov & Rumshisky 2017)
- **Computational social science:**
  - Modeling civil conflict and information biases in social media (Rumshisky et al, SocInfo 2017)
  - Predicting debate winners (Potash & Rumshisky EMNLP 2017)
- **Natural Language Processing/Computational linguistics**
  - Semantic representation & knowledge integration (Potash et al IJCNLP 2017, Potash et al 2016)
  - Extracting order and timing of events from text (Meng et al EMNLP 2017)
  - Identifying humour and sentiment (Potash et al 2017, Donahue et al 2017, Boag et al 2015)
  - Argument parsing (Potash et al EMNLP 2017)
  - Creative language generation – Ghostwriting rap lyrics (Potash et al EMNLP 2017)

Adversarial Generation of Text
for diversifying conditioned responses in conversation

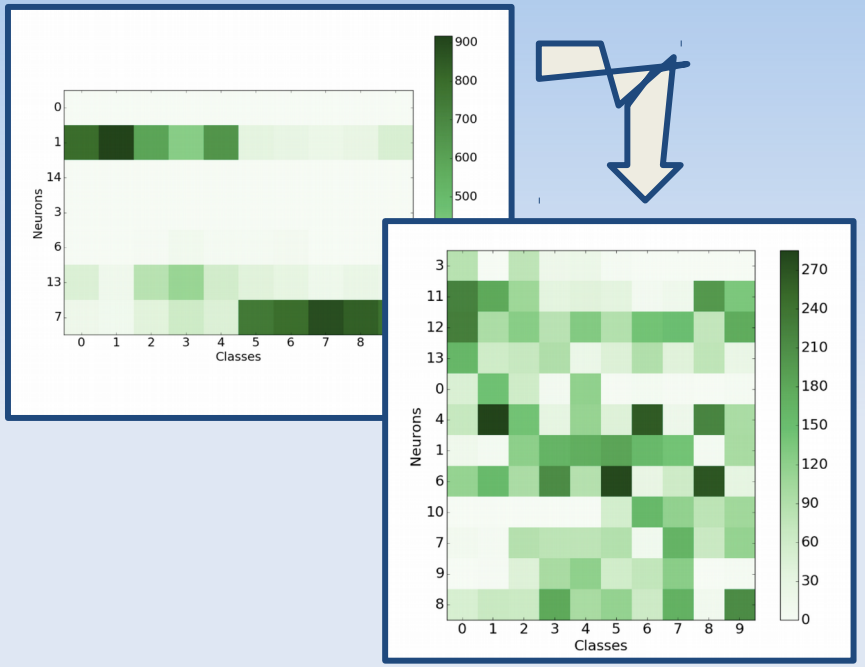


- Autoencoder-learned low-dimensional continuous representations for GAN-driven Text generation



Donahue and Rumshisky (under review)

Learning Better Representations from Limited Labels



Single-layer loss:

- KL-divergence on softmax of the rows of the weight matrix of a given layer
- Forces the rows to be different from each other, leading to different activation patterns

Multi-layer loss:

- KL-divergence on softmax of the outputs of the target hidden layer
- Affects the full network

Romanov and Rumshisky, ICLR 2017 workshop track

Conflict tracking in social networks

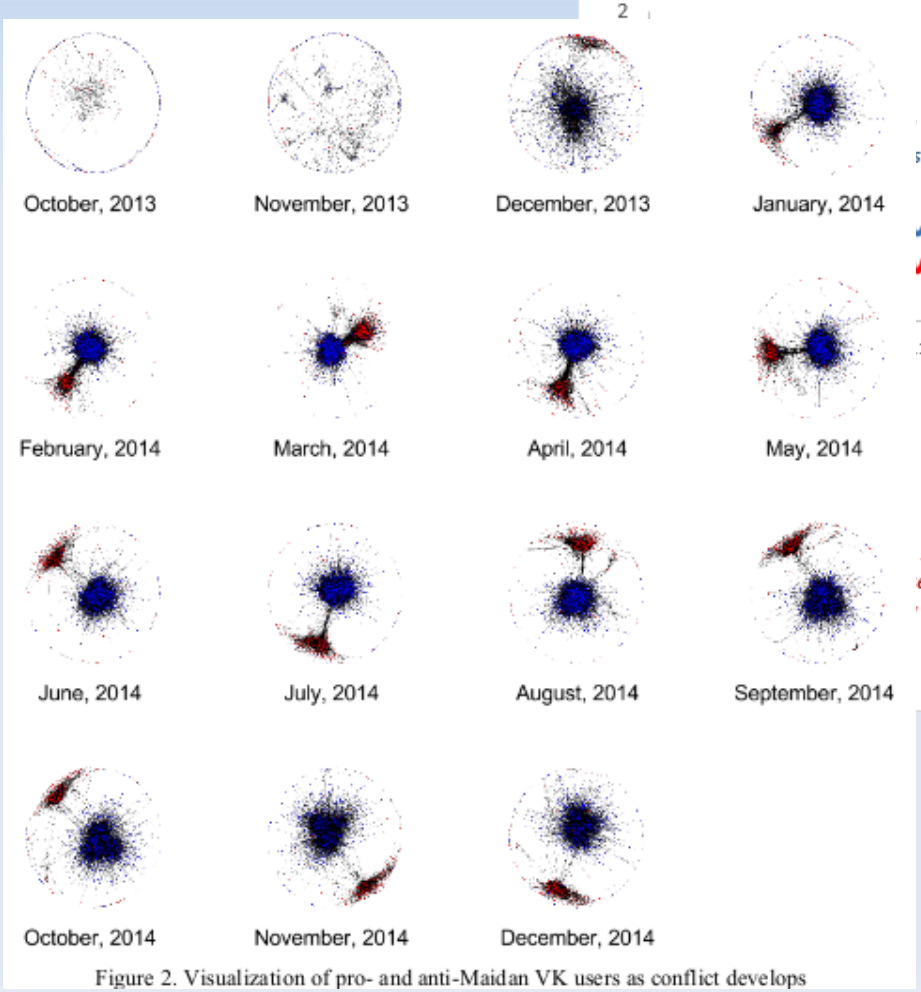
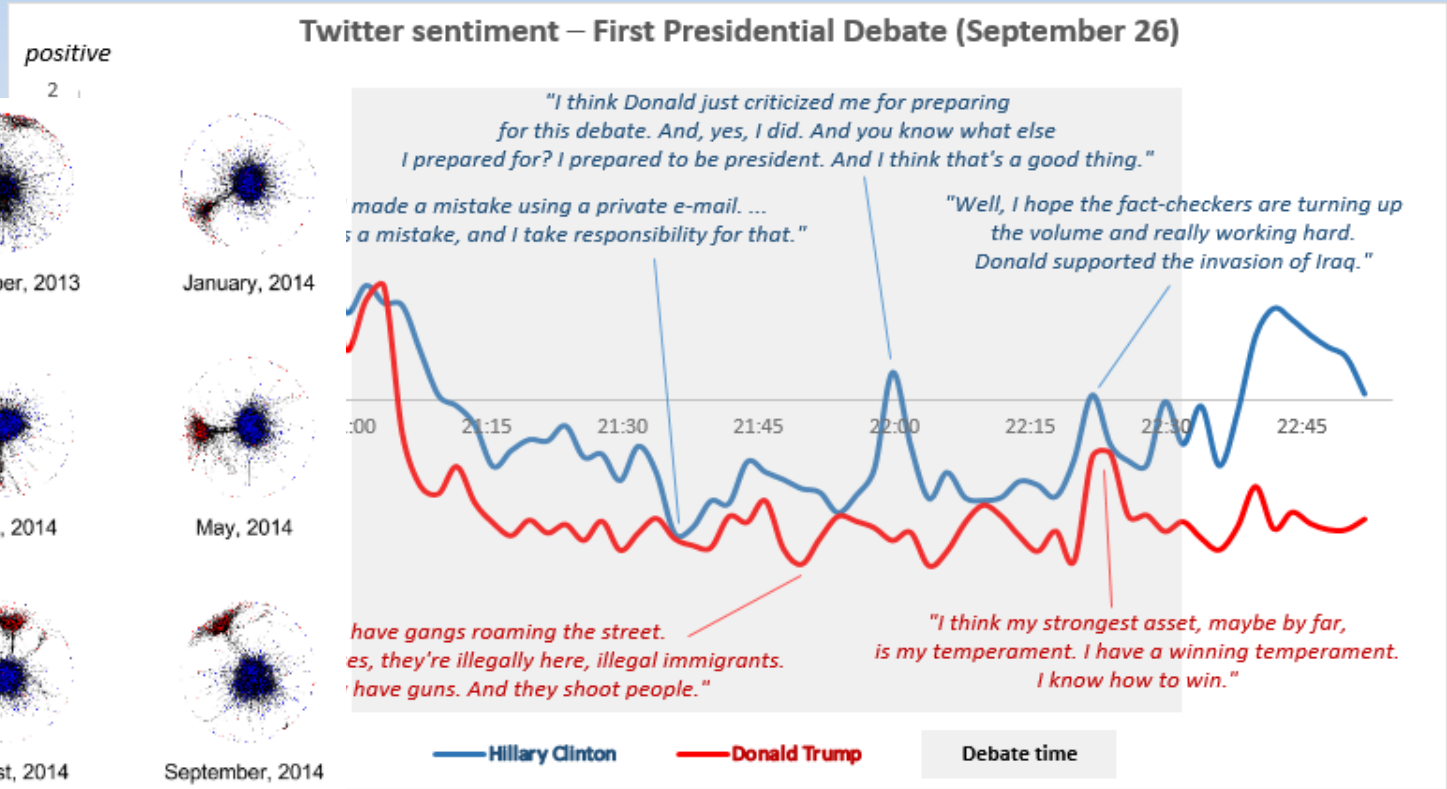


Figure 2. Visualization of pro- and anti-Maidan VK users as conflict develops



Rumshisky et al, SocInfo 2017

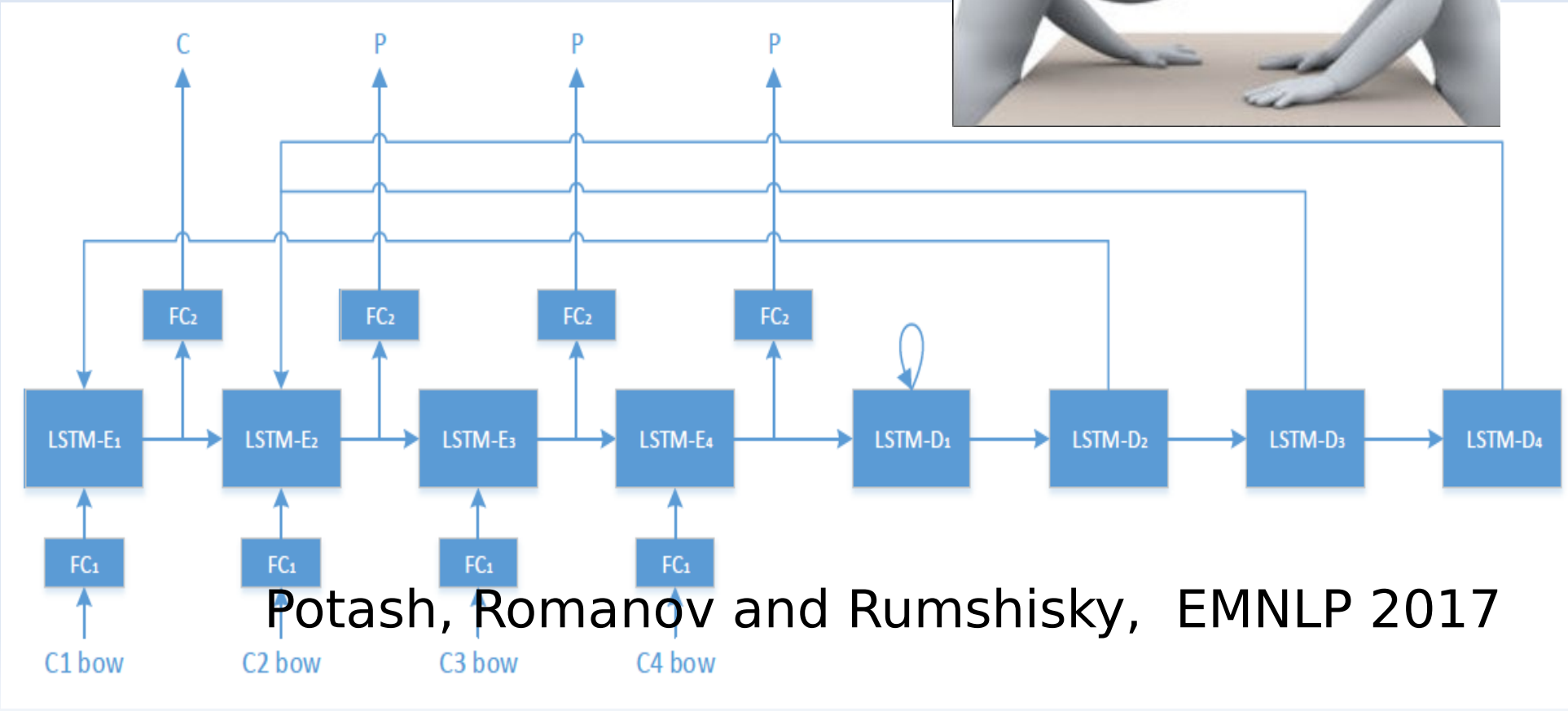
GhostWriting Rap Lyrics

Evaluation of Creative Language Generation

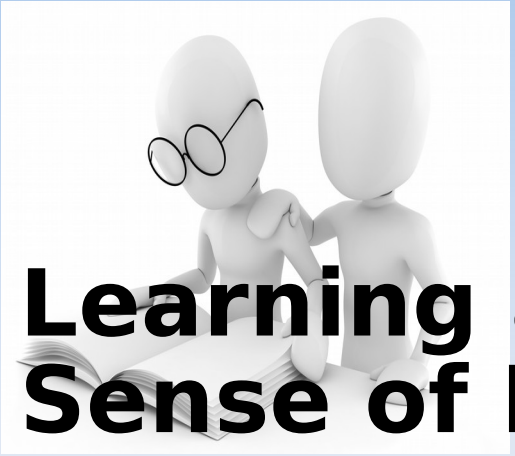
- Token-level sequence model to generate verses with rhymes
- Evaluation:
  - the verse should be **similar in style**, but **distinct from existing verse**
- Manual evaluation:
  - style matching, language fluency, verse coherence
- Automatic evaluation: thyme density, verse uniqueness

Potash, Romanov and Rumshisky, EMNLP 2015

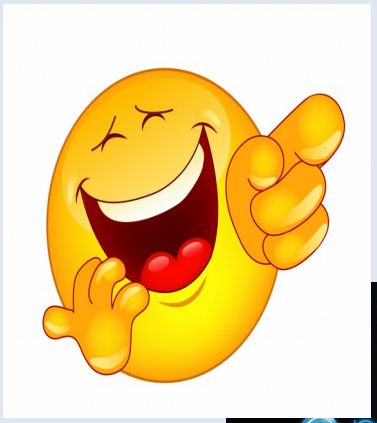
Parsing Arguments

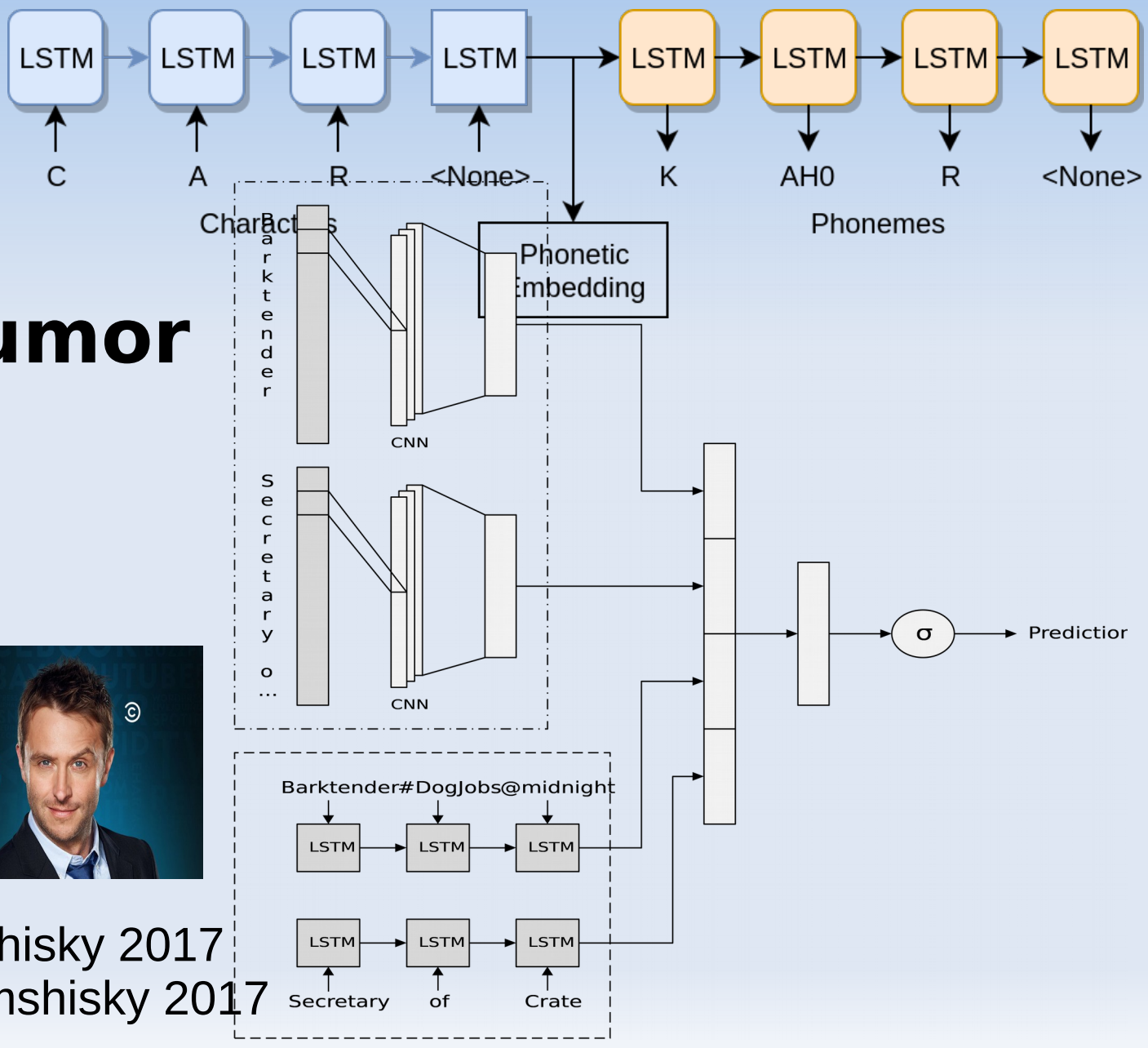


Potash, Romanov and Rumshisky, EMNLP 2017



Learning a Sense of Humor

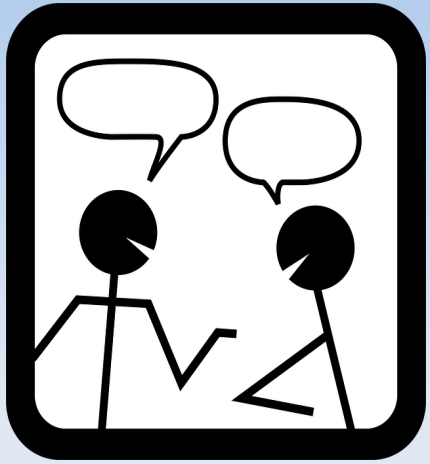




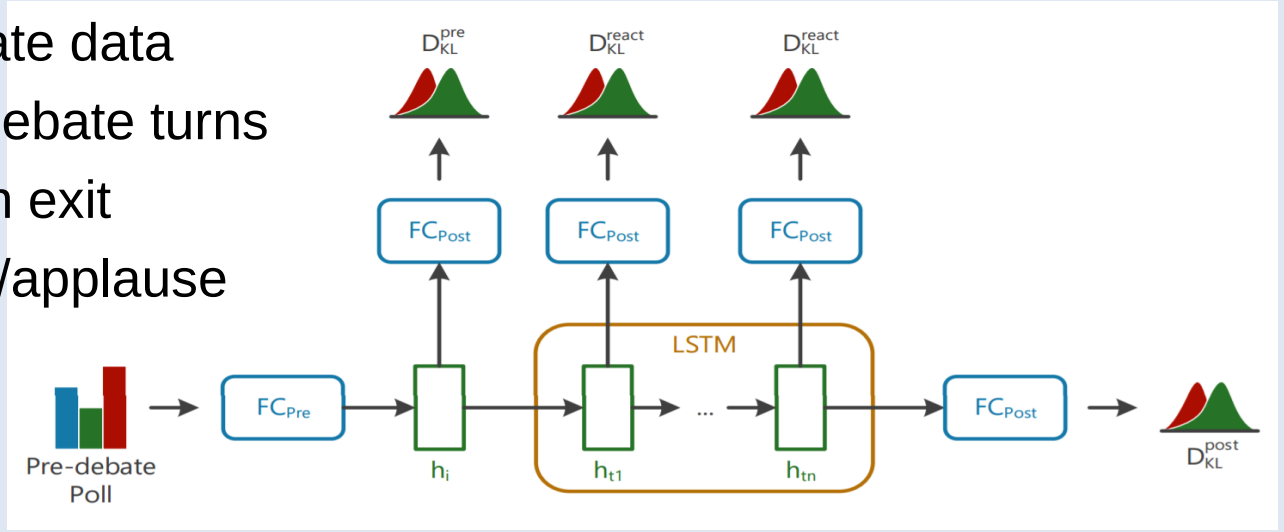
Potash, Romanov, Rumshisky 2017
Donahue, Romanov, Rumshisky 2017

Predicting Debate Winners

Identifying Persuasive Arguments



- Intelligence Squared debate data
- Modeling a sequence of debate turns
- to predict audience poll on exit
- Regularize using laughter/applause



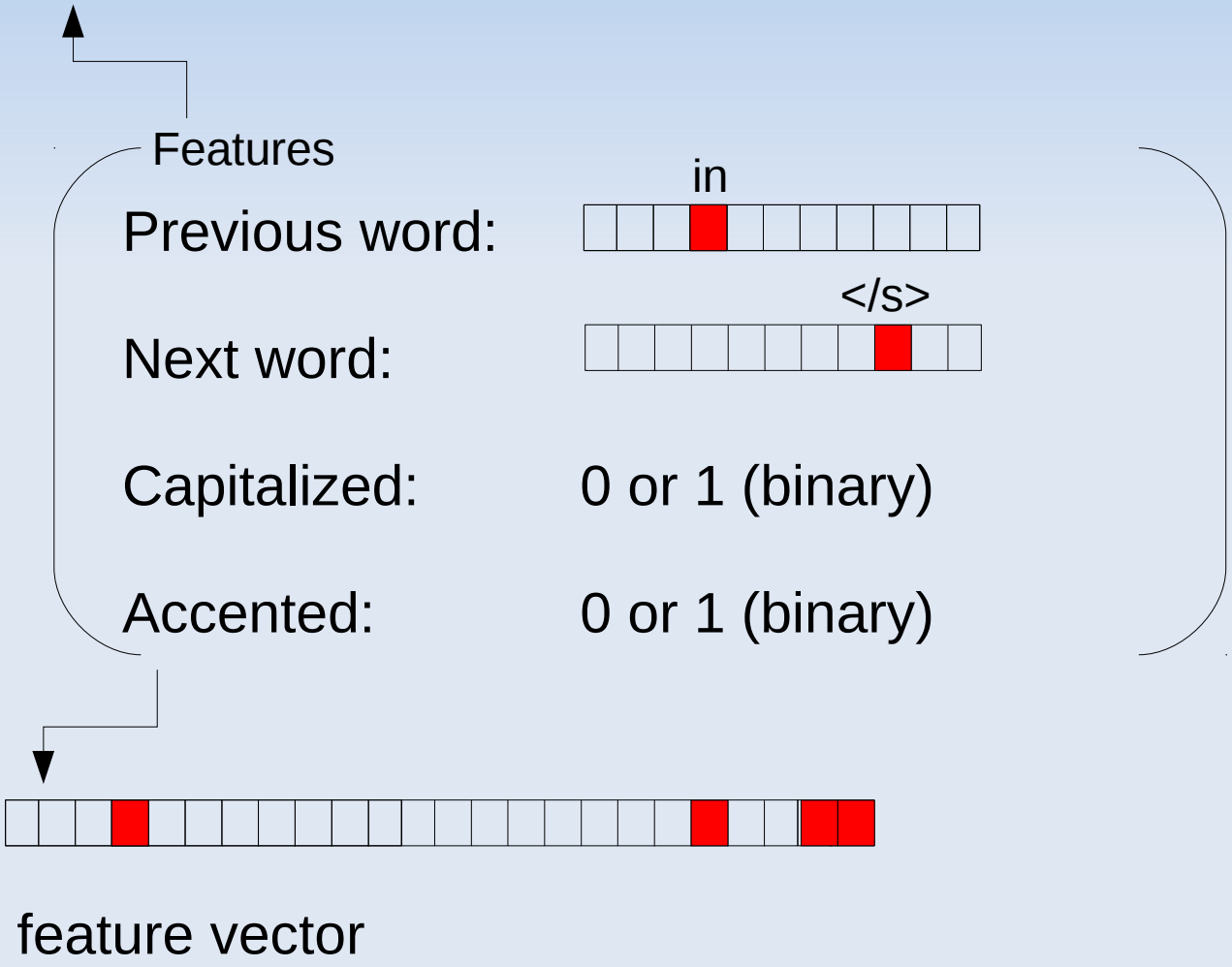
Potash, Romanov, and Rumshisky, EMNLP 2017

EXTRA SLIDES – INTRO

Example – Feature Representations: Detecting Location Names

<s> He lives in Québec </s>

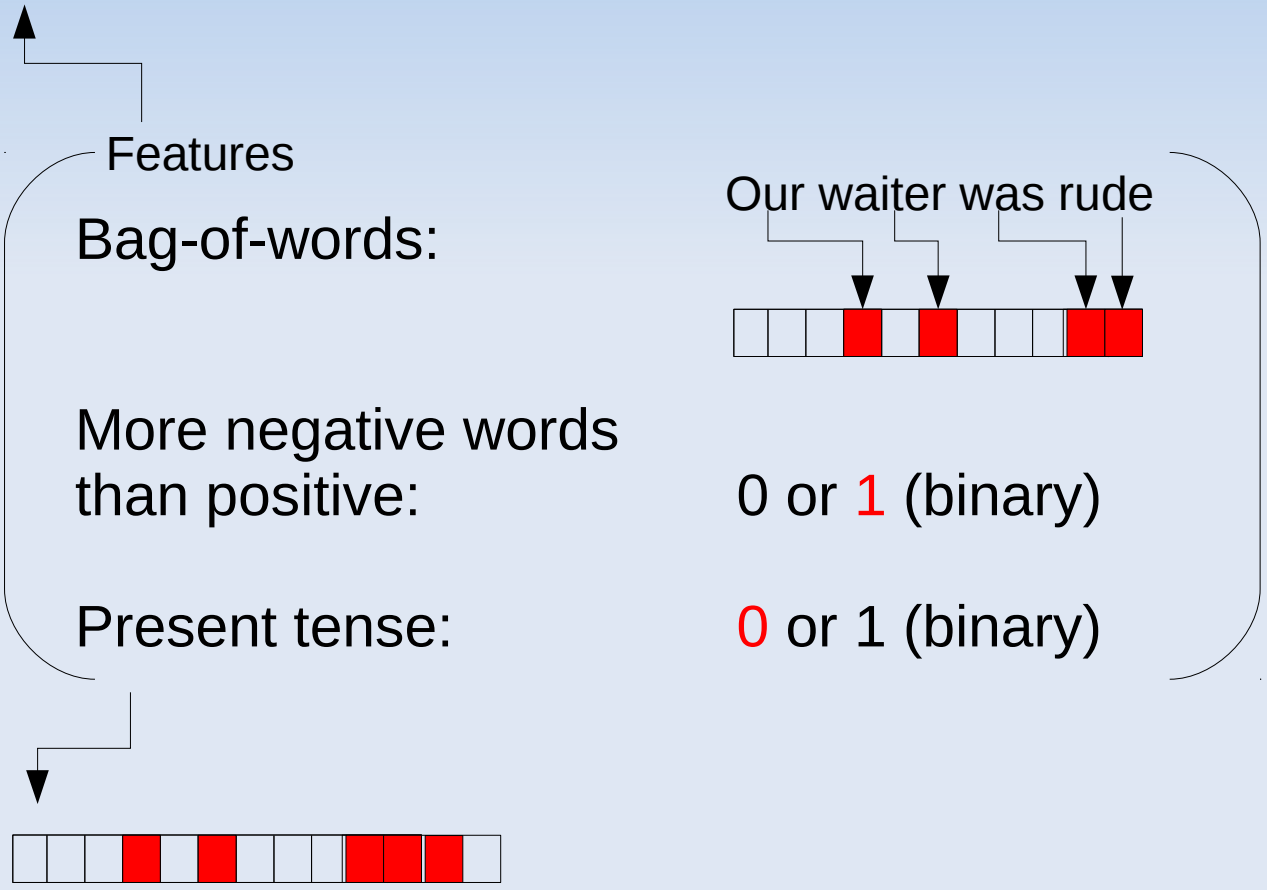
LABEL: Yes-Location



Example – Feature Representations: Sentiment Analysis

<s> Our waiter was rude . </s>

LABEL: NEGATIVE



EXTRA SLIDES – DISSONET

Wasserstein GAN

- Predicting a distribution over whether an example is real or generated is replaced with producing a real number
  - non-linearity removed from the output layer
- Predicting that the input was a real example corresponds to large positive values. For generated examples, a proper discriminator should produce negative values.
- Generator tries to minimize the value produced by the discriminator for fake examples (i.e. get it to produce values closer to the ones it should produces for real examples).

Wasserstein GAN

- $z \sim p(z)$ is randomly generated from a distribution.
- x is drawn from a batch of real data, m is the batch size.
- **Discriminator loss**

$$L_d = 1/m \; \Sigma^m{}_{i=1} \; D(G(z)) - 1/m \; \Sigma^m{}_{i=1} D(x)$$

$$L_d(i) = D(G(z)) - D(x)$$

- When the discriminator loss L_d is minimized,
  - $\mathrm{D(G(z))}$ decreases (negative)
  - $\mathrm{D(x)}$ increases (positive)
- **Generator loss**

$$L_g = - \; 1/m \; \Sigma^m{}_{i=1} D(G(z))$$

$$L_g(i) = - \; D(G(z))$$

- When the generator loss L_g is minimized $D(G(z))$ increases

Wasserstein GANs – Details

- GANs suffer from a number of convergence problems (Goodfellow et al., 2014). For an over-trained discriminator, generated examples receive almost zero probability. This can result in vanishing gradients for the generator and slow convergence.
- Simultaneous optimization of the generator and discriminator can result in changing objectives and failed optimization for both models (e.g. mode collapse)
- To solve these issues, Arjovsky et al. (2017) proposed the Wasserstein GAN (WGAN), which uses the Earth-Mover distance metric for optimization, and clips the discriminator weights during training.
- They show the Earth-Mover distance provides stable gradients for all points in the solution space. Gulrajani et al. (2017) introduce a regularization of the WGAN which encourages the norm of discriminator gradients to approach unity.

EXTRA SLIDES – GCL

Understanding a Simple Narrative

- ADMISSION DATE: 09/24/2006
  - The patient is a (XX)-year-old white female with known history of asthma since infancy, who presented with progressive wheezing and respiratory distress for the past two days. The patient had been doing well on only p.r.n. medications per family's report. However, just previous to admission, the patient was exposed to dust and other particles after moving into a new house. After conservative treatment at home, the patient was brought into the emergency room where she did not improve on albuterol, Atrovent treatments or intravenous steroids immediately.

Understanding a Simple Narrative

- ADMISSION DATE: 09/24/2006
  - The patient is a (XX)-year-old white female with known history of asthma since infancy, who presented with progressive wheezing and respiratory distress for the past two days. The patient had been doing well on only p.r.n. medications per family's report. However, just previous to admission, the patient was exposed to dust and other particles after moving into a new house. After conservative treatment at home, the patient was brought into the emergency room where she did not improve on albuterol, Atrovent treatments or intravenous steroids immediately.

Understanding a Simple Narrative

- ADMISSION DATE: 09/24/2006
  - The patient is a (XX)-year-old white female with known history of asthma since infancy, who presented with progressive wheezing and respiratory distress for the past two days. The patient had been doing well on only p.r.n. medications per family's report. However, just previous to admission, the patient was exposed to dust and other particles after moving into a new house. After conservative treatment at home, the patient was brought into the emergency room where she did not improve on albuterol, Atrovent treatments or intravenous steroids immediately.

Understanding a Simple Narrative

- ADMISSION DATE: 09/24/2006
  - The patient is a (XX)-year-old white female with known history of asthma since infancy, who presented with progressive wheezing and respiratory distress for the past two days. The patient had been doing well on only p.r.n. medications per family's report. However, just previous to admission, the patient was exposed to dust and other particles after moving into a new house. After conservative treatment at home, the patient was brought into the emergency room where she did not improve on albuterol, Atrovent treatments or intravenous steroids immediately.

Problem

- Recovering the ordering and timing of events from the narrative often involves reasoning over multiple relations.
- This requires representing and retrieving long-distance dependencies between different elements in text.

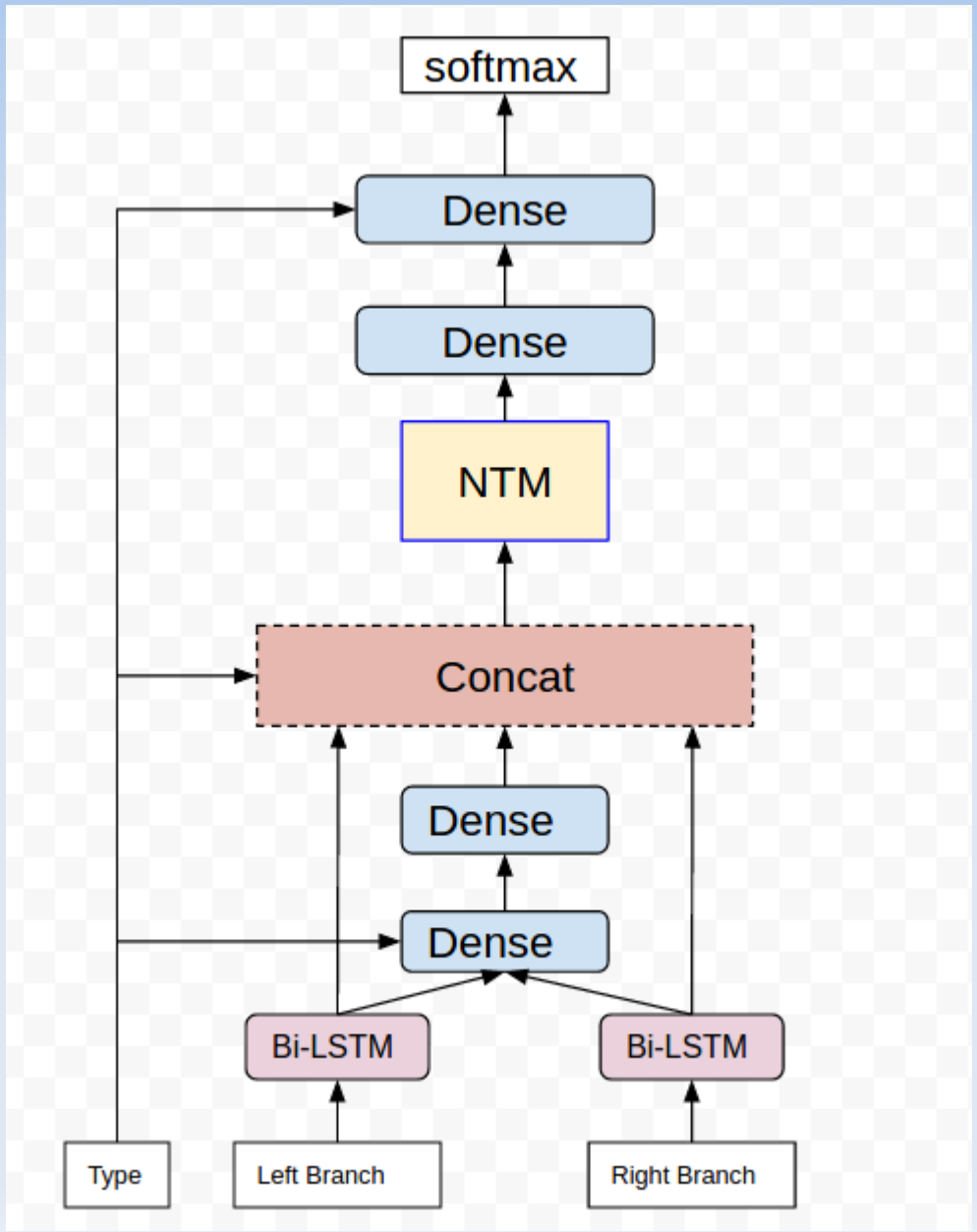
Task Description

- Given text files with event and time expression (timex) tags, find the temporal relations between events and timexes.
- TimeBank-Dense dataset defines 6 basic temporal relations: SIMULTANEOUS, BEFORE, AFTER, INCLUDES, IS_INCLUDED and VAGUE.
  - The categories are imbalanced. VAGUE is the largest category (~40% instances), while SIMULTANEOUS is the smallest (<2%)
- All event/timex pairs within sentences or in consecutive sentences are labeled (so it is called “dense”).
  - Human annotators often do not agree with each other, especially for VAGUE.
- Long-distance relations are not labeled. If needed, they can be inferred from relations between other entities.

Approach

- NLP models usually perform pairwise classification. Each time one pair of events and/or timexes is labeled. At the end, some heuristics may be applied to resolve conflicts in the timegraph in a larger scope.
- Ideally, pairwise relations should not be independent of each other. Relations between a certain pair depends on relations between other entities. When a person read an article, context information is used to build temporal relations.
- We propose a model with a Global Context Layer (GCL) to achieve this goal. The model produces pairwise labels, but the classification procedure is context-aware.
- GCL is inspired by Neural Turing machine (NTM). In the following we will introduce our model first, and then discuss the differences from NTM.

Global Context Layer



Results:

TimeBank-Dense dev set accuracy
0.512 without any pruning process,

Old results:
0.527 – old system with pruning.

Four branches
- flat context
- parsed context

GCL Model

With the weight vector for addressing, we can now read information from the memory content.

$$\mathbf{r} = \sum_i \overline{W}_{read}[i] M[i]$$

The context information vector ***r*** is a weighted sum over all locations. Relevant locations have higher weights, and thus are paid more “attention” to, so the weight vector can also be called attention vector.

Vector ***r*** is concatenated with input ***x*** *(dense layer – output of pairwise model)* and processed by the controller of GCL. A controller can be just a fully connected layer (*stateless*), or an LSTM cell with *internal states*. Its output $\boldsymbol{h}_t$ is sent to the next layer, and also written in GCL memory.

GCL Model

GCL basically uses the same addressing method for reading and writing. However, the attention vector is shifted before using for writing. So the updated memory does not necessarily lose the relevant old information.

$$\widetilde{W}[i] = \sum_{j=0}^{n-1} W[j]\mathbf{s}[i-j]$$

Here ***s*** is a shift kernel. It is obtained from shift weights $\boldsymbol{C}_t$:

$$C_t = \mathrm{softmax}(W_s[\mathbf{x}_t, \mathbf{h}_t] + b_s)$$

$\boldsymbol{C}_t$ and ***s*** are almost equivalent, but we have some additional process to make sure **s**[i] has zero values in most places, except when *i* is close to 0. In other words, the attention vector should not be shifted too far. W_s and b_s are parameters to be trained.

GCL Model

Finally, the shifted attention vector for writing is also sharpened.

$$\gamma_t = \text{relu}(W_{sharp}[\mathbf{x}_t, \mathbf{h}_t] + b_{sharp}) + c_\gamma$$

$$\overline{W}_{write} = \text{softmax}(\widetilde{W}^\gamma)$$

Output of GCL controller is used to update the memory.

$$M_t[i] = M_{t-1}[i] + \overline{W}_{write}(\mathbf{h}_t - M_{t-1}[i])$$

So the memory is dynamically accessed and updated, depending on input entities.

Comparison with NTM (Graves et al 2014)

NTM incorporates a neural network with memory and attention. The combined system is analogous to a Turing Machine or Von Neumann architecture but is differentiable end-to-end, allowing it to be trained with gradient descent.

Many of the techniques of GCL are the same as NTM, but there are some major differences too.

- GCL directly uses representations of entity pairs as keys. NTM uses key functions to compute keys from input and output, in order to address memory.
- GCL allows the attention vector to fully control memory updates. NTM computes an erase vector and an add vector on top of attention vector. So the updating involves more mechanisms.
- NTM has gates to interpolate the attention vector in current time step with the one in previous time step. For GCL, the previous attention vector is totally ignored.

Motivation for Differences

NTM has a lot of parameters and hyper-parameters, which makes it computationally powerful.

Given our task, we can make some functions more specific. For example, we look for context information based on input entities. Therefore the “key function” is specific to our task. There is no need to use very complex attention mechanisms.

The dataset is relatively small. Having too many parameters is both unnecessary and problematic.

However, the major architecture of GCL is the same as NTM. It has read heads, write heads, and a controller. It is a generalized RNN with a memory. It can process series data but does not require contiguity of series.

GCL Model

W is often too blurred (having similar values in many places), so we sharpen it.

$$\overline{W}_{read} = \mathrm{softmax}(W^{\beta})$$

Here β is a number greater than 1. It applies to every component of ***W***. Instead of using a constant number here, we compute it each time, using the current input and the GCL output of previous time step.

Here W_{sharp} and b_{sharp} are parameters to be trained. c_{β} is a chosen non-negative constant = 5. This already makes GCL a recurrent neural network (RNN), because it uses previous output of the controller.

$$\beta_t = \mathrm{relu}(W_{sharp}[\mathbf{x}_t, \mathbf{h}_{t-1}] + b_{sharp}) + c_{\beta}$$