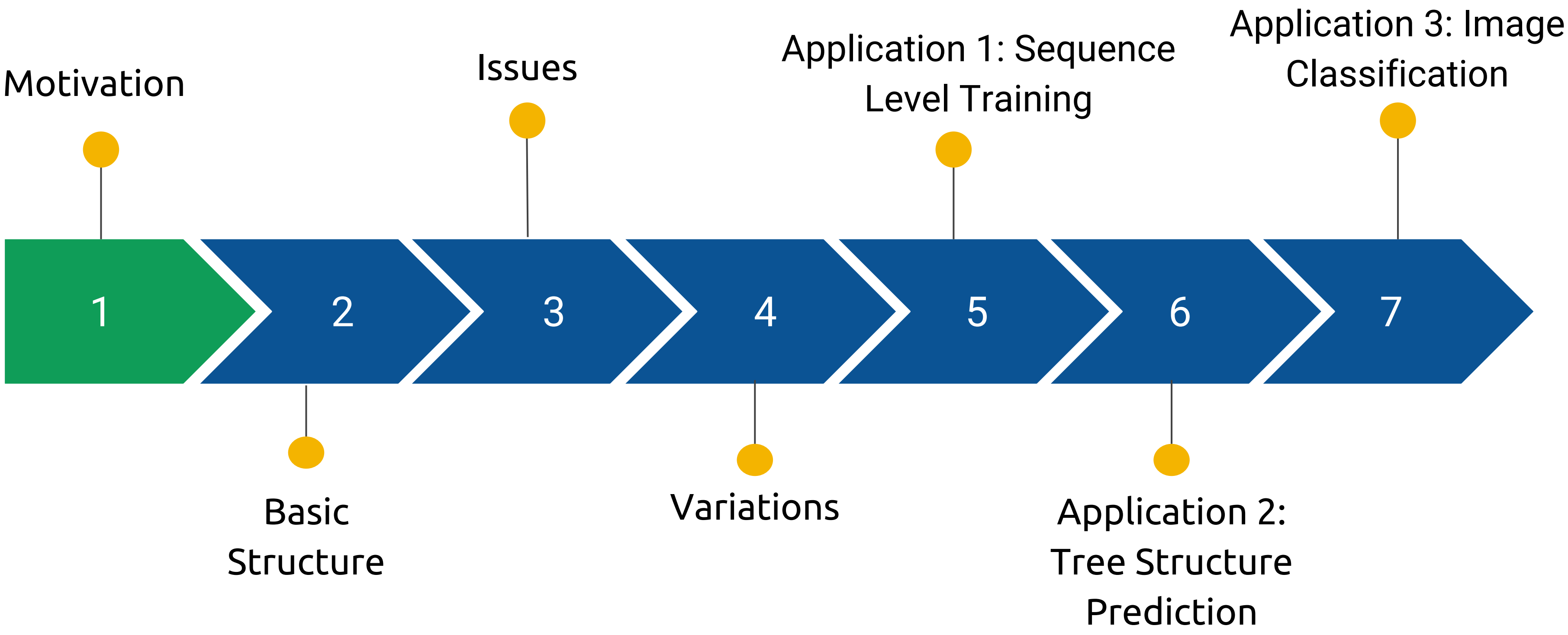
Recurrent Neural Networks

Nand Kishore, Audrey Huang, Rohan Batra

Roadmap


Motivation
Basic Structure
Issues
Variations
Application 1: Sequence Level Training
Application 2: Tree Structure Prediction
Application 3: Image Classification
1
2
3
4
5
6
7

Motivation

- Sequential data is found everywhere
  - Sentence is sequence of words
  - Video is sequence of images
  - Time-series data is sequence across time

- Given a sequence (x_1, x_2, ...x_i)
  - The x_i's are not i.i.d!
  - Might be strongly correlated

- Can we take advantage of sequential structure?

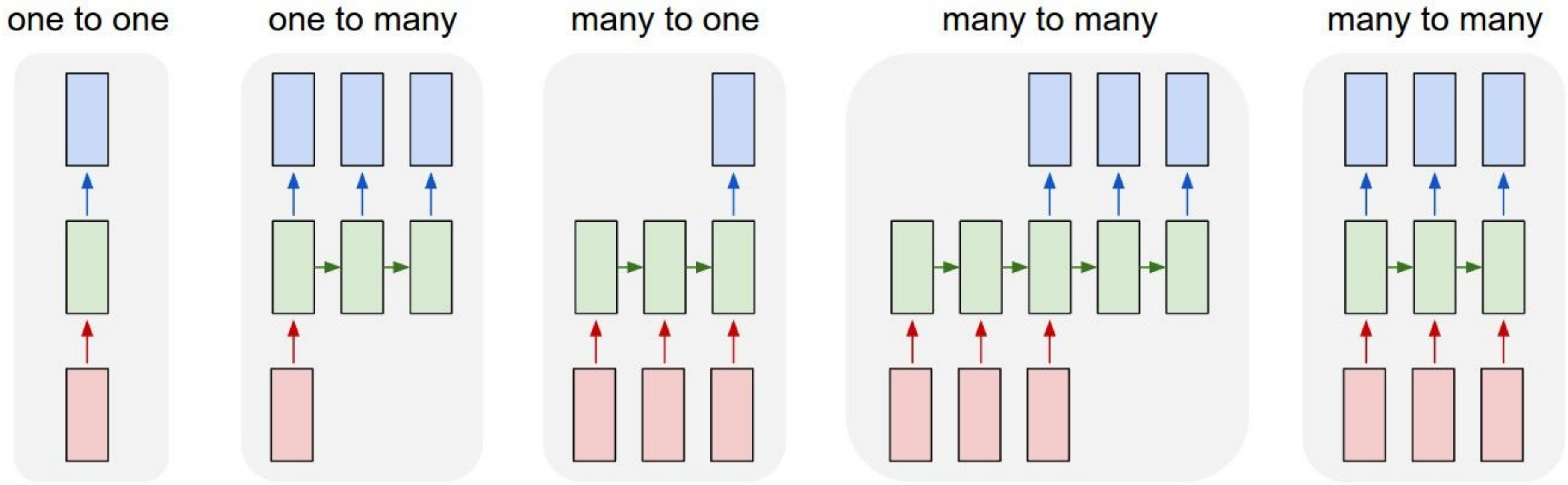
Machine Learning Tasks

- Sequence Prediction
  - Predict next element in sequence: $(x_1, x_2, \ldots, x_i) \rightarrow x_{i+1}$

- Sequence Labelling/Seq-to-Seq
  - Assign label to each element in sequence: $(x_1, x_2, \ldots, x_i) \rightarrow (y_1, y_2, \ldots y_i)$

- Sequence Classification
  - Classify entire sequence $(x_1, x_2, \ldots, x_i) \rightarrow y$

Machine Learning Tasks



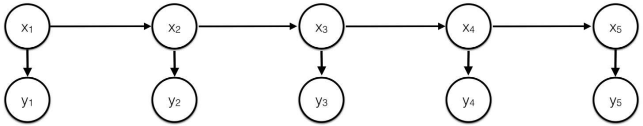
Hidden Markov Model (HMM)

Hidden Markov Models

$p(y_t|x_t)$ observation probability

$p(x_t|x_{t-1})$ transition probability

$$p(X, Y) = p(x_1) \prod_{t=1}^{T-1} p(x_{t+1}|x_t) \prod_{t'=1}^{T} p(y_{t'}|x_{t'})$$

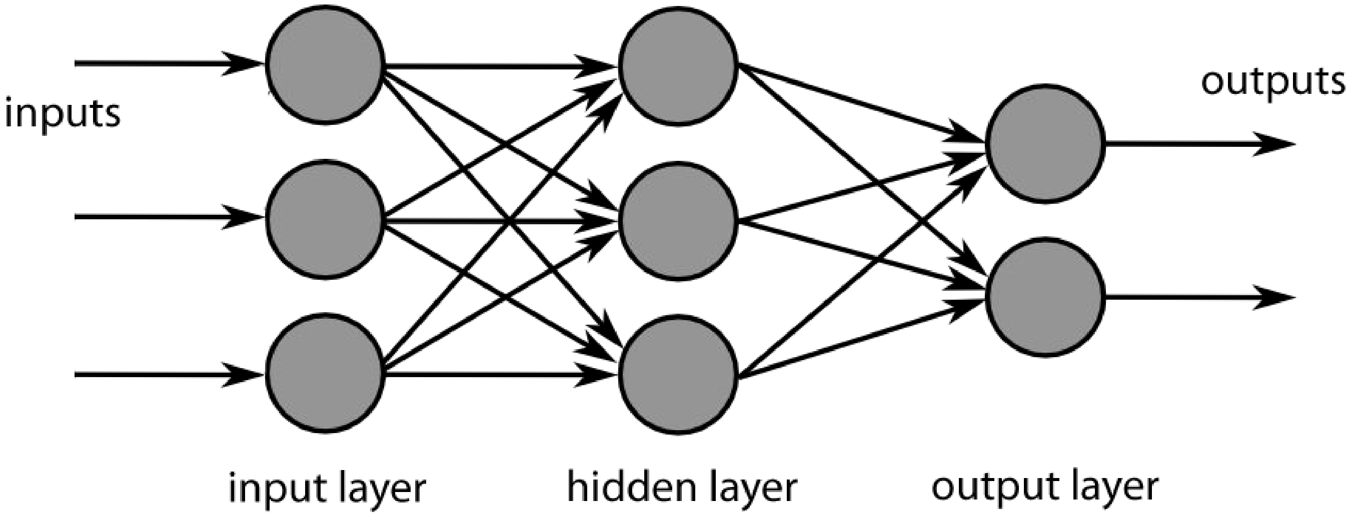


- Generative Model
- Markov Assumption on Hidden States

Hidden Markov Model (HMM)

- Weaknesses
    - N states can only encode log(N) bits of information
    - In practice, generative models tend to be more computationally intensive/less accurate

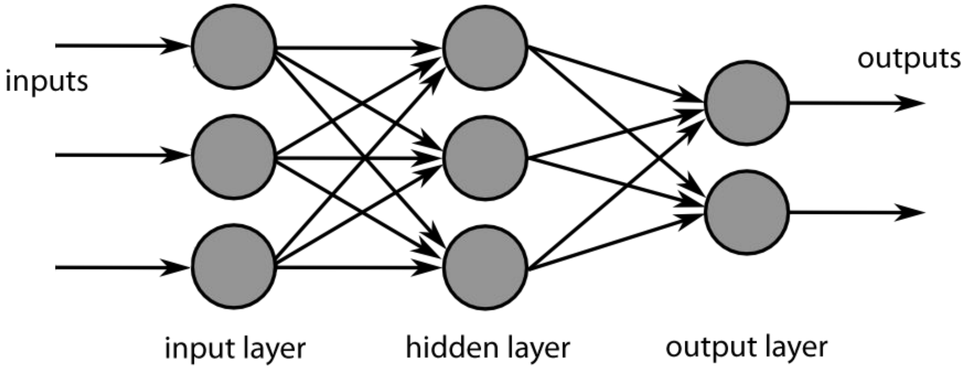
Feedforward Neural Networks



$$s = f(U \cdot x)$$

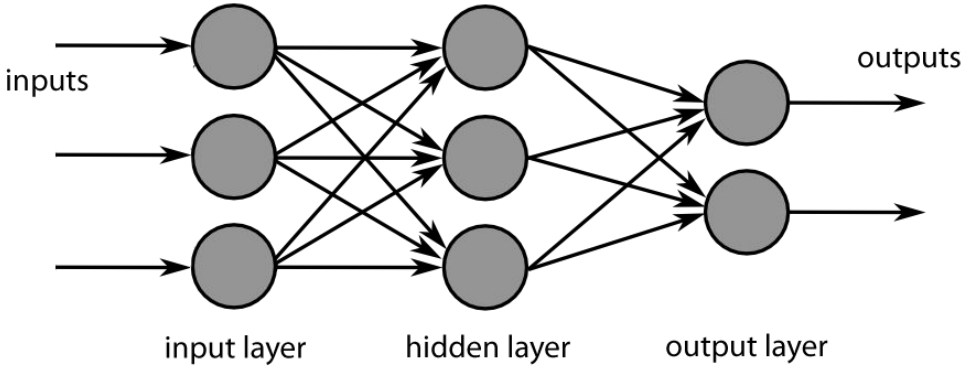
$$o = softmax(V \cdot s)$$

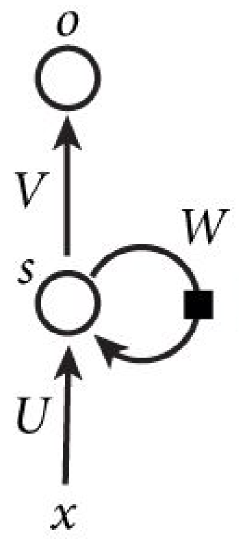
Weaknesses of Feedforward Networks



- Fixed # outputs and inputs
- Inputs are independent

Recurrent Neural Networks Overcome These Problems

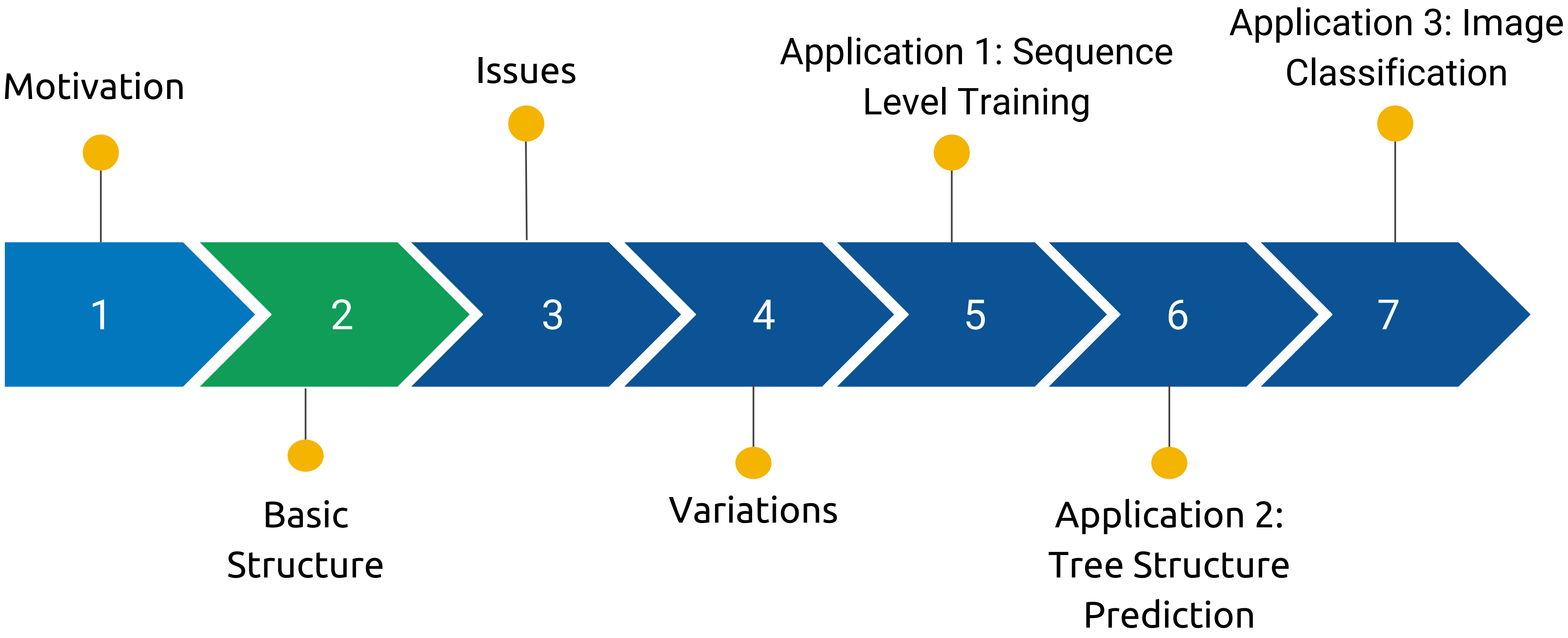




- Fixed # outputs and inputs
- Inputs are independent

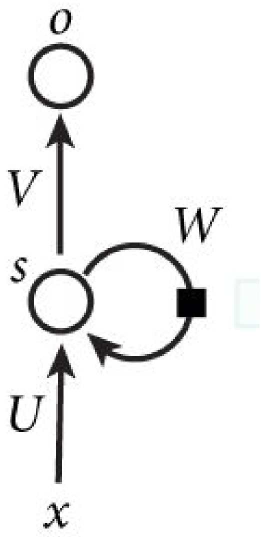
- Variable # outputs and inputs
- Inputs can be correlated

Roadmap


Motivation
Basic Structure
Issues
Variations
Application 1: Sequence Level Training
Application 2: Tree Structure Prediction
Application 3: Image Classification
1
2
3
4
5
6
7

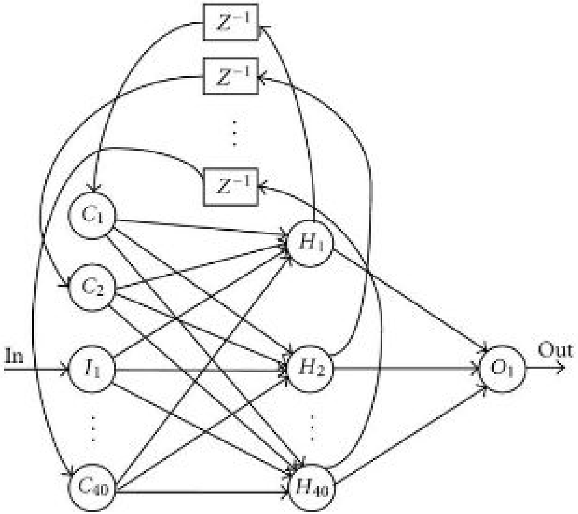
Recurrent Neural Network - Structure

- RNNs are neural networks with loops
- Previous hidden states or predictions used as inputs

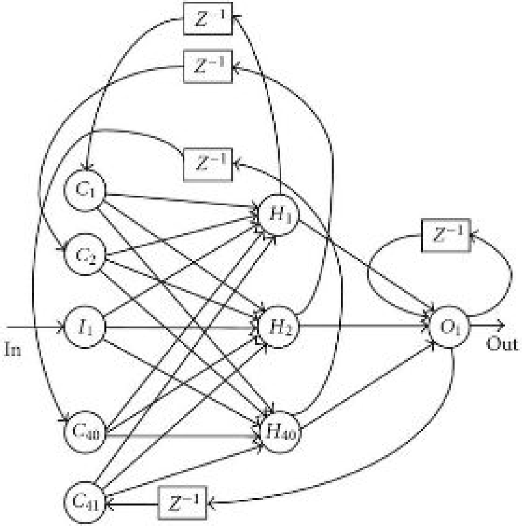


Recurrent Neural Network - Structure

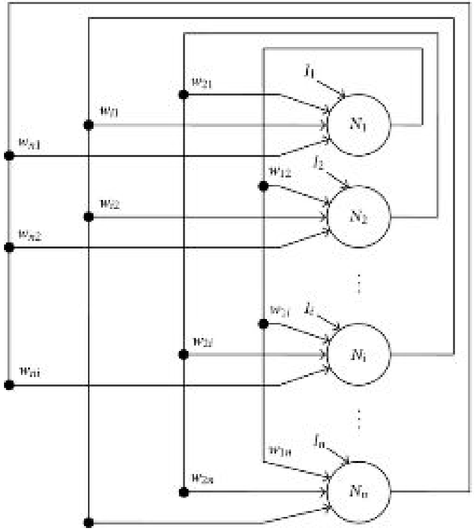
- How you loop the nodes depends on the problem



Elman Neural Network



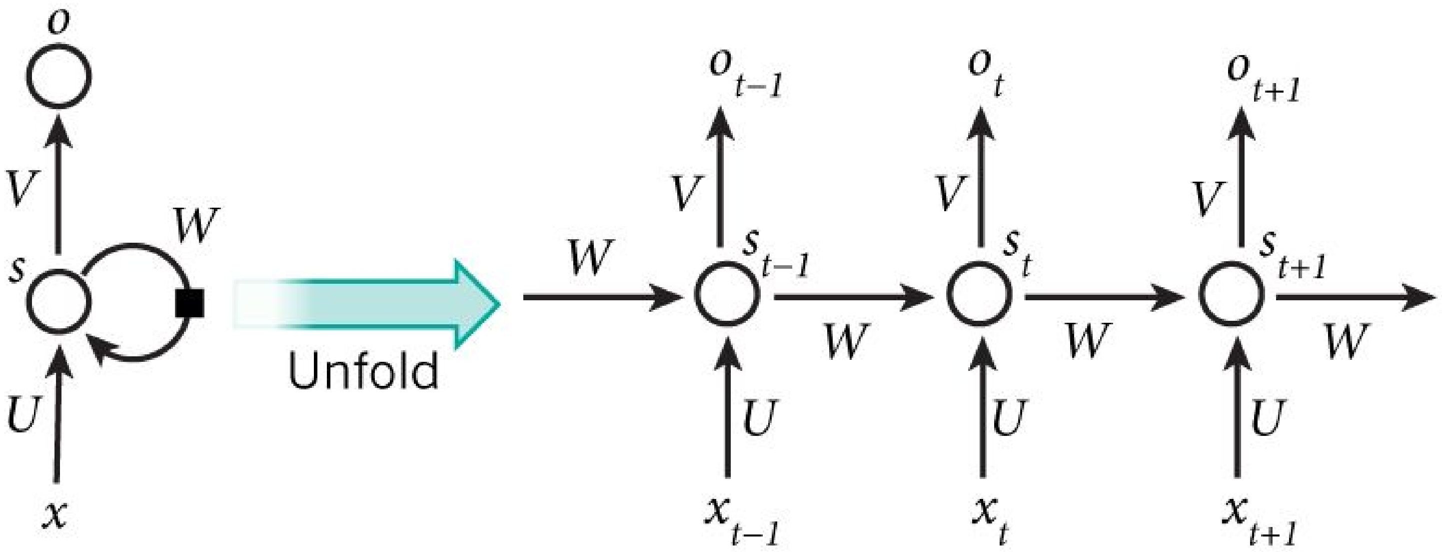
Jordan Neural Network



Hopfield Neural Network

Recurrent Neural Networks - Training

- “Unroll” the network through time
  - Becomes a feedforward network
- Train using gradient descent
  - backpropagation through time (BPTT)

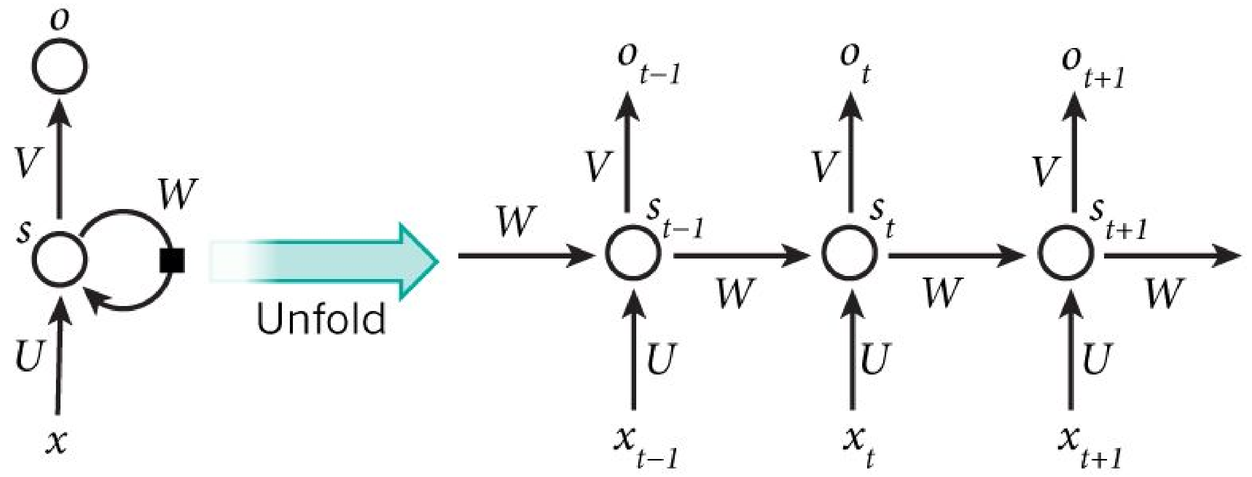


Recurrent Neural Network - Equations

Updating Hidden States s_t at time t

$$s_t = f(Ux_t + Ws_{t-1})$$

- s_t function of previous hidden state s_{t-1} and current input x_t
  - encodes prior information ("memory")

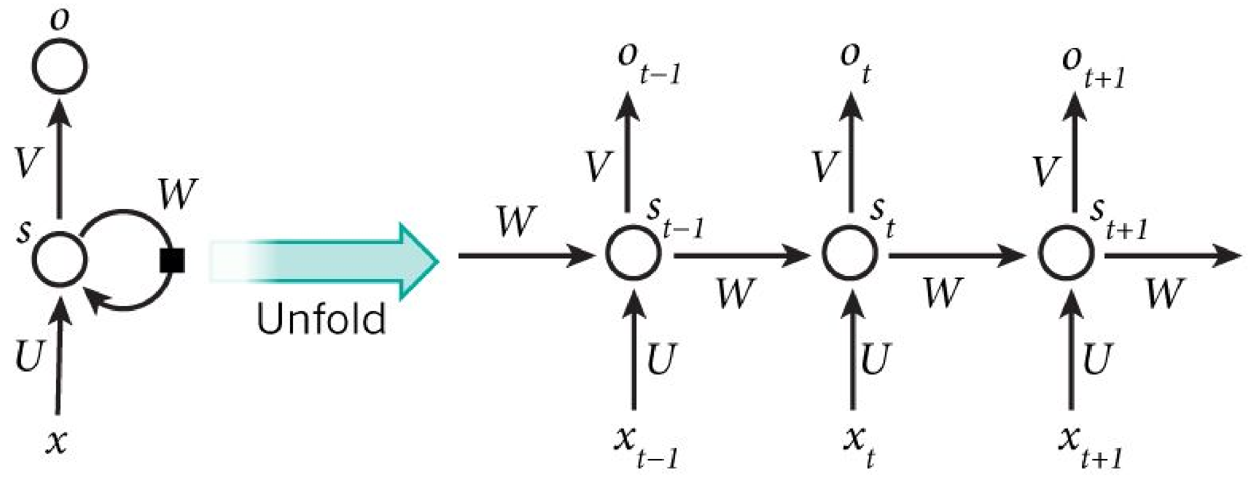


Recurrent Neural Network - Equations

Making Sequential Predictions o_t at time t

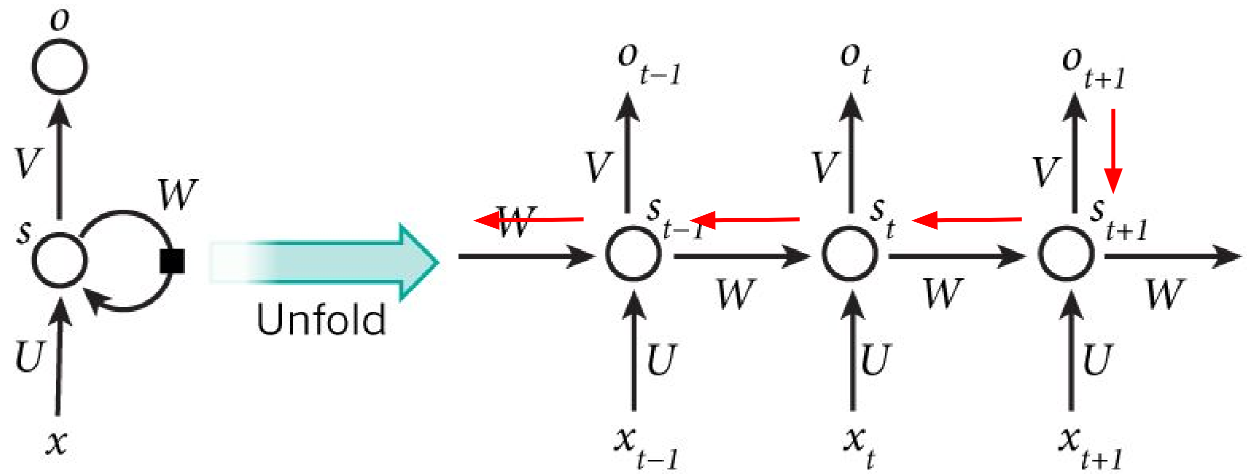
$$o_t = \mathrm{softmax}(V s_t)$$

- o_t is function of current hidden state s_t



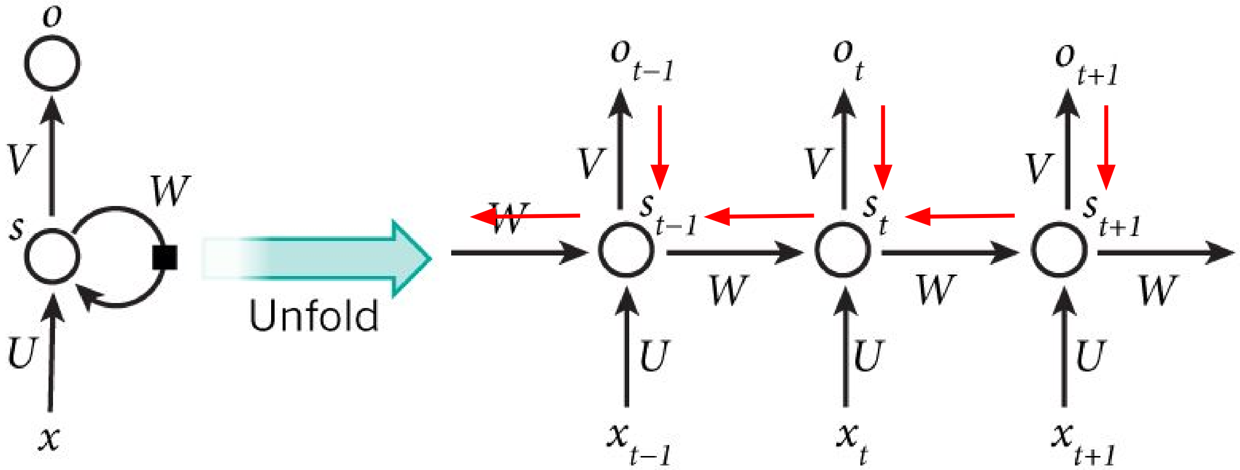
Backpropagation - Chain Rule

$$\frac{\partial \mathcal{L}_{t+1}}{\partial W} = \frac{\partial \mathcal{L}_{t+1}}{\partial o_{t+1}} \frac{\partial o_{t+1}}{\partial s_{t+1}} \frac{\partial s_{t+1}}{\partial W}$$

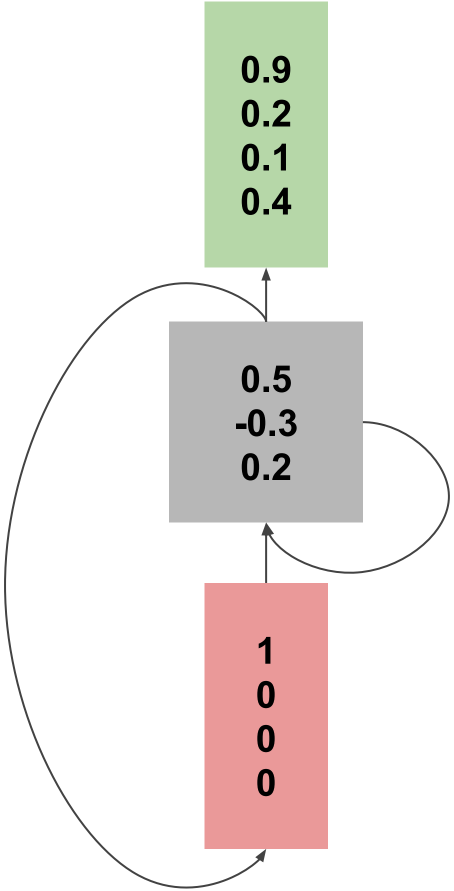


Backpropagation - Chain Rule

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}_t}{\partial W}$$



Example 1 - Learning A Language Model

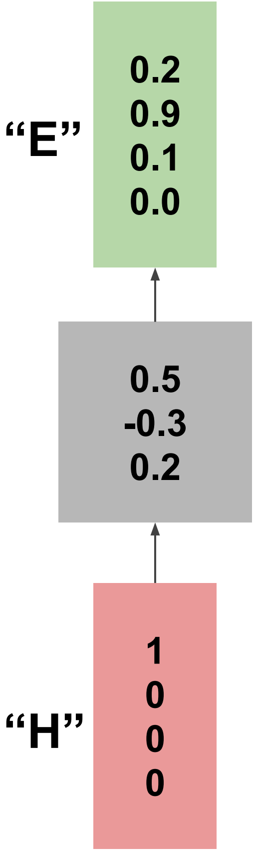


Predicted distribution over letters

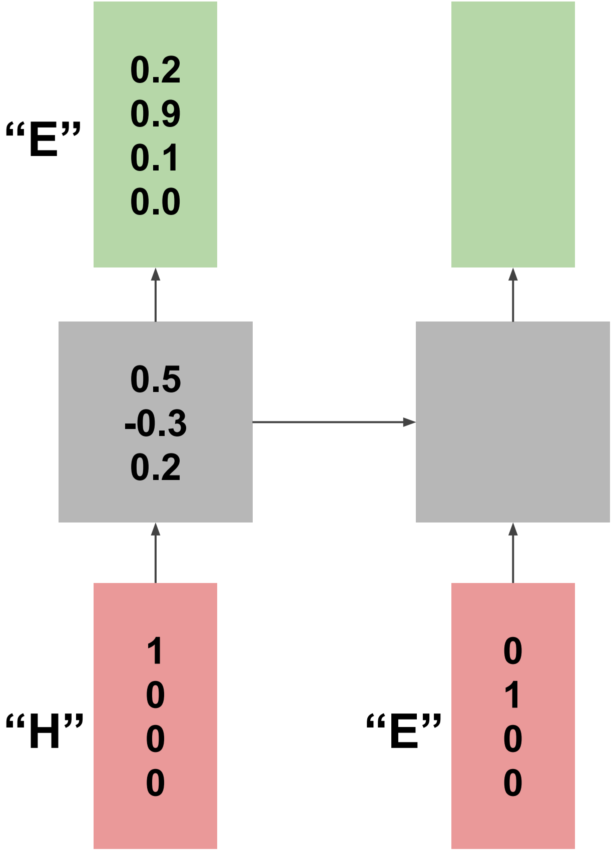
Hidden states

One-hot encoding of letters

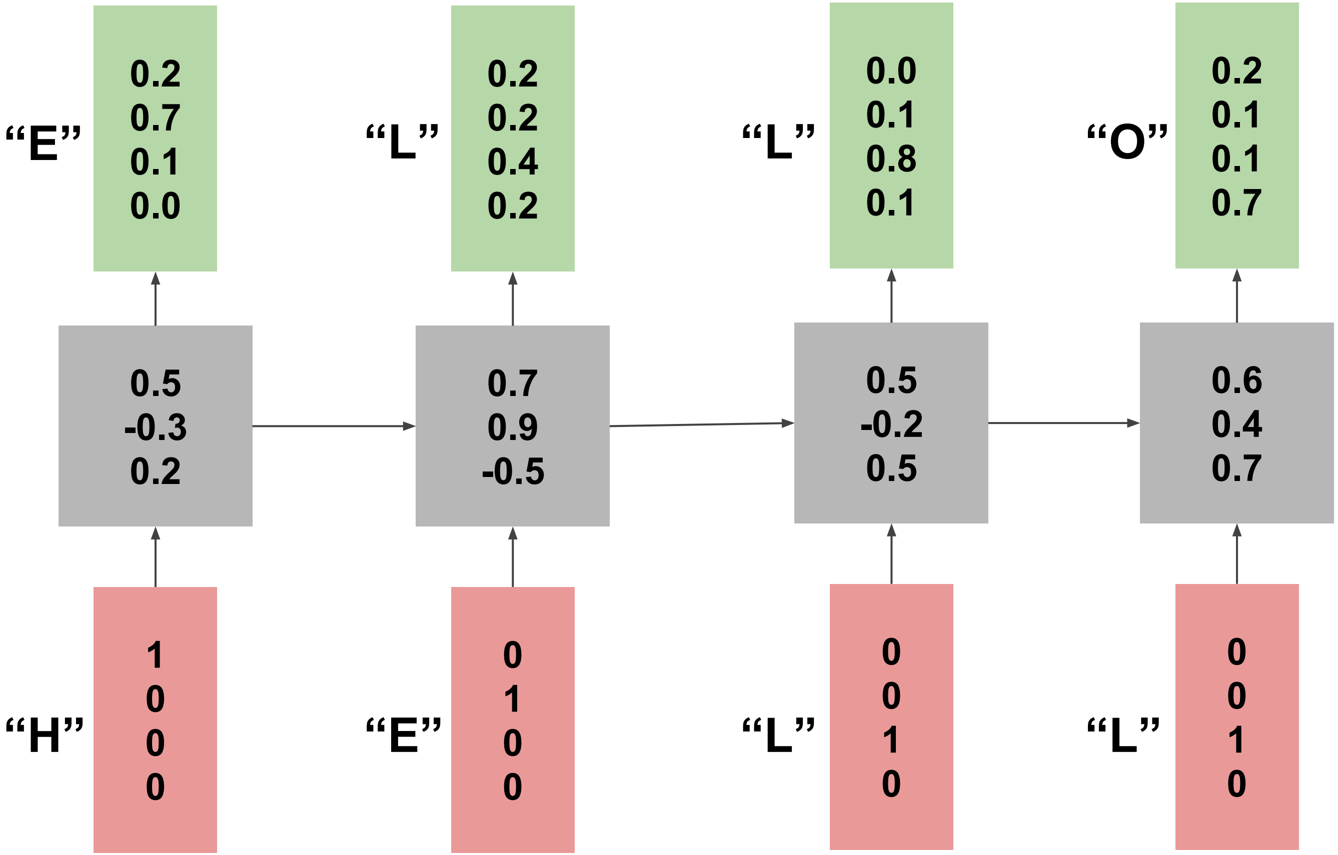
Example 1 - Learning A Language Model



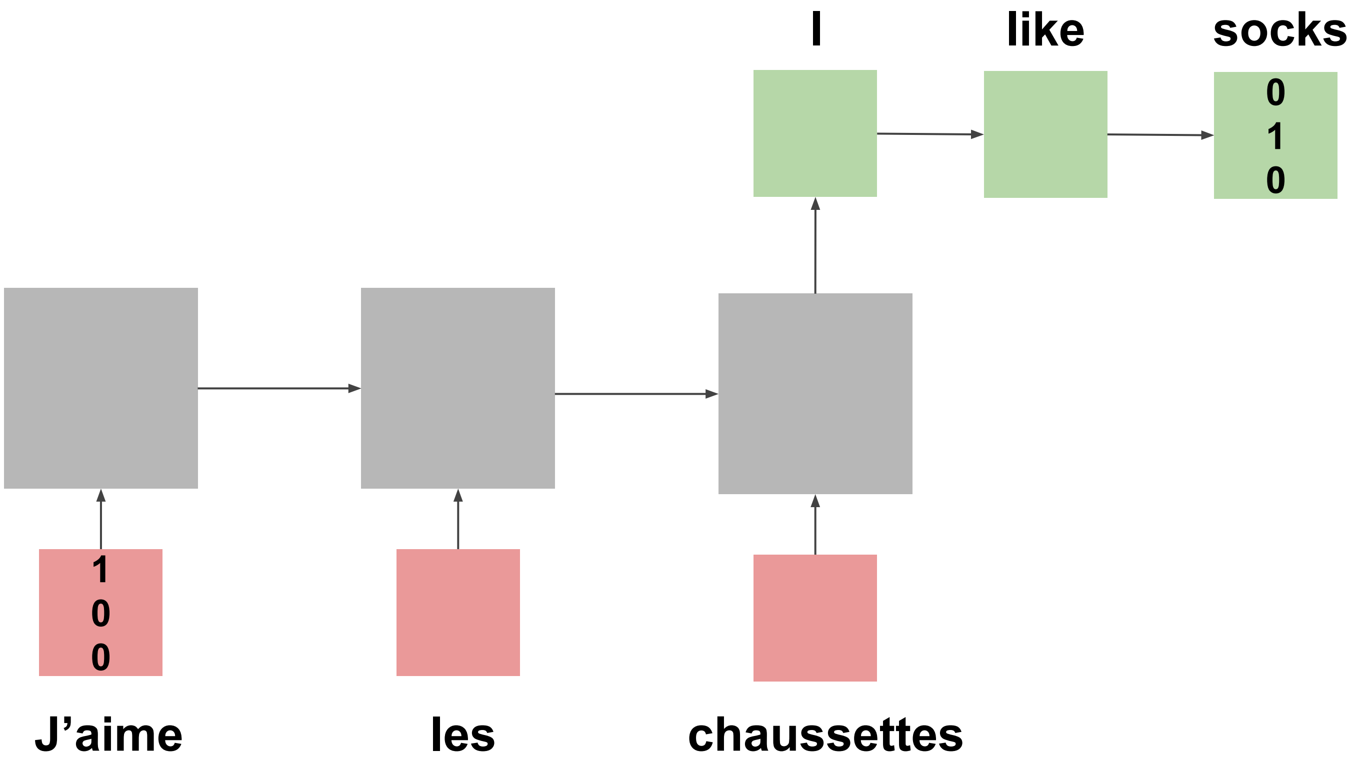
Example 1 - Learning A Language Model



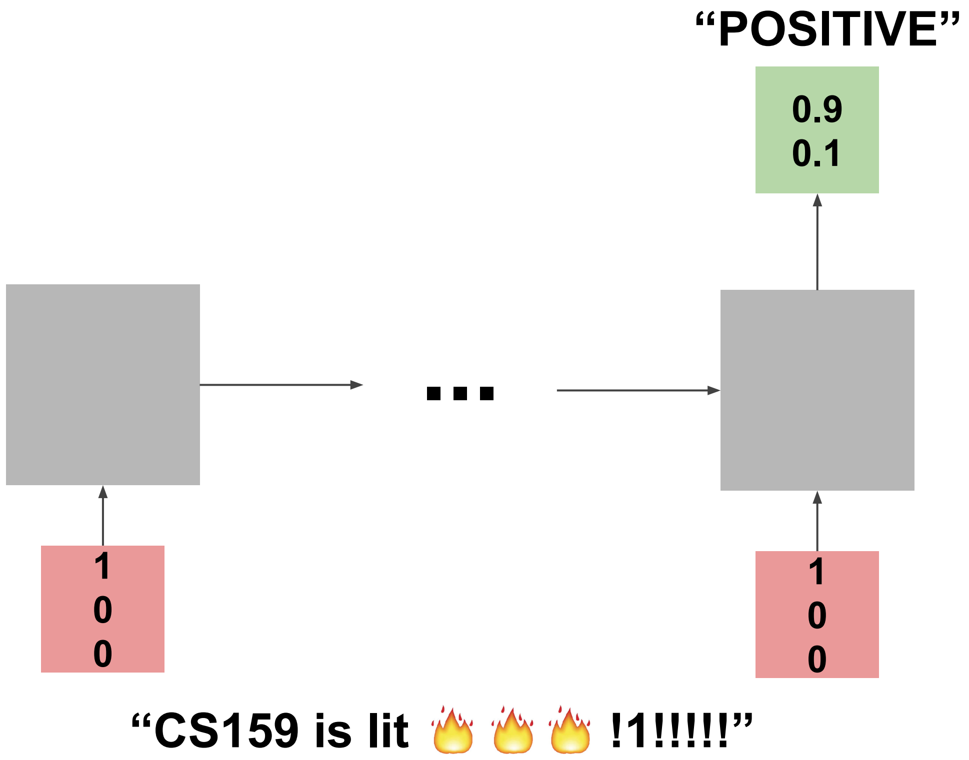
Example 1 - Learning A Language Model



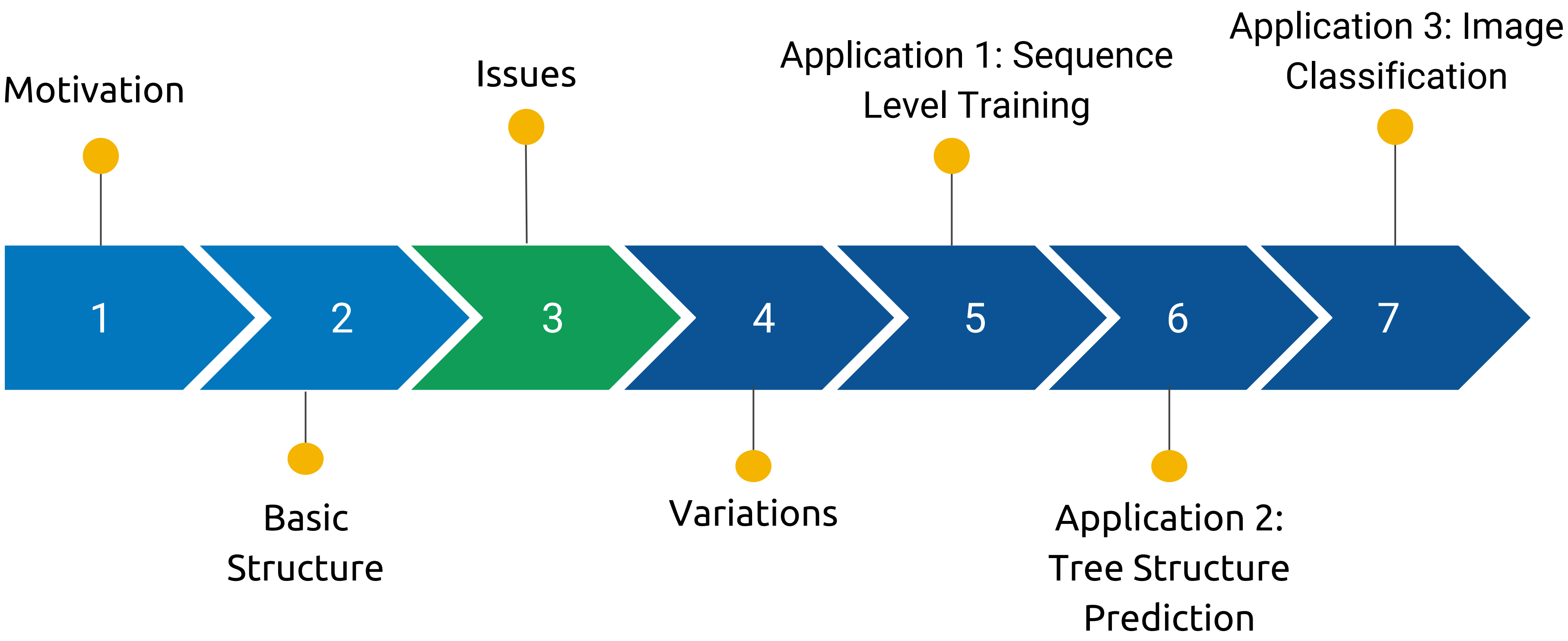
Example 2 - Machine Translation



Example 3 - Sentiment Analysis



Roadmap

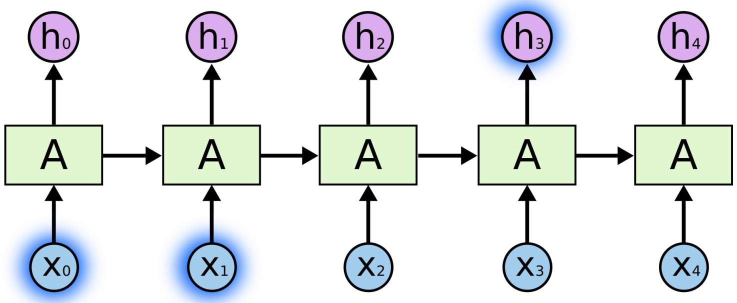

Motivation
Basic Structure
Issues
Variations
Application 1: Sequence Level Training
Application 2: Tree Structure Prediction
Application 3: Image Classification
1
2
3
4
5
6
7

Vanishing and Exploding Gradient

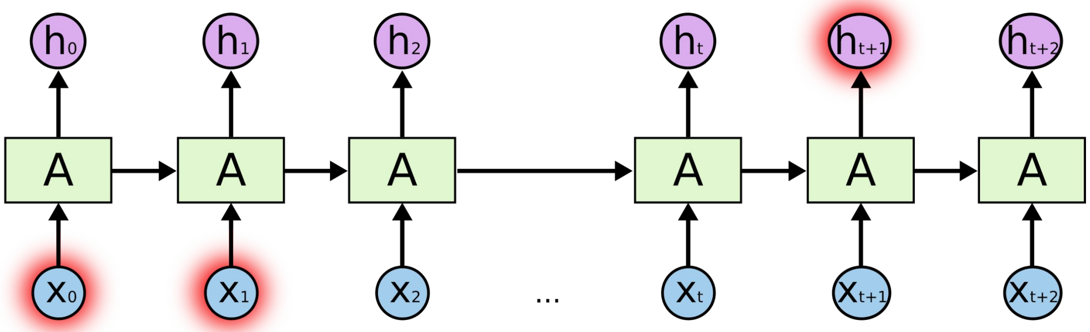
- Composition of many nonlinear functions can lead to problems
- $s_t = f(Ux_t + Ws_{t-1})$
- When we backpropagate, gradient increases exponentially with *W*
- Behavior depends on eigenvalues of *W*
  - If eigenvalues > 1, W^n may cause gradient to explodes
  - If eigenvalues < 1, W^n may cause gradient to vanish

Short-term Memory

- In practice, simple RNN's tend to only retain information from few time-steps in the past

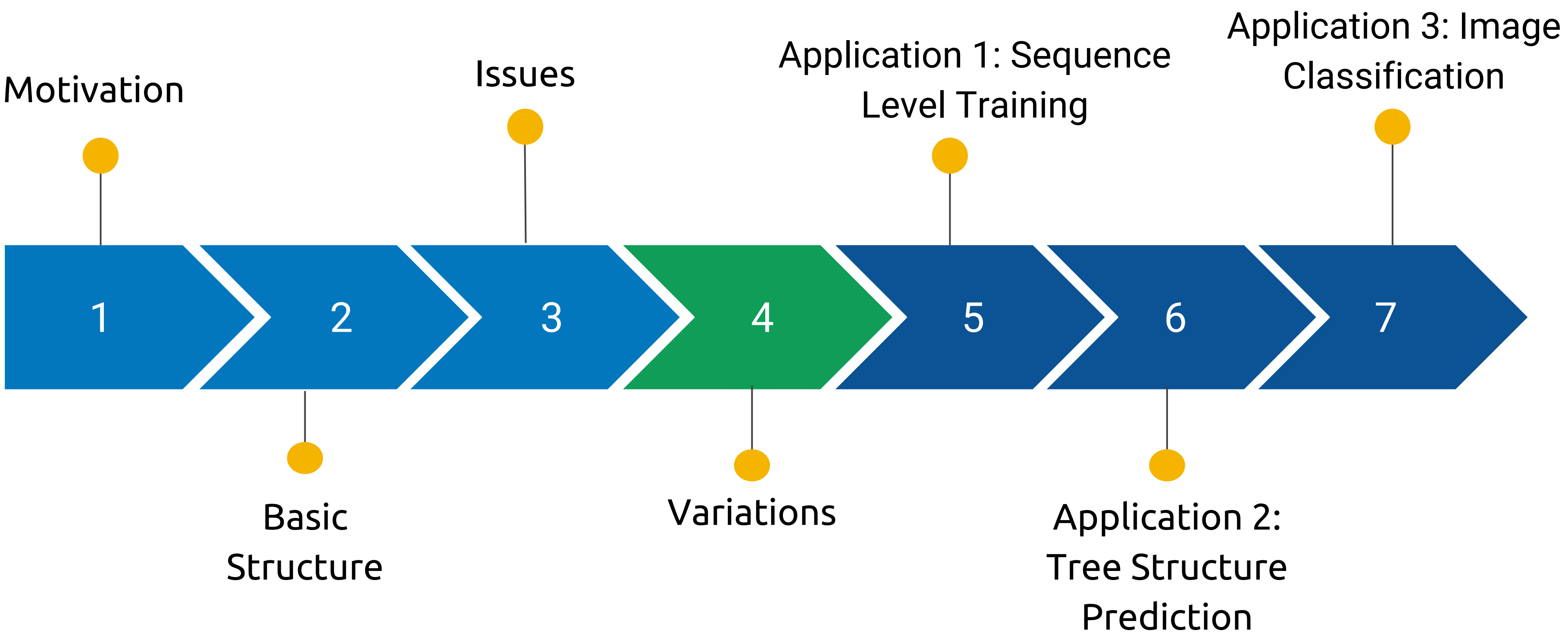


RNN's are good at remembering recent information



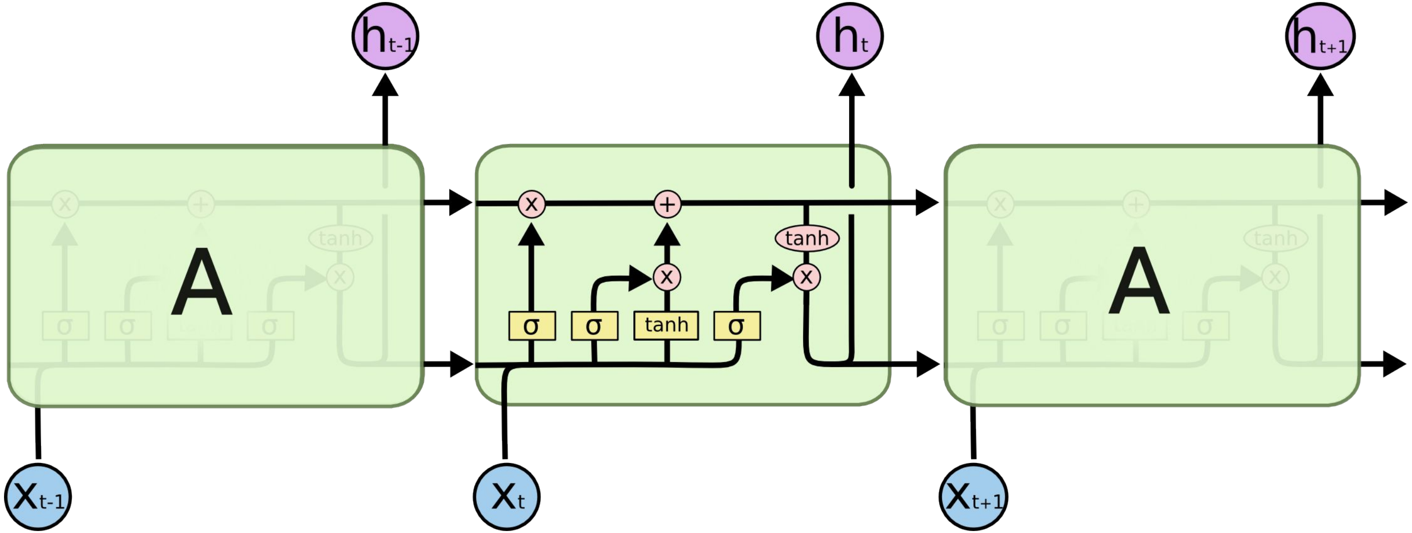
RNN's have a harder time remembering relevant information from farther back

Roadmap

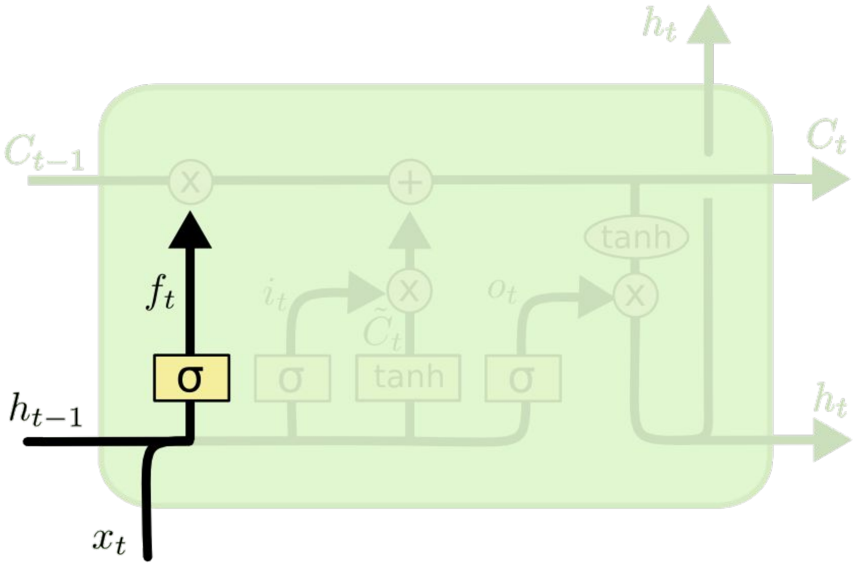

Motivation
Basic Structure
Issues
Variations
Application 1: Sequence Level Training
Application 2: Tree Structure Prediction
Application 3: Image Classification
1
2
3
4
5
6
7

Long short-term memory (LSTM)

- Adds a memory cell with input, forget, and output gates
- Can help learn long-term dependencies
- Helps solve exploding/vanishing gradient problem

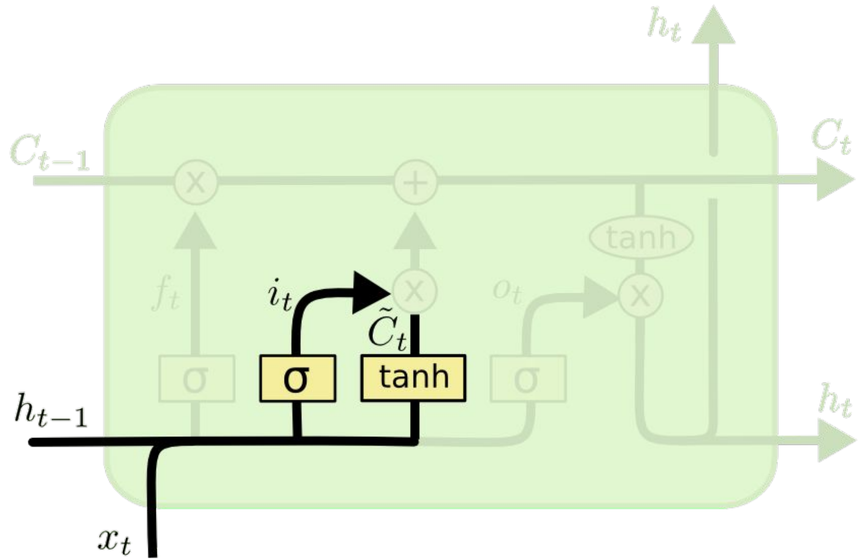


Forget gate layer



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

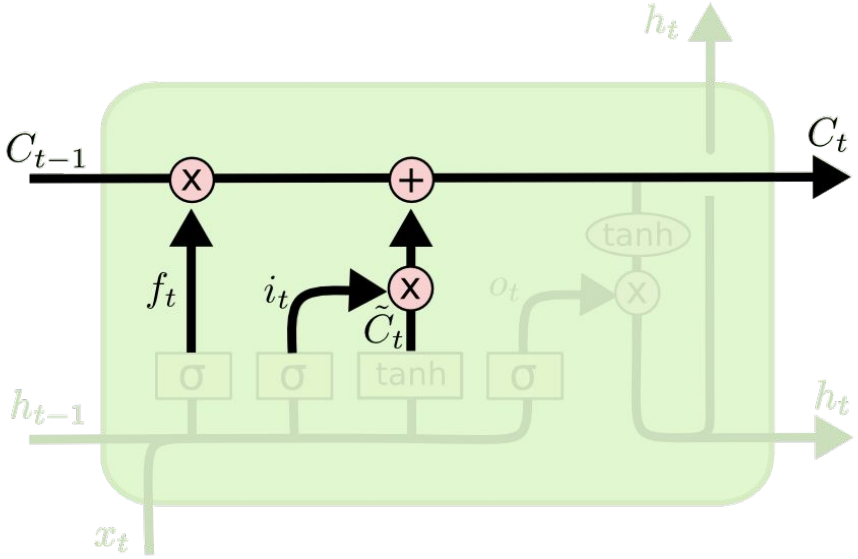
Input gate layer



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

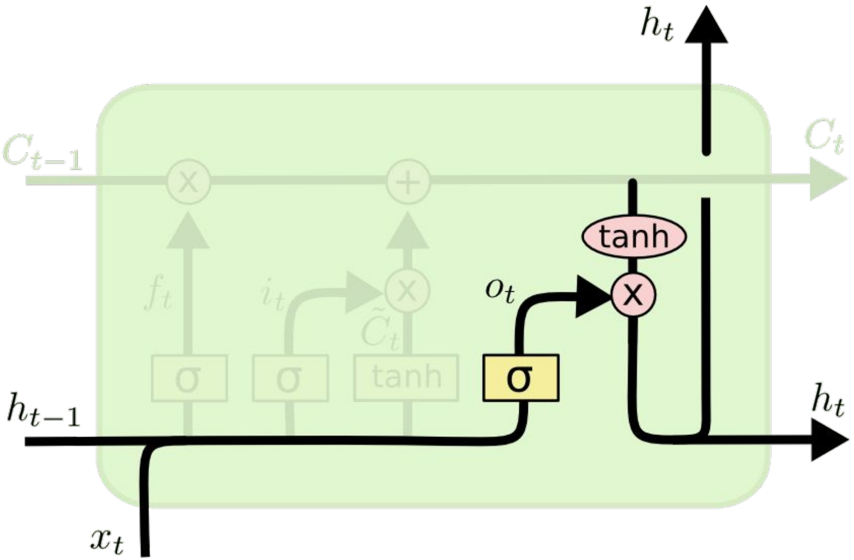
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Update cell state



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Generate filtered output



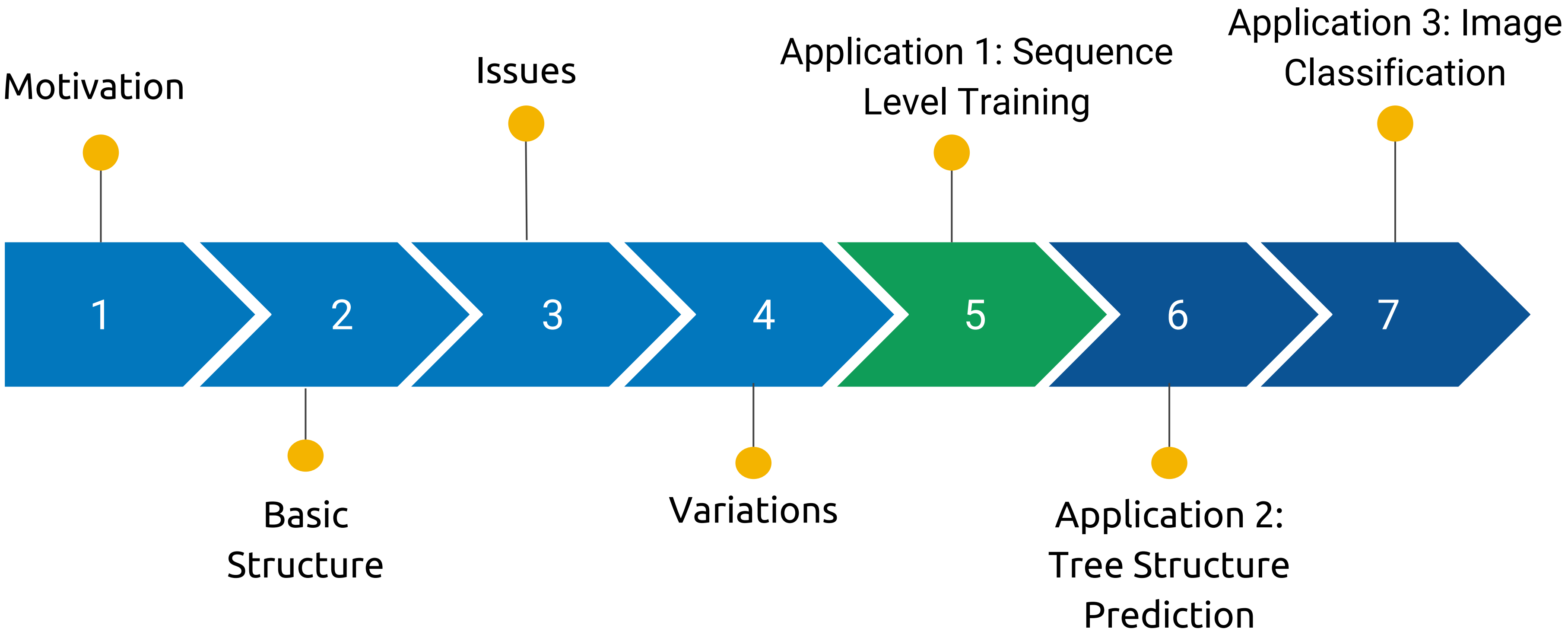
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

But wait... there's more

- A number of challenging problems remain in sequence learning
- Let's take a look at how the papers address these issues

Roadmap


Motivation
1
Basic Structure
2
Issues
3
Variations
4
Application 1: Sequence Level Training
5
Application 2: Tree Structure Prediction
6
Application 3: Image Classification
7

SEQUENCE LEVEL TRAINING WITH RECURRENT NEURAL NETWORKS

By: Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, Wojciech Zaremba

Text Generation

- Consider the problem of text generation
- Machine Translation
- Summarization
- Question Answering

Text Generation

- We want to predict a sequence of words $[w_1, w_2, \ldots, w_T]$ (that makes sense)
- At each time step, may take as input some context c_t

RNN

- Let's use an RNN

$$\mathbf{h}_{t+1} = \phi_\theta(w_t, \mathbf{h}_t, \mathbf{c}_t),$$
$$w_{t+1} \sim p_\theta(w|w_t, \mathbf{h}_{t+1}) = p_\theta(w|w_t, \phi_\theta(w_t, \mathbf{h}_t, \mathbf{c}_t))$$

Text Generation

- Simple Elman RNN:

$$\mathbf{h}_{t+1} = \sigma(M_i \mathbf{1}(w_t) + M_h \mathbf{h}_t + M_c \mathbf{c}_t),$$
$$\mathbf{o}_{t+1} = M_o \mathbf{h}_{t+1},$$
$$w_{t+1} \sim \text{softmax}(\mathbf{o}_{t+1}),$$

Training

- Optimize cross-entropy loss at each time-step
- Given a target sequence [w_1, w_2, …, w_T]

$$L = -\log p(w_1, \ldots, w_T) = -\log \prod_{t=1}^{T} p(w_t | w_1, \ldots, w_{t-1}) = -\sum_{t=1}^{T} \log p(w_t | w_1, \ldots, w_{t-1})$$

Why is this a problem?

- Notice that the RNN is trained to maximize $p_\theta(w|w_t,h_{t+1})$, where w_t is the ground truth
- Loss function causes model to be good at predicting the next word given previous ground-truth word

Why is this a problem?

- But we don’t have the ground truth at test time!!!
- Remember that we’re trying to generate sequences
- At test time, the model needs to use it’s own predictions to generate the next words

Why is this a problem?

- This can lead to predicting sub-optimal sequences

$$\prod_{t=1}^{T} \max_{w_{t+1}} p_\theta(w_{t+1}|w_t^g, \mathbf{h}_{t+1}) \leq \max_{w_1,\ldots,w_T} \prod_{t=1}^{T} p_\theta(w_{t+1}|w_t^g, \mathbf{h}_{t+1})$$

- The most likely sequence might actually contain a sub-optimal word at some time-step

Techniques

- Beam Search
  - At each point instead of just taking highest scoring word, look at k next word candidates
  - Significantly slows down word generation process

- Data as Demonstrator
  - During training, with certain probability take either model prediction or ground truth
  - Alignment issues
    - “I took a long walk” vs. “I took a walk”

- End to End Backprop
  - Instead of ground-truth, propagate top-k words at previous time-step
  - Weigh each word by its score and re-normalize

Reinforcement Learning

- Suppose our RNN is an agent
  - Parameters define a policy
  - Picks an action

- After taking an action, updates internal states

- At the end of sequence, observes a reward
  - Can use any reward function, authors use BLUE and ROUGE-2

REINFORCE

- We want to find parameters that maximize expected reward

- Loss is negative expected reward

$$L_\theta = -\sum_{w_1^g,\ldots,w_T^g} p_\theta(w_1^g,\ldots,w_T^g) r(w_1^g,\ldots,w_T^g) = -\mathbb{E}_{[w_1^g,\ldots w_T^g]\sim p_\theta} r(w_1^g,\ldots,w_T^g),$$

- In practice, we actually approximate the expected reward with a single sample...

REINFORCE

- For estimating the gradients, $$\frac{\partial L_\theta}{\partial \theta} = \sum_t \frac{\partial L_\theta}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \theta}$$

$$\frac{\partial L_\theta}{\partial \mathbf{o}_t} = \left(r(w_1^g, \ldots, w_T^g) - \bar{r}_{t+1}\right) \left(p_\theta(w_{t+1}|w_t^g, \mathbf{h}_{t+1}, \mathbf{c}_t) - \mathbf{1}(w_{t+1}^g)\right)$$

- $\bar{r}_{t+1}$ is a baseline estimator that can reduce variance
  - Authors use linear regression with the hidden states of RNN as input to estimate this
  - Unclear what is best technique to select this

- If $r > \bar{r}_{t+1}$ encourages word choice, or else discourages word choice

MIXED INCREMENTAL CROSS-ENTROPY REINFORCE

- Instead of starting from poor random policy, start from RNN trained using ground truth with cross-entropy
- Start using REINFORCE according to an annealing schedule

Data: a set of sequences with their corresponding context.
Result: RNN optimized for generation.
Initialize RNN at random and set N^{XENT}, $N^{\text{XE+R}}$ and Δ;
for $s = T, 1, -\Delta$ **do**
    if $s == T$ **then**
        train RNN for N^{XENT} epochs using XENT only;
    else
        train RNN for $N^{\text{XE+R}}$ epochs. Use XENT loss in the first s steps, and REINFORCE (sampling from the model) in the remaining $T - s$ steps;
    end
end

Algorithm 1: MIXER pseudo-code.

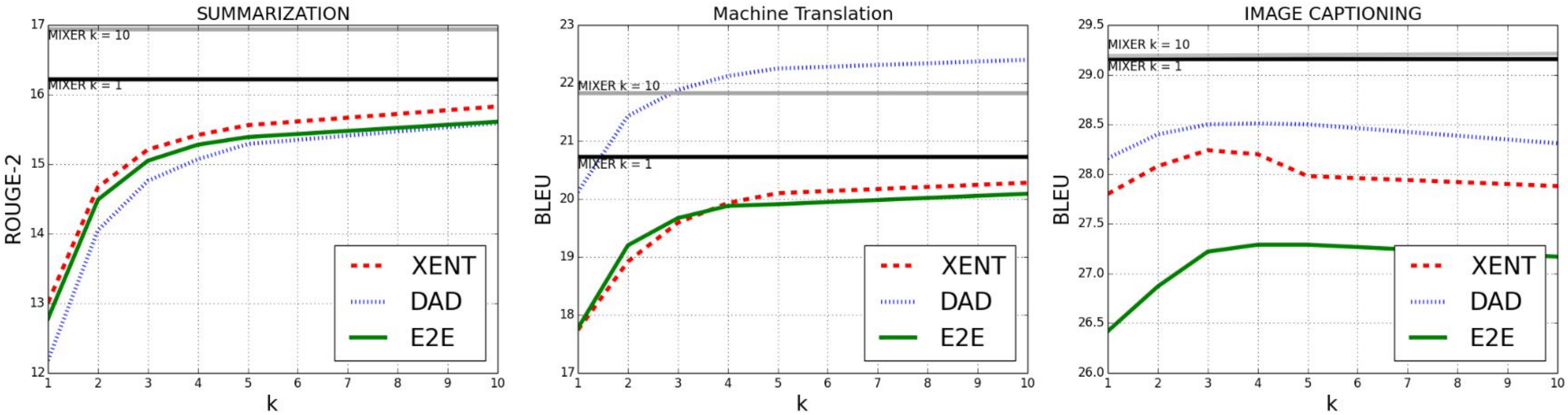
Results

TASK	XENT	DAD	E2E	MIXER
summarization	13.01	12.18	12.78	**16.22**
translation	17.74	20.12	17.77	**20.73**
image captioning	27.8	28.16	26.42	**29.16**

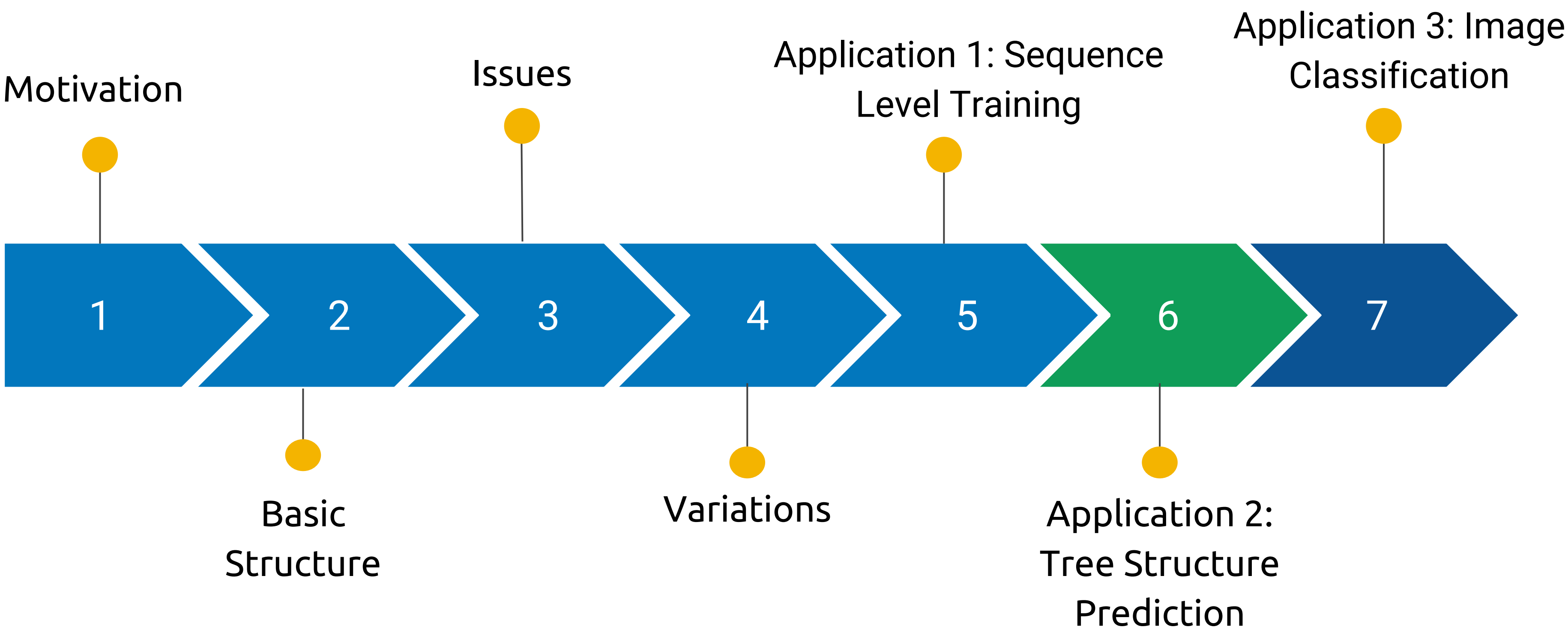
ROUGE-2 score for summarization
BLEU-4 score for translation and image captioning

Results

Algorithms + k Beam Search



Roadmap



Motivation
Issues
Application 1: Sequence Level Training
Application 3: Image Classification
1
2
3
4
5
6
7
Basic Structure
Variations
Application 2: Tree Structure Prediction

Tree-Structured Decoding with Doubly-Recurrent Neural Networks

By: David Alvarez-Melis, Tommi S. Jaakkola

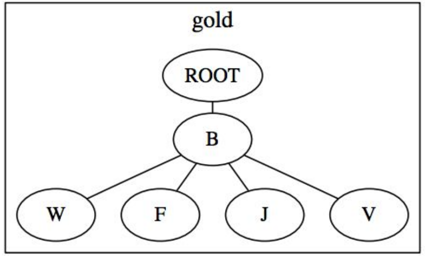
Motivation

- Given an encoding as input, generate a tree structure
- RNN’s best suited for sequential data
  - Trees and graphs do not naturally conform to linear ordering
- Various types of sequential data can be represented in trees
  - Parse trees for sentences
  - Abstract syntax trees for computer programs
- Problem: generate full tree structure with node-labels using encoder-decoder framework

Encoder-Decoder Framework

- Given ground-truth tree and string representation of tree
- Use RNN to encode vector representation from string
- Use RNN to decode tree from vector representation

Example



Input Tree

Preorder Traversal

"ROOT B W F J V"

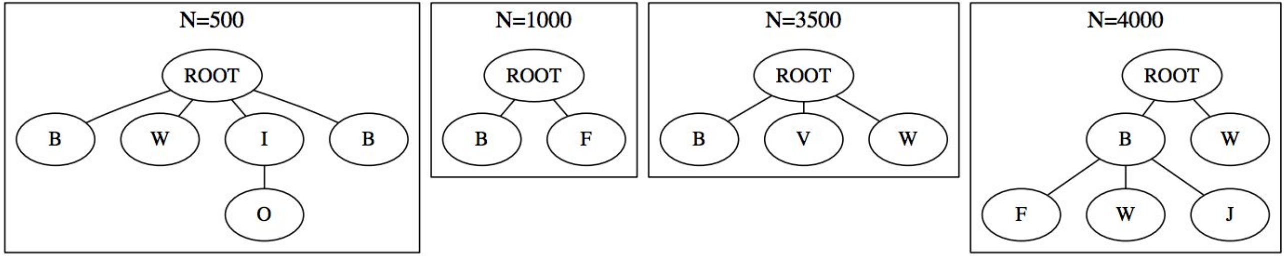
String Representation

Encoder

X

Vector Representation

Decoder

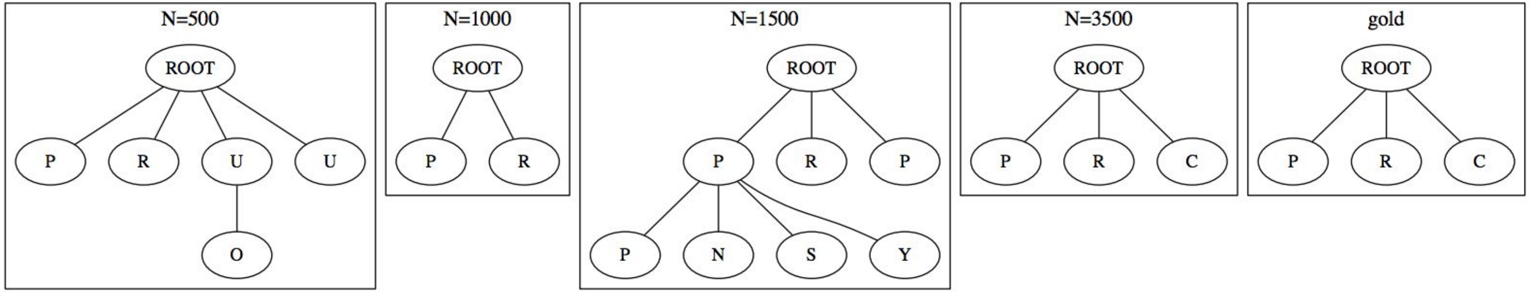


Output

Challenges with decoding

- Must generate tree top-down
  - Don't know node labels beforehand
  - Generating child node vs sibling node

- How to pass information?
  - Label of siblings not independent
  - A verb in a parse tree reduces chance of sibling nodes of being verb

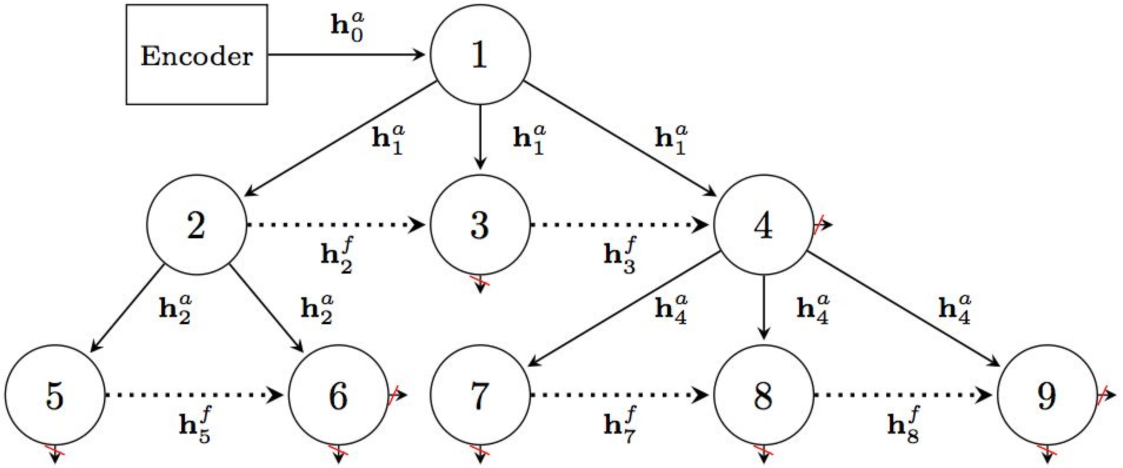


(a) Encoder sentence input: "ROOT P R C"

Doubly Recurrent Neural Networks (DRNN)

- Ancestral and sibling flows of information

- Two input states:
  - Receive from parent node, update, send to descendent
  - Receive from previous sibling, update, send to next sibling



- Unrolled DRNN

- Nodes labeled in order generated

- Solid lines are ancestral, dotted lines are fraternal connections

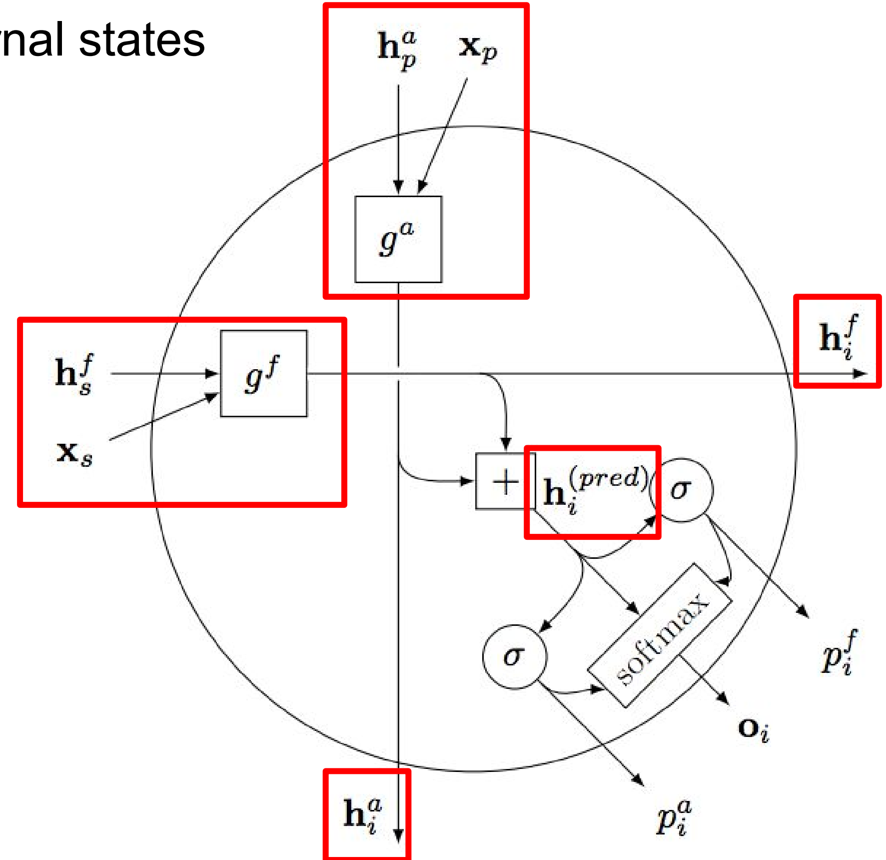
Inside a node

- Update node's hidden ancestral and fraternal states

$$\mathbf{h}_i^a = g^a(\mathbf{h}_{p(i)}^a, \mathbf{x}_{p(i)})$$
$$\mathbf{h}_i^f = g^f(\mathbf{h}_{s(i)}^f, \mathbf{x}_{s(i)})$$

- Combine to obtain predictive hidden state

$$\mathbf{h}_i^{(pred)} = \tanh\left(\mathbf{U}^f \mathbf{h}_i^f + \mathbf{U}^a \mathbf{h}_i^a\right)$$



Producing an output

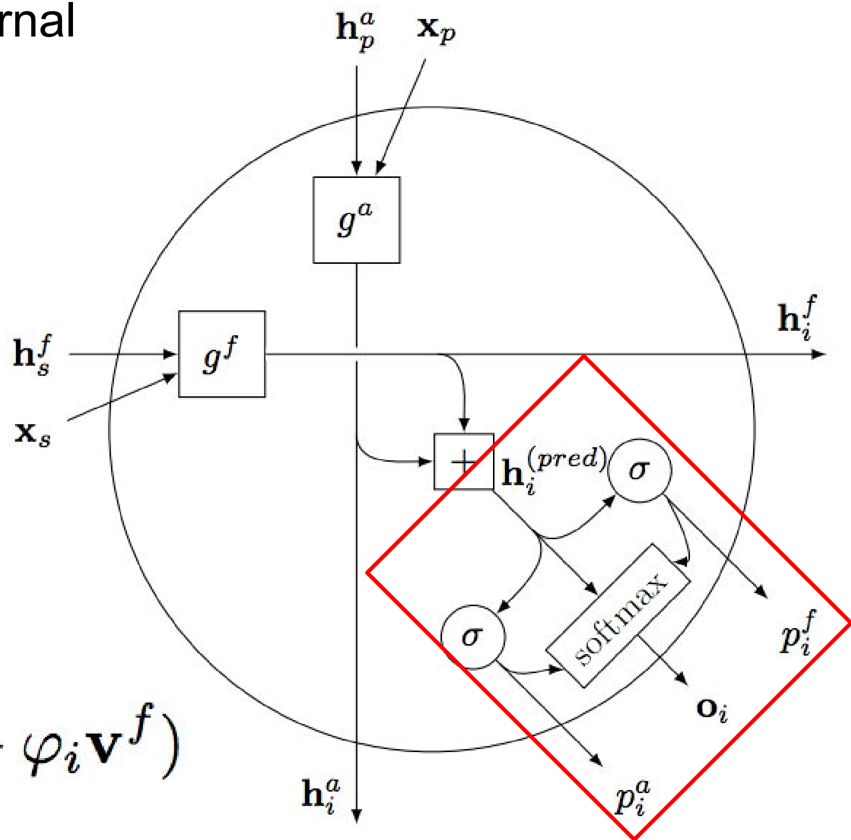
- Get probability of stopping ancestral or fraternal branch

$$p_i^f = \sigma(\mathbf{u}^f \cdot \mathbf{h}_i^{(pred)})$$

$$p_i^a = \sigma(\mathbf{u}^a \cdot \mathbf{h}_i^{(pred)})$$

- Combine to get output
  - α_i, φ_i binary variables corresponding to ground truth (training) or p_i^a , p_i^f (testing)
  - Assign node label

$$\mathbf{o}_i = \text{softmax}(\mathbf{W}\mathbf{h}_i^{(pred)} + \alpha_i \mathbf{v}^a + \varphi_i \mathbf{v}^f)$$



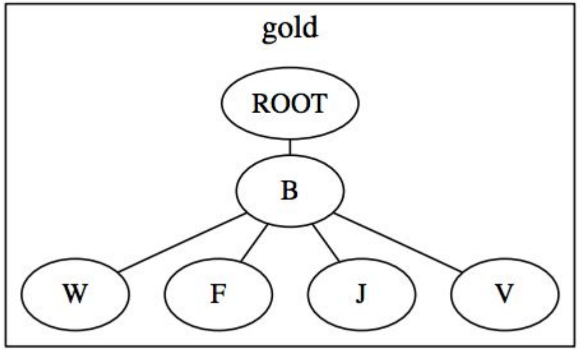
Experiment: Synthetic Tree Recovery

- Generate dataset of labeled trees
  - Vocabulary is 26 letters
  - Condition label of each node on ancestors and siblings
  - Probabilities of children or next-siblings dependent only on label and depth

- Generate string representations with pre-order traversal

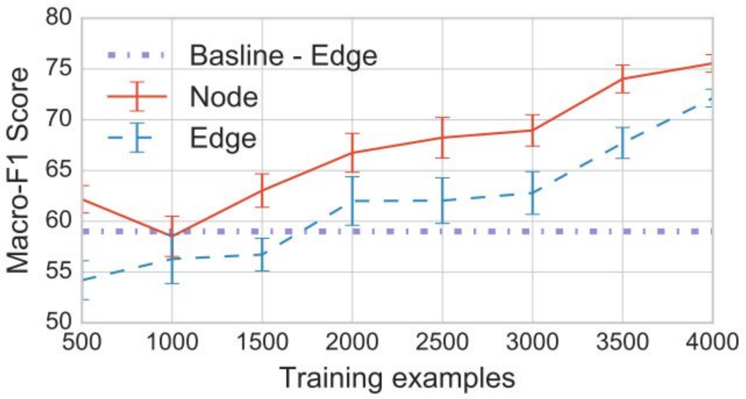
- RNN encoder
  - String to vector

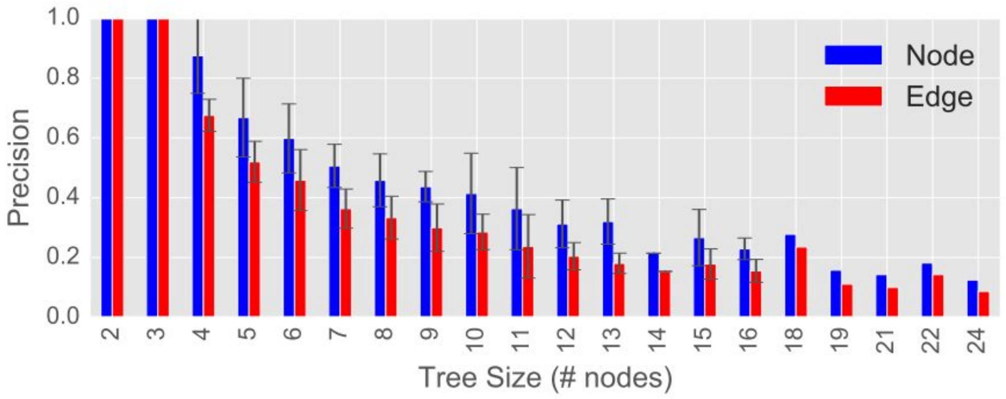
- DRNN with LSTM module decoder
  - Vector to tree



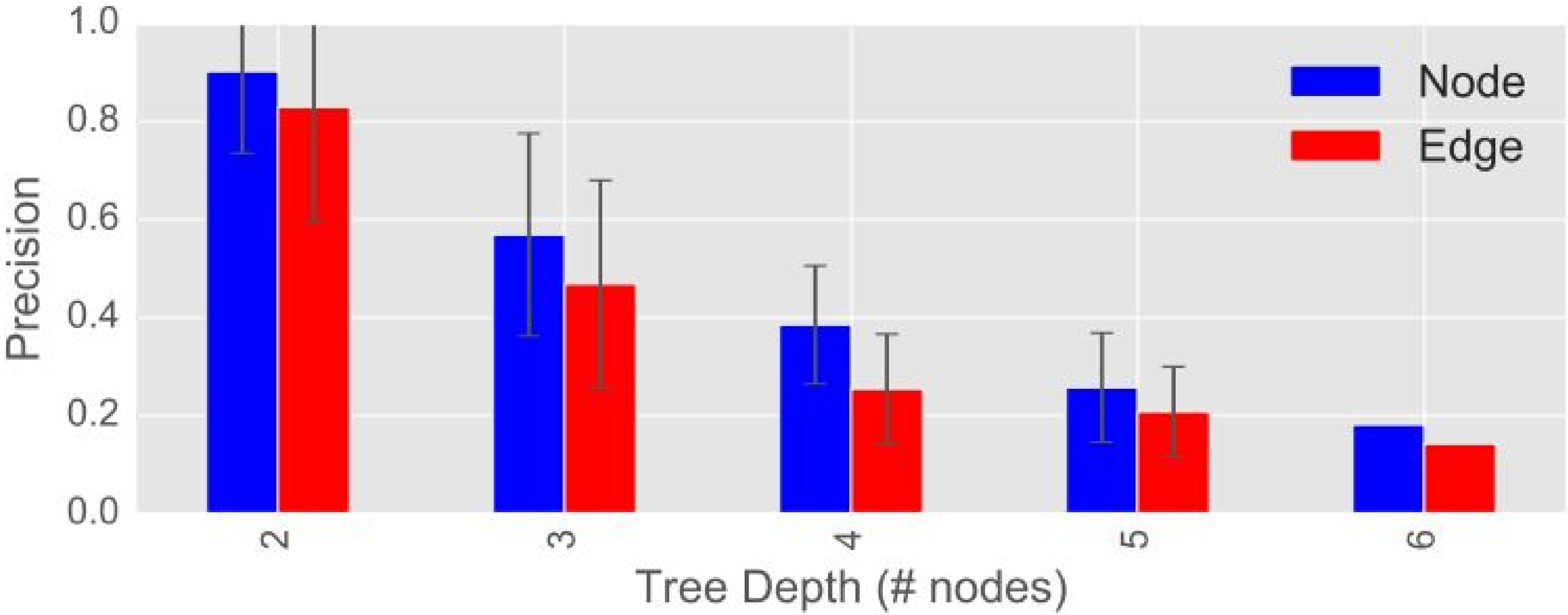
Results

- Node retrieval: 75%
- Edge retrieval: 71%

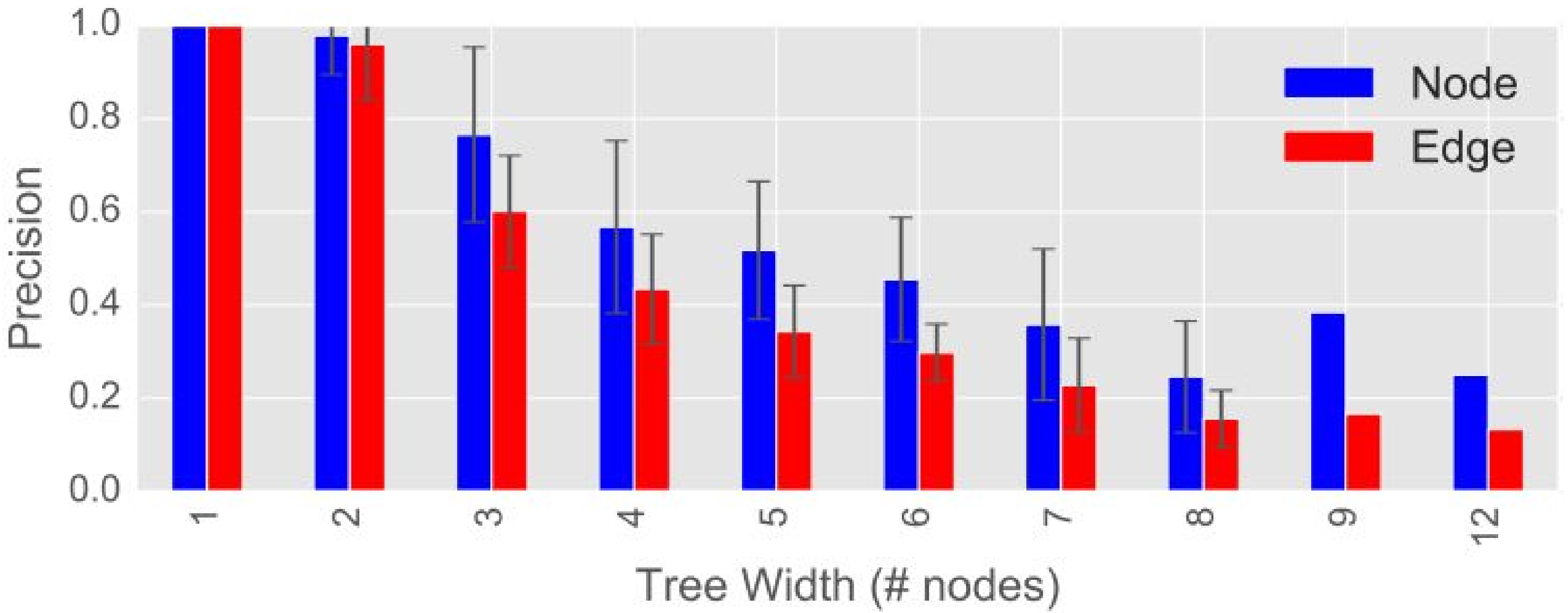




Results



Results

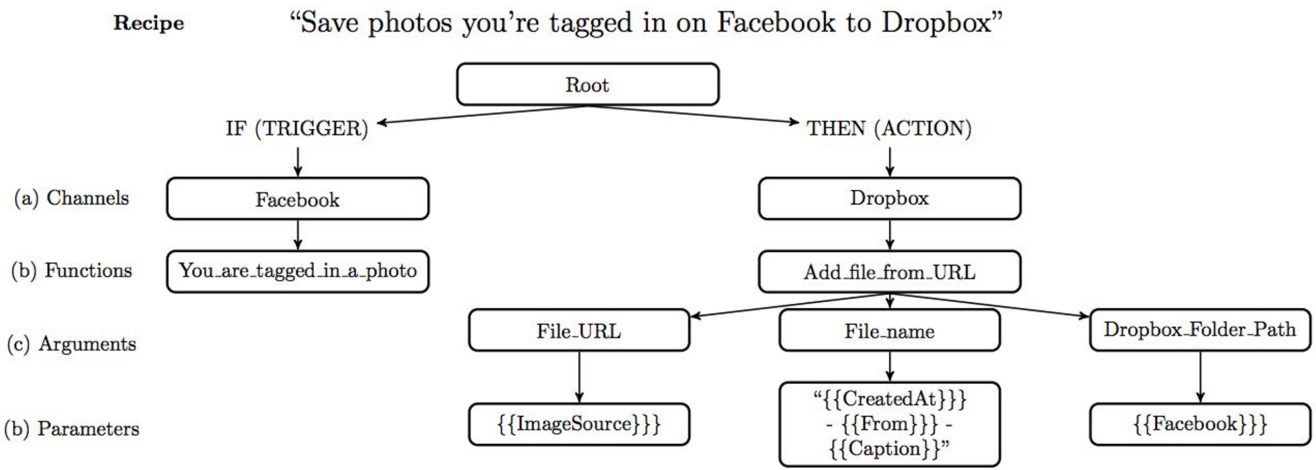


Precision
1.0
0.8
0.6
0.4
0.2
0.0
Node
Edge
1
2
3
4
5
6
7
8
9
12
Tree Width (# nodes)

Experiment: Computer program generation

- Mapping sentences to functional programs
  - Given sentence description of computer program, generate abstract syntax tree

- DRNN performed better than all other baselines



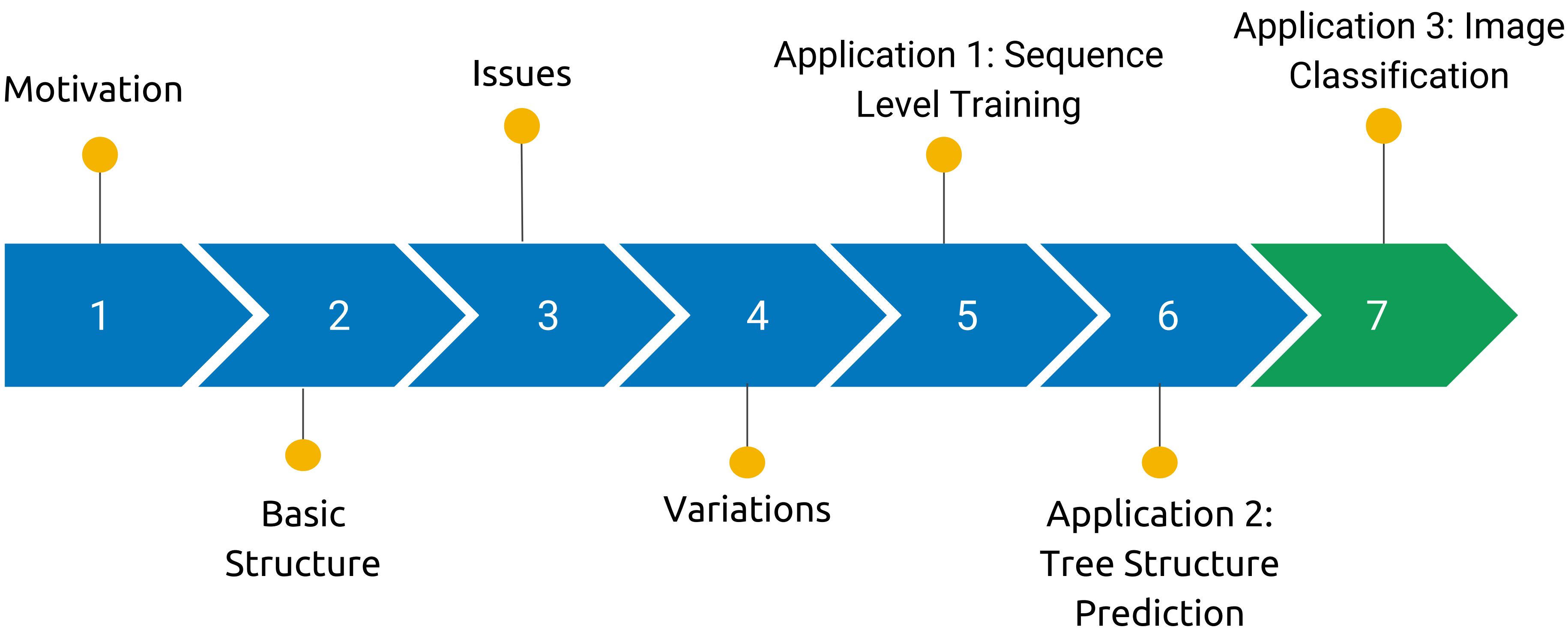
Summary

- DRNN's are an extension of sequential recurrent architectures to tree structures

- Information flow
    - Parent to offspring
    - Sibling to sibling

- Performed better than baselines on tasks involving tree generation

Roadmap


Motivation
Issues
Application 1: Sequence Level Training
Application 3: Image Classification
1
2
3
4
5
6
7
Basic Structure
Variations
Application 2: Tree Structure Prediction

Structure Inference Machines: Recurrent Neural Networks for Analyzing Relations in Group Activity Recognition

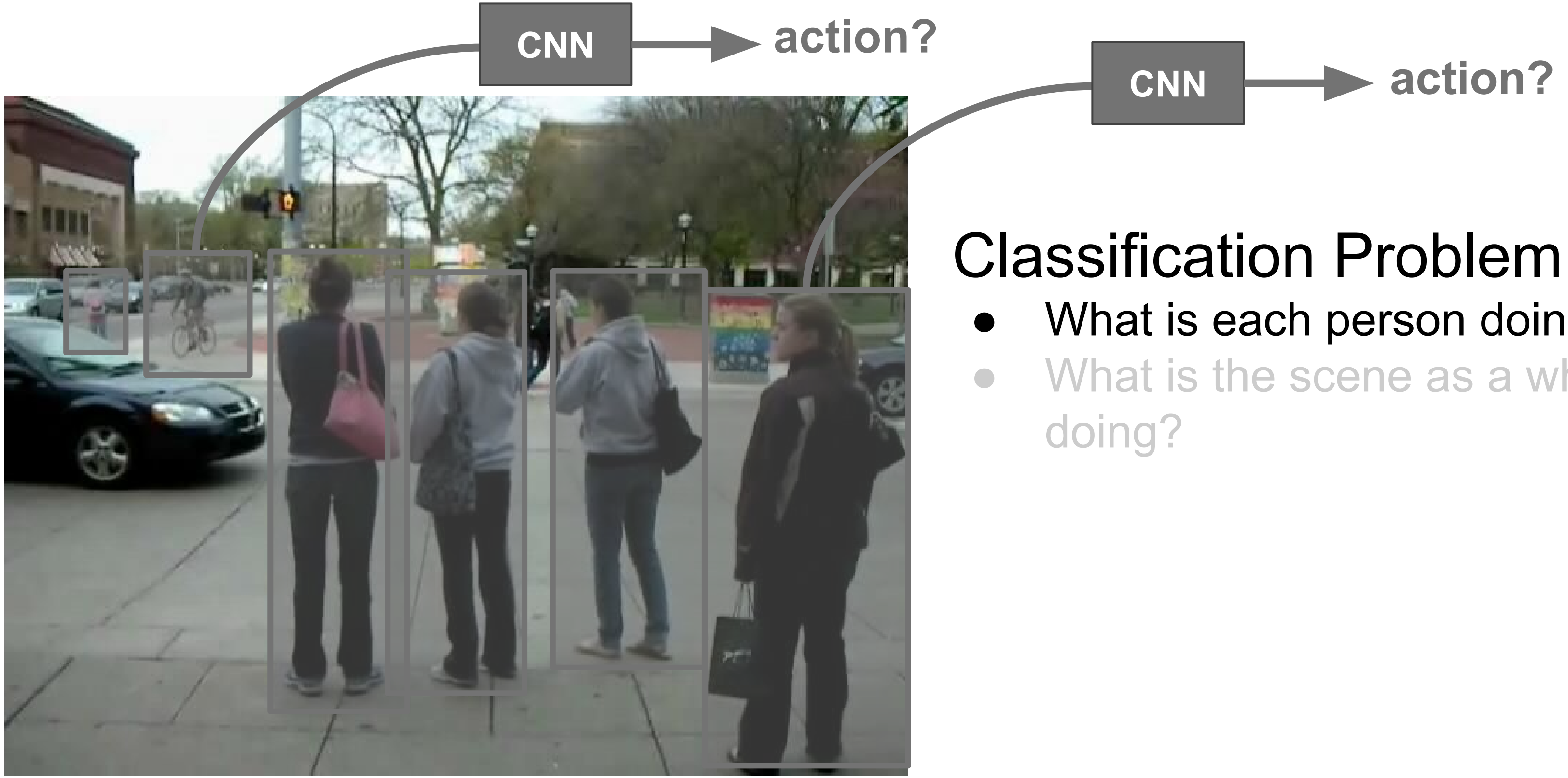
By: Zhiwei Deng, Arash Vahdat, Hexiang Hu, Greg Mori

Group Activity Recognition

Classification Problem:

- What is each person doing?
- What is the scene as a whole doing?

Group Activity Recognition

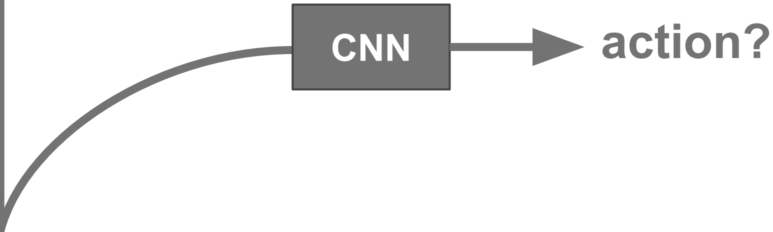


Classification Problem:

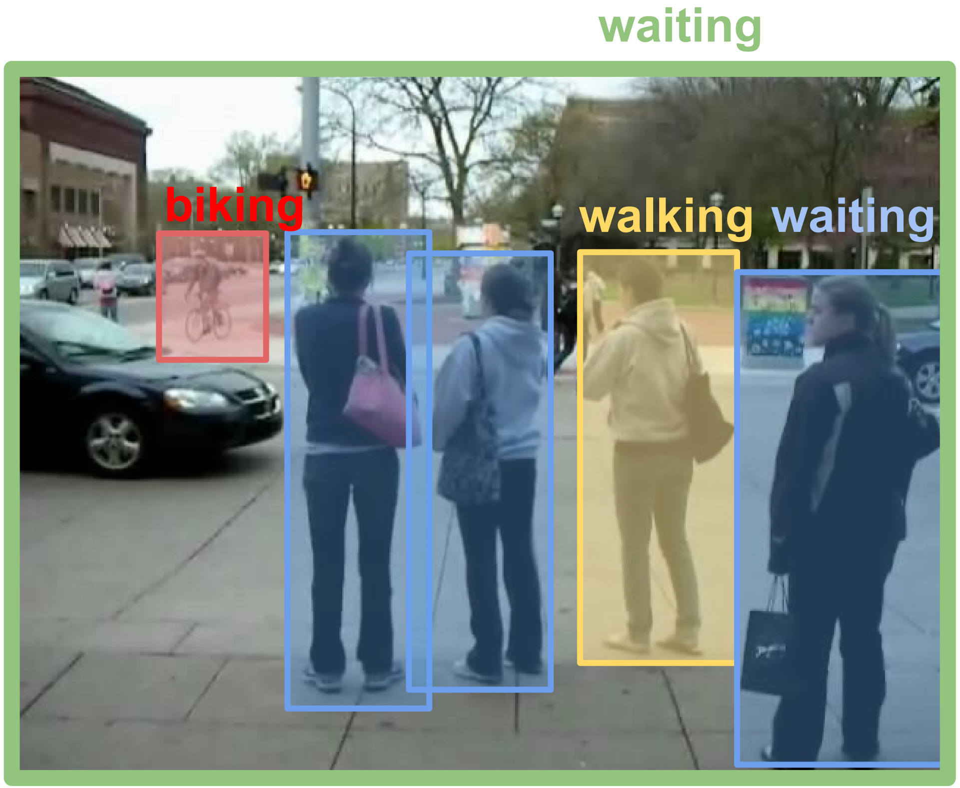
- What is each person doing?
- What is the scene as a whole doing?

Classification Problem:

- What is each person doing?
- What is the scene as a whole doing?



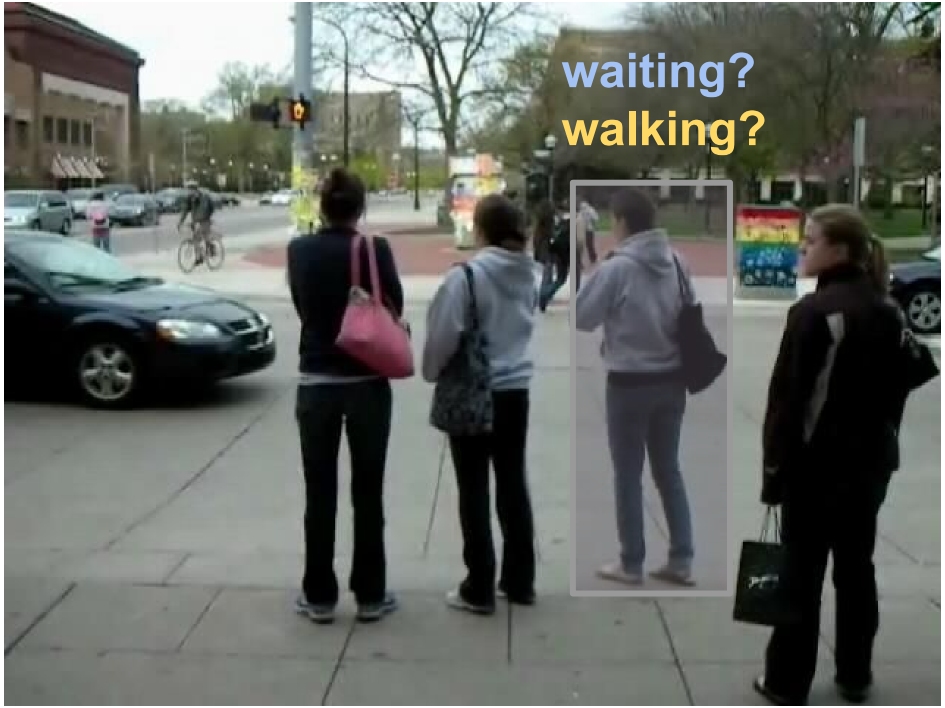
Group Activity Recognition



Classification Problem:

- What is each person doing?
  - waiting
  - walking
  - biking
- What is the scene as a whole doing?
  - waiting

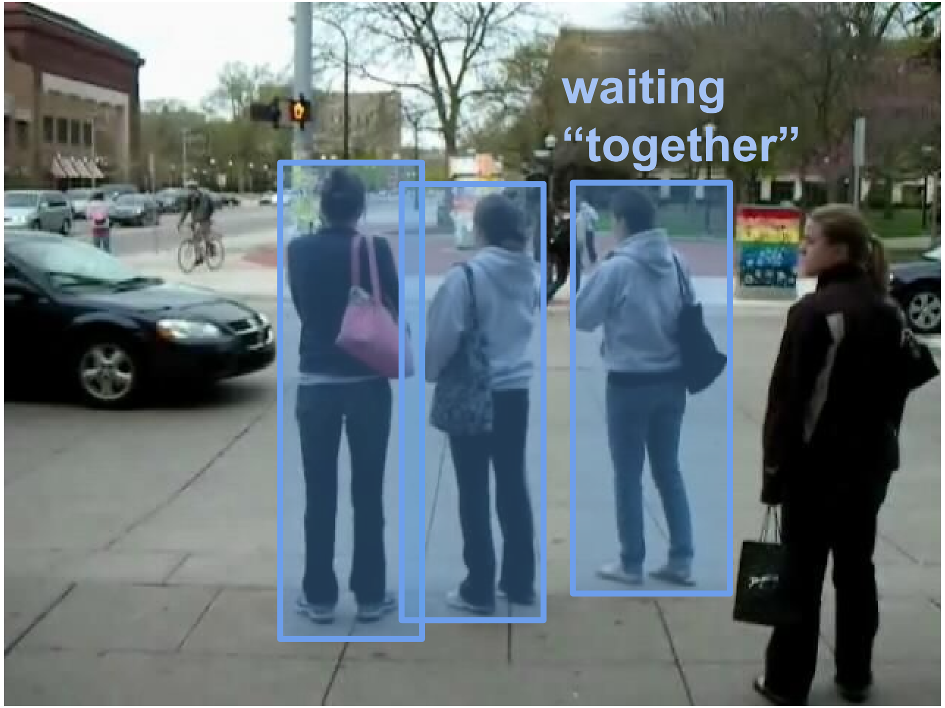
Improving Group Activity Recognition - Model Relationships



Individual actions inform other individual actions & the scene action

- relationships

Improving Group Activity Recognition - Model Relationships

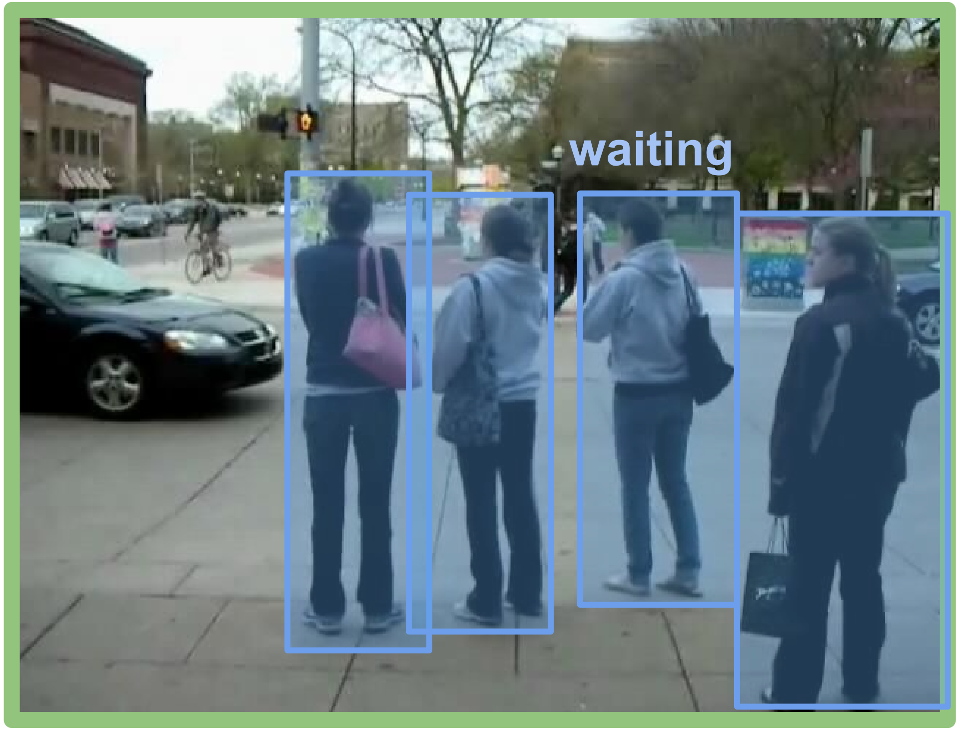


Individual actions inform other individual actions & the scene action

- relationships

Group Activity Recognition

waiting

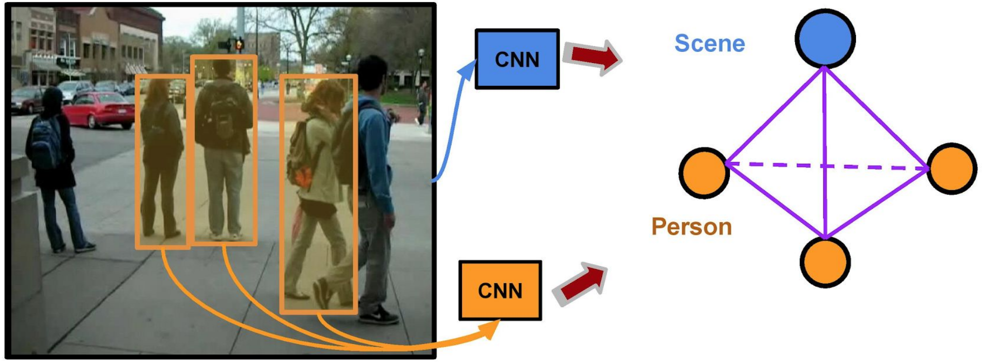


Relationships depend on

- Spatial distance
- Relative motions
- Concurrent actions
- Number of people

Model Relationships Using RNNs

Model the problem as a graphical model



RNN structure reflects learning problem

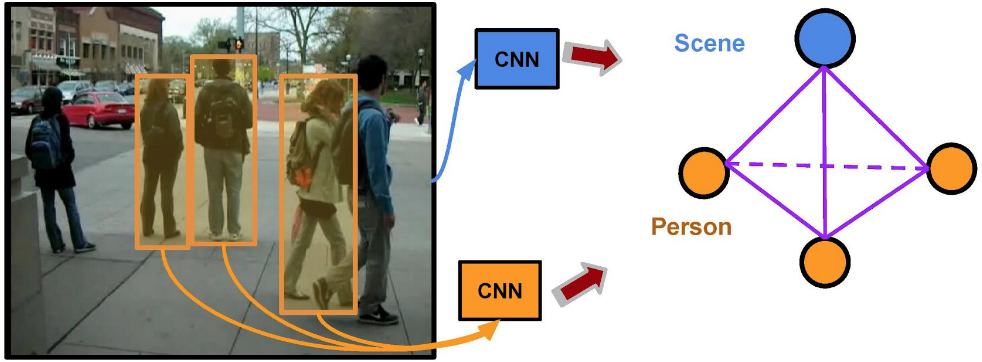
- Fully connected
- Each "edge" represents a relationship

Model learns how important each relationship is

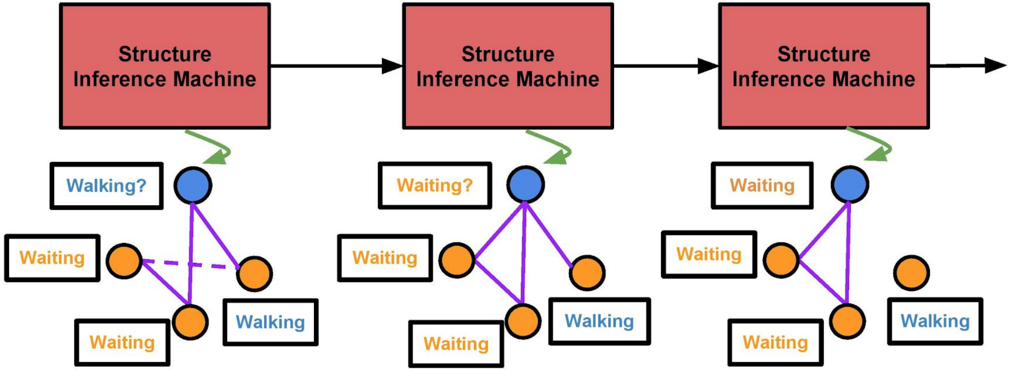
- Train edge weights

Train Using Iterative Message Passing/Gradient Descent

Model the problem as a graphical model

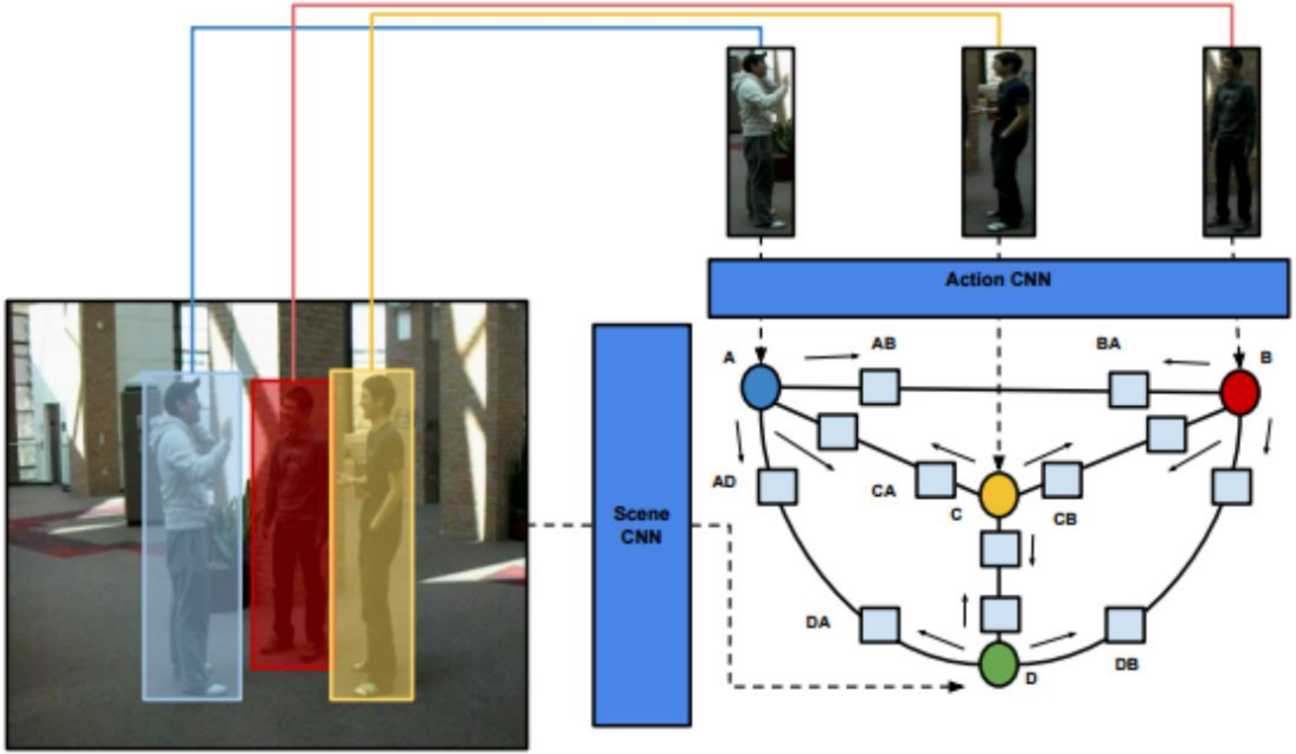


Solve the graphical model as a structure inference machine(SIM)

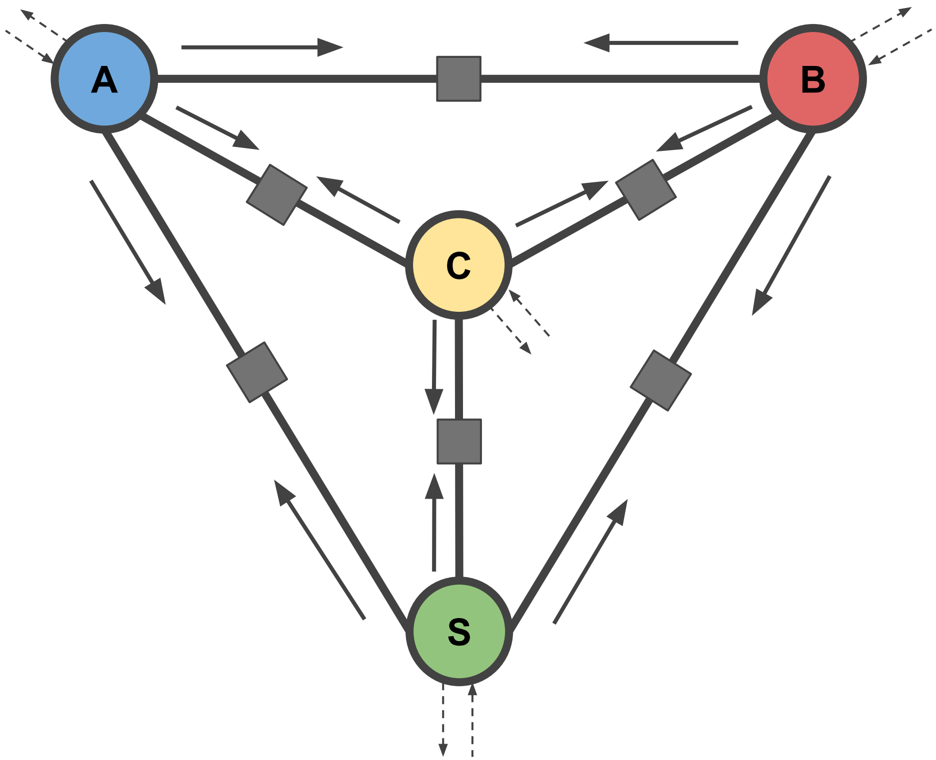


Why do we need multiple iterations of message passing?

Graphs are cyclic so exact inference isn't possible


Action CNN
Scene CNN
A
B
C
D
AB
BA
AD
CA
CB
DA
DB

Graph Model



Training the Model

For each iteration t

For each edge (i, j)

Update messages $m_{i \to j}^{(t)}$ and $m_{j \to i}^{(t)}$

Update gates $g_{i \to j}^{(t)}$ and $g_{j \to i}^{(t)}$

Impose gates on messages

For each node i

Calculate prediction $c_i^{(t)}$

Output: Final predictions at time T, $c_i^{(T)}$

General Message-Passing Update Equations

$$
\begin{aligned}
m^{(t)} &= f(W_{mm} m^{(t-1)} + W_{xm} x + W_{cm} c^{(t-1)} + b_m) \\
c^{(t)} &= f(W_{mc} m^{(t)} + W_{xc} x + b_c)
\end{aligned}
$$

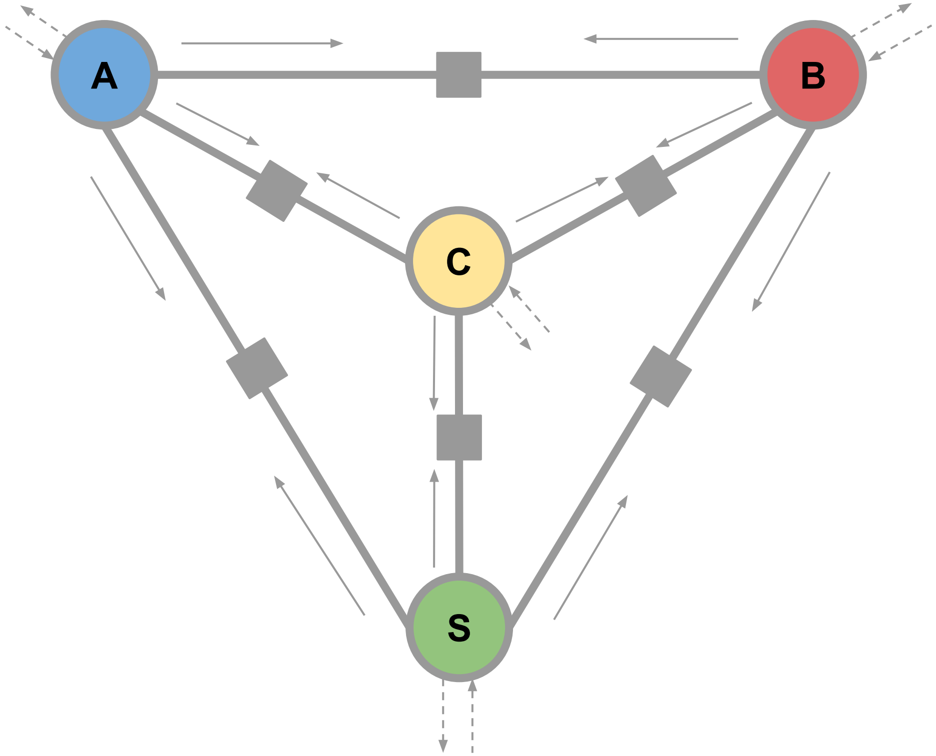
At time *t*, message $m^{(t)}$ is a function *f* of weighted sum with

- Input features *x*
- Last message $m^{(t-1)}$
- Last prediction $c^{(t-1)}$

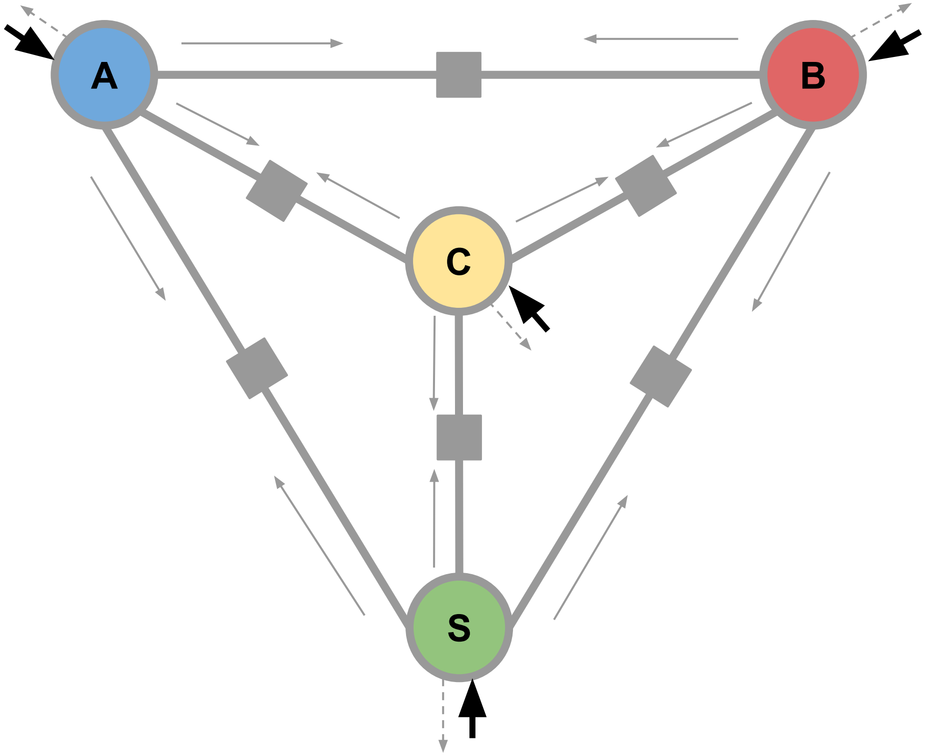
At time *t*, prediction $c^{(t)}$ is a function *f* of weighted sum with

- Input features *x*
- Current message $m^{(t)}$

RNN Model



RNN Model - Get Feature Inputs



Individual Features

$\text{CNN} \rightarrow x_A$

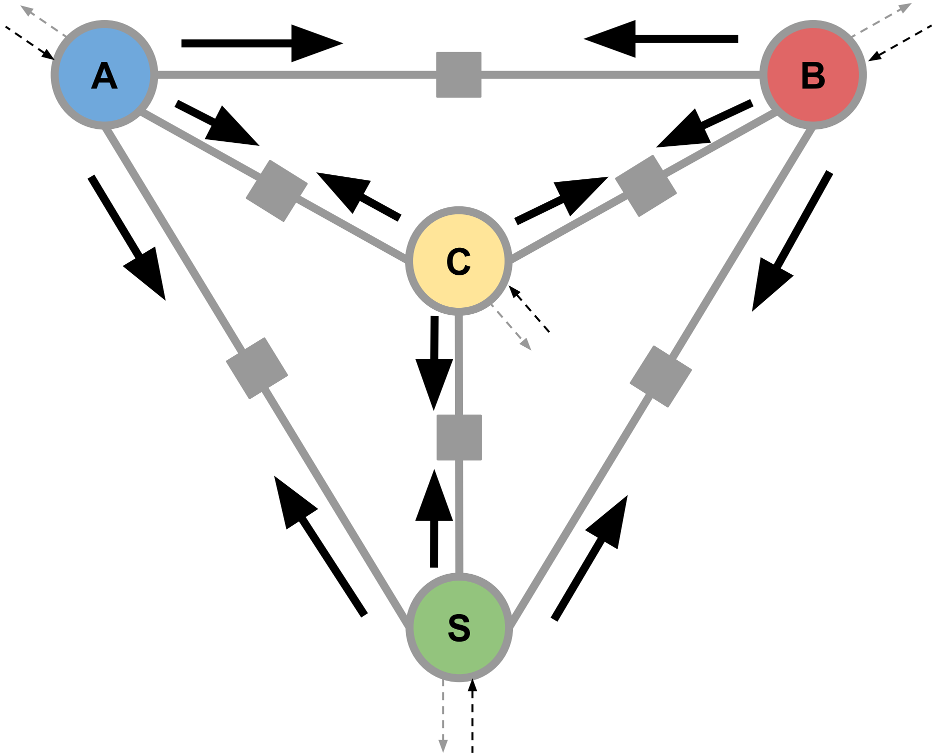
$\text{CNN} \rightarrow x_B$

$\text{CNN} \rightarrow x_C$

Scene Features

$\text{CNN} \rightarrow x_S$

RNN Model - Initialize Messages at Time *t=0*



$$m^{(0)}_{S\to A,B,C} = x_S$$

$$m^{(0)}_{A\to S,B,C} = x_A$$

$$m^{(0)}_{B\to A,S,C} = x_B$$

$$m^{(0)}_{C\to A,B,S} = x_C$$

Training the Model

For each iteration t

For each edge (i, j)

Update messages $\boldsymbol{m_{i \to j}^{(t)}}$ and $\boldsymbol{m_{j \to i}^{(t)}}$

Update gates $g_{i \to j}^{(t)}$ and $g_{j \to i}^{(t)}$

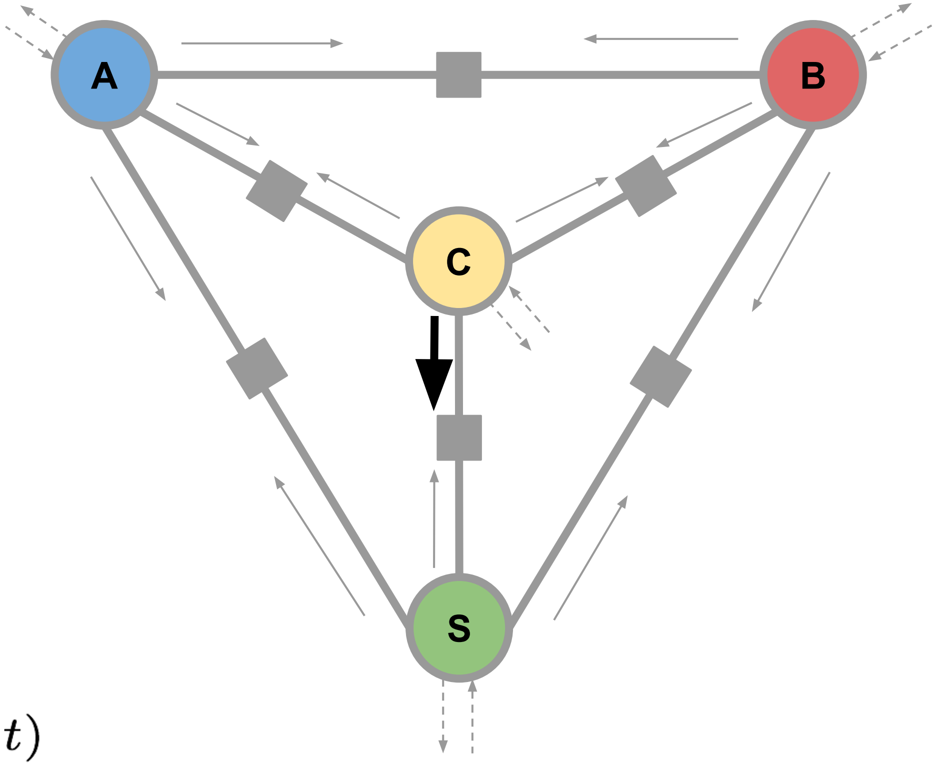
Impose gates on messages

For each node i

Calculate prediction $c_i^{(t)}$

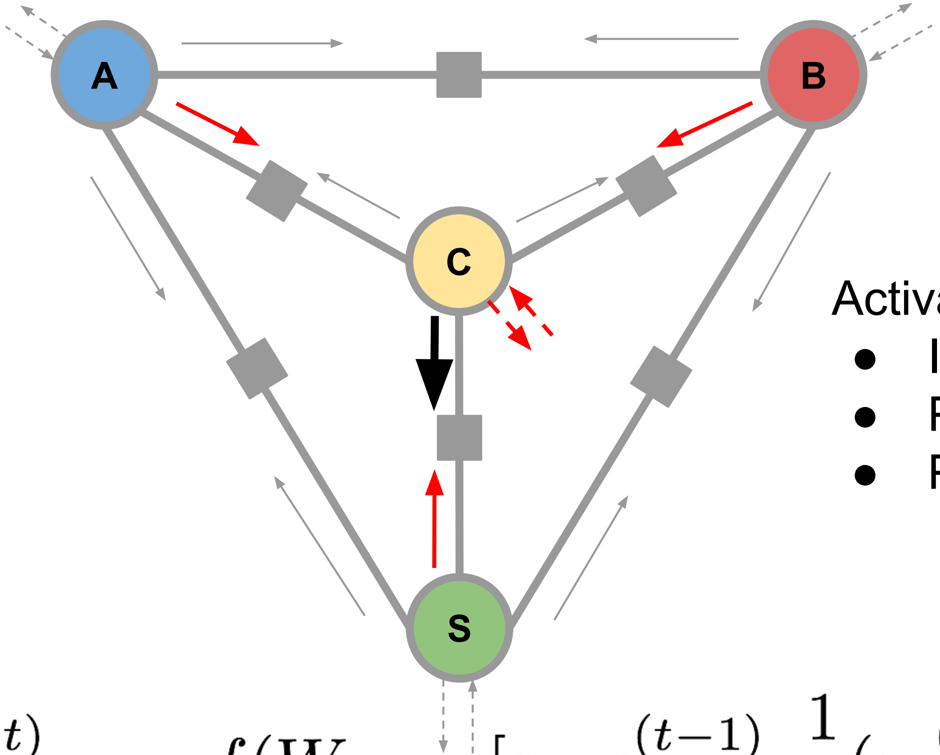
Output: Final predictions at time T, $c_i^{(T)}$

Training the Model - Updating Messages at Time t



$$m^{(t)}_{C \to S}$$

Training the Model - Updating Messages at Time *t*

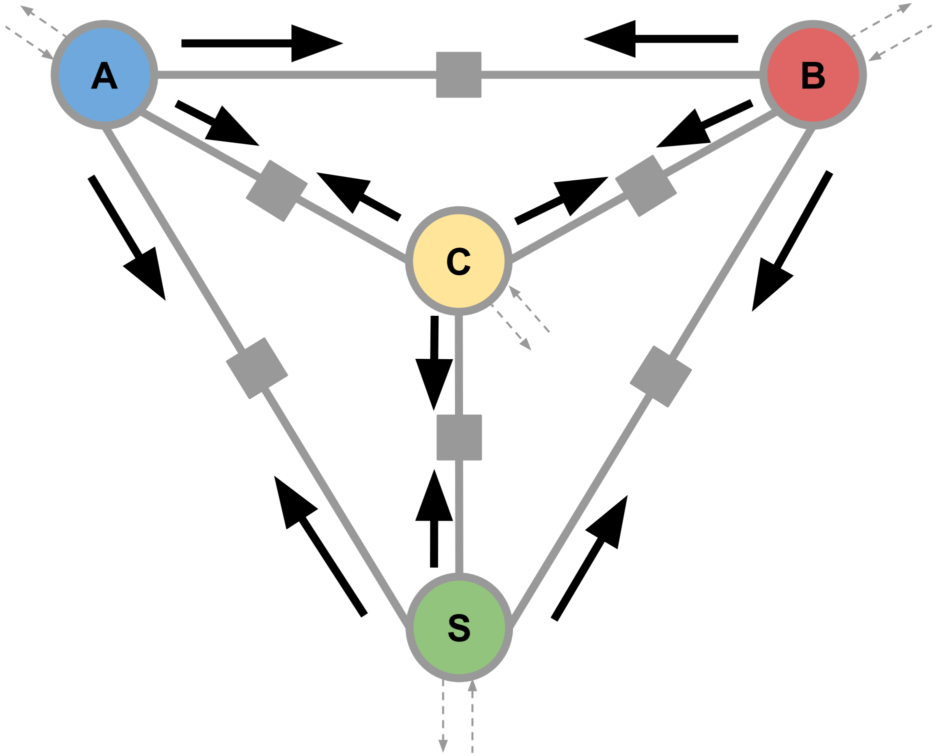


Activation function on weighted sum of

- Input features to messaging node
- Prev. prediction by messaging node
- Prev. messages to messaging node

$$m_{C\to S}^{(t)} = f(W_{mm} \cdot [x_c, c_C^{(t-1)}, \frac{1}{3}(m_{A\to C}^{(t-1)} + m_{B\to C}^{(t-1)} + m_{S\to C}^{(t-1)})])$$

Training the Model - Updating Messages at Time *t*



Training the Model

For each iteration t

For each edge (i, j)

Update messages $m_{i \to j}^{(t)}$ and $m_{j \to i}^{(t)}$

Update gates $g_{i \to j}^{(t)}$ and $g_{j \to i}^{(t)}$

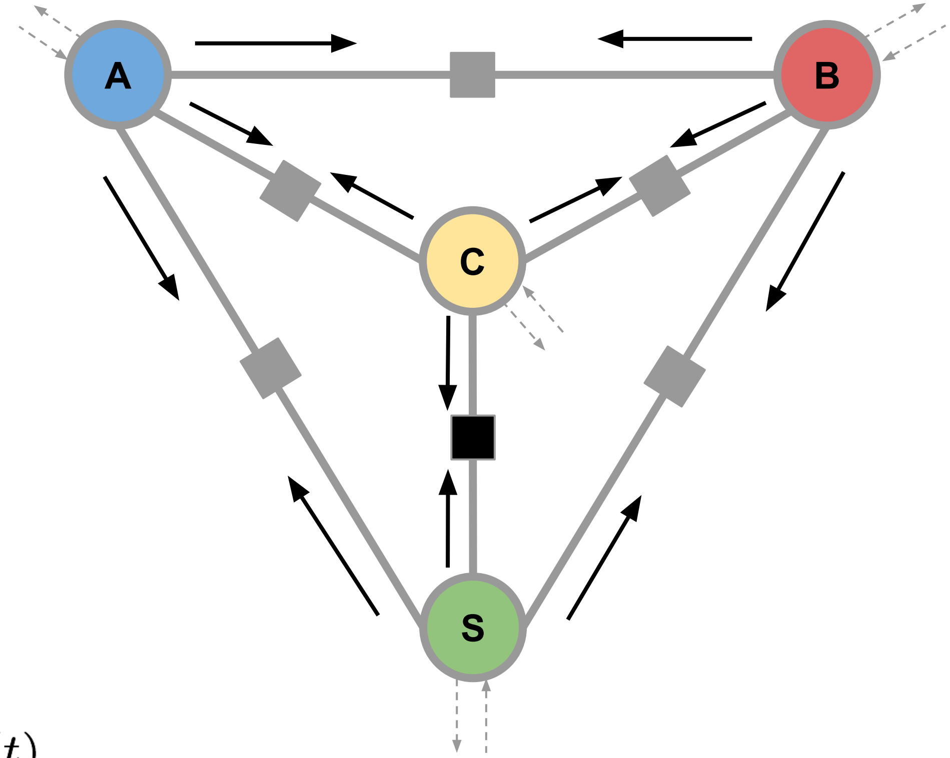
Impose gates on messages

For each node i

Calculate prediction $c_i^{(t)}$

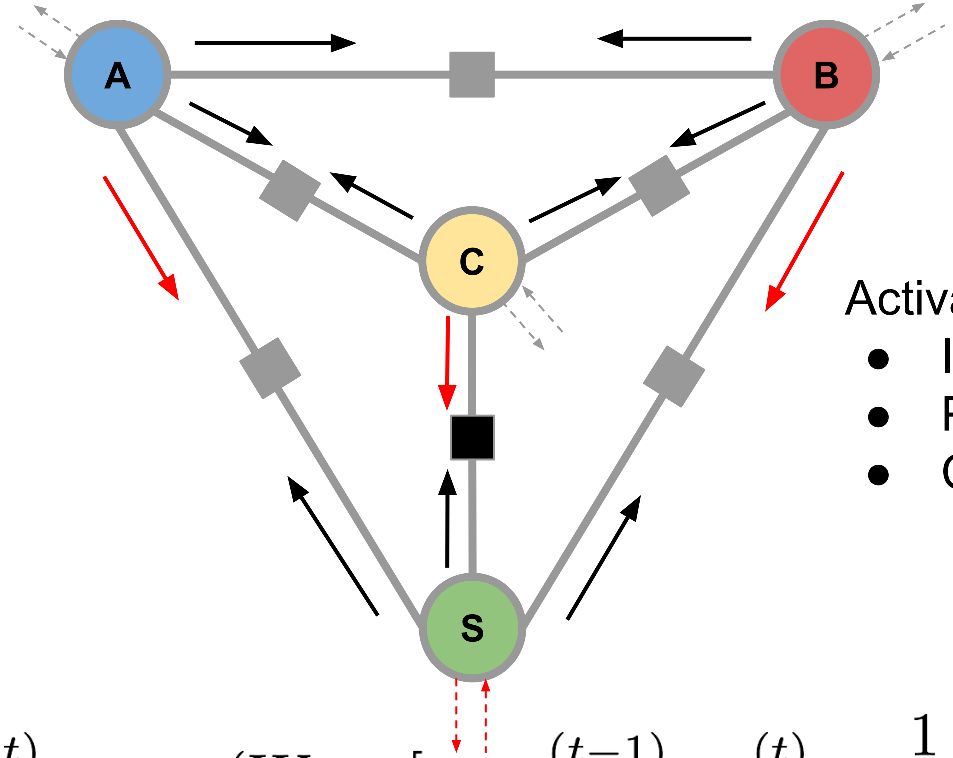
Output: Final predictions at time T, $c_i^{(T)}$

Training the Model - Updating Gates at Time t



$$g^{(t)}_{C \to S}$$

Training the Model - Updating Gates at Time *t*

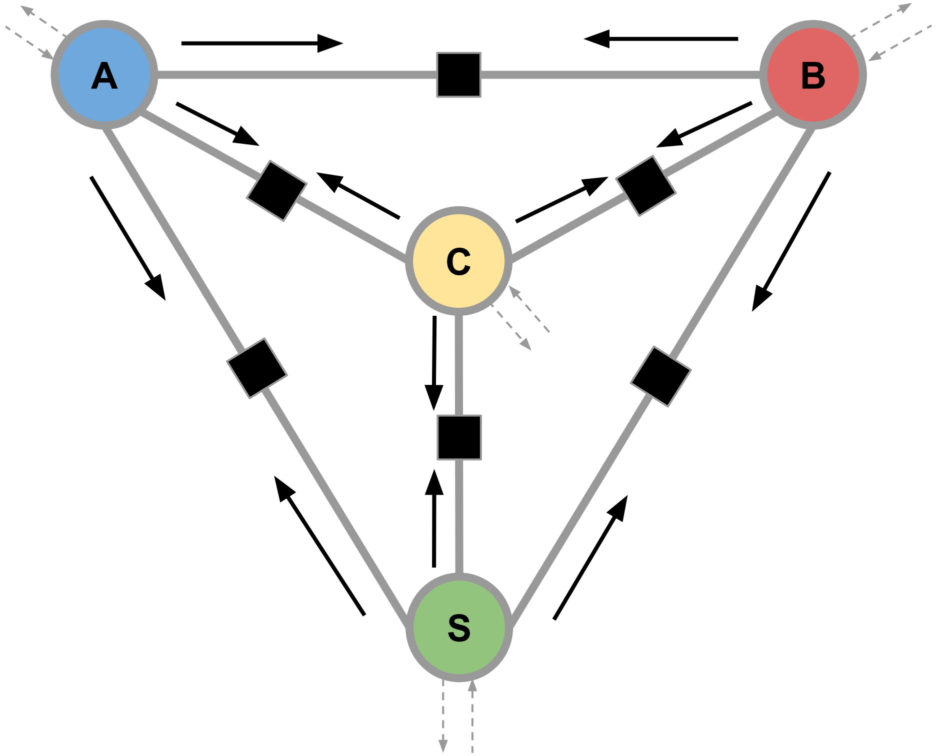


Activation function on weighted sum of

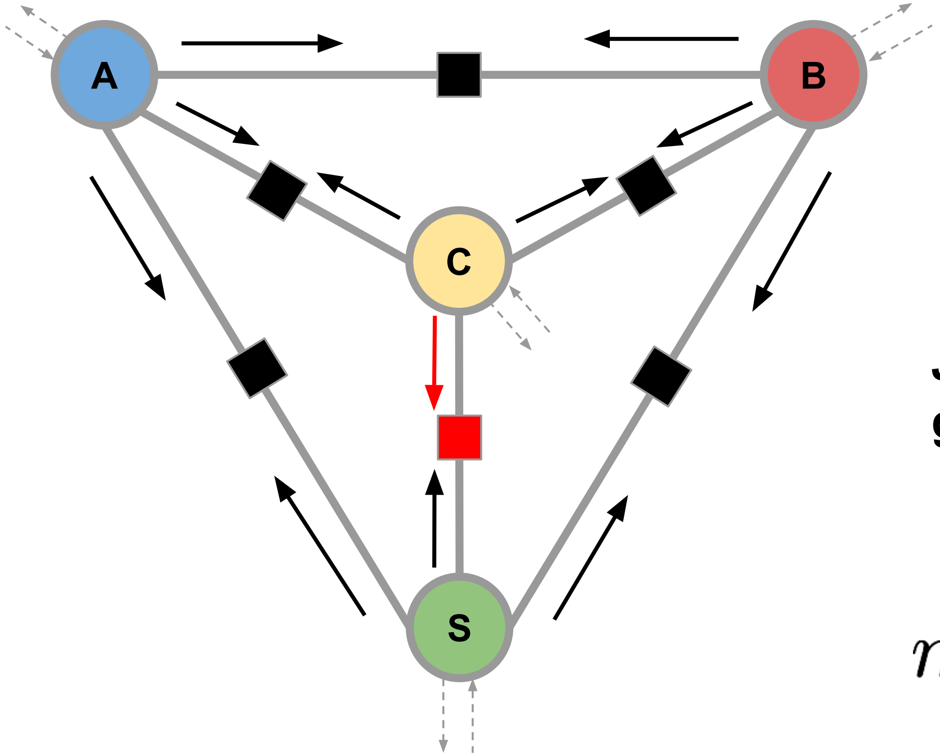
- Input features to target node
- Prev. prediction by target node
- Current messages to target node

$$g_{C\to S}^{(t)} = \sigma(W_{hg} \cdot [x_S, c_S^{(t-1)}, m_{C\to S}^{(t)}, \frac{1}{2}(m_{A\to S}^{(t)} + m_{B\to S}^{(t)})])$$

Training the Model - Updating Gates at Time t



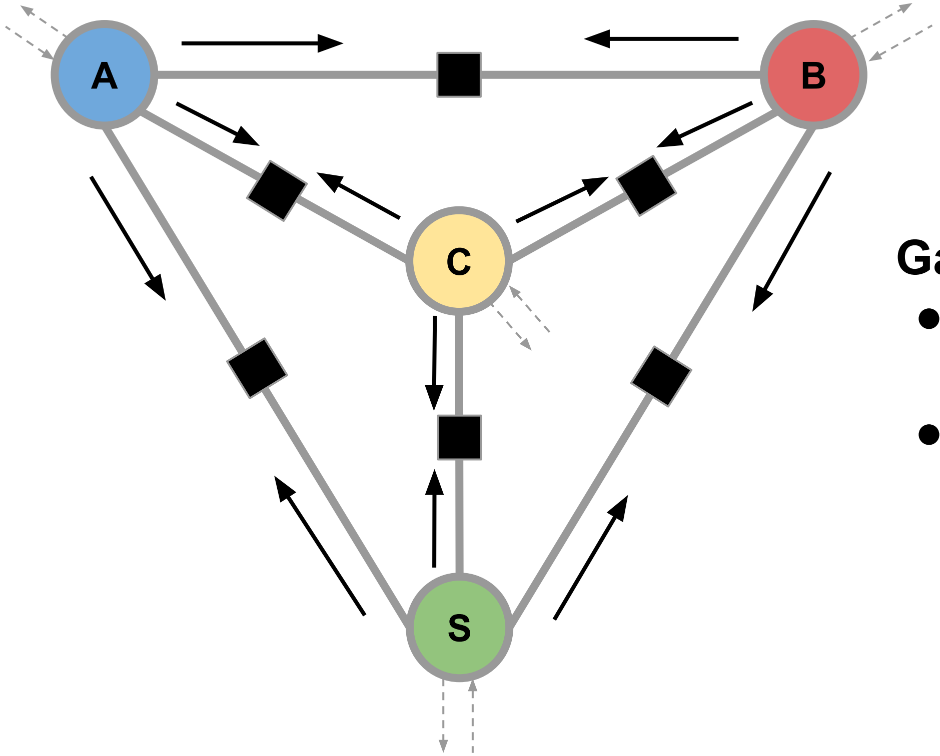
Training the Model - Imposing Gates at Time *t*



Just multiply message vectors by gate scalars

$$m'^{(t)}_{i\to j} = g^{(t)}_{i\to j} \odot m^{(t)}_{i\to j}$$

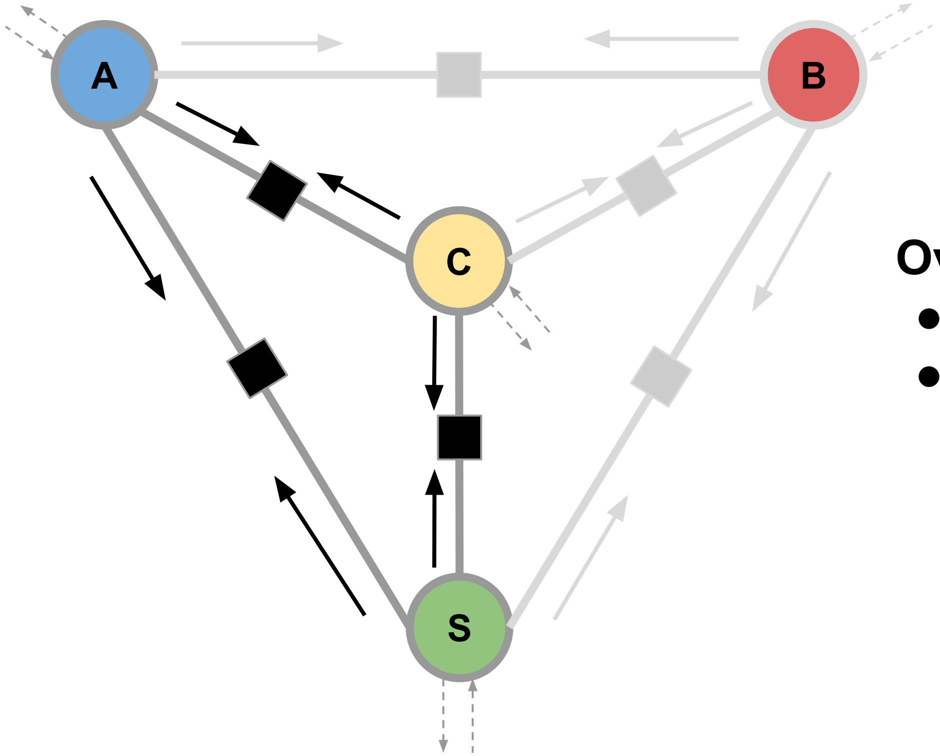
Training the Model - Updating Gates at Time *t*



Gate Intuition:

- Determines whether edge is "useful"
- Modulates importance of edge

Training the Model - Imposing Gates



Over Several Iterations…

- Learns to ignore some dges
- And focus more on others

Training the Model

For each iteration t

For each edge (i, j)

Update messages $m_{i \to j}^{(t)}$ and $m_{j \to i}^{(t)}$

Update gates $g_{i \to j}^{(t)}$ and $g_{j \to i}^{(t)}$

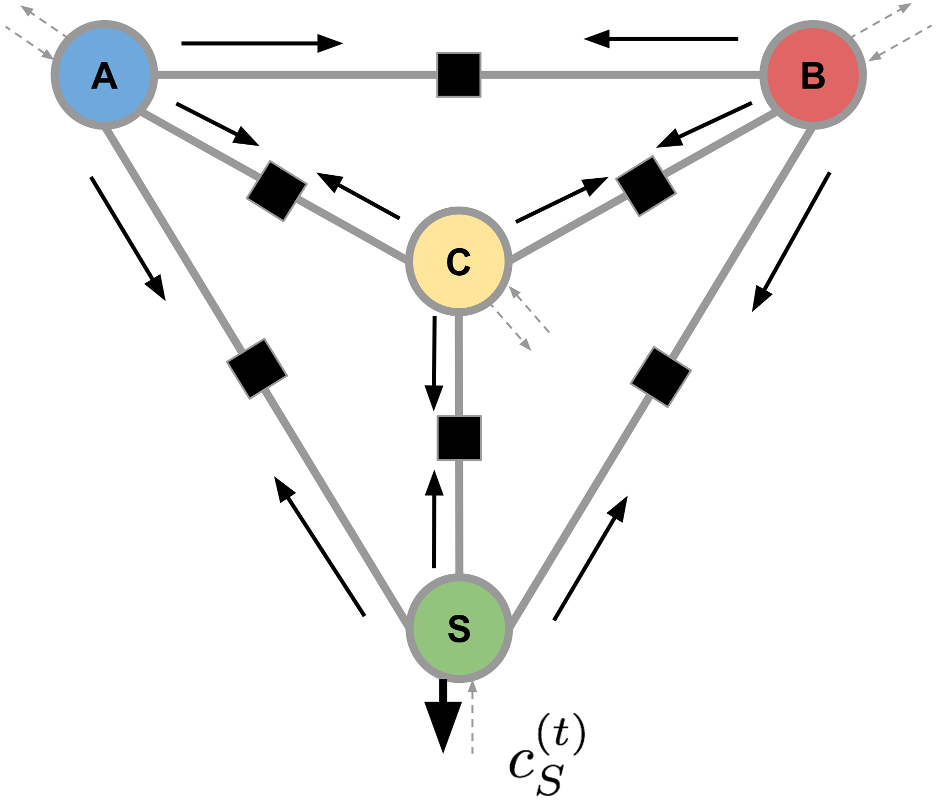
Impose gates on messages

For each node i

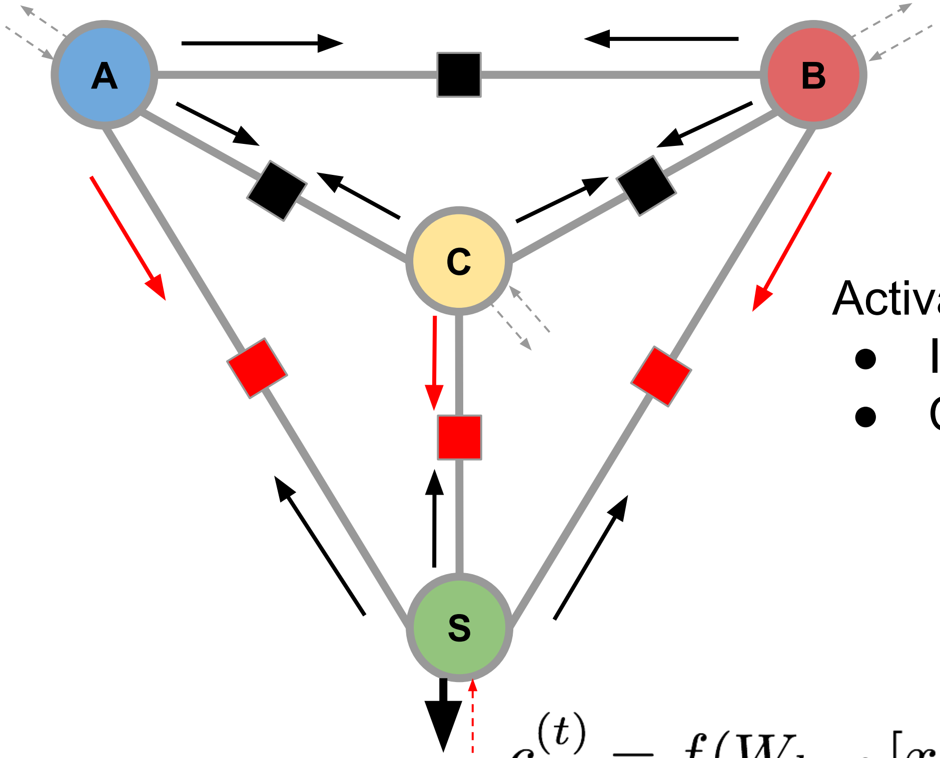
Calculate prediction $c_i^{(t)}$

Output: Final predictions at time T, $c_i^{(T)}$

Training the Model - Getting Predictions $c_i^{(t)}$ at Time t



Training the Model - Getting Predictions $c_i^{(t)}$ at Time t

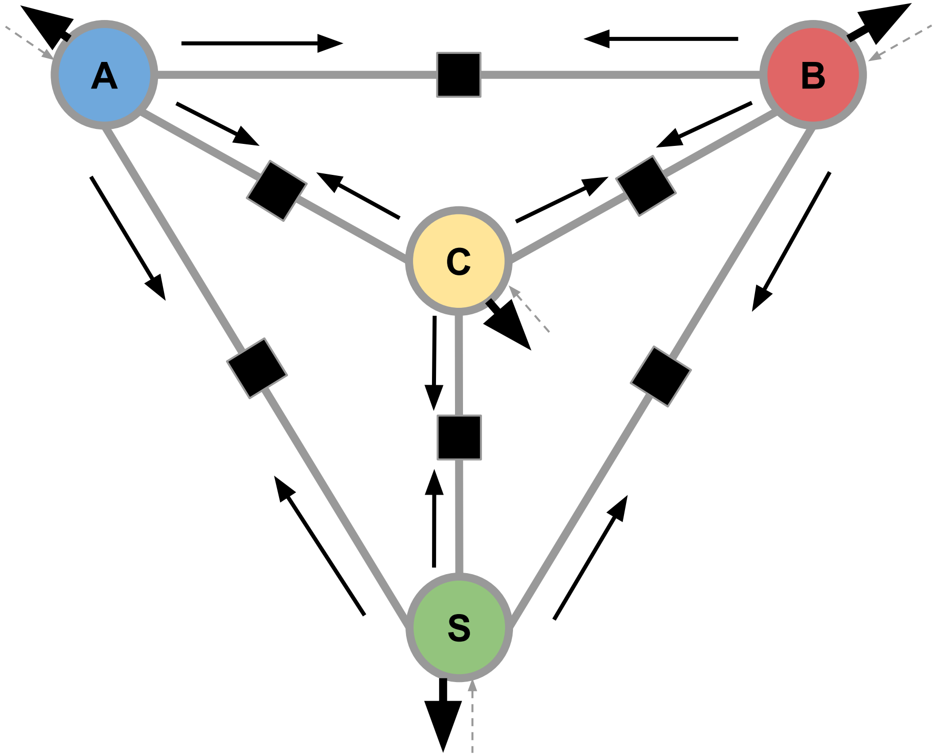


Activation function on weighted sum of

- Input features to node
- Current messages to node

$$c_S^{(t)} = f(W_{hc} \cdot [x_S, \frac{1}{3}(m'^{(t)}_{A\to S} + m'^{(t)}_{B\to S} + m'^{(t)}_{C\to S})])$$

Training the Model - Getting Predictions $c_i^{(t)}$ at Time t



Training the Model

For each iteration t

For each edge (i, j)

Update messages $m_{i \to j}^{(t)}$ and $m_{j \to i}^{(t)}$

Update gates $g_{i \to j}^{(t)}$ and $g_{j \to i}^{(t)}$

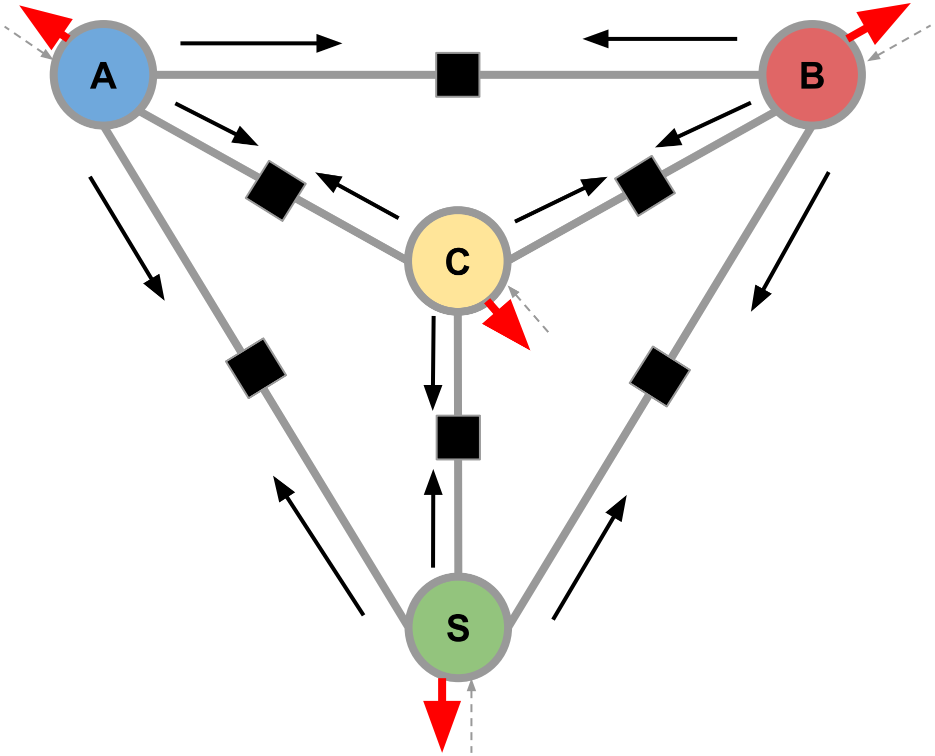
Impose gates on messages

For each node i

Calculate prediction $c_i^{(t)}$

Output: Final predictions at time T, $c_i^{(T)}$

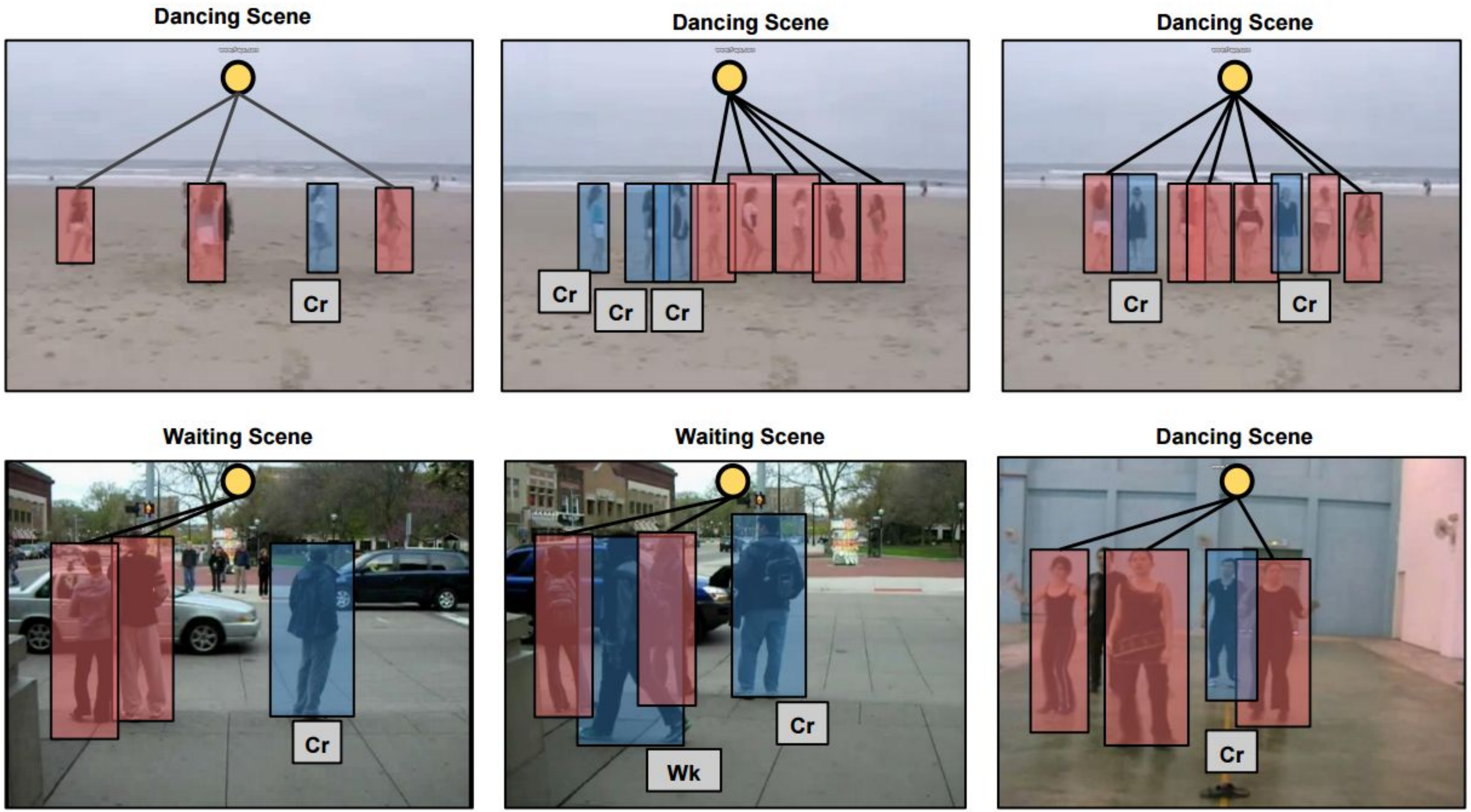
Output - Getting Final Predictions $c_i^{(T)}$ at Time T



Applied to Collective Activity Dataset

- 44 videos
- 7 actions
    - Crossing
    - Waiting
    - Queueing
    - Talking
    - Jogging
    - Dancing
    - N/A

Visualization of Results on Collective Activity Dataset



Results on Collective Activity Dataset

Method	Accuracy
Learning Latent Constituent [4]	75.1%
Latent SVM with Optimized Graph [28]	79.7%
Deep Struct. Model [13]	80.6%
Unified Tracking And Recognition[9]	80.6%
Cardinality Kernel [17]	**83.4%**
Our Model	81.2%

Key Takeaways

- Previous outputs and hidden states can be looped back in as inputs
    - Model problems where inputs are highly correlated

thanks