6.874, 6.802, 20.390, 20.490, HST.506
Computational Systems Biology
Deep Learning in the Life Sciences

Lecture 4:
Recurrent Neural Networks
LSTMs, Transformers,
Graph Neural Networks

Prof. Manolis Kellis
Guest lecture: Neil Band

Massachusetts Institute of Technology

http://mit6874.github.io

Slides credit: Geoffrey Hinton, Ian Goodfellow, David Gifford, 6.S191 (Ava Soleimany, Alex Amini)

Recurrent Neural Networks (RNNs) + Generalization

1. **How do you read/listen/understand/write? Can machines do that?**
   - Context matters: characters, words, letters, sounds, completion, multi-modal
   - Predicting next word/image: from unsupervised learning to supervised learning
2. **Encoding temporal context: Hidden Markov Models (HMMs), RNNs**
   - Primitives: hidden state, memory of previous experiences, limitations of HMMs
   - RNN architectures,unrolling,back-propagation-through-time(BPTT),param reuse
3. **Vanishing gradients, Long-Short-Term Memory (LSTM), initialization**
   - Key idea: gated input/output/memory nodes, model choose to forget/remember
   - Example: online character recognition with LSTM recurrent neural network
4. **Transformer modules**
   - Learning temporal relationships without unrolling and without RNNs
   - Encoder/Decoder output architecture and multi-head attention modeule
5. **Graph Neural Networks**
   - Applications: social, brain, chemical drug design, graphics, transport, knowledge
   - Define each node's computation graph, from its neighborhood
   - Classical network/graph problems: Node/graph classification, link prediction
   - Research frontiers: deep generative models, latent graph inferences

1a. What do you hear and why?

Context matters

Top-down processing

THE CAT

Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

7H15 M3554G3 53RV35 7O PR0V3 H0W 0UR M1ND5 C4N D0 4M4Z1NG 7H1NG5! 1MPR3551V3 7H1NG5! 1N 7H3 B3G1NN1NG 17 WA5 H4RD BU7 N0W, 0N 7H15 LIN3 Y0UR M1ND 1S R34D1NG 17 4U70M471C4LLY W17H 0U7 3V3N 7H1NK1NG 4B0U7 17, B3 PROUD! 0NLY C3R741N P30PL3 C4N R3AD 7H15. PL3453 F0RW4RD 1F U C4N R34D 7H15.

Phonemic restoration

[FREN] butters
@NoisyButters

do yall pronounce it data or data?

data 70.3%

data 29.7%

6,504 votes · Final results

7:54 PM · Sep 14, 2018 · Twitter for iPhone

56 Retweets **485** Likes

Hearing lips and seeing voices (McGurk, MacDonald, Nature 1976)

https://youtu.be/PWGeUztTkRA?t=35

Split class into 4 groups: (1) close your eyes, (2) look left, (3) middle, (4) right

Delayed Auditory Feedback Stuttering -DAF

Chinjja Music & Audio

Everyone

Contains Ads · Offers in-app purchases

This app is compatible with your device.

Adults: 200 ms delay max disruption.
Children: 500 ms

Delayed typing: Google Docs, zoom video screen sharing, slow computer

"When we listen to someone talking, the change in our brain's processing from not caring what kind of sound it is to recognizing it as a word happens surprisingly early," said Simon. "In fact, this happens pretty much as soon as the linguistic information becomes available."

When it is engaging in speech perception, the brain's auditory cortex analyzes complex acoustic patterns to detect words that carry a linguistic message. It seems to do this so efficiently, at least in part, by anticipating what it is likely to hear: by learning what sounds signal language most frequently, the brain can predict what may come next. It is generally thought that this process -- localized bilaterally in the brain's superior temporal lobes -- involves recognizing an intermediate, phonetic level of sound.

https://www.sciencedaily.com/releases/2018/11/181129142352.htm

Recurrent Neural Networks (RNNs) + Generalization

1. **How do you read/listen/understand/write? Can machines do that?**
   - Context matters: characters, words, letters, sounds, completion, multi-modal
   - Predicting next word/image: from unsupervised learning to supervised learning
2. **Encoding temporal context: Hidden Markov Models (HMMs), RNNs**
   - Primitives: hidden state, memory of previous experiences, limitations of HMMs
   - RNN architectures,unrolling,back-propagation-through-time(BPTT),param reuse
3. **Vanishing gradients, Long-Short-Term Memory (LSTM), initialization**
   - Key idea: gated input/output/memory nodes, model choose to forget/remember
   - Example: online character recognition with LSTM recurrent neural network
4. **Transformer modules**
   - Learning temporal relationships without unrolling and without RNNs
   - Encoder/Decoder output architecture and multi-head attention modeule
5. **Graph Neural Networks**
   - Applications: social, brain, chemical drug design, graphics, transport, knowledge
   - Define each node's computation graph, from its neighborhood
   - Classical network/graph problems: Node/graph classification, link prediction
   - Research frontiers: deep generative models, latent graph inferences

2a. Encoding time

Getting targets when modeling sequences

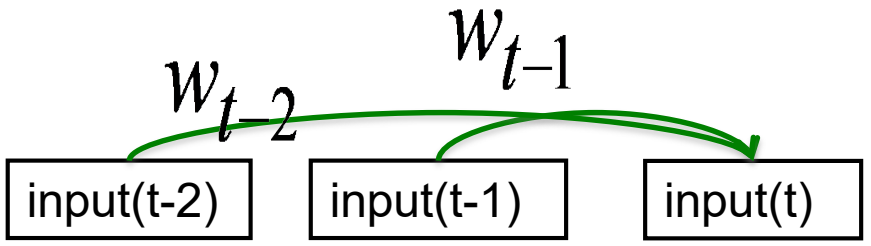
- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.

- When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.
  - The target output sequence is the input sequence with an advance of 1 step.
  - This seems much more natural than trying to predict one pixel in an image from the other pixels, or one patch of an image from the rest of the image.
  - For temporal sequences there is a natural order for the predictions.

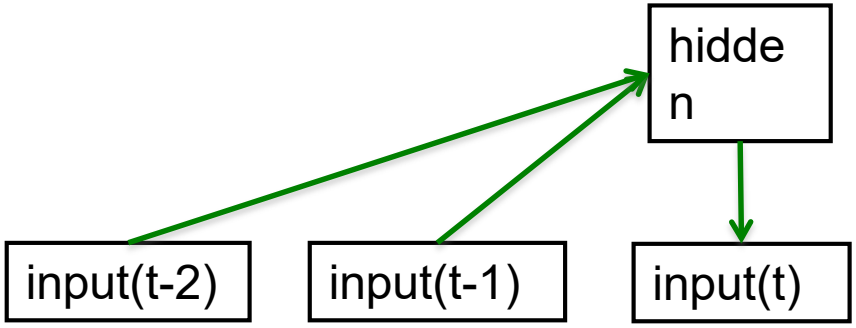
- Predicting the next term in a sequence blurs the distinction between supervised and unsupervised learning.
  - It uses methods designed for supervised learning, but it doesn't require a separate teaching signal.

Memoryless models for sequences

- Autoregressive models
  Predict the next term in a sequence from a fixed number of previous terms using “delay taps”.



- Feed-forward neural nets
  These generalize autoregressive models by using one or more layers of non-linear hidden units.

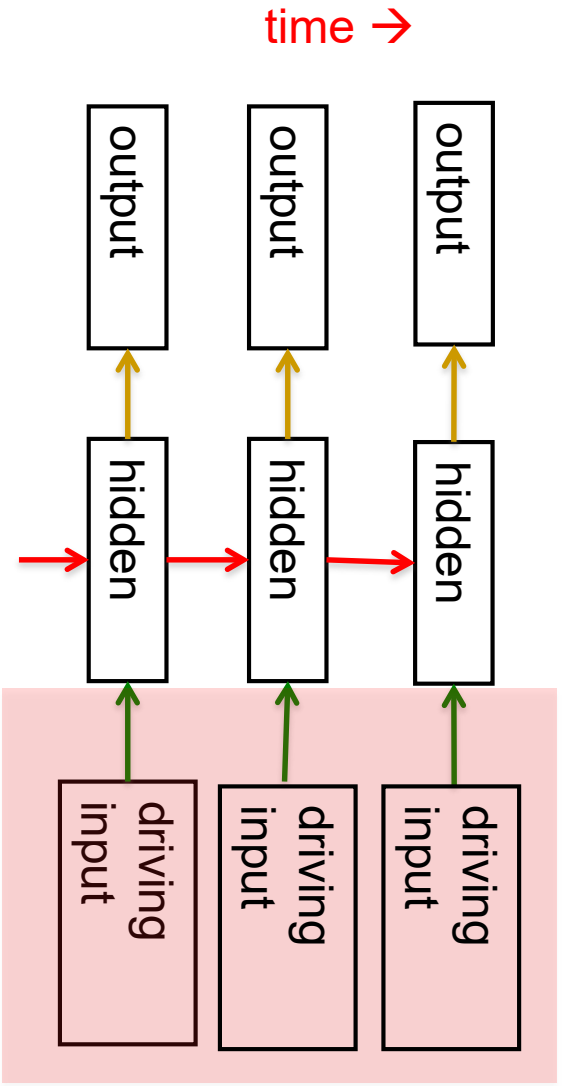


Beyond memoryless models

- If we give our generative model some hidden state, and if we give this hidden state its own internal dynamics, we get a much more interesting kind of model.
  - It can store information in its hidden state for a long time.
  - If the dynamics is noisy and the way it generates outputs from its hidden state is noisy, we can never know its exact hidden state.
  - The best we can do is to infer a probability distribution over the space of hidden state vectors.
- This inference is only tractable for two types of hidden state model.

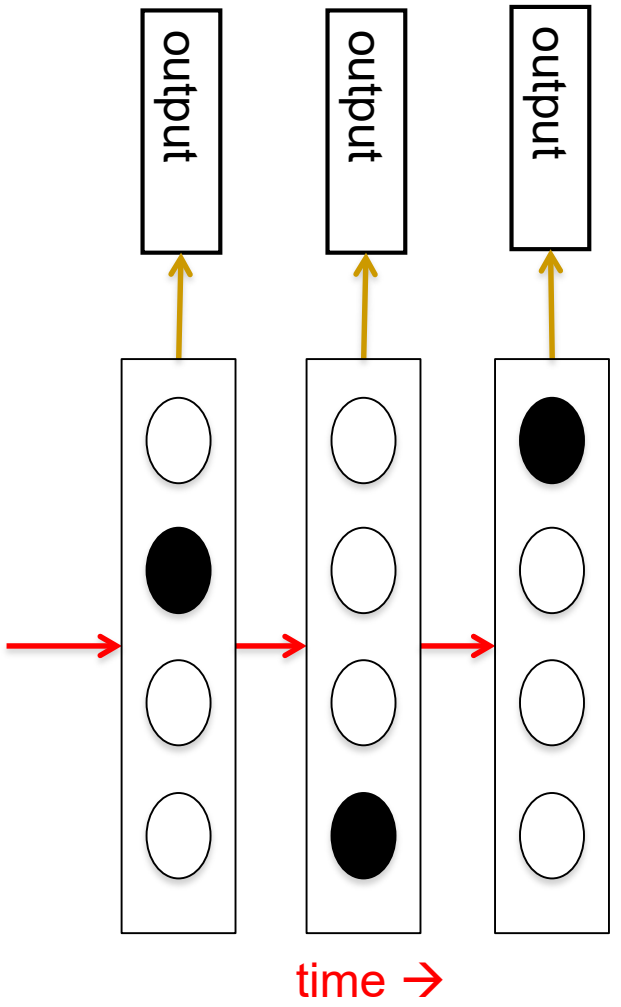
Linear Dynamical Systems (engineers love them!)

- These are generative models. They have a real-valued hidden state that cannot be observed directly.
  - The hidden state has linear dynamics with Gaussian noise and produces the observations using a linear model with Gaussian noise.
  - There may also be driving inputs.
- To predict the next output (so that we can shoot down the missile) we need to infer the hidden state.
  - A linearly transformed Gaussian is a Gaussian. So the distribution over the hidden state given the data so far is Gaussian. It can be computed using "Kalman filtering".



Hidden Markov Models (computer scientists love them!)

- Hidden Markov Models have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.
  - We cannot be sure which state produced a given output. So the state is "hidden".
  - It is easy to represent a probability distribution across N states with N numbers.
- To predict the next output we need to infer the probability distribution over hidden states.
  - HMMs have efficient algorithms for inference and learning.



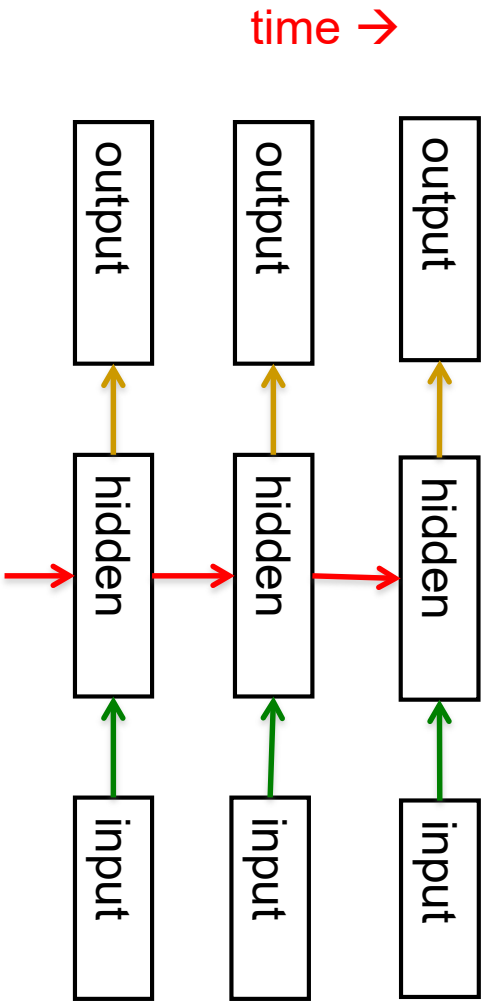
A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data.
  - At each time step it must select one of its hidden states. So with N hidden states it can only remember log(N) bits about what it generated so far.
- Consider the information that the first half of an utterance contains about the second half:
  - The syntax needs to fit (e.g. number and tense agreement).
  - The semantics needs to fit. The intonation needs to fit.
  - The accent, rate, volume, and vocal tract characteristics must all fit.
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half. 2^100 is big!

2b. Recurrent Neural Networks (RNNs)

Recurrent neural networks

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.



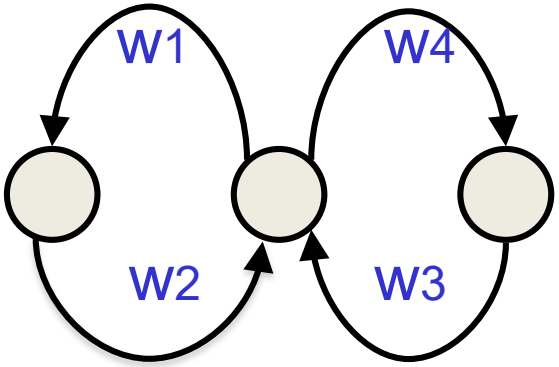
Do generative models need to be stochastic?

- Linear dynamical systems and hidden Markov models are stochastic models.
  - But the posterior probability distribution over their hidden states given the observed data so far is a deterministic function of the data.
- Recurrent neural networks are deterministic.
  - So think of the hidden state of an RNN as the equivalent of the deterministic probability distribution over hidden states in a linear dynamical system or hidden Markov model.

Recurrent neural networks

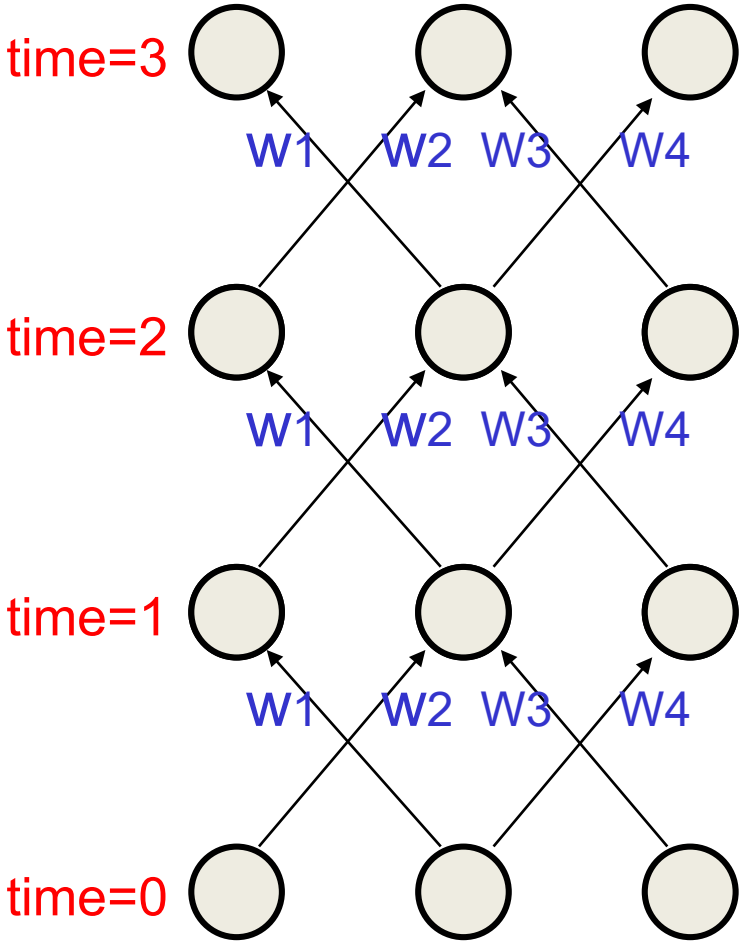
- What kinds of behaviour can RNNs exhibit?
  - They can oscillate. Good for motor control?
  - They can settle to point attractors. Good for retrieving memories?
  - They can behave chaotically. Bad for information processing?
  - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects.
- But the computational power of RNNs makes them very hard to train.
  - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer).

The equivalence between feedforward nets and recurrent nets



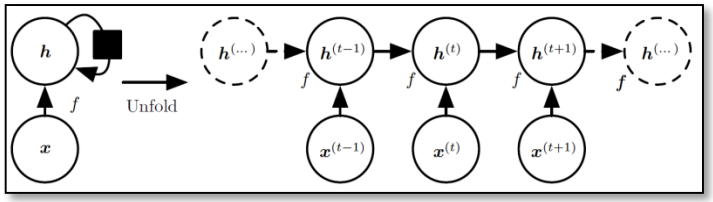
Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.

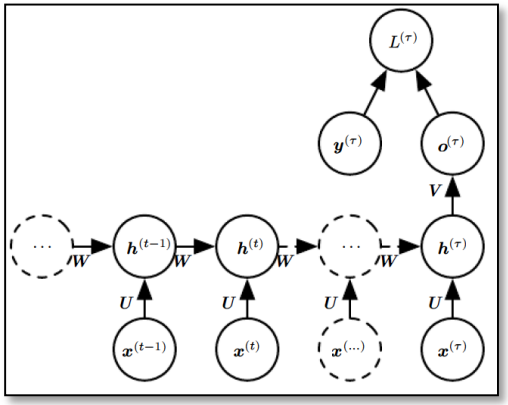


2c. Alternative architectures for RNNs

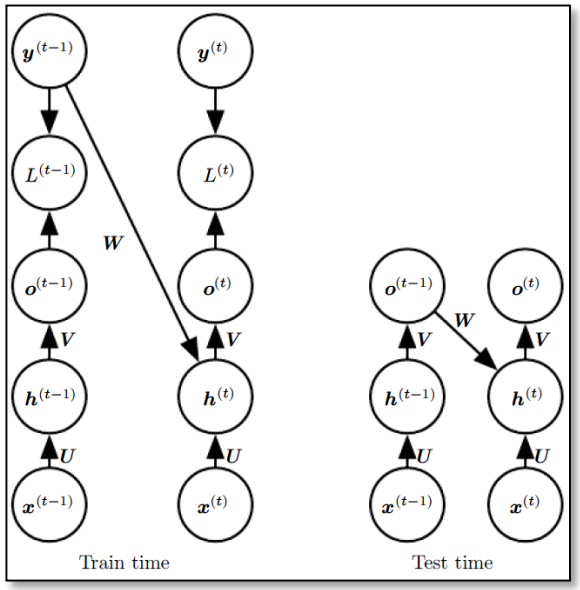
Different RNN remembering architectures



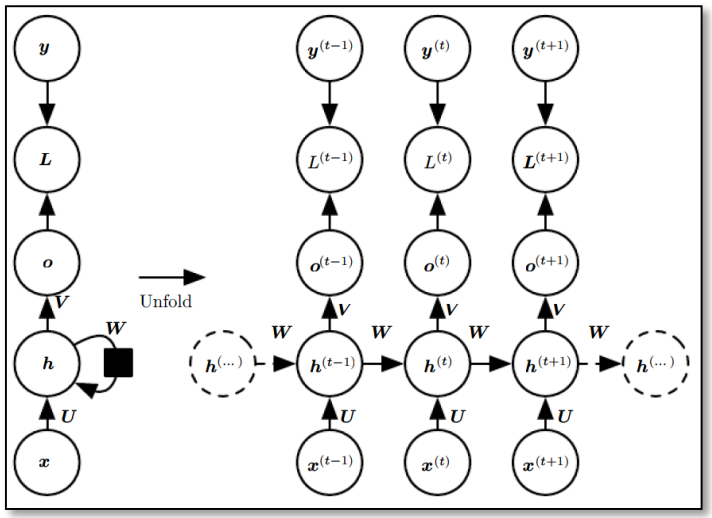
Recurrent network with no outputs



Single output after entire sequence

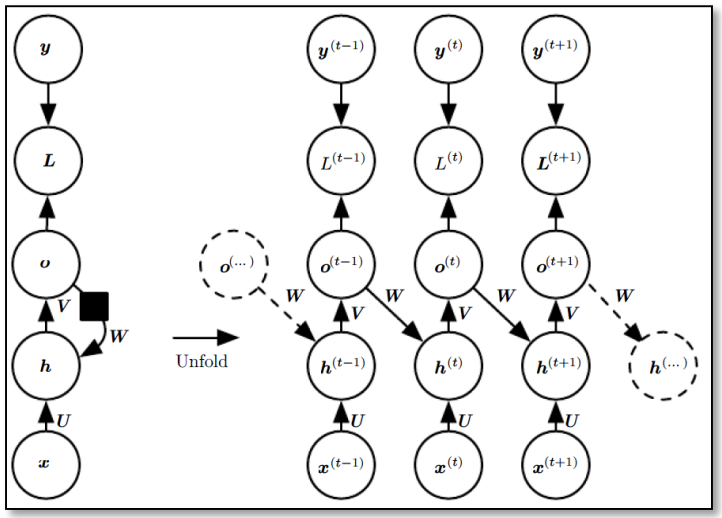


Teacher-forcing: train from **y** and **x** in parallel



o: output, y: target, L: loss

Memory: $h^{(t-1)} \rightarrow h^{(t)}$



o: output, y: target, L: loss

Memory: $o^{(t-1)} \rightarrow h^{(t)}$. Only train sequentially

2d. Back-propagation through time (BPTT)

Reminder: Backpropagation with weight constraints

- It is easy to modify the backprop algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

$$To\ \ constrain: \quad w_1 = w_2$$

$$we\ \ need: \quad \Delta w_1 = \Delta w_2$$

$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

Backpropagation through time

- We can think of the recurrent net as a layered, feed-forward net with shared weights and then train the feed-forward net with weight constraints.
- We can also think of this training algorithm in the time domain:
  - The forward pass builds up a stack of the activities of all the units at each time step.
  - The backward pass peels activities off the stack to compute the error derivatives at each time step.
  - After the backward pass we add together the derivatives at all the different times for each weight.

Getting targets when modeling sequences

- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.

- When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.
  - The target output sequence is the input sequence with an advance of 1 step.
  - This seems much more natural than trying to predict one pixel in an image from the other pixels, or one patch of an image from the rest of the image.
  - For temporal sequences there is a natural order for the predictions.

- Predicting the next term in a sequence blurs the distinction between supervised and unsupervised learning.
  - It uses methods designed for supervised learning, but it doesn't require a separate teaching signal.

Recurrent Neural Networks (RNNs) + Generalization

1. **How do you read/listen/understand/write? Can machines do that?**
   - Context matters: characters, words, letters, sounds, completion, multi-modal
   - Predicting next word/image: from unsupervised learning to supervised learning
2. **Encoding temporal context: Hidden Markov Models (HMMs), RNNs**
   - Primitives: hidden state, memory of previous experiences, limitations of HMMs
   - RNN architectures,unrolling,back-propagation-through-time(BPTT),param reuse
3. **Vanishing gradients, Long-Short-Term Memory (LSTM), initialization**
   - Key idea: gated input/output/memory nodes, model choose to forget/remember
   - Example: online character recognition with LSTM recurrent neural network
4. **Transformer modules**
   - Learning temporal relationships without unrolling and without RNNs
   - Encoder/Decoder output architecture and multi-head attention modeule
5. **Graph Neural Networks**
   - Applications: social, brain, chemical drug design, graphics, transport, knowledge
   - Define each node's computation graph, from its neighborhood
   - Classical network/graph problems: Node/graph classification, link prediction
   - Research frontiers: deep generative models, latent graph inferences

3a. Remembering for longer time periods

Four effective ways to increase length of memory

- Long Short Term Memory
  Make the RNN out of little modules that are designed to remember values for a long time.
- Hessian Free Optimization: Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature.
  - The HF optimizer (Martens & Sutskever, 2011) is good at this.
- Echo State Networks: Initialize the input→hidden and hidden→hidden and output→hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input.
  - ESNs only need to learn the hidden→output connections.
- Good initialization with momentum
  Initialize like in Echo State Networks, but then learn all of the connections using momentum.

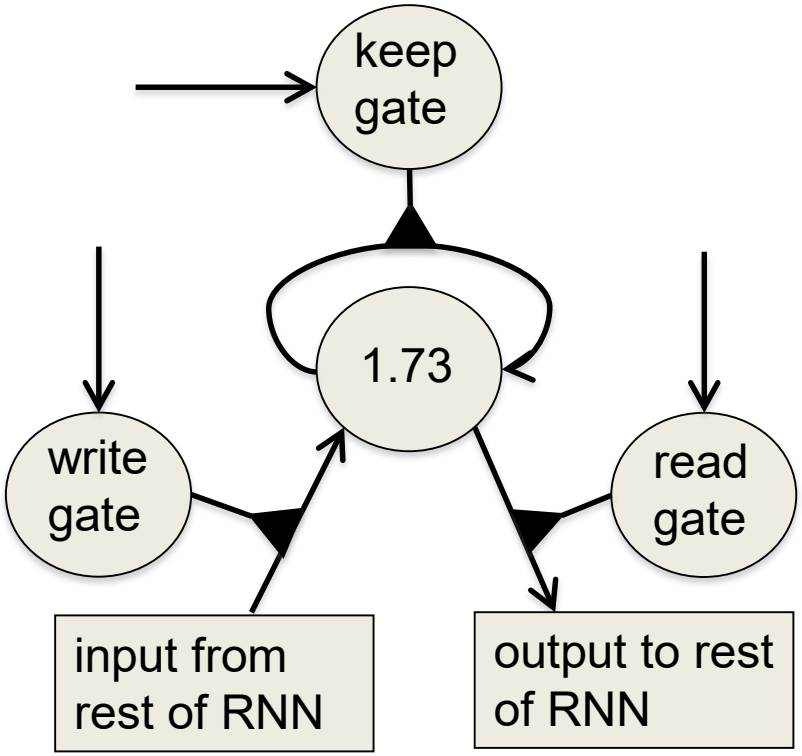
Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- They designed a memory cell using logistic and linear units with multiplicative interactions.
- Information gets into the cell whenever its “write” gate is on.
- The information stays in the cell so long as its “keep” gate is on.
- Information can be read from the cell by turning on its “read” gate.

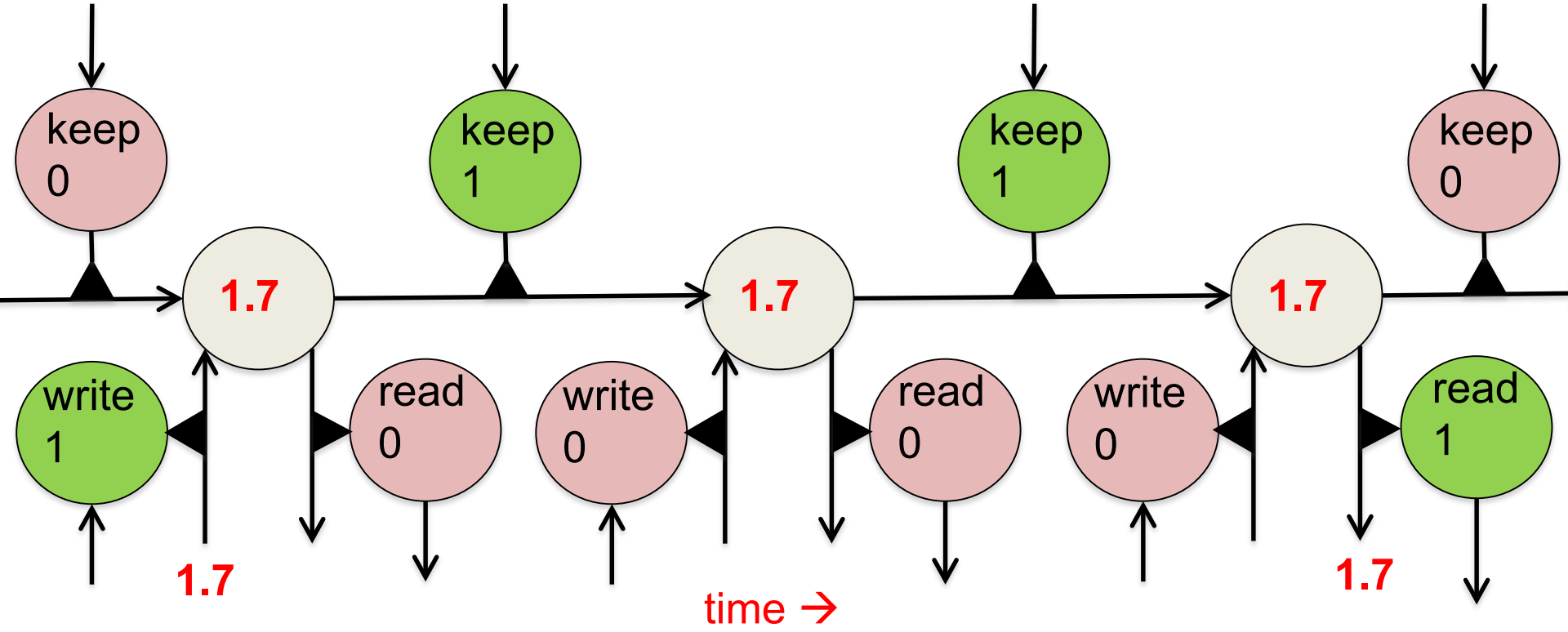
Implementing a memory cell in a neural network

To preserve information for a long time in the activities of an RNN, we use a circuit that implements an analog memory cell.

- A linear unit that has a self-link with a weight of 1 will maintain its state.
- Information is stored in the cell by activating its write gate.
- Information is retrieved by activating the read gate.
- We can backpropagate through this circuit because logistics are have nice derivatives.



RNN LSTM Backpropagation through a memory cell

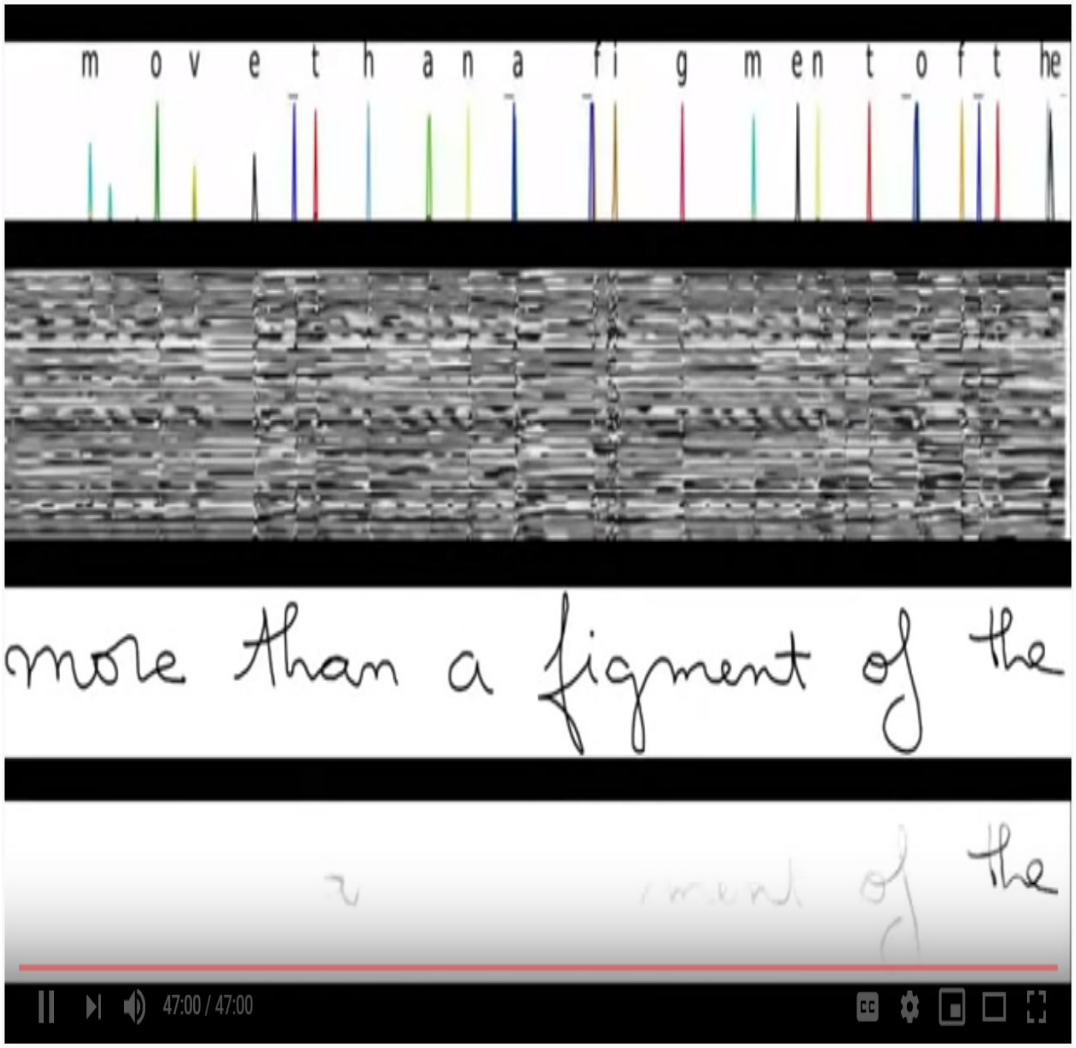


Reading cursive handwriting

- This is a natural task for an RNN.
- The input is a sequence of (x,y,p) coordinates of the tip of the pen, where p indicates whether the pen is up or down.
- The output is a sequence of characters.

- Graves & Schmidhuber (2009) showed that RNNs with LSTM are currently the best systems for reading cursive writing.
  - They used a sequence of small images as input rather than pen coordinates.

Demonstration of online handwriting recognition by an RNN with Long Short Term Memory (from Alex Graves)



https://youtu.be/9T2X6WRUwFU?t=2791

Row 1: Shows when characters are recognized.

- It never revises its output so difficult decisions are more delayed.

Row 2: Shows the states of a subset of the memory cells.

- Notice how they get reset when it recognizes a character.

Row 3: Shows the writing. The net sees the x and y coordinates.

- Optical input actually works a bit better than pen coordinates.

Row 4: Shows the gradient backpropagated all the way to the x and y inputs from the currently most active character.

- This lets you see which bits of the data are influencing the decision.

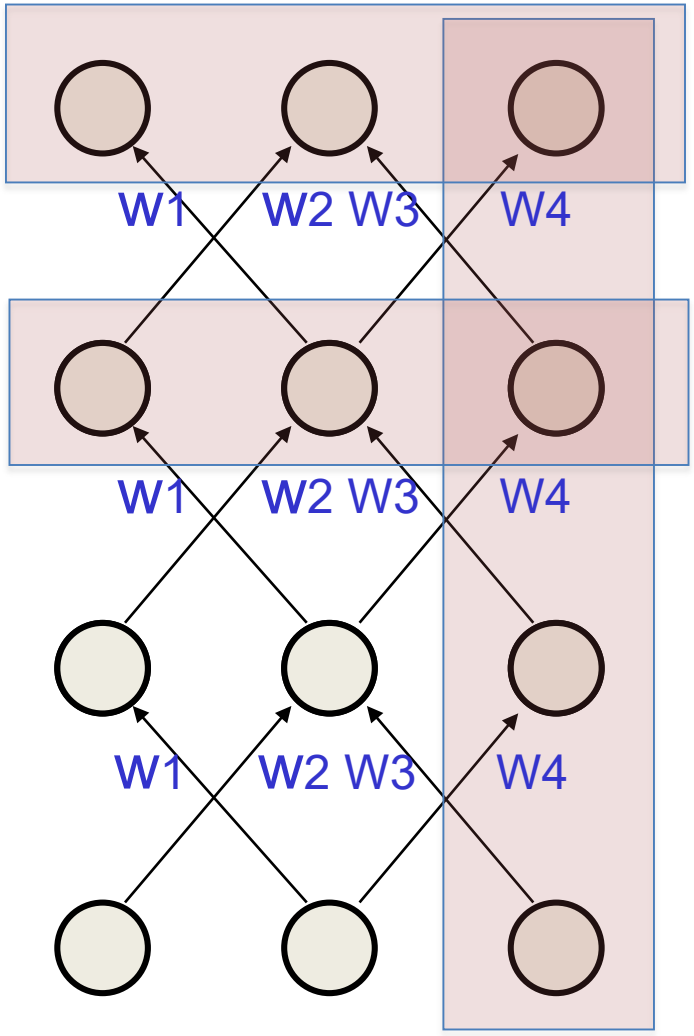
3b. Initialization

Initialization: Dealing with boundary cases

- We need to specify the initial activity state of all the hidden and output units.
- We could just fix these initial states to have some default value like 0.5.
- But it is better to treat the initial states as learned parameters.
- We learn them in the same way as we learn the weights.
  - Start off with an initial random guess for the initial states.
  - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state.
  - **Adjust the initial states** by following the negative gradient.

Teaching signals for recurrent networks

- We can specify targets in several ways:
  - Specify **desired final activities** of all the units
  - Specify desired activities of all units for the **last few steps**
    - Good for learning **attractors**
    - It is easy to add in extra error derivatives as we backpropagate.
  - Specify the desired activity of a subset of the units.
    - The other units are input or hidden units.

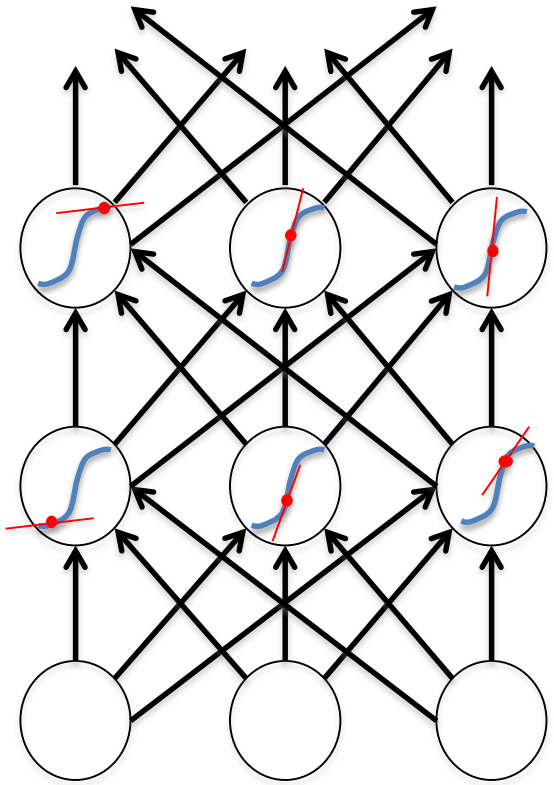


What the network learns

- It learns four distinct patterns of activity for the 3 hidden units. These patterns correspond to the nodes in the finite state automaton.
  - Do not confuse units in a neural network with nodes in a finite state automaton. Nodes are like activity vectors.
  - The **automaton** is restricted to be in **exactly one state** at each time. The hidden units are restricted to have exactly **one vector of activity at each time**.
- A recurrent network can **emulate a finite state automaton**, but it is **exponentially more powerful**. With N hidden neurons it has 2^N possible binary activity vectors (but only N^2 weights)
  - This is important when the input stream has two separate things going on at once.
  - A finite state automaton needs to square its number of states.
  - An RNN needs to double its number of units.

The backward pass is linear

- There is a big difference between the forward and backward passes.
- In the **forward** pass we use **squashing** functions (like the logistic) to **prevent** the activity vectors from **exploding**.
- The **backward** pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double.
  - The forward pass determines the slope of the linear function used for backpropagating through each neuron.



The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the **gradients shrink exponentially**.
  - If the weights are big the gradients **grow exponentially**.
- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.
- In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with **long-range dependencies**.
  - Can use ideas for residual networks (ResNet), pass info from the input to far away nodes

Recurrent Neural Networks (RNNs) + Generalization

1. **How do you read/listen/understand/write? Can machines do that?**
   - Context matters: characters, words, letters, sounds, completion, multi-modal
   - Predicting next word/image: from unsupervised learning to supervised learning
2. **Encoding temporal context: Hidden Markov Models (HMMs), RNNs**
   - Primitives: hidden state, memory of previous experiences, limitations of HMMs
   - RNN architectures,unrolling,back-propagation-through-time(BPTT),param reuse
3. **Vanishing gradients, Long-Short-Term Memory (LSTM), initialization**
   - Key idea: gated input/output/memory nodes, model choose to forget/remember
   - Example: online character recognition with LSTM recurrent neural network
4. **Transformer modules**
   - Learning temporal relationships without unrolling and without RNNs
   - Encoder/Decoder output architecture and multi-head attention modeule
5. **Graph Neural Networks**
   - Applications: social, brain, chemical drug design, graphics, transport, knowledge
   - Define each node's computation graph, from its neighborhood
   - Classical network/graph problems: Node/graph classification, link prediction
   - Research frontiers: deep generative models, latent graph inferences

4. Attention and transformer models

Encoder/Decoder/Attention modules in Transformer

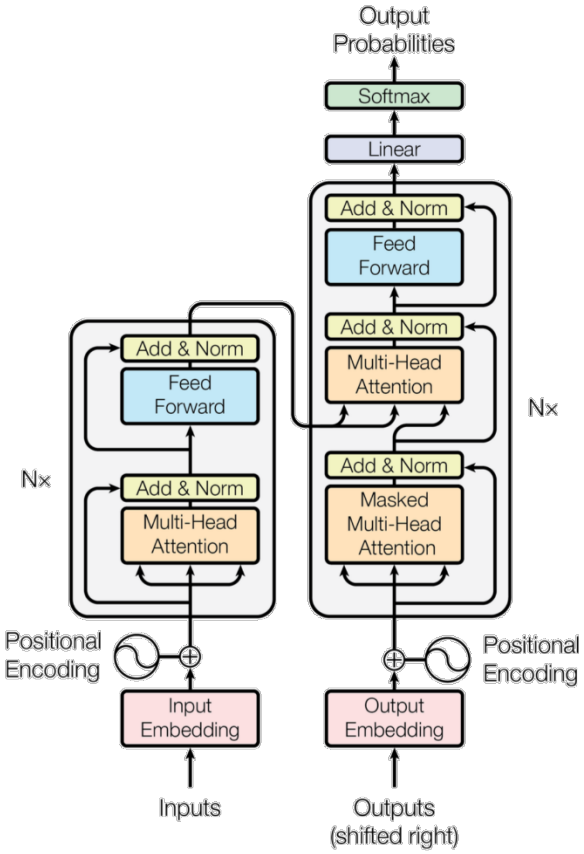


Figure 1: The Transformer - model architecture.

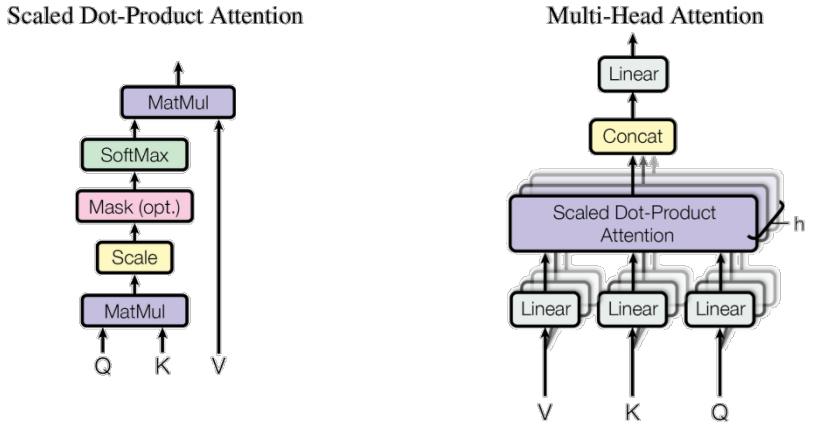


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q: matrix, query, vector representation of one word in sequence
K: all keys, vector representations of all words in sequence
V: values, vector representations of all words in sequence

Encoder, decoder, multi-head attention module: V = same word sequence as Q
Attention module = V different from Q, uses encoder and decoder sequences

Training setup: Predict next work each time, decoder shifted by one

Transforms one sequence into another sequence, using full context for each (e.g. sentence translation, or any other sequential task)

Time explicitly encoded
No need for RNN structure

Recurrent Neural Networks (RNNs) + Generalization

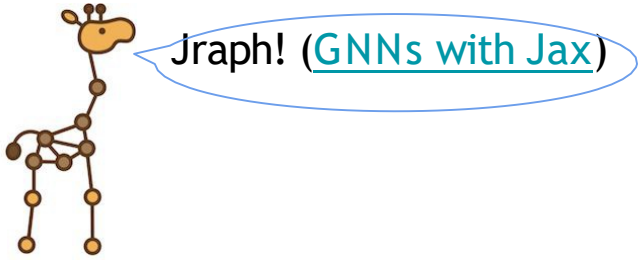
1. **How do you read/listen/understand/write? Can machines do that?**
   - Context matters: characters, words, letters, sounds, completion, multi-modal
   - Predicting next word/image: from unsupervised learning to supervised learning
2. **Encoding temporal context: Hidden Markov Models (HMMs), RNNs**
   - Primitives: hidden state, memory of previous experiences, limitations of HMMs
   - RNN architectures,unrolling,back-propagation-through-time(BPTT),param reuse
3. **Vanishing gradients, Long-Short-Term Memory (LSTM), initialization**
   - Key idea: gated input/output/memory nodes, model choose to forget/remember
   - Example: online character recognition with LSTM recurrent neural network
4. **Transformer modules**
   - Learning temporal relationships without unrolling and without RNNs
   - Encoder/Decoder output architecture and multi-head attention modeule
5. **Graph Neural Networks**
   - Applications: social, brain, chemical drug design, graphics, transport, knowledge
   - Define each node's computation graph, from its neighborhood
   - Classical network/graph problems: Node/graph classification, link prediction
   - Research frontiers: deep generative models, latent graph inferences

5. Graph Neural Networks

Guest lecture by Neil Band

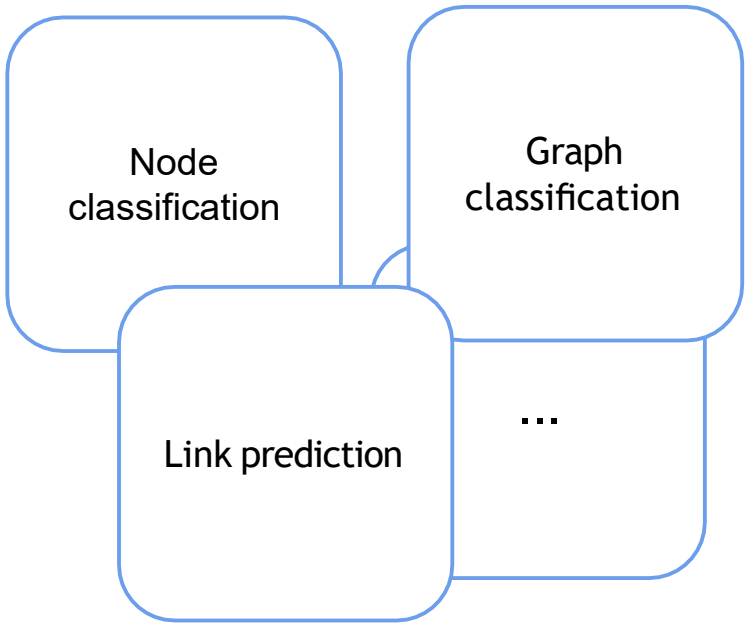
Sources / Further Reading

- Adapted from
  - Thomas Kipf's presentations (Cambridge CompBio, IPAM UCLA)
  - Graph Neural Networks by Xavier Bresson (Guest lecture in Yann LeCun's NYU DL course)
  - CS224 Machine Learning on Graphs by Jure Leskovec (Course @ Stanford)
  - Junction Tree Variational Autoencoder (Wengong Jin, ICML 2018)
- Mining and Learning with Graphs at Scale (Google Graph Mining team @ NIPS 2020)
- Graph Representation Learning (Book by Will Hamilton, 2020)
- Thomas Kipf's thesis (Deep Learning with Graph Structured Representations, 2020)
- **Further reading:** Petar Veličković's thread of resources



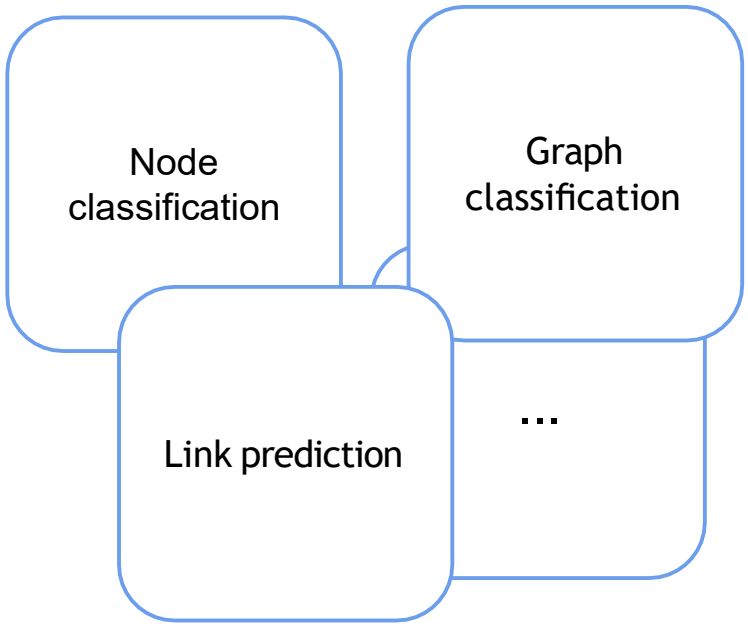
Outline

1. **Motivation**
2. **Graph neural nets (GNNs)**
   Introduction and history
3. **GNNs for classic network problems**

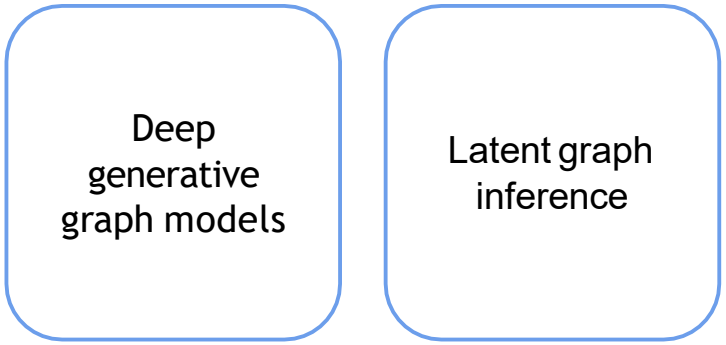


Outline

1. **Motivation**
2. **Graph neural nets (GNNs)**
   Introduction and history
3. **GNNs for classic network problems**



4. **Research frontiers**

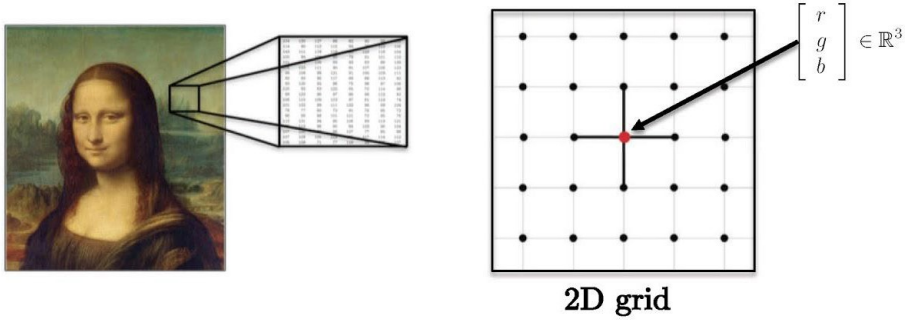


With applications in...

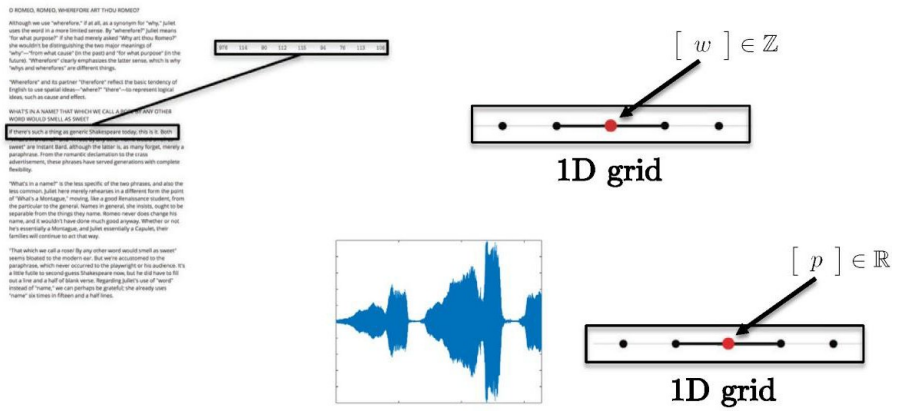
- Chemical synthesis
- Interacting systems (physical, multi-agent, biological)
- Causal inference
- Program induction

The ML canon lives in grid world

- **Images, volumes, videos** lie on **2D, 3D, 2D + t grids**



- **Sentences, words, speech** lie on **1D grids**



- **Deep neural nets on grids exploit:**
  - translation equivariance (weight sharing)
  - hierarchical compositionality

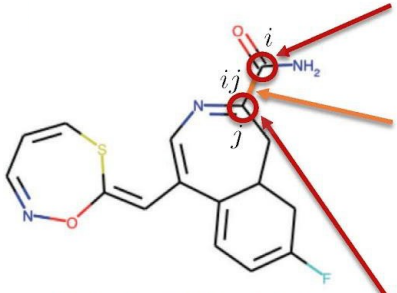
But there's so much more...

Social networks
(Advertisement/
recommendation)

$$\text{User}_i \begin{bmatrix} \text{messages} \\ \text{images} \\ \text{videos} \end{bmatrix}_i \in \mathbb{R}^d$$

$$\text{User connection}_{ij} \quad A_{ij} = \begin{cases} 1 & \text{if ij friends} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{User}_j \begin{bmatrix} \text{messages} \\ \text{images} \\ \text{videos} \end{bmatrix}_j \in \mathbb{R}^d$$



Quantum Chemistry
(novel molecules for drugs
and materials)

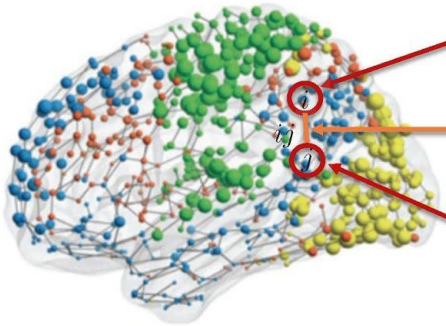
$$\text{Atom}_i \begin{bmatrix} \text{type} \\ \text{coordinates} \\ \text{charge} \end{bmatrix}_i \in \mathbb{R}^{d_v}$$

$$\text{Bond}_{ij} \quad A_{ij} = \begin{cases} 1 & \text{if ij bond} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{bmatrix} \text{type} \\ \text{energy} \end{bmatrix}_{ij} \in \mathbb{R}^{d_e}$$

$$\text{Atom}_j \begin{bmatrix} \text{type} \\ \text{coordinates} \\ \text{charge} \end{bmatrix}_j \in \mathbb{R}^{d_v}$$

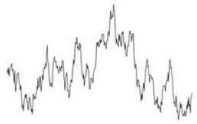
Brain connectivity
(sMRI)



$$\text{ROI}_i \begin{bmatrix} a_1 \\ \vdots \\ a_T \end{bmatrix}_i \in \mathbb{R}^T$$

$$\text{Cerebral connection}_{ij} \quad A_{ij} \in \mathbb{R}_+$$

$$\text{ROI}_j \begin{bmatrix} a_1 \\ \vdots \\ a_T \end{bmatrix}_j \in \mathbb{R}^T$$

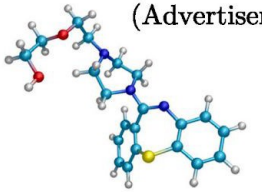


Brain analysis
(Neuroscience/neuro-diseases)

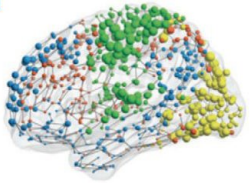
Functional
activations (fMRI)

But there's so much more...

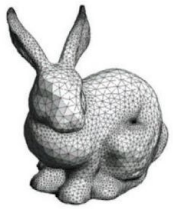
Social networks (Advertisement)



Drug/Material molecules (Chemistry)

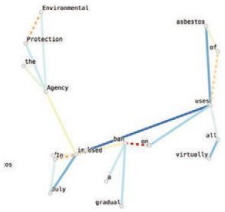


Brain connectivity (Neuroscience)

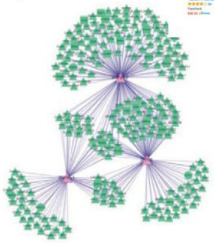


3D Meshes (Computer Graphics)

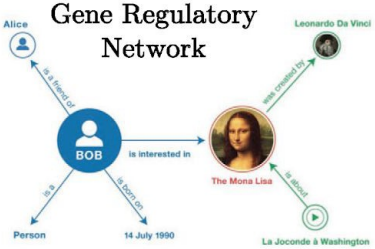
Transportation networks



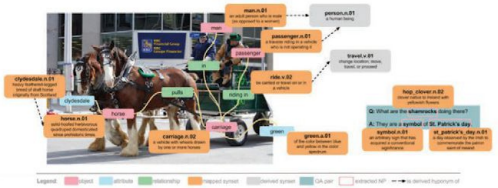
Words relationships (NLP)



Gene Regulatory Network

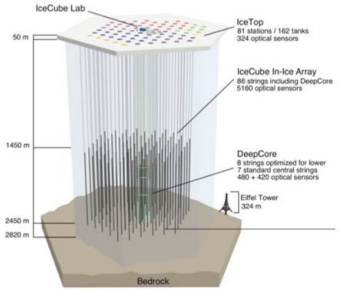


Knowledge graphs



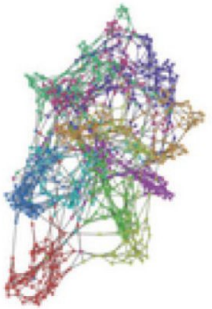
Scene understanding

Recommender systems (Amazon, Netflix)



Neutrino detection (High-energy Physics)

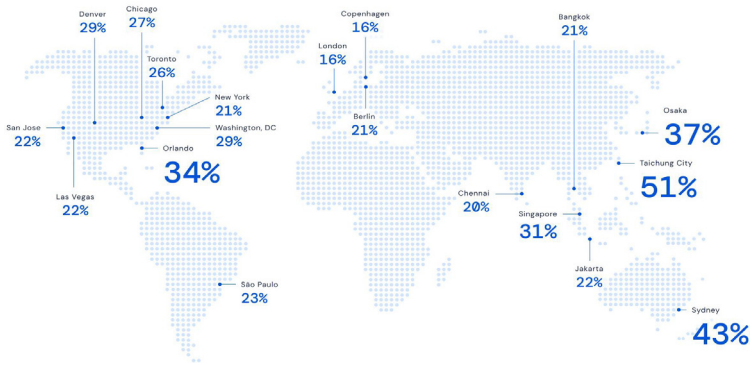
=



Graph

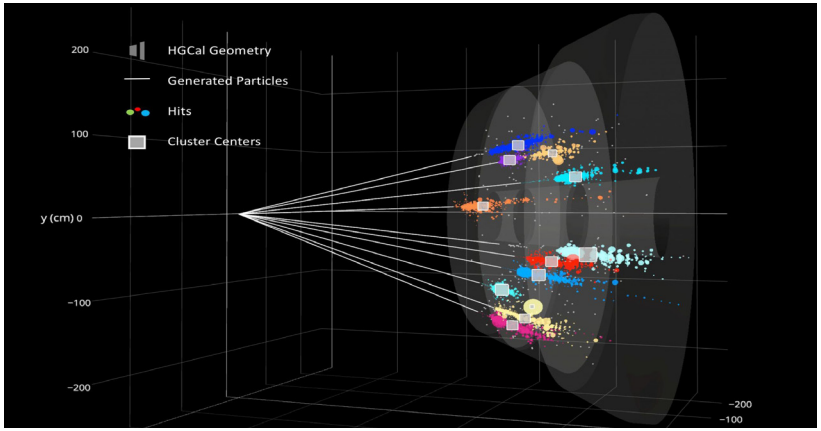
Cool applications (in the last year!)

DeepMind / Google Maps ETA improvements across world

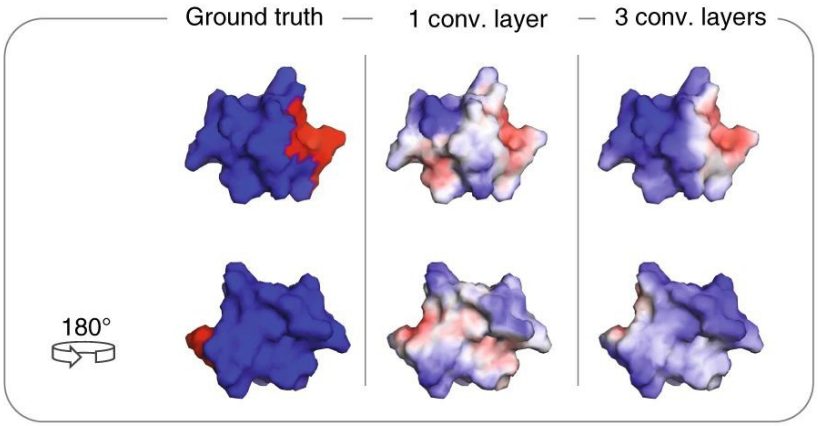


SuperGlue (Magic Leap) feature matching

Large Hadron Collider real-time collision analysis

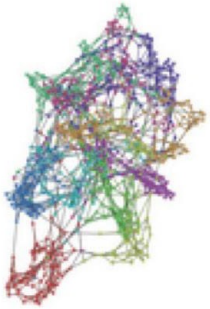


MaSIF predicts protein-protein interactions

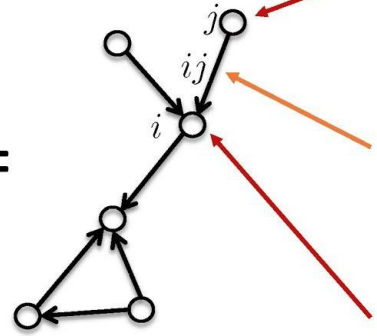


Setup

- Graphs G are defined by :
  - Vertices V
  - Edges E
  - Adjacency matrix A



=



$$h_j = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_v}$$

$$h_{(ij)} = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_e}$$

$$h_i = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_v}$$

- Graph features :
  - Node features : $h_i,\ h_j$ (atom type)
  - Edge features : $h_{(ij)}$ (bond type)
  - Graph features : g (molecule energy)

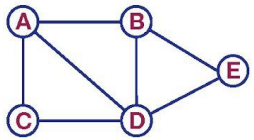
$$\mathcal{G} = (V, E, A)$$

$$V = \{1, .., n\}$$

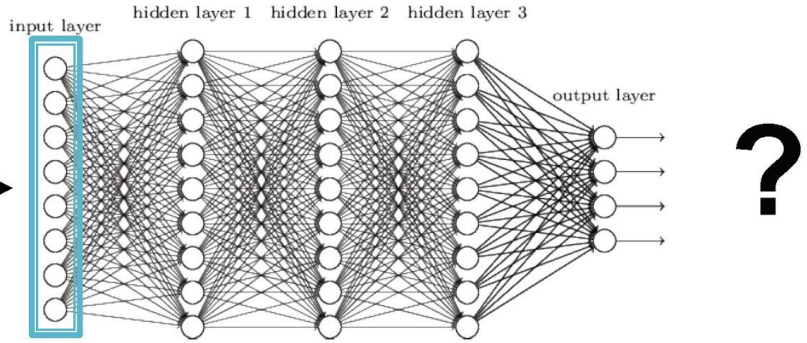
$$g = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_g}$$

Naive approach

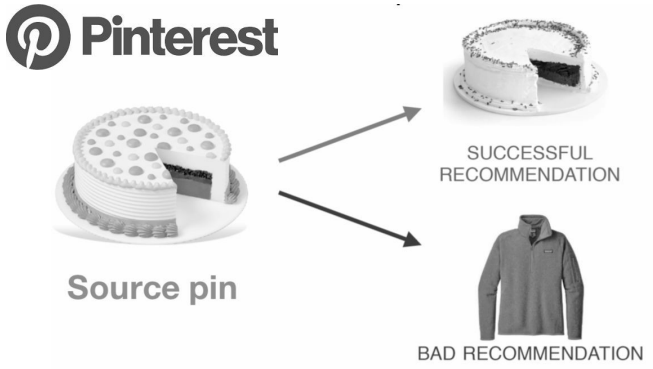
1. Join adjacency matrix and node features
2. Plug them into a deep neural net



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



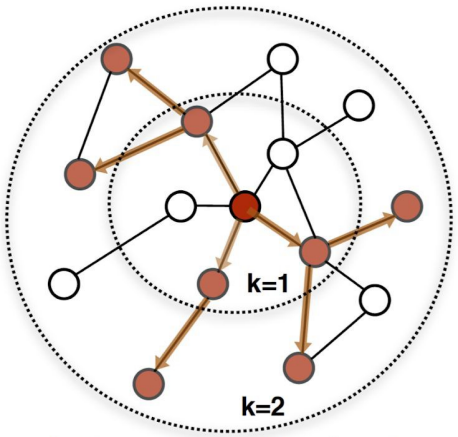
- Issues with this idea:
  - O(N) parameters → 6 billion nodes in Pinterest
  - Not applicable to graphs of different sizes → graphs change!
  - Not invariant to node ordering → expensive sorting



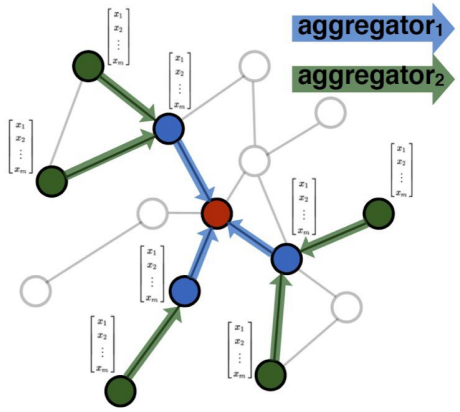
Graph neural nets

Aggregating neighbors

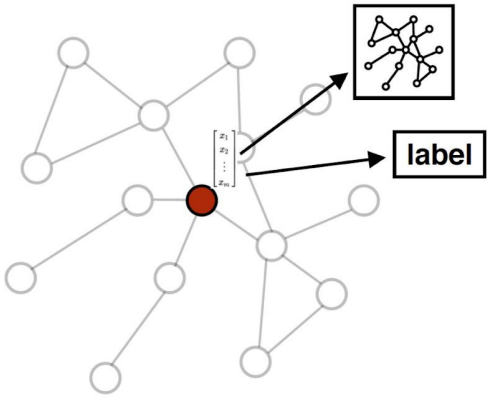
Idea: Node's neighborhood defines a computation graph



1. Sample neighborhood



2. Aggregate feature information from neighbors

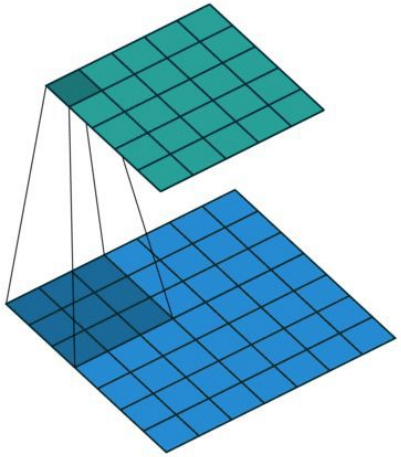


3. Predict graph context and label using aggregated information

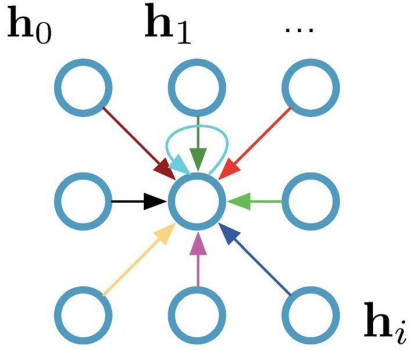
Learn how to propagate information across the graph to compute node features

Recap: CNN (on grids) as message passing

Single CNN layer with 3x3 filter



Animation by Vincent Dumoulin



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

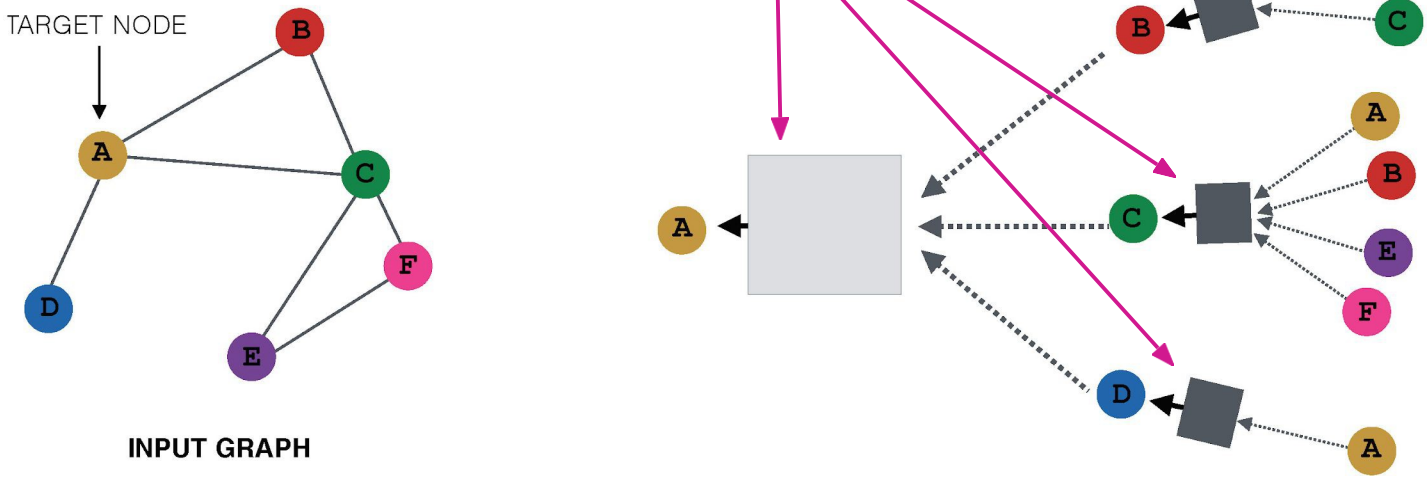
$\mathbf{h}_i \in \mathbb{R}^d$ are (hidden layer) activations of a pixel/node

Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

Aggregating neighbors

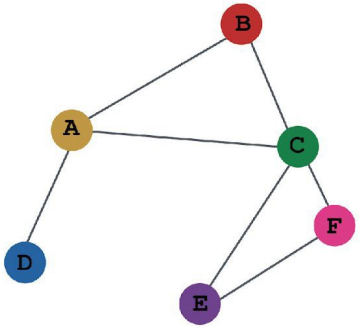
- **Key idea:** Generate node embeddings based on **local network neighborhoods**



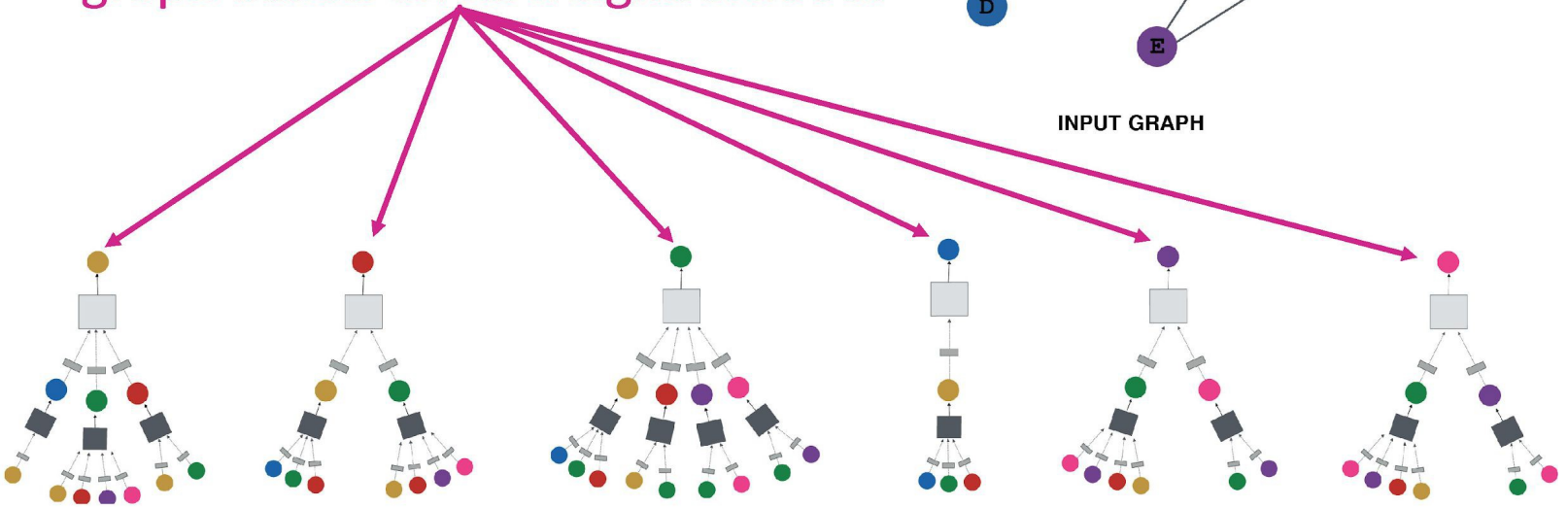
Aggregating neighbors

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



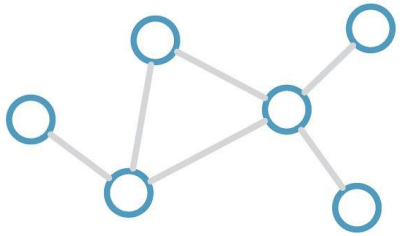
INPUT GRAPH



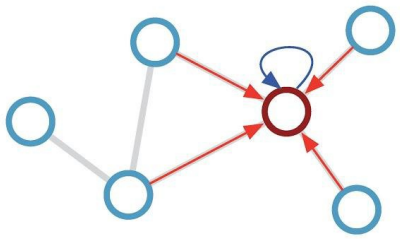
Graph convolutional networks (GCNs)

Kipf & Welling (ICLR 2017), related previous works: Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)
- Applicable both in transductive and inductive settings

Limitations:

- Requires gating mechanism / residual connections for depth
- Only indirect support for edge features

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

GCN classification on citation networks

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

Input: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

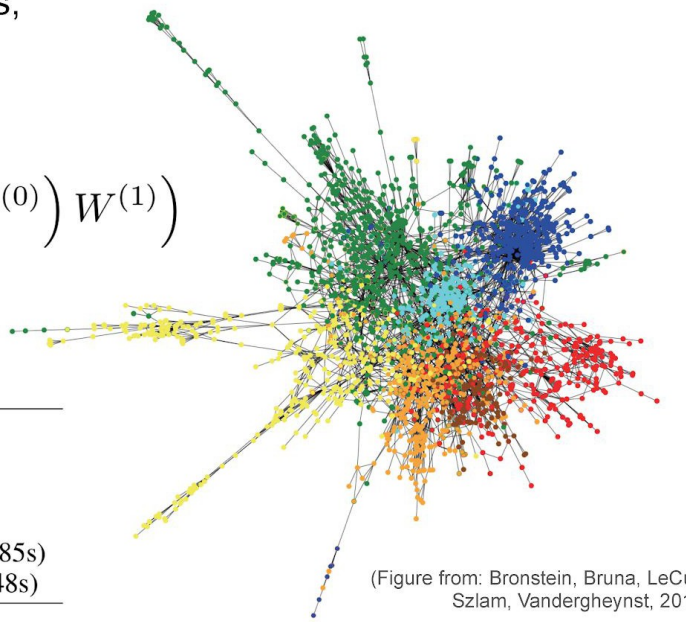
Target: Paper category (e.g. stat.ML, cs.LG, …)

Model: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$

Classification results (accuracy)

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [24]	59.6	59.0	71.1	26.7
LP [27]	45.3	68.0	63.0	26.5
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

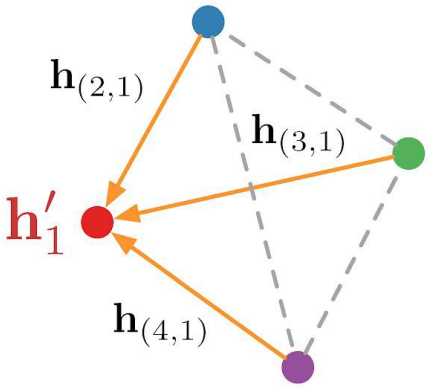
no input features



(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

GNNs with edge embeddings (Neural message passing)

Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)



Formally:

$$v \to e: \quad \mathbf{h}_{(i,j)} = f_e([\mathbf{h}_i, \mathbf{h}_j])$$

$$e \to v: \quad \mathbf{h}'_j = f_v\left(\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}\right)$$

Pros:

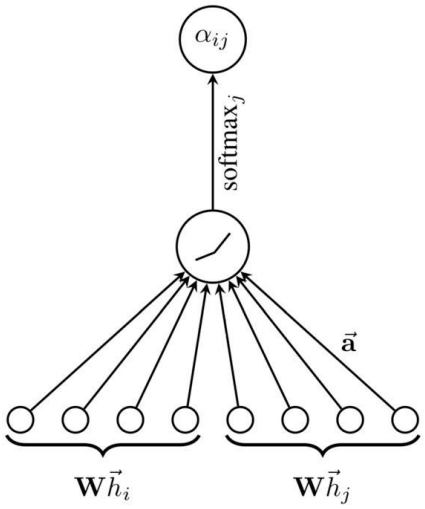
- Support for edge features
- Very flexible / expressive parameterization
- Supports sparse ops

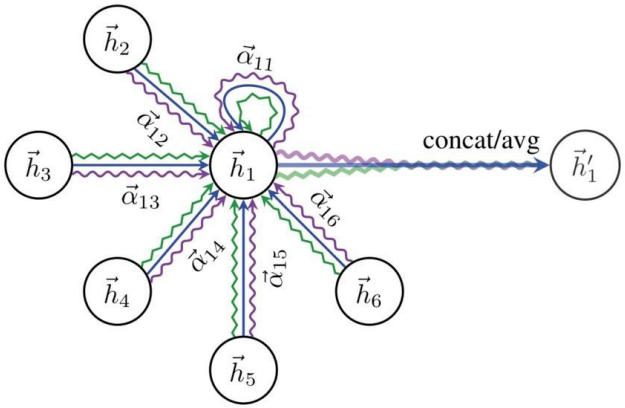
Cons:

- Need to store intermediate edge-based activations
- In practice significantly slower than GCN / self-attention models

Graph neural nets with attention

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličković et al. (ICLR 2018)





[Figure from Veličković et al. (ICLR 2018)]

Pros:

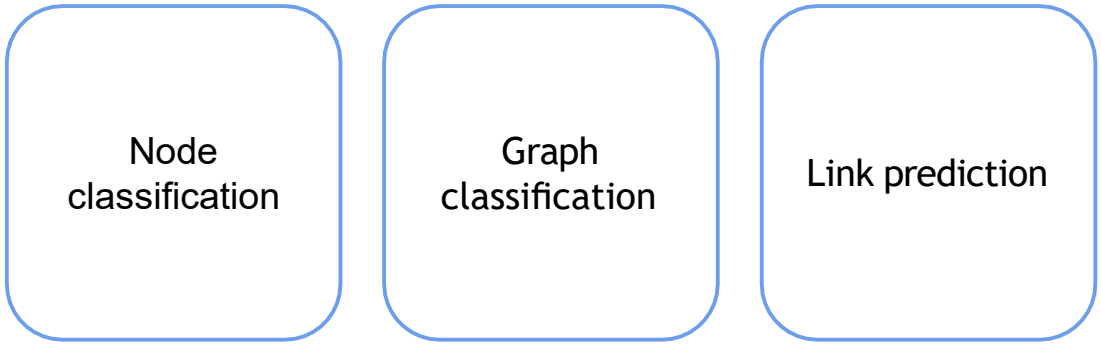
- No need to store intermediate edge-based activation vectors (when using dot-product attn.)
- Slower than GCNs but faster than GNNs with edge embeddings

Cons:

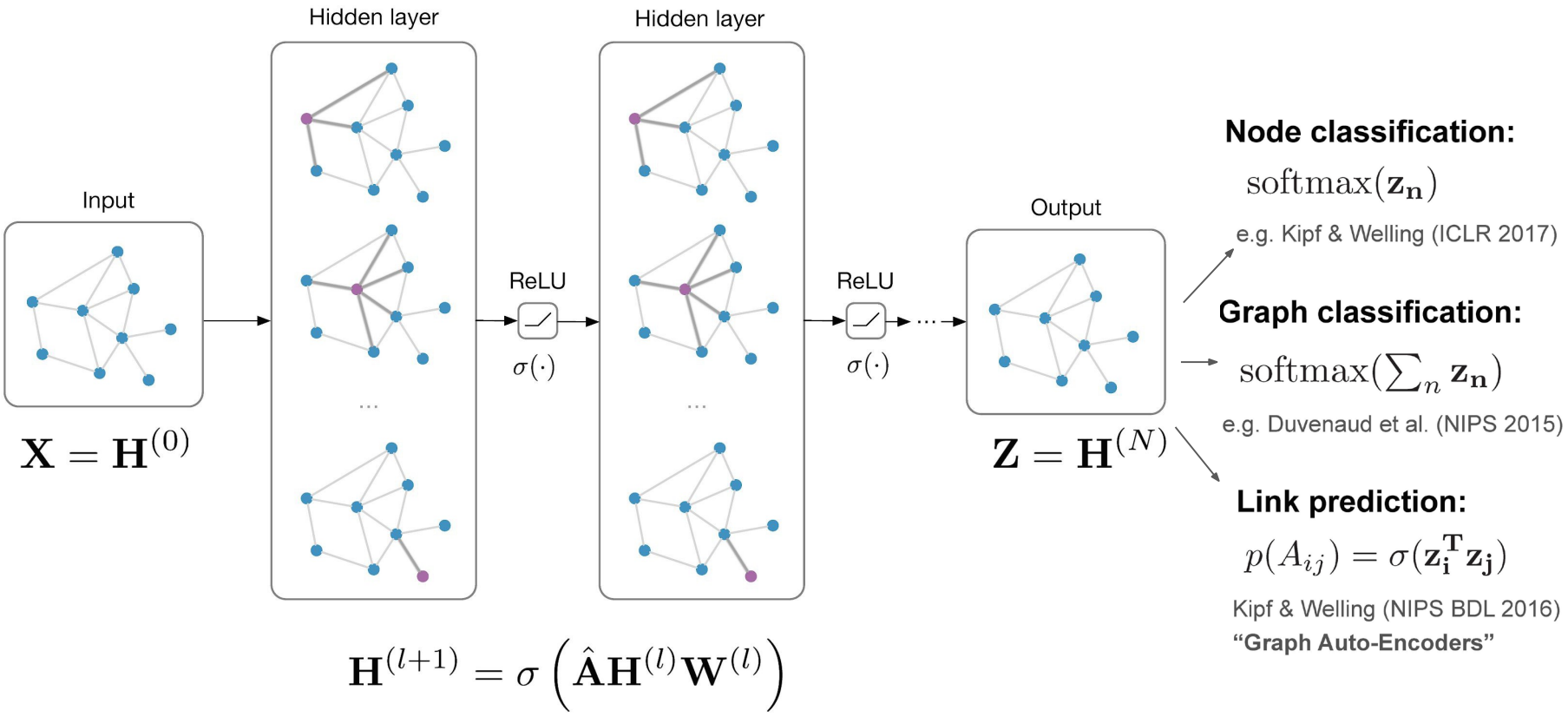
- (Most likely) less expressive than GNNs with edge embeddings
- Can be more difficult to optimize

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right) \qquad \alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

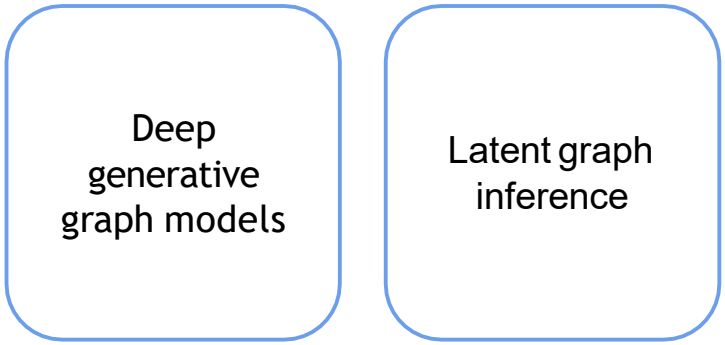
2 Application to “classical” network problems



One fits all: Classification and link prediction with GNNs / GCNs



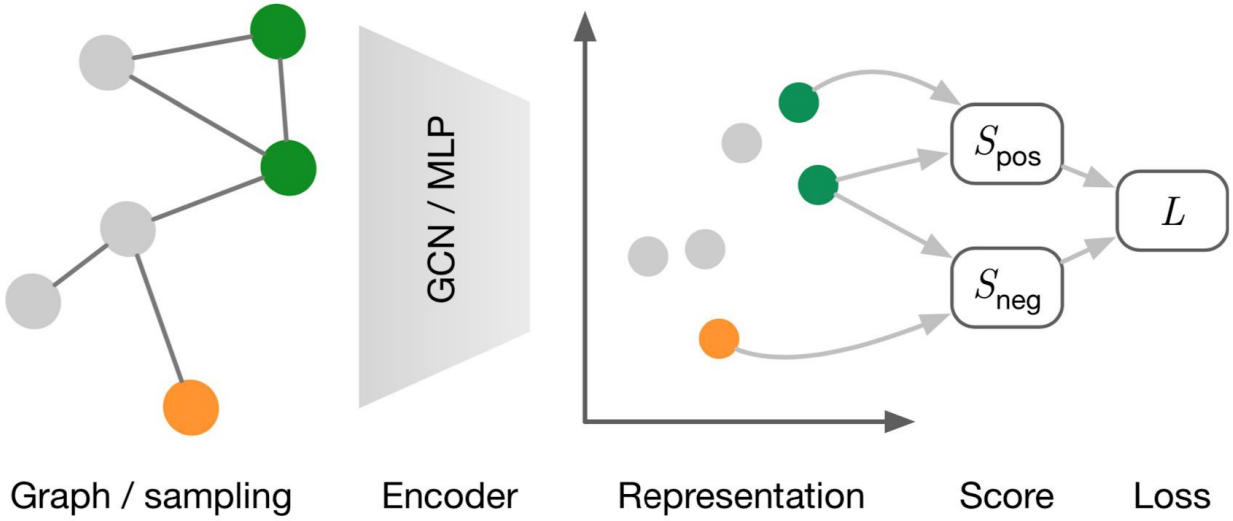
3 Research frontiers



Unsupervised learning with GNNs

Objective: Learn node embeddings for downstream tasks

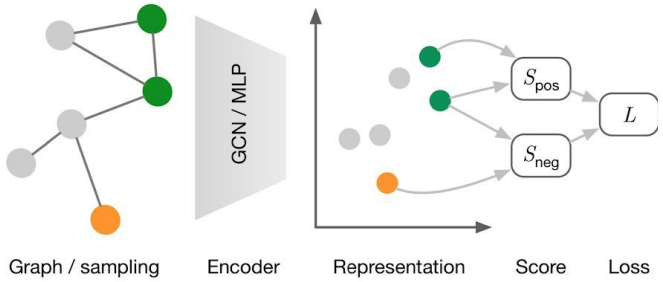
Most approaches follow a contrastive learning approach:



Unsupervised learning with GNNs

Objective: Learn node embeddings for downstream tasks

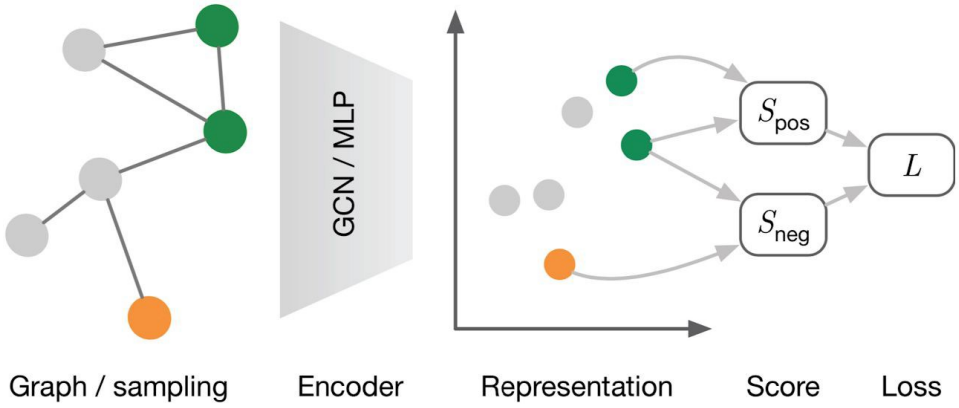
Most approaches follow a contrastive learning approach:



- Sampling strategies
  e.g. positive: neighbor; negative: random node
- Encoder variants
  GCN, GAT, MLP, Lookup table
- Node representations
  Geometry of latent space, distributional embeddings (e.g. Hyperbolic GCNN, Chami et al. 2019)
- Score functions
  ocal vs. global)
  Inner/bilinear product, local vs. global (e.g. Deep Graph Infomax, Velickovic et al. 2019)
- Loss
  (Cross-entropy, MSE, exponential)

Unsupervised learning takeaways

A Modular Framework for Unsupervised Graph Representation Learning, Daza & Kipf (WIP)



- Graph-based encoders often improve performance
- Neighbor-based scoring (GAE) effective for both link prediction & node classification
- Local-global scoring (DGI) especially effective for node classification
- Ideal node representation (distributional, hyperspherical, etc.) heavily data-dependent

Likelihood-based (deep) graph generation

Version 1: Generate graph (or predict new links) between known entities

Graph-based autoencoders:

- Encoder: GNN/GCN
- Decoder: Pairwise scoring function

$$p(A_{ij}) = f(\mathbf{z}_i, \mathbf{z}_j)$$

e.g. $p(A_{ij}) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$

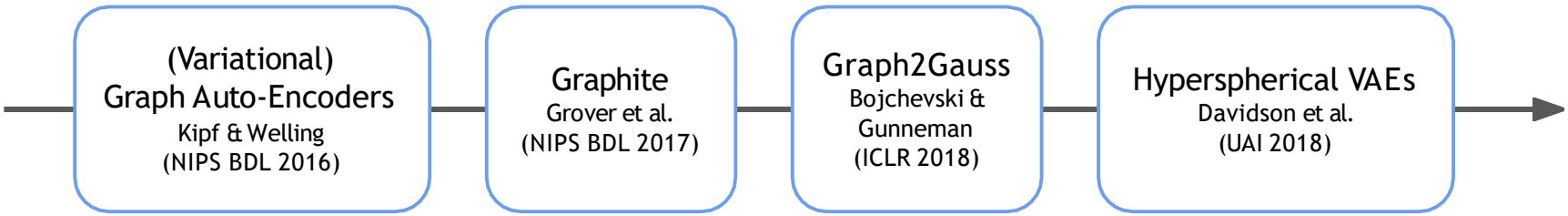
Likelihood-based:

- we have some ground truth graphs
- define likelihood as how well a generated graph matches a ground truth graph

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j) \ , \quad \text{with} \quad p(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j)$$

VGAE generative model (with ELBO loss)

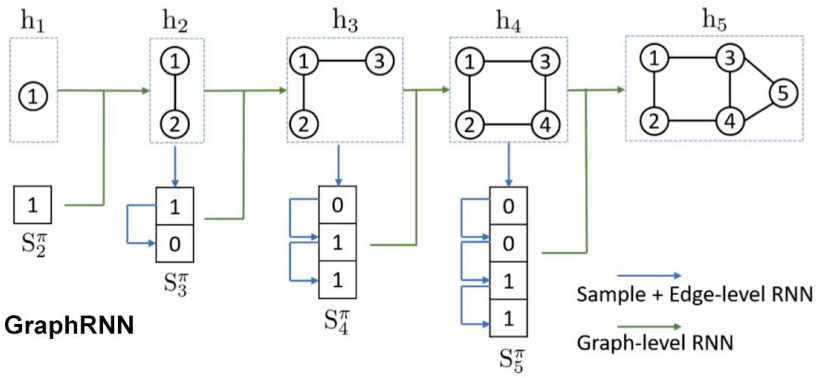
(Incomplete) History:



Likelihood-based (deep) graph generation

Version 2: Generate graphs from scratch (single embedding vector)

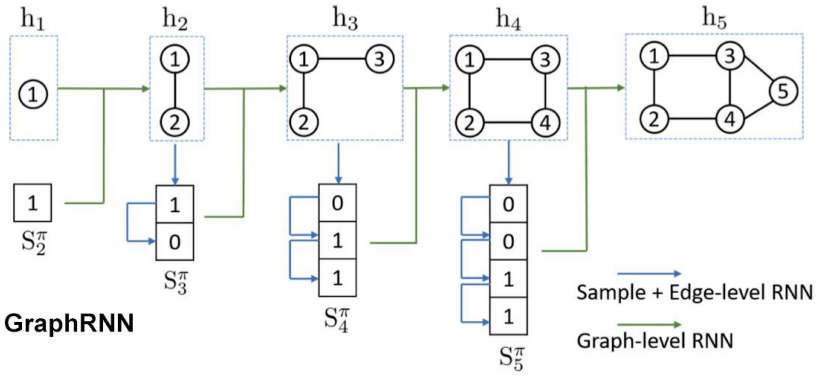
Sequentially:



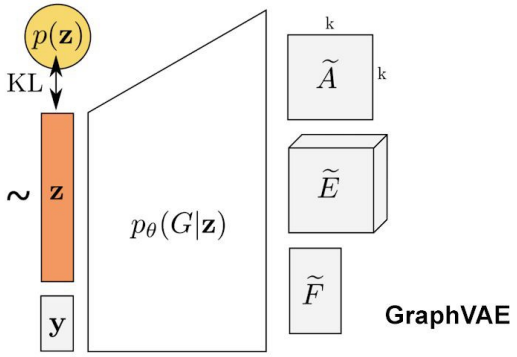
Likelihood-based (deep) graph generation

Version 2: Generate graphs from scratch (single embedding vector)

Sequentially:



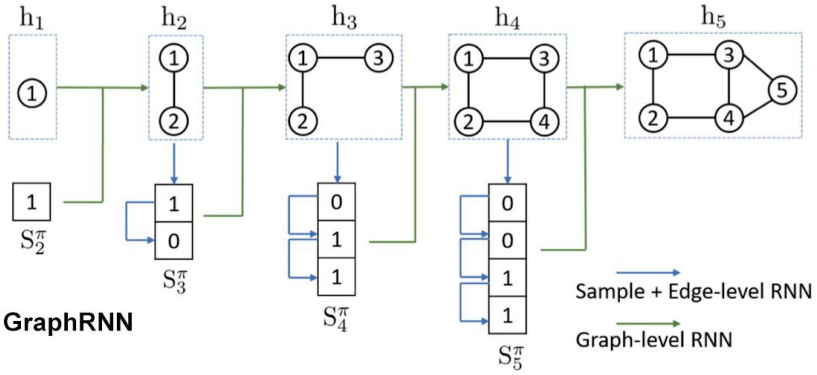
Or in a single step:



Likelihood-based (deep) graph generation

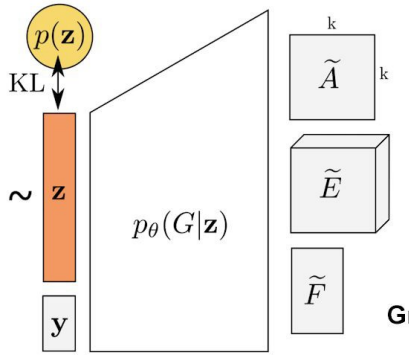
Version 2: Generate graphs from scratch (single embedding vector)

Sequentially:



GraphRNN

Or in a single step:



GraphVAE

Learning Graphical State Transitions
Johnson
(ICLR 2017)

Deep Generative Models of Graphs
Li et al.
(arXiv 2018)

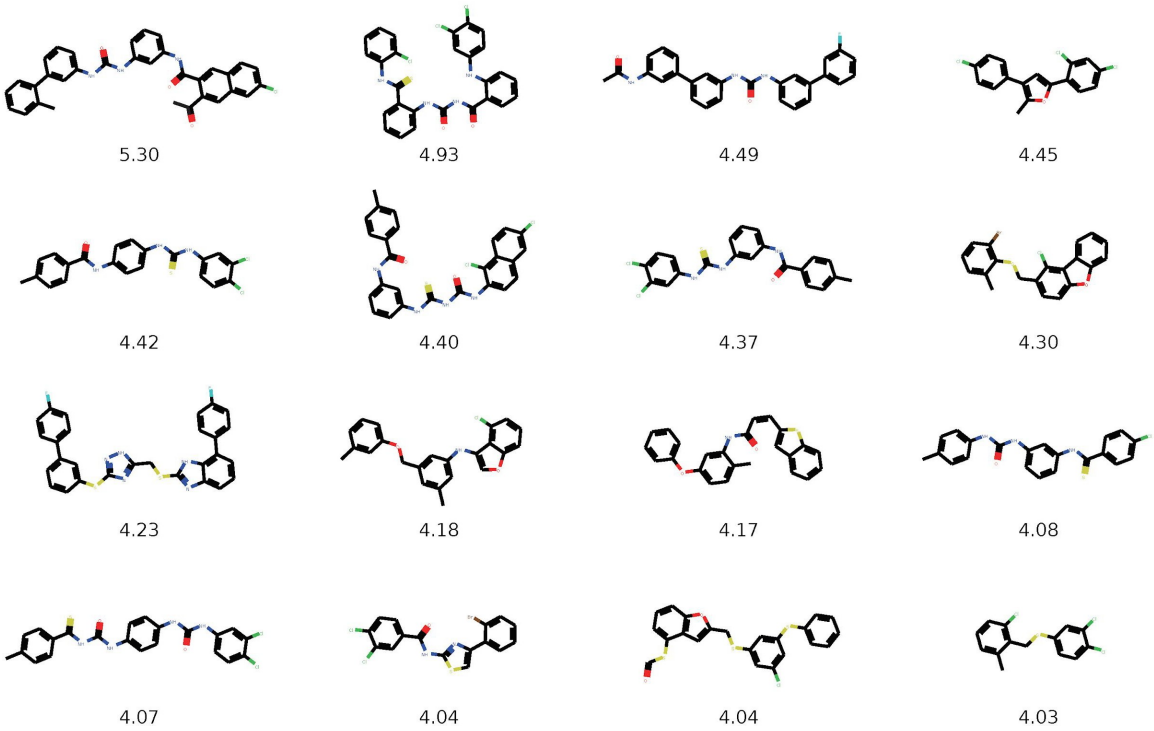
GraphVAE
Simonovsky et al.
(arXiv 2018)

GraphRNN
You et al.
(ICML 2018)

Graph generation for drug discovery

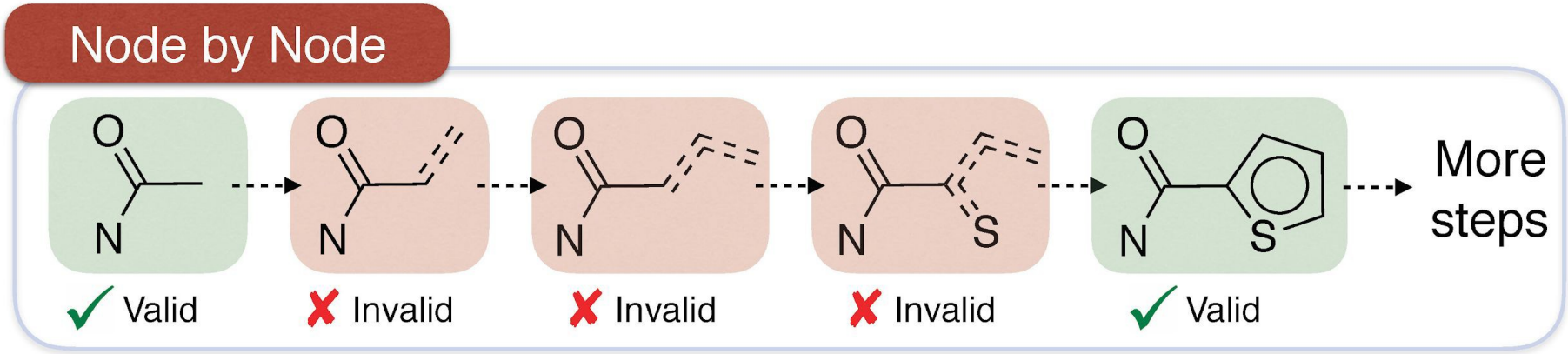
Junction Tree Variational Autoencoder, Jin et al. (ICML 2018)

Aim: generate molecules with high potency



How should we decode the graph?

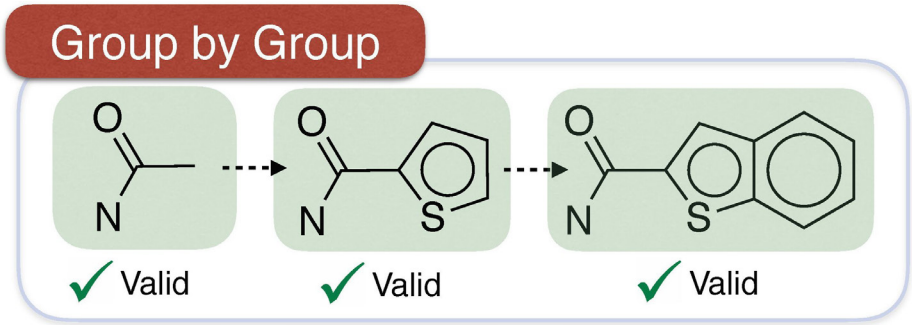
Junction Tree Variational Autoencoder, Jin et al. (ICML 2018)

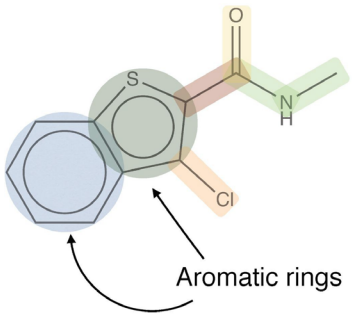


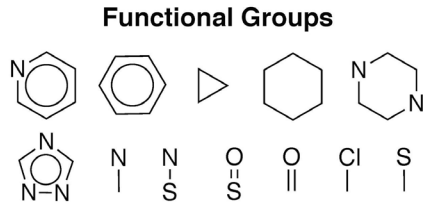
- Not every graph is chemically valid
- Invalid intermediate states → hard to validate
- Many intermediate states (i.e. long sequences) → difficult to train (Li et al. 2018)

How should we decode the graph?

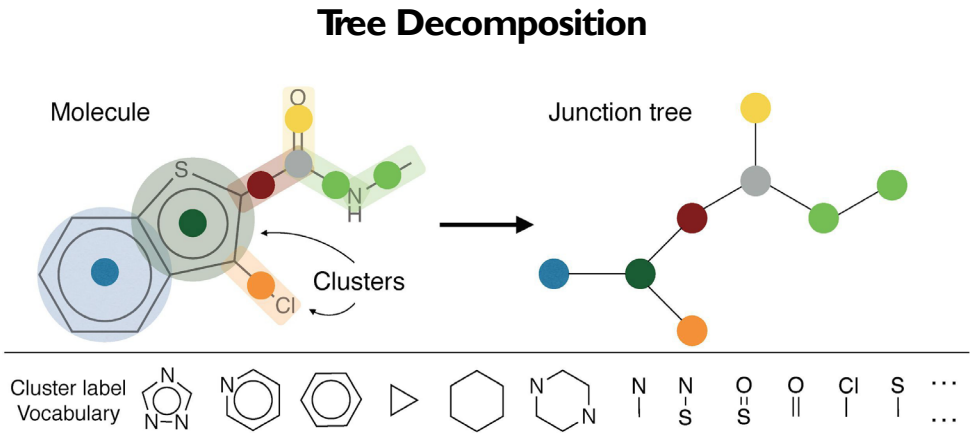
Junction Tree Variational Autoencoder, Jin et al. (ICML 2018)





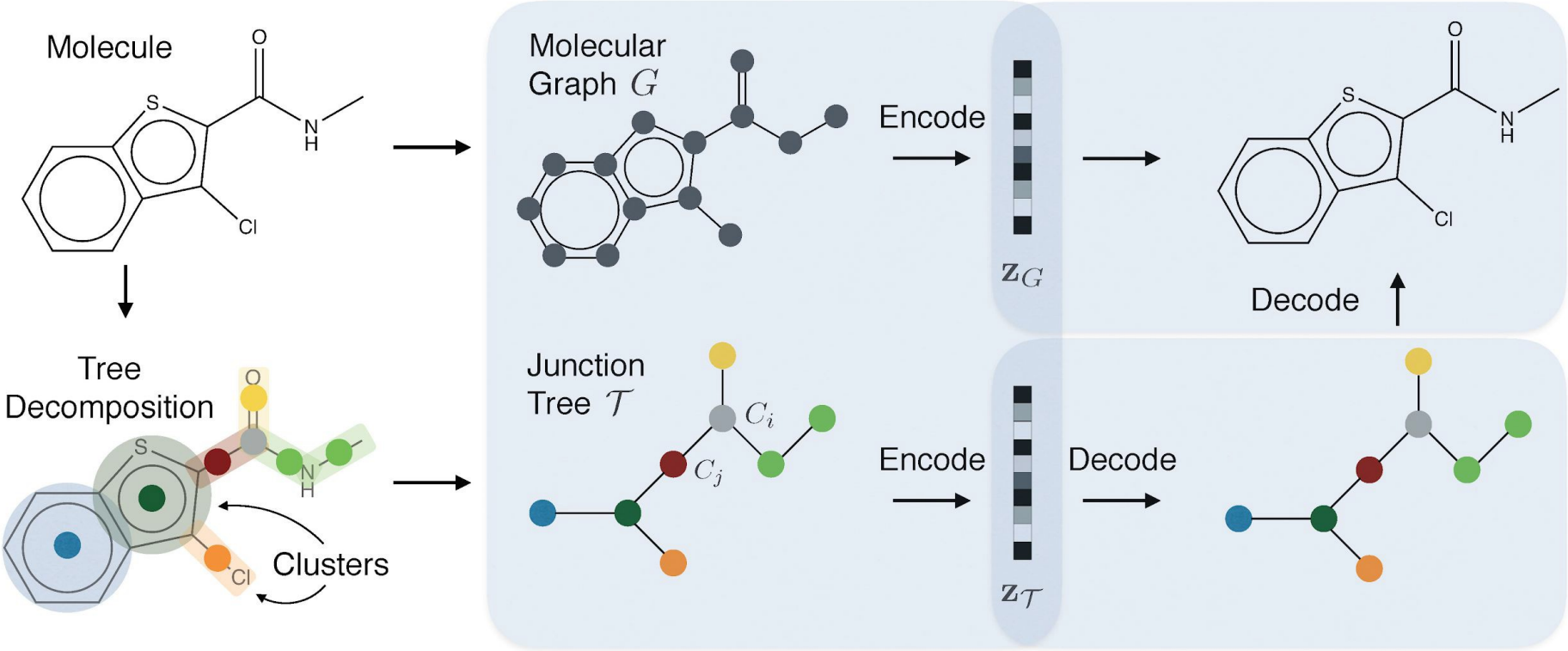


- Shorter action sequence
- Easy to check validity as we construct
- Vocabulary size: ~800 for 250K molecules



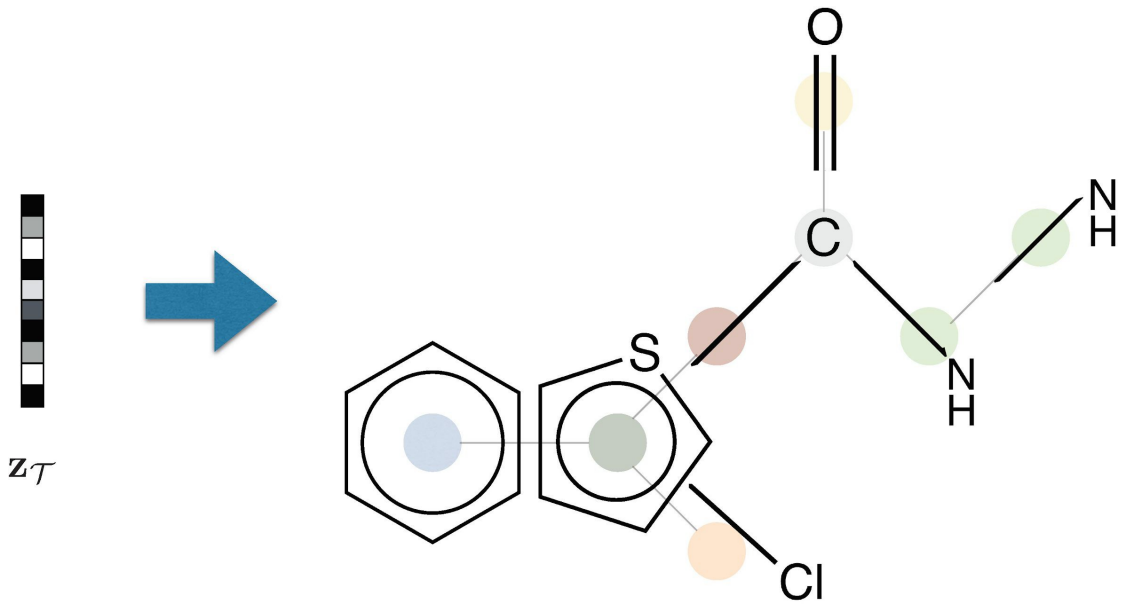
High-level approach

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018)



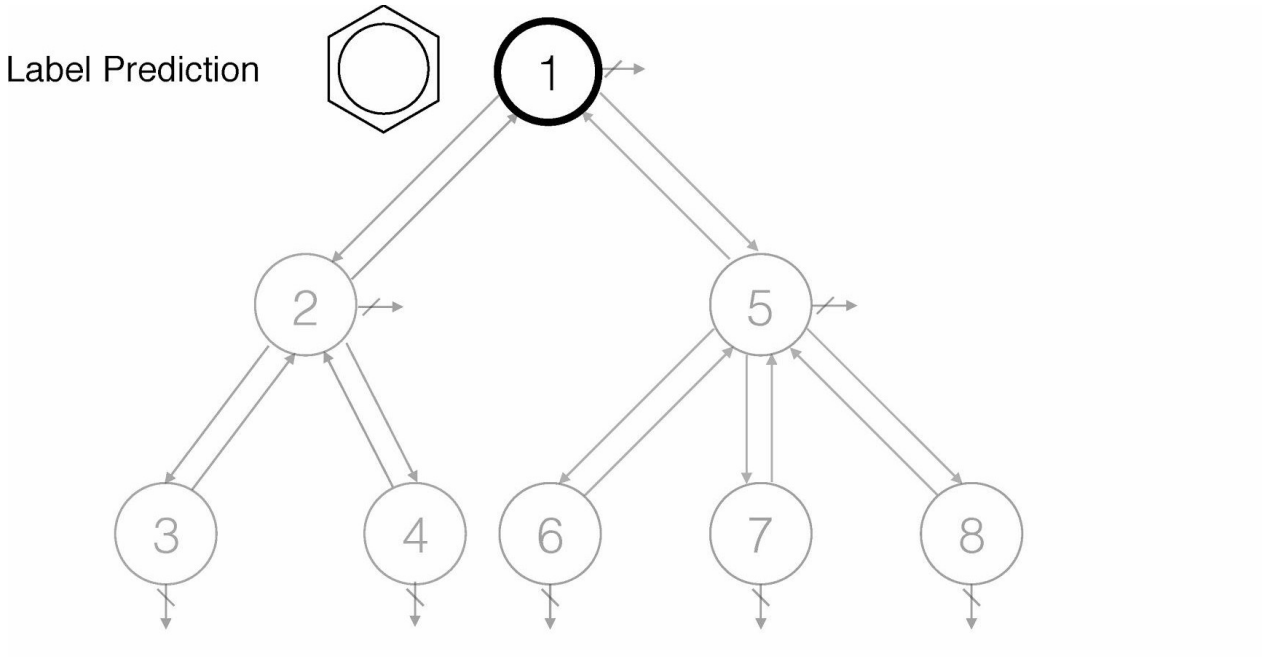
Focus on the cool part: tree decoding

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018)



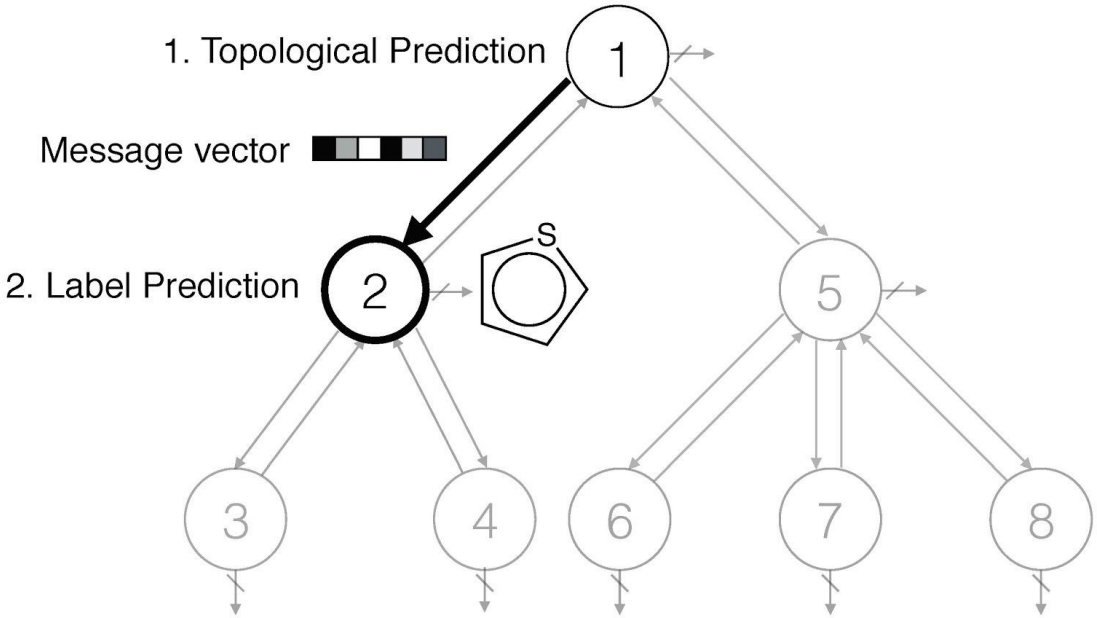
Tree decoding

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018); Tree-Structured Decoding, Alvarez-Melis & Jaakkola (ICLR 2017)



Tree decoding

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018); Tree-Structured Decoding, Alvarez-Melis & Jaakkola (ICLR 2017)

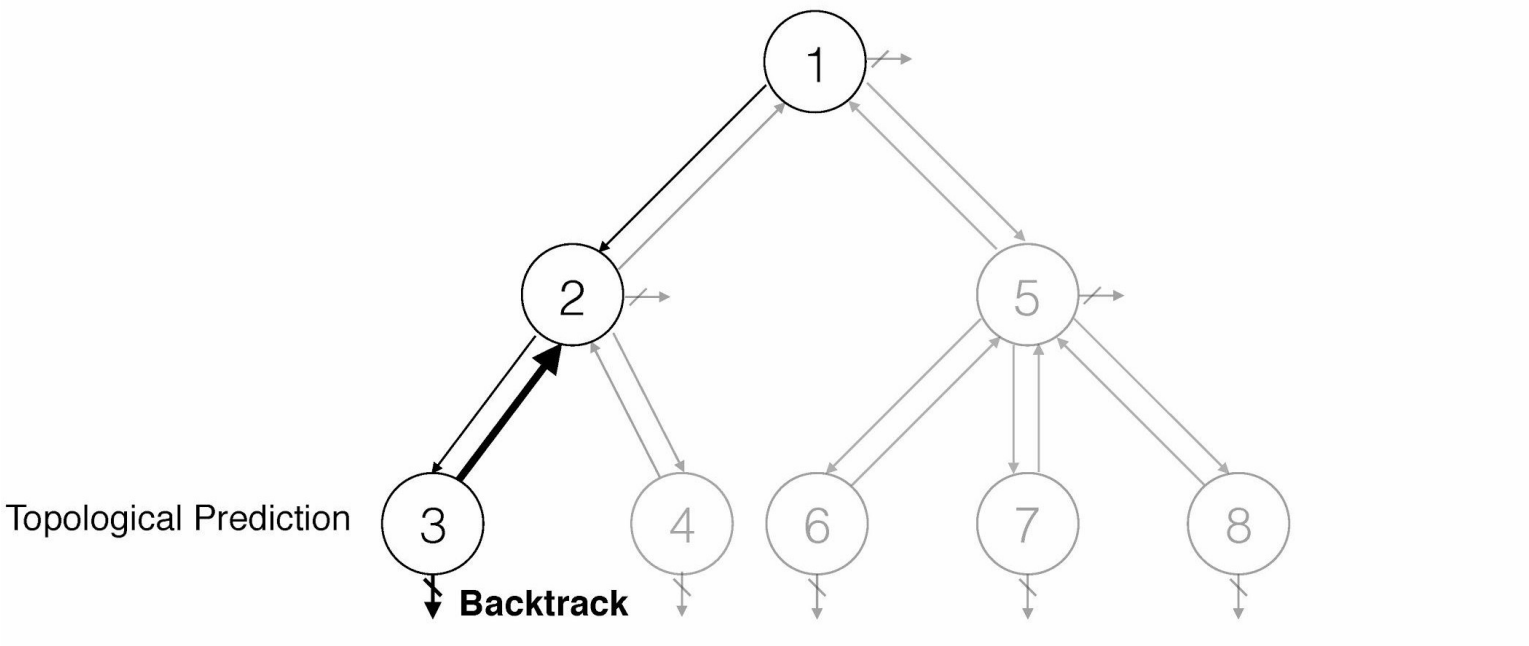


Topological Prediction: Should we add a child node, or backtrack?

Label Prediction: What do we label the new node?

Tree decoding

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018); Tree-Structured Decoding, Alvarez-Melis & Jaakkola (ICLR 2017)

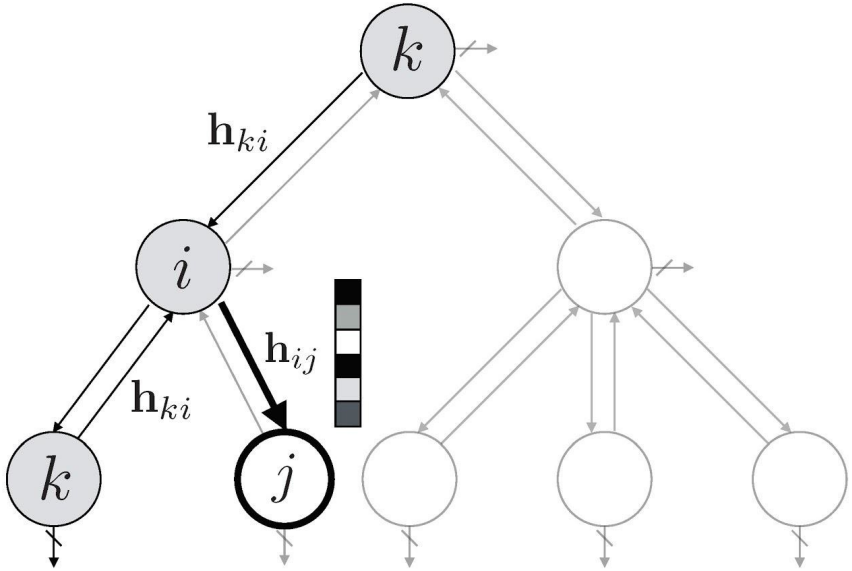


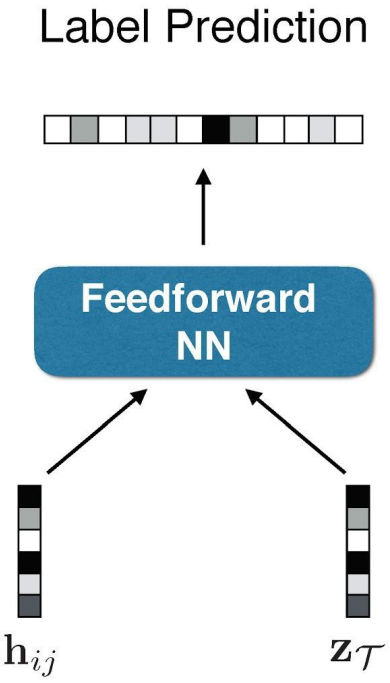
Topological Prediction: Should we add a child node, or backtrack?

Label Prediction: What do we label the new node?

Tree decoding

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018); Tree-Structured Decoding, Alvarez-Melis & Jaakkola (ICLR 2017)





$$\mathbf{h}_{ij} = \mathrm{GRU}(\mathbf{x}_i, \{\mathbf{h}_{ki}\}_{k \in N_t(i) \setminus j})$$

Encodes state of subtree thus far

Functional group features

JTVAE evaluation

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018)

Method	Reconstruction	Validity
CVAE	44.6%	0.7%
GVAE	53.7%	7.2%
SD-VAE	76.2%	43.5%
GraphVAE	-	13.5%
Atom-by-Atom LSTM	-	89.2%
JT-VAE	**76.7%**	**100.0%**

Method	1st	2nd	3rd
CVAE	1.98	1.42	1.19
GVAE	2.94	2.89	2.80
SD-VAE	4.04	3.50	2.96
JT-VAE	**5.30**	**4.93**	**4.49**

1 Molecule Reconstruction

100 forward passes per molecule, report portion of decoded molecules identical to input

2 Molecule Validity

Random samples from latent z, report portion that are chemically valid (RDKit)
JTVAE without validity checking: 93.5%

3 Bayesian Optimization

1. Train a VAE, associate each molecule with latent vector (mean of encoding distribution)
2. Train a **sparse GP** to **predict target chemical property y(m)** given the latent representation
3. Use property predictor for BO

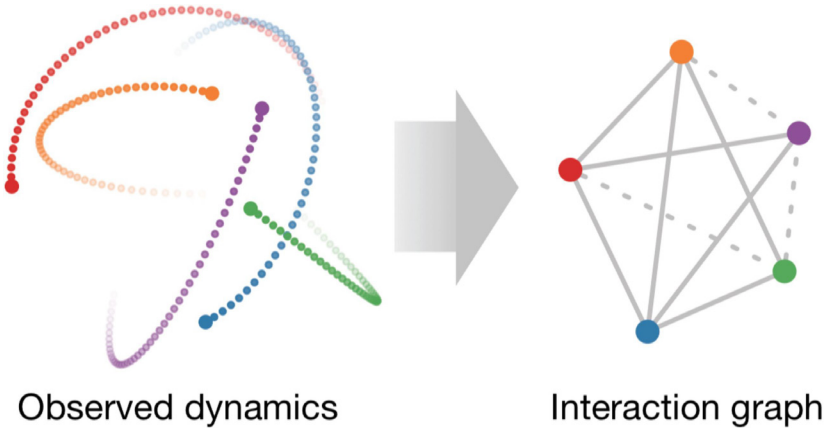
3 Research frontiers

Deep generative graph models

Latent graph inference

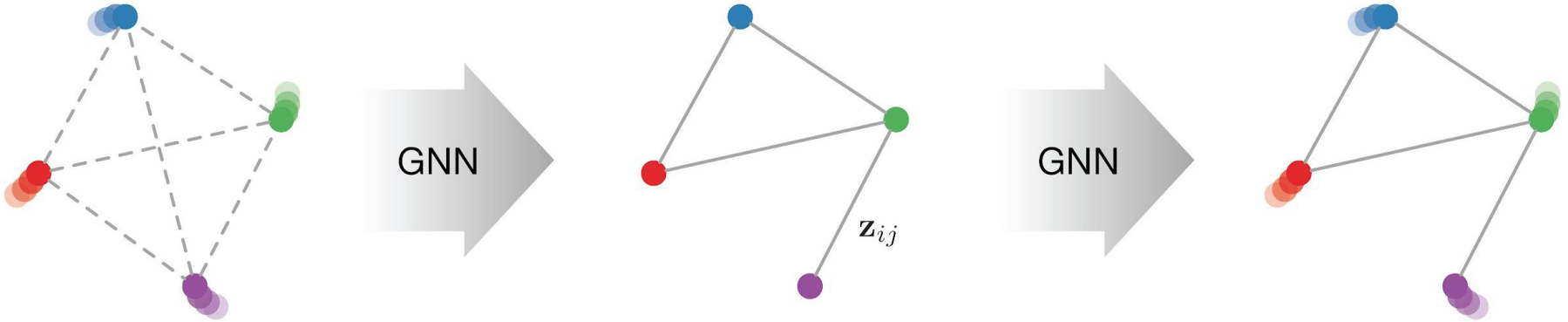
Modeling implicit/hidden structure

Neural Relational Inference for Interacting Systems, Kipf & Fetaya et al. (ICML 2018)



Neural Relational Inference with GNNs

Neural Relational Inference for Interacting Systems, Kipf & Fetaya et al. (ICML 2018)



Observe dynamics

$\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^T)$

Interaction graph proposal

$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$

Discrete (Gumbel softmax trick)
[Jang et al., 2016, Maddison et al., 2016]

Reconstruct dynamics*

$p_\theta(\mathbf{x}|\mathbf{z})$

VAE Objective (ELBO) $\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]$

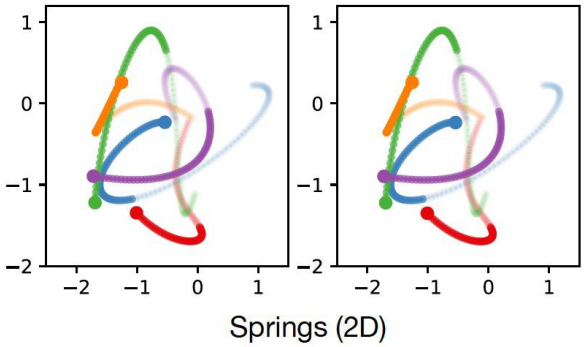
Reconstruction **Regularization**

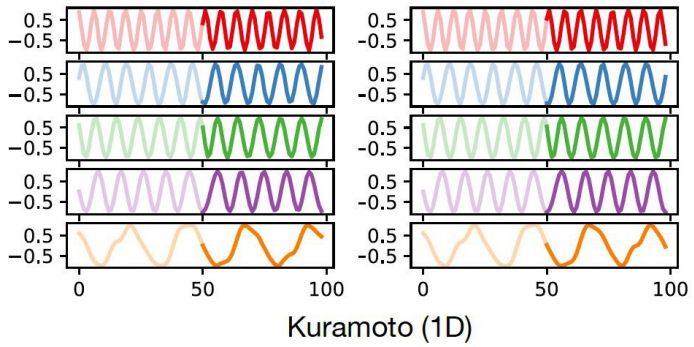
NRI evaluation - toy data

Neural Relational Inference for Interacting Systems, Kipf & Fetaya et al. (ICML 2018)

Table 6.1: Accuracy (in %) of unsupervised interaction recovery.

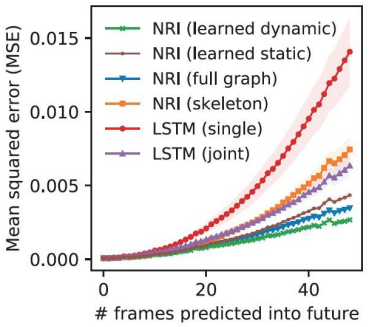
	Springs		**Kuramoto**	
Number of objects	5	10	5	10
Correlation (path)	52.4±0.0	50.4±0.0	62.8±0.0	59.3±0.0
Correlation (LSTM)	52.7±0.9	54.9±1.0	54.4±0.5	56.2±0.7
NRI (simulation decoder)	**99.8**±0.0	**98.2**±0.0	–	–
NRI (learned decoder)	**99.9**±0.0	**98.4**±0.0	**96.0**±0.1	**75.7**±0.3
Supervised	99.9±0.0	98.8±0.0	99.7±0.0	97.1±0.1



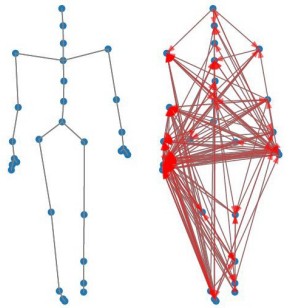


NRI evaluation - CMU Motion Capture (e.g. walking)

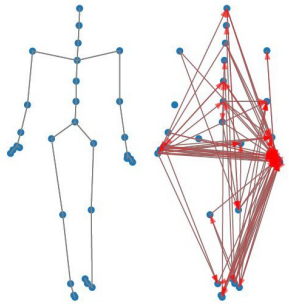
Neural Relational Inference for Interacting Systems, Kipf & Fetaya et al. (ICML 2018)



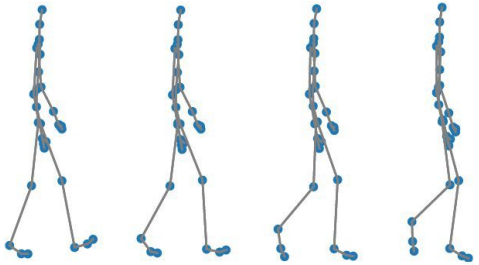
(a) Test MSE comparison



(b) Latent graph (left step)



(c) Latent graph (right step)



(c) Motion capture data

NRI applications - causal discovery

Amortized Causal Discovery: Learning to Infer Causal Graphs from Time-Series Data, Lowe et al. 2020

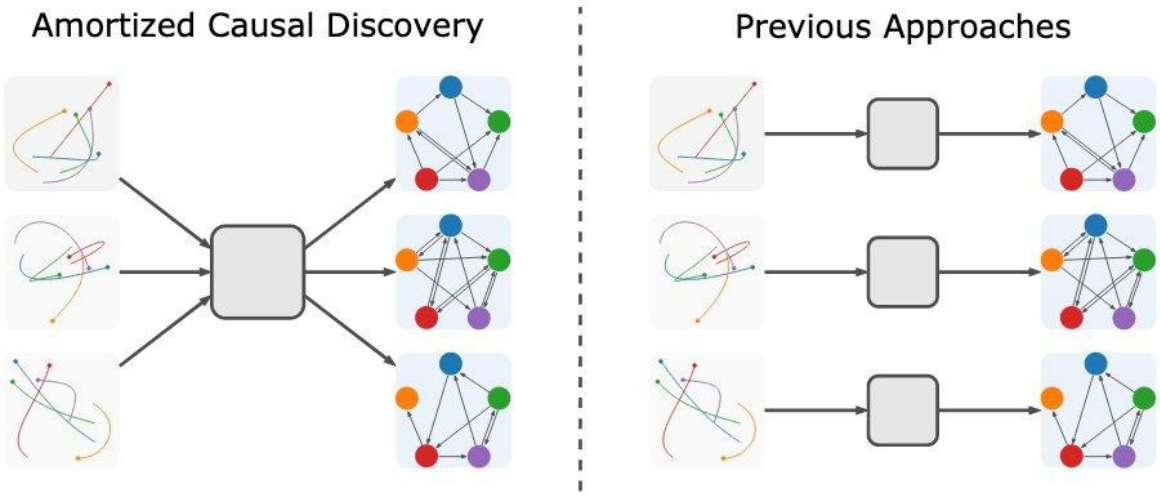


Figure 1: Amortized Causal Discovery. We propose to train a single model that infers causal relations across samples with different underlying causal graphs but shared dynamics. This allows us to generalize across samples and to improve our performance with additional training data. In contrast, previous approaches fit a new model for every sample with a different underlying causal graph.

Challenges and future work in graph neural nets

- **Problems of neighborhood aggregation / message passing**
  - Theoretical relation to WL isomorphism, simple graph convolutions; tree-structured computation graphs → **bounded power**
  - **Oversmoothing** (residual/gated updates help, but don't solve)
  - See recent work from Max Welling e.g. Natural Graph Networks
- **Scalable, stable generative models**
- Learning on **large, evolving data**
- (Mostly) **assume a graph structure is provided** as input
  - Neural Relational Inference is a preliminary work here, also see Pointer Graph Networks (Velickovic et al., NeurIPS 2020)
- **Multi-modal and cross-modal learning** (e.g. sequence2graph)

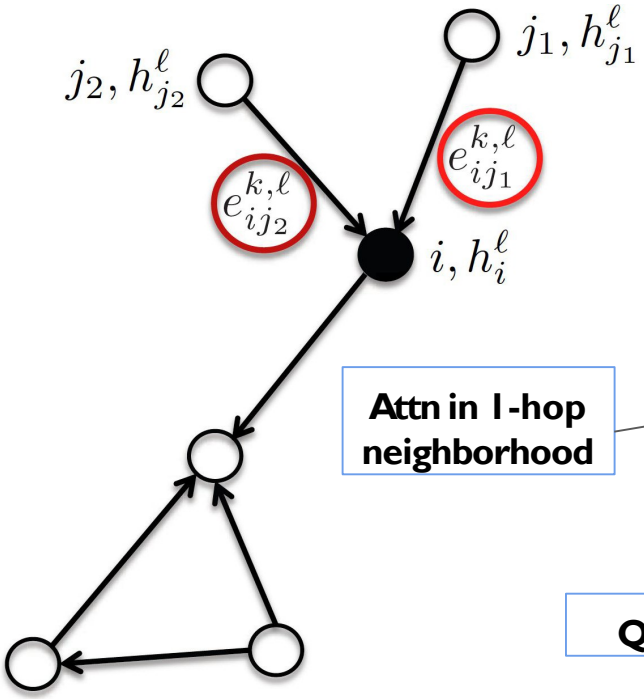
Recurrent Neural Networks (RNNs) + Generalization

1. **How do you read/listen/understand/write? Can machines do that?**
   - Context matters: characters, words, letters, sounds, completion, multi-modal
   - Predicting next word/image: from unsupervised learning to supervised learning
2. **Encoding temporal context: Hidden Markov Models (HMMs), RNNs**
   - Primitives: hidden state, memory of previous experiences, limitations of HMMs
   - RNN architectures,unrolling,back-propagation-through-time(BPTT),param reuse
3. **Vanishing gradients, Long-Short-Term Memory (LSTM), initialization**
   - Key idea: gated input/output/memory nodes, model choose to forget/remember
   - Example: online character recognition with LSTM recurrent neural network
4. **Transformer modules**
   - Learning temporal relationships without unrolling and without RNNs
   - Encoder/Decoder output architecture and multi-head attention modeule
5. **Graph Neural Networks**
   - Applications: social, brain, chemical drug design, graphics, transport, knowledge
   - Define each node's computation graph, from its neighborhood
   - Classical network/graph problems: Node/graph classification, link prediction
   - Research frontiers: deep generative models, latent graph inferences

Appendix

Graph Transformers (Li et al. 2018)

A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, A Gomez, L Kaiser, I Polosukhin, Attention is all you need (2017)



Value

scalar

$$h^{l+1}_{i\ d\times 1} = W^{l}_{d\times d}\ \mathrm{Concat}^{K}_{k=1}\Big(\sum_{j\in\mathcal{N}_i} e^{k,l}_{ij}\ V^{k,l}_{\frac{d}{K}\times d} h^{l}_{j}\Big)$$

$$e^{k,l}_{ij} = \mathrm{Softmax}_{\mathcal{N}_i}(\hat{e}^{k,l}_{ij}) = \frac{\exp(\hat{e}^{k,l}_{ij})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}^{k,l}_{ij'})}$$

$$\hat{e}^{k,l}_{ij} = (Q^{k,l}_{1\times d} h^{l}_{i})^{T}\,(K^{k,l}_{d\times 1} h^{l}_{j})$$

Attn in 1-hop neighborhood

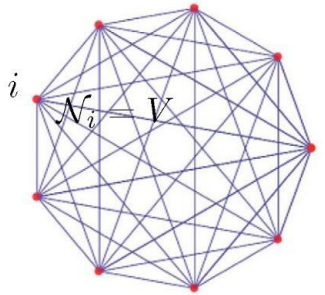
Query

Key

Graph Transformers (Li et al. 2018)

A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, A Gomez, L Kaiser, I Polosukhin, Attention is all you need (2017)

- We can frame transformers as a special case of GCNs when the **graph is fully connected.**
- The neighborhood $\mathcal{N}_i$ is the **whole graph**.



Fully connected graph

$$h_i^{\ell+1} = W^\ell \ \text{Concat}_{k=1}^{K}\Big(\sum_{j \in \mathcal{N}_i} e_{ij}^{k,\ell} \ V^{k,\ell} \ h_j^\ell \Big)$$

$$e_{ij}^{k,\ell} = \text{Softmax}_{\mathcal{N}_i}(\hat{e}_{ij}^{k,\ell}) = \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j' \in \mathcal{N}_i} \exp(\hat{e}_{ij'}^{k,\ell})}$$

$$\hat{e}_{ij}^{k,\ell} = (Q^{\ell,k} h_i^\ell)^T (K^{\ell,k} h_j^\ell)$$

$$\mathcal{N}_i = V \Longrightarrow$$

$$h^{\ell+1} = \text{Concat}_{k=1}^{K}\Big(\text{Softmax}(Q^\ell {K^\ell}^T) V^\ell\Big) W^\ell$$

$h^{\ell+1}$: $n \times d$; W^ℓ: $d \times d$; $\text{Softmax}(Q^\ell {K^\ell}^T)$: $n \times n$ ($n \times \frac{d}{K}$ times $\frac{d}{K} \times n$); V^ℓ: $n \times \frac{d}{K}$

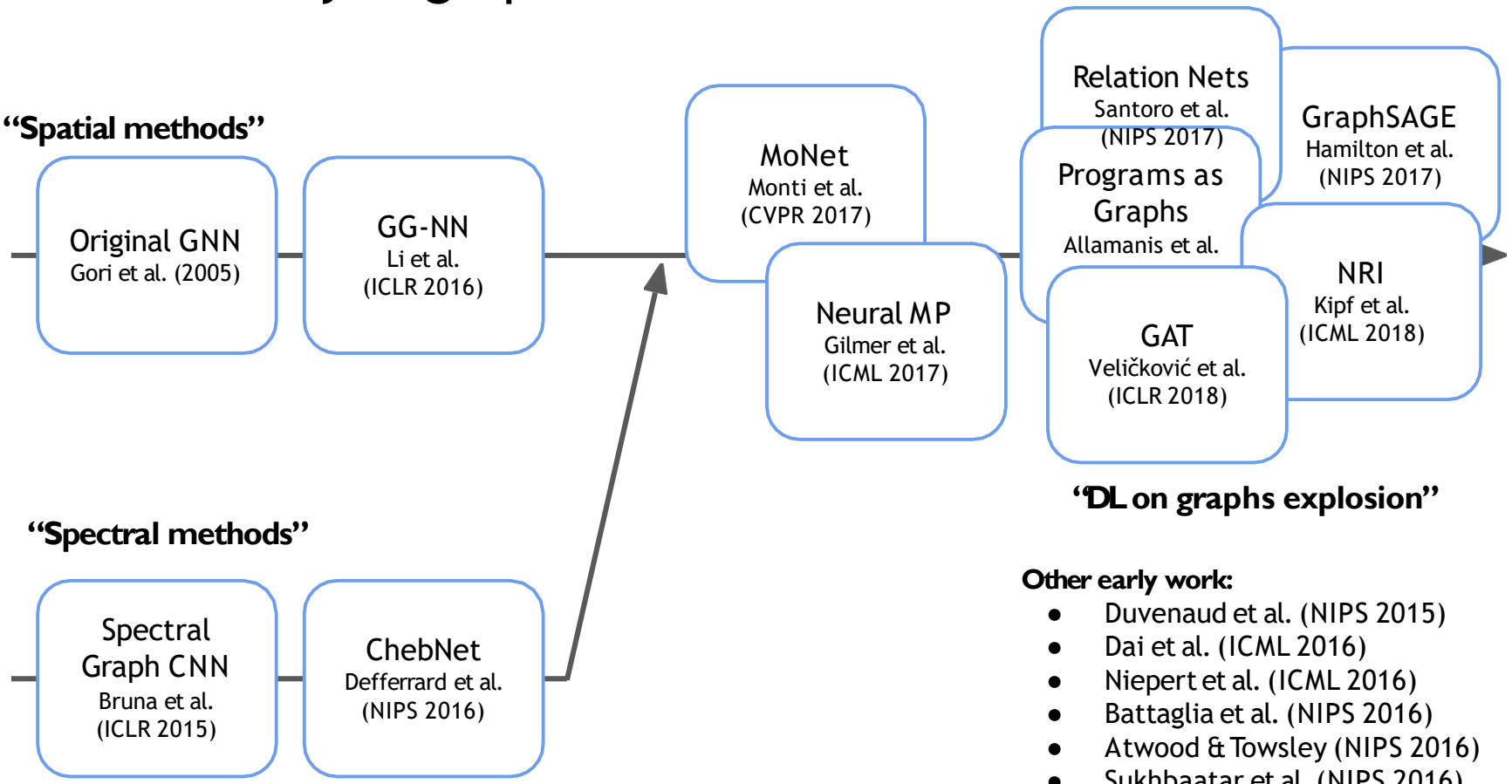
$$Q^\ell = h^\ell \ W_Q^\ell$$

$$K^\ell = h^\ell \ W_K^\ell$$

$$V^\ell = h^\ell \ W_V^\ell$$

Q^ℓ: $n \times \frac{d}{K}$; h^ℓ: $n \times d$; W_Q^ℓ: $d \times \frac{d}{K}$

A brief history of graph neural nets



Other early work:

- Duvenaud et al. (NIPS 2015)
- Dai et al. (ICML 2016)
- Niepert et al. (ICML 2016)
- Battaglia et al. (NIPS 2016)
- Atwood & Towsley (NIPS 2016)
- Sukhbaatar et al. (NIPS 2016)

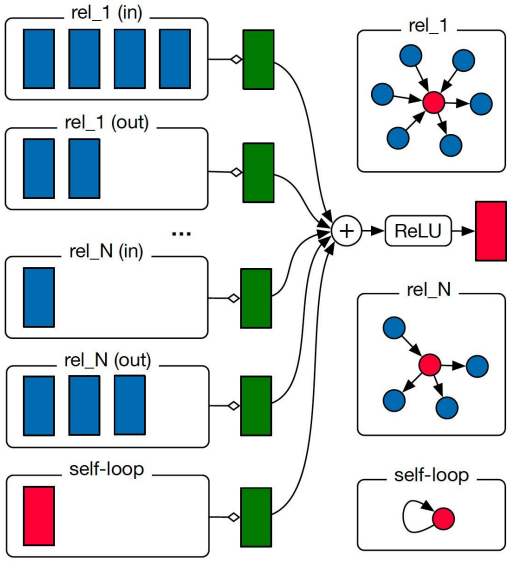
MoNet & Relational GCN for modeling (multi-)relational data

Monti et al. (CVPR 2017), Schlichtkrull & Kipf et al. (ESWC 2018)

$$\mathbf{h}_i' = \sigma\left(\sum_{r=1}^{R}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^r \mathbf{W}_r \mathbf{h}_j\right)$$

α_{ij}^r **based on:**

- Edge type (Relational GCN)
- Auxiliary features (MoNet), e.g. node degree



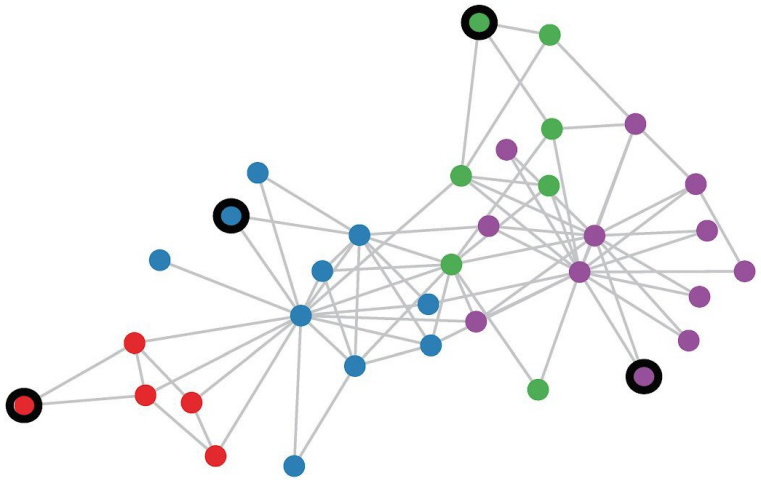
Relational GCN update rule

Semi-supervised classification on graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:
Predict node label of unlabeled nodes



Evaluate loss on labeled nodes only:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

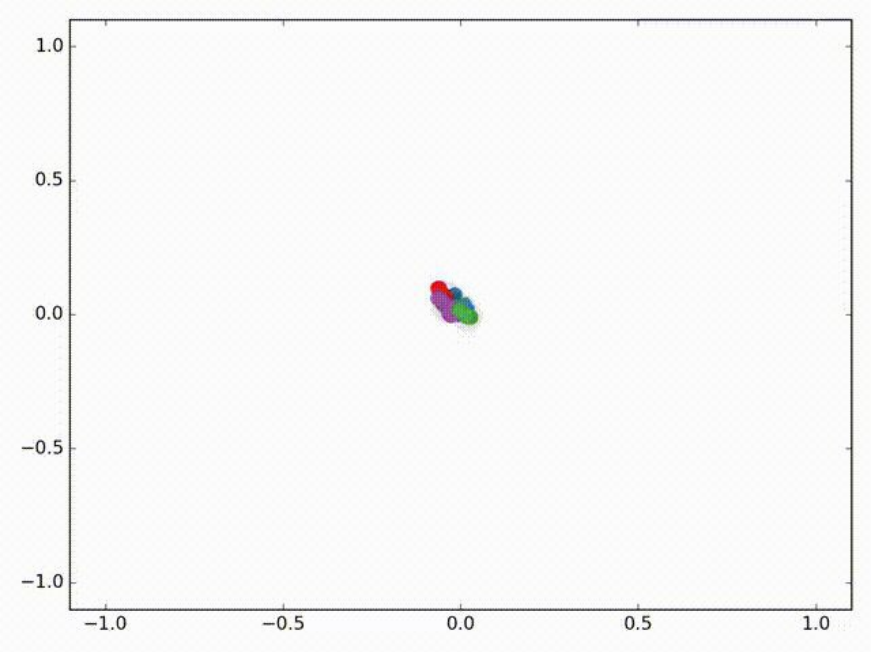
$\mathcal{Y}_L$ set of labeled node indices

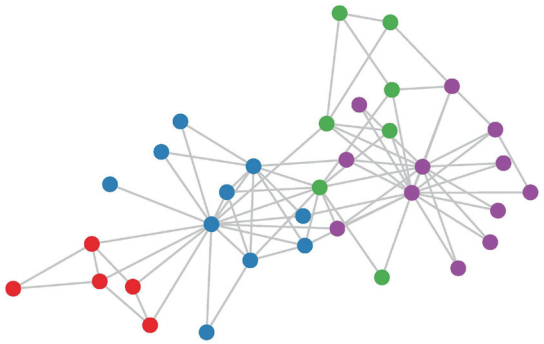
$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

Toy example (semi-supervised learning)

from tkipf.github.io/graph-convolutional-networks





Latent space dynamics for 300 training iterations. Labeled nodes are highlighted.

GCN model manages to linearly separate classes with only **1 training example per class, no node features**!

Tree decoding

Junction Tree Variational Autoencoder, Jin et al. (ICML 2018); Tree-Structured Decoding, Alvarez-Melis & Jaakkola (ICLR 2017)

Algorithm 1 Tree decoding at sampling time

Require: Latent representation $\mathbf{z}_{\mathcal{T}}$

Initialize: Tree $\widehat{\mathcal{T}} \leftarrow \emptyset$
function SampleTree(i, t)
  Set $\mathcal{X}_i \leftarrow$ all cluster labels that are chemically compatible with node i and its current neighbors.
  Set $d_t \leftarrow expand$ with probability p_t. ▷ Eq.(11)
  if $d_t = expand$ **and** $\mathcal{X}_i \neq \emptyset$ **then**
    Create a node j and add it to tree $\widehat{\mathcal{T}}$.
    Sample the label of node j from $\mathcal{X}_i$ ▷. Eq.(12)
    SampleTree($j, t+1$)
  end if
end function

Topological Prediction

$$p_t = \sigma(\mathbf{u}^d \cdot \tau(\mathbf{W}_1^d \mathbf{x}_{i_t} + \mathbf{W}_2^d \mathbf{z}_{\mathcal{T}} + \mathbf{W}_3^d \sum_{(k,i_t)\in\tilde{\mathcal{E}}_t} \mathbf{h}_{k,i_t}) \quad (11)$$

Label Prediction

$$\mathbf{q}_j = \text{softmax}(\mathbf{U}^l \tau(\mathbf{W}_1^l \mathbf{z}_{\mathcal{T}} + \mathbf{W}_2^l \mathbf{h}_{ij})) \quad (12)$$

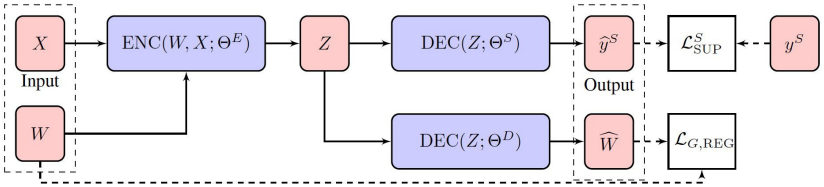
Training

$$\mathcal{L}_c(\mathcal{T}) = \sum\nolimits_t \mathcal{L}^d(p_t, \hat{p}_t) + \sum\nolimits_j \mathcal{L}^l(\mathbf{q}_j, \hat{\mathbf{q}}_j) \quad (13)$$

+ Teacher forcing -- replace topological and label predictions with ground truth at train time

Generalizing the space of ML approaches on graphs

Machine Learning on Graphs: A Model and Comprehensive Taxonomy (Chami et al., preprint)



GraphEDM model