CS5824: Advanced Machine Learning

Dawei Zhou

CS, Virginia Tech

Please keep your face covering on!

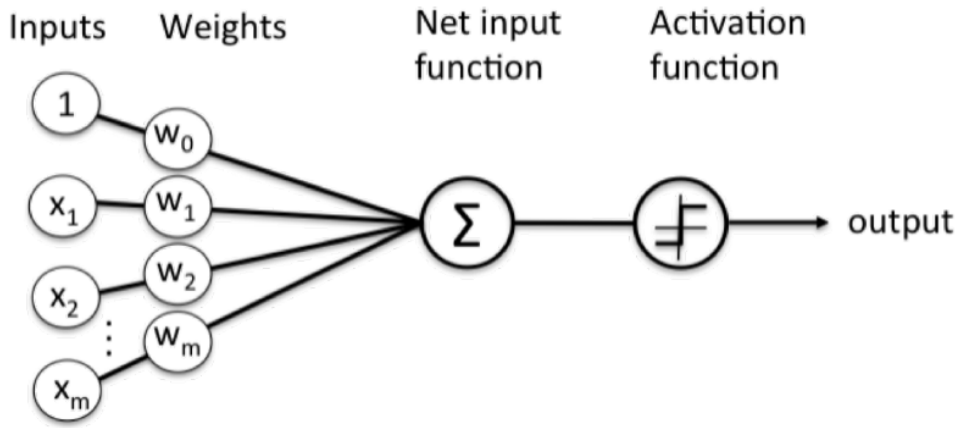
COMPUTER SCIENCE
VIRGINIA TECH

Roadmap

- Deep Learning: Basic Techniques
- Techniques to Improve Deep Learning Training
- Convolutional Neural Networks
- Recurrent Neural Networks
- Graph Neural Networks
- Summary

Deep Learning: Basic Techniques

Perceptron: Predecessor of a Neural Network



$$\hat{y} = f(\boldsymbol{W}^T \boldsymbol{x}) = f(\sum W_i x_i + b)$$

Activation Function
Adding non-linearity to the model

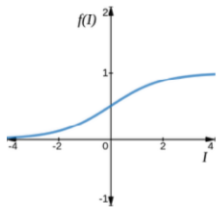
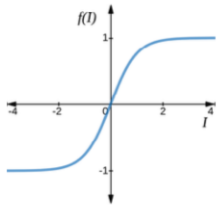
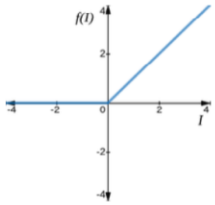
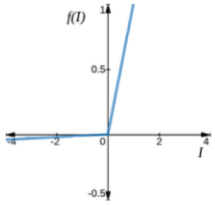
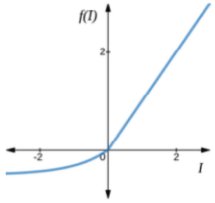
Weights

Bias
A measure of how easy it is to get the perceptron to output a 1

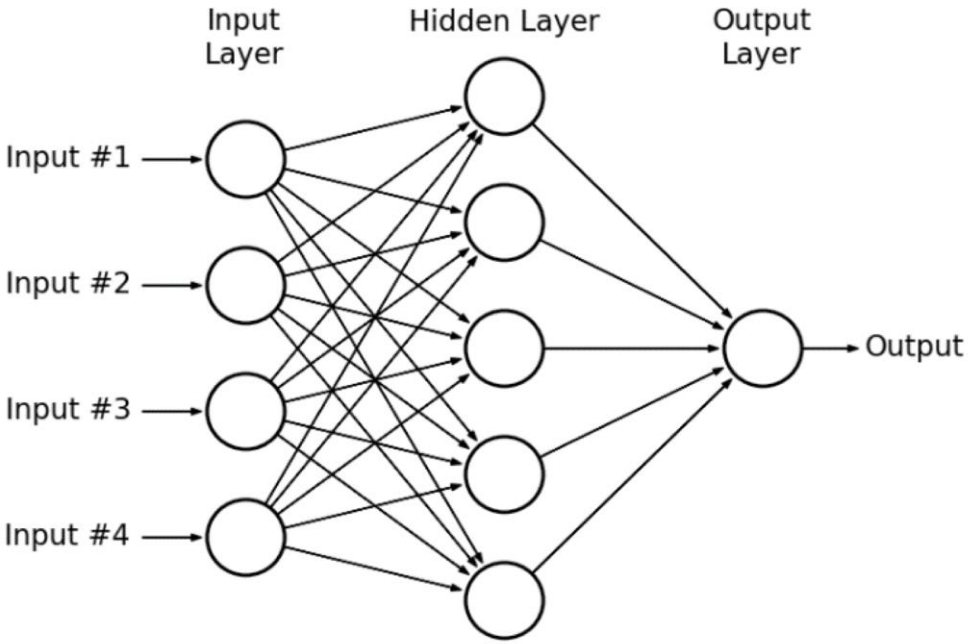
- Computes a weighted sum of inputs
- Invented in 1957 by Frank Rosenblatt. The original perceptron model does not have a non-linear activation function

Perceptron: Predecessor of a Neural Network

- Examples of activation functions

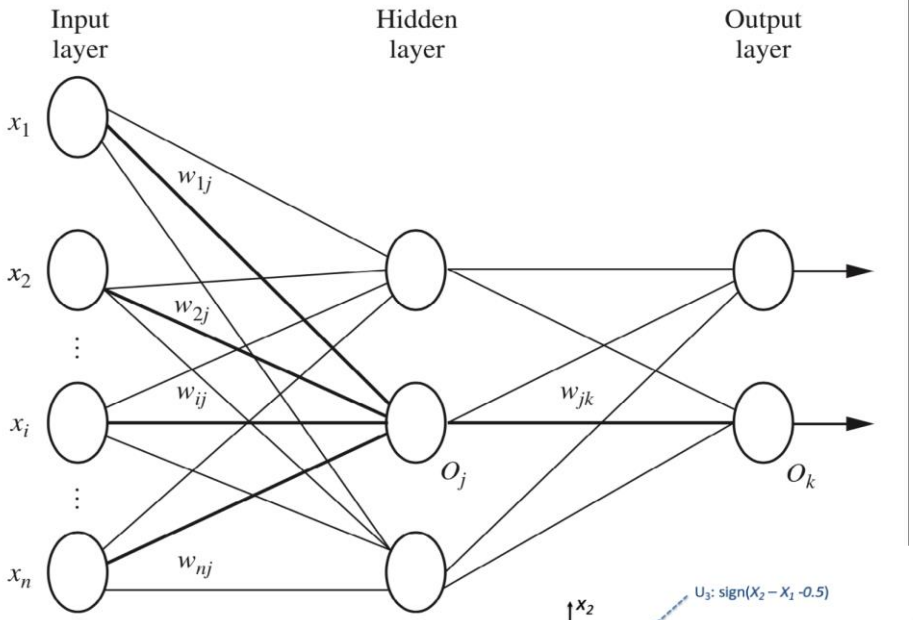
Sigmoid	$\frac{1}{1+e^{-I}}$	
Tanh	$\frac{e^{I}-e^{-I}}{e^{I}+e^{-I}}$	
ReLU	$\begin{cases} 0, I \leq 0 \\ I, I > 0 \end{cases}$	
Leaky ReLU	$\begin{cases} 0.01 \times I, I < 0 \\ I, I \geq 0 \end{cases}$	
ELU	$\begin{cases} \alpha(e^{I}-1), I \leq 0 \\ I, I > 0 \end{cases}$	

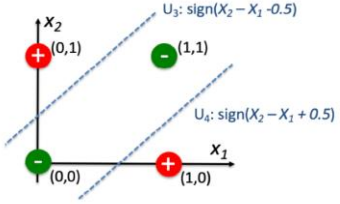
Multilayer Perceptron (MLP)



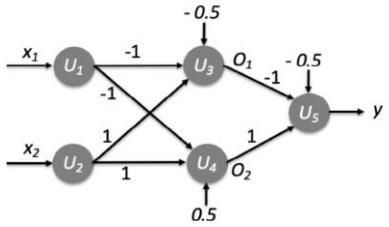
- Stacking multiple layers of perceptrons (adding hidden layers) makes a multilayer perceptron (MLP)
- MLP can engage in sophisticated decision making, where perceptrons fail
  - E.g. XOR problem
- Try it: http://playground.tensorflow.org

Feed-forward Neural Networks

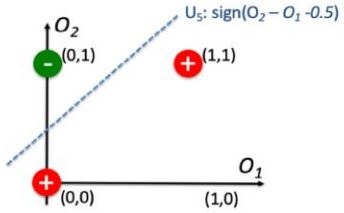




(a) Input 4 tuples



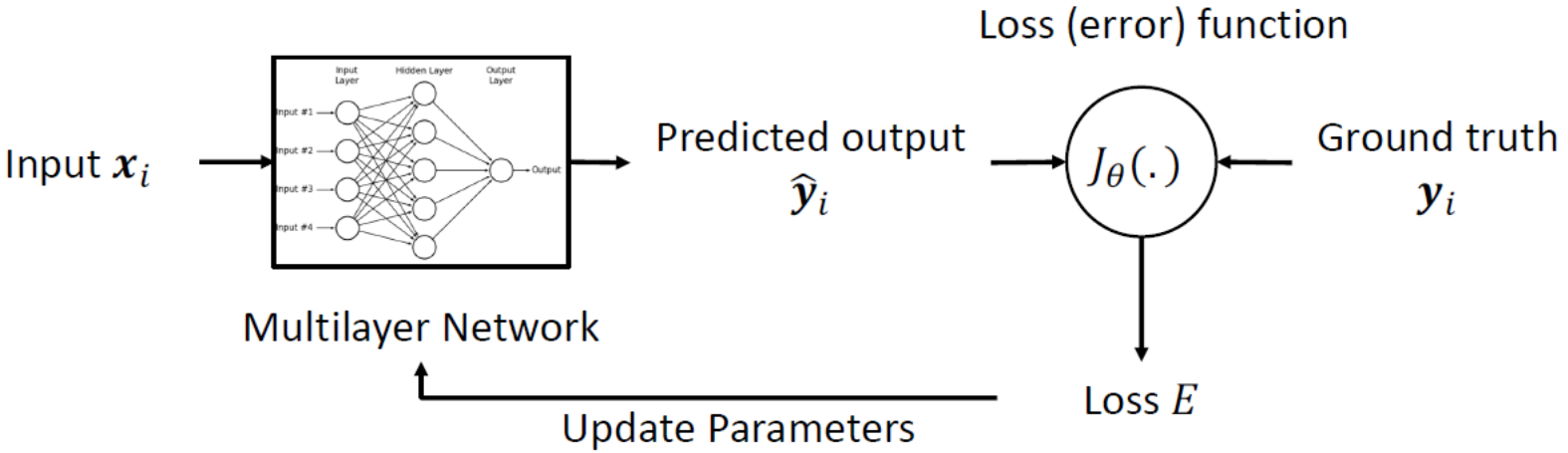
(b) MLP



(c) Outputs of two hidden units

- Why multiple layers
  - Automatic feature/representation learning
  - Learn complicate (nonlinear) mapping function

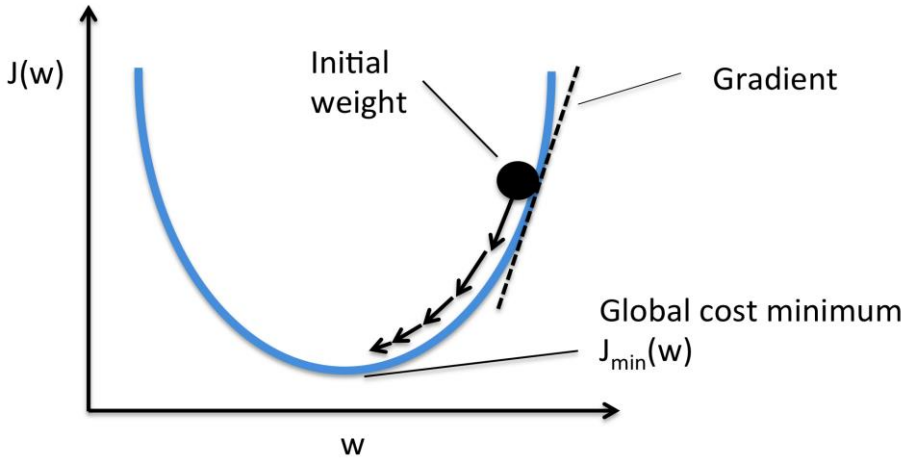
Learning NN Parameters



- **Gradient Descent Algorithm**
- **Input**: Training sample x_i and its label y_i
  - **Feed Forward**: Get prediction $\hat{y}_i = MLP(x_i)$, and loss $E = J(\hat{y}_i, y_i)$
  - **Compute Gradient**: For each parameter θ_i (weights, bias), compute its gradient $\frac{\partial}{\partial \theta_i} J_\theta$
  - **Update Parameter**: $\theta_i = \theta_i - \eta \cdot \frac{\partial}{\partial \theta_i} J_\theta$

Explain later

Empirical Explanation of Gradient Descent

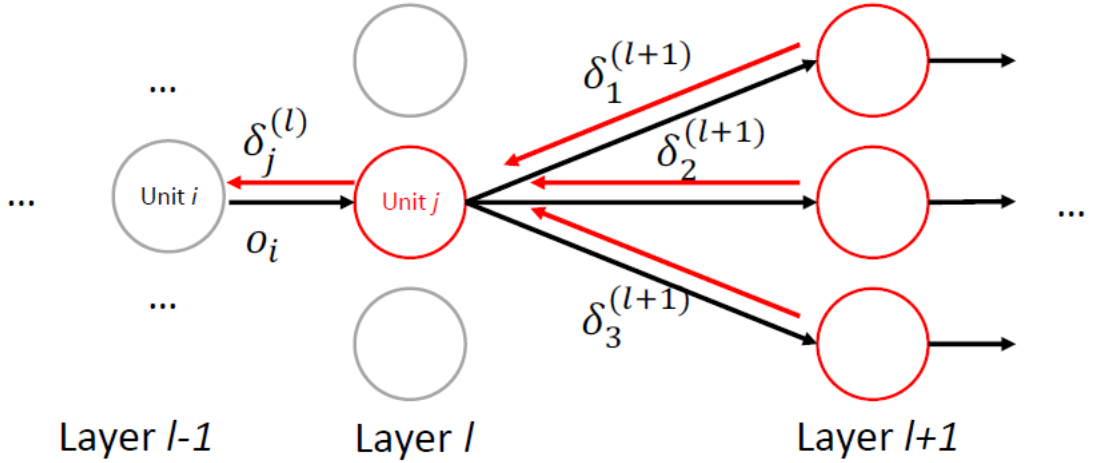


$$\theta_j = \theta_j - \eta \cdot \frac{\partial}{\partial \theta_j} J_\theta$$

η is the **learning rate**, which controls the 'step size' of the optimization

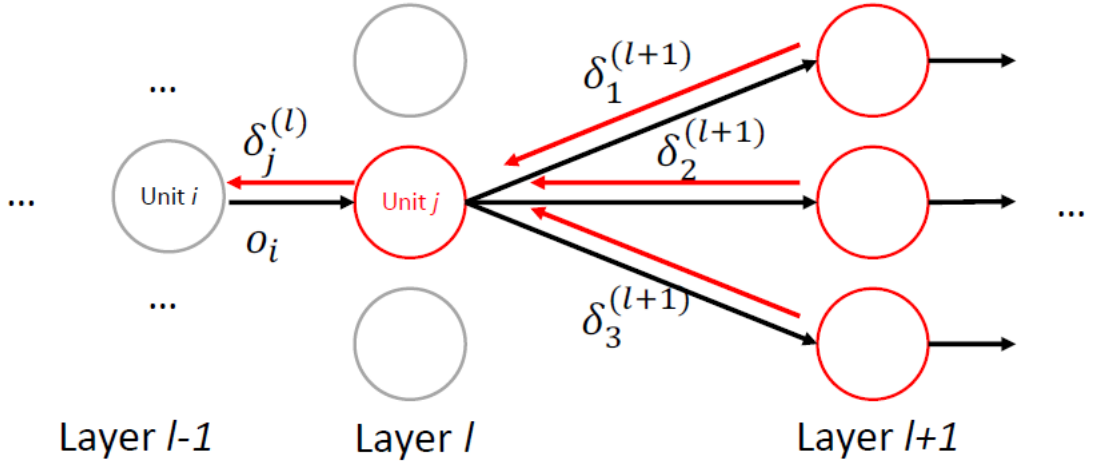
- Our objective is to minimize the loss function J, which is a function of the model parameters
- A gradient measures how much the output of a function changes if you change the inputs a little bit
- We update the parameters, based on their gradients, so that the loss function is going 'downhill'

Gradient Computation: Backpropagation



- The gradient of w_{ij} in the lth layer (corresponding to unit j in layer l, connected to unit i in layer $l-1$) is a function of
  - All 'error' terms from layer $l+1$ $\delta_k^{(l+1)}$ -- An auxiliary term for computation, not to be confused with gradients
  - Output from unit i in layer $l-1$ (input to unit j in layer l) -- Can be stored at the feed forward phase of computation

Gradient Computation: Backpropagation

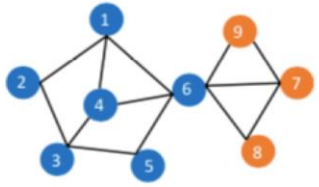
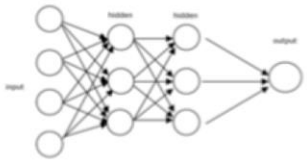
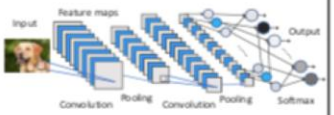
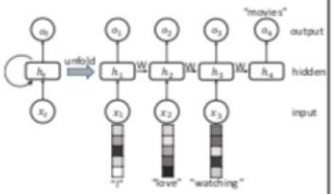
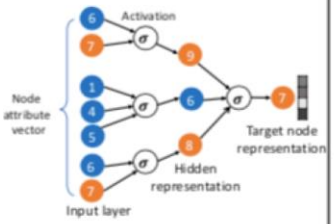


- The 'error' terms $\delta_j^{(l)}$ is a function of
  - All $\delta_k^{(l+1)}$ in the layer $l+1$, if layer l is a hidden layer
  - The overall loss function value, if layer l is the output layer
- We can compute the error at the output, and distribute backwards throughout the network's layers (backpropagation)

From Neural Networks to Deep Learning

- **Deep Learning** refers to training (deep) neural networks with many layers
  - More neurons, more layers
  - More complex ways to connect layers
- Deep Learning has some major advantages, making it popular
  - Tremendous improvement of performance in many tasks
    - Image recognition, natural language processing, AI game playing…
  - Requires no (or less) feature engineering, making end-to-end models possible
- Several factors lead to deep learning's success
  - Very large data sets
  - Massive amounts of computation power (GPU acceleration)
  - Advanced neural network structures and tricks
    - Convolutional neural networks, recurrent neural networks, graph convolution network …
    - Dropout, ReLU, residual connection, …

Overview of Typical Deep Learning Architectures

Data type	Multi-dimensional Features: credit rating, account balance $x = (4.5, 500, 3, 5)$ #deposits, #withdraws	Grid 	Sequence x = "I love watching movies."	Graph 
DL Architecture	Feed-forward Network 	CNN 	RNN 	GNN 

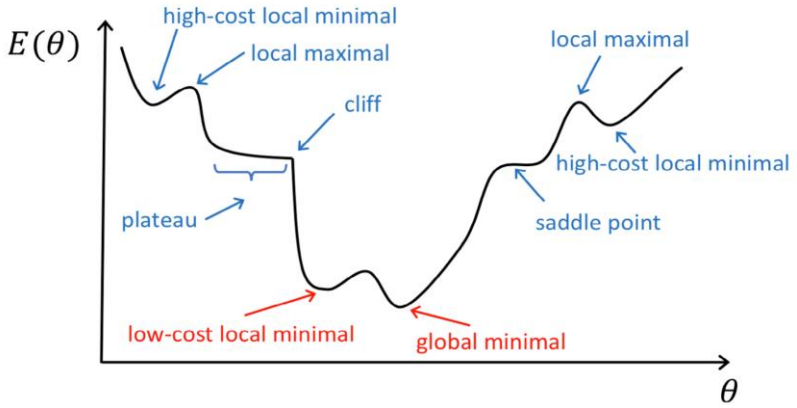
Techniques to Improve Deep Learning Training

Techniques to Improve Deep Learning Training

- Key Challenges for Training Deep Learning Models
- Responsive Activation Functions
- Adaptive Learning Rate
- Dropout
- Pretraining
- Cross Entropy

Key Challenges

- **Optimization Problem in Deep Learning**
  - Minimize the (approximated) training error E $E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{l=1}^{m} \mathrm{Loss}(\hat{T}(X^l, \boldsymbol{\theta}), T^l)$
  - (Stochastic) gradient descent to find the model parameter $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{g}_t$



- **Challenge 1: Optimization**
  - E is non-convex in general
  - How to find a high-quality local optima
- **Challenge 2: Generalization**
  - What we do: minimize (approximated) training error
  - What we really want: minimize the generalization error
  - How to mitigate over-fitting

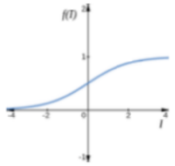
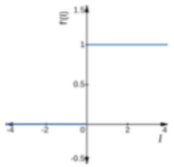
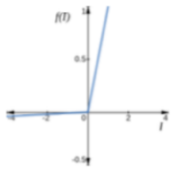
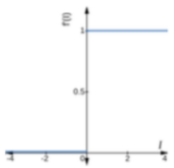
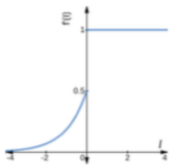
Responsive Activation Function

- **Saturation of Sigmoid Activation Function**
  - The output $O = \sigma(I) = \frac{1}{1+e^{-I}} \in (0,1)$
  - The derivative $\frac{\partial O}{\partial I} = O(1-O)$
  - The error of an output unit $\delta_j = O_j(1-O_j)(T_j - O_j)$

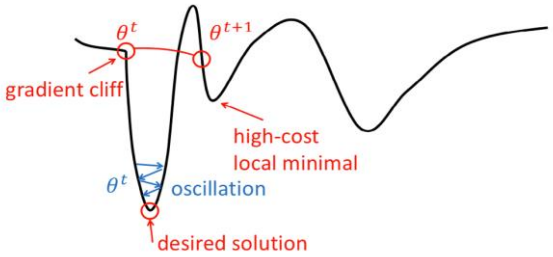
decaying

- **Saturation of Sigmoid Activation Function**
  - If $O_j \approx 1 \; or \; O_j \approx 0$, both derivative and error will be close to 0
  - Further exacerbated due to backpropagation → gradient vanishing → B.P. is stuck or takes long time to terminate
- **A More Responsive Activation Function: Rectified Linear Unit (RELU)**
  - The output $O = f(I) = I$ if $I > 0$, $O = 0$ otherwise
  - The gradient: $\frac{\partial O}{\partial I} = 1 \; if \; I > 0 \; and \; \frac{\partial O}{\partial I} = 0$
  - The error: 0 if the unit is inactive ($I < 0$), otherwise, aggregate all error terms from the units in the next higher layer the unit is connected to,
  - w/o decaying → avoid gradient vanishing

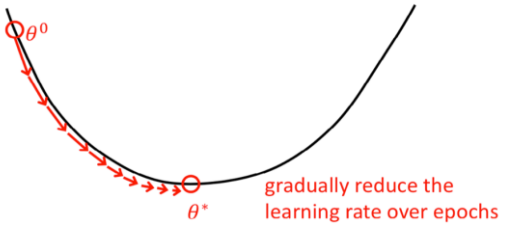
Activation Functions

Name	Definition ($f(I)$)	Plot	Derivative of $f(I)$	Plot
Sigmoid	$\frac{1}{1+e^{-I}}$		$f(I)(1-f(I))$	
Tanh	$\frac{e^{I}-e^{-I}}{e^{I}+e^{-I}}$		$1-f(I)^2$	
ReLU	$\begin{cases} 0, I \leq 0 \\ I, I > 0 \end{cases}$		$\begin{cases} 0, I \leq 0 \\ 1, I \geq 0 \end{cases}$	
Leaky ReLU	$\begin{cases} 0.01 \times I, I < 0 \\ I, I \geq 0 \end{cases}$		$\begin{cases} 0.01, I < 0 \\ 1, I \geq 0 \end{cases}$	
ELU	$\begin{cases} \alpha(e^{I}-1), I \leq 0 \\ I, I > 0 \end{cases}$		$\begin{cases} \alpha e^{I}, I \leq 0 \\ 1, I > 0 \end{cases}$	

Adaptive Learning Rate



- **Stochastic gradient descent to find a local optima**
  - Default choice for η: a fixed, small positive constant $\theta_{t+1} = \theta_t - \eta g_t$
  - Problems: slow progress, or jump over 'gradient cliff', or oscillation
- **Adaptive learning rate: let η change over epoch**

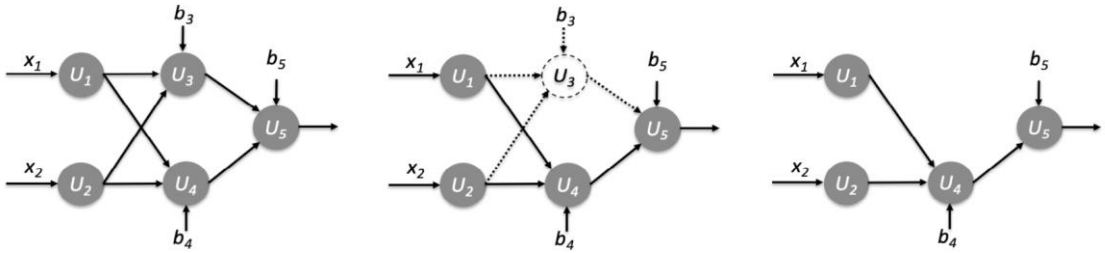


  - Strategy #1: $\eta_t = \frac{1}{t}\eta_0$
  - Strategy #2: $\left(1 - \frac{t}{T}\right)\eta_0 + \frac{t}{T}\eta_\infty \; if \; t \leq T \; and \; \eta_t = \eta_\infty$
    - Intuitions: smaller adjustment as algorithm progresses
    - e.g., $\eta_0 = 0.9$; $\eta_\infty = 10^{-9}$
  - Strategy #3 (AdaGrad): $\eta_t = \frac{1}{\rho + r_i}\eta_0 \quad r_i = \sqrt{\sum_{k=1}^{t-1} g_{i,k}^2}$
    - Intuition: The magnitude of gradient g_t: indicator of the overall progress
    - e.g., $\rho = 10^{-8}$
  - Strategy #4 (RMSProp): exponential decaying weighted sum of squared historical gradients

Dropout

- **The Purpose of Dropout: to Prevent Overfitting**
- **How does it work?**
  - At each epoch (ρ dropout rate, e.g., $\rho = 0.5$)
    - randomly dropout some non-output units
    - Perform B.P. on the dropout network
  - Scale the final model parameters

$$\boldsymbol{\theta}^* \leftarrow \rho \cdot \boldsymbol{\theta}^*$$

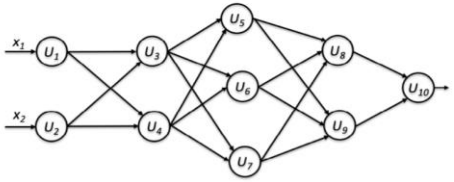


(a) Original network (b) Dropout U_3 (c) Dropout network

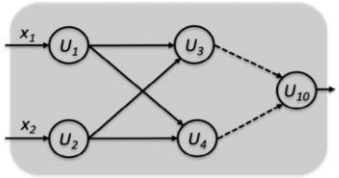
- **Why does Dropout Work?**
  - Dropout can be viewed as a regularization technique
    - Force the model to be more robust to noise, and to learn more generalizable features
  - Dropout vs. Bagging
    - Bagging: Each base model is trained *independently* on a bootstrap sample
    - Dropout: the model parameters of the current dropout network are updated based on that of the previous dropout network(s)

Pretraining

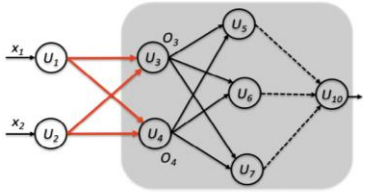
- **Pretraining**: the process of initializing the model in a suitable region
- **Greedy supervised pretraining**
  - pre-set the model parameters layer-by-layer in a greedy way
  - Start with a simple model, add one additional layer at a time,
  - pretrain the parameters of the newly added layer, while fixing for other layers
  - Each time, equivalent to training a two-layered network
  - Followed up a fine—tuning process
- **Other Pretraining Strategies**
  - Unsupervised pretraining: based on auto-encoder
  - Hybrid strategy
- **Pretraining for transfer learning**



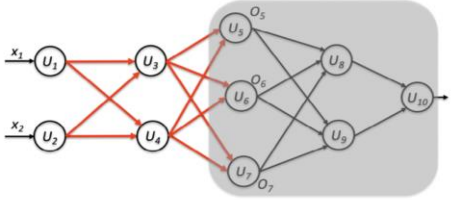
(a) original model to pretrain



(b) first iteration (1 hidden layer)



(c) second iteration (2 hidden layers)

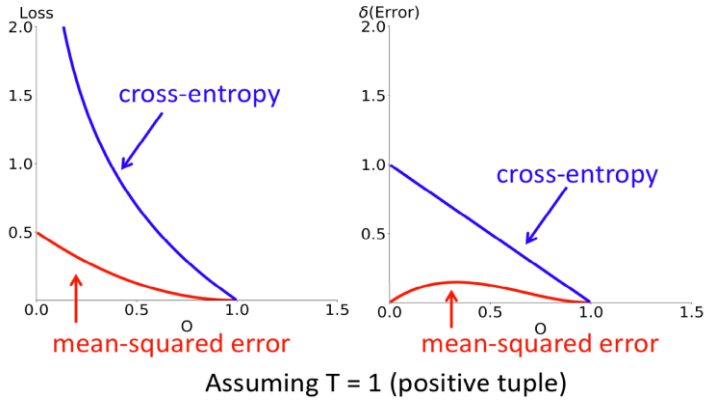


(d) third iteration (3 hidden layers)

Cross Entropy

- **Measure the disagreement between the actual (T) and predicted (O) target values**
  - For regression: mean-squared error
  - For (binary) classification: cross-entropy
- **Mean-squared error vs. Cross-entropy**

	Mean-squared error	Cross entropy
Loss	$\frac{1}{2}(T-O)^2$	$-T\log O-(1-T)\log(1-O)$
Error δ	$O(1-O)(T-O)$	$T-O$



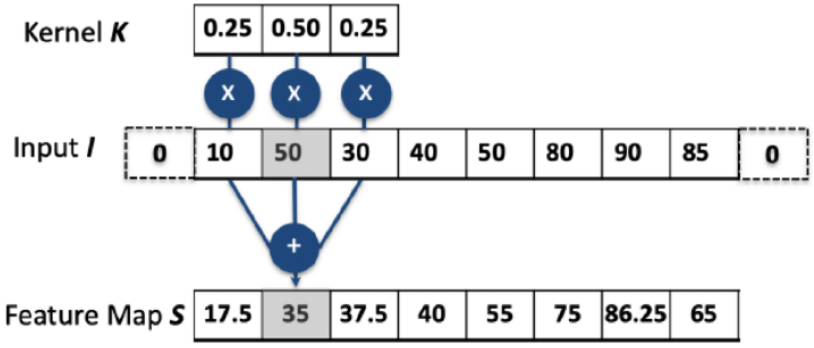
- **Cross-entropy for Multiclass Problem**
  - Actual target $T=(T_1,T_2,...,T_C)$
  - Predicted output $O=(O_1,O_2,...,O_C)$

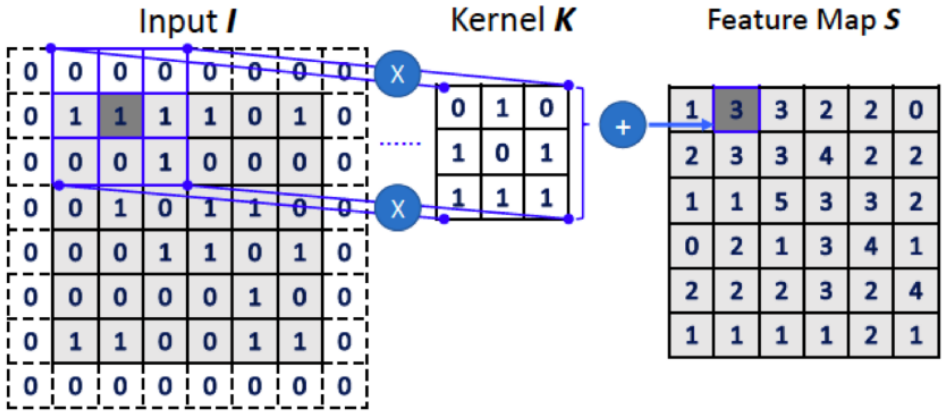
$$\text{Loss}(T,O)=-\sum_{j=1}^{C}T_j\log O_j$$

Convolutional Neural Networks

Convolutional Neural Networks (CNN)

- What is convolution? $\boldsymbol{S} = \boldsymbol{I} * \boldsymbol{K}$





1D Convolution

$$\boldsymbol{S}[i] = \sum_{p=-\frac{P-1}{2}}^{\frac{P-1}{2}} \boldsymbol{I}[i+p]\boldsymbol{K}[p], \quad (i = 1, ..., N)$$

2D Convolution

$$\boldsymbol{S}[i][j] = \sum_{q=-\frac{Q-1}{2}}^{\frac{Q-1}{2}} \sum_{p=-\frac{P-1}{2}}^{\frac{P-1}{2}} \boldsymbol{I}[i+p][j+q]\boldsymbol{K}[p][q]$$

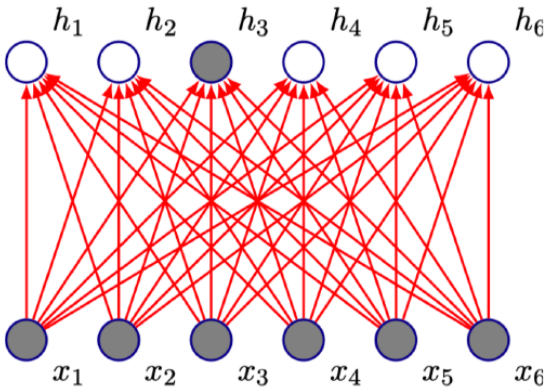
- The outputs are computed by sliding a **kernel** (of weights) on the inputs, and computing weighted sum locally

CNN Motivation

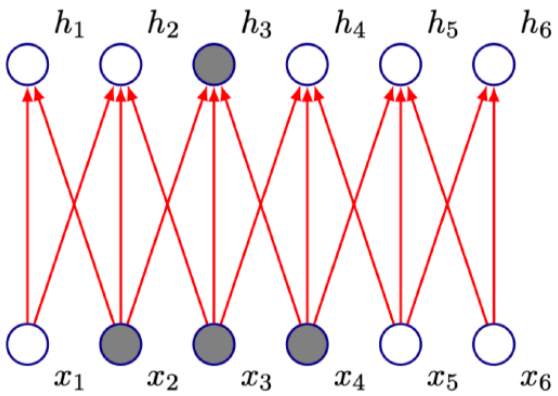
- Why not deep MLP (or feed-forward neural network)?
  - Deep multilayer perceptrons are **computationally expensive**, and **hard to train**.
    - Long training time, slow convergence, local minima
- Motivations of convolution
  - Sparse interactions
  - Parameter sharing
  - Translational equivalence
- The properties of CNNs are well aligned with properties of many forms of data (e.g. images, text), making them very successful

CNN Motivation

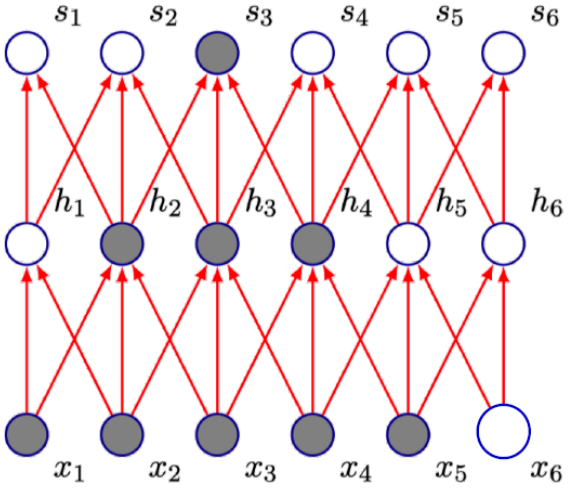
- Motivations of convolution
  - Sparse interactions
    - e.g. 1D convolution with kernel size 3
    - Units in deeper layers still connect to a wide range of inputs



MLP



CNN



CNN Motivation

- Motivations of convolution
  - Parameter sharing
    - Each kernel is used on all locations of input
    - Reduce #parameters
- Translational equivalence $f(g(x)) = g(f(x))$
  - Same input at different location gives the same output
  - E.g., a cat at the upper right corner and at the lower-left corner of an image, will produce the same outputs
  - E.g., “Hokie Spa” at the start of the sentence and at the end of the sentence produces the same outputs

CNN: Pooling Layer

- Pooling (Subsampling)
  - Pool hidden units in the same neighborhood
    - Introduces invariance to local translations
    - Reduces the number of hidden units in hidden layer

1	3	2	4
3	9	2	4
1	3	3	2
4	2	2	5

(a) Feature map

9	4
4	5

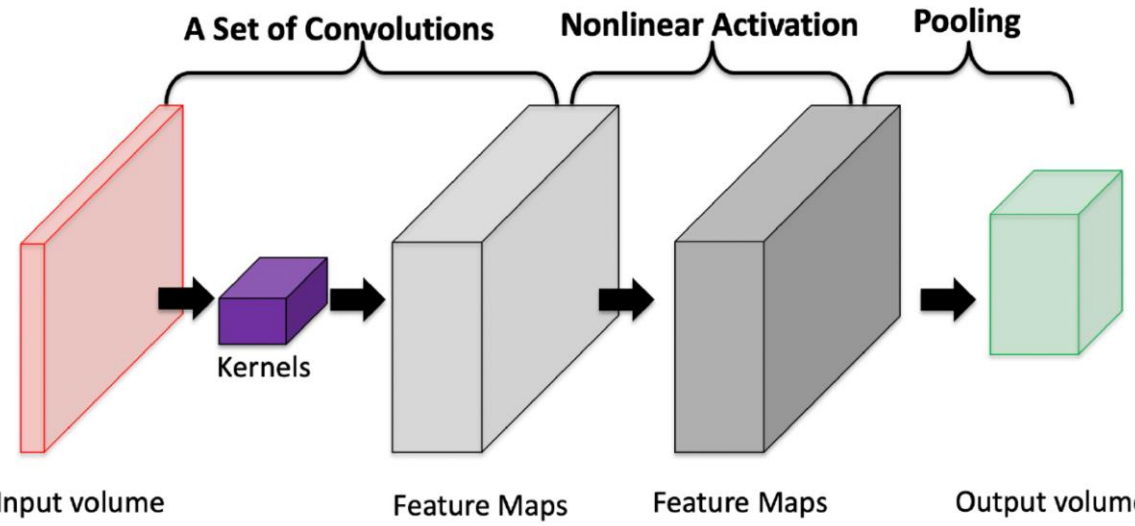
(b) Max pooling

4	3
2.5	3

(c) Mean pooling

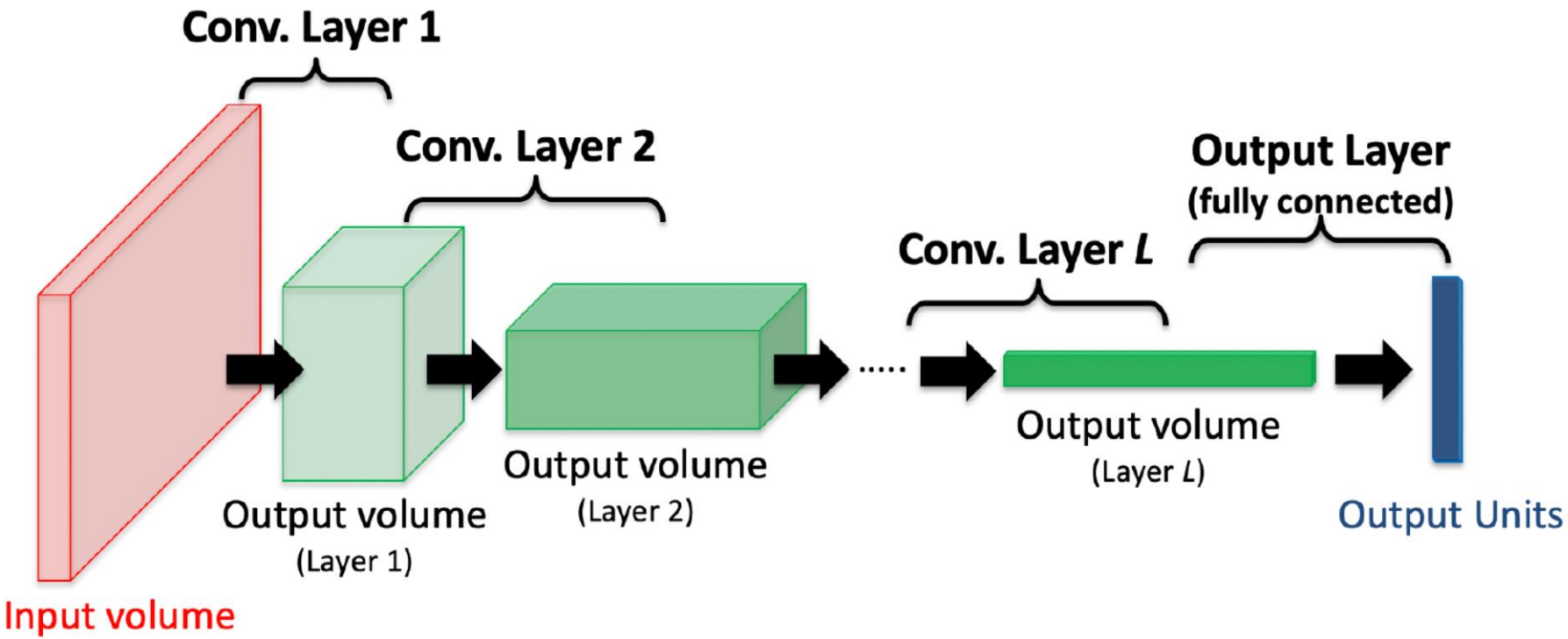
CNN: Convolutional Layer

- Multiple kernels (organized in a tensor)
  - Applied separately to the input volume
  - Full in-depth: producing a 2D feature map
  - Multiple feature maps are organized in a 3D tensor
- Followed by
  - nonlinear activation
  - pooling

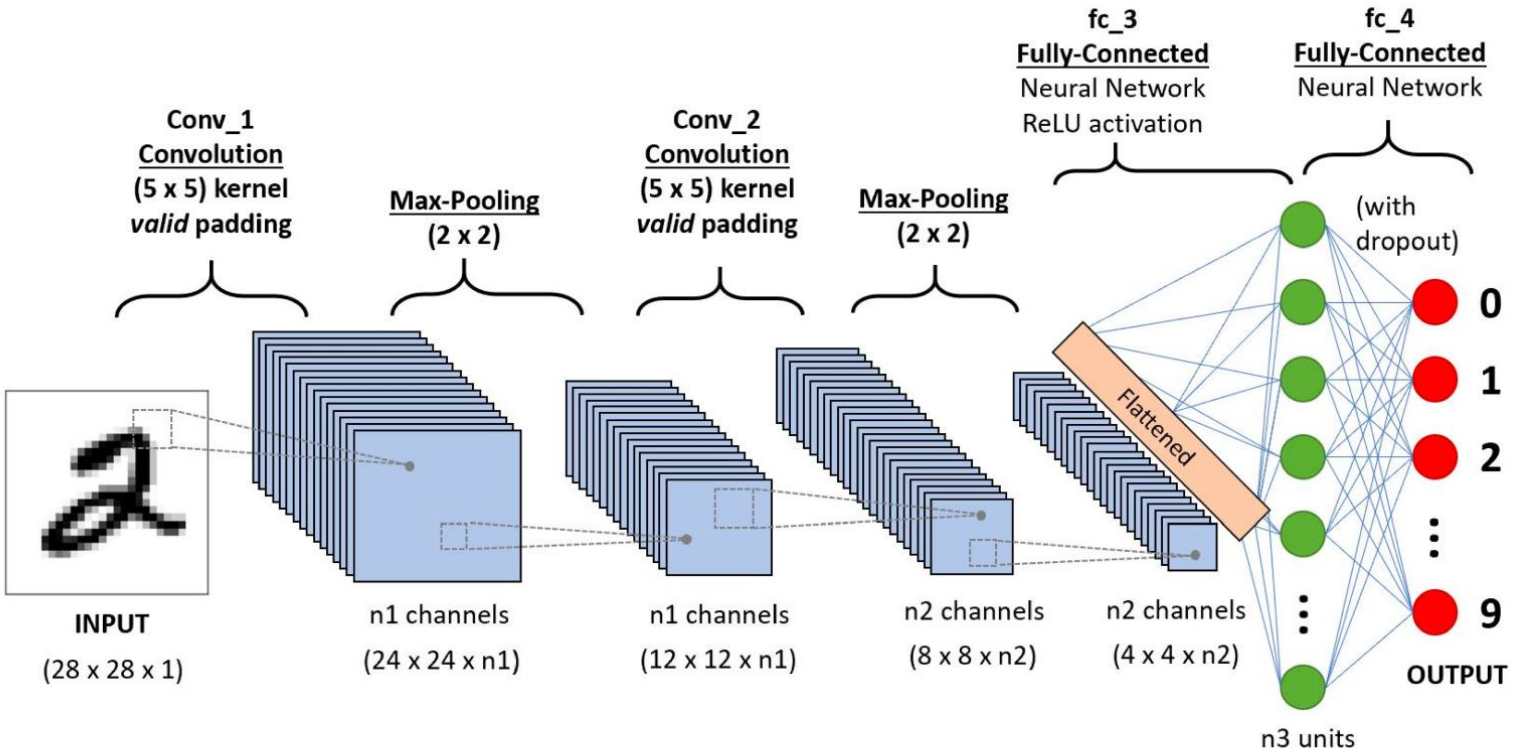


Convolutional Neural Networks

- CNN = NN has at least one convolutional layer
- Importance of # of layers



CovoluNN for Image Recognition: Example



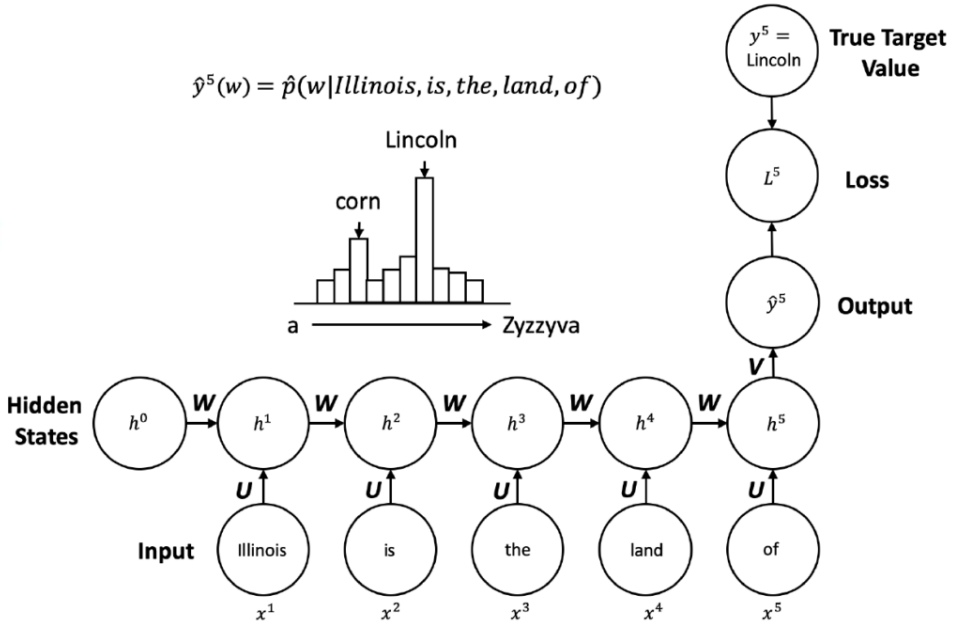
An example CNN for hand written digit recognition

Recurrent Neural Networks

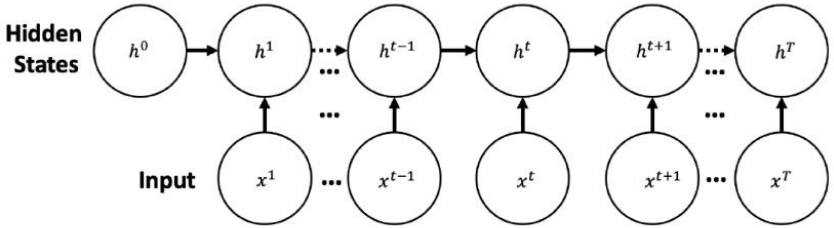
Recurrent Neural Networks

- Handling sequences with Recurrent Neural Networks (RNN)
- At each time step, the input and the previous hidden state are fed into the
- network

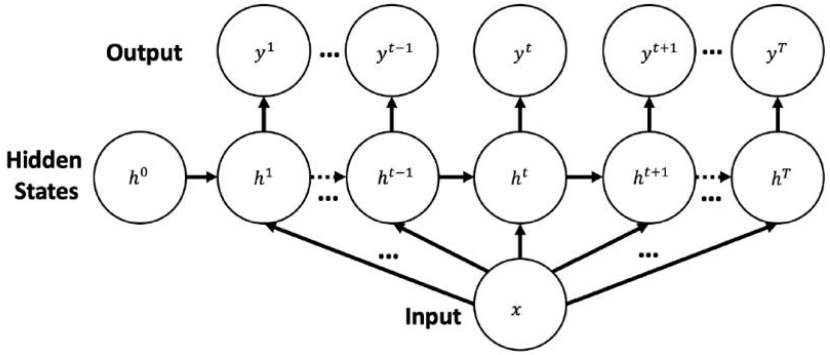
$$\boldsymbol{h}^t = f(\boldsymbol{U}\boldsymbol{x}^t + \boldsymbol{W}\boldsymbol{h}^{t-1} + \boldsymbol{a})$$
$$\hat{\boldsymbol{y}}^T = g(\boldsymbol{V}\boldsymbol{h}^T + \boldsymbol{b}).$$



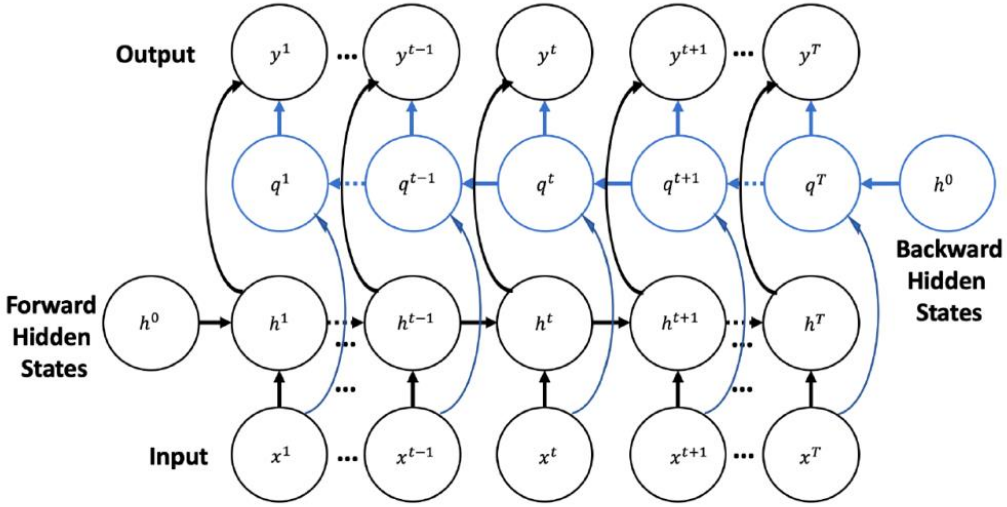
Different Types of RNNs



(a) RNN without output



(b) RNN with vector input



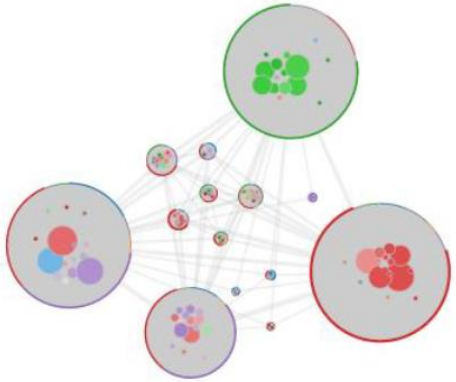
(c) bi-directional RNN

Recurrent Neural Networks: General Concepts

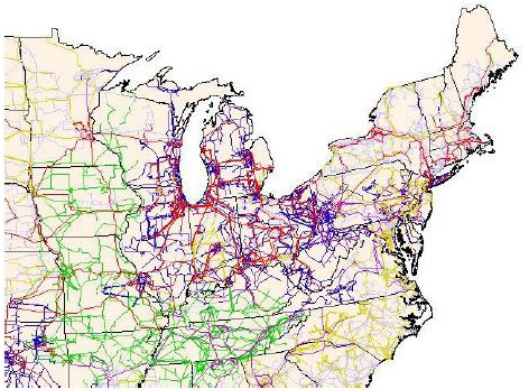
- Modeling the time dimension
  - Adding feedback loops connected to past decisions
  - Long-term dependencies: Use hidden states to preserve sequential information
- **Training RNNs**: Compute a gradient with BPTT (backpropagation through time)
- **Major obstacles of RNN**: Long-term dependence
  - An example: "The Blacksburg Chocolate Festival is held in every Oct .... There will be live music, food, beers ... . Enjoy the hot chocolate/coffee."
  - Long-term dependence calls for deep RNN, which makes gradient vanishing and/or exploding problems even worse.
  - Solutions:
    - Gated RNNs (LSTM, GRU)
    - Attention mechanism (Transformer, BERT)

Graph Neural Networks

Networks as Data: A ubiquitous data type!

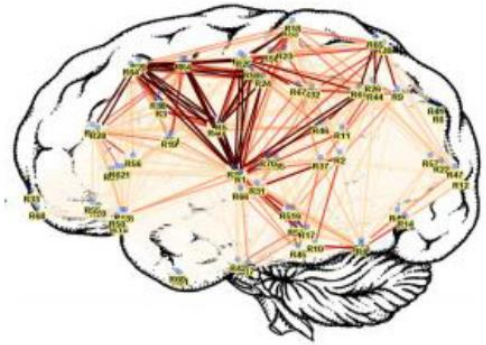


Collaboration Networks

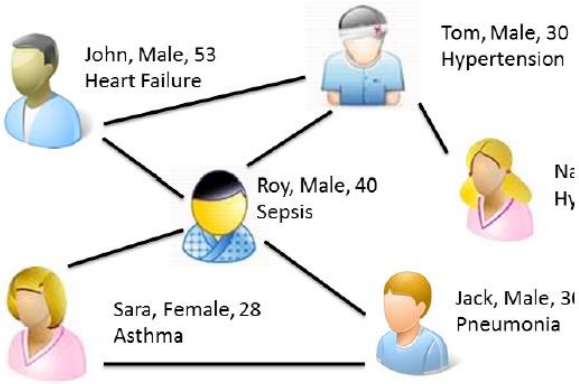


US Power Grid

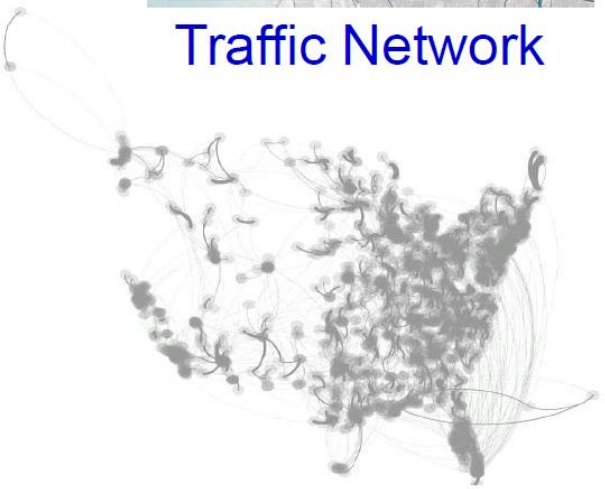
Traffic Network



Brain Networks



Patient Networks



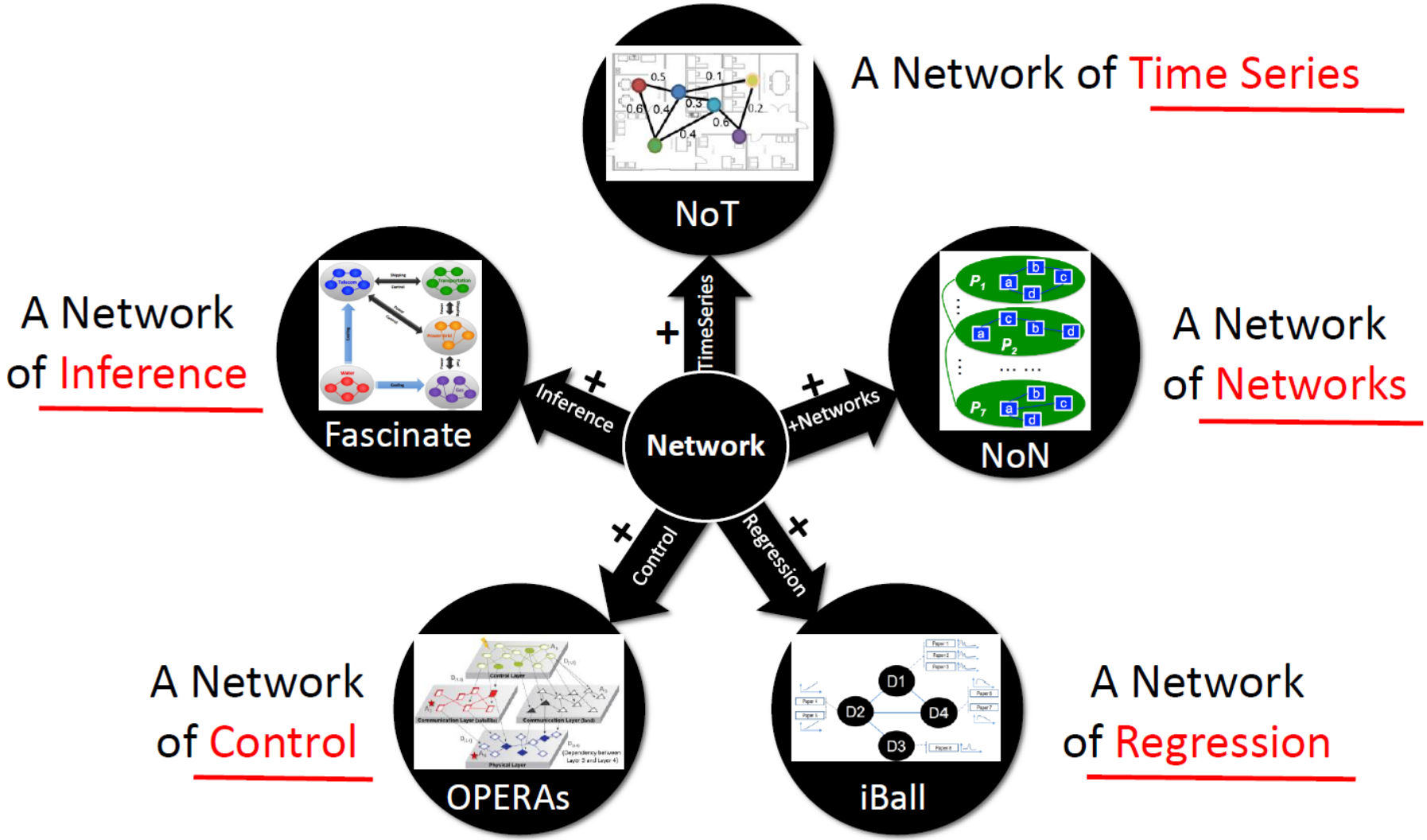
Hospital Networks

Networks as Context: Network of X

- **Observation: Each X being an Entity**
  - X = user: Social Network
  - X = information: Information Network
  - X = webpage: Web Graph
  - X = device: Internet of Things

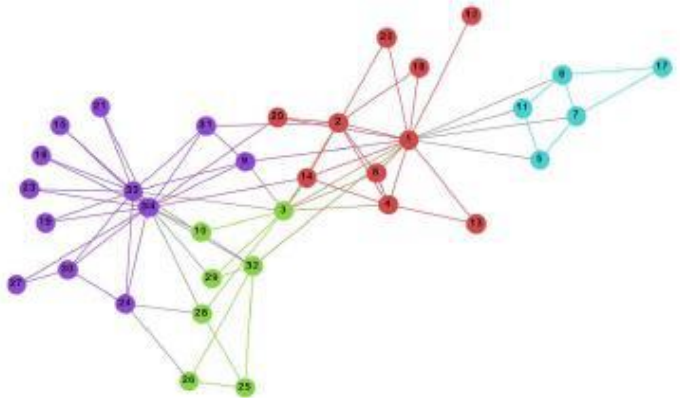
- **Question: What if Each X is**
  - A dataset (a set of entities, e.g., a network), or
  - A data mining model?

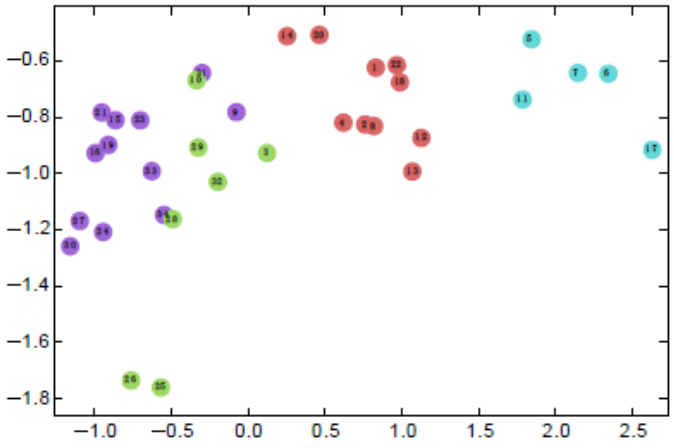
Network of X: Examples



Representation Learning on Networks

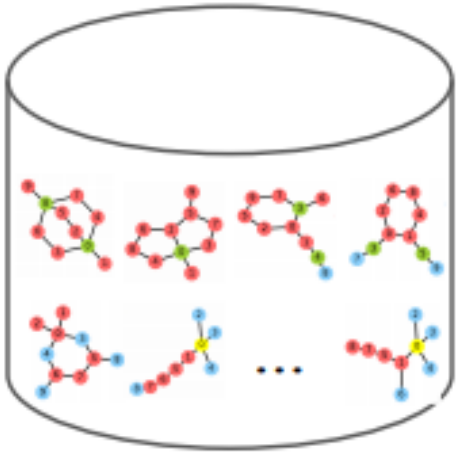
- Node level representations
  - To learn low dimensional node features
  - Similar nodes are close to each other
  - Dissimilar nodes are far from each other





Representation Learning on Networks

- Graph level representations
  - To learn low dimensional graph features Input



Graph database

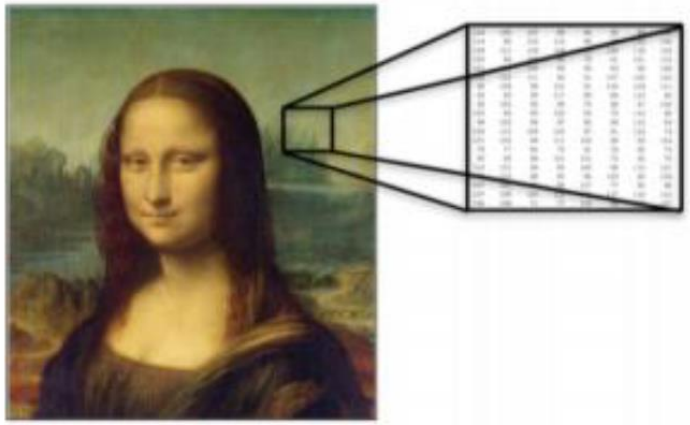
Graph embedding

Classic CNN on Graphs

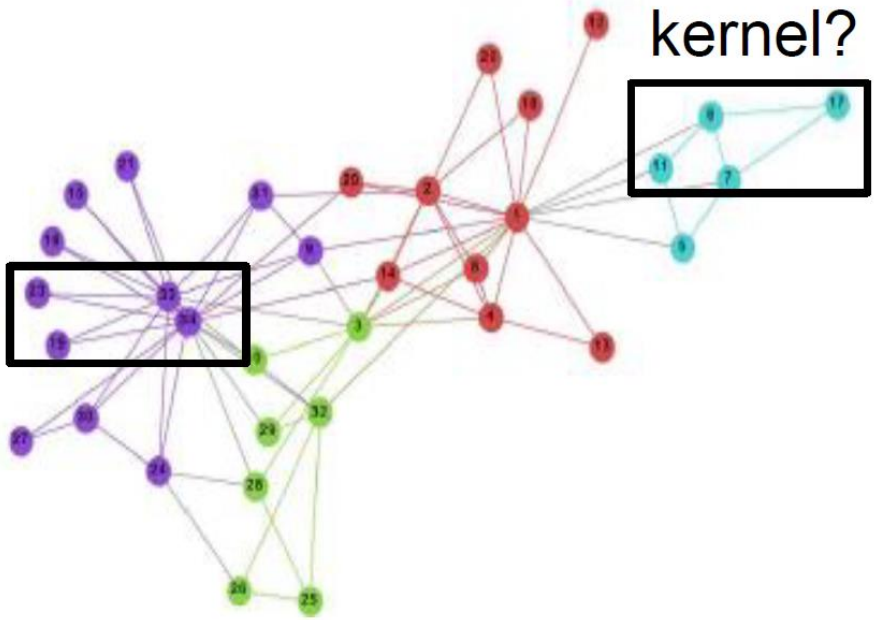
- Can we directly apply CNN?

No!

fixed-size kernel

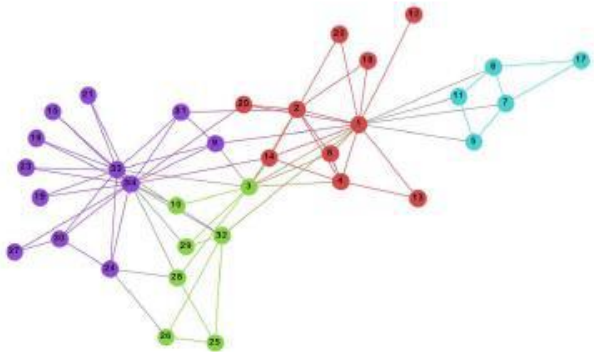


fixed-size kernel?



Challenges

- Irregular graph structure (non-Euclidean)
  - Unfixed size of node neighborhoods
- Permutation invariance
  - Node ordering does not matter
- Undefined convolution computation



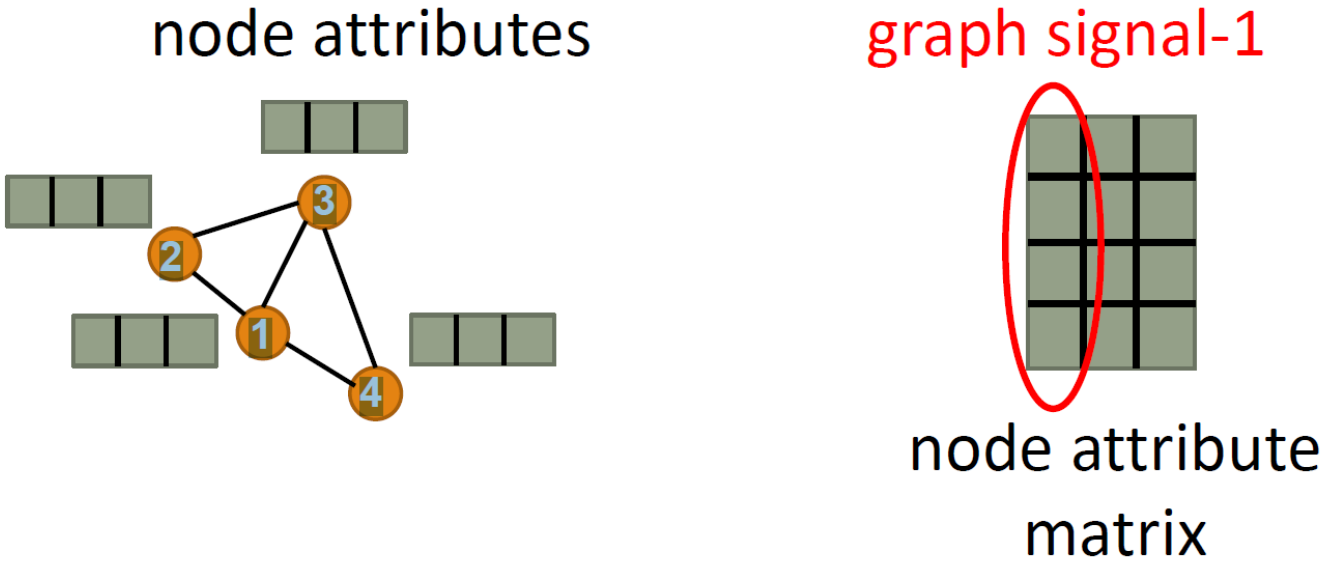
$$\boldsymbol{C}_{i,j} = \sum_{p,q} \boldsymbol{I}_{i-p,j-q} \boldsymbol{K}_{p,q}$$

Translations on graphs?

Graph Conv in Spectral Domain

- Graph signal processing

- Graph signal:
  - vector $\boldsymbol{x} \in R^n$ where $\boldsymbol{x}(i)$ is for node i



Graph Conv in Spectral Domain

- Graph spectral convolution

How to read? **By analogy!**

	Classic signal	Graph signal
Laplace Operator	$\Delta(e^{2\pi i\omega t})$	$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ (Laplacian matrix)
Frequency	$2\pi\omega$	Eigenvalues of Laplacian $\{\lambda_l\}$
Eigenfunction	$e^{2\pi i\omega t}$	Eigenvectors of Laplacian $\{\boldsymbol{u}_l\}$
Fourier transform	$\hat{f}(\omega) = \langle f, e^{2\pi i\omega t} \rangle$	$\hat{\boldsymbol{x}}(\lambda_l) = \langle \boldsymbol{x}, \boldsymbol{u}_l \rangle = \sum_i \boldsymbol{x}(i)\boldsymbol{u}_l^*(i)$
Convolution	$(f * g)(t) = \int f(t')g(t-t')dt'$	$(\boldsymbol{x} *_g \boldsymbol{y})(i) = \sum_{l=1}^{N} \hat{\boldsymbol{x}}(\lambda_l)\hat{\boldsymbol{y}}(\lambda_l)\boldsymbol{u}_l(i)$

Spectral Based Model (GCN)

- Spectral graph convolution

$$(\boldsymbol{x} *_g \boldsymbol{y})(i) = \sum_{l=1}^{N} \hat{\boldsymbol{x}}(\lambda_l)\hat{\boldsymbol{y}}(\lambda_l)\boldsymbol{u}_l(i)$$

filter

filter in spectral domain

- Filter choice: K polynomial filter (K=2)

$$\hat{\boldsymbol{y}}(\lambda_l) = \sum_{k=1}^{K} \theta_k \lambda_l^{k-1} = \theta - \theta\lambda_l$$

$$\theta_1 = -\theta_2 = \theta$$

Spectral Based Model (GCN)

- Approximate convolution (with Cheby polynomials)

$$\boldsymbol{x} *_{\theta} \boldsymbol{y} \approx \theta\left(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\right)\boldsymbol{x}, \qquad \widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n$$

- For node features $\boldsymbol{X} \in R^{n \times C}$ (i.e., C signals)

$$\boldsymbol{Z} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}\boldsymbol{\Theta}$$

Parameters

Output features

Adjacency matrix with self-loops

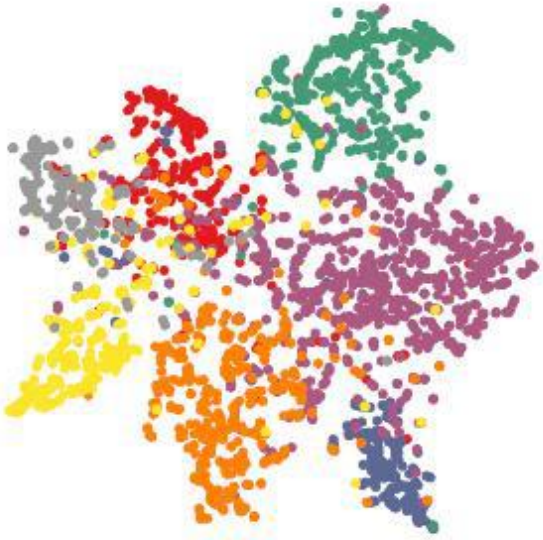
Input features

Spectral Based Model (GCN)

- A two-layer architecture for node classification

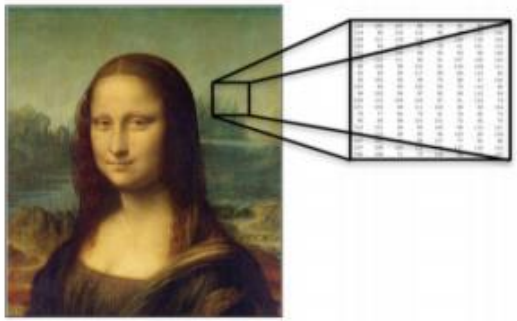
$$\widetilde{\boldsymbol{Y}} = \mathrm{softmax}\left(\widehat{\boldsymbol{A}}\sigma\left(\widehat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{\Theta}_1\right)\boldsymbol{\Theta}_2\right), \quad \widehat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$$

- Transductive learning
- Spatial interpretations
  - Linear transformation on input features
  - Propagations along edges

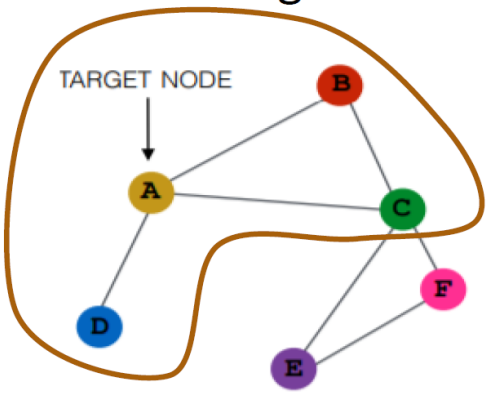


Graph Conv in Spatial Domain

- Local aggregations



local neighborhood



- Spatial graph convolution

$$\boldsymbol{x}_{out}(i) = w_{i,i}\boldsymbol{x}(i) + \sum_{j \in \mathcal{N}(i,K)} w_{i,j}\boldsymbol{x}(j)$$

K-hop neighborhood

Spatial Based Model

- Key idea: message passing
  - Defines how to aggregate node representations

- Key components:
  - Messages: $\phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$ by differentiable functions
  - Neighborhood aggregation

$$\text{Aggregate}\left(\left\{\phi\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right), j \in \mathcal{N}(i)\right\}\right)$$

  - Update by differentiable functions

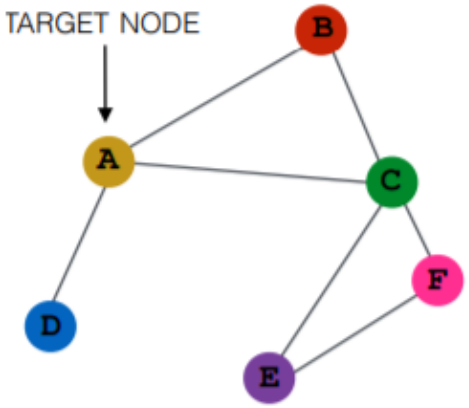
$$\boldsymbol{x}_i \leftarrow \gamma\left(\boldsymbol{x}_i, \text{Aggregate}\left(\left\{\phi\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right), j \in \mathcal{N}(i)\right\}\right)\right)$$

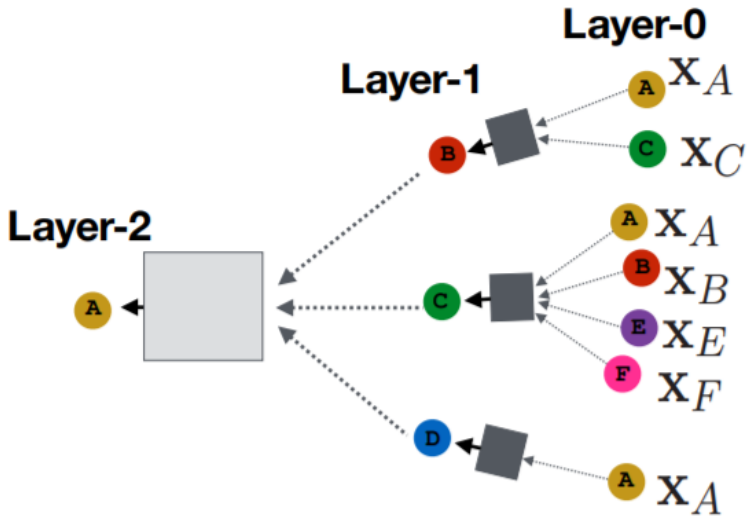
Spatial Based Model (GraphSage)

- Message passing:

$$\mathbf{h}_{\mathcal{N}(v)}^{k} \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$
$$\mathbf{h}_v^{k} \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^{k})\right)$$

  - Aggregation: mean, max pooling, LSTM





Spatial Based Model (GAT)

- Message Passing:

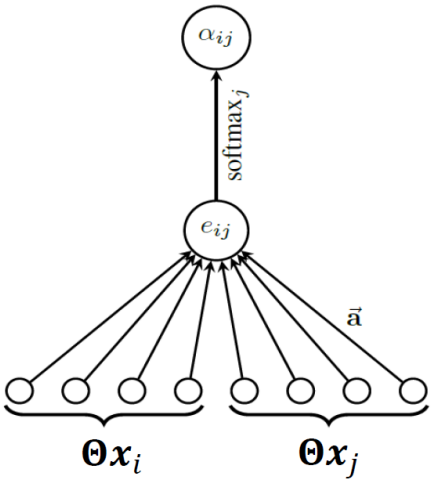
$$\boldsymbol{x}_i \leftarrow \alpha_{i,i}\boldsymbol{\Theta}\boldsymbol{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\boldsymbol{\Theta}\boldsymbol{x}_j$$

spatial convolution

$$\boldsymbol{x}_{out}(i) = w_{i,i}\boldsymbol{x}(i) + \sum_{j\in\mathcal{N}(i,K)} w_{i,j}\boldsymbol{x}(j)$$

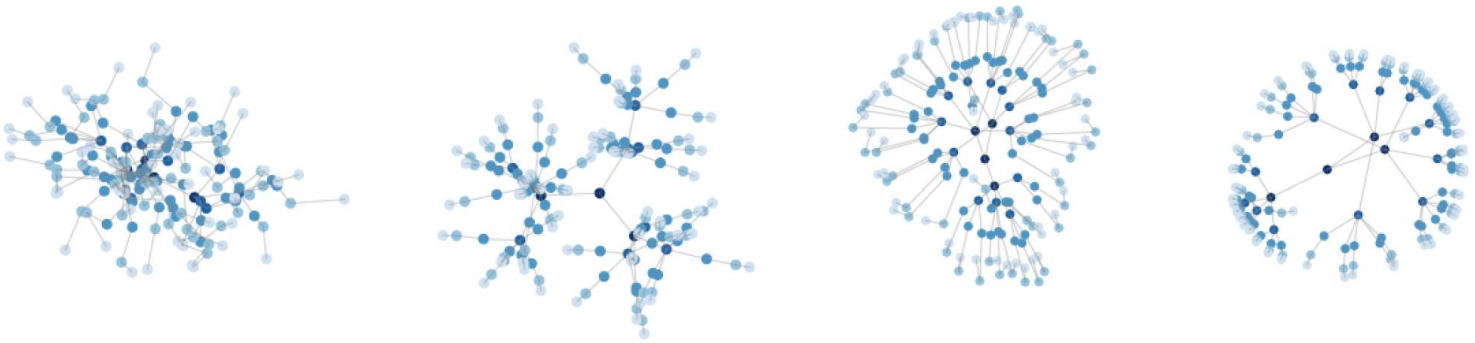
  - Attention coefficients can be viewed as importance

$$\alpha_{ij} = \frac{\exp(\boldsymbol{a}^T[\boldsymbol{\Theta}\boldsymbol{x}_i||\boldsymbol{\Theta}\boldsymbol{x}_j])}{\sum_{k\in\mathcal{N}(i)}\exp(\boldsymbol{a}^T[\boldsymbol{\Theta}\boldsymbol{x}_i||\boldsymbol{\Theta}\boldsymbol{x}_k])}$$



Spatial Based Model (HGCN)

- Hyperbolic graph convolutional networks
  - Key idea: message passing in hyperbolic space
  - Why: better capture graph hierarchical structure



(a) GCN layers.

(b) HYPERGCN layers.

Other Aspects in Graph Conv Nets (Optional)

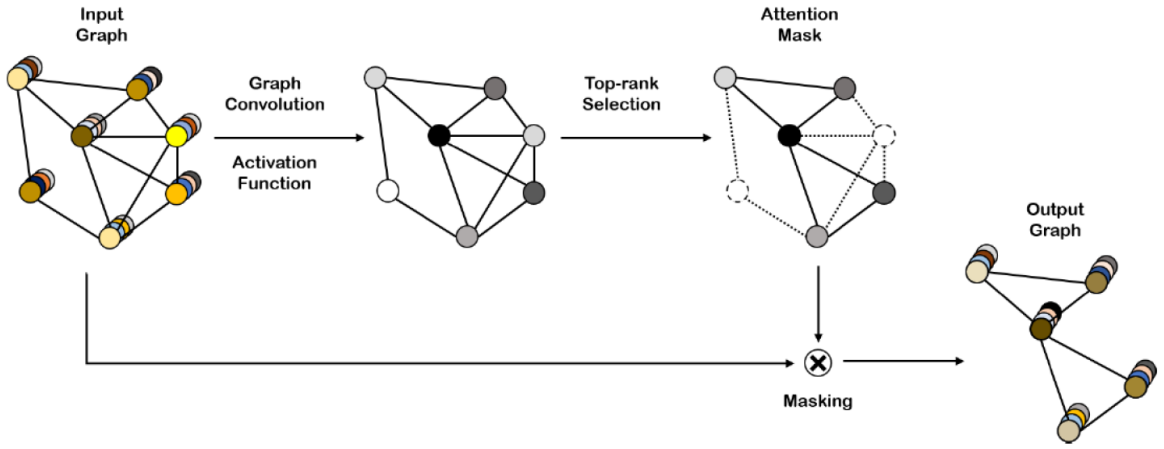
- Pooling methods
  - To better capture graph hierarchical structure.
  - Deterministic methods:
    - Multi level graph coarsening/clustering, etc.
  - Adaptive methods:
    - Neural pooling layer

- Training methods:
  - Stochastic training minibatch sampling
    - Uniform sampling, importance sampling, etc.

Self Attention Pooling (Optional)

- Key idea: self-attention-based importance

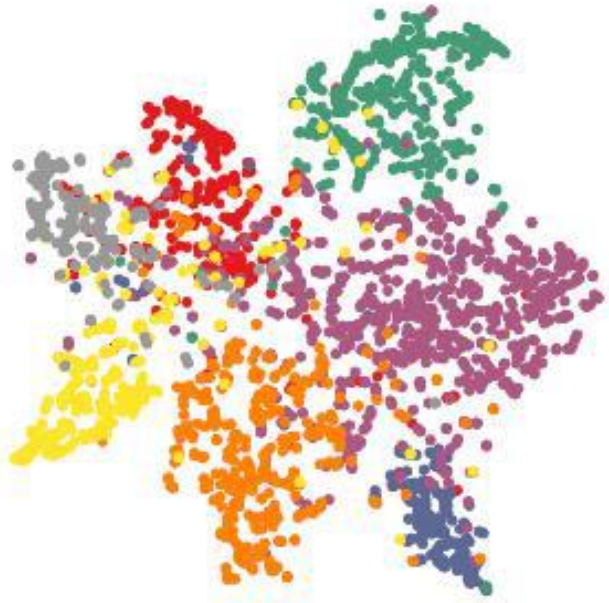
$$\boldsymbol{Z} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\mathbf{X}\boldsymbol{\theta}_{att}$$



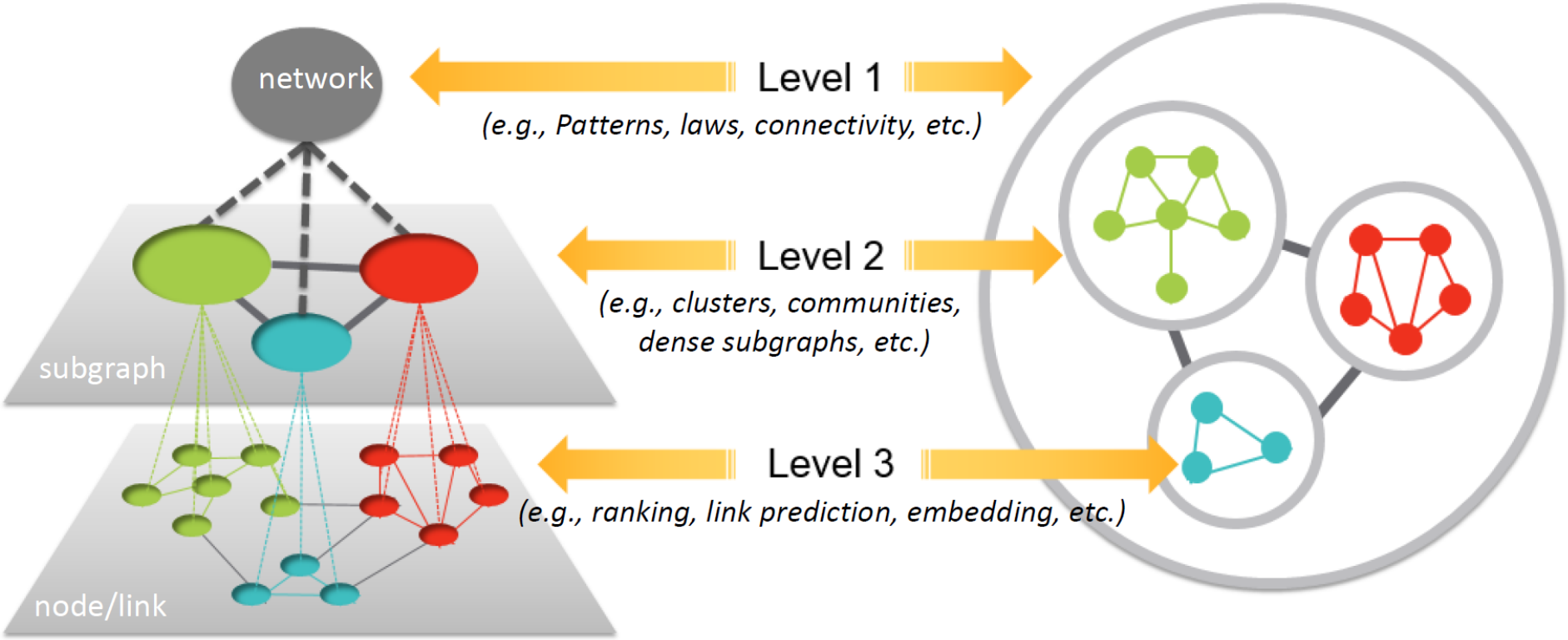
- Tasks:
  - Graph representation
  - Node representation (need a good architecture)

Applications

- Node classification
  - Predict node categories
- Link prediction
  - Predict edges in the network
- Graph classification
  - Predict graph categories
- Many more …
  - Recommendation
  - Multiple network mining



Applications: An Overview



Summary

- Basic Concepts
  - Neuron, activation functions
  - MLP, feed-forward neural networks
  - Backpropagation algorithm
- Techniques to Improve Deep Learning Training
  - Key challenges: optimization vs. generalization
  - Responsive activation functions, adaptive learning rate, dropout, pretraining, cross-entropy
- Convolutional Neural Networks
  - Convolution, pooling, parameter sharing
- Recurrent Neural Networks
  - Sequence data, long term dependence
- Graph Neural networks
  - GCN, GAT, GraphGAN

References (1)

- C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995
- S. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2008
- Quoc V Le. Building high-level features using large scale unsupervised learning. In 2013 IEEE international conference on acoustics, speech and signal processing, pages 8595{8598. IEEE, 2013.
- Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzel. Learning to diagnose with lstm recurrent neural networks. In International Conference on Representation Learning, 2016.
- Martin L angkvist, Lars Karlsson, and Amy Lout . A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters, 42:11{24, 2014.
- Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Recti er nonlinearities improve neural network acoustic models.
- Vu Pham, Theodore Bluche, Christopher Kermorvant, and Jer^ome Louradour. Dropout improves recurrent neural networks for handwriting recognition. In 2014 14th international con- ference on frontiers in handwriting recognition, pages 285{290. IEEE, 2014.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, Parallel Distributed Processing. MIT Press, 1986.

References (2)

- Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2015.
- Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. arXiv preprint arXiv:1710.05941, 2017.
- Nitish Srivastava, Geo rey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from over tting. The journal of ma- chine learning research, 15(1):1929{1958, 2014.
- Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru
- Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1{9, 2015.
- Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. volume 45, pages 2673{2681. Ieee, 1997.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 310-3112, 2014.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

References (3)

- Shuman, David I., et al. "The emerging field of signal processing on graphs: Extending high dimensional data analysis to networks and other irregular domains." IEEE signal processing magazine 30.3 (2013): 83 98
- Kipf , Thomas N., and Max Welling. "Semi supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).
- Hamilton, Will, Zhitao Ying, and Jure Leskovec . "Inductive representation learning on large graphs." Advances in neural information processing systems . 2017.
- Veličković , Petar , et al. "Graph attention arXiv preprint arXiv:1710.10903 (2017).
- Lee, Junhyun , Inyeop Lee, and Jaewoo Kang. "Self attention graph pooling." arXiv preprint arXiv:1904.08082 (2019).
- Wang, Xiang, et al. "Neural graph collaborative filtering." Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval . 2019.
- Chami , Ines, et al. "Hyperbolic graph convolutional neural Advances in Neural Information Processing Systems . 2019.
- Zhang, Si, et al. "Graph convolutional networks: a comprehensive review." Computational Social Networks 6.1 (2019): 11.