CS 412 Intro. to Data Mining

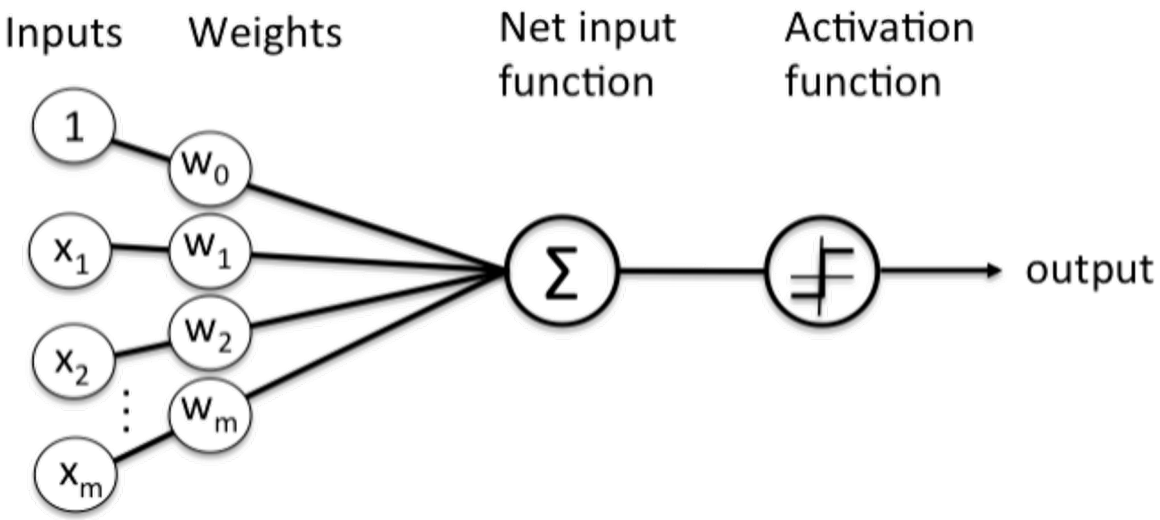
Chapter 11. Deep Learning

Arindam Banerjee, Computer Science, UIUC, Fall 2022

Chapter 11. Deep Learning

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

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

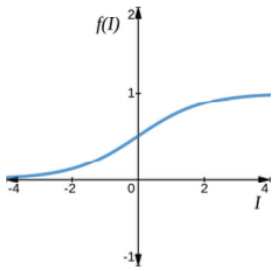
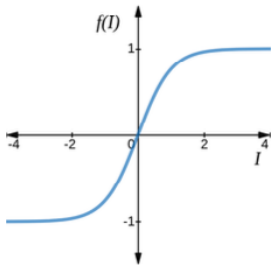
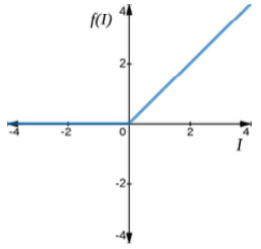
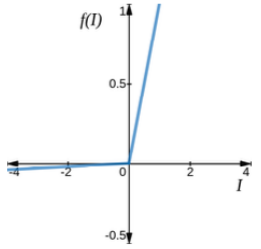
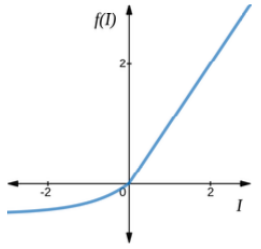
Weights

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

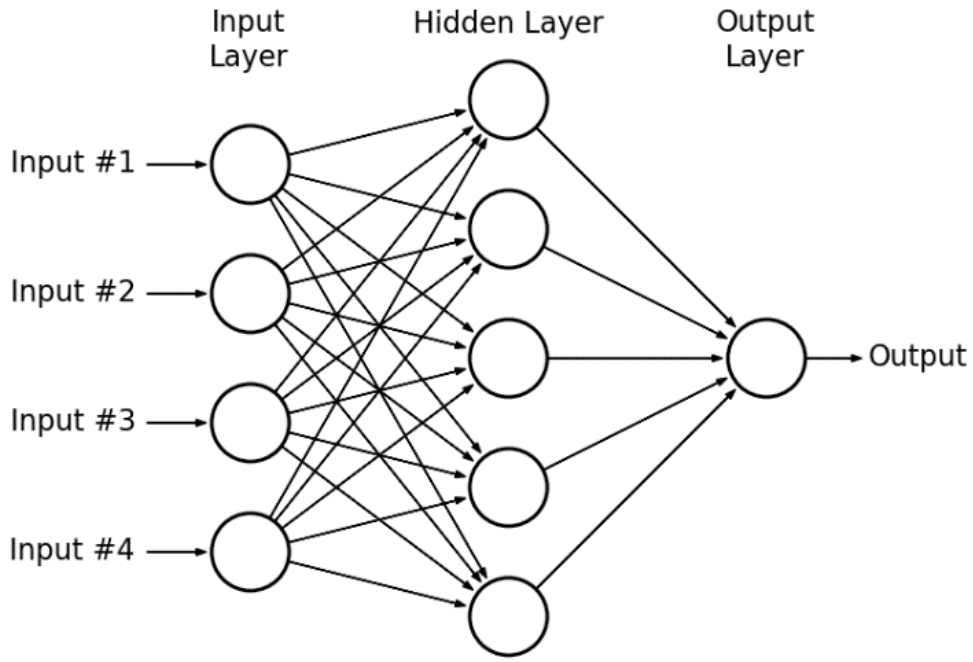
- Computes a weighted sum of inputs
- Invented in 1957 by Frank Rosenblatt. The original perceptron model does not have a non-linear activation function

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	$\begin{cases} \alpha(e^{I}-1), I \leq 0 \\ I, I > 0 \end{cases}$	

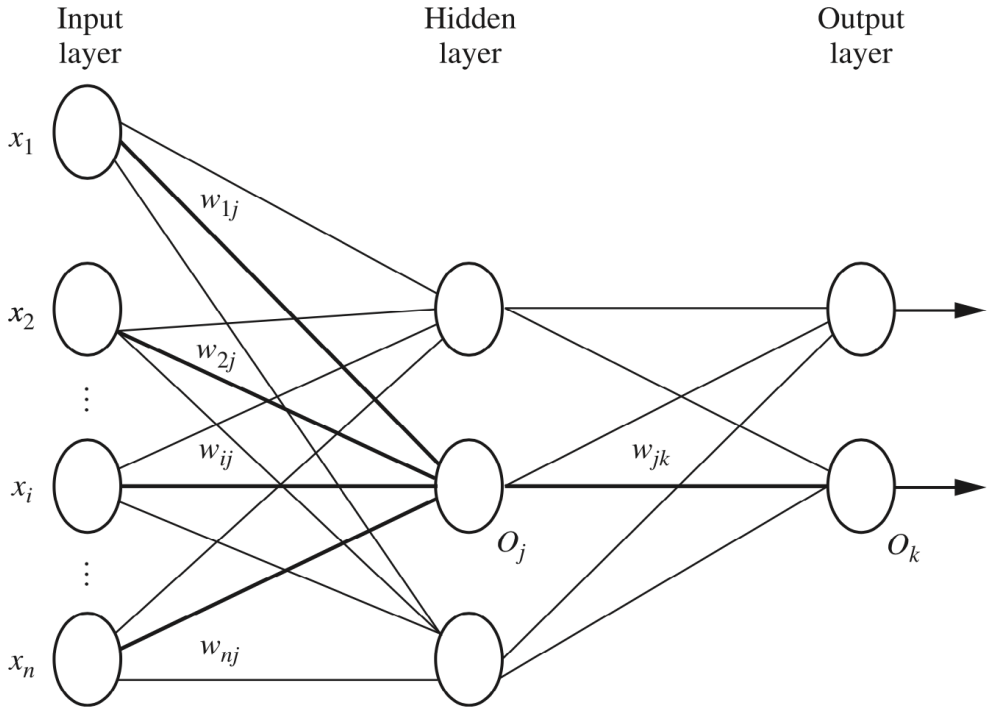
Multilayer Perceptron (MLP)

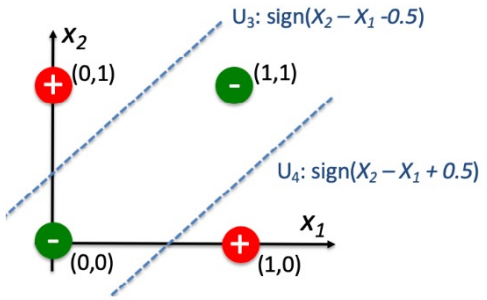


- Stacking multiple layers of perceptrons (adding hidden layers) makes a multilayer perceptron (MLP)
- MLP can engage in sophisticated decision making, where basic perceptrons fail
  - E.g. XOR problem

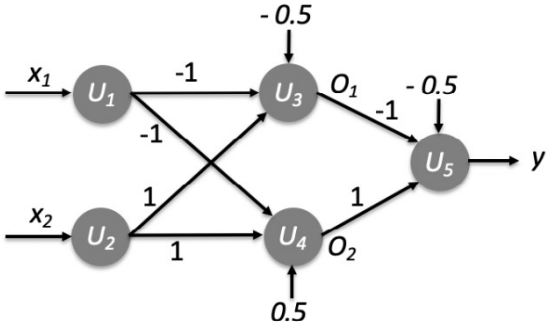
Feed-forward Neural Networks

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

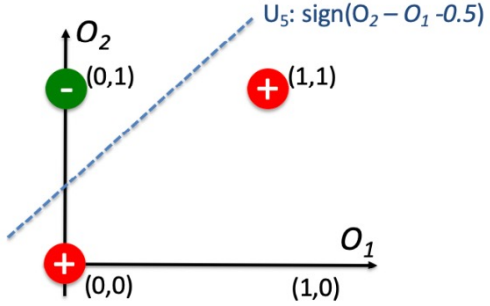




(a) Input 4 tuples

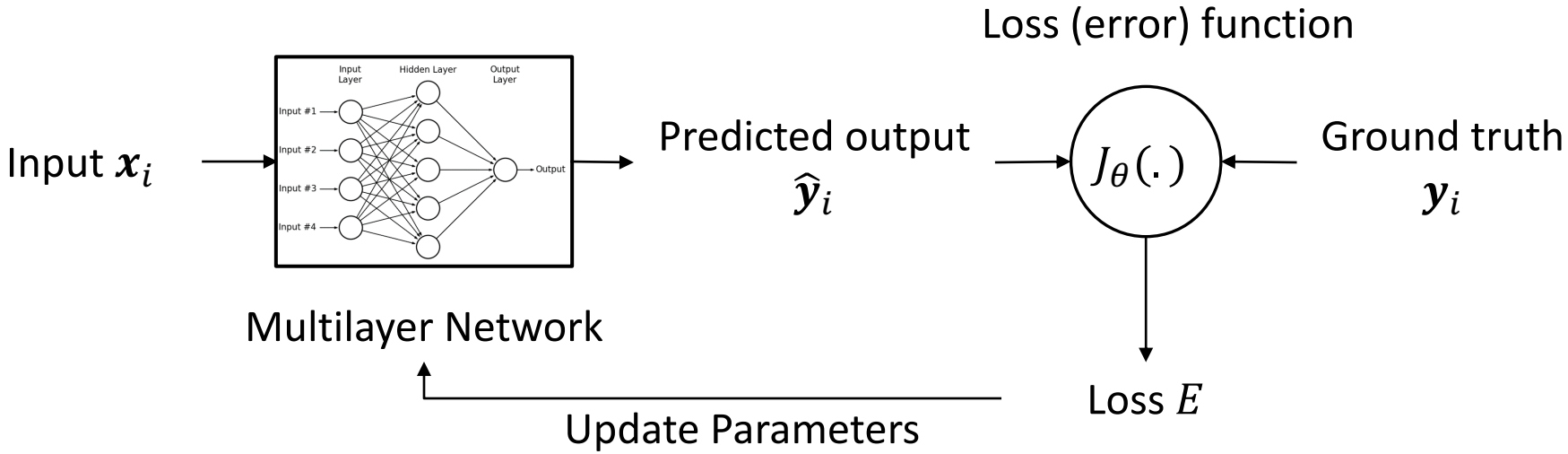


(b) MLP



(c) Outputs of two hidden units

Learning NN Parameters

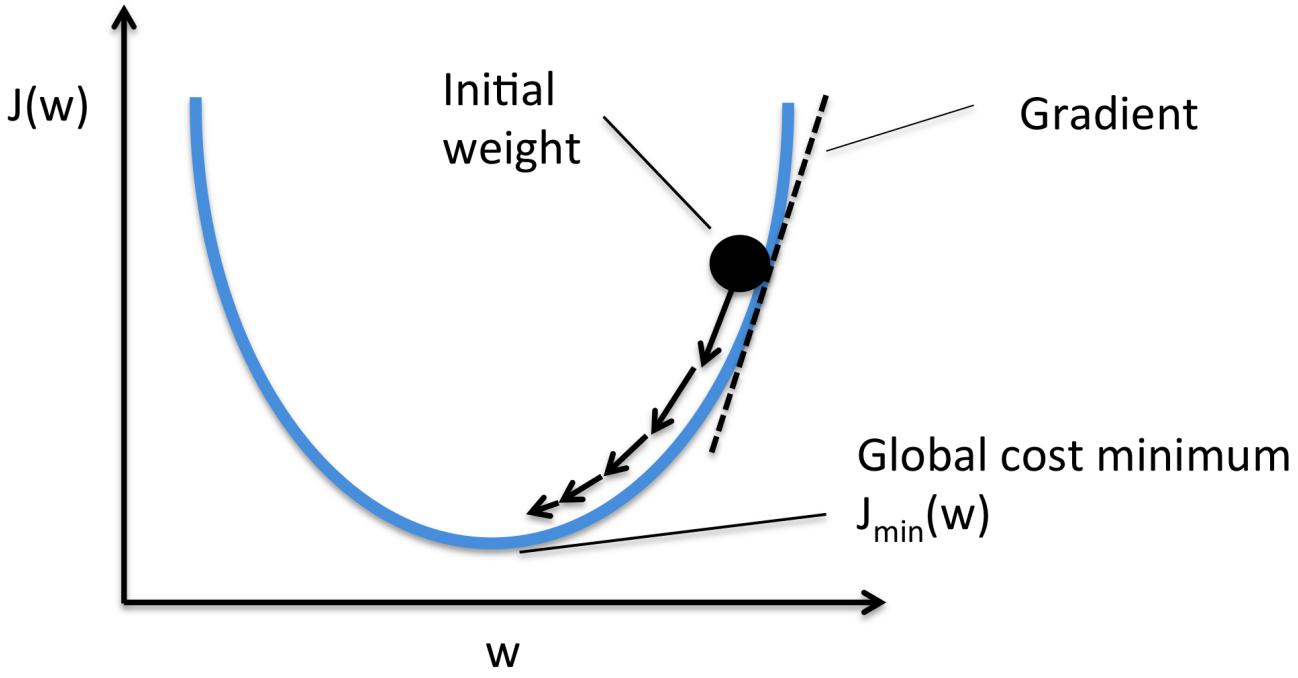


- **Gradient Descent Algorithm**
- **Input**: Training sample $\boldsymbol{x}_i$ and its label $\boldsymbol{y}_i$

1. **Feed Forward**: Get prediction $\hat{\boldsymbol{y}}_i = \mathrm{MLP}(x_i)$, and loss $E = J(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$
2. **Compute Gradient**: For each parameter θ_j (weights, bias), compute its gradient $\frac{\partial}{\partial \theta_j} J_\theta$
3. **Update Parameter**: $\theta_j = \theta_j - \eta \cdot \frac{\partial}{\partial \theta_j} J_\theta$

Explained later

Empirical Explanation of Gradient Descent

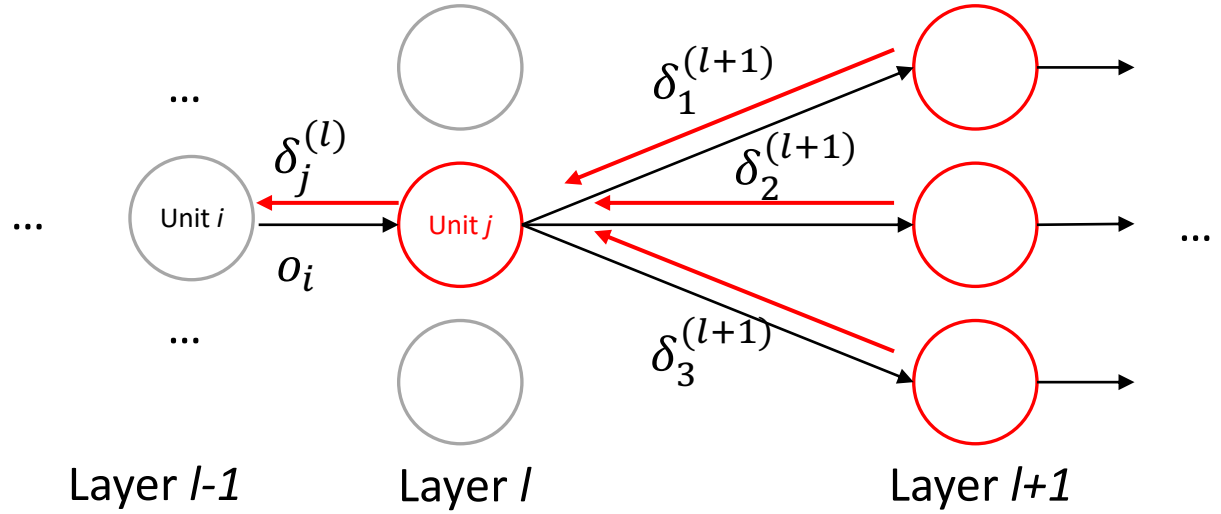


$$\theta_j = \theta_j - \eta \cdot \frac{\partial}{\partial \theta_j} J_\theta$$

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

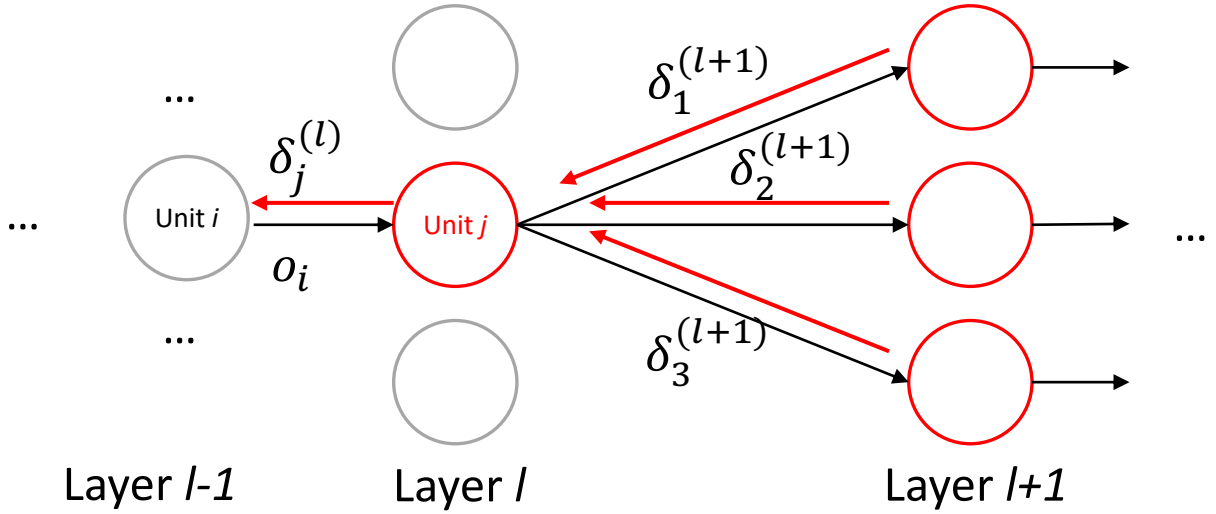
- Our objective is to minimize the loss function J, which is a function of the model parameters
- A gradient measures how much the output of a function changes if you change the inputs a little bit
- We update the parameters, based on their gradients, so that the loss function is going 'downhill'

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

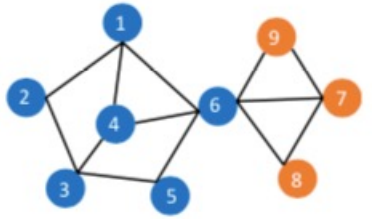
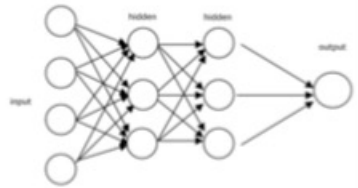
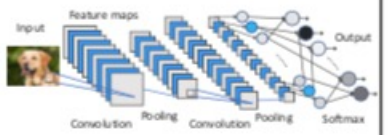
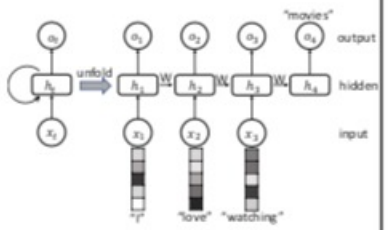
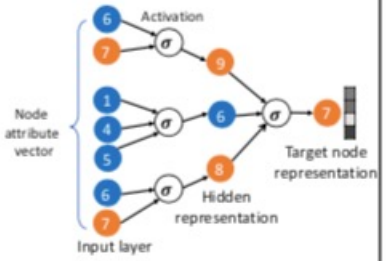


- The 'error' terms $\delta_j^{(l)}$ is a function of
  - All $\delta_k^{(l+1)}$ in the layer l+1, if layer l is a hidden layer
  - The overall loss function value, if layer l is the output layer
- We can compute the error at the output, and distribute backwards throughout the network's layers (backpropagation)

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 

Chapter 11. Deep Learning

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

Techniques to Improve Deep Learning Training

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

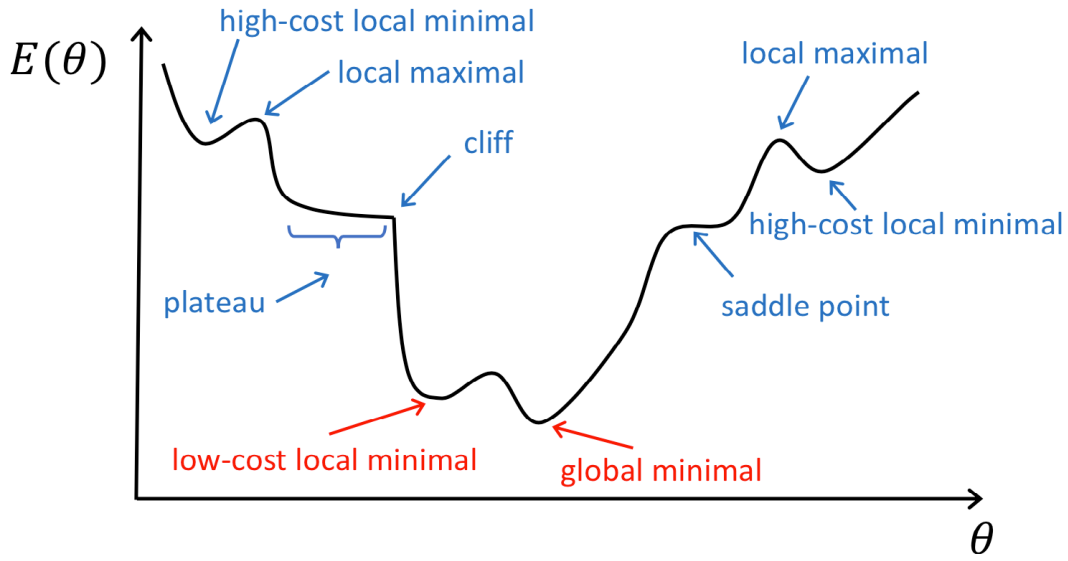
Key Challenges

- **Optimization Problem in Deep Learning**
  - Minimize the (approximated) training error *E*
  - (Stochastic) gradient descent to find the model parameter

$$E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{l=1}^{m} \mathrm{Loss}(\hat{T}(X^l, \boldsymbol{\theta}), T^l)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{g}_t$$

- **Challenge 1: Optimization**
  - *E* is non-convex in general
  - How to find a high-quality local optima

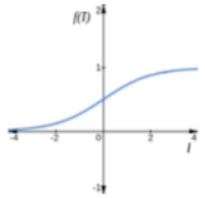
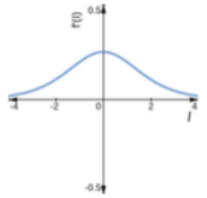
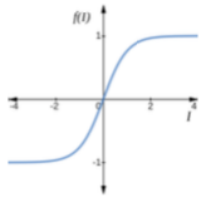
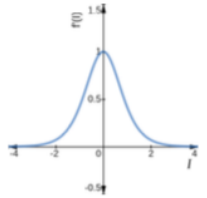
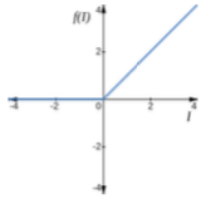
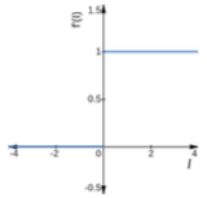
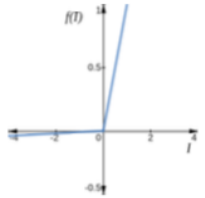
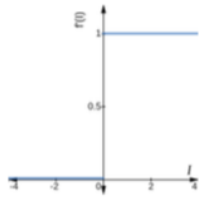
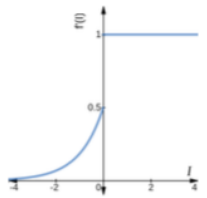


- **Challenge 2: Generalization**
  - What we do: minimize (approximated) training er
  - What we really want: minimize the generalization error
  - How to mitigate over-fitting

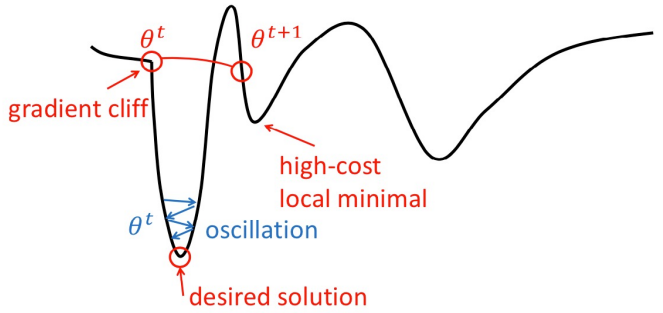
Responsive Activation Function

- **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 $\delta_j = O_j(1-O_j)(T_j - O_j)$ — $O_j(1-O_j)$ 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

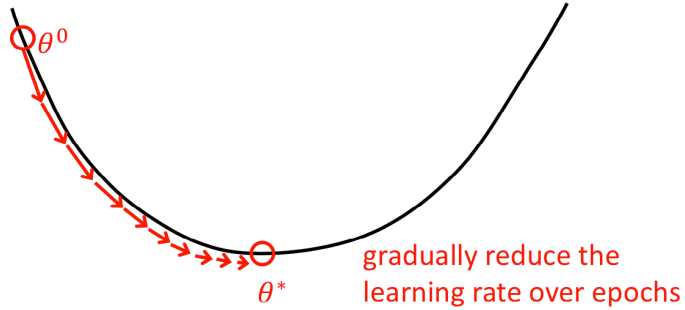
Activation Functions

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 $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{g}_t$
  - 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: $\eta_t = (1 - \frac{t}{T})\eta_0 + \frac{t}{T}\eta_\infty$ 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): $w_i^{(t+1)} \leftarrow w_i^{(t)} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}$
    - Intuition: Magnitude of gradient g_t: used to slow down/speed up progress
  - Strategy #4 (Adam, RMSProp): decaying weighted sum of squared historical gradients



Adaptive Learning Rates: Adam

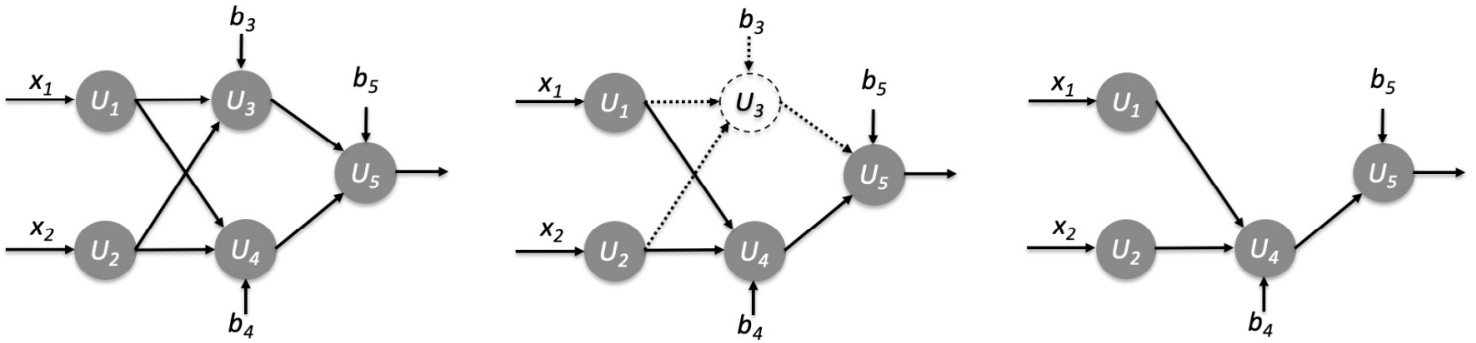
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Dropout

- **The Purpose of Dropout: to Prevent Overfitting**
- **How does it works?**
  - 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

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

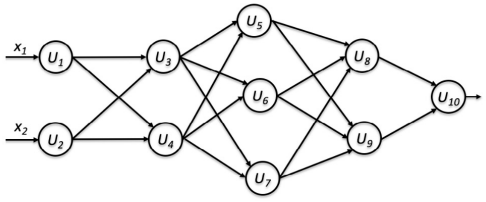


(a) Original network (b) Dropout U_3 (c) Dropout network

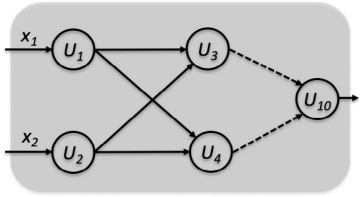
- **Why does Dropout Work?**
  - Dropout can be viewed as a regularization technique
    - Force the model to be more robust to noise, and to learn more generalizable features
  - Dropout vs. Bagging
    - Bagging: Each base model is trained *independently* on a bootstrap sample
    - Dropout: the model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

Pretraining

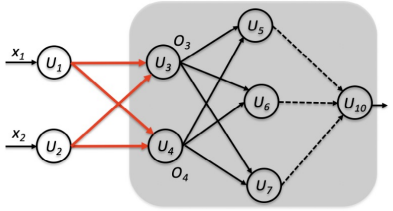
- **Pretraining:** the process of initializing the model in a suitable region
- **Greedy supervised pretraining**
  - pre-set the model parameters layer-by-layer in a greedy way
  - Start with simple model, add one additional layer at a time,
  - pretrain the parameters of the newly added layer, while 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**



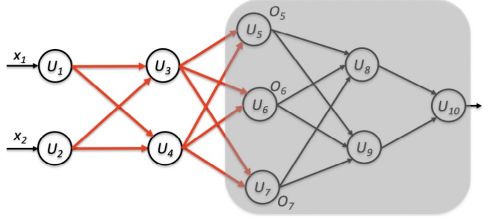
(a) original model to pretrain



(b) first iteration (1 hidden layer)



(c) second iteration (2 hidden layers)

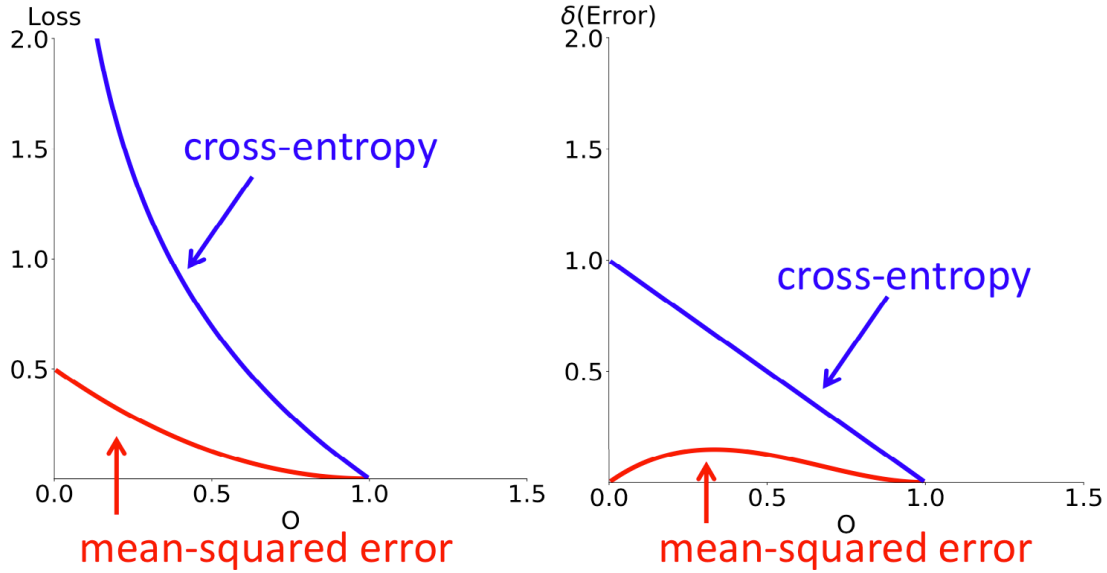


(d) third iteration (3 hidden layers)

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	$O(1-O)(T-O)$	$T-O$



Assuming T = 1 (positive tuple)

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

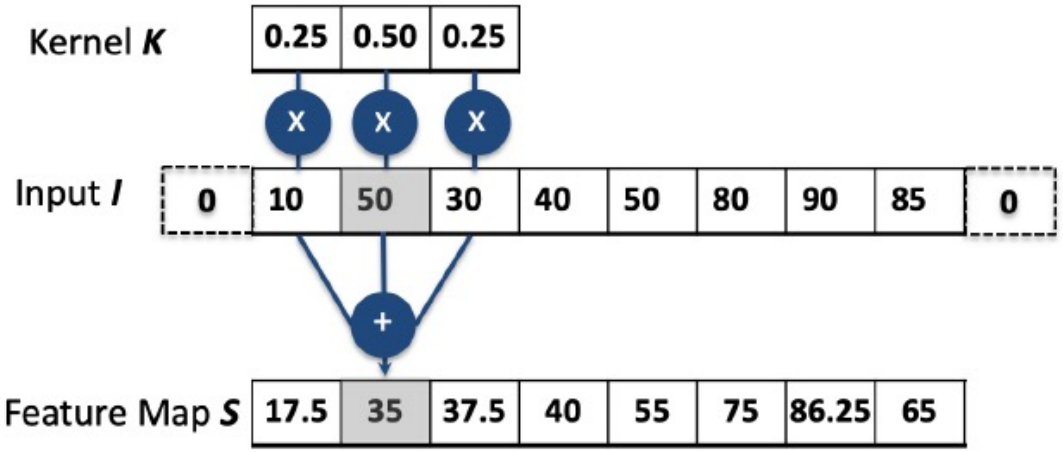
$$\text{Loss}(T,O)=-\sum_{j=1}^{C}T_j\log O_j$$

Chapter 11. Deep Learning

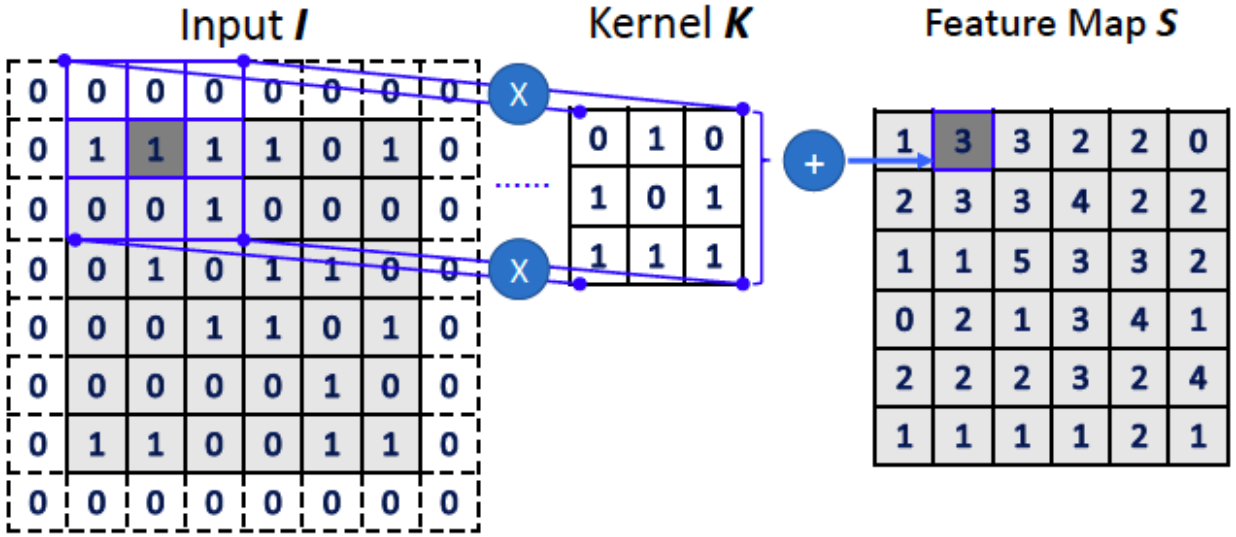
- Basic Techniques
- Techniques to Improve Deep Learning Training
- Convolutional Neural Networks 
- Recurrent Neural Networks
- Summary

Convolutional Neural Networks (CNN)

- What is convolution? $S = I * K$



1D Convolution



2D Convolution

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

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

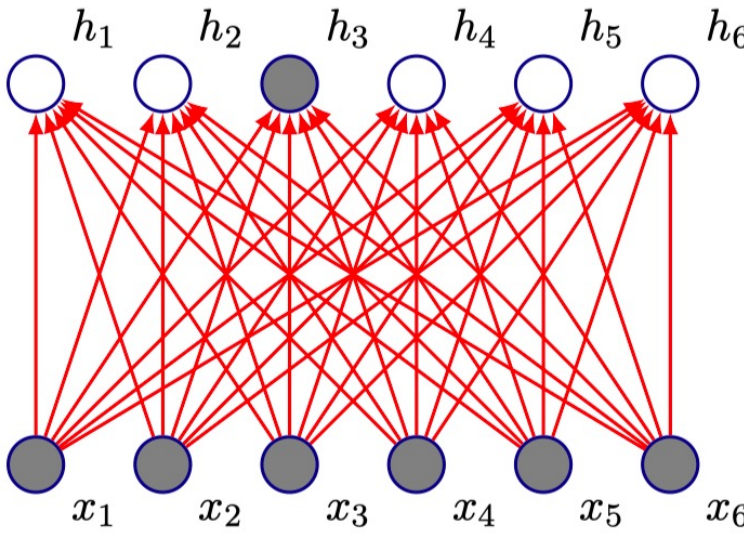
- The outputs are computed by sliding a **kernel** (of weights) on the inputs, and computing weighted sum locally

CNN Motivation

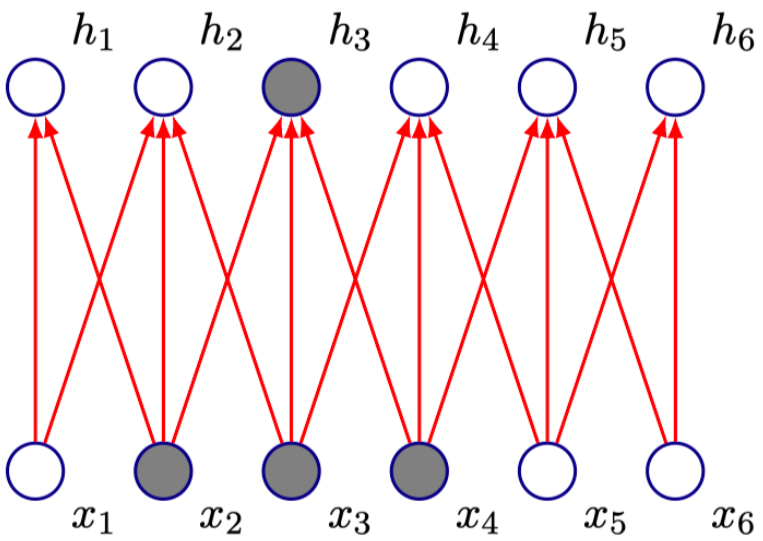
- Why not deep MLP (or feed-forward neural network)?
  - Deep multilayer perceptrons are **computationally expensive**, and **hard to train**.
    - Long training time, slow convergence, local minima
- Motivations of convolution
  - Sparse interactions
  - Parameter sharing
  - Translational equivalence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

CNN Motivation

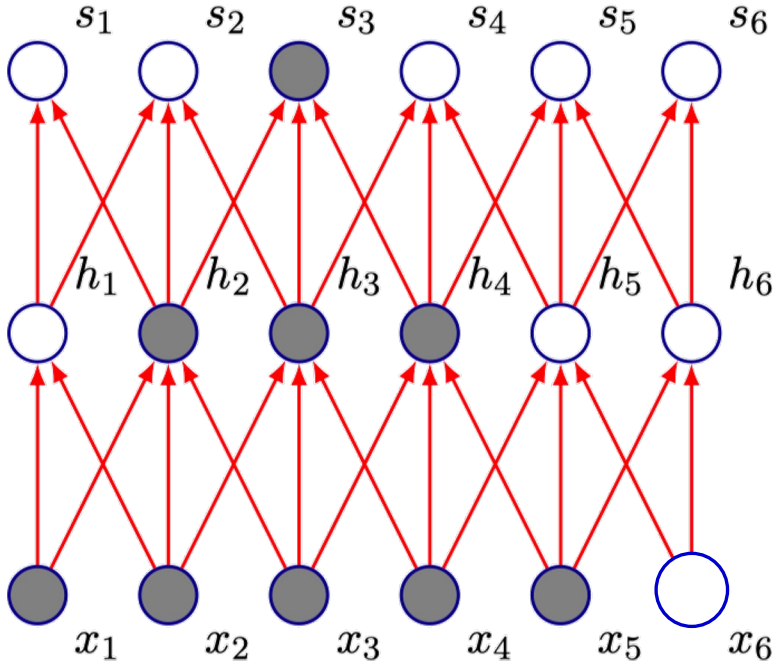
- Motivations of convolution
  - Sparse interactions
    - E.g. 1D convolution with kernel size 3
    - Units in deeper layers still connect to a wide range of inputs



MLP



CNN

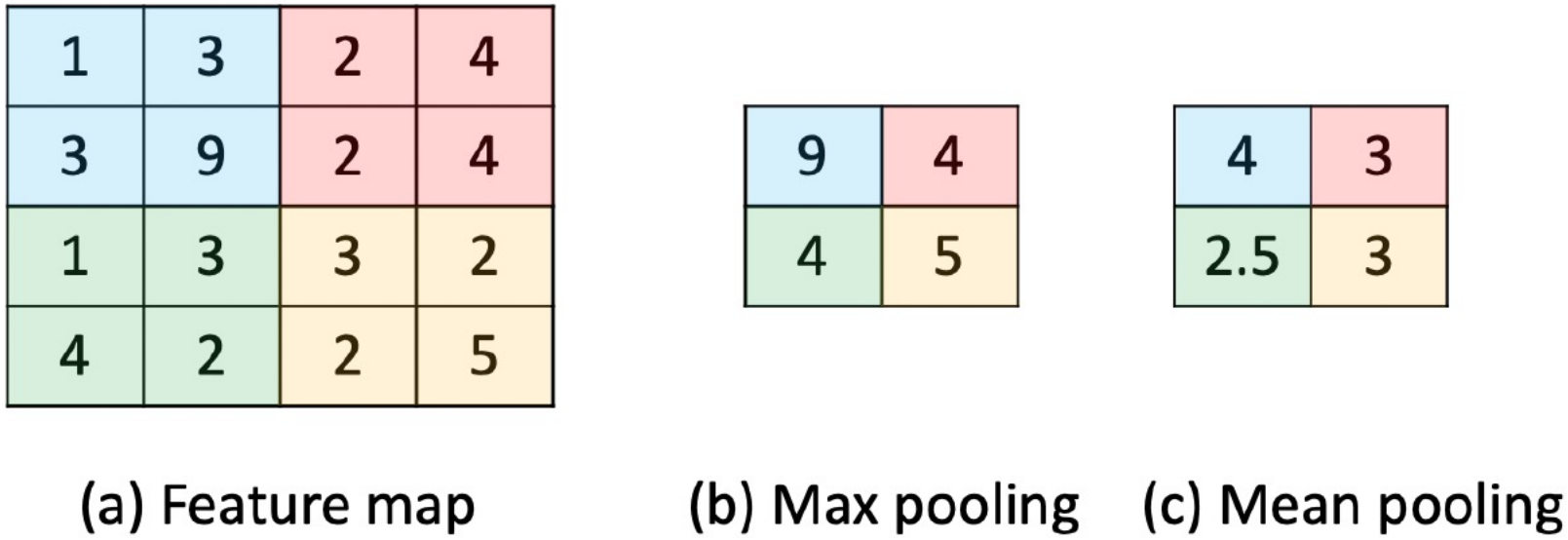


CNN Motivation

- 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 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. “University of Illinois” at the start of the sentence and at the end of the sentence produce the same outputs

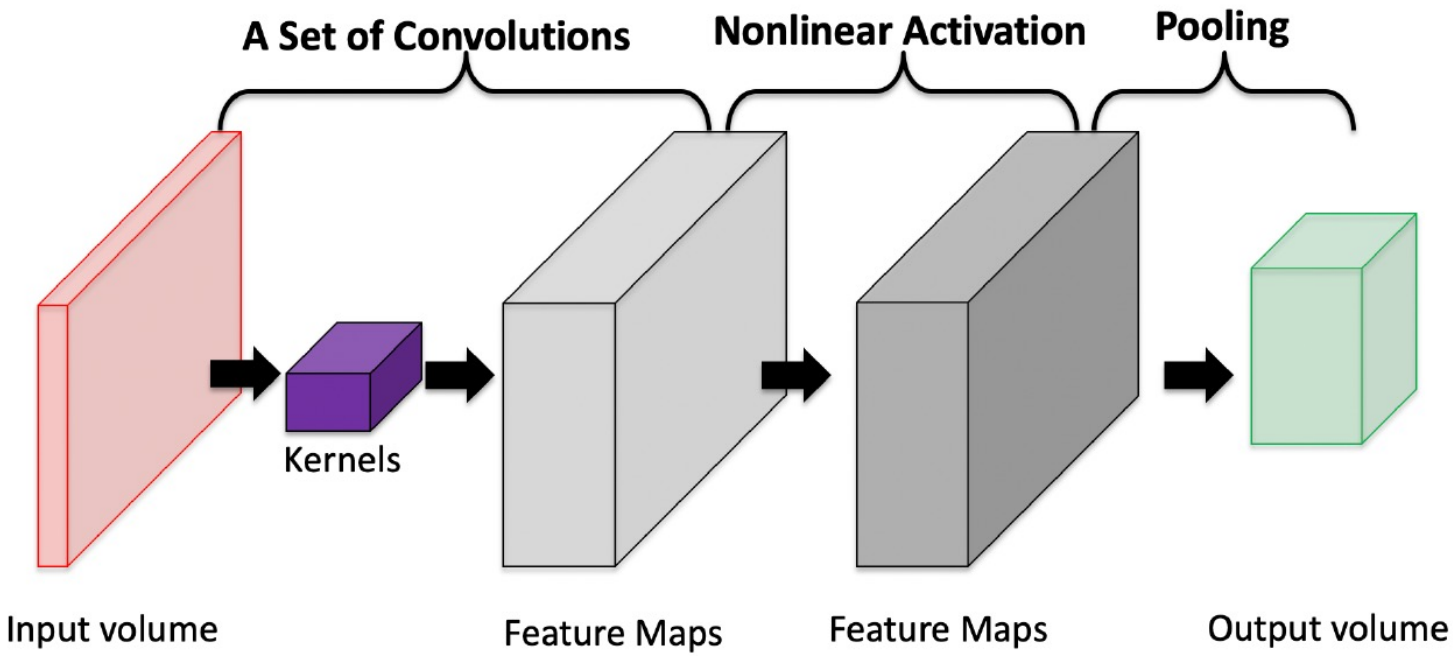
CNN: Pooling Layer

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



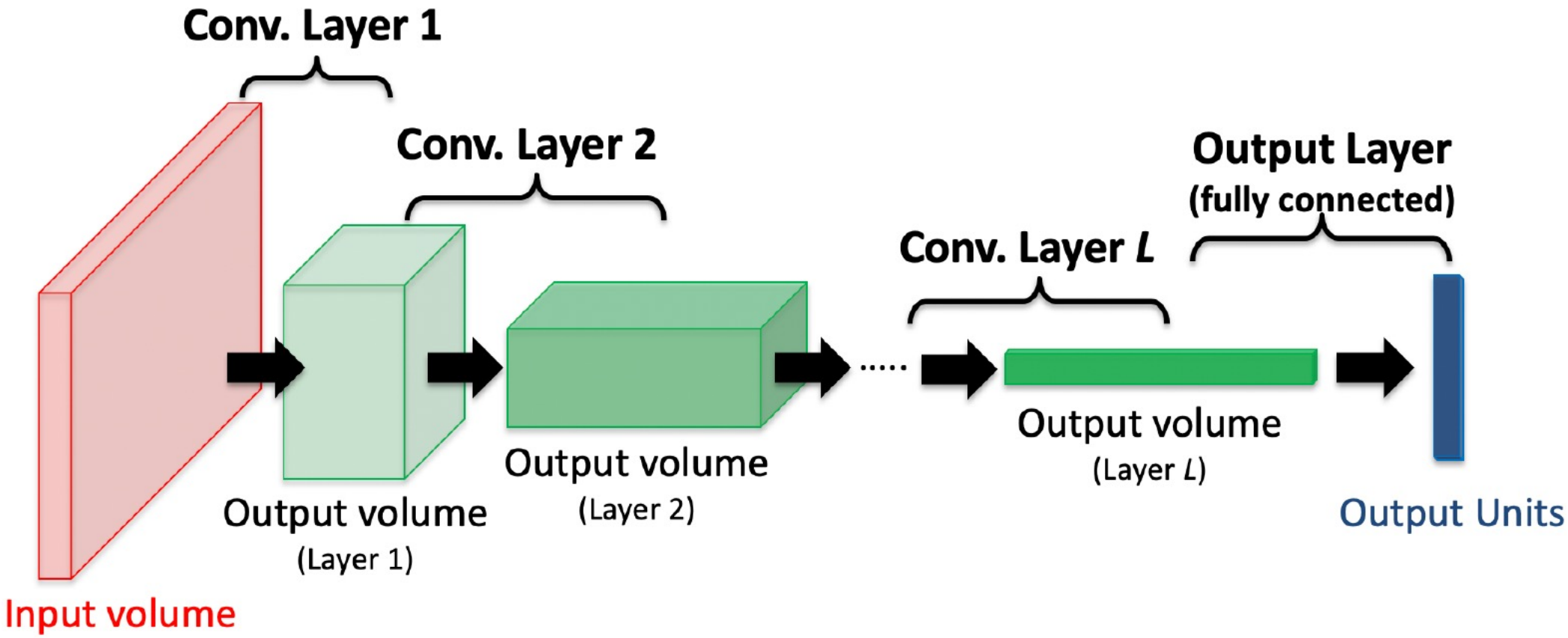
CNN: Convolutional Layer

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

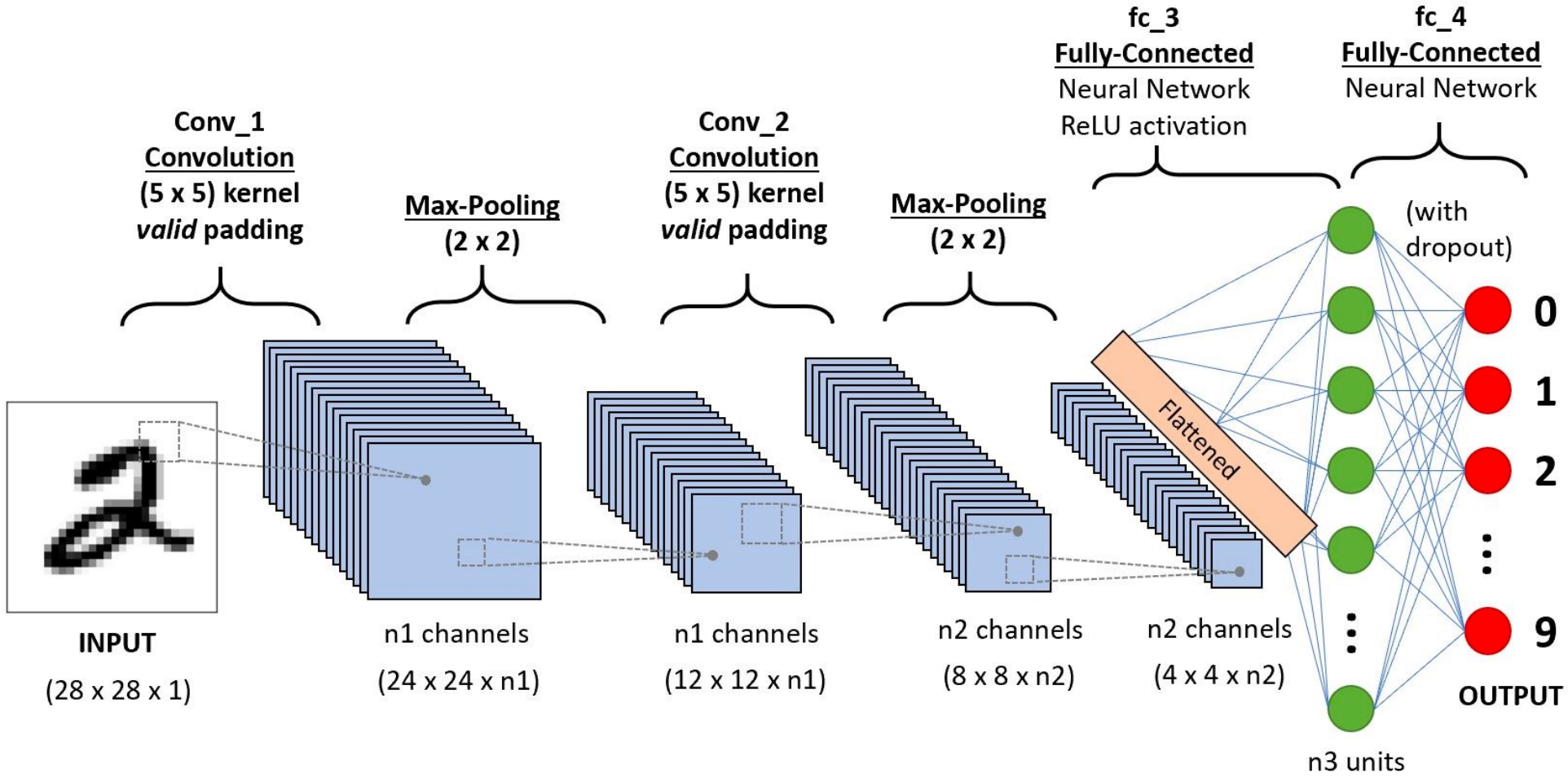


Convolutional Neural Networks

- 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

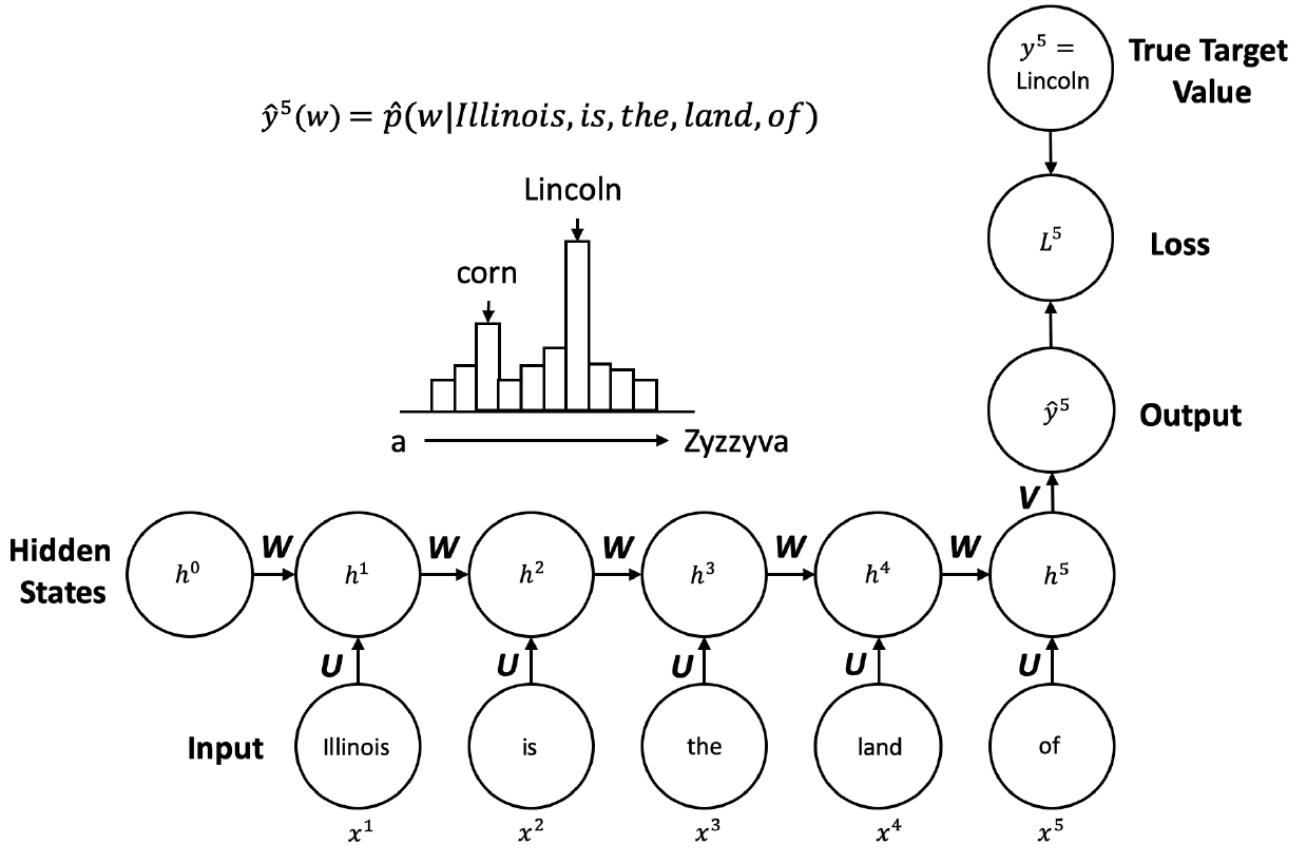
Chapter 11. Deep Learning

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

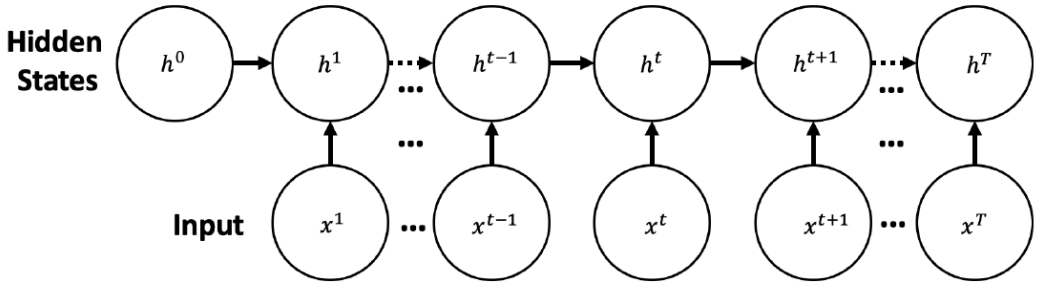
Recurrent Neural Networks

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

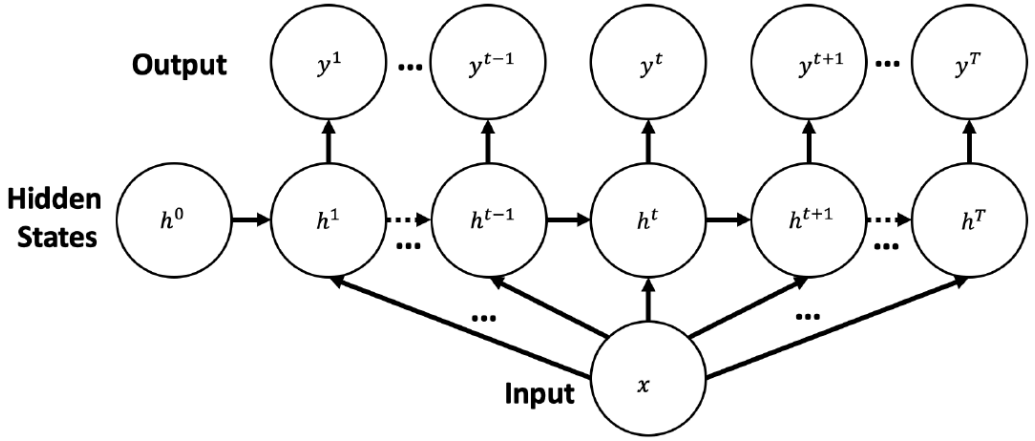
$$\boldsymbol{h}^t = f(\boldsymbol{U}\boldsymbol{x}^t + \boldsymbol{W}\boldsymbol{h}^{t-1} + \boldsymbol{a})$$
$$\hat{\boldsymbol{y}}^T = g(\boldsymbol{V}\boldsymbol{h}^T + \boldsymbol{b}).$$



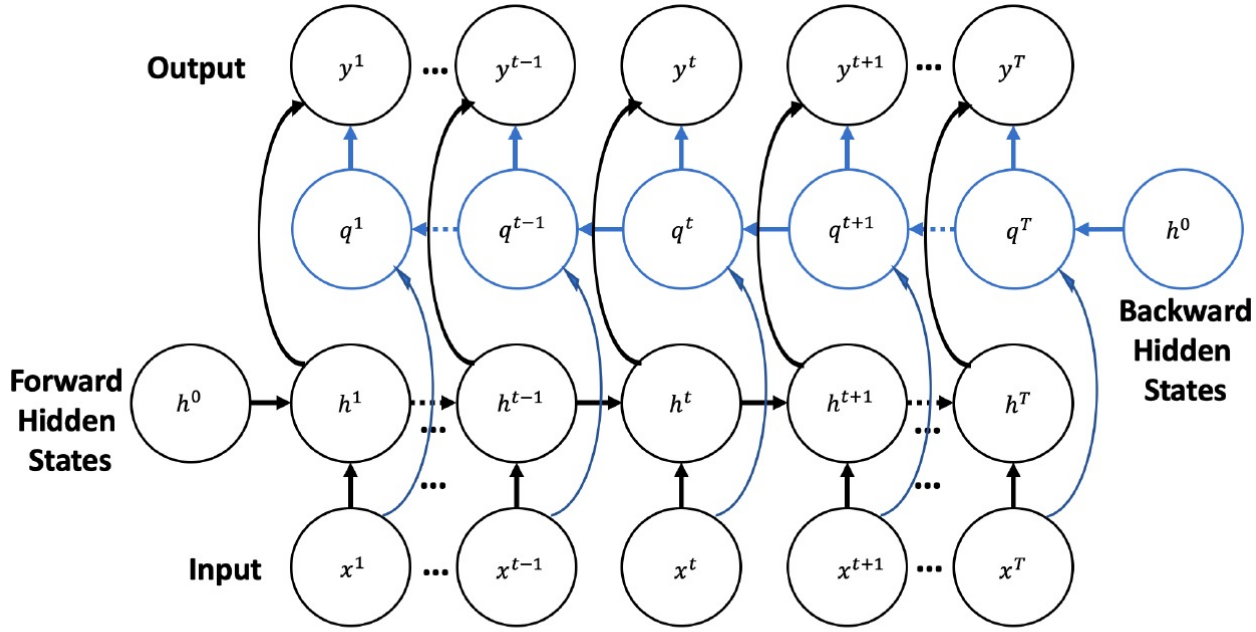
Different Types of RNNs



(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 Urbana Sweetcorn Festival is held in every August .... There will be live music, food, beers ... . Enjoy the fresh **corn**.”
  - Long-term dependence calls for deep RNN, which make gradient vanishing and/or exploding problems even worse.
  - Solutions:
    - Gated RNNs (LSTM, GRU)
    - Attention mechanism (Transformer, BERT)

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

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.