National workshop on
Applications of Deep Learning in Computer Vision
SSN Engineering College

Introduction to Deep Learning

S Milton Rajendram
Computer Science and Engineering
SSN College of Engineering
miltonrs@ssn.edu.in

10 February 2022

Logistic Regression

Linear classification

- $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$
- y is binary-valued (discrete-valued)
- Linear combination of features of $\mathbf{x}$

  $w_0 + w_1 x_1 + \ldots + w_d x_d = w_0 + \sum_{i=1}^{d} w_i x_i$

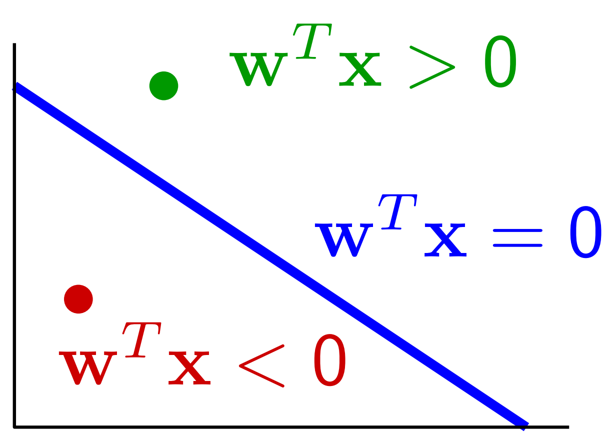
  $w_0 x_0 + w_1 x_1 + \ldots + w_d x_d = \sum_{i=0}^{d} w_i x_i,\ x_0 = 1$

- Vectors in matrix notation

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ x_d \end{bmatrix}$$

$$\hat{y} = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

$$\hat{y} = \text{sign}(\mathbf{w}^T\mathbf{x})$$



- Linear combination

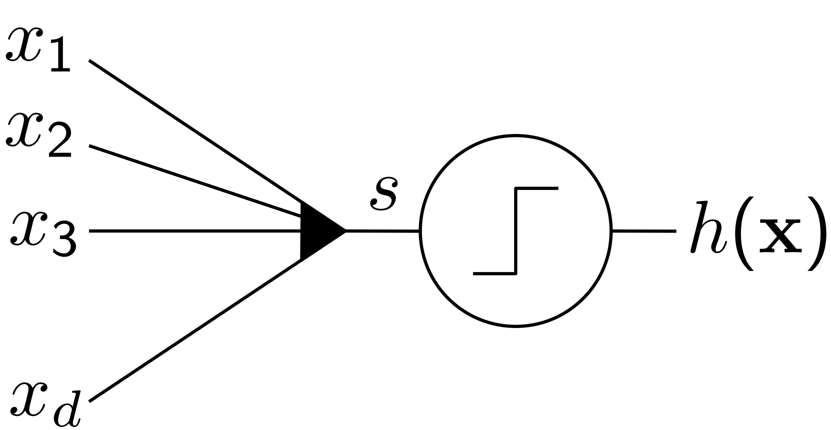
  $\sum_{i=0}^{d} w_i x_i = \mathbf{w}^T\mathbf{x} = \mathbf{w}.\mathbf{x}$

Linear models

$$
\begin{tikzcd}[column sep=huge]
\text{Input} & \text{Signal} & \text{Output} \\
\mathbf{x} \arrow[r] \arrow[rr, bend right, "h(\mathbf{x})"'] & \mathbf{s} = \mathbf{w}^T\mathbf{x} \arrow[r, "\theta(s) = \theta(\mathbf{w}^T\mathbf{x})"] & y
\end{tikzcd}
$$

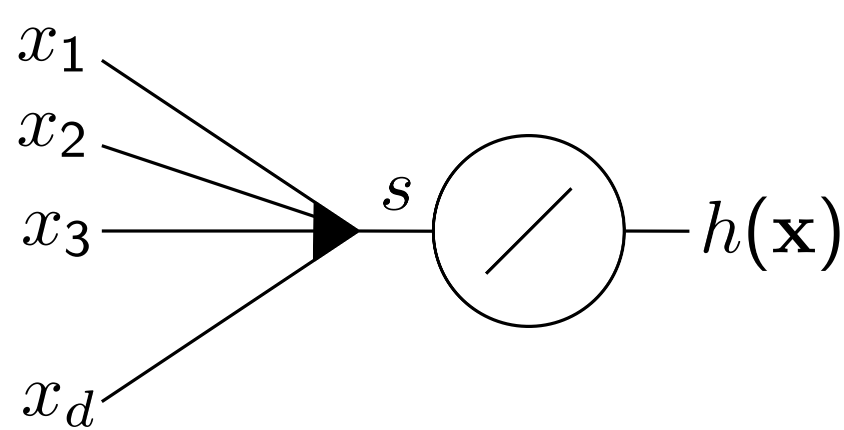
Linear classification

$h(\mathbf{x}) = \text{sign}\ (\mathbf{w}^T\mathbf{x})$



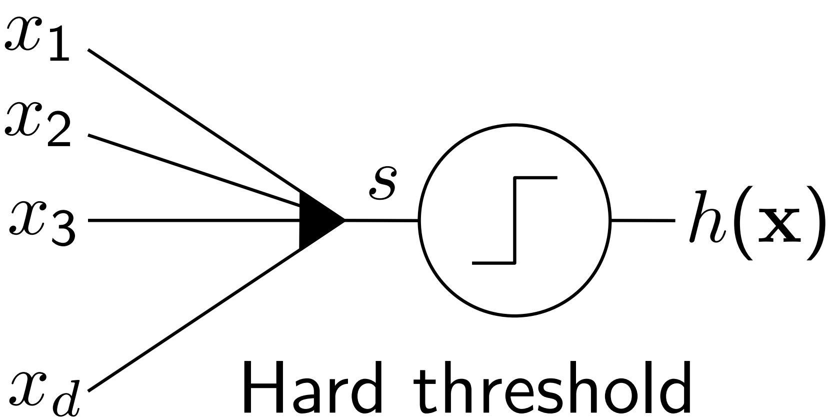
Linear regression

$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

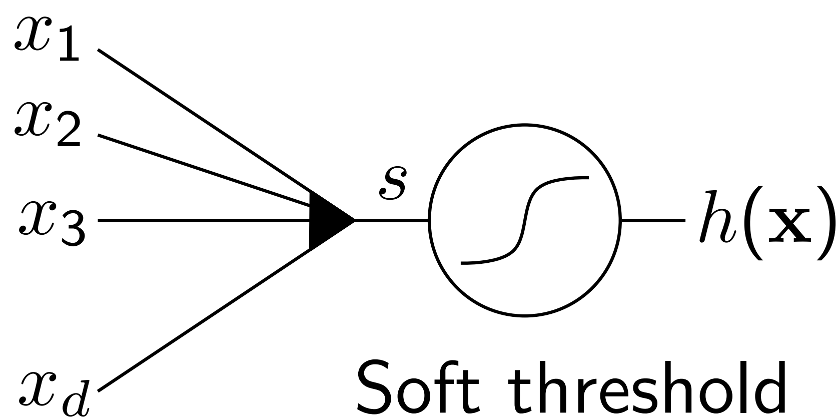


Logistic regression

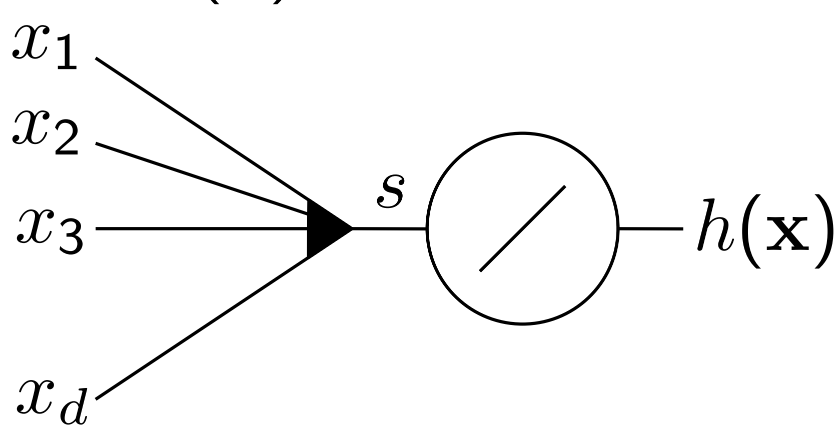
Linear classification
$h(\mathbf{x}) = \text{sign}\ (\mathbf{w}^T\mathbf{x})$



Logistic regression
$0 \leq h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x}) \leq 1$



Linear regression
$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

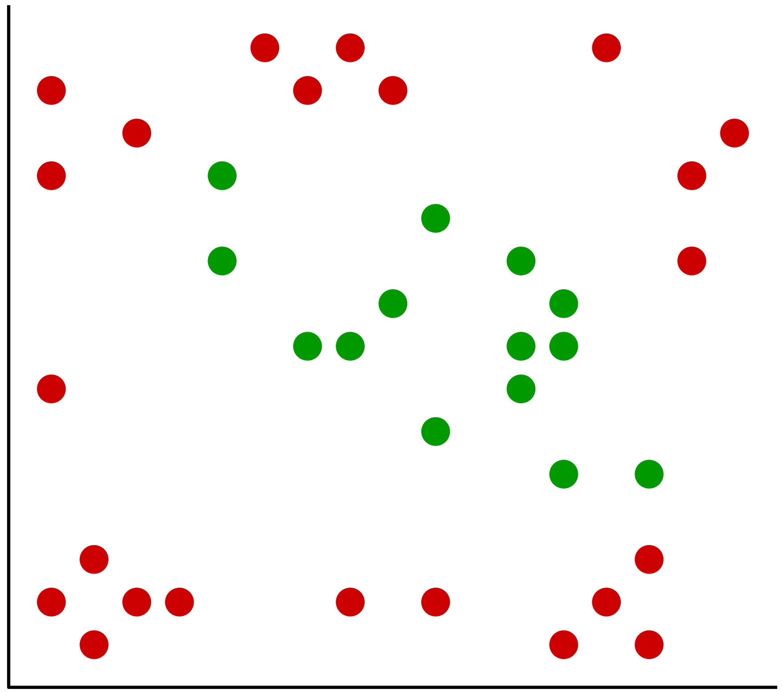


Logistic regression — overview

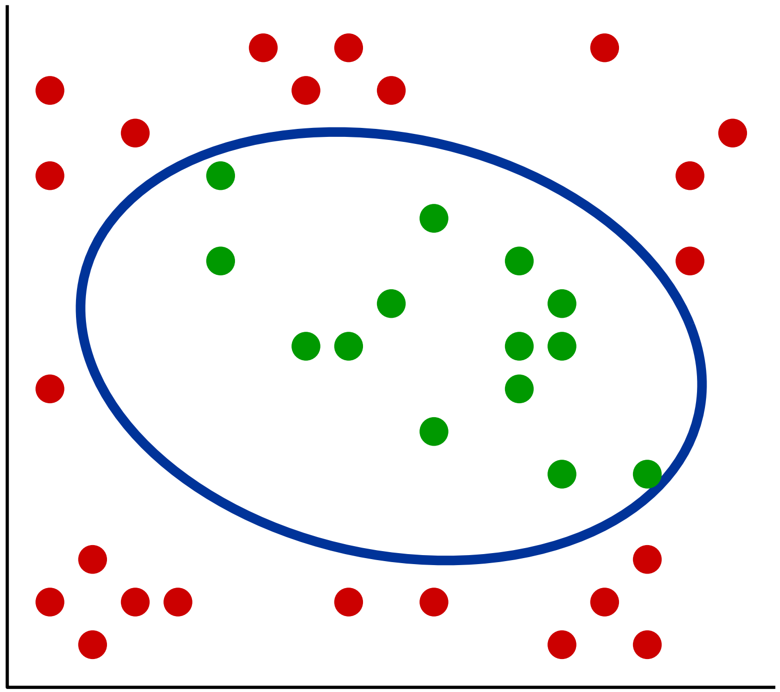
- $h(\mathbf{x}) = \dfrac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}} = P(y|\mathbf{x})$, hypothesis set
- $e(h(\mathbf{x}_i), y) = \ln(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i})$, cross-entropy error
- $E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ln(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i})$, in-sample error or likelihood
- Minimize in-sample error $E(\mathbf{w})$
- Gradient $\nabla_{\mathbf{w}}E(\mathbf{w})$, in which direction of $\mathbf{w}$ error decreases most rapidly and how much
- Gradient descent:
  $\mathbf{w}_{(t+1)} \leftarrow \mathbf{w}_{(t)} + \eta\nabla_{\mathbf{w}}E(\mathbf{w})$

Linear is limited

Data



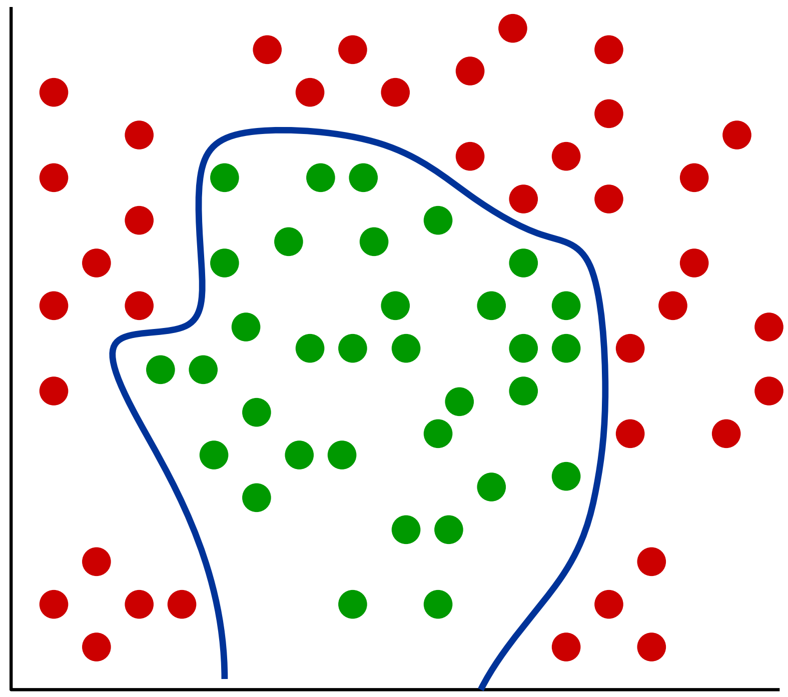
Hypothesis



- Credit limit is affected by 'years in residence'
- But not in a linear way!

Idea

- Generalization of perceptron.
- Can model complex (non-linear) target function.
- Fits the data by minimizing in-sample error E
- Flexible, but risk of overfitting.

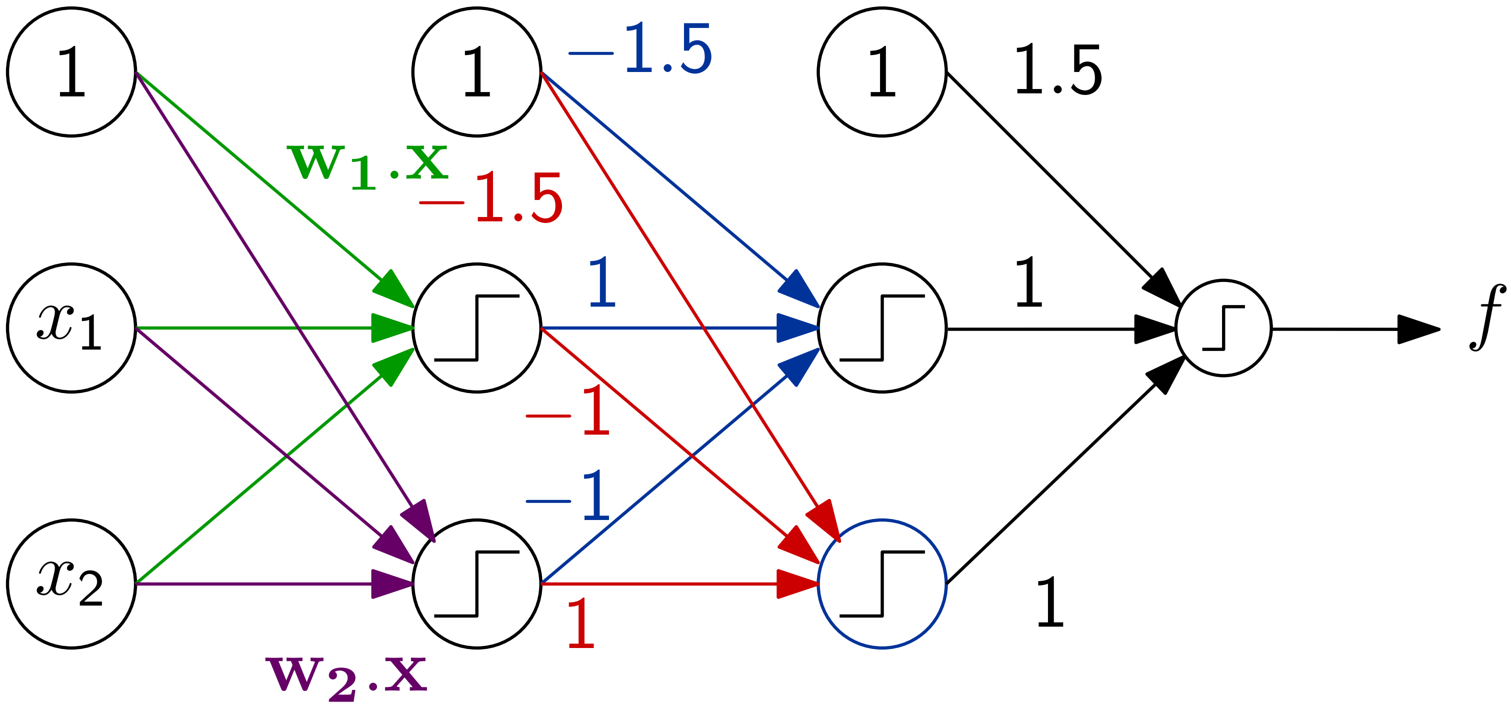


Neural Networks

Idea

- Generalization of perceptron.
- Can model complex (non-linear) target function.
- Fits the data by minimizing in-sample error E
- Flexible, but risk of overfitting.

Multi-layer perceptron (MLP) of XNOR



3-layer feed-forward

$$h_1(\mathbf{x}) = \mathbf{w}_1^T\mathbf{x} \qquad h_1(\mathbf{x}).\overline{h_2(\mathbf{x})} \qquad \overline{h_1(\mathbf{x}).\overline{h_2(\mathbf{x})}} +$$

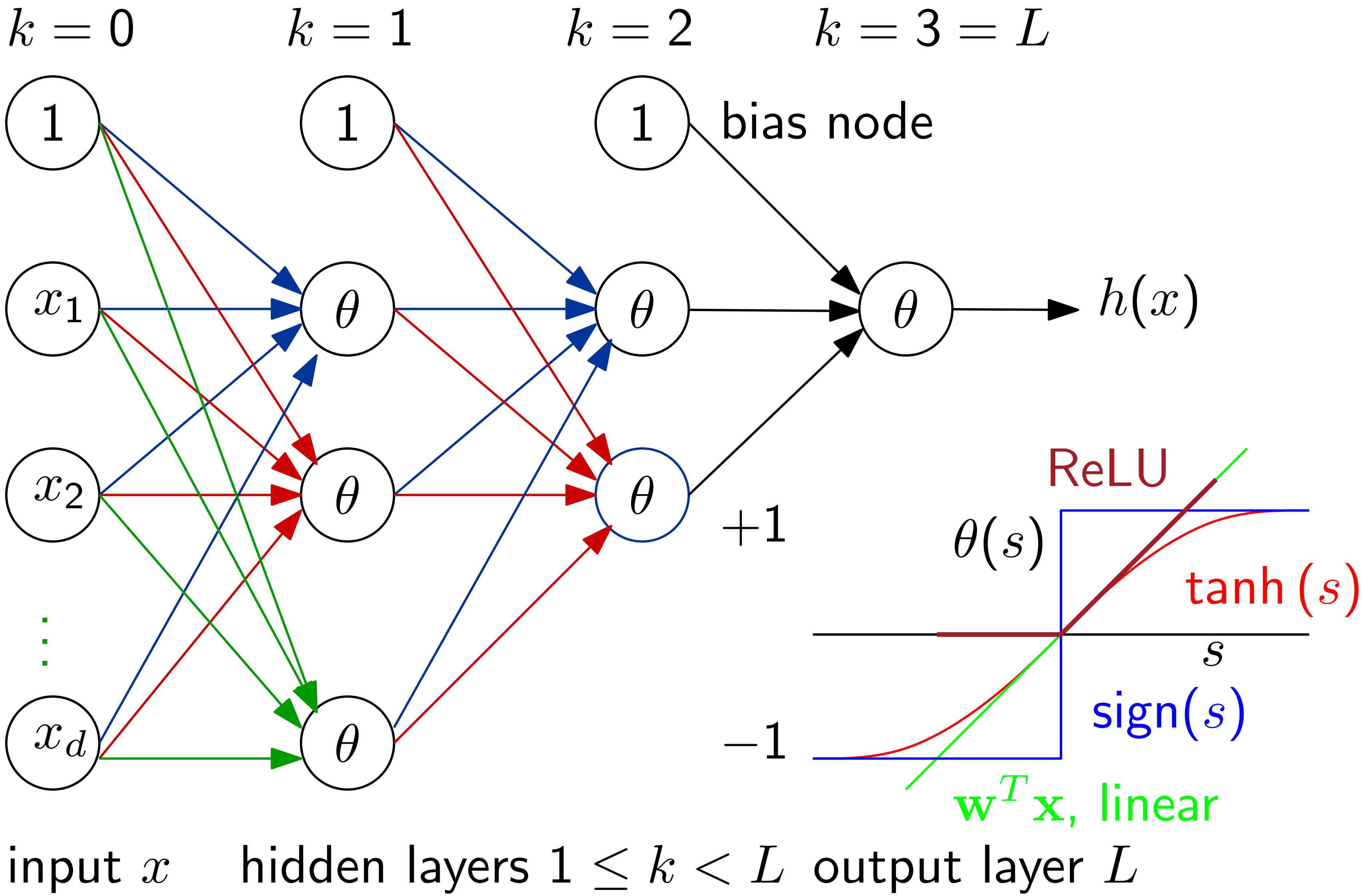
$$h_2(\mathbf{x}) = \mathbf{w}_2^T\mathbf{x} \qquad \overline{h_1(\mathbf{x})}.h_2(\mathbf{x}) \qquad \overline{h_1(\mathbf{x})}.h_2(\mathbf{x})$$

- Between input and output, multiple hidden layers.

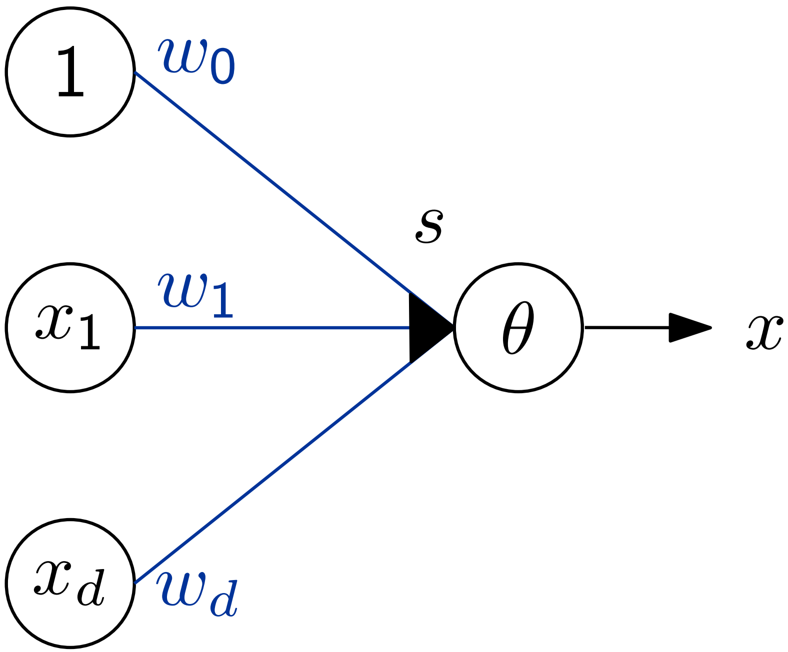
Feed-forward MLP

- Each layer feed-forward to the next layer only. Layer i to layer $i+1$ only.
- No feed-backward, no jump forward to other layers.
- Perceptron has just input and output.
- Multi-layer perceptron
  - input layer with d_{in} nodes,
  - output layer with d_{out} node,
  - 2 hidden layers with 3 nodes (hidden units) each.
- Architecture of MLP: number of hidden layers and number of hidden units in each layer.
- A 3-layer MLP with suitably many hidden units can model any target function — can fit any data.
- May overfit! May not generalize.

Neural network



Activation function



- Desirable that activation functions are differentiable, and
- Expressed in terms of the function itself.

- Tanh: output $\theta(s)$ $\theta(s) = \tanh(s) = \dfrac{e^s - e^{-s}}{e^s + e^{-s}}$
- ReLU: output $\theta(s) = \max(s, 0)$
- Sigmoid: output $\theta(s) = \dfrac{1}{1 + e^{-s}}$

Output-input for one layer

$$\mathbf{x}^{(k-1)} \xrightarrow{W^{(k)}} \qquad \mathbf{s}^{(k)} \xrightarrow{\theta} \qquad \mathbf{x}^{(k)}$$

$$\begin{bmatrix} x_0^{(k-1)} \\ x_1^{(k-1)} \\ \vdots \\ x_R^{(k-1)} \end{bmatrix} \begin{bmatrix} w_{01} & w_{02} & \dots & w_{0S} \\ w_{11} & w_{12} & \dots & w_{1S} \\ w_{21} & w_{22} & \dots & w_{2S} \\ \vdots & & & \\ w_{R1} & w_{R2} & \dots & w_{RS} \end{bmatrix} \qquad \begin{bmatrix} s_0^{(k)} \\ s_1^{(k)} \\ \vdots \\ s_S^{(k)} \end{bmatrix} \qquad \begin{bmatrix} x_0^{(k)} \\ x_1^{(k)} \\ \vdots \\ x_S^{(k)} \end{bmatrix}$$

$$R \times 1 \qquad R \times S \qquad S \times 1 \qquad S \times 1$$

$$(\mathbf{s}^{(\mathbf{k})})^{\mathbf{T}} = (\mathbf{x}^{(\mathbf{k-1})})^{\mathbf{T}}.\mathbf{W}^{(\mathbf{k})}$$

$$\mathbf{x}^{(\mathbf{k})} = \theta(\mathbf{s}^{(\mathbf{k})})$$

Forward propagation idea

$$\mathbf{x}^{(0)} \xrightarrow{W^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{W^{(2)}} \mathbf{y}$$

$$\begin{bmatrix} x_0^{(0)} \\ x_1^{(0)} \\ \vdots \\ x_{D_1}^{(0)} \end{bmatrix} \quad \begin{bmatrix} s_0^{(1)} \\ s_1^{(1)} \\ \vdots \\ s_H^{(1)} \end{bmatrix} \quad \begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ \vdots \\ x_H^{(1)} \end{bmatrix} \quad \begin{bmatrix} y_0^{(2)} \\ y_1^{(2)} \\ \vdots \\ y_{D_2}^{(2)} \end{bmatrix}$$

$$\mathbf{s} = \mathbf{x}.\mathbf{W}^{(1)}$$
$$\mathbf{h} = \theta(\mathbf{s})$$
$$\mathbf{y} = \mathbf{h}.\mathbf{W}^{(2)}$$

$$\mathbf{x}_{(1\times D_1)} \xrightarrow{W1_{(D_1\times H)}} \mathbf{s}_{(\mathbf{1}\times\mathbf{H})} \xrightarrow{\theta} \mathbf{h}_{(\mathbf{1}\times\mathbf{H})} \xrightarrow{W2_{(H\times D_2)}} \mathbf{y}_{(\mathbf{1}\times\mathbf{D_2})}$$

$$\underset{D_1\times 1}{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{D_1} \end{bmatrix}} \quad \underset{H\times 1}{\begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_H \end{bmatrix}} \quad \underset{H\times 1}{\begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_H \end{bmatrix}} \quad \underset{D_2\times 1}{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{D_2} \end{bmatrix}}$$

Training error

- Training error

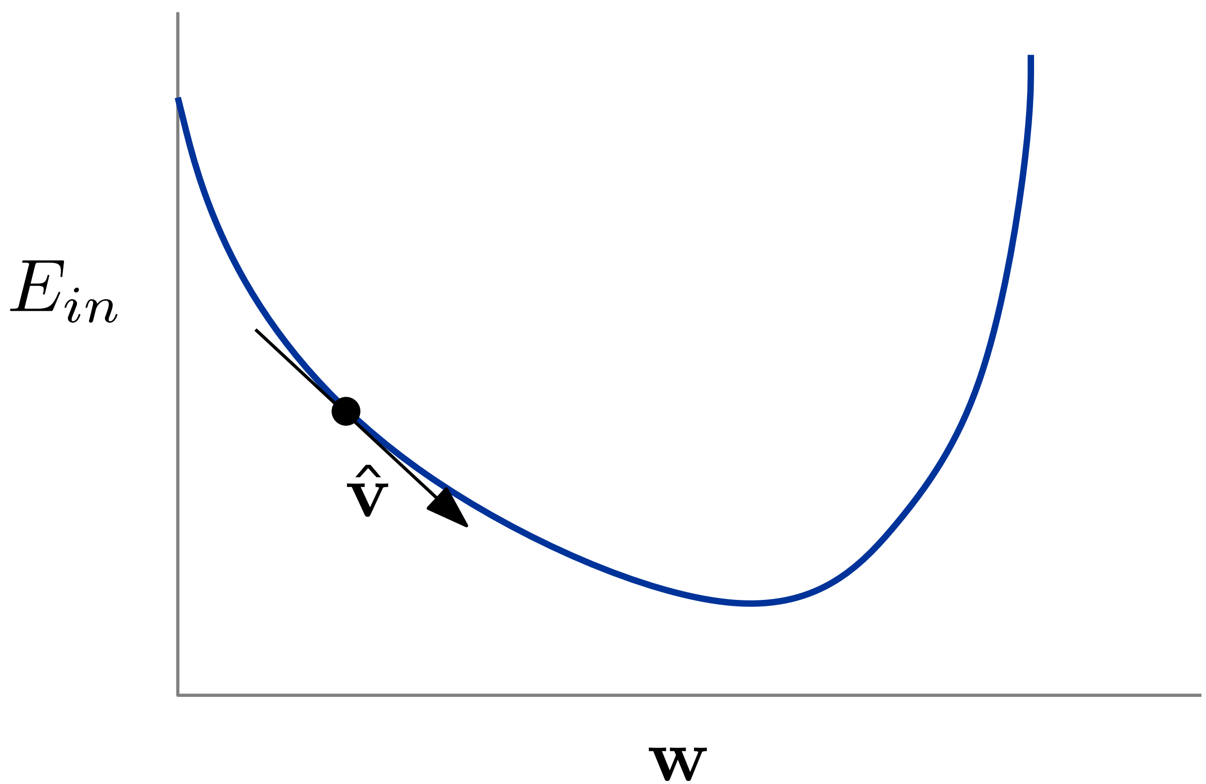
$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} e_i$$

$$= \frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

- To minimize $E(\mathbf{w})$, solve $\nabla E(\mathbf{w}) = 0$ for $\mathbf{w}$.

Iterative method: gradient descent

- General method for non-linear optimization
- Start at $\mathbf{w}_{(0)}$, descend the surface along the steepest slope
- Let $\hat{\mathbf{v}}$ (unit vector) be the direction of steepest slope.
- Take fixed-size steps along $\hat{\mathbf{v}}$: $\mathbf{w}_{(t+1)} = \mathbf{w}_{(t)} + \eta\hat{\mathbf{v}}$



Stochastic Gradient Descent (SGD)

1. Initialize $\mathbf{w}_{(0)}$
2. for $t = 1, 2, 3, \ldots$ until $\mathbf{w}$ converges do
3. for $i = 1 \ldots N$ do
4. Compute gradient
$$\nabla E(\mathbf{w})$$
5. Update weight: $\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) - \eta \nabla E(\mathbf{w}(t))$
6. Return $\mathbf{w}$

- For updating $\mathbf{w}$ once, it considers only one training example, chosen at random.
- Converges faster, but may oscillate.

Backpropagation idea

$$\text{Sensitivity of layer } k,\ \delta^{(k)} = \frac{\partial e}{\partial s^{(k)}}$$

- Partial derivatives in layer $k-1$ in terms of partial derivatives of layer k

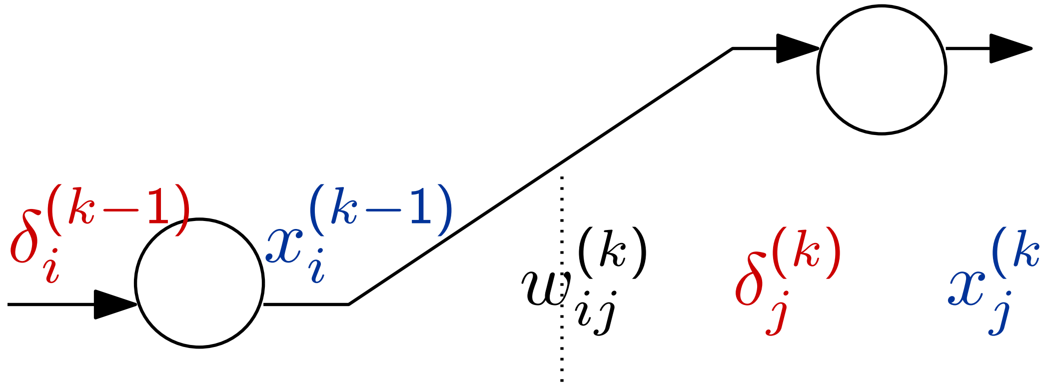
$$\frac{\partial e(\mathbf{w})}{\partial W^{(k)}} = x^{(k-1)} \times (\delta^{(k)})^T$$

$$\delta^{(k)} = \theta'(s^{(k)}) \otimes [W^{(k+1)} \times \delta^{(k+1)}]_1^{D_k}$$

$$\delta^{(L)} = 2(\hat{\mathbf{y}} - \mathbf{y})\theta'(\hat{\mathbf{y}})$$

Backpropagation algorithm

1: Initialize all weights w_{ij}^{k} at random
2: for $t = 1, 2 \ldots$ do
4: Forward: Compute all $x_j^{(k)}$
5: Backward: Compute all $\delta_j^{(k)}$
6: Update weights: $w_{ij}^{(k)} \leftarrow w_{ij}^{(k)} - \eta x_i^{(k-1)} \delta_j^{(k)}$
7: Return the final weights w_{ij}^{k}



Issues in Back-propagation algorithm

- It has been shown that finding weights to minimize error is NP-complete [Blum and Rivest, 1989]
- Not guaranteed to converge
- Over-fitting to the training samples
- Error minimization process can get trapped in a local minima
- Extremely slow convergence
- Selection of initial weights may be critical
- Selection of optimal parameters
- Extremely difficult to explain the model and its predictions

Deep Neural Networks

Large datasets

- Today, we have very large datasets available – especially for images, videos, speech, and text.
- However, traditional feature engineering and classifiers based on given-representations have saturated – "shallow" networks have relatively small model parameters to capture information content of large data sets.
- More than two hidden layers.
- Very deep neural networks: up to hundreds of layers.

Feature Engineering

- Major focus of machine learning has been selecting and extracting appropriate features
- Images: color, texture, edges, connected components, shapes, moments, SIFT, SURF, HOG, Wavelets-based, etc.
- Text: n-grams, character n-grams, POS tags, Named Entities, tags from shallow parsing, word sense disambiguation, etc.
- Speech: Mel Cepstral coefficients (MFCC), energy, zero crossing rates, pitch, timbre, etc.
- Which features are suitable for a given task? Leads to feature engineering.
- Which classifier is better? Ensemble of classifiers?
- Is a machine learning algorithm really "intelligent"?

Challenges with deep neural networks

- Vanishing gradients
- Extremely slow convergence
- Lack of generalization
- Lack of large data – number of model parameters increases with depth
- Lack of labeled data – supervised learning requires examples with corresponding targets
- Computational inefficiency

Vanishing gradient in backpropagation

- Network parameters are updated proportional to the partial derivative of the cost function w.r.t. the current parameters in each iteration of training.
- tanh has gradients in the range (0, 1)
- Chain rule multiplies n of these small numbers to compute gradients of the leftmost layers in an n-layer network.
- Gradient decreases exponentially with n.
- Effectively preventing some parameters from changing their values.
- Earlier layers train very slowly, if at all.
- May completely stop the neural network from further training.
- Several improvements combined together: ReLU, LSTM, skip connections used in residual neural networks.

New deep learning ideas

- New activation functions
- Regularization methods
- Initialization methods
- Data augmentation
- Optimization techniques
- GPU-based and distributed algorithms

Activation Function: Rectifier

- Rectifier is a simple activation function defined as

$$f(x) = \max(0, x)$$

- Neuron using rectifier is popularly known as ReLU (Rectified Linear Unit)
- One variation of this is "softplus" function

$$f(x) = \ln(1 + ex)$$

$$f'(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

- ReLU converges faster than networks using traditional 'tanh' [Krizhevsky et al., 2012]
- Variations of ReLU

Activation Function: Softmax

- Squashes a k-dimension vector of real values to k-dimension vector of real values in the range (0,1) that add up to 1
- Converts a k-dimension vector to probability distribution over k different possible outcomes
- Useful in multi-class classification
- For example, softmax of [1,2,3,4,1,2,3] is [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]

Regularization methods: Dropout

- Set the output of a hidden neuron to 0 with a probability of 0.5 (effectively "drop" those neurons out).
- Dropped out neurons do not take part in forward pass and back-propagation.
- Hence, different architecture is sampled every time, but these networks share weights.
- Reduces the complex co-adaption of neurons and reduces the model parameters to be learned.
- However, dropout almost doubles the number of iterations required.

Data augmentation

- We may not have enough data for learning a large number of model parameters (especially labeled data)
- Augment the existing data with some "label-preserving" operations
  - Image translations and horizontal reflections. Given, 256x256 images, extract random 224x224 patches and their horizontal reflections – object labels do not change!
  - Alter the intensities of RGB channels – one idea may be to perform PCA to find principal components and add a magnitude of this to the intensities.

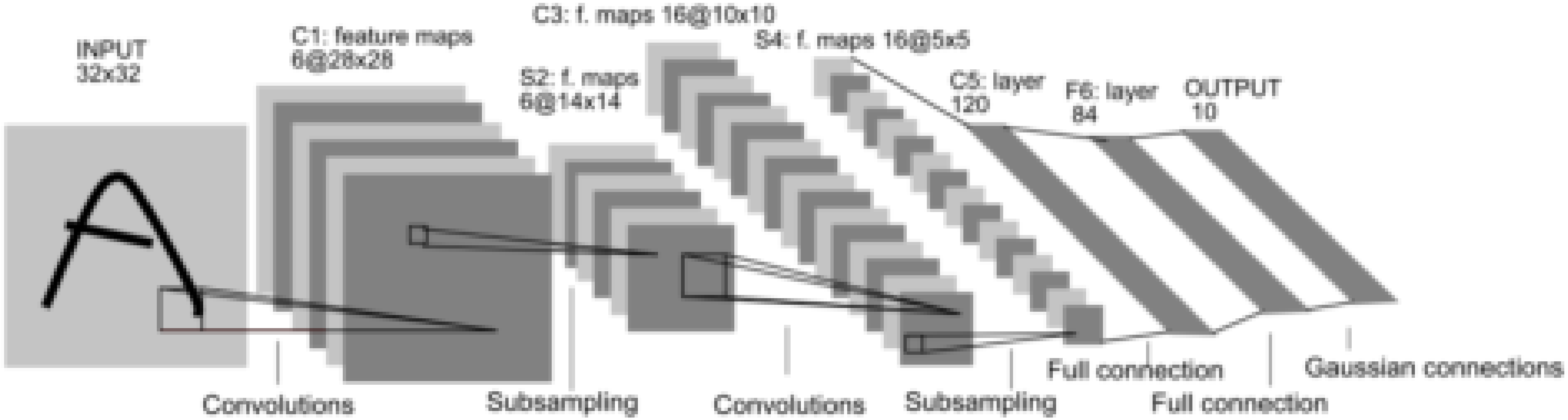
Stochastic Gradient Descent

- Stochastic Gradient Descent (SGD) incrementally updates the weights for each training example.
- This may result in wide swings the weights (which may be good sometimes!)
- To overcome this problem, learning rate is slowly decreased as the learning progresses.
- In other words, fluctuations are permitted initially, but controlled with the progress of the iterations – simulated annealing idea.
- It is possible to take best of both worlds and perform mini-batch updates – gradients are calculated for a batch of n training examples.
- Mini-Batch updates provide better exploitation of GPU and distributed computing

Challenges in Mini-Batch SGD

- How to choose appropriate learning rate? A high value may lead to undesirable fluctuations.
- What may a better schedule for reducing the learning rate? Extremely slow schedule may lead us to a better minima, but then convergence will also be extremely slow.
- Should all the parameters have the same learning rate?

LeNet5



- Trained using 32x32 pixel size images from MNIST (at most 20x20 characters centered in 28*28)
- Cx are convolutional layers (C1, C3, C5)
- Sx are sumsampling (pooling) layers (S1, S4)
- One fully connected hidden layer (F6) with 84 neurons and an output layer with 10 neurons

Convolutional Neural Networks

High dimensional input

- Add one layer k with D_k neurons.
- Add $(D_{k-11} + 1) \times D_k)$ parameters – $W^{(k)}$ and bias.
- Optimization become intractable.
- Images input is very high-dimensional.
- Each pixel of an image is a feature. If image is 100 by 100 pixels, then there are 10,000 features.

Convolutional neural networks

- Reduces the number of parameters in a deep neural network with many units without losing too much in the quality of the model.
- CNNs have found applications in image and text processing.

Learn locally

- In images, pixels that are close to one another usually represent the same type of information: sky, water, leaves, fur, bricks.
- Exception from the rule are the edges: the parts of an image where two different objects “touch” one another.
- Train the network to recognize regions of the same information as well as the edges $\Rightarrow$ Can predict the object represented in the image.
- Split the image into square patches using a moving window approach.
- Learn multiple smaller regression models at once.
- Train small regression model for a square patch as input – learns to detect a specific kind of pattern in the input patch.
- One to detect the sky; another one for the grass, the third one for edges of a building.

Convolution

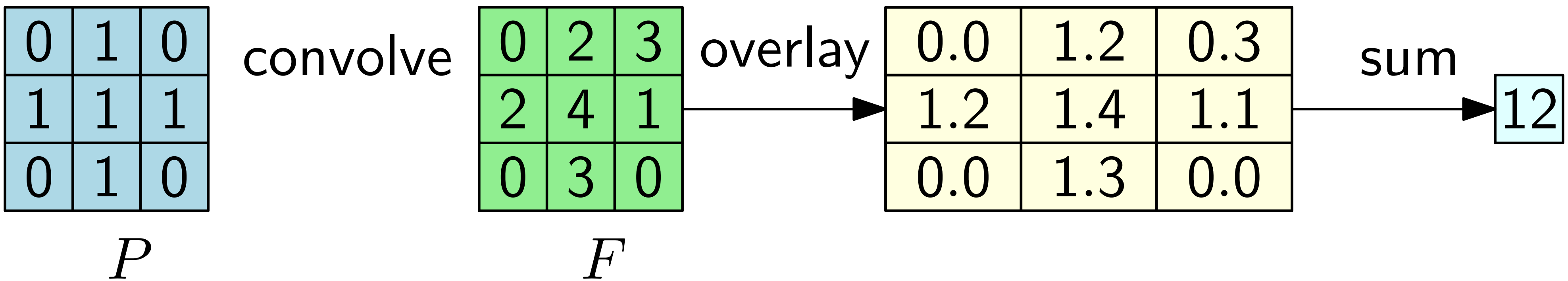
- Input image is made of black and white pixels.
- A patch is a 3×3.
- Learn a regression model to detect a cross pattern in patches.

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Learn a 3×3 parameter matrix $\mathbf{F}$ (filter): parameters corresponding to 1's will be positive and corresponding to 0's will be close to 0.

$$\mathbf{F} = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

Convolution between two matrices



$$\mathbf{P} \text{ convolution } \mathbf{F} = \mathbf{12}$$

- Higher the convolution value, the more similar $\mathbf{F}$ is to $\mathbf{P}$.
- A bias added before applying activation function.

$$\mathbf{P} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$

$$\mathbf{P} \text{ convolution } \mathbf{F} = \mathbf{9}$$

- Each filter slides (convolves) across the input image, left to right, top to bottom, and convolution is computed at each slide.

A filter convolves across an image

Conv 1

1	0	0	1
1	0	1	0
1	1	0	0
0	1	0	1

-1	2
4	-2

Filter

1

Bias

4		

Conv 2

1	0	0	1
1	0	1	0
1	1	0	0
0	1	0	1

-1	2
4	-2

1

4	-1	

Conv 3

1	0	0	1
1	0	1	0
1	1	0	0
0	1	0	1

-1	2
4	-2

1

4	-1	7

Conv 4

1	0	0	1
1	0	1	0
1	1	0	0
0	1	0	1

-1	2
4	-2

1

4	-1	7
2		

Conv 5

1	0	0	1
1	0	1	0
1	1	0	0
0	1	0	1

-1	2
4	-2

1

4	-1	7
2	7	

Conv 6

1	0	0	1
1	0	1	0
1	1	0	0
0	1	0	1

-1	2
4	-2

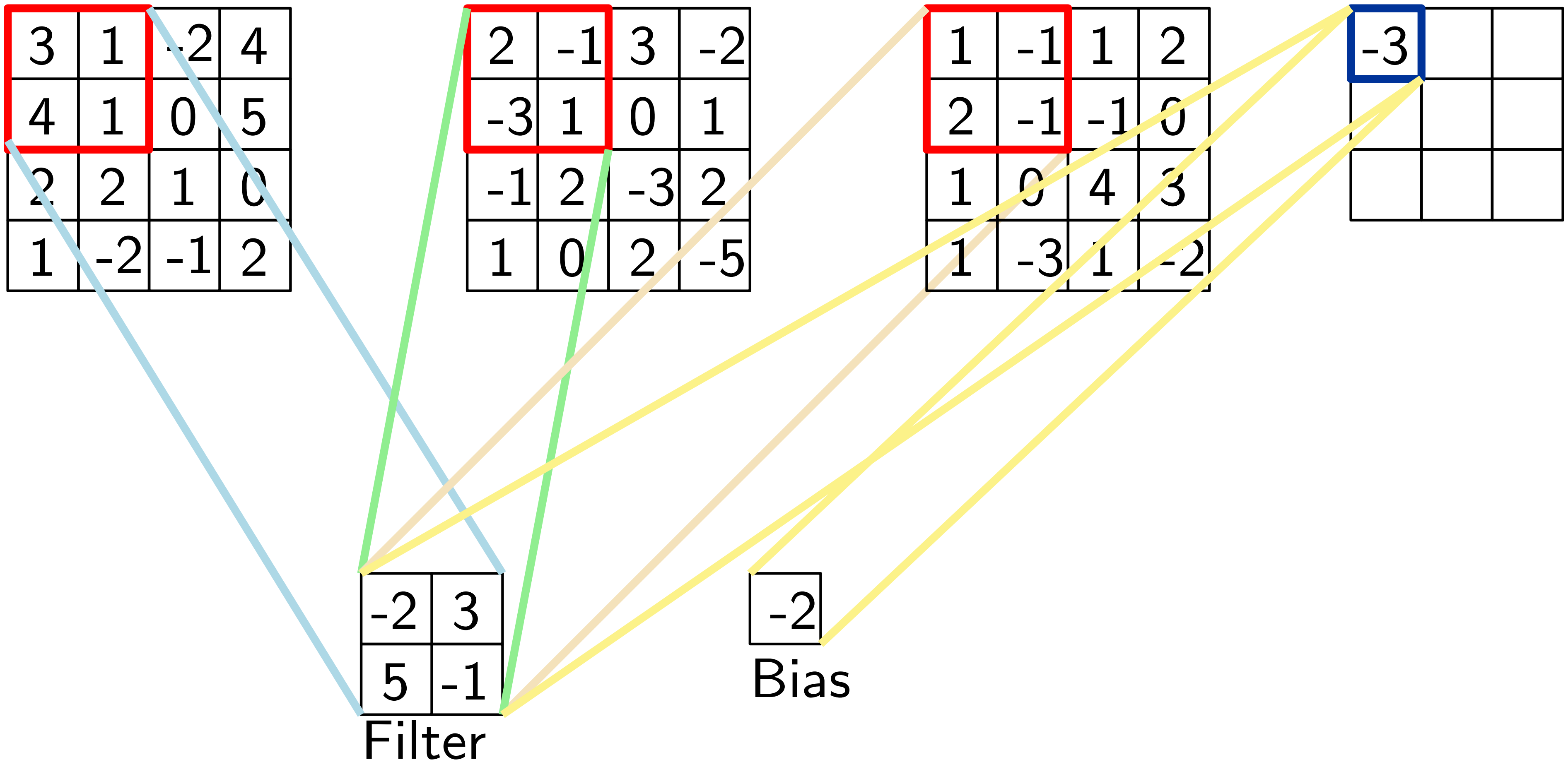
1

4	-1	7
2	7	6

Convolution layer

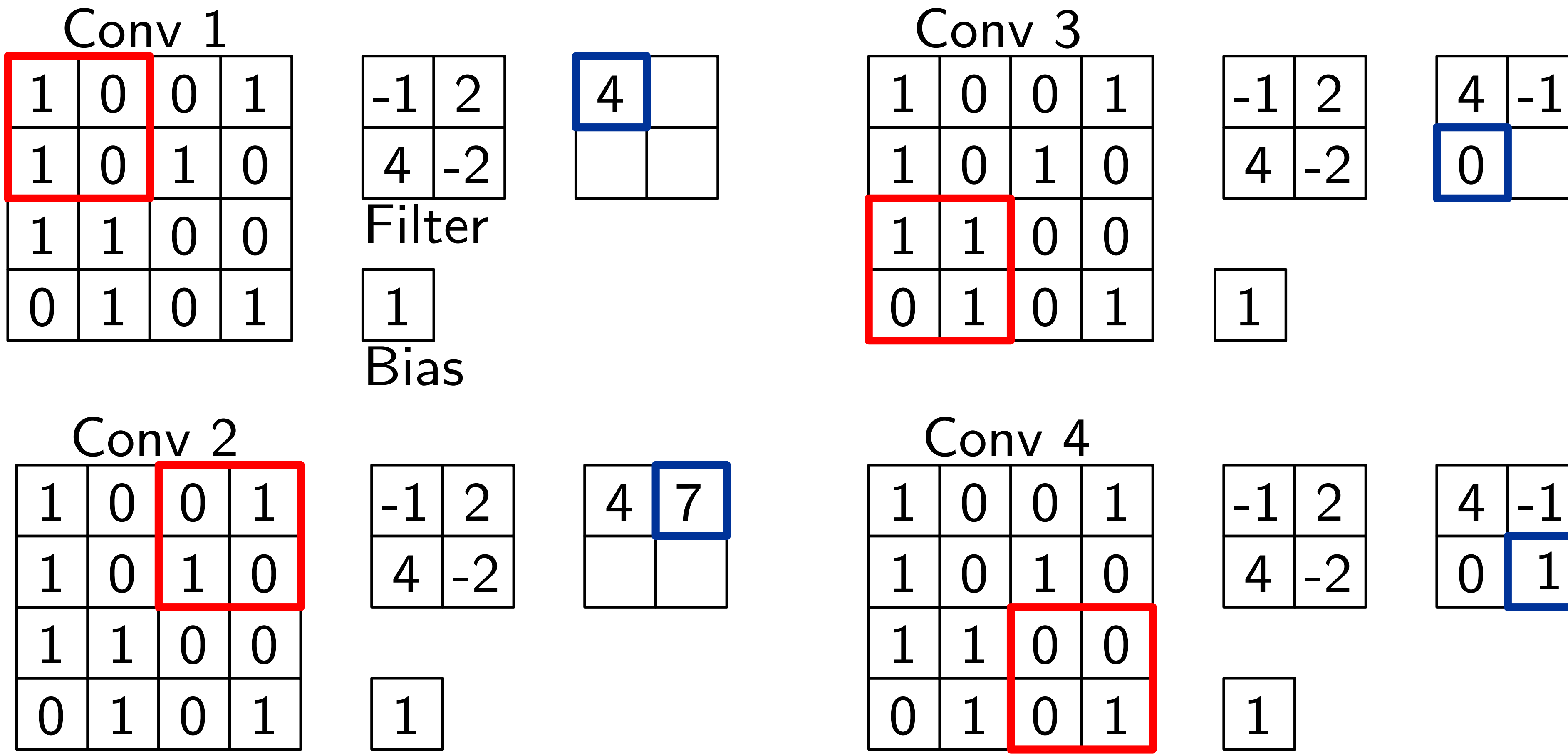
- One convolution layer made of multiple filters.
- Filter matrices and bias are parameters learned through gradient descent.
- ReLU activationfunction in convolution layer.
- Output layer activation function?
- In convolution layer k, each filter outputs one matrix, D_k filters output D_k matrices.
- If layer $k+1$ is convolutional, it treats the D_k matrices as D_k image matrices, volume of depth D_k.
- Convolution of a patch of a volume is the sum of convolutions of the corresponding patches of individual matrices of the volume.
- Input image is a volume of 3 channels: R, G, B.

Convolution of a volume



$$(-2.3 + 3.1 + 5.4 + -1.1) + (-2.2 + 3. - 1 + 5. - 3 + -1.1)+$$
$$(-2.1 + 3. - 1 + 5.2 + -1. - 1) + (-2)$$

Stride



- Step size of the sliding window.
- Larger the slide, the smaller the output matrix.

Padding

Conv 1

0	0	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	0
0	0	1	0	1	0
0	0	0	0	0	0

-1	2
4	-2

Filter

1

Bias

-1		

Conv 2

0	0	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	0
0	0	1	0	1	0
0	0	0	0	0	0

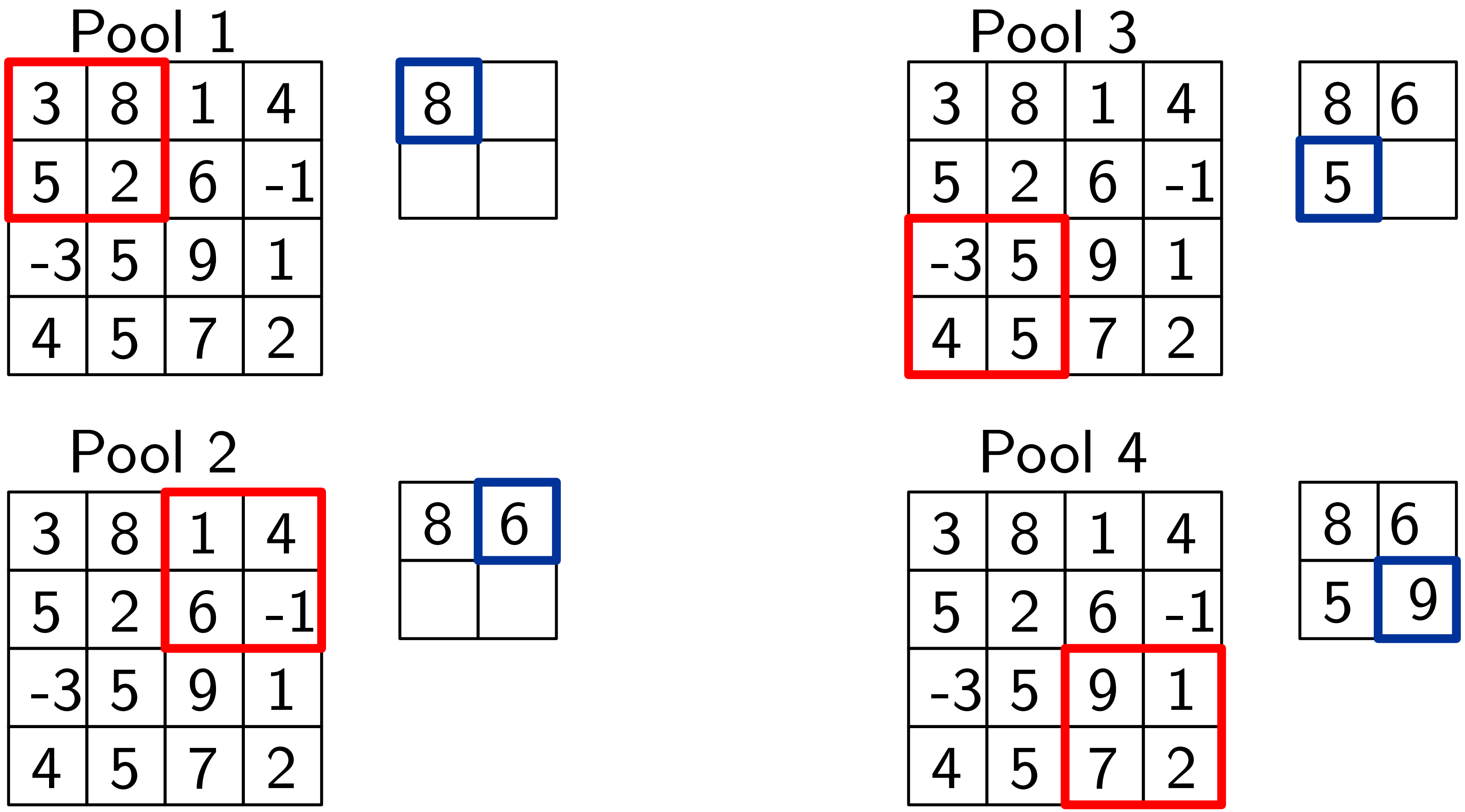
-1	2
4	-2

1

-1	0	

- Border around the image – how many cells wide?
- The larger the padding, the larger the output matrix.
- Helpful with larger filters – allows to better “scan” the boundaries of the image.

Pooling with filter size 2 and stride 2



- Pooling applies a fixed operator, max or average – no parameters to learn. Max pooling popular.
- Hyperparameters: filter size, stride. Usually, pooling layer follows a convolution layer.
- Reduces the number of parameters, speeds up training.

Recurrent Neural Networks

Recurrent Neural Networks (RNN)

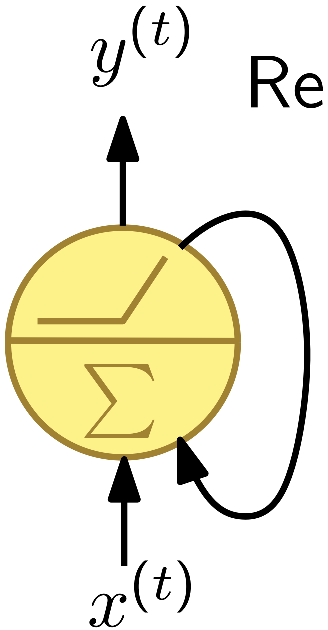
- Used to label, classify, or generate sequences.
- A sequence is a matrix: each row is a feature vector and the order of rows matters.
- To label a sequence is to predict a class for each feature vector in a sequence.
- To classify a sequence is to predict a class for the entire sequence.
- To generate a sequence is to output another sequence (of a possibly different length) somehow relevant to the input sequence.

Applications

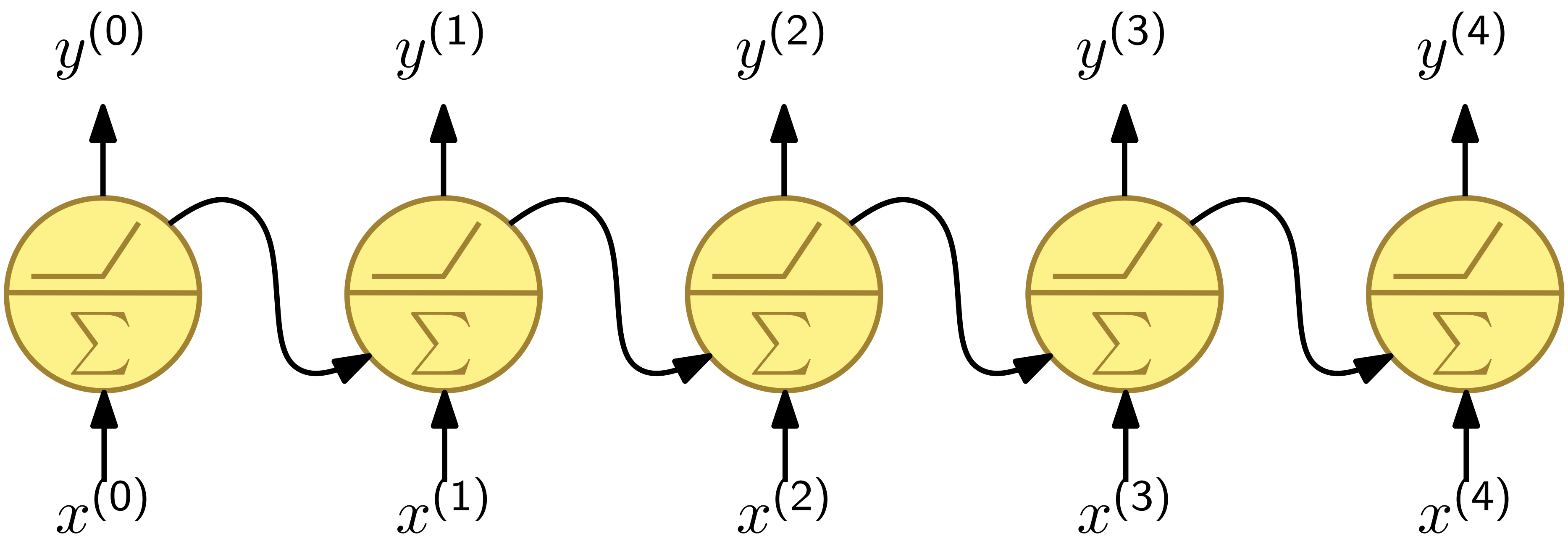
- Text processing
- Speech processing

Recurrent neurons

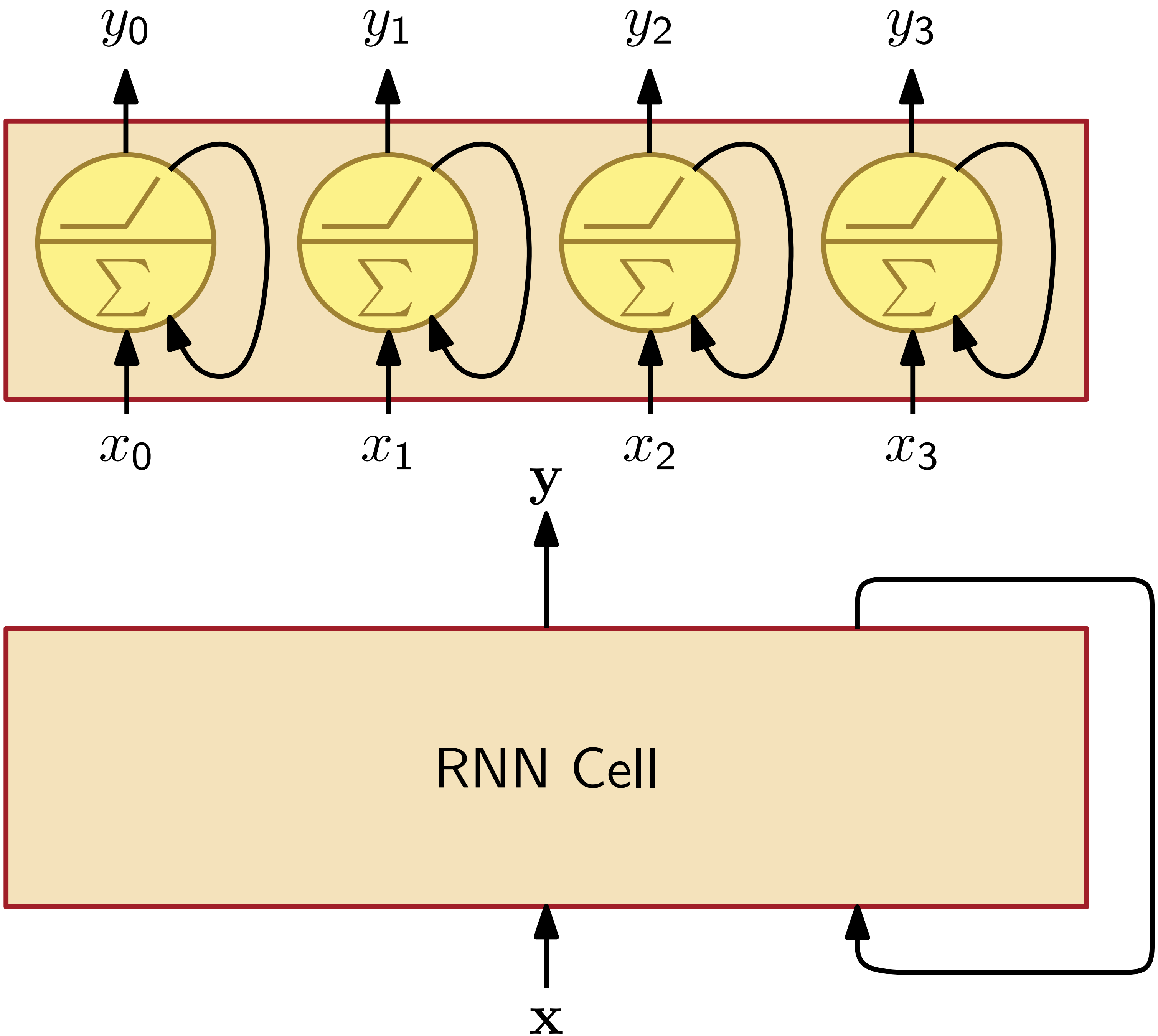
Recurrent neuron



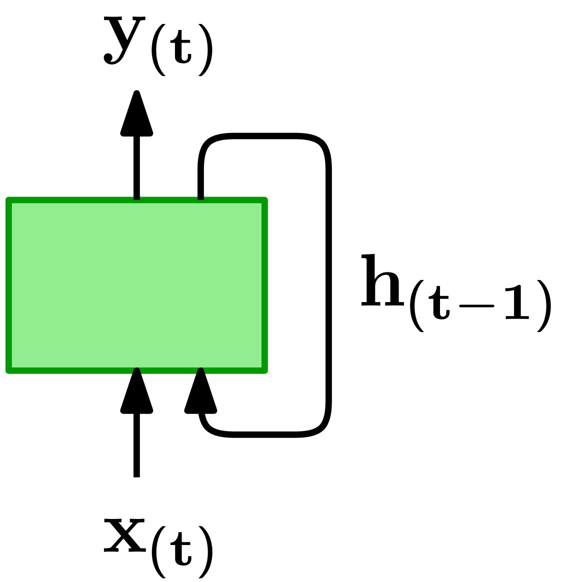
Recurrent neuron unrolled in time



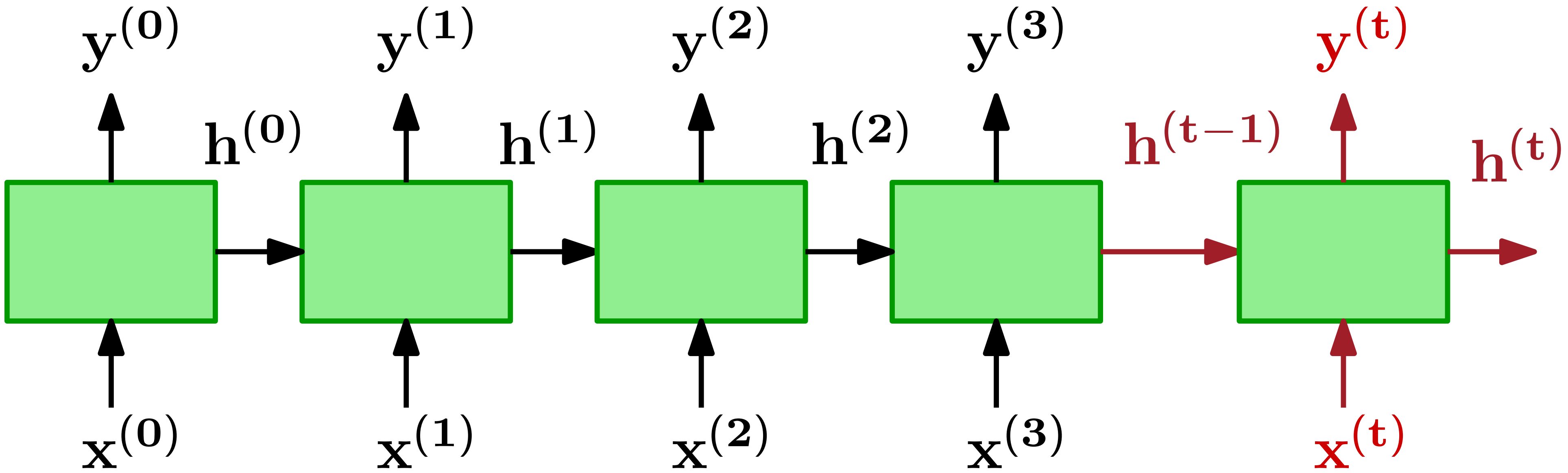
Recurrent layer



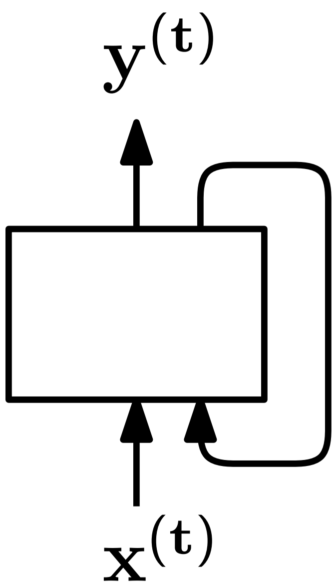
Memory cell



$$\mathbf{h}^{(\mathbf{t})} = f(\mathbf{h}^{(\mathbf{t-1})}, \mathbf{x}^{(\mathbf{t})})$$
$$\mathbf{y}^{(\mathbf{t})} = g(\mathbf{h}^{(\mathbf{t-1})}, \mathbf{x}^{(\mathbf{t})})$$



RNN



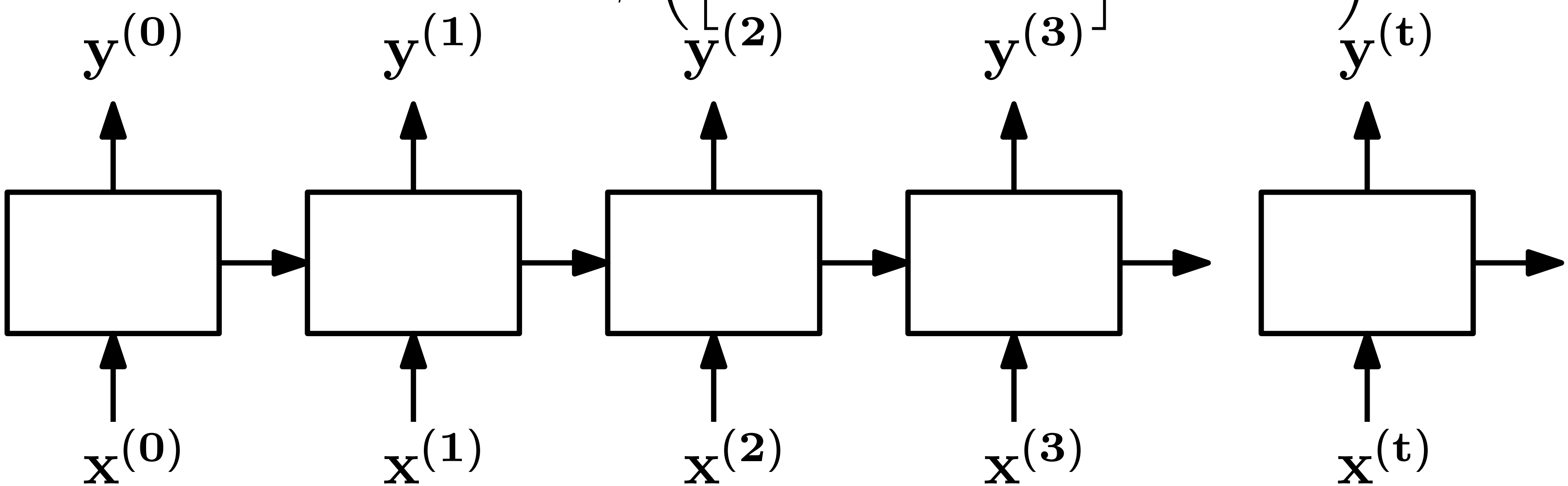
Output of a RNN layer for one example.

$$\mathbf{h}^{(\mathbf{t})} = \phi\left({\mathbf{x}^{(\mathbf{t})}}^{\mathbf{T}}.\mathbf{W}_{\mathbf{x}} + {\mathbf{h}^{(\mathbf{t-1})}}^{\mathbf{T}}.\mathbf{W}_{\mathbf{h}} + \mathbf{b}\right)$$

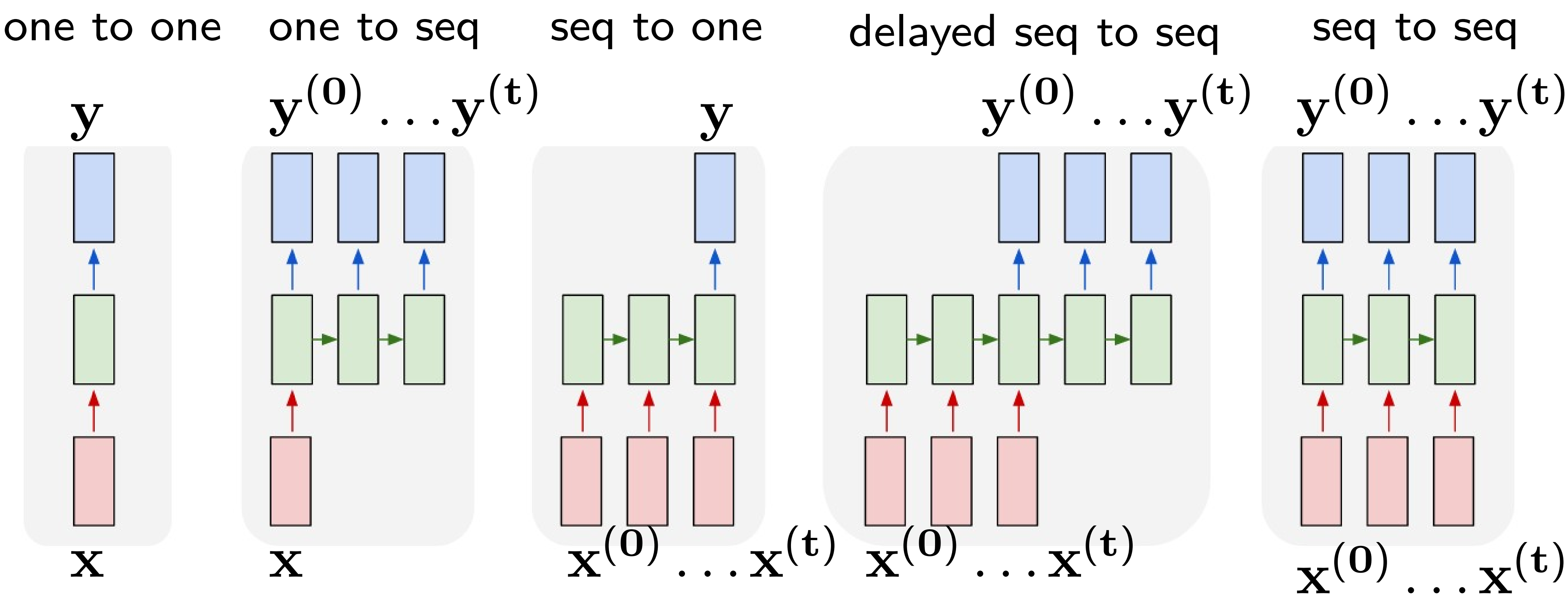
Output of a RNN layer for a mini-batch.

$$\mathbf{Y}^{(\mathbf{t})} = \phi\left(\mathbf{X}^{(\mathbf{t})}.\mathbf{W}_{\mathbf{x}} + \mathbf{Y}^{(\mathbf{t-1})}.\mathbf{W}_{\mathbf{h}} + \mathbf{b}\right)$$

$$= \phi\left(\left[\mathbf{X}^{(\mathbf{t})} + \mathbf{Y}^{(\mathbf{t-1})}\right].\mathbf{W} + \mathbf{b}\right)$$



Input Output sequences



- one to one: Image classification
- one to seq : Image captioning
- seq to one: Text classification
- delayed seq to seq: Translation
- seq to seq: Time series prediction

Difficulties

- Propagation through time is used to compute the parameters.
- Because of the sequential nature of the input, backpropagation has to "unfold" the network over time.
- The longer the input sequence, the deeper is the unfolded network.
- Even if our RNN has just one or two recurrent layers, both tanh and $softmax$ suffer from the vanishing gradient problem.
- Long-term dependencies.
- The feature vectors from the beginning of the sequence tend to be "forgotten", because the state of each unit becomes significantly affected by the feature vectors read more recently.
- In text or speech processing, the cause-effect link between distant words in a long sentence can be lost.

Gated RNN

- Long short-term memory (LSTM) networks.
- Networks based on the Gated Recurrent Unit (GRU).

- Units make decisions about what information to store, and when to allow reads, writes, and erasures.
- Those decisions are learned from data and implemented through the concept of gates.
- There are several architectures of gated units.
- Minimal gated GRU and is composed of a memory cell, and a forget gate.
- Can store information in their units for future use.
- Reading, writing, and erasure of information stored in each unit is controlled by activation functions – values in the range (0, 1).

Tools

Python

- `https://www.python.org/`

- Very high-level data structures.
- Clean syntax.
- Eco-system (comes with batteries attached!)
- De Facto programming language of ML.
- A large collection of ML packages.
- Python 2.7, Python 3

Anaconda

- https://anaconda.org/

- Most popular Python data science platform.
- Leads open source projects like Anaconda, NumPy and SciPy that form the foundation of modern data science.

NumPy

- `http://www.numpy.org/`

Fundamental package for scientific computing with Python.

- A powerful N-dimensional array object.
- Sophisticated functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities.

scikit-learn

- `http://scikit-learn.org/stable/`

- Classification: SVM, Nearest neighbors, Random forests.
- Regression: SVR, Ridge regression, Lasso.
- Clustering: k-means, Spectral clustering.
- PCA, Feature selection, Non-negative matrix factorization.
- Model selection: Grid search, Cross validation, Metrics.
- Preprocessing: Feature extraction.

TensorFlow

- Open source software library for numerical computation using data flow graphs.
- Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.
- Deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.
- Originally developed by Google Brain Team for machine learning and deep neural networks research.
- General enough to be applicable in a wide variety of other domains as well.

Keras

- High-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
- Allows for easy and fast prototyping.
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

PyTorch

- Open source machine learning framework based on the Torch library.
- Applications such as computer vision and natural language processing.
- Developed by Facebook's AI Research lab (FAIR).
- Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

References

- Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, 2016
- Eli Stevens, Luca Antiga, Thomas Viehmann, “Deep Learning with PyTorch”, 2020

Questions?