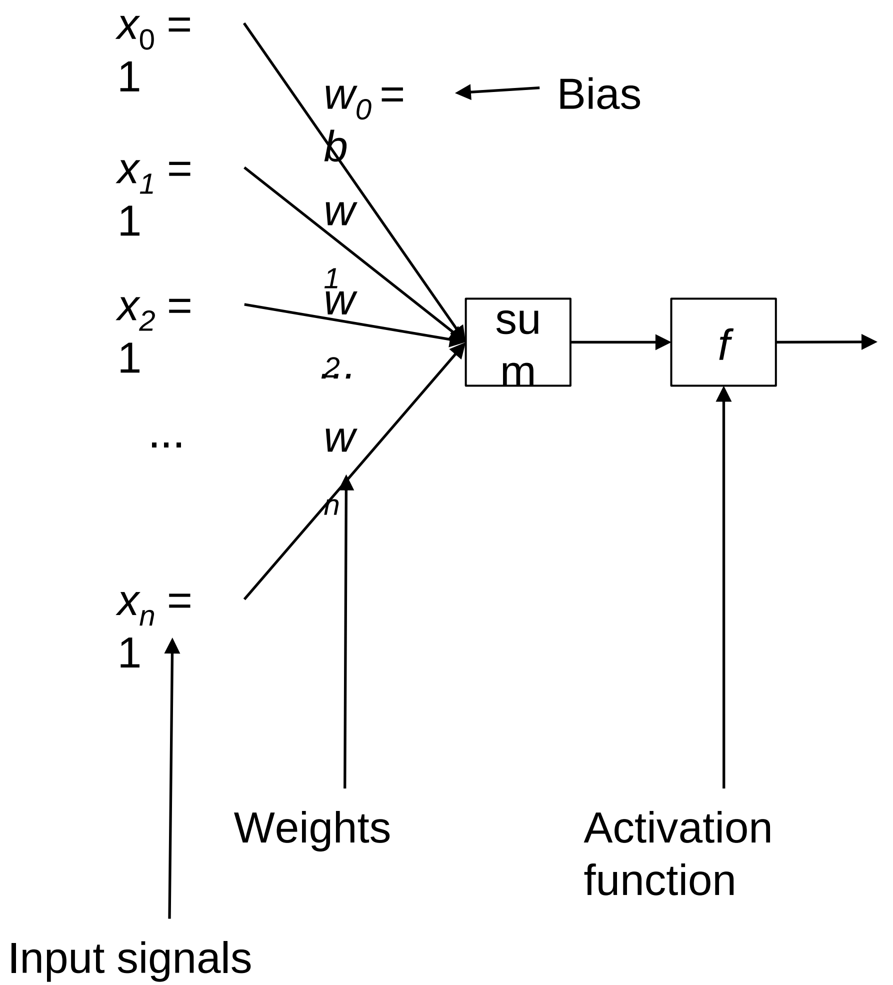
ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Lesson 2:
Machine Learning
(cont.)

7. Feedforward Neural Network

- Artificial Neural Network (ANN) simulates biological neural system, which is a network of interconnected artificial neurons. ANN can be considered as a distributed and parallel computing architecture
- Each neuron receives input signals, performs local computations to form the output signal. The output value depends on the characteristics of each neuron and its connections with other neurons in the network.
- ANNs perform learning, memory, and generalization by updating the weights of connections between neurons.
- The objective function depends on the network architecture, the characteristics of each neuron, the learning strategy, and the learning data

Perceptron

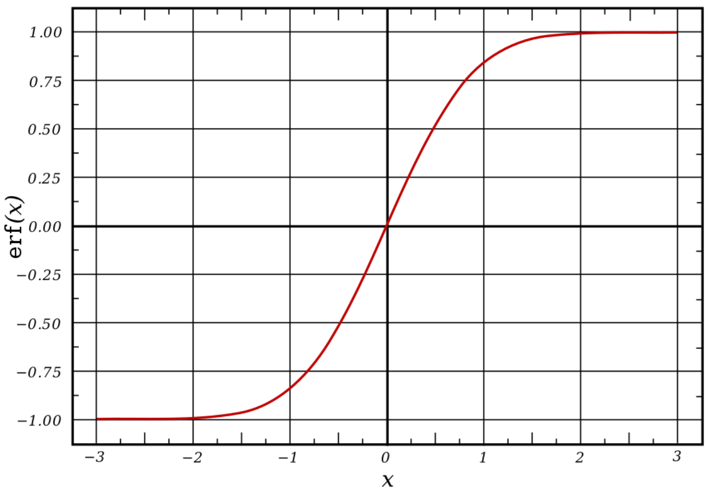


x0 = 1
x1 = 1
x2 = 1
...
xn = 1
w0 = b
Bias
w1
w2
.
wn
sum
f
Weights
Activation function
Input signals

Sigmoid activation function

$$f(u) = \frac{1}{1 + e^{-\alpha(u + \theta)}}$$

- Popular
- Slope parameter α
- Value between (0,1)
- Continuous and continuous derivative

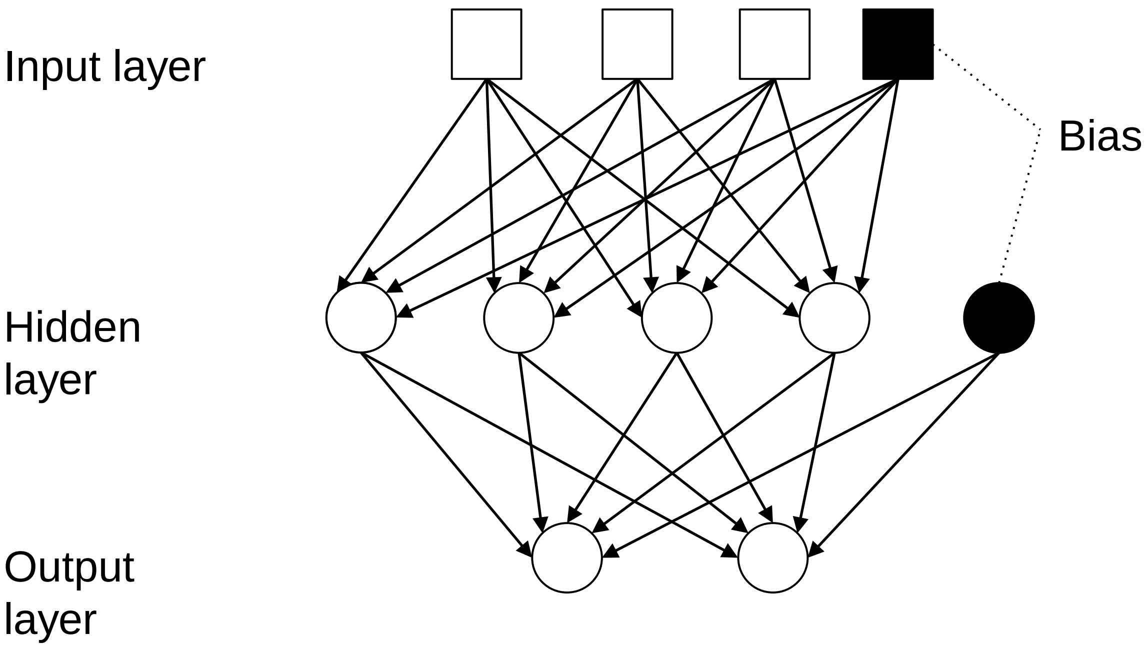


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ANN structure

- ANN network architecture is defined by:
  - Number input/output signal
  - Number of layers
  - Number of neurons in each layer
  - Connection between neurons
- Eachlayer consists of a group of neurons
  - Input layer receives input signals
  - Output layer return output signals
  - Hidden layers between input and output layer
- In Feedforward Neural Network connection comes from the previous layer to the following layer. There is no backward or intra-layer connection.

FNN example



A 3-layer FNN

- Input layer consists of 4 neurons
- Hidden layer consists of 5 neurons
- Output layer returns 2 output signal as it consists of 2 neurons

Number of parameters: 4 x 4 + 5 x 2 = 26

(practical neural network has ~10^6 parameters)

Loss function

- Consider an ANN with 1 output signal
- Input (x, y) as input signal and ground-true output
- Loss function

$E_{\boldsymbol{x}}(\boldsymbol{w}) = 1/2(y\text{-}y')^2$

where y’ is model’s output

- Loss function of the whole dataset D

$E_D(\boldsymbol{w}) = 1/|D| \sum_{\boldsymbol{x}\in D} E_{\boldsymbol{x}}(\boldsymbol{w})$

Gradient descent

- Gradient of loss function E (denoted as ∇E) is a vector proportional to slope of E
- ∇E determines the fastest direction to increase value of E

$$\nabla E(\boldsymbol{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n} \right)$$

where n is number of parameters

- Find fastest direction to decrease value of E

$$\Delta \boldsymbol{w} = -\eta . \nabla E(\boldsymbol{w})$$

where η is learning rate

- Activation function must be continuous and continuous derivative

Backpropagation

- Perceptron only perform linearly separable functions
- Multi-layer neural network can perform non-linearly separable functions
- Backpropagation
  - Propagate input signal from input layer through layers to output layers
  - Backpropagate error signal:
    - Evaluate error of output layer
    - Error signal backpropagate through layers back to input layer
    - Error values are evaluated depending on local error of each neuron

Gradient descent algorithm

Algorithm Gradient_descent_incremental((D, η))

Init $\mathbf{w}$ ($w_i \leftarrow$ small random value)
do
    for each example ($\mathbf{x}$, d) $\in D$ **do**
        Compute output
        for each parameter w_i **do**
            $w_i \leftarrow w_i - \eta(\partial E_{\mathbf{x}}/\partial w_i)$
        endfor
    endfor
until (terminal condition)
return $\mathbf{w}$

Initialize weights

- Random
- Large weight causes sigmoid function saturation. Objective function
- w^0_{ab} (connect from neuron b to neuron a)
  - $w^0_{ab} \in [-1/n_a, 1/n_a]$ where n_a is number of neurons in layer containing *a*
  - $w^0_{ab} \in [-3/k_a^{1/2}, 3/k_a^{1/2}]$ where k_a is number of neurons in layer containing b

Learning rate

- Large learning rate accelerates learning process, but can overlook global optimum
- Small learning slow down learning process
- Learning rate is often chosen based on experiment
- Change the learning rate during learning process

Number of neurons in hidden layers

- Number of neurons is chosen based on experiment
- Start with small number
- Increase if model can not converge
- Consider decreasing if model converged

Learning limitation of ANN

- 1-hidden-layer ANN can perform binary functions
- 1-hidden-layer ANN can perform stricted continuous functions
- 2-hidden-layer ANN can perform continuous functions

Pros and cons

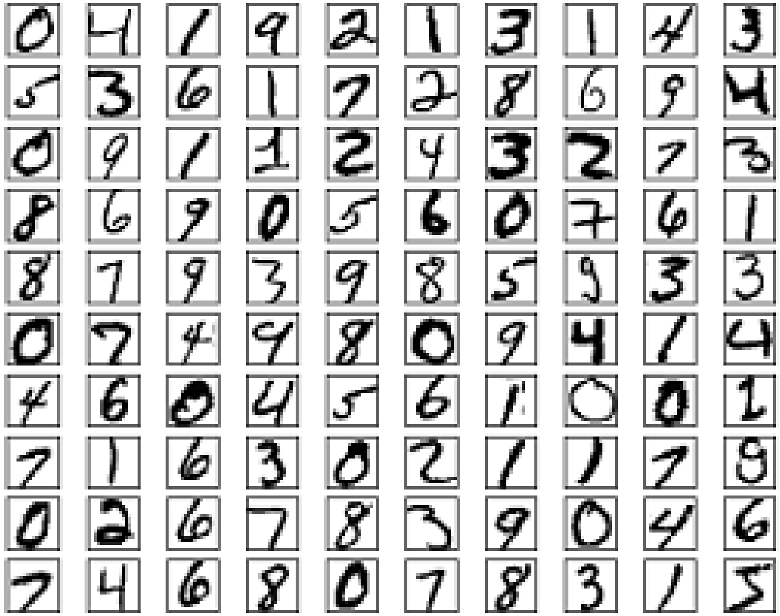
- Pros:
  - Support parallel computing
  - Fault tolerant
  - Self-adaptive
- Cons:
  - Network structure and hyper parameter chosen by experiment
  - A blackbox method, no clue to explain exactly what happened inside
  - Hard to explain to customer

Applications of ANN

- High dimensions input
- Output has type of number, vector
- There is noise in data
- Results explanation is not mandatory
- Long training time is acceptable
- Requires promptly predictions

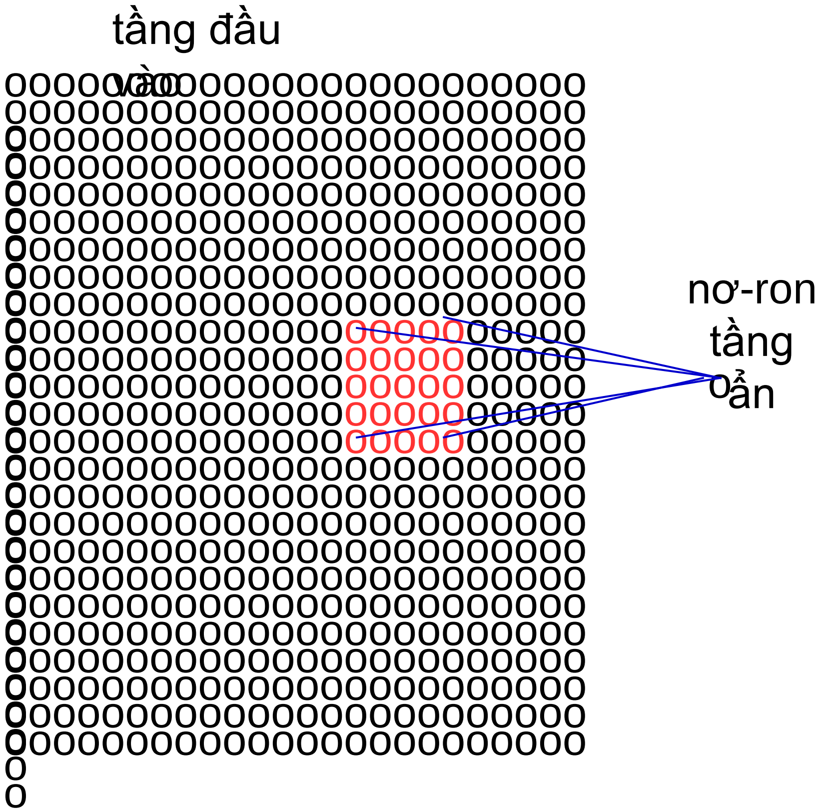
8. Convolutional Neural Network

- Handwritten number recognition [0..9]
  - Input: picture
  - Output: number [0..9]
- MNIST dataset:
  - Picture size 28 x 28
  - Train dataset size: 60K
  - Test dataset size: 10K
- FNN can not utilize spatial relation of pixels in picture

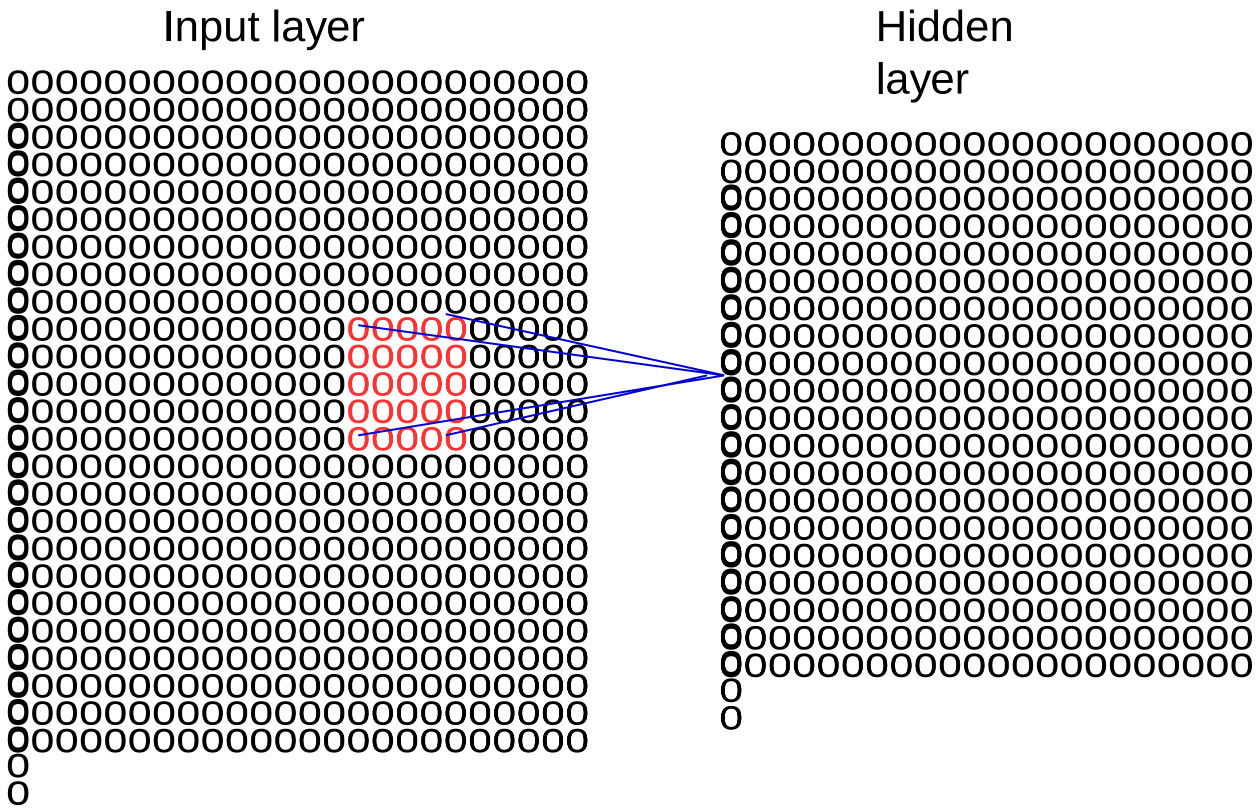


Filter

- Convolutional Neural Network (CNN) simulates visual activity
- Input as 28 x 28 matrix
- Each neuron in hidden layer receives input signal from a 5x5 area (25 pixel)
- Stride filter on input picture, each area connect to a neural on hidden layer
- Hidden layer has 24x24 neurons



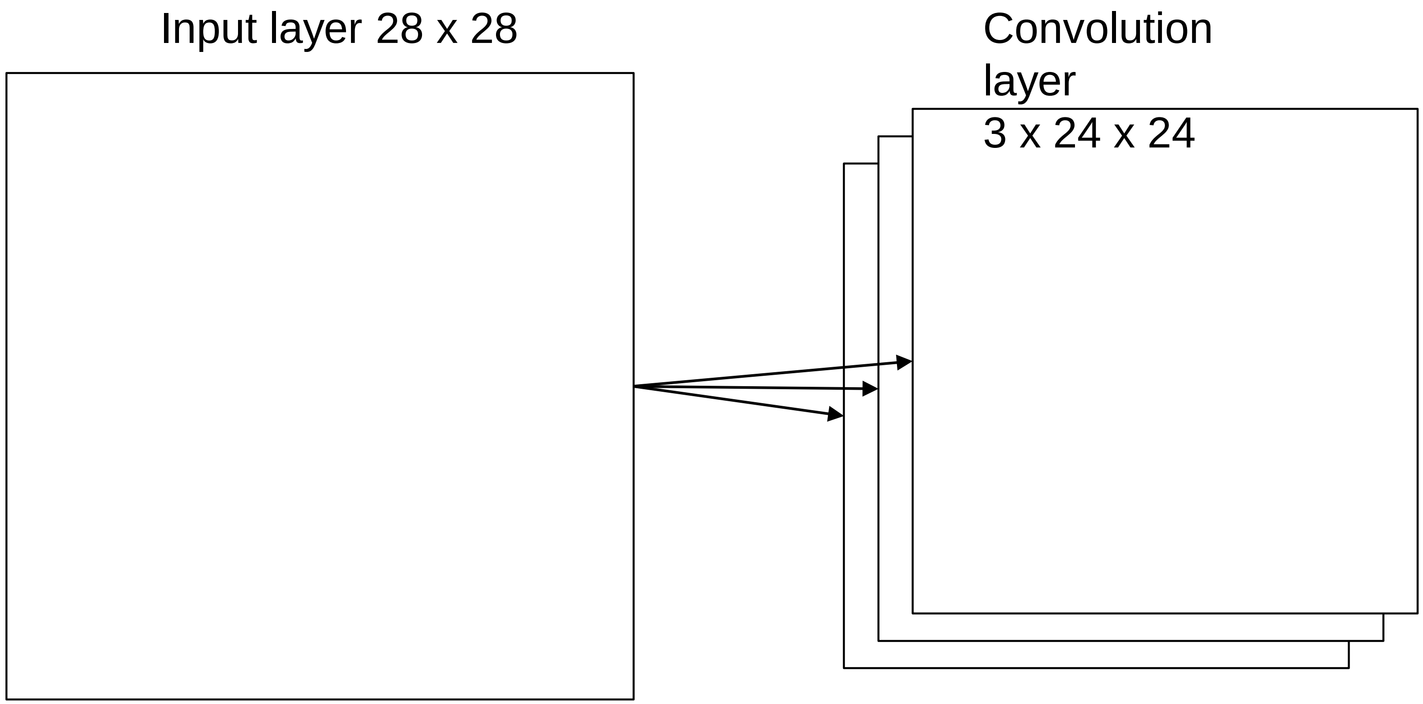
Example with 5x5 filter



Parameters sharing

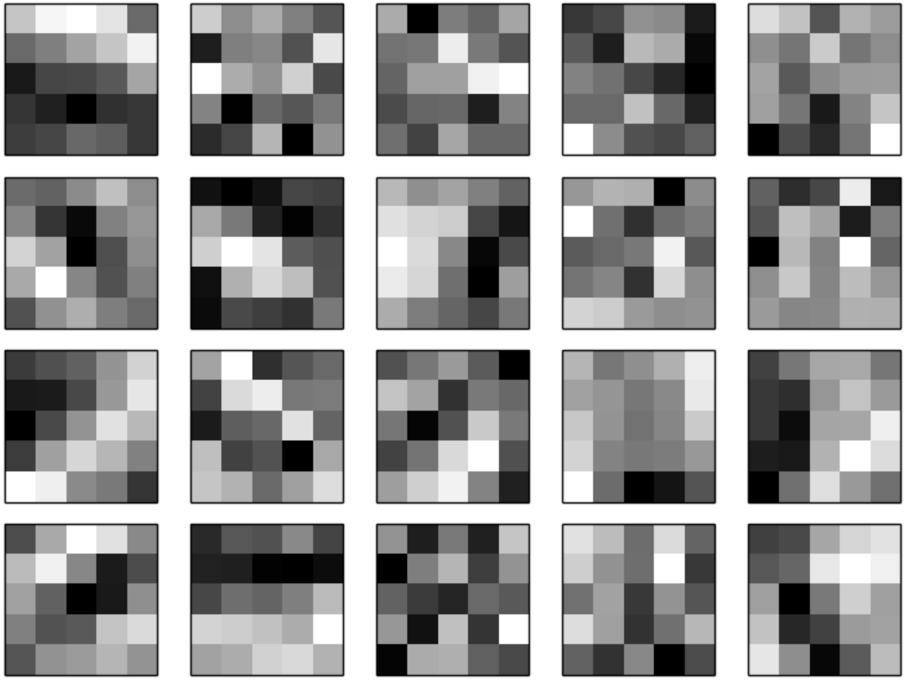
- Weights and biases are share among filters
- Hidden layer is supposed to detect a visual feature at different areas of input picture, based on translational invariance of pictures
- Filter is sometimes called kernel
- Output of hidden layer is called feature map
- Different filter results in different feature maps

Example of 3 filter



Example of filter

- The whiter little square, the smaller value
- The darker little square, the higher value
- Filter learn patterns in picture



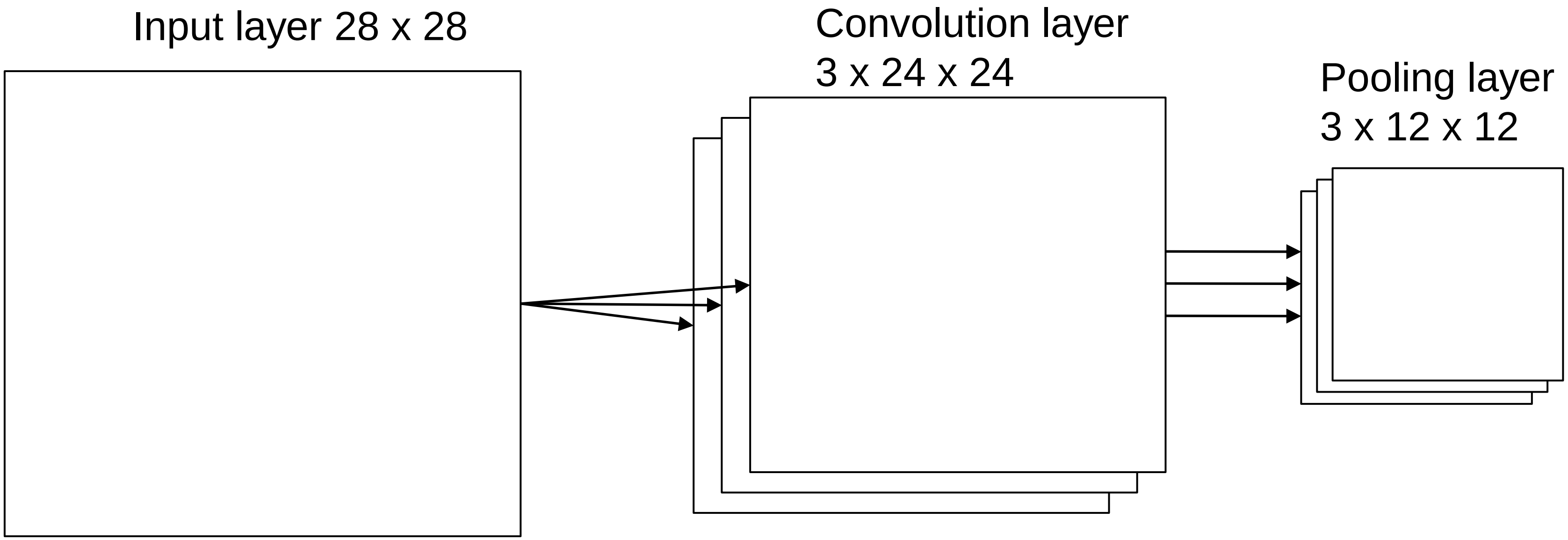
Number of parameters

- Fewer number of parameters than FNN
- A filter has 5x5 =25 weights and 1 bias
- 20 filters 20 x 26 = 520 parameters
- A FNN has hidden layer of 30 neurons has 30 x 28 x 28 (connection) + 30 (bias) = 23,550

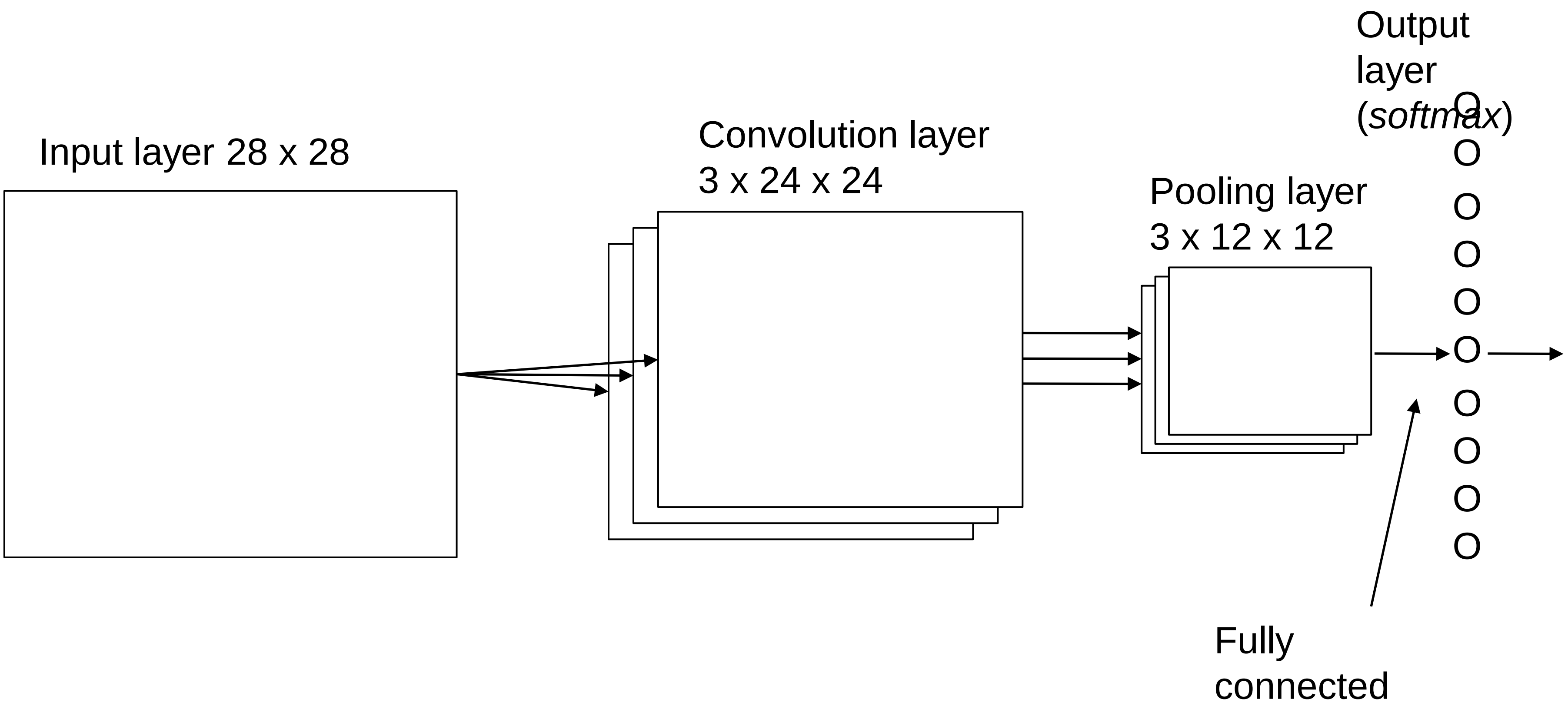
Pooling layer

- Pooling layer is installed after convolution layer
- Aggregate important features of convolution layer
- E.g.: feature map divided into 2x2 squares. Pooling layer return largest value in each square and aggregate into a smaller feature map (max pooling)
- Pooling is applied to every filter
- Pooling layer is supposed to remove positional information and keep useful pattern
- L2 pooling: square root of sum of square of values

Example of pooling layer



Example of full network



Softmax activation function

- Tackle problem of inefficient of mean square error
- z_j^L is input signal of *j-th neuron* in output layer *L*
- a_j^L is output signal of *j-th* neuron of output layer L after *softmax*
- *log-likelihood* loss function

$$a_j^L = \frac{e^{x_j^L}}{\sum_k e^{z_k^L}}$$

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1.$$

$$C \equiv -\ln a_j^L$$

Generalizability of CNN

- Learn high-level abstraction by convolution and pooling
- Activation function chosen based on experiment
- Activation function and loss function: sigmoid - cross entropy vs softmax - log likelihood
- Techniques to avoid overfitting

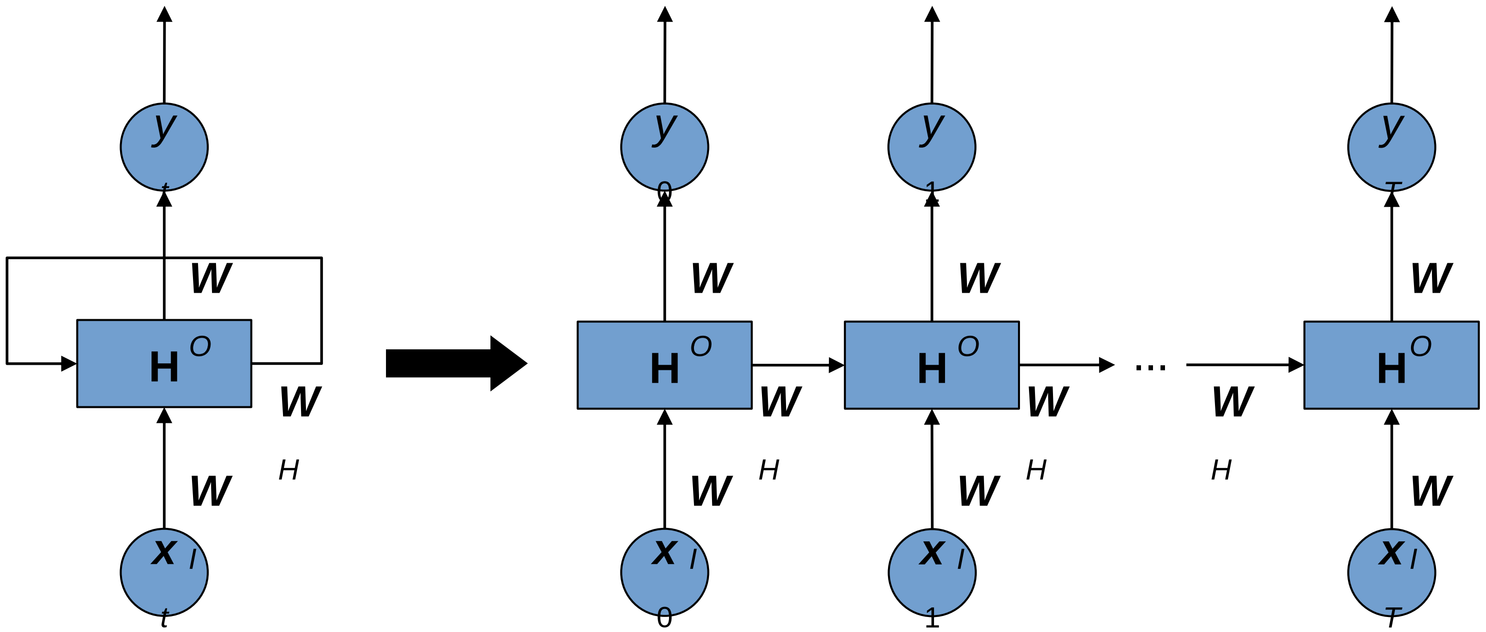
9. Recurrent Neural Network

- Sometimes, output depends on current input and other previous outputs
- Time-series problem (stock prices, weather forecast,…)
- Speech processing and natural languague processing (speech recognition, part-of-speech tagging, sentiment analysis)

Recurrent Neural Network

- Recurrent Neural Network is able to store information in recent history
- State of neuron at time (*t-1*) is used as input value at time *t*
- Parameters are shared by time

Example of RNN



Number of parameters

- I: number of neurons of input layer
- H: number of cell RNN
- K: number of neuron of output layer
- Number of parameters: I x H + H x H + H + H x K + K
  - Connection between input and hidden layer: I x H
  - Number of recurrent connection in hidden layer: H x H
  - Number of connection between hidden and output layer: H x K
  - Number of hidden layer biases: H
  - Number of output layer biases: K

Backpropagation: Forward phase

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

where:

- x^t_i is input signal of *i-th* neuron at time *t*

- w_i^h is connection weight between *i-th* neuron of input layer and *h-th* neuron of hidden layer

- $w_{h'h}$ is connection weight between *h'-th* neuron and *h-th* neuron of hidden layer

- $b_{h'}^{t-1}$ is output signal of *h'-th* neuron at time (*t-1)*

- a^t_h is input signal of *h-th* neuron of hidden layer at time *t*

$$b_h^t = \theta_h(a_h^t)$$

where:

- b^t_h is output signal of *h-th* neuron of hidden layer at time *t*
- θ_h is activation function of *h-th* neuron of hidden layer
- α^t_h is input signal of *h-th* neuron of hidden layer at time *t*

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

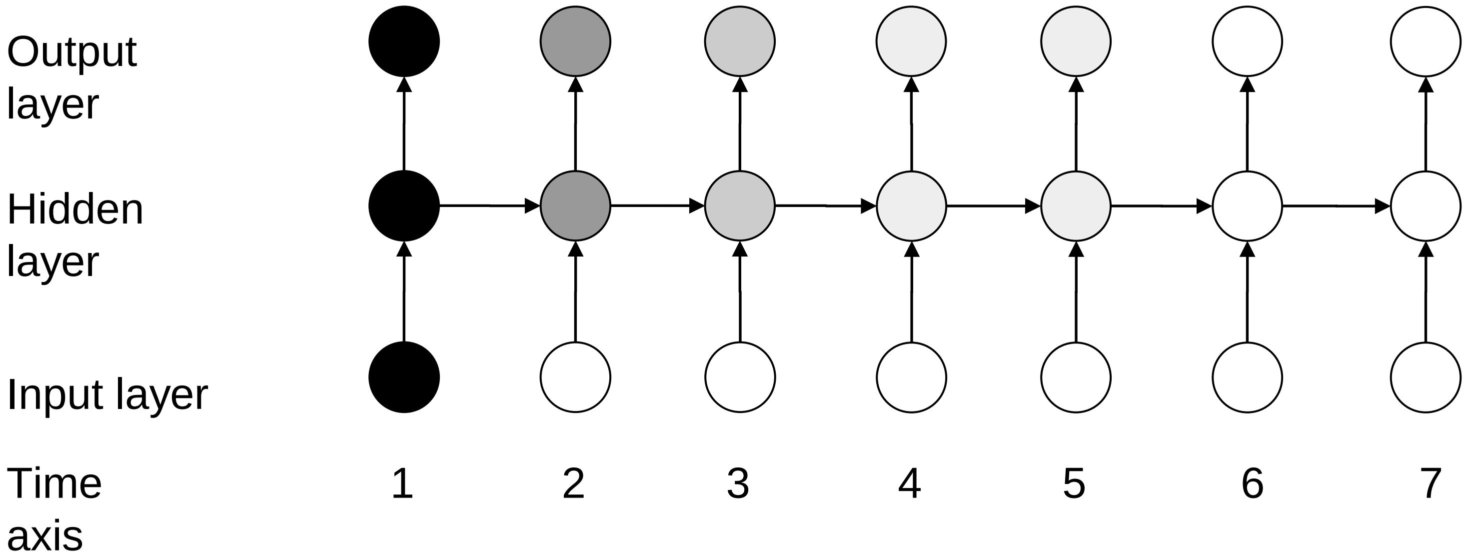
where:

- α^t_k is input signal of *k-th* neron of output layer at time *t*
- w_{hk} is connection weight between *h-th* neuron of hidden layer and *k-th* neuron of output layer
- b^t_h is output layer of *h-th* neuron of hidden layer at time *t*

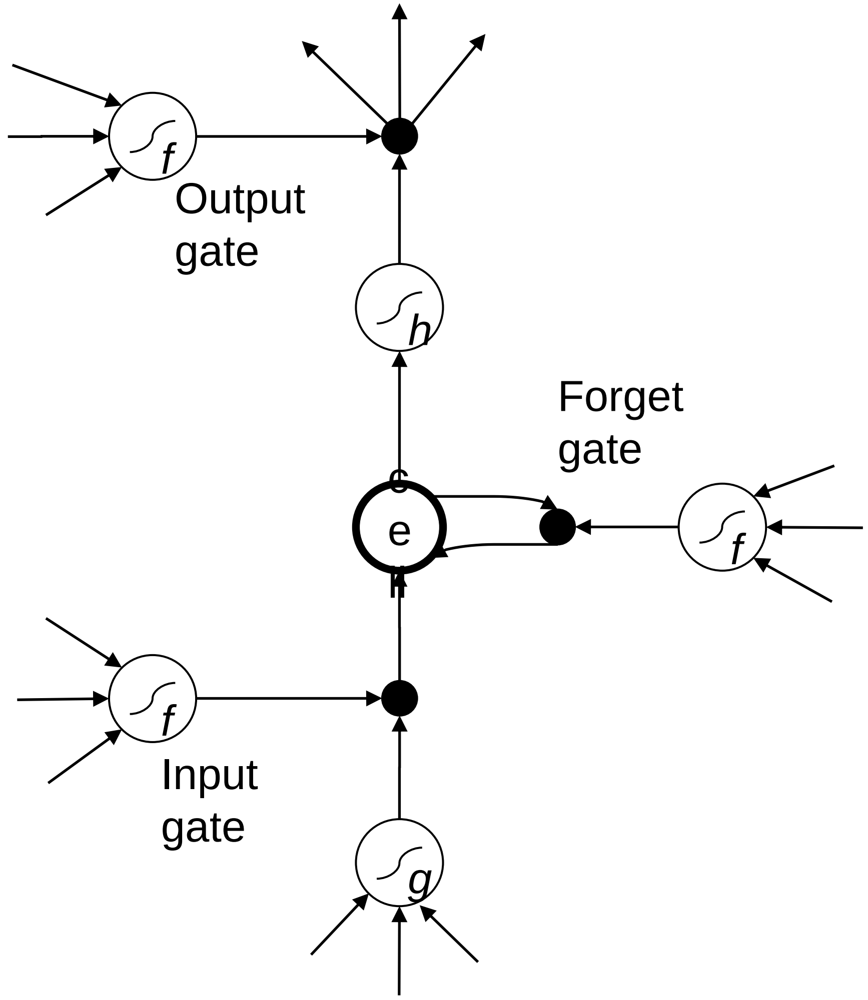
Gradient vanishing

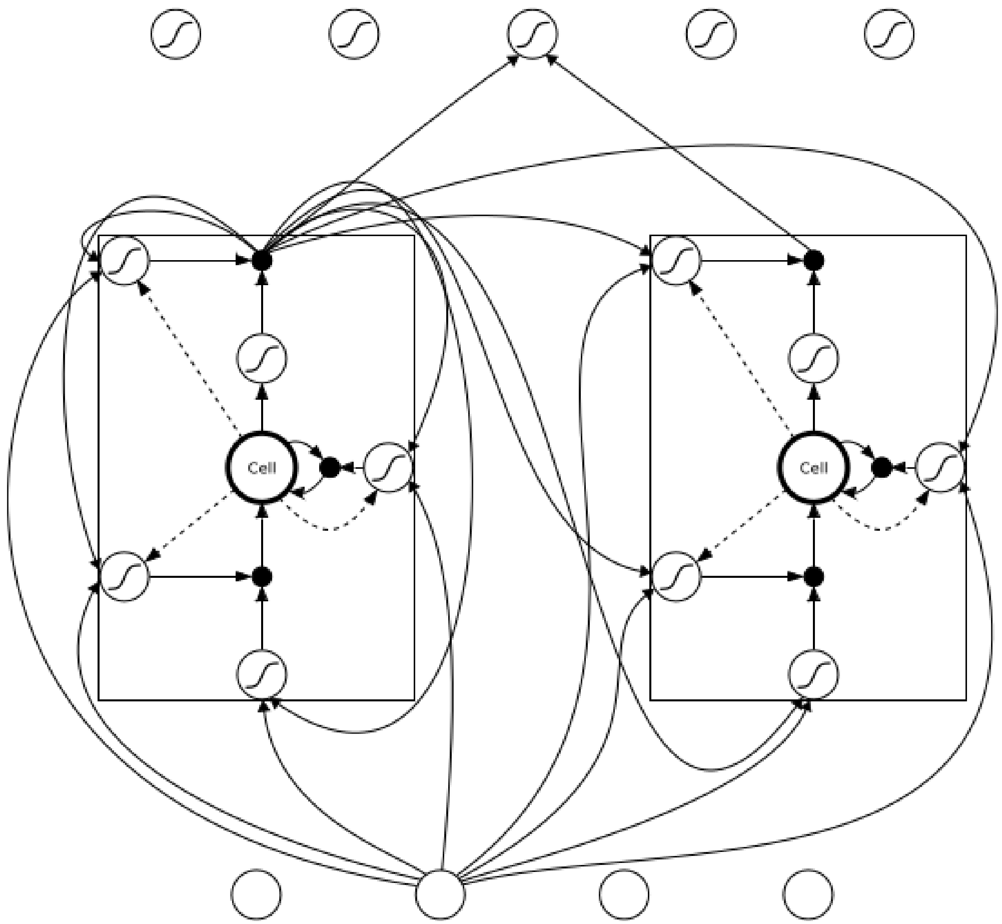
- In multi-layer neural network, first layers has slower "learning speed" than last layers
- Gradient vanishes when backpropagating from last layers to first layers
- Similar problem happens to RNN in time dimension
- Use LSTM cell to deal with the problem

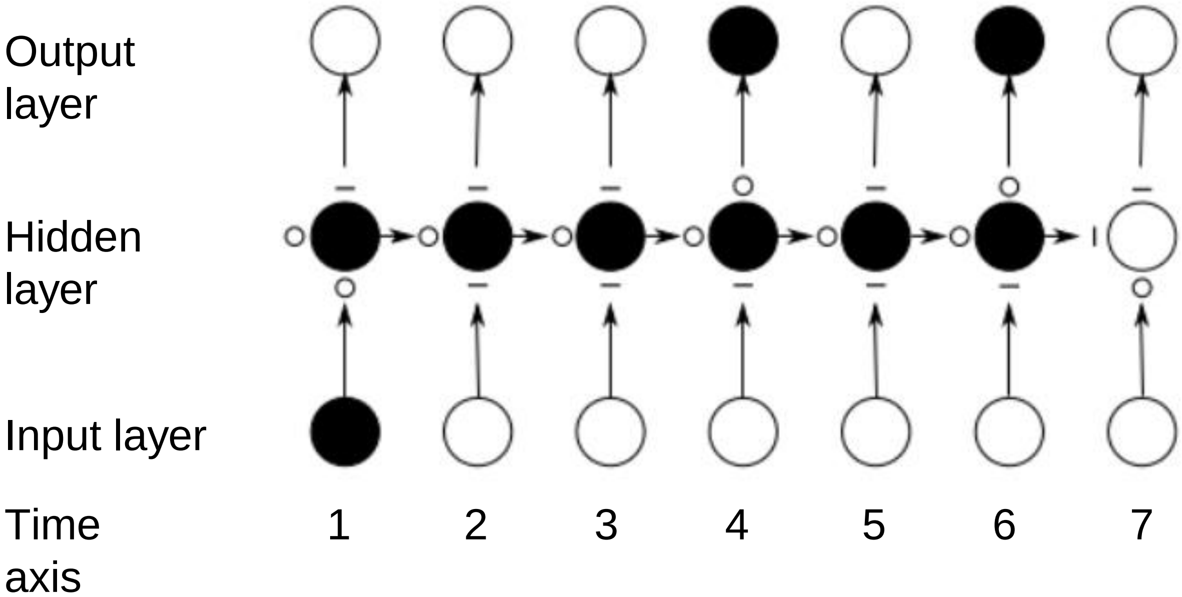
Gradient vanishing in RNN



Long-short Term Memory (LSTM)



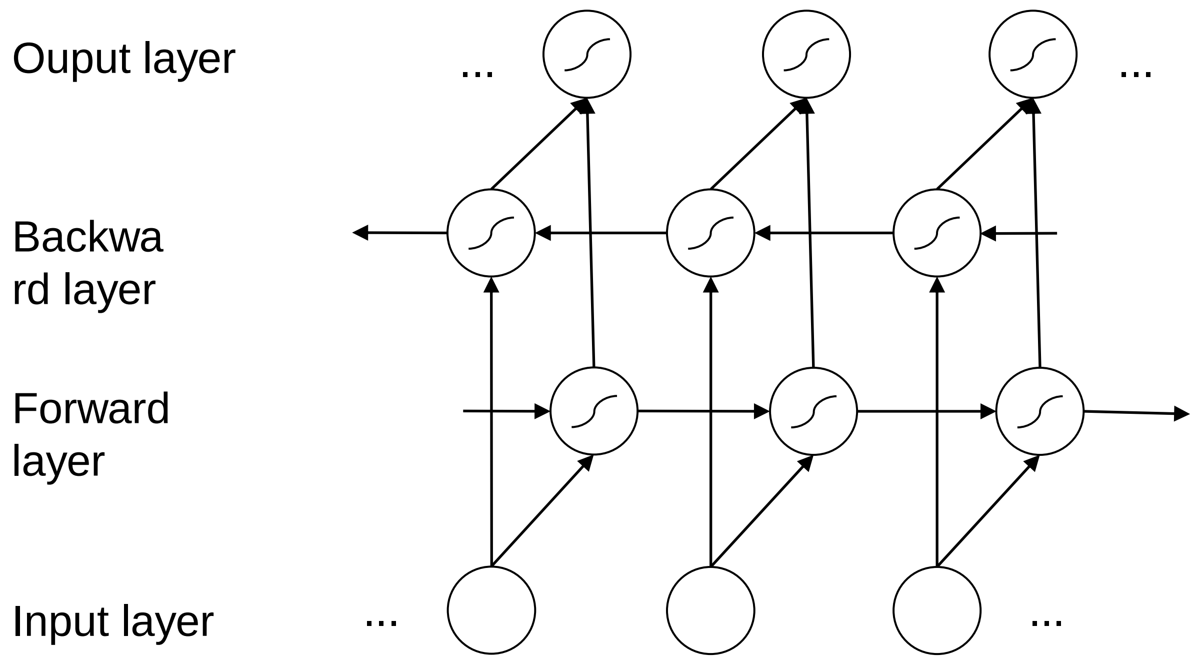

Cell
Cell


Output layer
Hidden layer
Input layer
Time axis
1
2
3
4
5
6
7

Number of parameters of LSTM

- I: Number of neurons in input layer
- H: Number of LSTM cells (hidden layer)
- K: Number of neurons in output layer
- Number of parameters: 4 x (I x H + H x H + H) + H x K + K
  - Number of connections between input and hidden layer: I x H
  - Number of recurrent connections in hidden layer: H x H
  - Number of connections between hidden and output layer: H x K
  - Number of biases of hidden layer: H
  - Number of biases of output layer: K

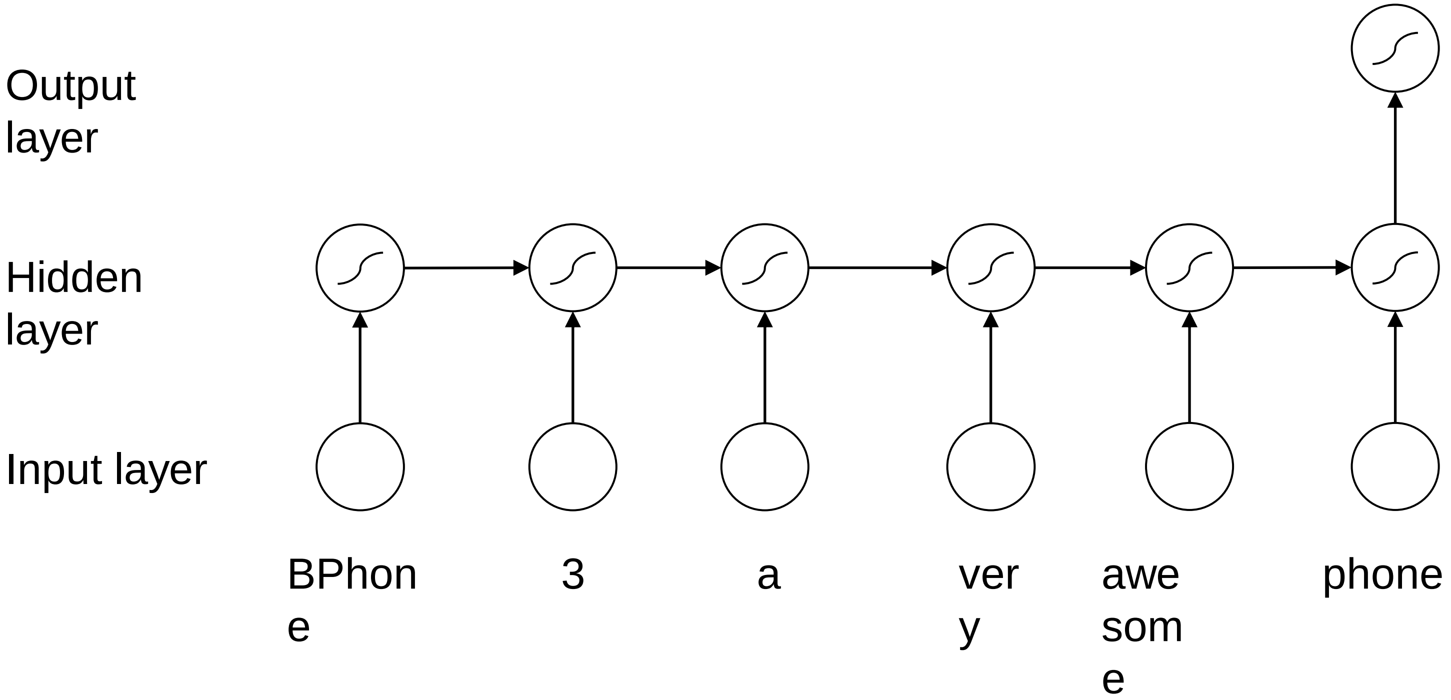
Bidirectional RNN



RNN for sentiment analysis

- Sentiment analysis:
  - Input: a document; $\boldsymbol{x}_t$ is *t-th* word in input document
  - Output: sentiment of document (positive, neutral, negative)
- Use output of the last word $\boldsymbol{x}_T$ for classifying: $\boldsymbol{x}_T$ is considered representation of the whole document(!)

- Input layer has V neurons (V is dictionary size)
- *One-hot* encoding: Each word $\boldsymbol{x}_t$ in document corresponds to a word t_i in dictionary, and *i-th* neuron receives input signal equal to 1, other neurons receive input signal equal to 0
- Output layer has 3 neurons as 3 sentiments output


Output layer
Hidden layer
Input layer
BPhon
e
3
a
ver
y
awe
som
e
phone

10. Ensemble learning
10.1 Bagging

- Boostrap Aggregating)
- Give a training dataset D has n observations and a learning algorithm
- Training:

1. Randomly sample with replacement from D, repeat k times. We have k set of examples S_1, S_2, …, S_k (n examples each). S_i contains average 63.2% observation in D. (lim n ∞ (1 – (1-1/n)^n) ~ 0.632)

2. Develop a classifier for each set S_i using the learning algorithm

- Testing: Classify every example using k classifiers, final results determined by voting mechanism with symmetry coefficients
- Bagging increases performance of **unstable** algorithm (decision tree)
- But bagging may decrease performance of **stable** algorithm (Naive Bayes, kNN)

10.2 Boosting

- Training:
  - Create a sequence of classifiers (each classifier denpends on its previous)
  - Use the same learning algorithm
  - Miss-classified examples will be adjusted weight, so that latter classifier can pay more attention on them
- Testing: On each example, results of k classifiers (at each step) are summed by weighted voting
- Boosting is suitable for unstable learning algorithms

Algorithm AdaBoost(D, Y, BaseLeaner, k)

Init $D_1(w_i) \leftarrow 1/n$ for each i; // Initialize

for $t = 1$ to k **do**

$f_t \leftarrow$ BaseLearner(D_t); // Create new classifier f_t

$e_t \leftarrow \sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$; // Computing error of f_t

if $e_t > 1/2$ **then** // if error value is large than threshold

$k \leftarrow k - 1$; // skip turn

exit-loop; // and exit

else

$\alpha_t \leftarrow e_t / (1 - e_t)$;

$$\begin{cases} \alpha_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 0 & \text{nếu không} \end{cases}$$

$D_{t+1}(w_i) \leftarrow D_t(w_i) \times$ ghts

$$D_{t+1}(w_i) \leftarrow \frac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$$

$$f_{final}(\mathbf{x}) \leftarrow \underset{y \in Y}{\operatorname{argmax}} \sum_{t: f_t(\mathbf{x}) = y} \log \frac{1}{\beta_t}$$ // adjust weights

endif

endfor

// final classifier

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!