ACDS Lecture Series

Lecture - 08

CSIR

Introduction to Deep Learning

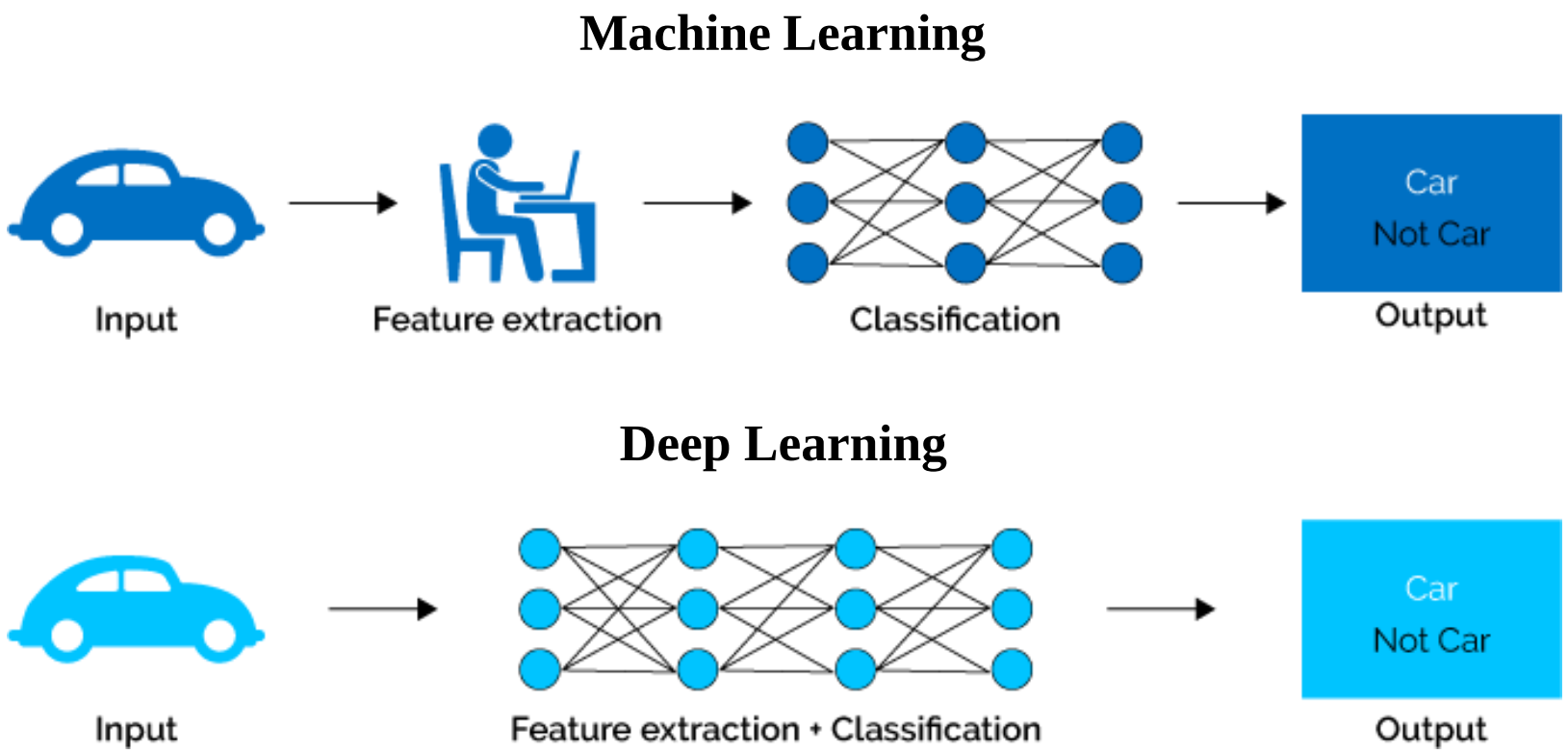
G. N. Sastry and Team

ADVANCED COMPUTATION AND DATA SCIENCES (ACDS) DIVISION

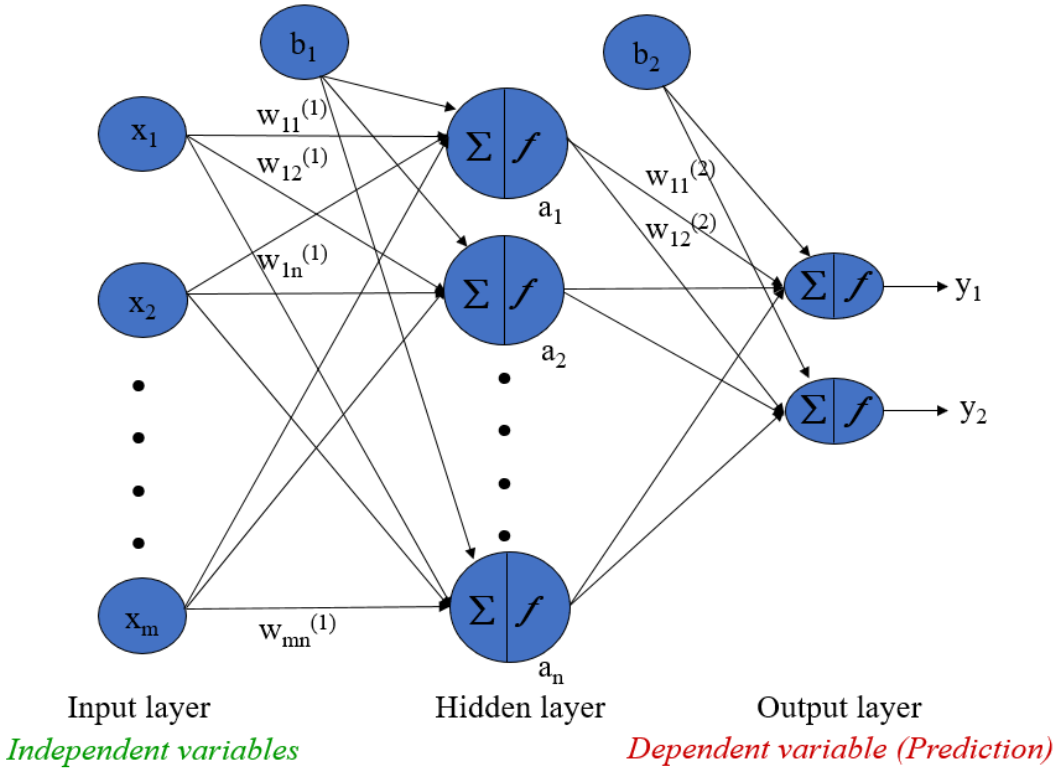
CSIR-North East Institute of Science and Technology, Jorhat, Assam, India

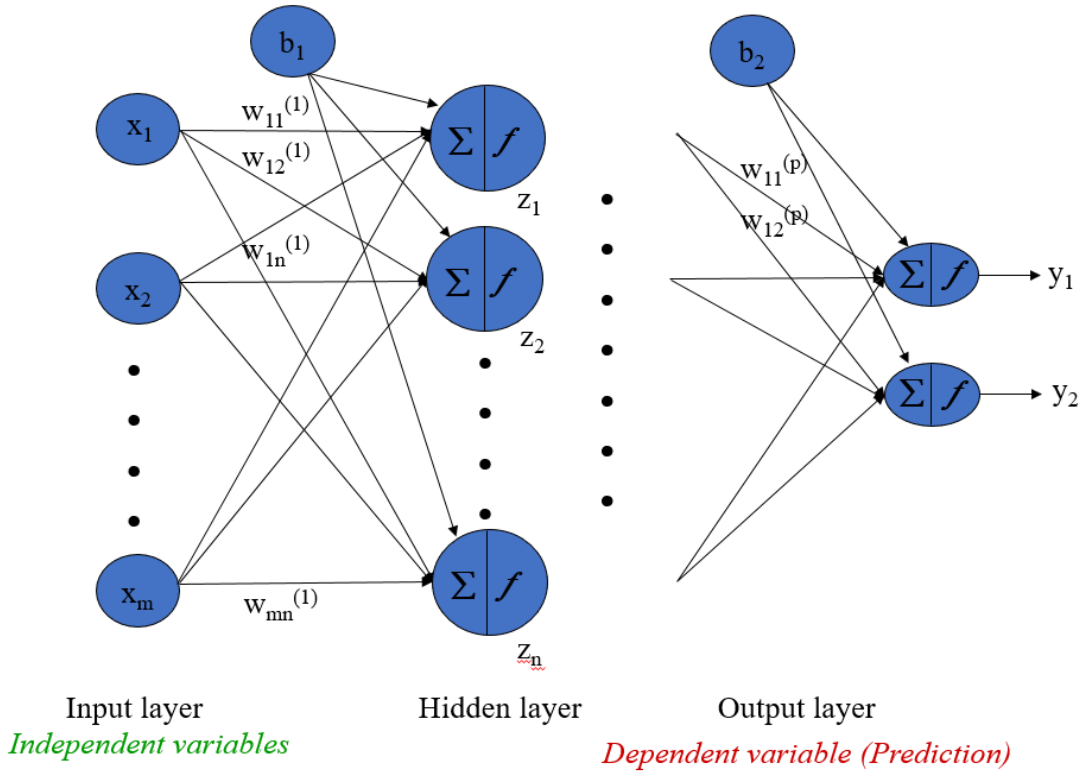
8. Introduction to Deep Learning

- DL is part of a broader family of machine learning methods based on ANN.
- DL uses multiple layers to progressively extract higher level features from the raw input.

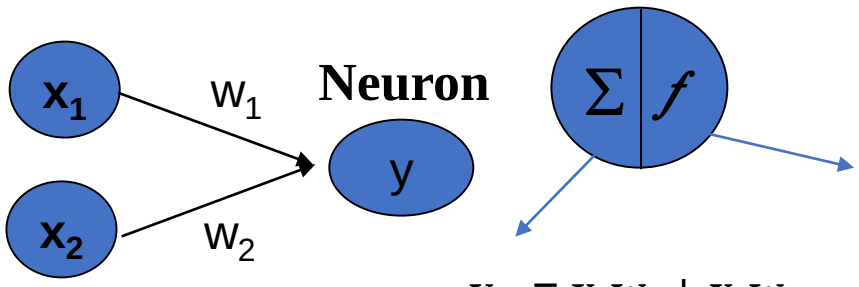


- Like ML, Deep Learning can be supervised, semi-supervised or Reinforcement.





Deep Neural Network → ANN with more than 3 hidden layer



$y_{in} = x_1w_1 + x_2w_2$

Activation Function:

if $y_{in} >= \theta$ $\quad f(y_{in}) = 1$

else $\quad f(y_{in}) = 0$

8.2 Fundamentals of neural network and deep learning

- 8.2.1 Common architectural principles of deep networks
  - 8.2.1.1 Parameters
  - 8.2.1.2 Layers
  - 8.2.1.3 Activation functions
  - 8.2.1.4 Loss functions
  - 8.2.1.5 Optimization algorithms
  - 8.2.1.6 Hyperparameters
- 8.2.2 Building blocks of deep networks
  - 8.2.2.1 RBMs
  - 8.2.2.2 Autoencoders

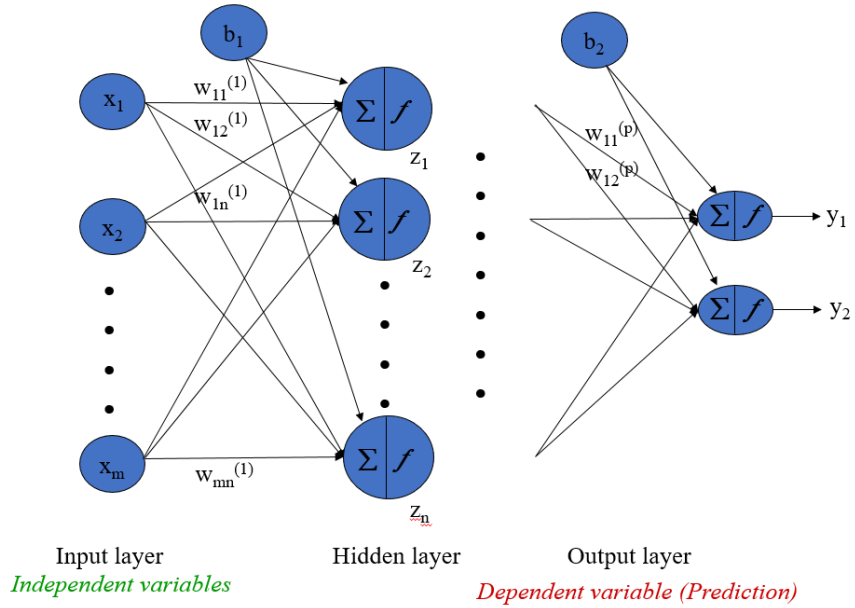
8.2.1.1 Parameters

- There are a number of different parameters that must be decided upon when designing a neural network.
- Among these parameters are the number of layers, the number of neurons per layer, the number of training iterations, etc.
- Some of the more important parameters in terms of training and network capacity are the number of hidden neurons, the learning rate and the momentum parameter.
  - #of input nodes: Depends upon the #of features.
  - #of input nodes: Depends upon the no of classes.
  - Training type
  - Momentum
  - Learning Rate
  - Minimum Error

8.2.1.2 Layers

- The input layer
  - Introduces input values the world outside into the network.
  - No activation function or other processing.

- The hidden layer(s)
  - Activation function or other processing.
  - Two hidden layers are sufficient to solve any problem.



- The output layer
  - Functionally just like the hidden layers.
  - Outputs are passed on to the world outside the neural network.

8.2.1.3 Activation Function

Activation Function	Mathematical Form	Applications
Identity	$f(x) = x$	-
Binary step function	$f(x) = 1$, if $x >= \Theta$ $f(x) = 0$, if $x < \Theta$	-
Bipolar step	$f(x)=1$, if $x >= \Theta$ $f(x) = -1$, if $x < \Theta$	-
Ramp functions	$f(x)=1$, if $x > 1$ $f(x) = x$, if $0<= x <=1$ $f(x) = 0$, if $x < 0$	-
Sigmoid	$f(x) = 1 / 1 + e^{-x}$	Regression
Tanh (Hyperbolic Tangent)	$f(x) = e^{x} - e^{-x} / e^{x} + e^{-x}$ $= 1 - e^{-2x} / 1 + e^{-2x}$ $= 2/1+e^{-2x} - 1$ $= 2\ \text{sigmoid}(2x) - 1$	Neural Network
ReLu (Rectified Linear Units)	$f(x) = \max(0, x)$	Neural Network

8.2.1.4 Loss Function

Loss function	Cross-entropy (i.e. negative log likelihood) $J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[\log\left(1 - y_k^{(i)}\right)\log\left(1 - \hat{y}_k^{(i)}\right)\right]$	Quadratic loss (i.e. Mean Squared Error) $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$ Mean Absolute Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\|y^{(i)} - \hat{y}^{(i)}\right\|$

8.2.1.5 Optimization Algorithms

- Methods used to change the attributes of the neural network such as **weights** and **learning rate** to reduce the losses.
- Optimizers are used to solve optimization problems by minimizing the function.
- Various optimizers are researched within the last few couples of years each having its advantages and disadvantages.

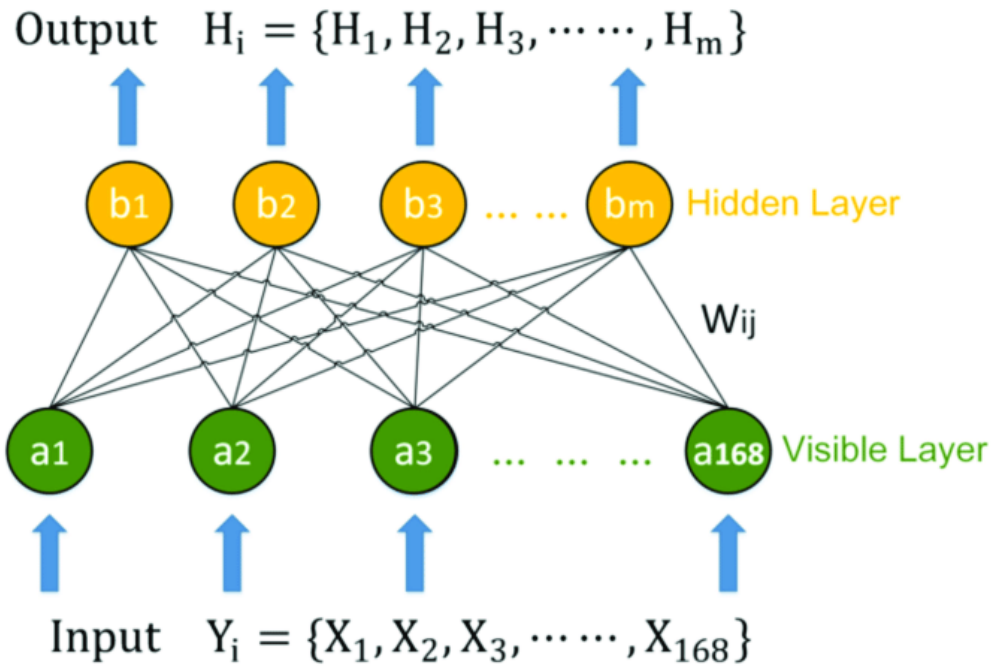
- Different types of optimizers and how they exactly work to minimize the loss function –
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-Batch Gradient Descent
  - Momentum
  - Nesterov Accelerated Gradient

8.2.1.6 Hyperparameters

- Hyperparameters are the variables which determines the network structure.
- The variables which determine how the network is trained.
- Hyperparameters are set before training(before optimizing the weights and bias).

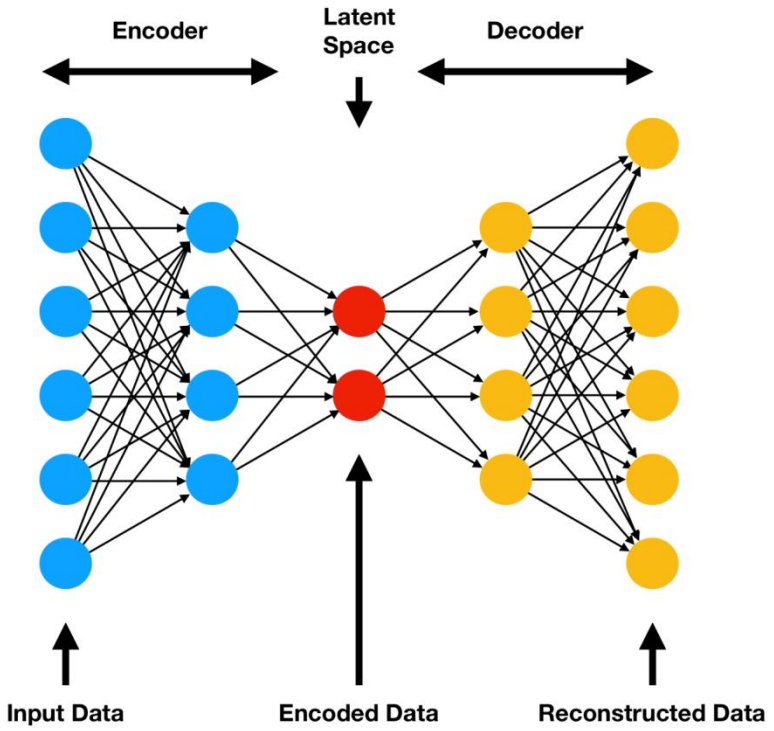
- Some of the hyperparameters are such as —
  - Network Weight Initialization
  - Activation function
  - Learning Rate
  - Momentum
  - Number of epochs
  - Batch size

- **RBM** is a generative stochastic ANN that can learn a probability distribution over its set of inputs.
- RBMs found applications in dimensionality reduction, classification, collaborative filtering, feature learning, topic modelling and even many body quantum mechanics.
- They can be trained in either supervised or unsupervised, depending on the task.
- DBN can be formed by "stacking" RBMs and optionally fine-tuning the resulting deep network with gradient descent and backpropagation.



8.2.2.2 Autoencoder

- An **autoencoder** is used to learn efficient data codings in an unsupervised manner.
- The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”.
- Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name.



8.3 Regularization for Deep Learning

8.3.1 Parameter norm penalties
8.3.2 Regularization and under-constrained problems
8.3.3 Dataset augmentation
8.3.4 Semi-supervised learning

8.3.1 Parameter Norm Penalties

Training data

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

Unregularized loss function

Regularizer on weights and bias vectors

Typical: bias vectors are not used for regularization

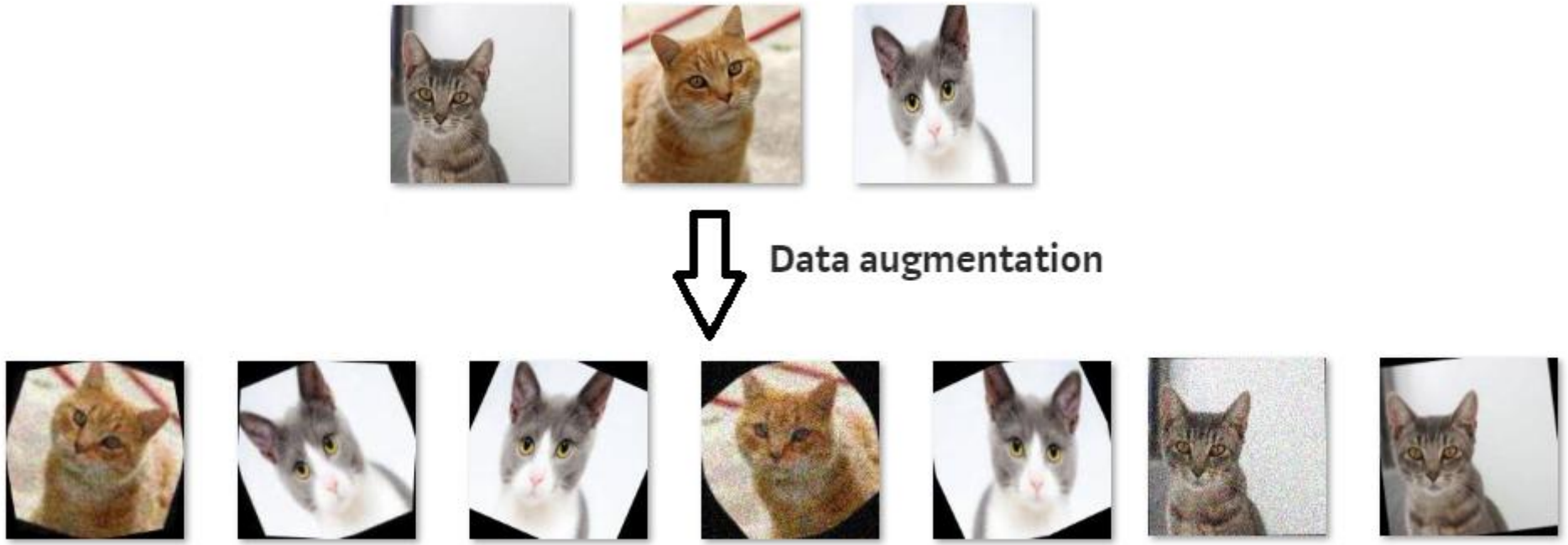
Convention:
$\boldsymbol{w}$ denotes only the weights used in regularization
$\boldsymbol{\theta}$ denotes all the parameters

8.3.2 Regularization and Under-constrained Problem

- In some cases, regularization is necessary for machine learning problems to be properly defined.
- Many linear models in machine learning, including linear regression and PCA, depend on inverting the matrix X^TX.
- This is not possible whenever X^TX is singular.
- This matrix can be singular whenever the data truly has no variance in some direction, or when there are fewer examples (rows of X)than input features (columns ofX).
- In this case, many forms of regularization correspond to inverting $X^TX+\alpha I$ instead.
- This regularized matrix is guaranteed to be invertible.

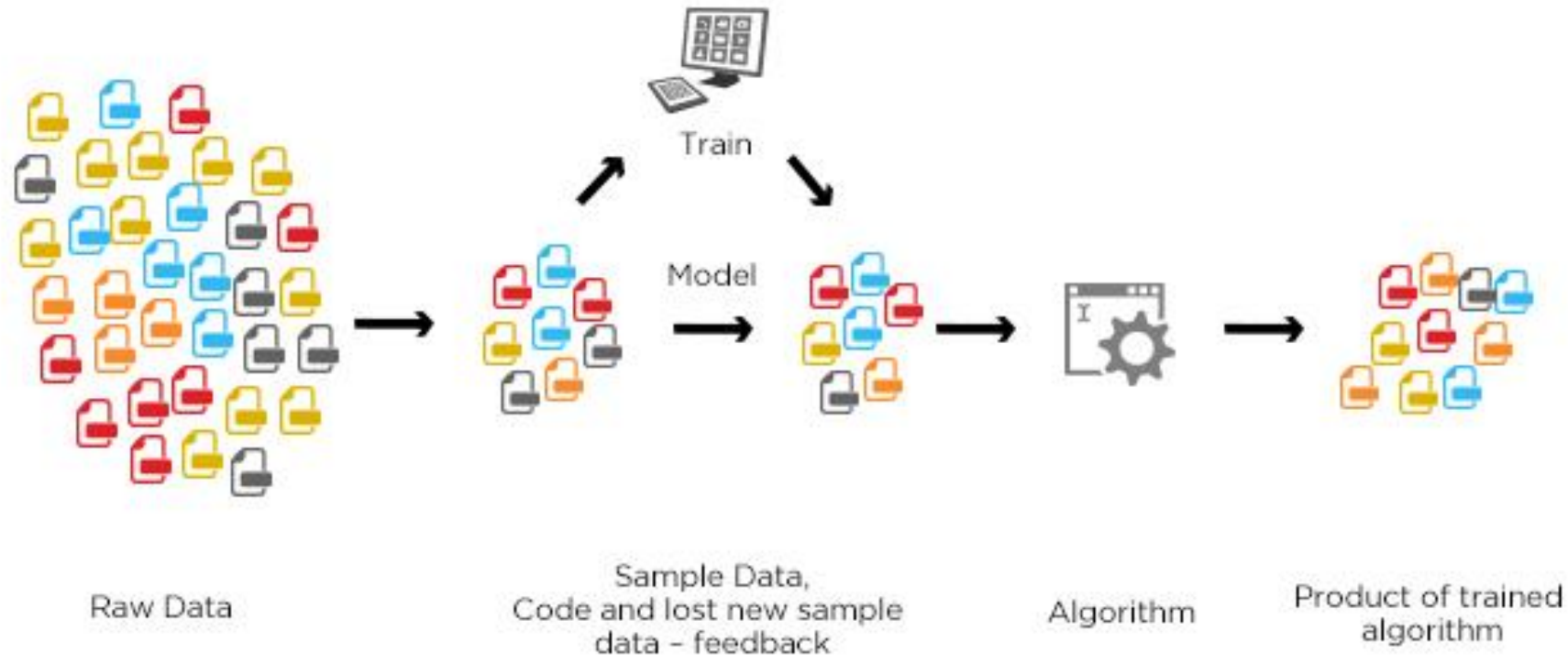
8.3.3 Data Augmentation

- Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data.
- It acts as a regularizer and helps reduce overfitting when training a machine learning model.
- It is closely related to oversampling in data analysis.



8.3.4 Semi Supervised Learning

- Semi-supervised learning is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data).



8.4 Major Architectures of Deep Networks

8.4.1 Unsupervised pretrained networks
- 8.4.1.1 Deep belief networks
- 8.4.1.2 Generative adversarial networks

8.4.2 Convolutional neural networks

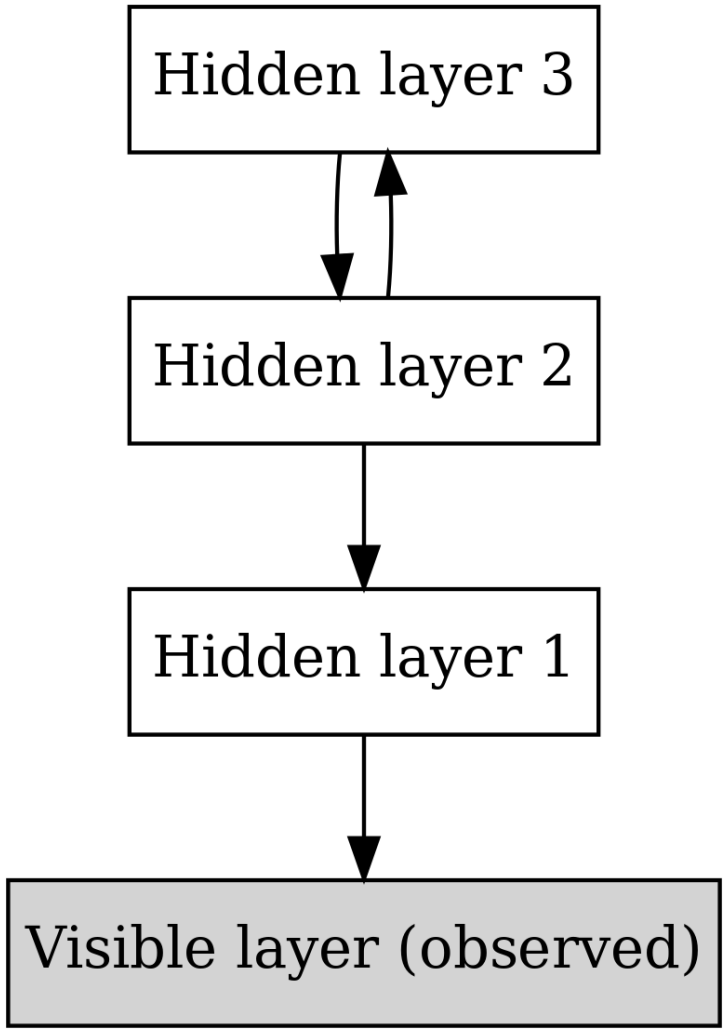
8.4.3 Recurrent neural networks

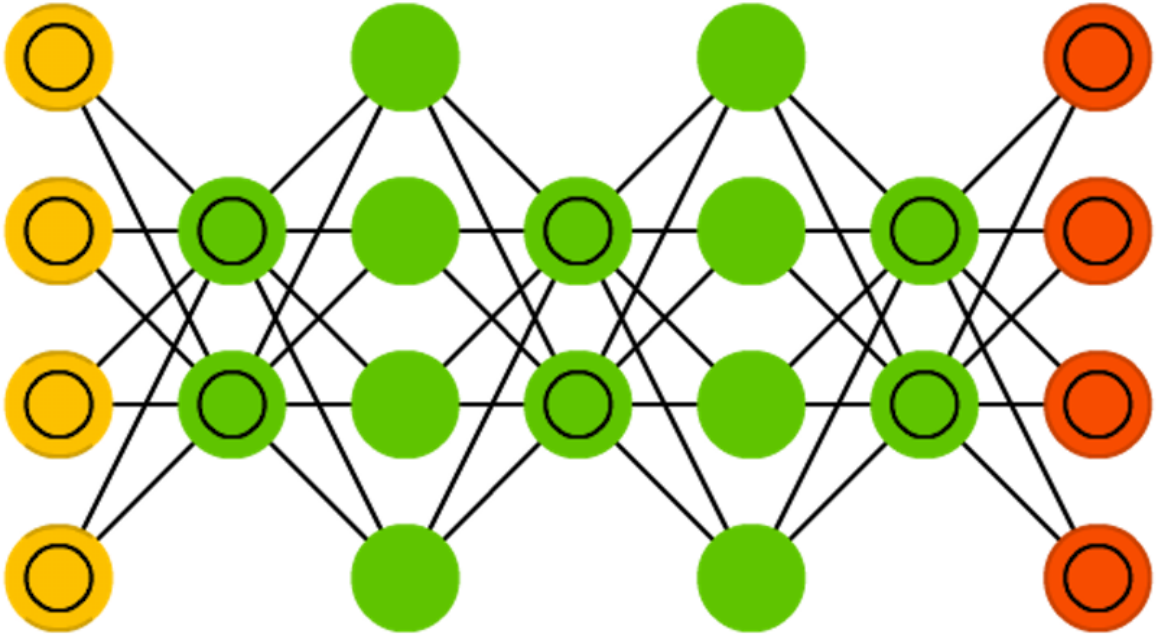
8.4.4 Recursive neural networks

8.4.1.1 Deep Belief Network

- **DBN** is a generative graphical model composed of multiple layers of latent variables ("hidden units"), with connections between the layers but not between units within each layer.
- When trained on a set of examples without supervision, a DBN can learn to probabilistically reconstruct its inputs.
- The layers then act as feature detectors.
- After this learning step, a DBN can be further trained with supervision to perform classification.
- DBNs can be viewed as a composition of simple, unsupervised networks such as restricted Boltzmann machines (RBMs) or autoencoders, where each sub-network's hidden layer serves as the visible layer for the next. An RBM is an undirected, generative energy-based model with a "visible" input layer and a hidden layer and connections between but not within layers.
- This composition leads to a fast, layer-by-layer unsupervised training procedure, where contrastive divergence is applied to each sub-network in turn, starting from the "lowest" pair of layers (the lowest visible layer is a training set).
- The observation that DBNs can be trained greedily, one layer at a time, led to one of the first effective deep learning algorithms. Overall, there are many attractive implementations and uses of DBNs in real-life applications and scenarios.

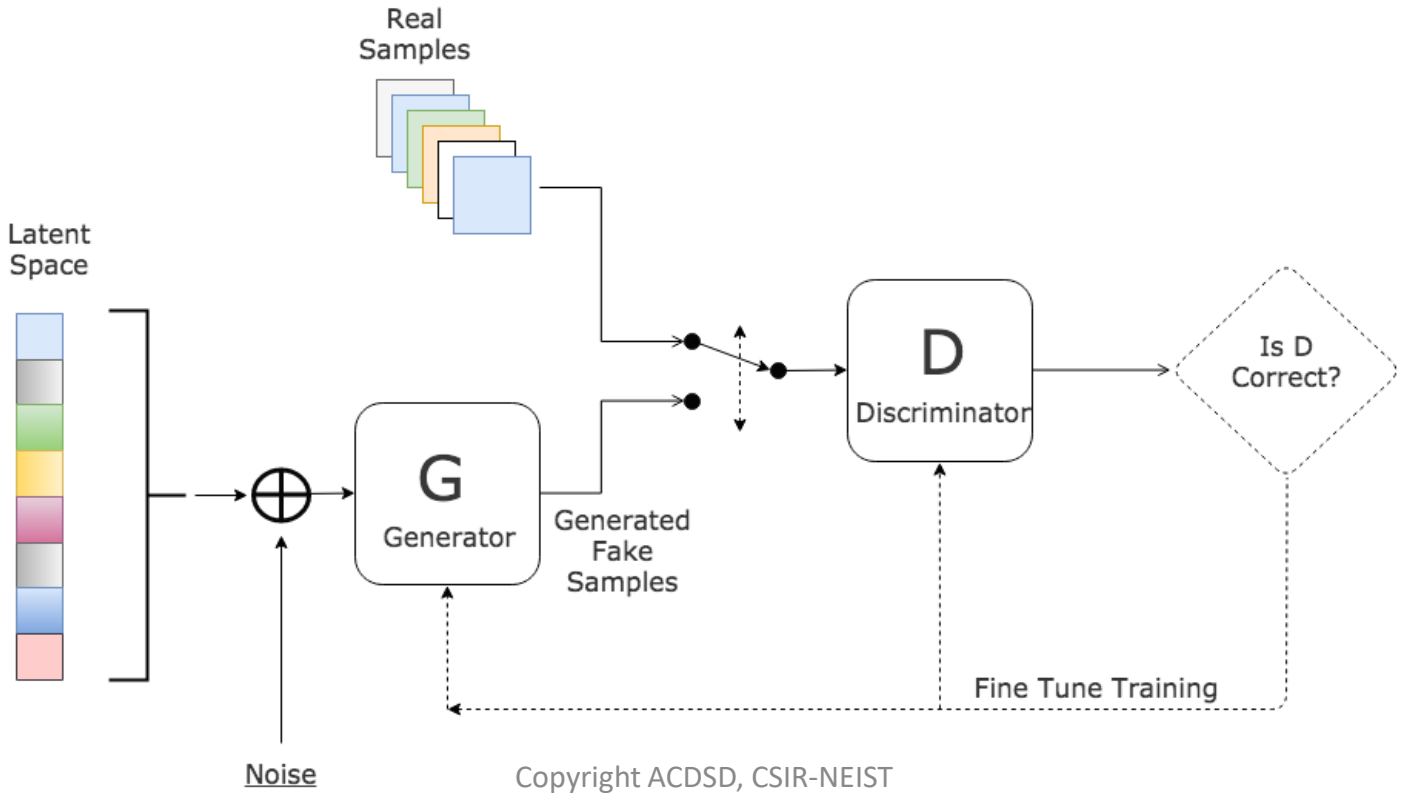
8.4.1.1 Deep Belief Network





8.4.1.2 Generative Adversial Network

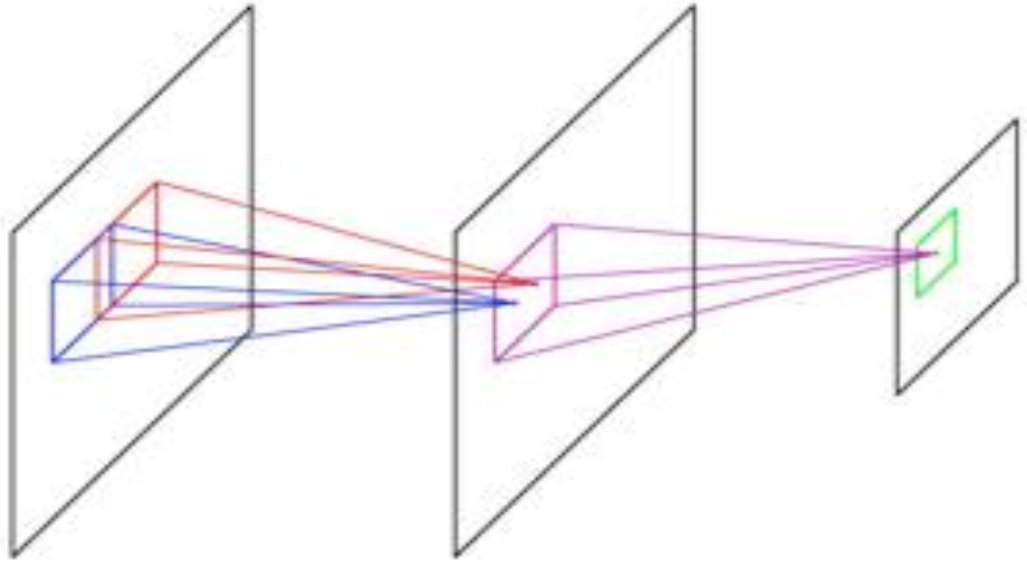
- **GAN** is a class of DL where two neural networks contest with each other in a game (in the form of a zero-sum game, where one agent's gain is another agent's loss).
- The core idea of a GAN is based on the "indirect" training through the discriminator, which itself is also being updated dynamically.[5] This basically means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.



8.4.2 Convolutional Neural Network

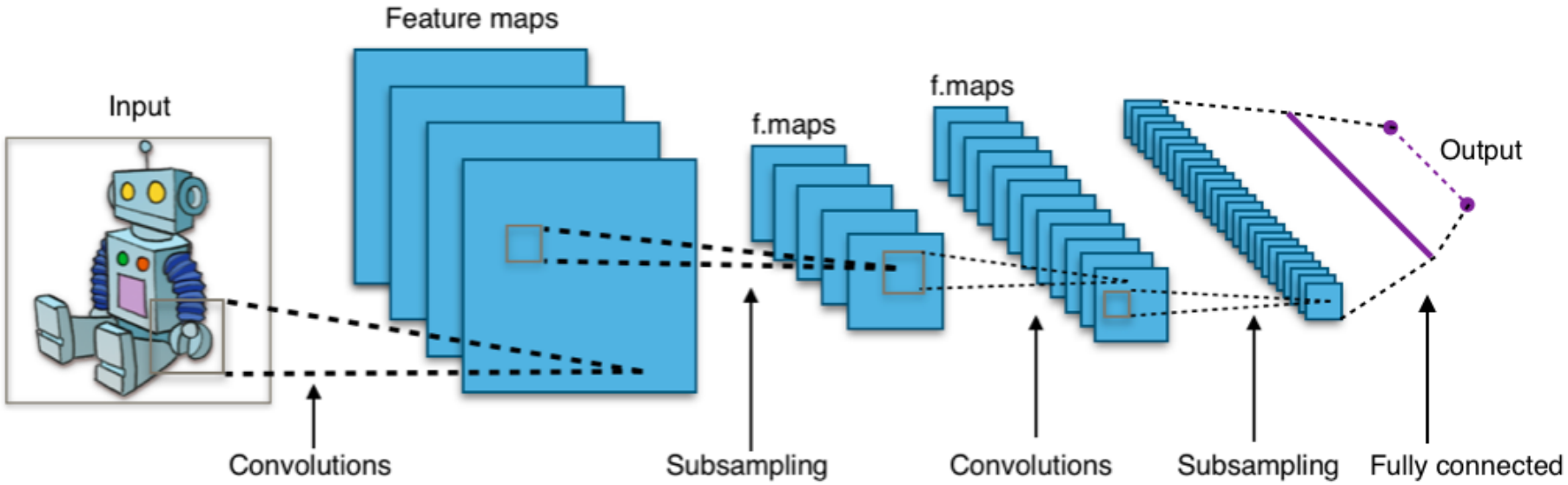
Convolutional Neural Network

- The network is not fully connected.
- Different nodes are responsible for different regions of the image.
- This allows for robustness to transformations.



8.4.2 Convolutional Neural Network

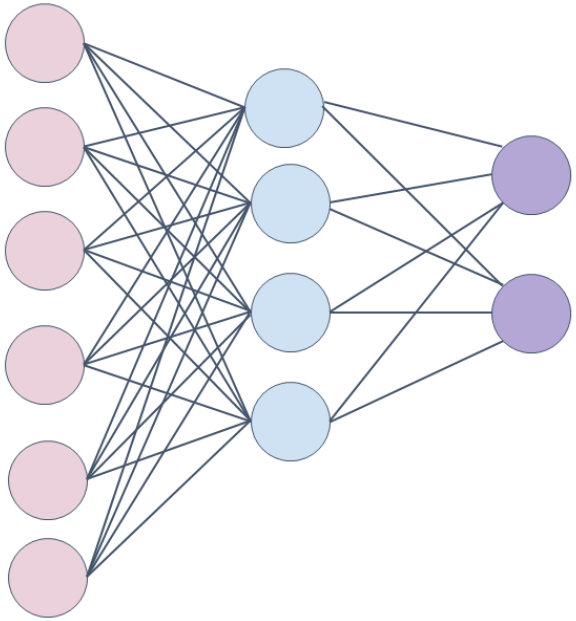
- Convolution or Feature Extraction Layer (+ ReLU)
- Pooling or Subsampling Layer (max or average Pooling)
- Fully Connected + ReLU
- Softmax



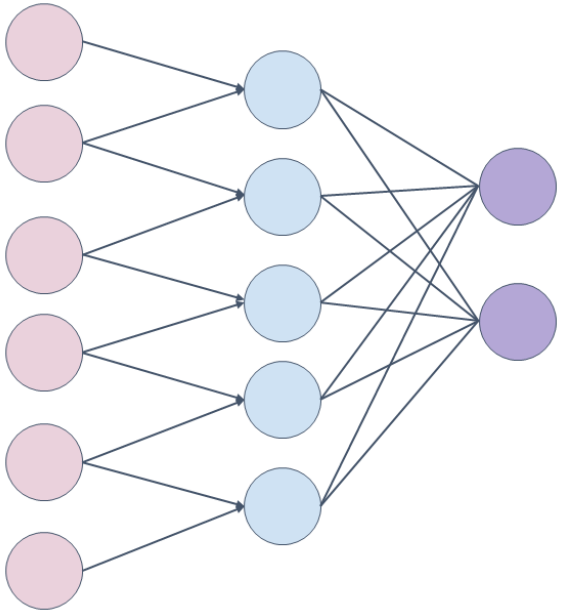
Note: CNN hidden layers are basically alternation of convolution and pooling layers.

Note: Though the layers are colloquially referred to as convolutions, this is only by convention. Mathematically, it is technically a *sliding dot product* or cross-correlation.

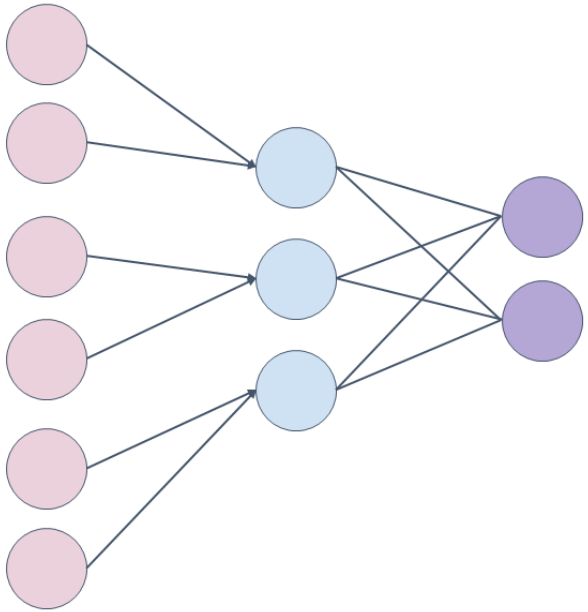
8.4.2 Convolutional Neural Network



DNN



CNN
Filter Size: 2
Filters: 1
Stride: 1



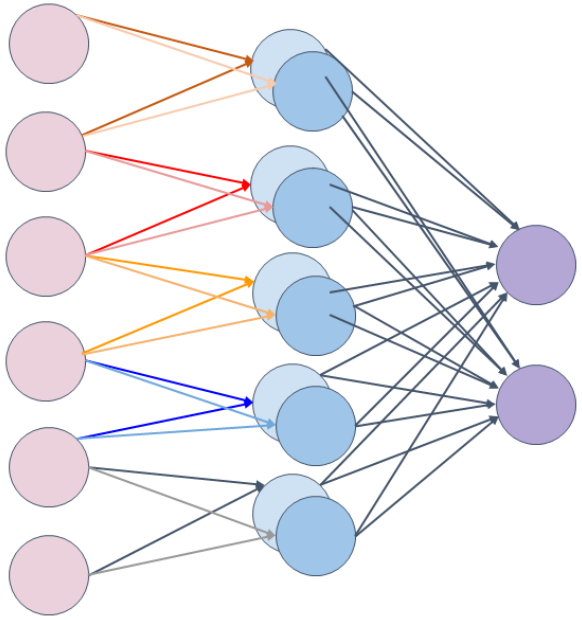
CNN
Filter Size: 2
Filters: 1
Stride: 2

Note: The weights can be treated as filter for edge detection! Where, $y = w_1x_1 + w_2x_2$

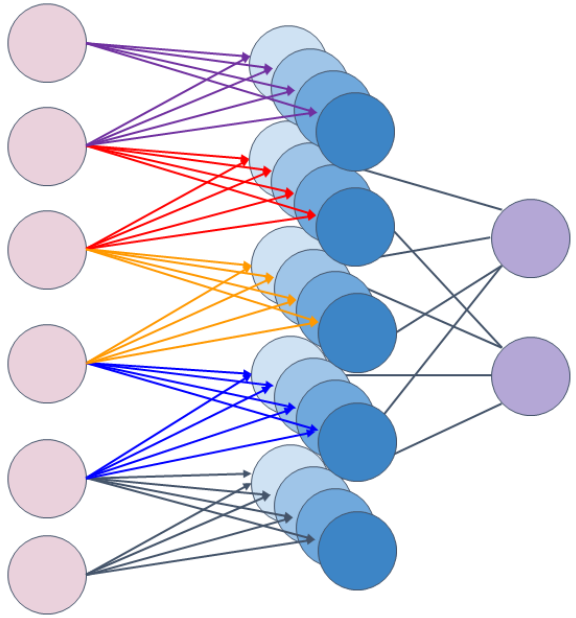
Note: We can do pooling for size reduction.

Note: add zero padding to include more edge pixels.

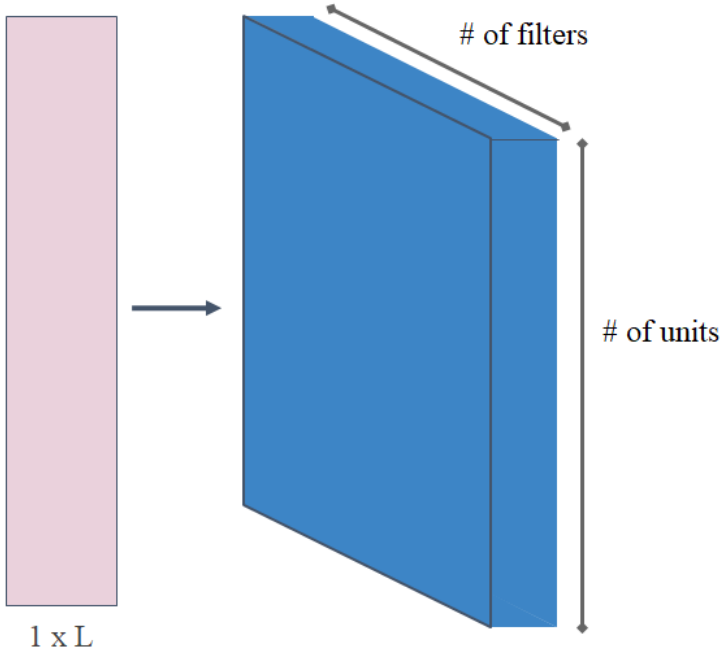
8.4.2 Convolutional Neural Network



CNN
Filters: 2
Filter Size: 2
Stride: 1



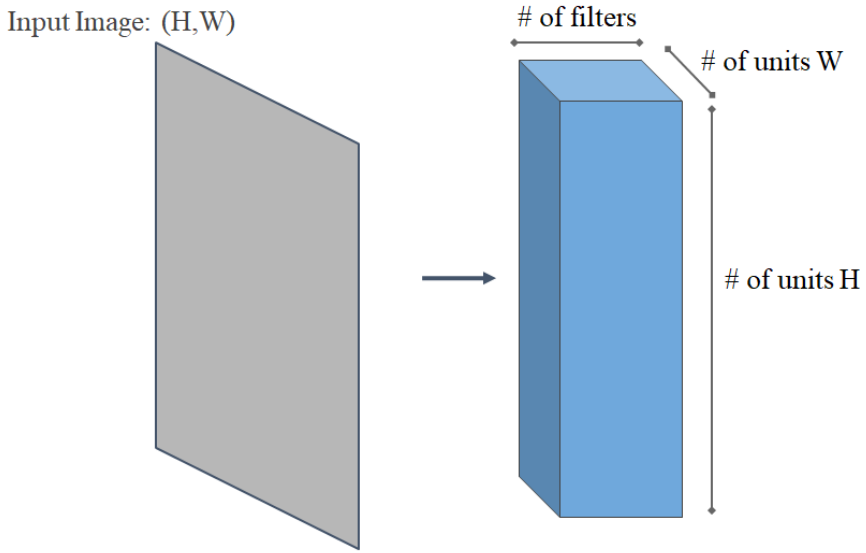
CNN
Filters: 4
Filter Size: 2
Stride: 2

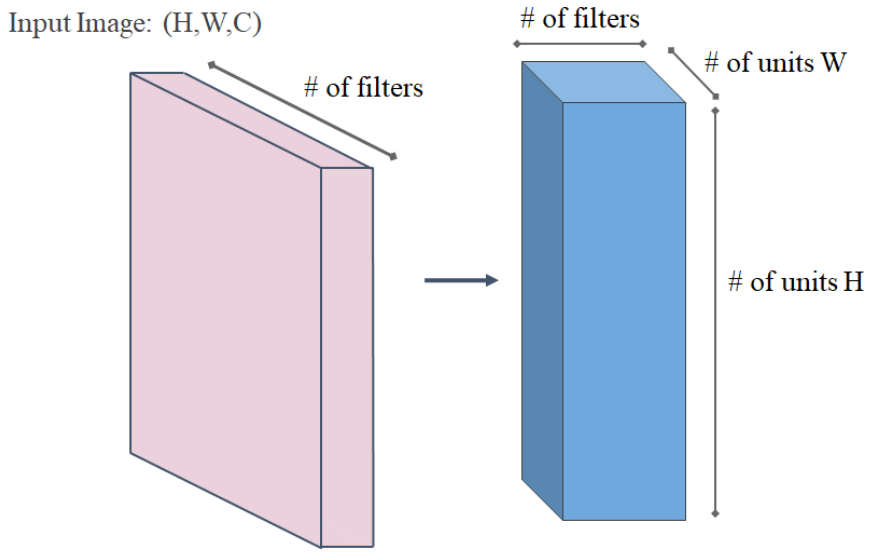


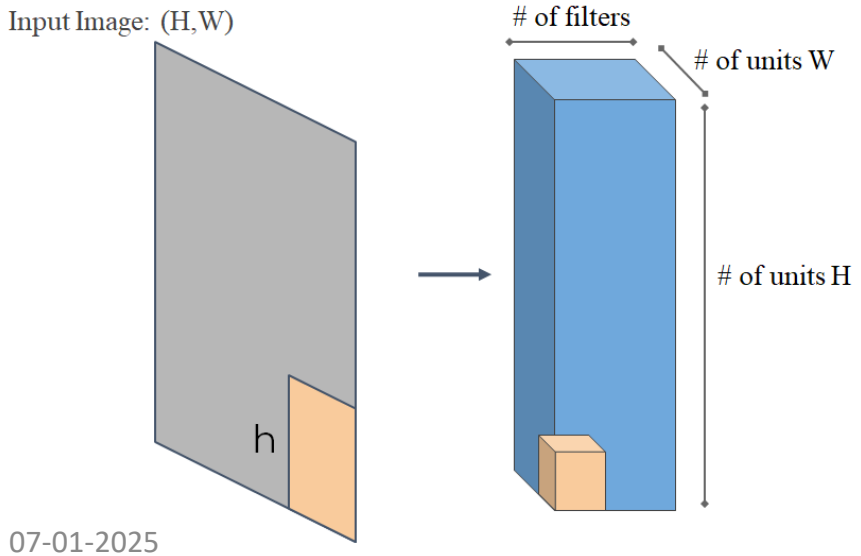
visualizing the neurons as blocks

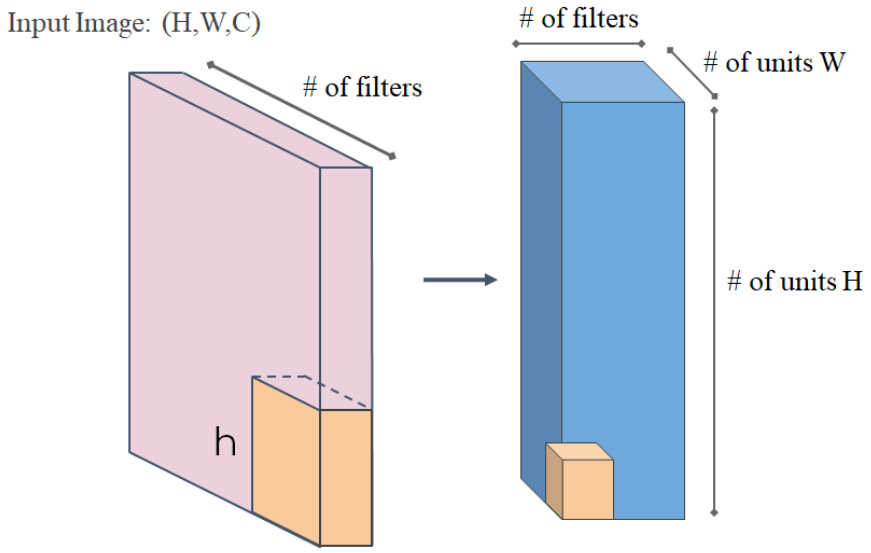
Note: Each filter is detecting a different feature

8.4.2 Convolutional Neural Network



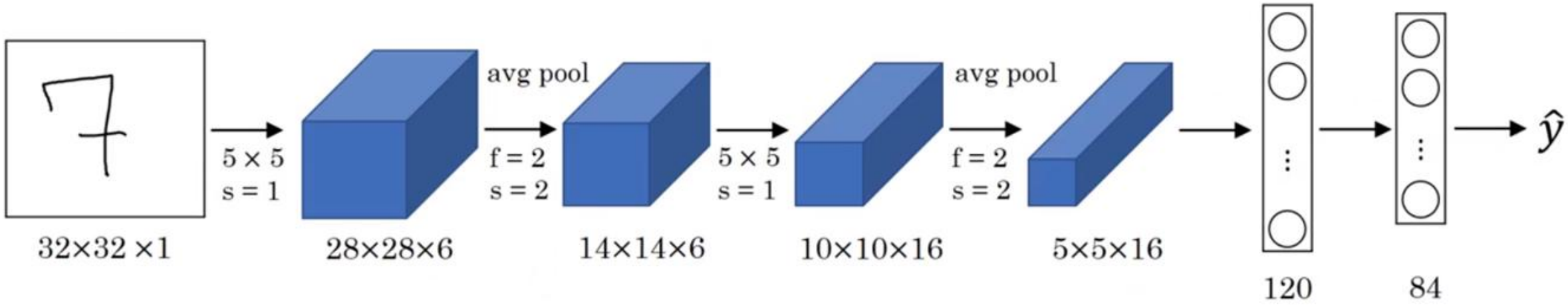






8.4.2 Convolutional Neural Network

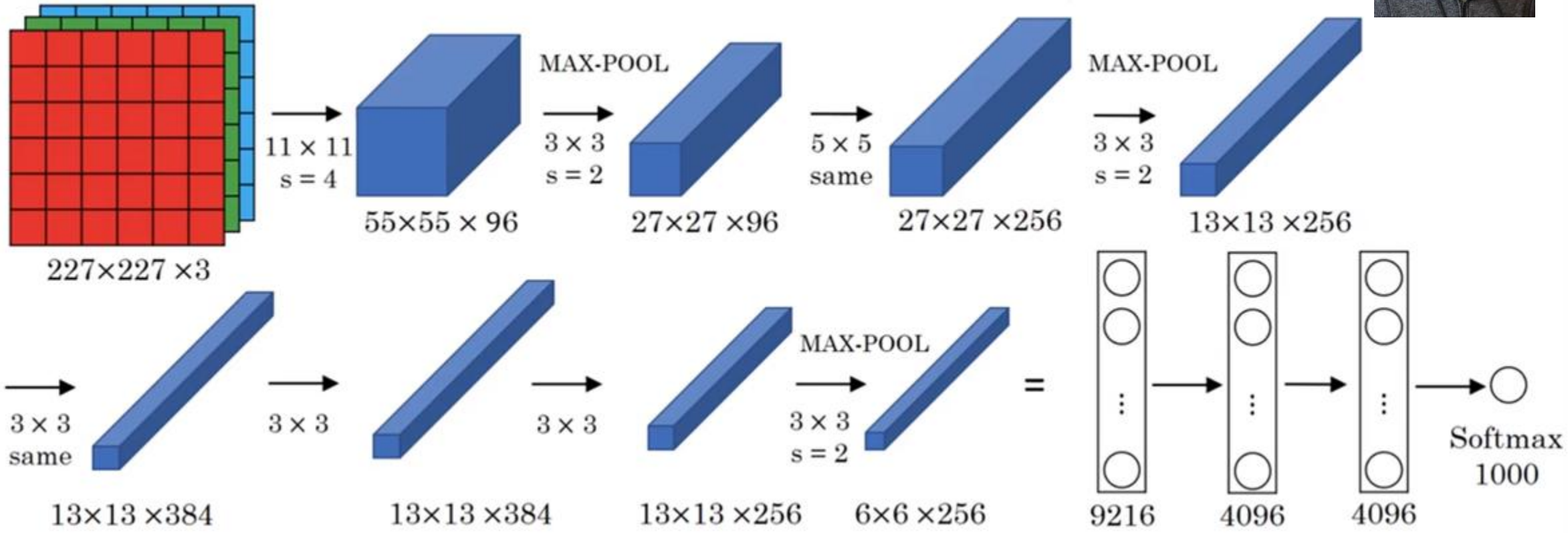
LeNet-5 [Yann LeCun]



- The LeNet-5 architecture was first used to classify the MNIST data set.

8.4.2 Convolutional Neural Network

AlexNet-5 [Alex Krizhevsky et al]

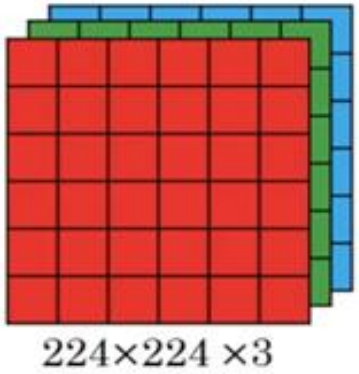


8.4.2 Convolutional Neural Network

VGG-16 [Karen Simonyan and Andrew Zisserman]

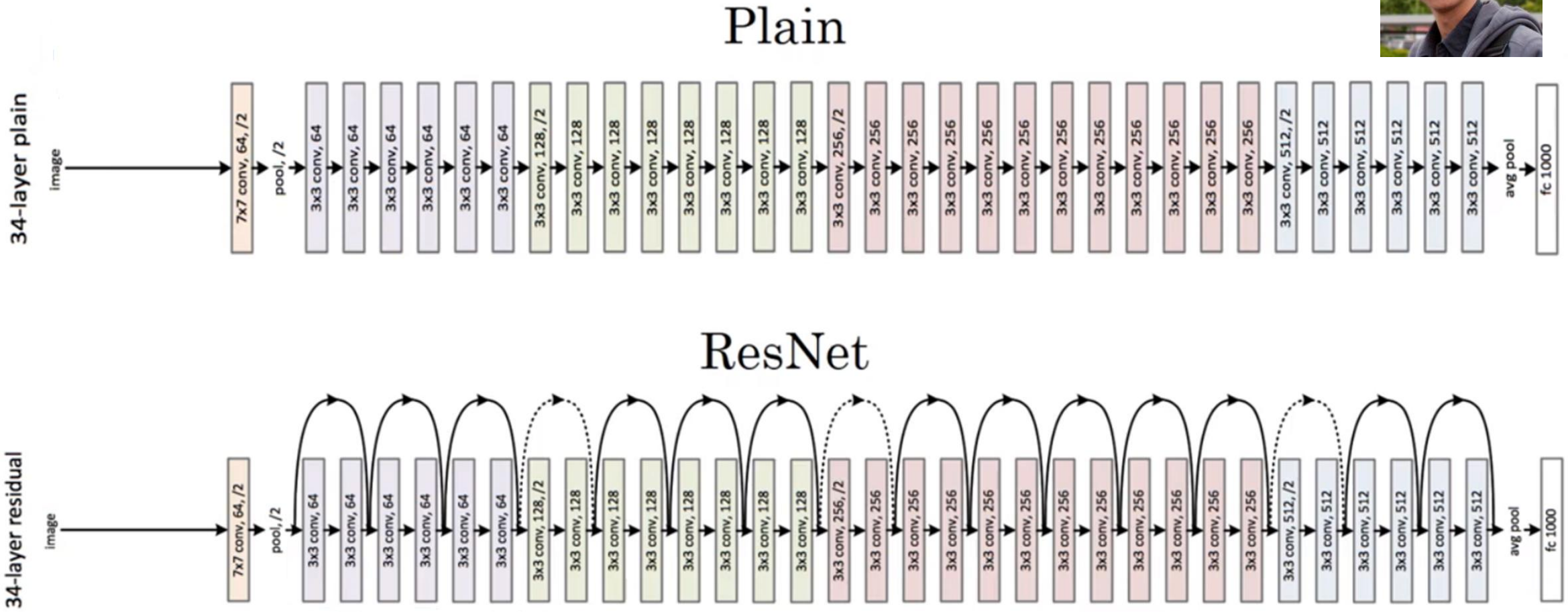
CONV = 3x3 filter, s=1

MAX-POOL = 2x2, s=2



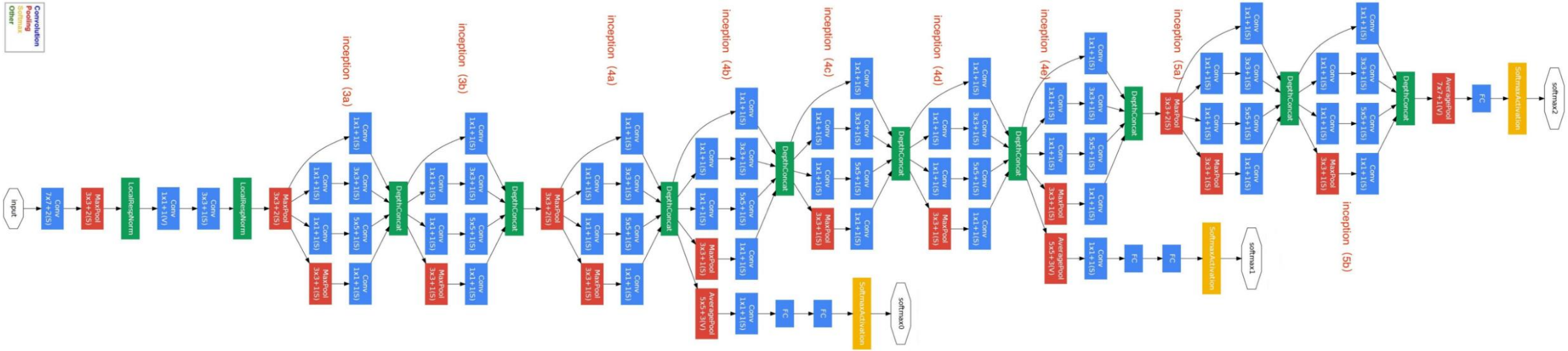
8.4.2 Convolutional Neural Network

ResNet [Kaiming He et al.]

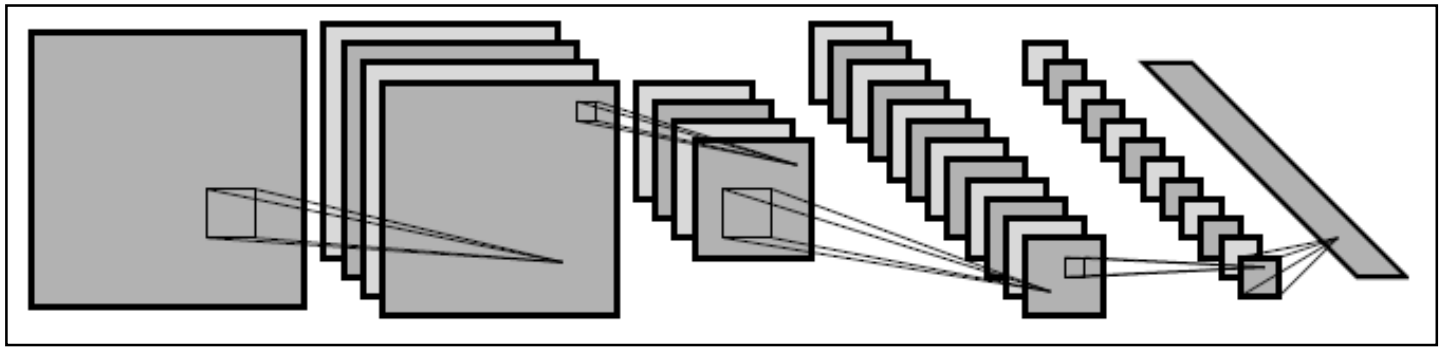


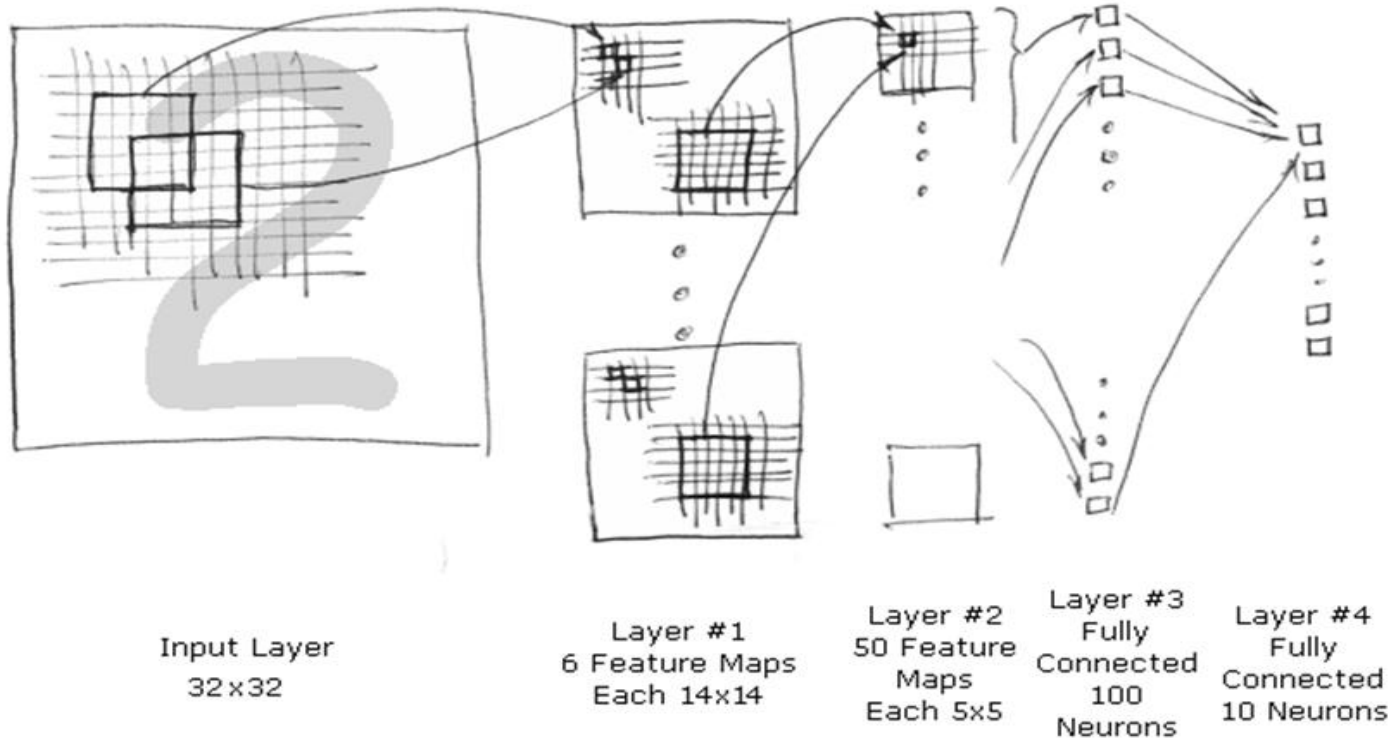
8.4.2 Convolutional Neural Network

GoogLeNet [Christian Szegedy at Google Research]



8.4.2 Convolutional Neural Network





8.4.2 Convolutional Neural Network

CNN disadvantage

- From a memory and capacity standpoint, CNN is not much bigger than a regular two layer ANN.

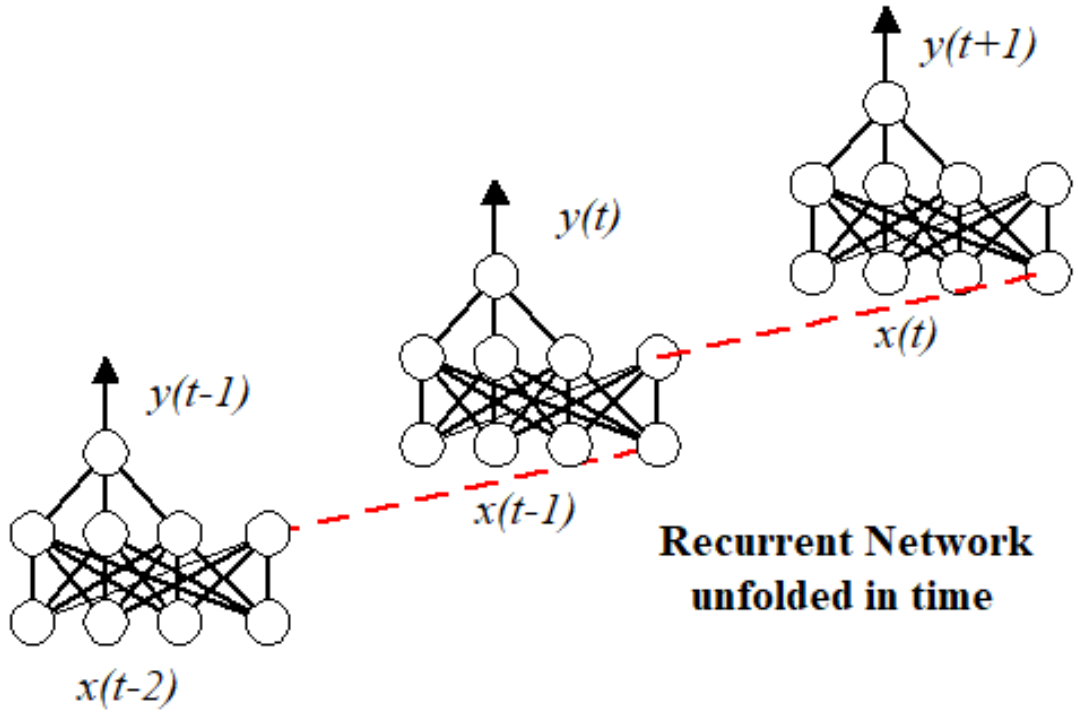
- Convolution operations are computationally expensive and take about 67% of the runtime.
- CNN's are about 3X slower than their fully connected equivalents (size-wise).
- Convolution operation 4 nested loops (2 loops on input image & 2 loops on kernel)
- Small kernel size make the inner loops very inefficient as they frequently JMP.

- Cash unfriendly memory access because
  - Back-propagation require both row and column-wise access to the input and kernel image.
  - 2-D Images represented in a row-wise-serialized order.
  - Column-wise access to data can result in a high rate of cash misses in memory subsystem.

8.4.3 Recurrent Neural Network

RNN Through Time

- Employ feedback (positive, negative, or both)
- Can learn temporal patterns (time series)



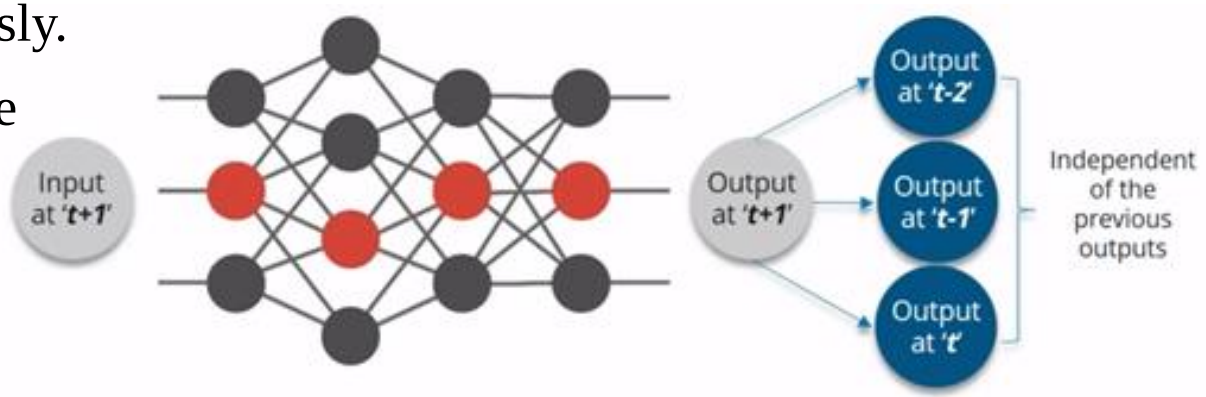
- Recurrent backpropagation networks: for small sequences, unfold network in time dimension and use backpropagation learning

8.4.3 Recurrent Neural Network

Why not regular ANN or CNN?

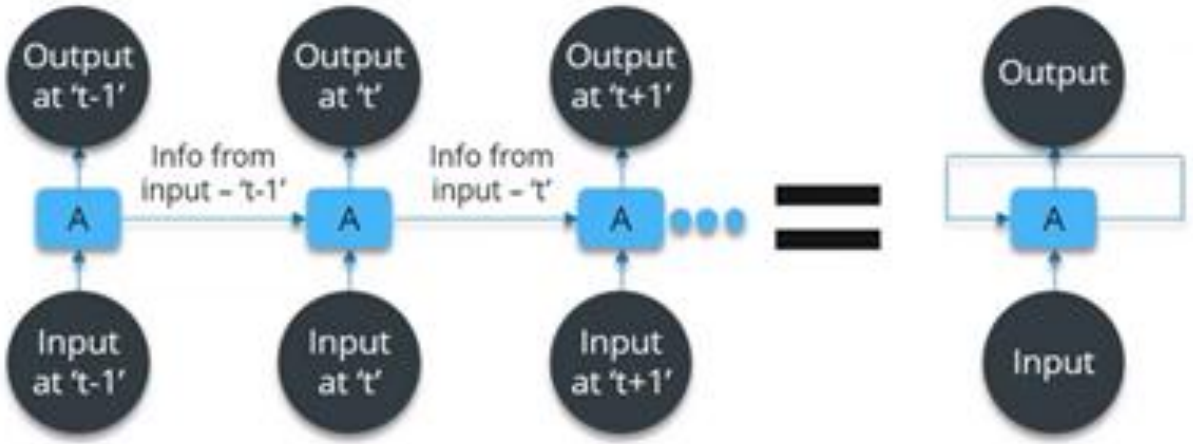
- Regular ANN is good at dealing numerical data. ***For example*** for weather prediction.
- CNN is good at dealing image data. ***For example*** for a image predicting or classifying the object.
- RNN is for sequential data. ***For example*** video, language, speech.

- ANNs and CNNs have
  - Fixed size inputs.
  - The whole input available simultaneously.
- But in video, language, speech data we have
  - Variable sized input
  - Sequential time-series information
- So Regular NN or CNN unable to handle.

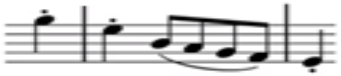


8.4.3 Recurrent Neural Network

How RNN handles?

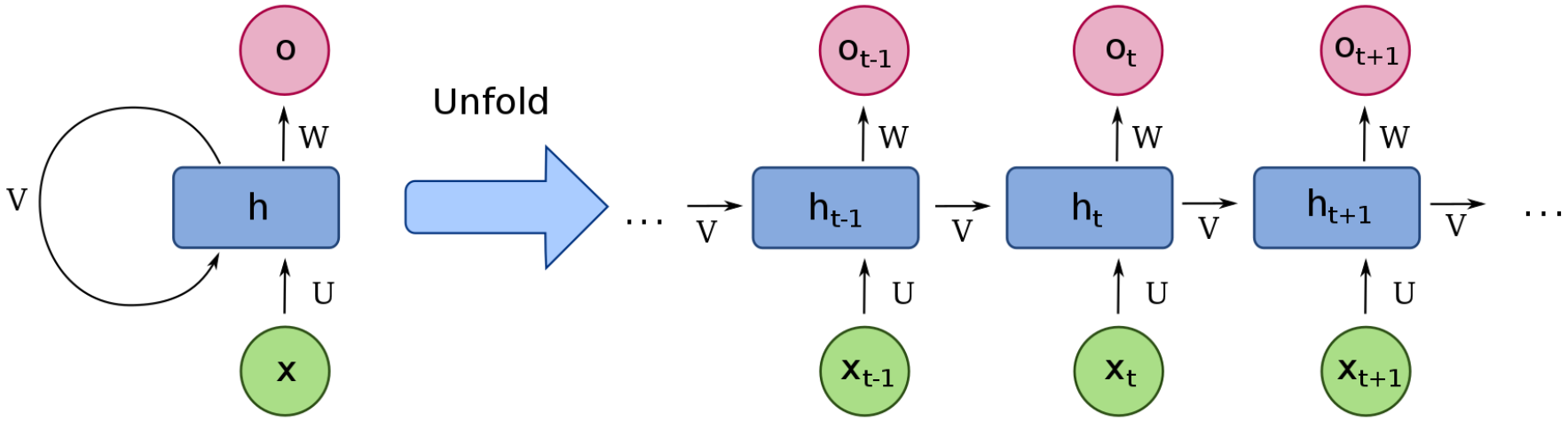


Examples of Sequence Data

- Video Activity Recognition  → A group young boy playing
- Machine Translation Assam is a beautiful state → অসম এখন ধুনীয়া ৰাজ্য
- Speech Recognition  → The class going on
- Music Generation ∅ → 
- DNA Sequence Analysis AGCCCCTGTGAGGAACTAG → AGCCCCTGTGAGGAACTAG

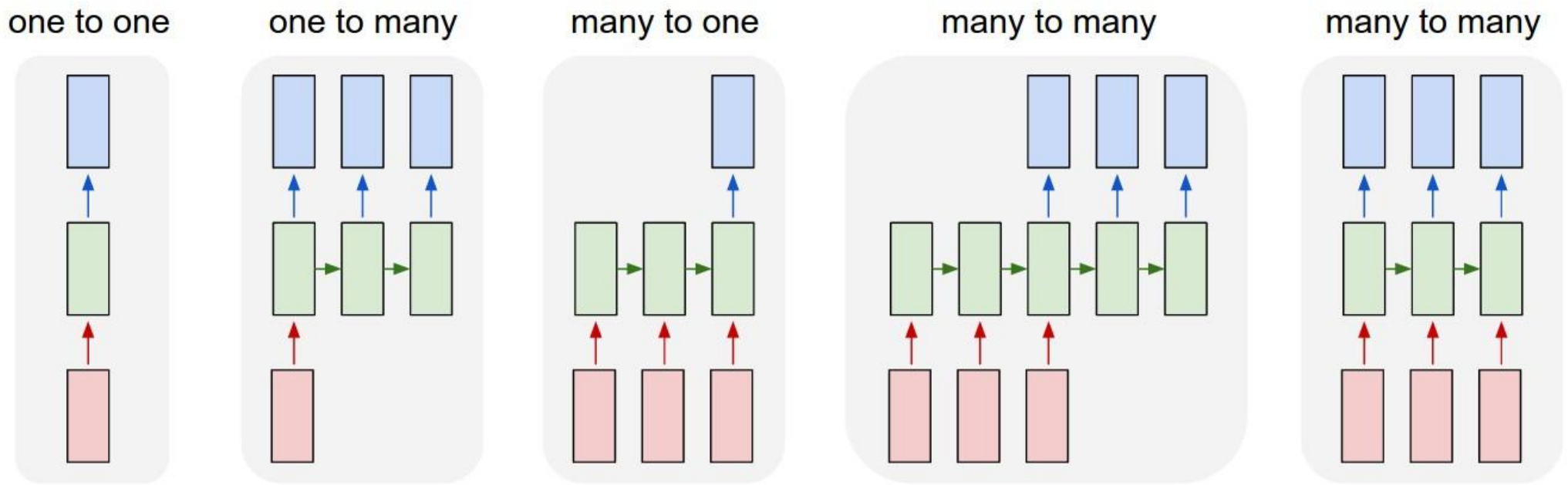
8.4.3 Recurrent Neural Network

- One hidden vector depends on previous hidden vector and input vector at that particular time.
- Cells that are a function of inputs from previous time steps are also known as *memory cells.*



Size of h may be any number of neurons

8.4.3 Recurrent Neural Network

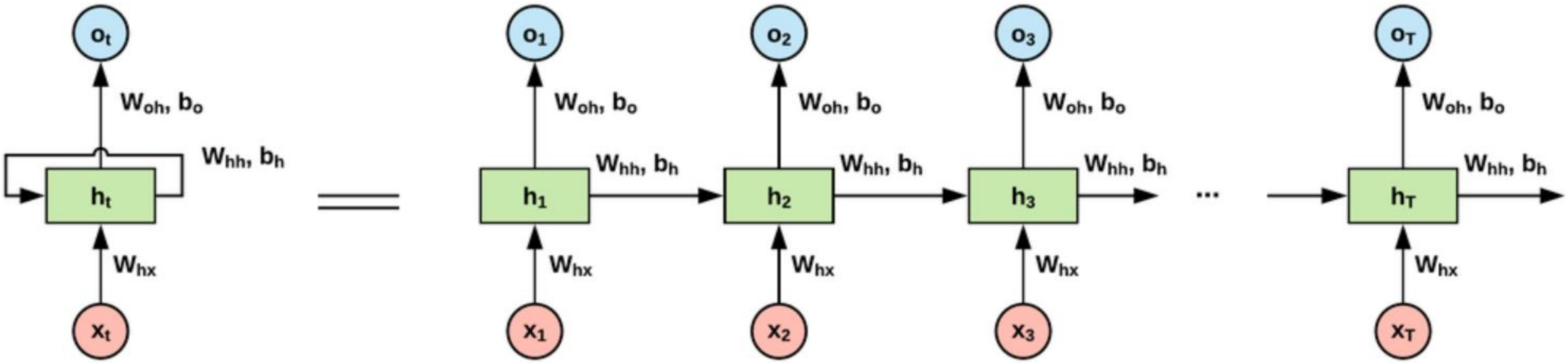


One to many: Vector to Sequence
(image →seq of words)
(Image captioning)

Many to one: Sequence to Vector
(seq of words → +ve or -ve)
Sentiment Analysis

Many to many: Sequence to Sequence
(seq of words/audio → seq of words)
(Language translation or speech recognition)

8.4.3 Recurrent Neural Network

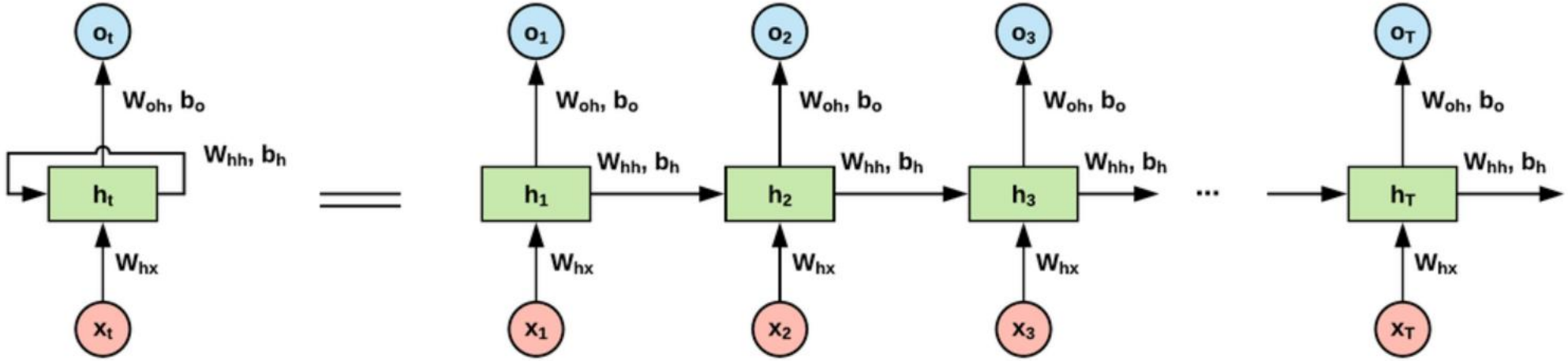


$$h_t = g\,(w_{xh}x_t + w_{hh}h_{t-1} + b_h)$$

$$o_t = g\,(w_{oh}h_t + b_o)$$

8.4.3 Recurrent Neural Network

- Calculating Loss by loss function
- The local loss function we can use either/or –
  - Cross Entropy: Whenever the problem is a classification problem.
  - Least Squares: whenever the problem is a regression problem.
- So the total loss is actually summation of all the intermediate losses through the layers.
- Note: It also depends whether we are take out outputs at all intermediate layers also or not.

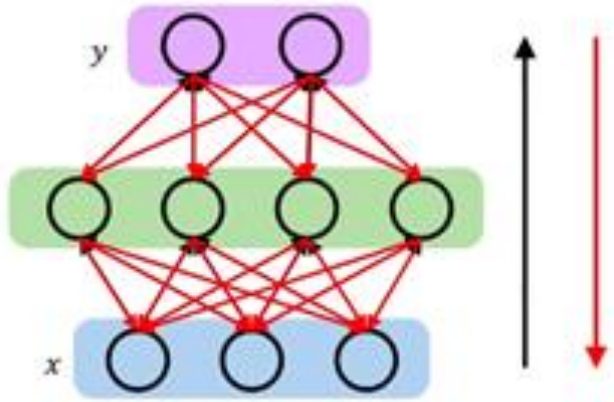


$L = \sum_{i=1}^{t} L_t$ L is total number of layers

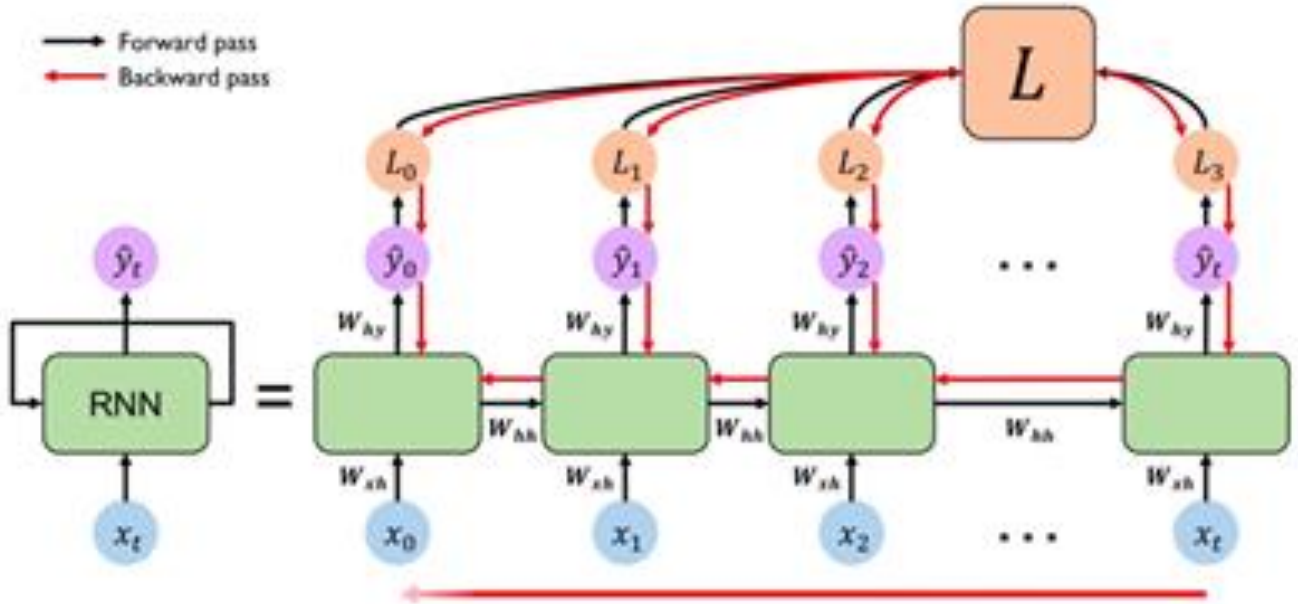
8.4.3 Recurrent Neural Network

Recall: Backpropagation

- Take the derivative (gradient) of the loss w.r.t. each parameter.
- Shift parameters in order to minimize loss.



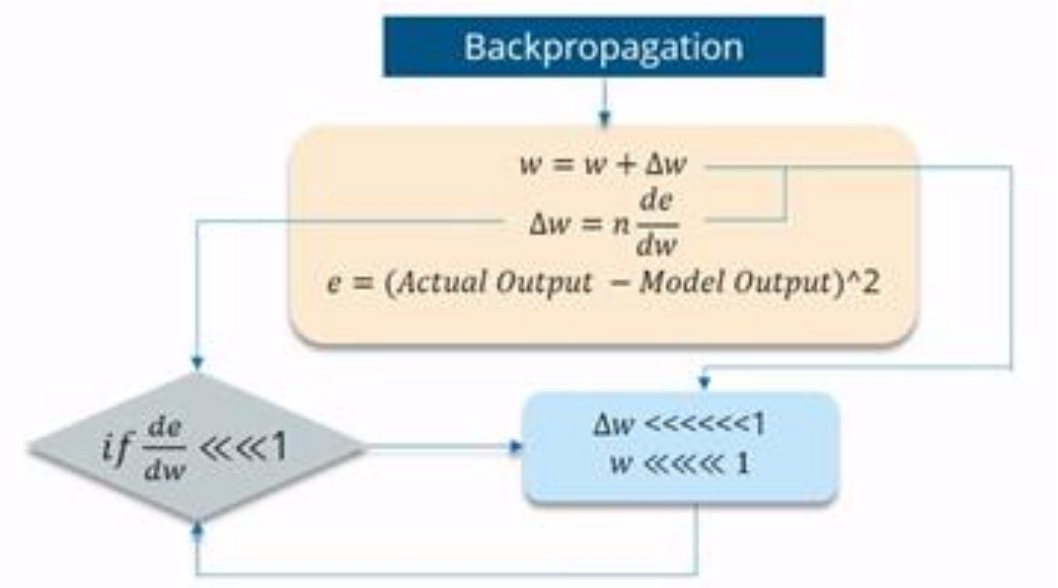
Backpropagation through time (BPTT) (as it is applied for every time stamp)



8.4.3 Recurrent Neural Network

Vanishing Gradient

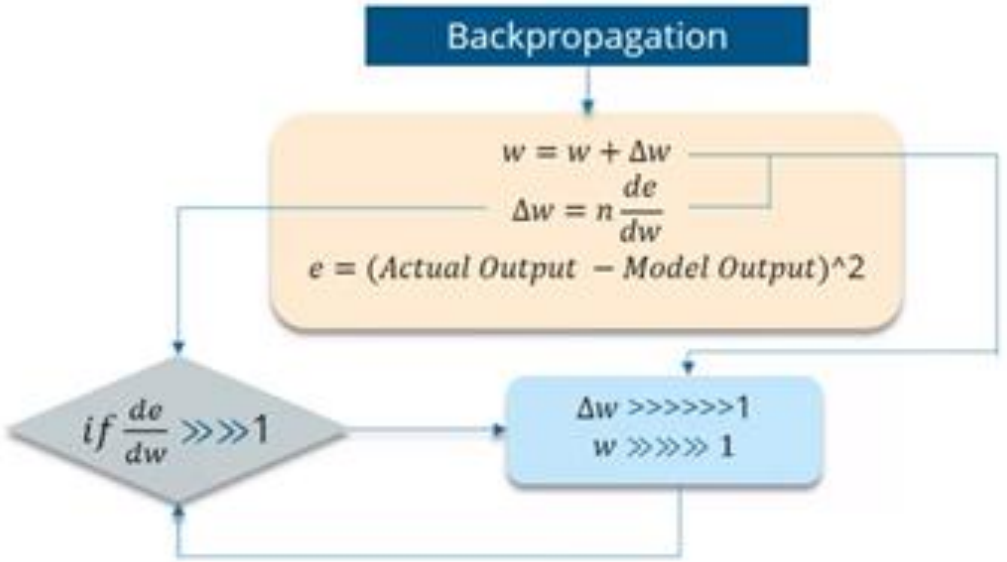
Errors due to further back time steps have smaller and smaller gradients



Solution:

- Activation Function - ReLu
- Architecture (Gated cell to track what information is passed through) - LSTM, GRU

Exploding Gradient



Solution:

- Gradient clipping to scale big gradient

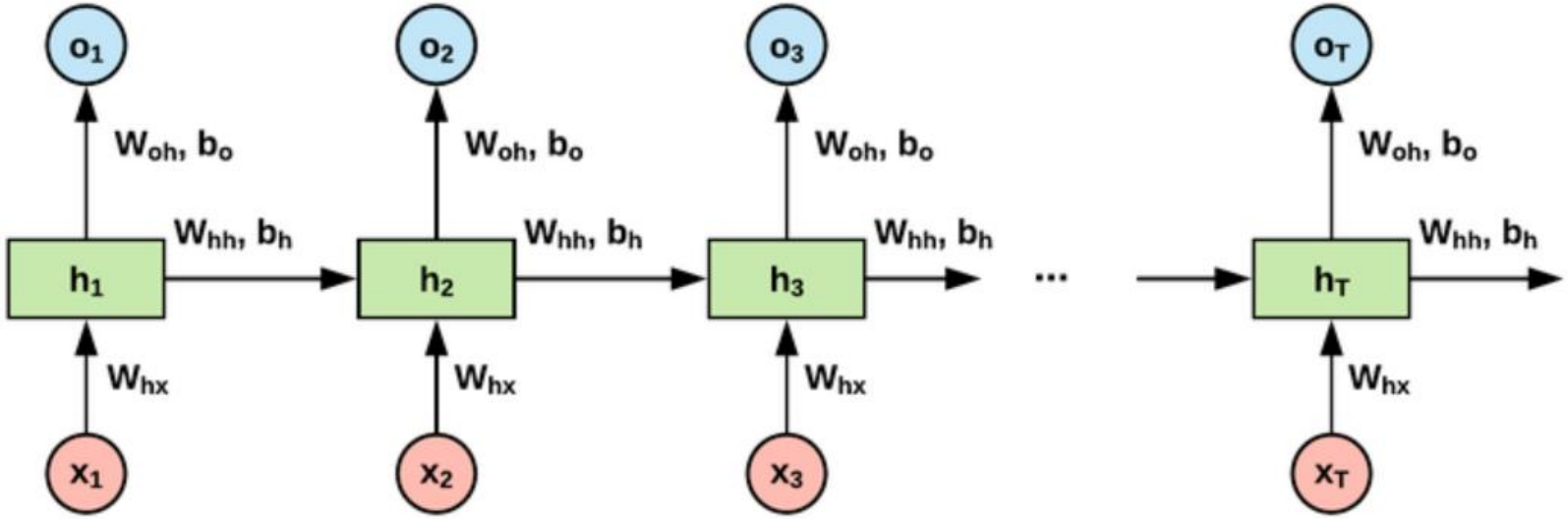
8.4.3 Recurrent Neural Network

- After awhile the network will begin to “forget” the first inputs, as information is lost at each step going through the RNN.
- We need some sort of “long-term memory” for our networks.

Example:

The cat, which actually ate ……., was full.

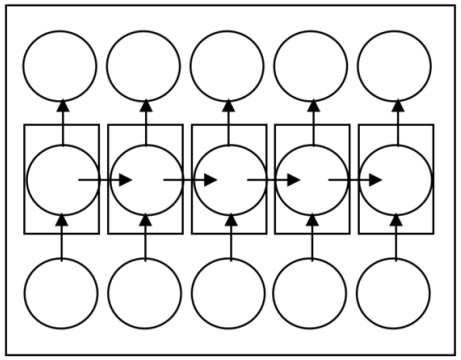
The cat, which actually ate ……., were full.



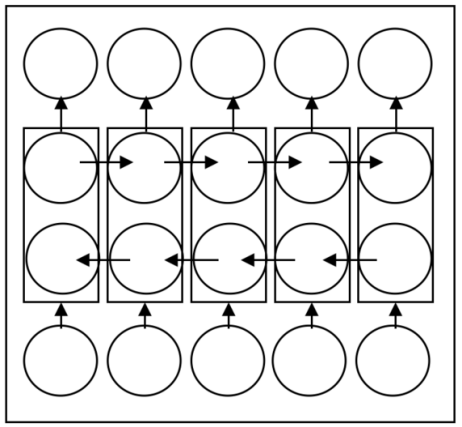
8.4.3 Recurrent Neural Network

- Output may not only depend on the **previous** elements in the sequence, but also **future** elements.
- BRNN uses a finite sequence to predict or label each element of the sequence based on the element's past and future contexts (left and right context).
- This is done by concatenating the outputs of two RNNs, one processing the sequence from left to right, the other one from right to left.
- Especially useful when combined with LSTM RNNs.

Unidirectional RNN



Bidirectional RNN



$$\vec{h}_t = \sigma(\vec{W}^{(hh)}\vec{h}_{t-1} + \vec{W}^{(hx)}x_t)$$

$$\overleftarrow{h}_t = \sigma(\overleftarrow{W}^{(hh)}\overleftarrow{h}_{t+1} + \overleftarrow{W}^{(hx)}x_t)$$

$$y_t = f(\,[\vec{h}_t; \overleftarrow{h}_t\,]\,)$$

past and future around a single token

- Two RNNs stacked on top of each other
- output is computed based on the hidden state of both RNNs $[\vec{h}_t; \overleftarrow{h}_t\,]$

8.4.3 Recurrent Neural Network


LSTM

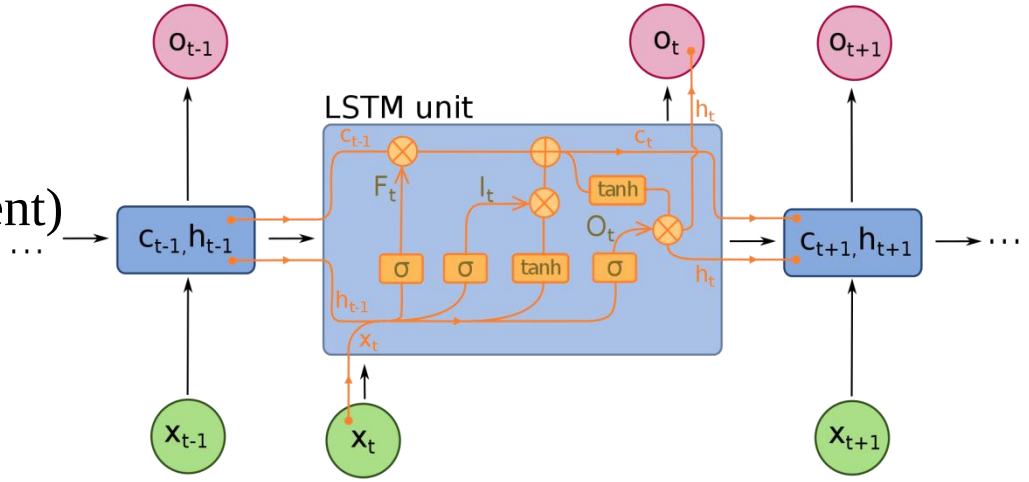
- Introduced by Sepp Hochreiter and Juergen Schmidhuber [1997].
- Special type of RNN to overcome the challenges of training RNN.
- Training RNN for long term dependencies is difficult.
- Relative insensitivity to gap length is an advantage of LSTM over RNNs, HMV and other.

Applications

- In Sequential Data to perform classification, sequence prediction, translation etc. specially for NLP.

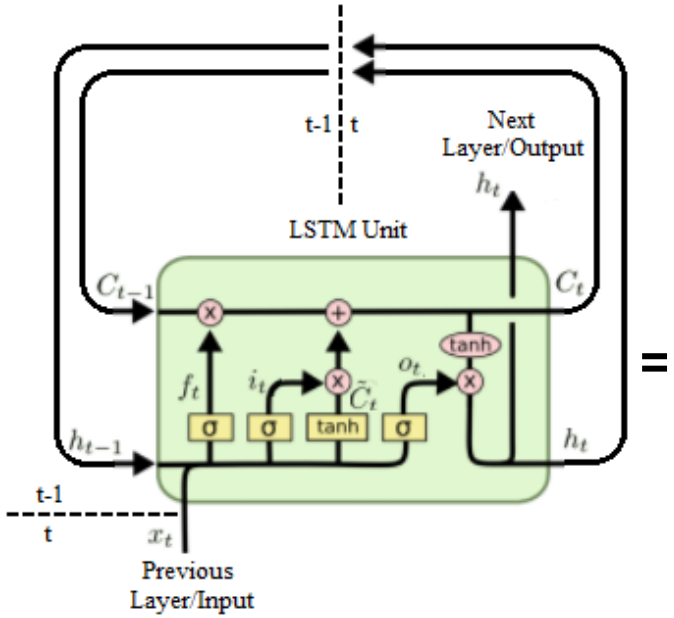
Algorithms to train RNN

- Backpropagation through time (using Gradient Descent)

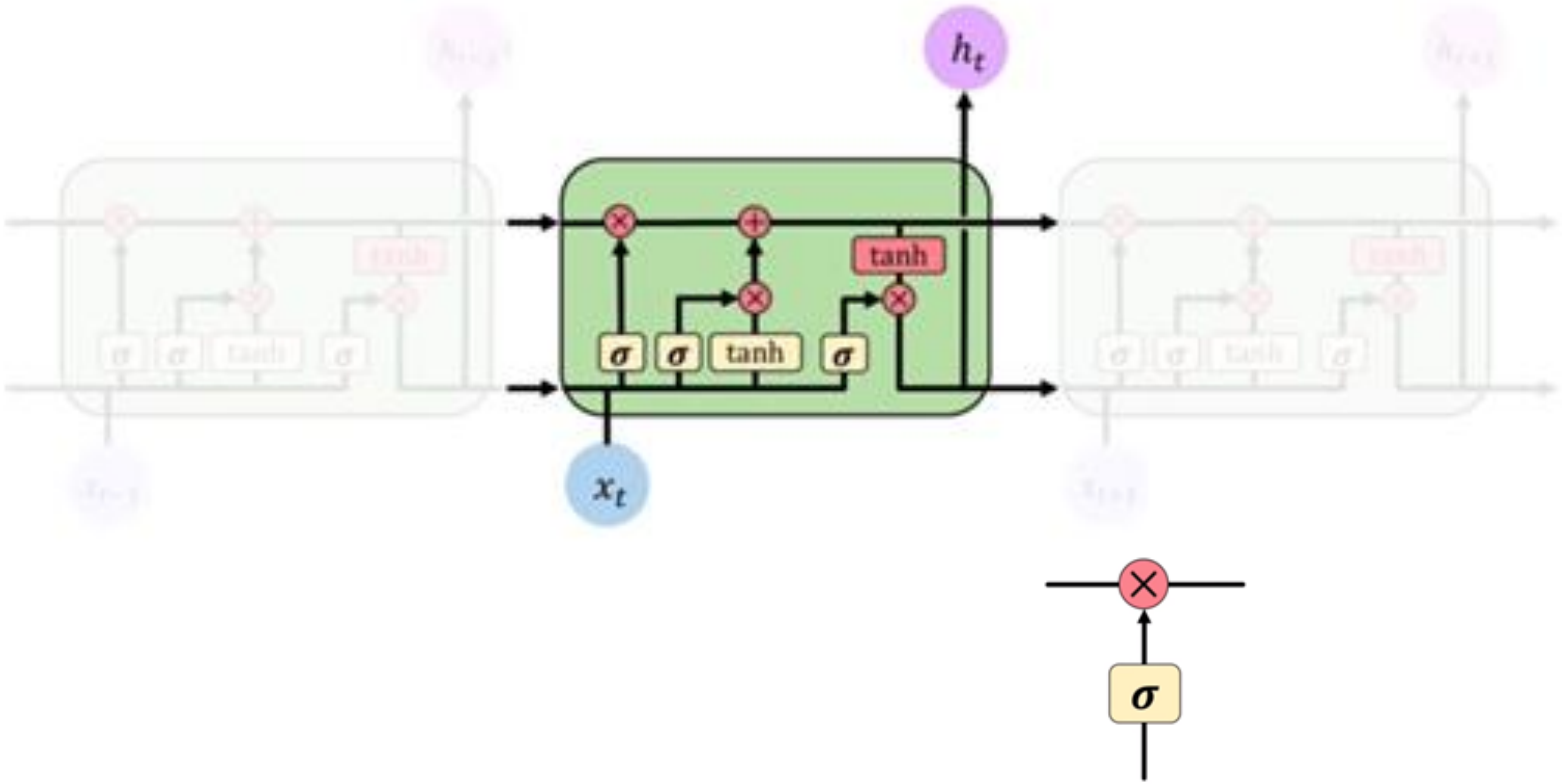


8.4.3 Recurrent Neural Network

- A LSTM unit is composed of a cell, an **input gate**, an **output gate** and a **forget gate**.
- The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.



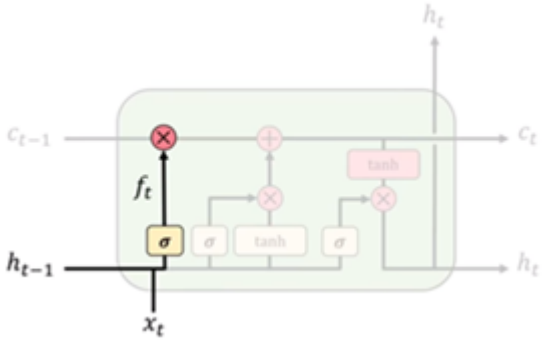
=



8.4.3 Recurrent Neural Network

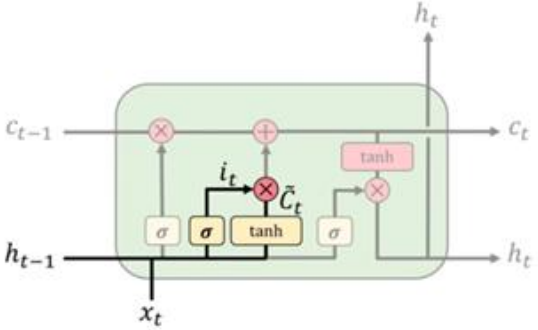
How it Works?

1. **Forget:** Forget irrelevant parts of the previous state
2. **New Input:** Identify new information to be stored
3. **Update:** Selectively update cell state value
4. **Output:** Output certain parts of the cell state



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

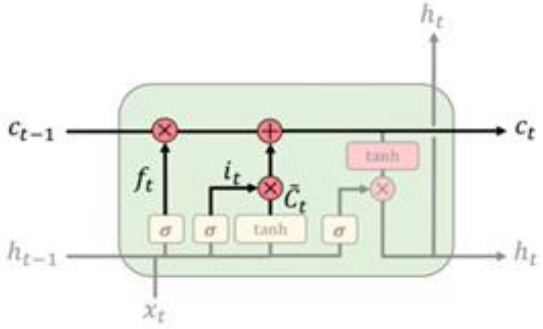
- Use previous cell output & input
- Sigmoid value: 0 for completely forget and 1 for completely keep



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

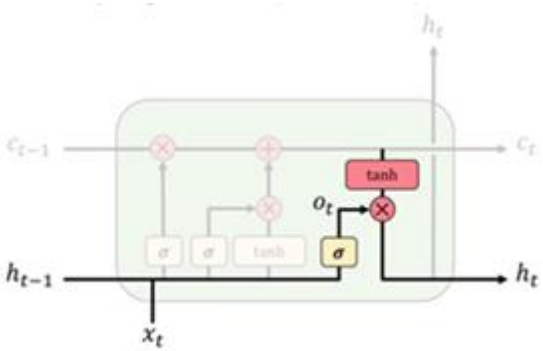
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- Sigmoid layer: decide what value to update
- Tanh layer: generate new vector that can be added



$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

- Apply forget operation to previous internal cell state
- Add new values scaled by how much need to update

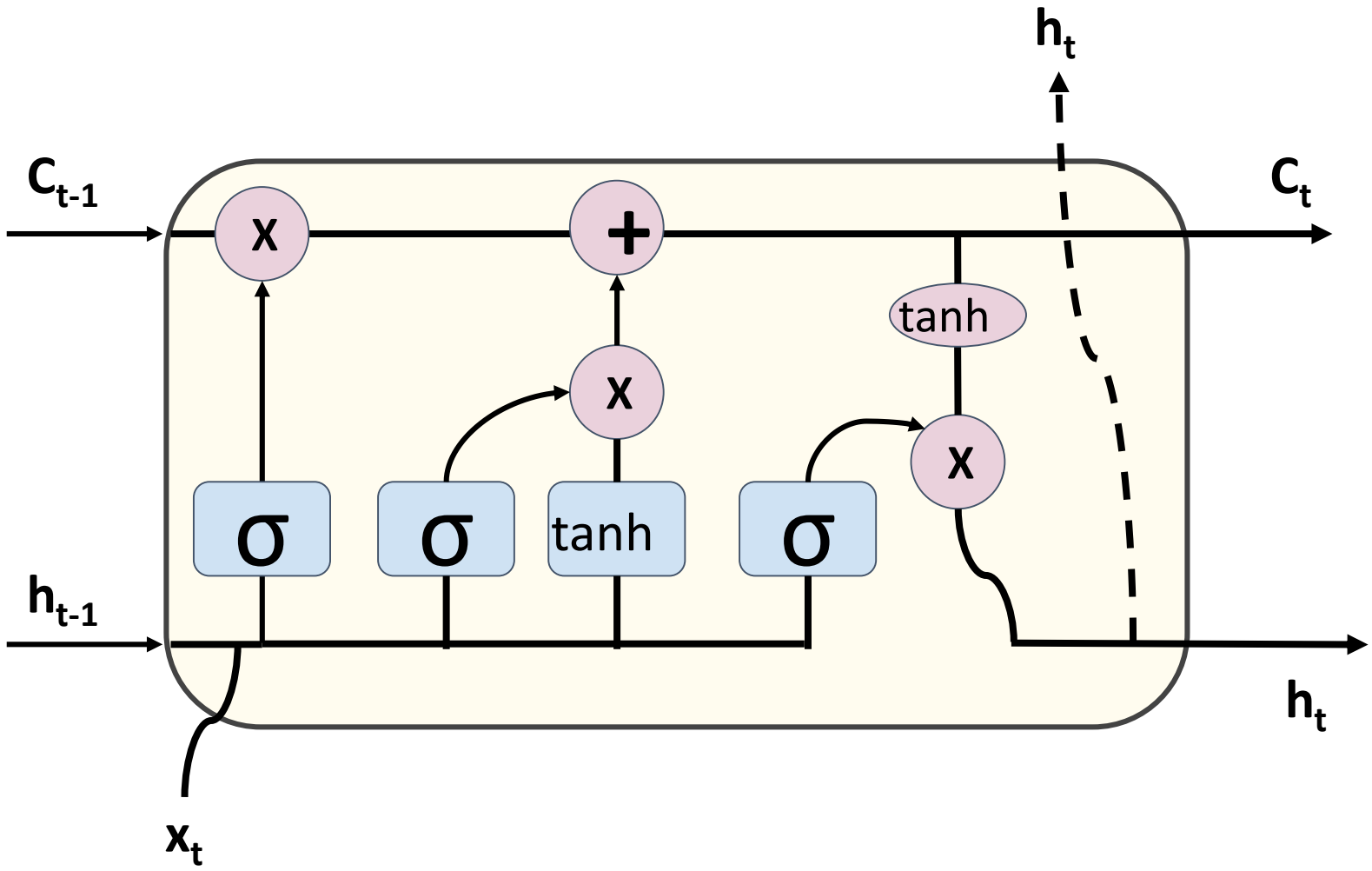


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

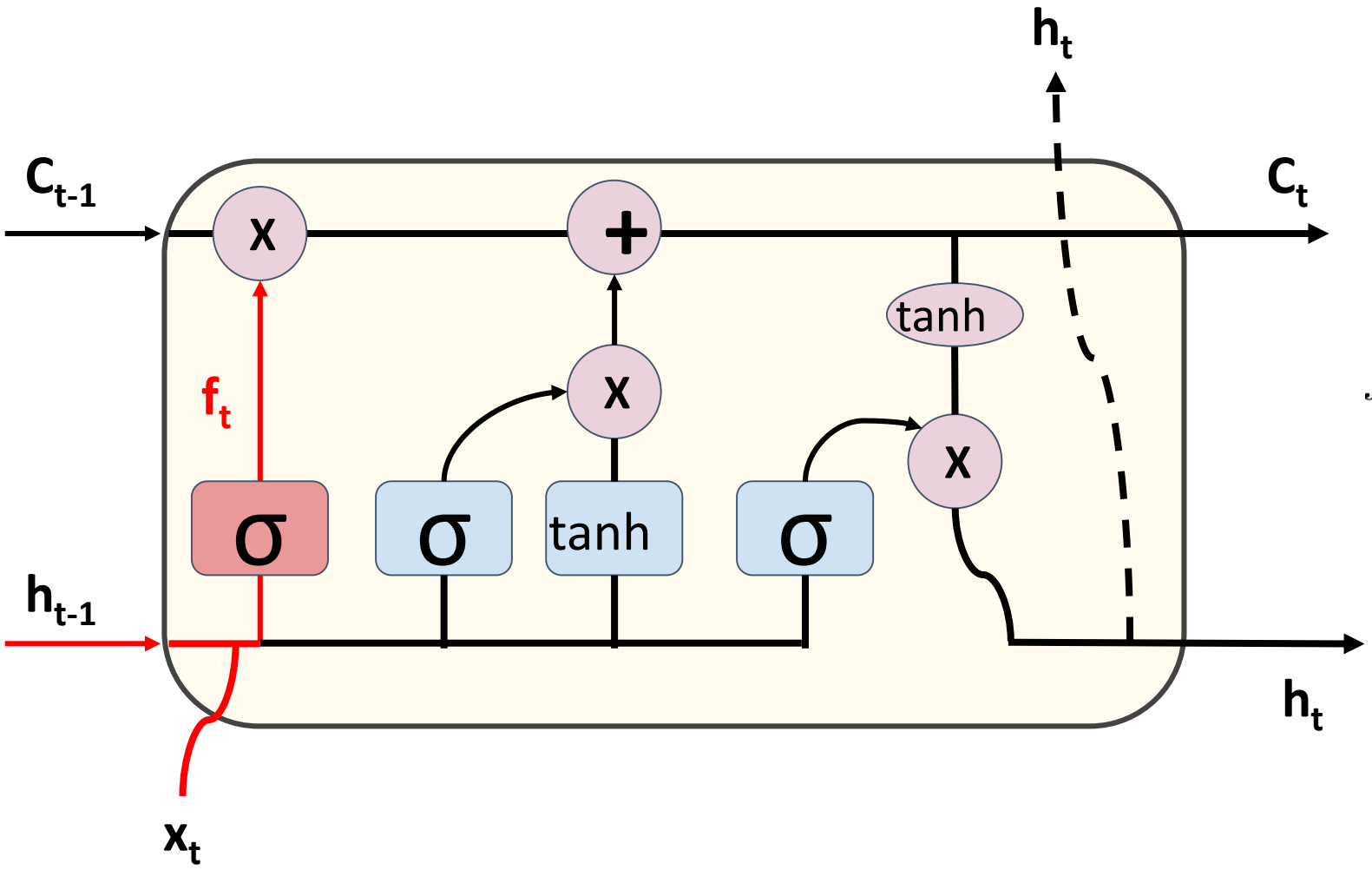
$$h_t = o_t \circ \tanh\left(C_t\right)$$

- Sigmoid layer: decides what parts of state to output
- Tanh layer: squash values between -1 and 1
- h_t: output of cell state

8.4.3 Recurrent Neural Network

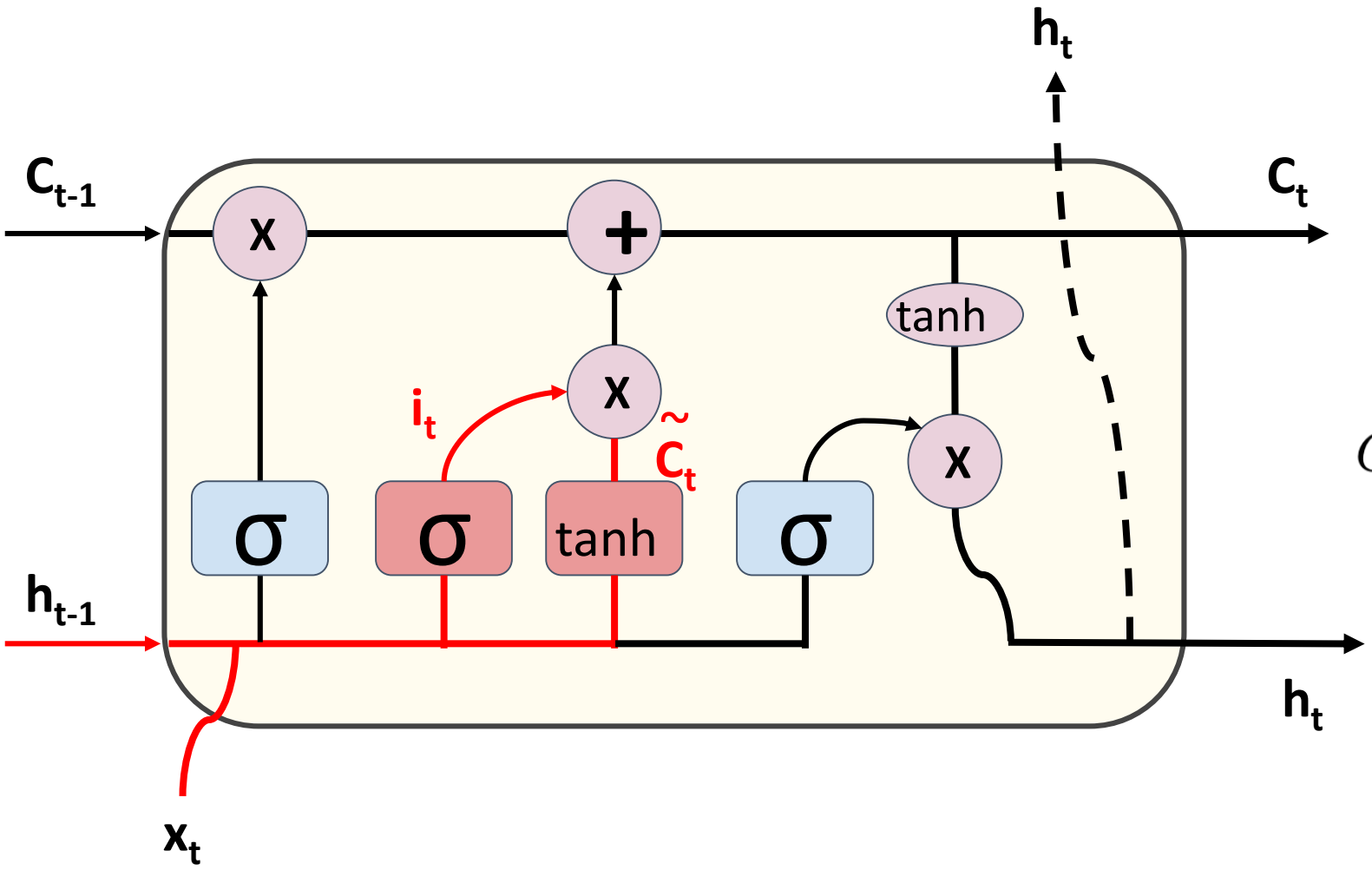


8.4.3 Recurrent Neural Network



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

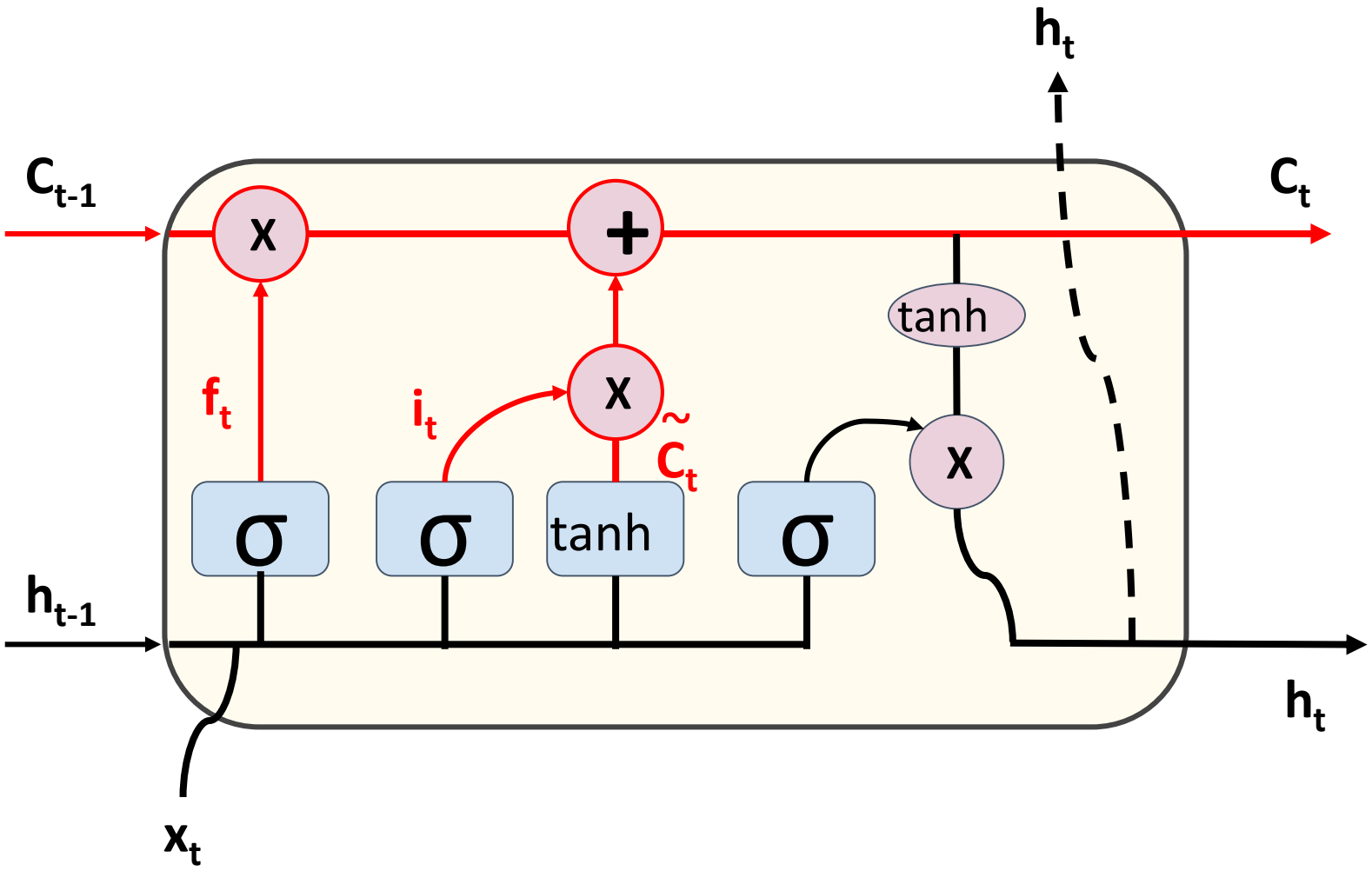
8.4.3 Recurrent Neural Network



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

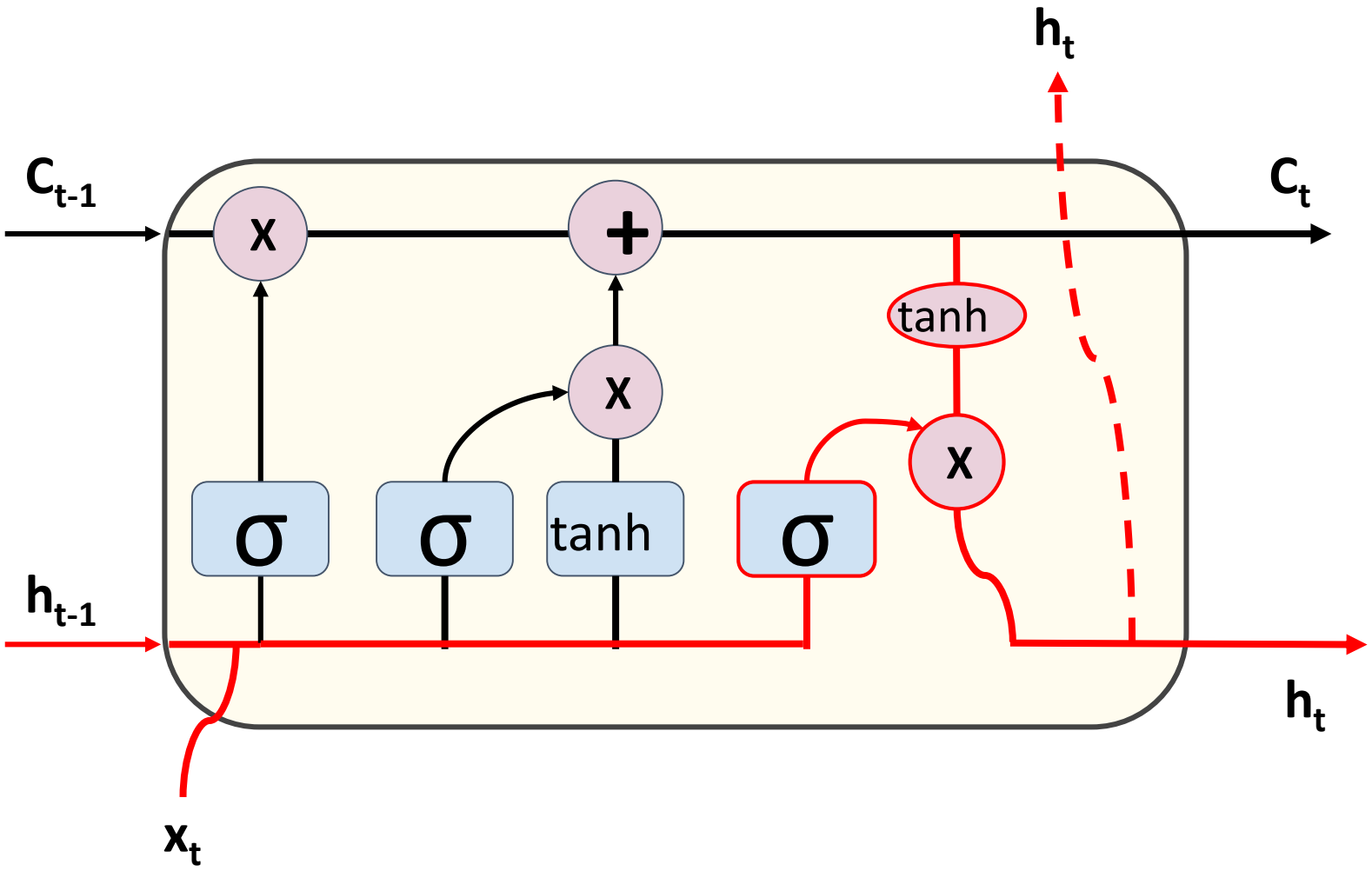
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

8.4.3 Recurrent Neural Network



$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

8.4.3 Recurrent Neural Network

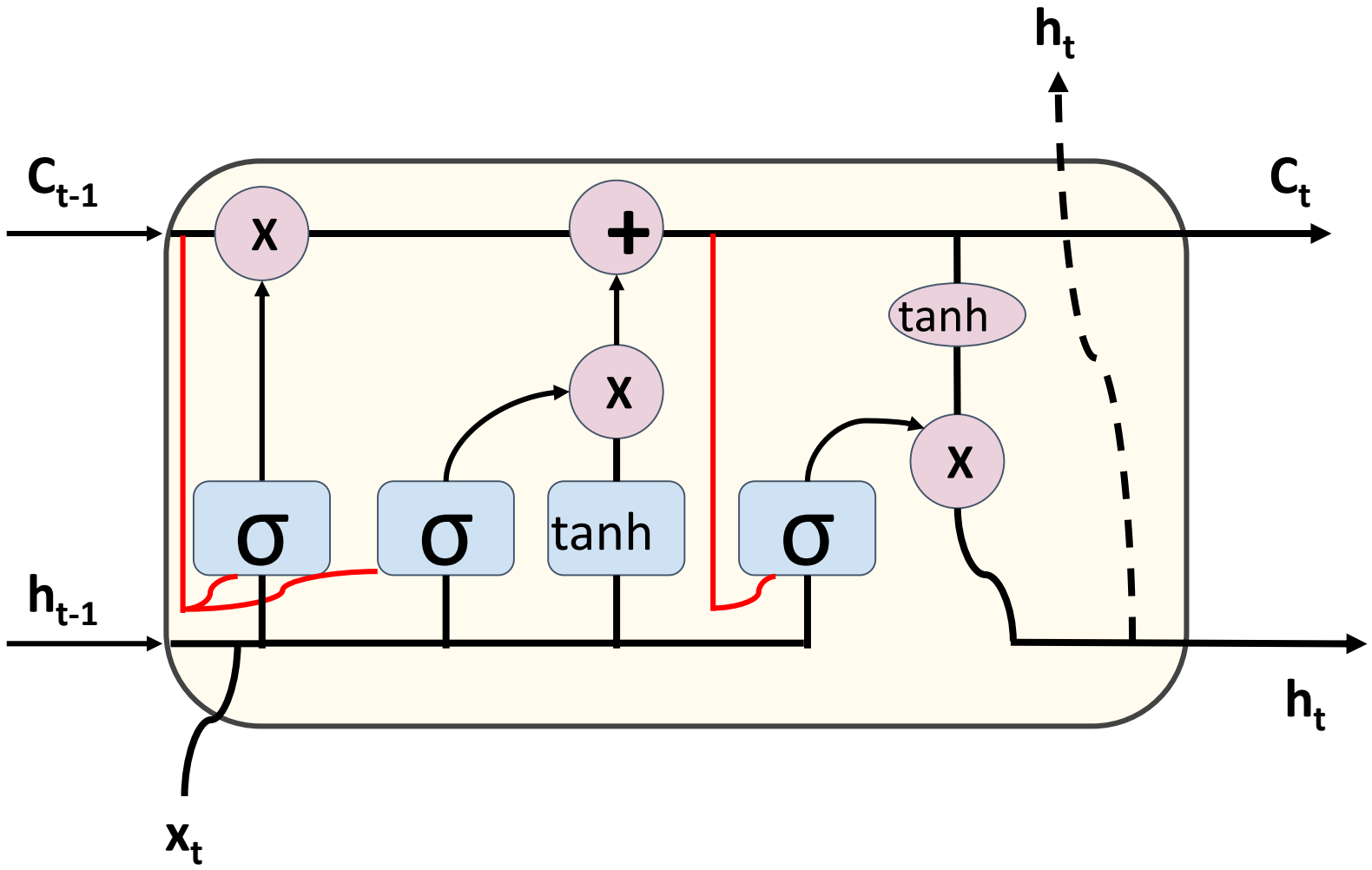


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t \circ \tanh\left(C_t\right)$$

8.4.3 Recurrent Neural Network

Peephole LSTM



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

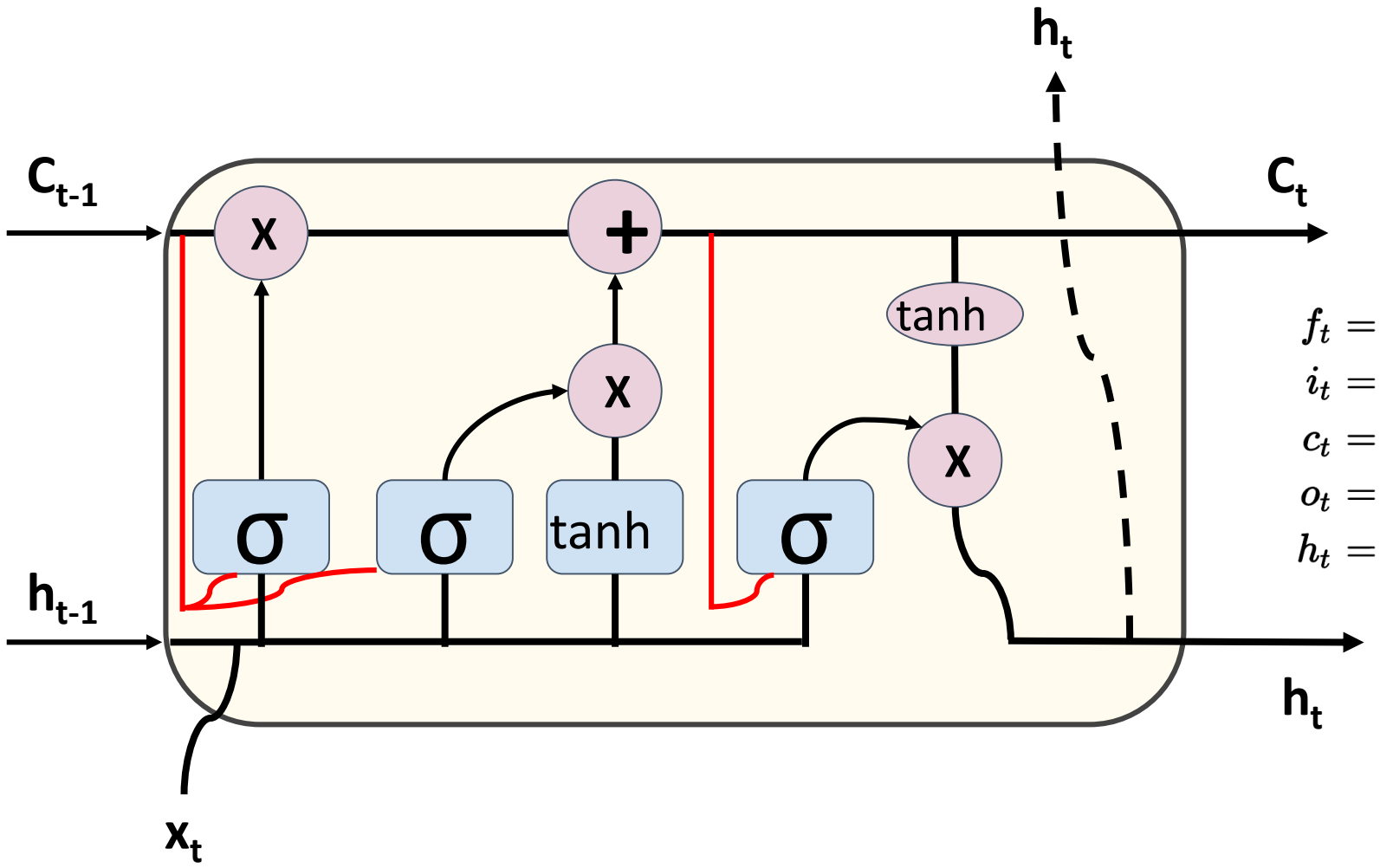
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t \circ \tanh\left(C_t\right)$$

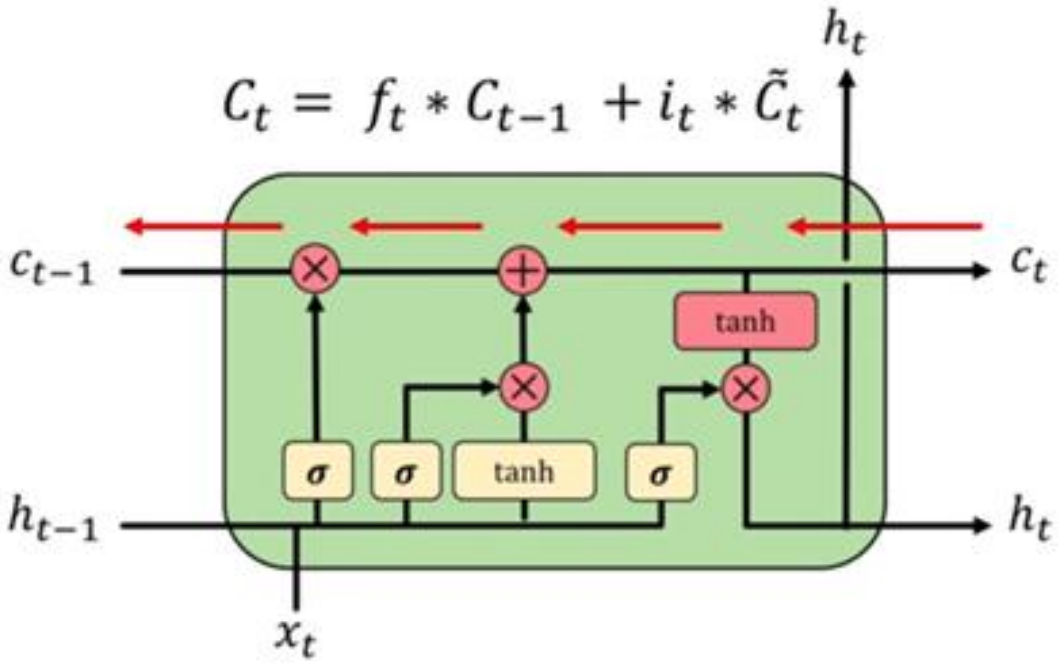
Peephole Convolutional LSTM



$$f_t = \sigma_g(W_f * x_t + U_f * h_{t-1} + V_f \circ c_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i * x_t + U_i * h_{t-1} + V_i \circ c_{t-1} + b_i)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c * x_t + U_c * h_{t-1} + b_c)$$
$$o_t = \sigma_g(W_o * x_t + U_o * h_{t-1} + V_o \circ c_t + b_o)$$
$$h_t = o_t \circ \sigma_h(c_t)$$

8.4.3 Recurrent Neural Network

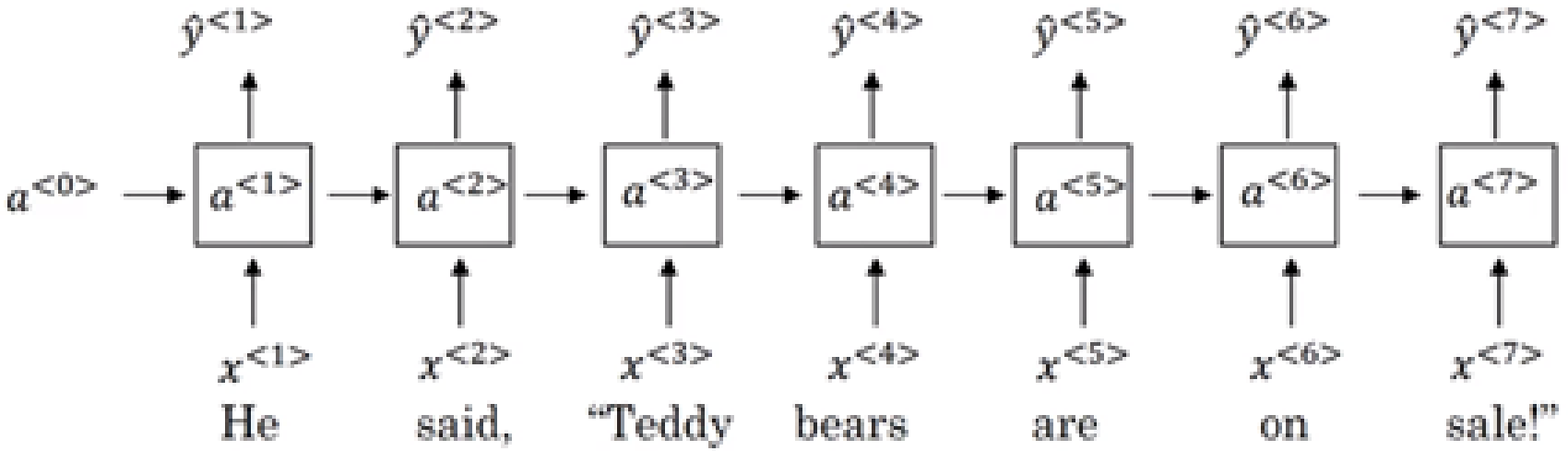
- Backpropagation from C_t to C_{t-1} requires only elementwise multiplication.
- No matrix multiplication → avoid vanishing gradient problem.



Getting Information from Future

He said, "Teddy bears are on sale!"

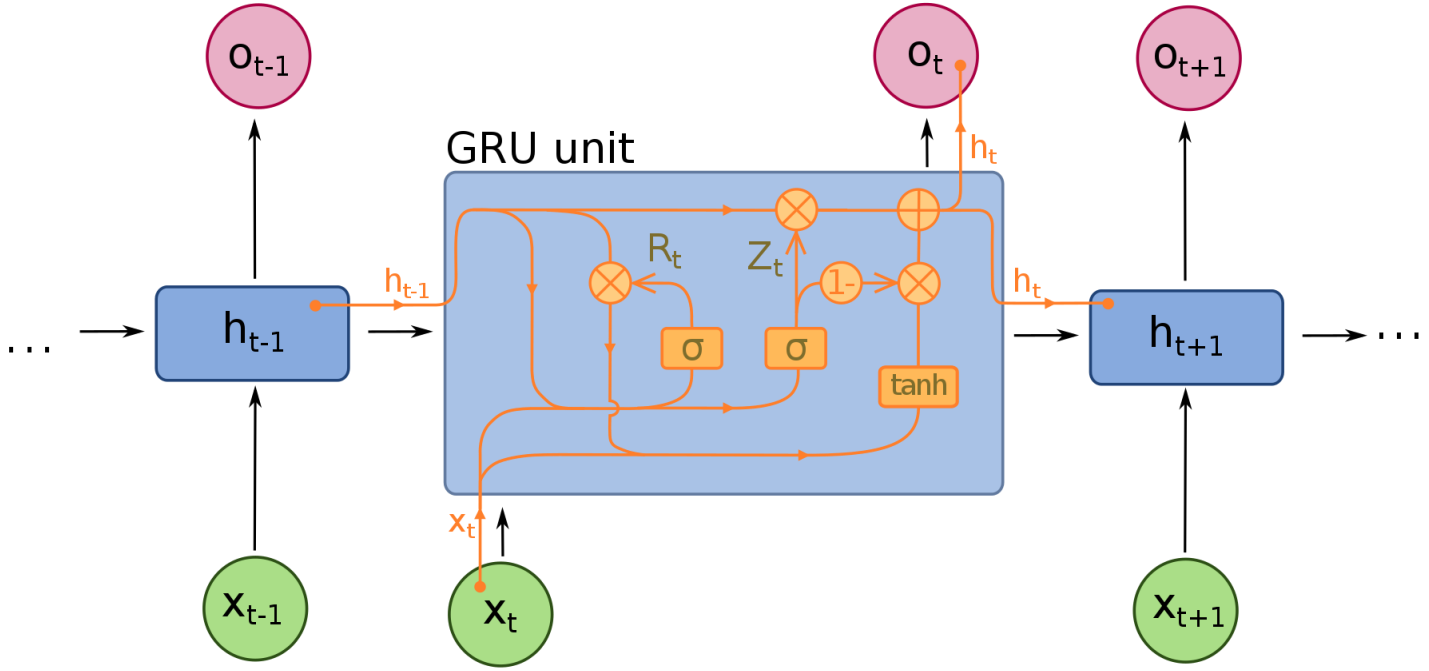
He said, "Teddy Roosevelt was a great president!"



8.4.3 Recurrent Neural Network

GRU

- Introduced by Kyunghyun Cho et. al. [2014] with gating mechanism in RNN.
- They are used in the full form and several simplified variants.
- They have fewer parameters than LSTM, as they lack an output gate.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$R_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [R_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

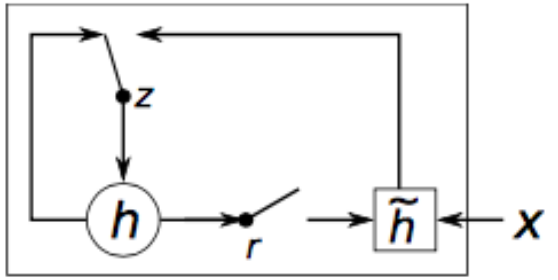
8.4.3 Recurrent Neural Network

- Standard RNN computes hidden layer at next time step directly

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

- Compute an update gate based on current input word vector and hidden state

$$z_t = \sigma(U^{(z)}h_{t-1} + W^{(z)}x_t)$$



- Controls how much of past state should matter now.
- If z close to 1, then copy information in that unit through many steps

- Compute a reset gate similarly but with different weights

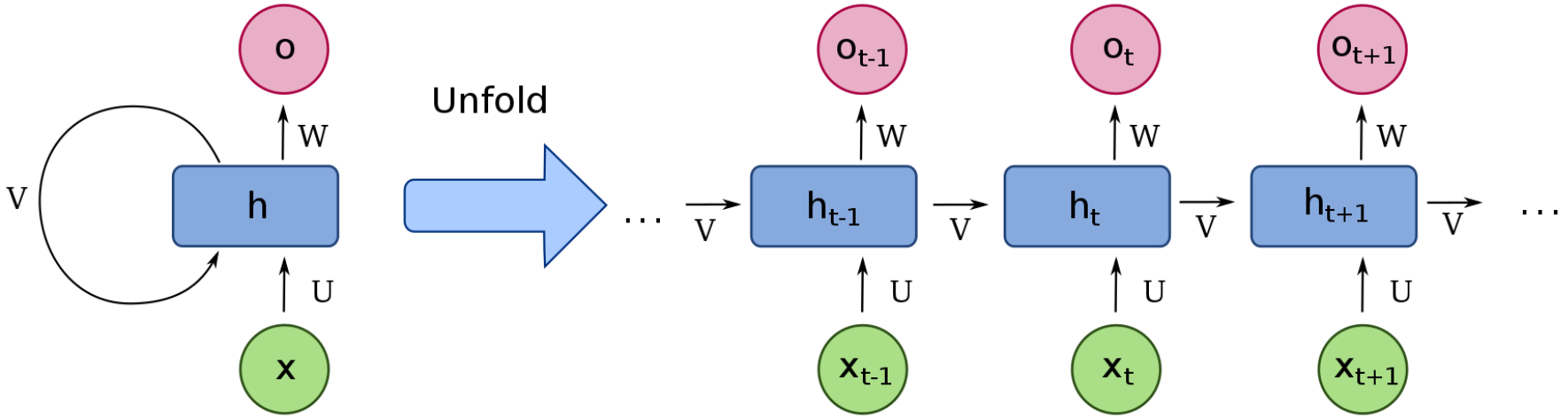
$$r_t = \sigma\left(U^{(r)}h_{t-1} + W^{(r)}x_t\right)$$

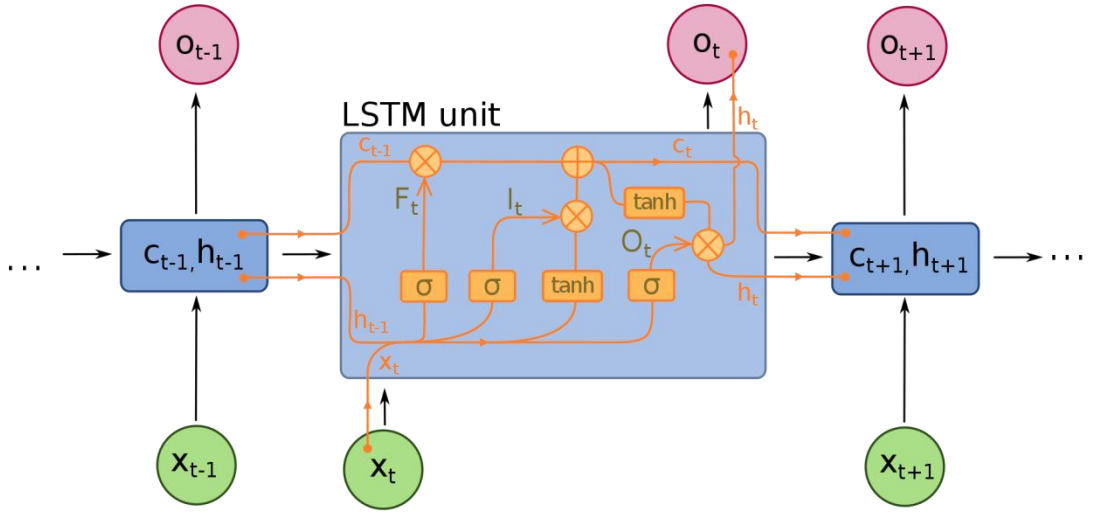
- If reset close to 0, ignore previous hidden state (allows model to drop information that is irrelevant in future)
- Units with short-term dependencies have reset gates very active and long-term dependencies have active update gates z

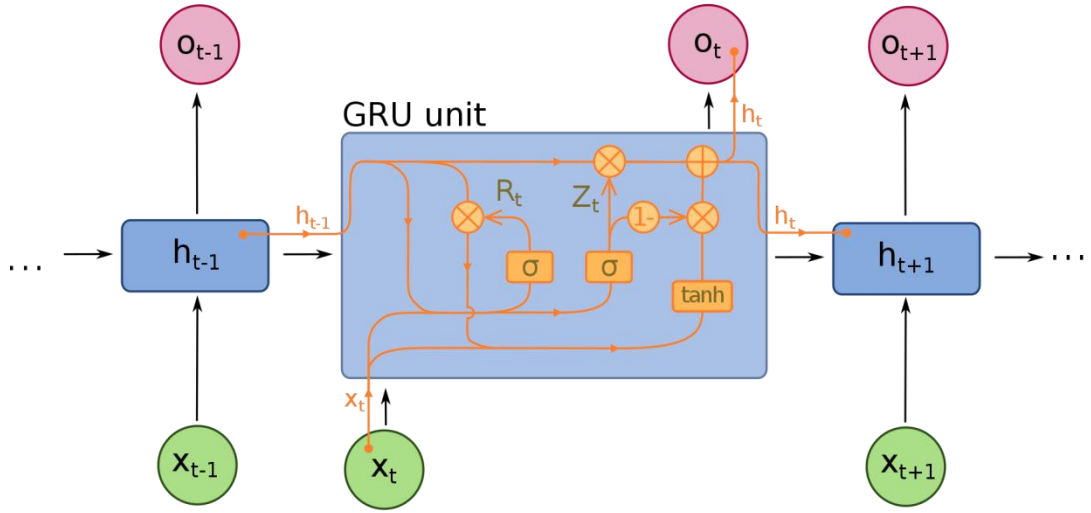
- New memory $\tilde{h}_t = tanh(r_t \circ Uh_{t-1} + Wx_t)$
- Final memory $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

- Combines current & previous time steps

8.4.3 Recurrent Neural Network

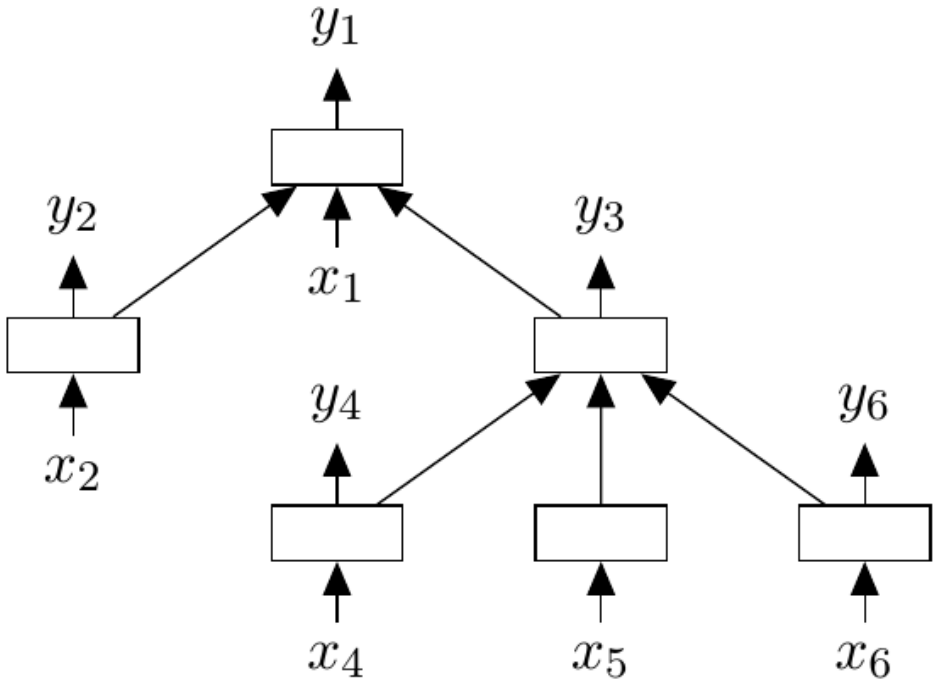






8.4.4 Recursive Neural Network

- A **RvNN** is a kind of neural network created by applying the same set of weights recursively over a structured input, to produce a structured prediction over variable-size input structures, or a scalar prediction on it, by traversing a given structure in topological order.
- RvNNs have been successful, for instance, in learning sequence and tree structures in natural language processing, mainly phrase and sentence continuous representations based on word embedding.



8.5 Building and Tuning Deep Networks

8.5.1 Modeling CSV data with multilayer perceptron networks
8.5.2 Setting up input data
8.5.3 Determining network architecture
8.5.4 Model training
8.5.5 Model evaluation
8.5.6 Concepts of tuning deep networks
8.5.7 Relating model goal and output layer
    8.5.7.1 Regression model output layer
    8.5.7.2 Classification model output layer
8.5.8 Working with layer count, parameter count and memory count

Dataset - Physicochemical Properties of Protein Tertiary Structure Data Set

Downloaded - UCL ML Repository (https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure)

Source: Prashant Singh Rana, psrana '@' gmail.com, ABV - Indian Institute of Information Technology & Management, Gwalior, MP, India.

Int. Jour. of App. Sc. and Eng. (IJASE) 3(1) : June 2015, 1-11

DOI NO. 10.5958/2322-0465.2015.00002.7

Protein Tertiary Structure Classification based on its Physicochemical property using Neural Network and KPCA-SVM: A Comparative Study

Ashwini M. Jani[1] and Kalpit R. Chandpa[2]

[1]Department of Computer Science Engineering and Information Technology, SVM Institute of Technology, Bharuch, India
[2]Department of Computer Science Engineering and Information Technology, SVM Institute of Technology, Bharuch, India

*Corresponding author: ashwini_jani@yahoo.com

ABSTRACT

Proteins are one of the most important molecules in living organisms so they play a vital structural role in the cells of living organism. They are constructed of several polypeptide chains of amino acids, which fold into complex tertiary Structure. The knowledge of the protein function is directly dependent on its three dimensional (tertiary) structure. The Physicochemical properties of proteins always guide to determine the quality of the protein tertiary structure. Therefore it has been rigorously used to distinguish native or native like structure from other predicted structure. The experiments were conducted on the CASP dataset to classify RMSD (target class) near to native protein tertiary structure or not. Kernel principal component analysis (KPCA) is used for feature extraction since it performs better than PCA on protein tertiary structure dataset due to their nonlinear structures. The proposed model compare with neural network classification method. The experiments conducted shows that support vector machine combined with KPCA feature extraction performs better than neural network classifier. More than our results show better performance in Gaussian KPCA feature extraction with respect to other kernels.

Keywords: Protein Tertiary Structure, Physicochemical Property, Data Mining, Classification, Kernel-PCA, CASP dataset, Support Vector Machine, Neural network.

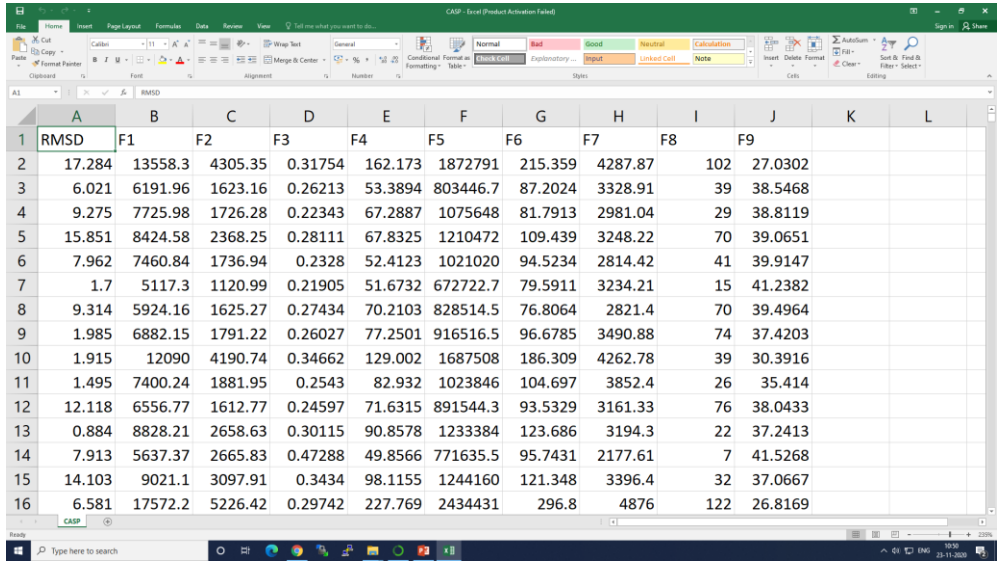
8.5.1 Modeling CSV data with Multilayer Perceptron Networks

Data Set Characteristics:	Multivariate	Number of Instances:	45730	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	9	Date Donated	2013-03-31
Associated Tasks:	Regression	Missing Values?	N/A	Number of Web Hits:	57788

The data set is taken from CASP 5-9. There are 45730 decoys and size varying from 0 to 21 armstrong.

Attribute Information:

RMSD-Size of the residue
F1 - Total surface area
F2 - Non polar exposed area
F3 - Fractional area of exposed non polar residue
F4 - Fractional area of exposed non polar part of residue
F5 - Molecular mass weighted exposed area
F6 - Average deviation from standard exposed area of residue
F7 - Euclidian distance
F8 - Secondary structure penalty
F9 - Spacial Distribution constraints (N,K Value)



	A	B	C	D	E	F	G	H	I	J	K	L
1	RMSD	F1	F2	F3	F4	F5	F6	F7	F8	F9		
2	17.284	13558.3	4305.35	0.31754	162.173	1872791	215.359	4287.87	102	27.0302		
3	6.021	6191.96	1623.16	0.26213	53.3894	803446.7	87.2024	3328.91	39	38.5468		
4	9.275	7725.98	1726.28	0.22343	67.2887	1075648	81.7913	2981.04	29	38.8119		
5	15.851	8424.58	2368.25	0.28111	67.8325	1210472	109.439	3248.22	70	39.0651		
6	7.962	7460.84	1736.94	0.2328	52.4123	1021020	94.5234	2814.42	41	39.9147		
7	1.7	5117.3	1120.99	0.21905	51.6732	672722.7	79.5911	3234.21	15	41.2382		
8	9.314	5924.16	1625.27	0.27434	70.2103	828514.5	76.8064	2821.4	70	39.4964		
9	1.985	6882.15	1791.22	0.26027	77.2501	916516.5	96.6785	3490.88	74	37.4203		
10	1.915	12090	4190.74	0.34662	129.002	1687508	186.309	4262.78	39	30.3916		
11	1.495	7400.24	1881.95	0.2543	82.932	1023846	104.697	3852.4	26	35.414		
12	12.118	6556.77	1612.77	0.24597	71.6315	891544.3	93.5329	3161.33	76	38.0433		
13	0.884	8828.21	2658.63	0.30115	90.8578	1233384	123.686	3194.3	22	37.2413		
14	7.913	5637.37	2665.83	0.47288	49.8566	771635.5	95.7431	2177.61	7	41.5268		
15	14.103	9021.1	3097.91	0.3434	98.1155	1244160	121.348	3396.4	32	37.0667		
16	6.581	17572.2	5226.42	0.29742	227.769	2434431	296.8	4876	122	26.8169		

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Importing libraries
- Loading Dataset

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline

import tensorflow as tf
```

In [2]:
```
#Importing the tensorflow libraries and packages
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout

#Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

In [3]:
```
dataset = pd.read_csv('C:/Users/NEIST/Downloads/RANJAN/Regression/Physicochemical Properties of Protein Tertiary Structur
```

In [4]:
```
dataset
```

Out[4]:

	RMSD	F1	F2	F3	F4	F5	F6	F7	F8	F9
0	17.284	13558.30	4305.35	0.31754	162.1730	1.872791e+06	215.3590	4287.87	102	27.0302
1	6.021	6191.96	1623.16	0.26213	53.3894	8.034467e+05	87.2024	3328.91	39	38.5468
2	9.275	7725.98	1726.28	0.22343	67.2887	1.075648e+06	81.7913	2981.04	29	38.8119
3	15.851	8424.58	2368.25	0.28111	67.8325	1.210472e+06	109.4390	3248.22	70	39.0651
4	7.962	7460.84	1736.94	0.23280	52.4123	1.021020e+06	94.5234	2814.42	41	39.9147
...	...	...	...	...	...	...	...	...	...	...
45725	3.762	8037.12	2777.68	0.34560	64.3390	1.105797e+06	112.7460	3384.21	84	36.8036
45726	6.521	7978.76	2508.57	0.31440	75.8654	1.116725e+06	102.2770	3974.52	54	36.0470
45727	10.356	7726.65	2489.58	0.32220	70.9903	1.076560e+06	103.6780	3290.46	46	37.4718
45728	9.791	8878.93	3055.78	0.34416	94.0314	1.242266e+06	115.1950	3421.79	41	35.6045
45729	18.827	12732.40	4444.36	0.34905	157.6300	1.788897e+06	229.4590	4626.85	141	29.8118

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Splitting Target variable

```
In [9]:
1 X = dataset.iloc[:, 1:10].values
2 y = dataset.iloc[:, 0].values

In [10]:
1 X

Out[10]: array([[1.35583e+04, 4.30535e+03, 3.17540e-01, ..., 4.28787e+03,
                 1.02000e+02, 2.70302e+01],
                [6.19196e+03, 1.62316e+03, 2.62130e-01, ..., 3.32891e+03,
                 3.90000e+01, 3.85468e+01],
                [7.72598e+03, 1.72628e+03, 2.23430e-01, ..., 2.98104e+03,
                 2.90000e+01, 3.88119e+01],
                ...,
                [7.72665e+03, 2.48958e+03, 3.22200e-01, ..., 3.29046e+03,
                 4.60000e+01, 3.74718e+01],
                [8.87893e+03, 3.05578e+03, 3.44160e-01, ..., 3.42179e+03,
                 4.10000e+01, 3.56045e+01],
                [1.27324e+04, 4.44436e+03, 3.49050e-01, ..., 4.62685e+03,
                 1.41000e+02, 2.98118e+01]])

In [11]:
1 y

Out[11]: array([17.284,  6.021,  9.275, ..., 10.356,  9.791, 18.827])
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Train and test set

```
In [12]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
In [13]:
X_train
```

```
Out[13]: array([[1.31050e+04, 6.05293e+03, 4.61870e-01, ..., 5.03564e+03,
                 4.20000e+01, 3.04798e+01],
                [6.41680e+03, 1.40747e+03, 2.19340e-01, ..., 3.00084e+03,
                 2.40000e+01, 3.84069e+01],
                [2.11188e+04, 6.64604e+03, 3.14690e-01, ..., 5.61597e+03,
                 6.90000e+01, 2.31530e+01],
                ...,
                [5.06648e+03, 1.66842e+03, 3.29300e-01, ..., 3.12092e+03,
                 4.50000e+01, 4.22870e+01],
                [8.16451e+03, 2.75723e+03, 3.37700e-01, ..., 3.80180e+03,
                 1.70000e+01, 3.76550e+01],
                [9.89490e+03, 4.22885e+03, 4.27370e-01, ..., 4.71361e+03,
                 3.10000e+01, 3.52404e+01]])
```

```
In [14]:
X_test
```

```
Out[14]: array([[7.61175e+03, 1.98350e+03, 2.60580e-01, ..., 3.57296e+03,
                 2.60000e+01, 3.75904e+01],
                [1.37913e+04, 4.53975e+03, 3.29170e-01, ..., 4.51513e+03,
                 1.31000e+02, 3.03820e+01],
                [3.81247e+03, 1.22802e+03, 3.22100e-01, ..., 1.39021e+03,
                 3.00000e+01, 4.44937e+01],
                ...,
                [1.90845e+04, 6.08779e+03, 3.18990e-01, ..., 5.77169e+03,
                 3.12000e+02, 2.11112e+01],
                [5.67580e+03, 1.97391e+03, 3.47770e-01, ..., 2.54630e+03,
                 2.40000e+01, 4.20062e+01],
                [9.27102e+03, 3.24221e+03, 3.49710e-01, ..., 3.76763e+03,
                 9.40000e+01, 3.53650e+01]])
```

```
In [15]:
y_train
```

```
Out[15]: array([18.376,  5.177,  3.613, ..., 14.866, 14.411, 19.538])
```

```
In [16]:
y_test
```

```
Out[16]: array([ 1.492, 20.056, 10.915, ...,  2.444,  2.643, 13.153])
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Feature Scaling

```
In [20]:
#Standard Scaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
In [21]:
X_train
```

```
Out[21]: array([[ 0.79459471,  2.06707967,  2.53431225, ...,  0.51618252,
                 -0.49513162, -0.67557006],
                [-0.84987578, -1.09670657, -1.31906408, ..., -0.48711339,
                 -0.81427563,  0.64806761],
                [ 2.76499894,  2.47101669,  0.19588019, ...,  0.802325  ,
                 -0.01641559, -1.89897187],
                ...,
                [-1.18188759, -0.91898681,  0.42800746, ..., -0.42790572,
                 -0.44194095,  1.29595227],
                [-0.42015515, -0.17745377,  0.56146873, ..., -0.0921852 ,
                 -0.93838719,  0.52251814],
                [ 0.0053069 ,  0.82479167,  1.98616776, ...,  0.35739965,
                 -0.69016407,  0.11933722]])
```

```
In [22]:
X_test
```

```
Out[22]: array([[-0.55606578, -0.70440187, -0.66383281, ..., -0.20501901,
                 -0.77881519,  0.51173147],
                [ 0.96333968,  1.03652983,  0.42594199, ...,  0.2595354 ,
                  1.08285825, -0.69190034],
                [-1.49021904, -1.21892083,  0.31361209, ..., -1.28126442,
                 -0.70789429,  1.66441883],
                ...,
                [ 2.26481259,  2.09082104,  0.26419964, ...,  0.87910564,
                  4.29202864, -2.23990404],
                [-1.03207019, -0.71093313,  0.72146337, ..., -0.71123278,
                 -0.81427563,  1.24906533],
                [-0.14809046,  0.15284141,  0.75228657, ..., -0.10903335,
                  0.42683999,  0.14014246]])
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- FFNN initialized
- Compiled and trained

In [98]:
```
ann = Sequential()

ann.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 9))

ann.add(Dense(units = 3, kernel_initializer = 'uniform', activation = 'relu'))

ann.add(Dense(units = 1, kernel_initializer = 'uniform'))
```

In [99]:
```
ann.compile(optimizer='rmsprop',loss='mse', metrics=['mae'])
```

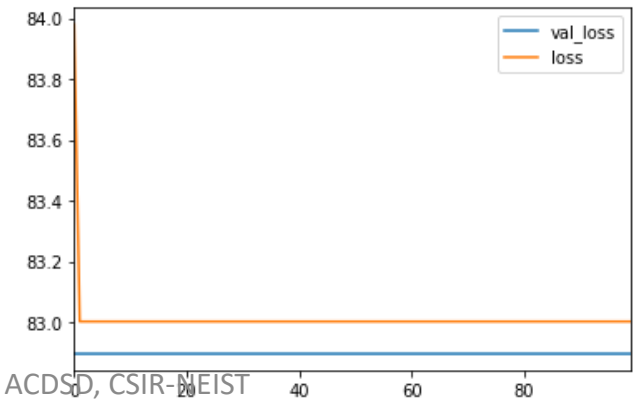
In [100]:
```
ann.fit(x=X_train, y=y_train, batch_size = 32, epochs = 10, validation_data=(X_test, y_test), verbose=1)
```

```
Train on 34297 samples, validate on 11433 samples
Epoch 1/10
34297/34297 [==============================] - 2s 58us/step - loss: 54.9469 - mae: 5.6298 - val_loss: 32.7965 - val_mae: 4.
6798
Epoch 2/10
34297/34297 [==============================] - 2s 45us/step - loss: 29.5085 - mae: 4.4936 - val_loss: 27.8537 - val_mae: 4.
4431
Epoch 3/10
34297/34297 [==============================] - 2s 45us/step - loss: 27.1373 - mae: 4.3574 - val_loss: 26.8822 - val_mae: 4.
3492
Epoch 4/10
34297/34297 [==============================] - 2s 46us/step - loss: 26.4673 - mae: 4.2655 - val_loss: 26.5485 - val_mae: 4.
2775
Epoch 5/10
34297/34297 [==============================] - 2s 44us/step - loss: 26.2442 - mae: 4.2161 - val_loss: 26.3390 - val_mae: 4.
2243
Epoch 6/10
                                                                                                    l_loss: 26.2962 - val_mae: 4.
                                                                                                    l_loss: 26.2951 - val_mae: 4.
                                                                                                    l_loss: 26.2212 - val_mae: 4.
                                                                                                    l_loss: 26.1840 - val_mae: 4.
                                                                                                    l_loss: 26.1949 - val_mae: 4.
```

In [29]:
```
model_loss.plot()
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x1df24574308>

Out[100]:



In [26]:
```
#Model summary
ann.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 6)                 60
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 7
=================================================================
Total params: 109
Trainable params: 109
Non-trainable params: 0
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Prediction

```
In [71]:
1 y_test = pd.DataFrame(y_test)
2 y_test
```

Out[71]:

	0
0	1.492
1	20.056
2	10.915
3	15.127
4	10.380
...	...
11428	16.706
11429	11.663
11430	2.444
11431	2.643
11432	13.153

11433 rows × 1 columns

```
In [105]:
1 y_pred = ann.predict(X_test)
2 y_pred
```

Out[105]:

```
array([[6.8994055],
       [8.871464 ],
       [7.9135284],
       ...,
       [5.3081393],
       [9.081087 ],
       [9.929531 ]], dtype=float32)
```

```
In [110]:
1 y_pred = pd.DataFrame(y_pred)
2 y_pred
```

Out[110]:

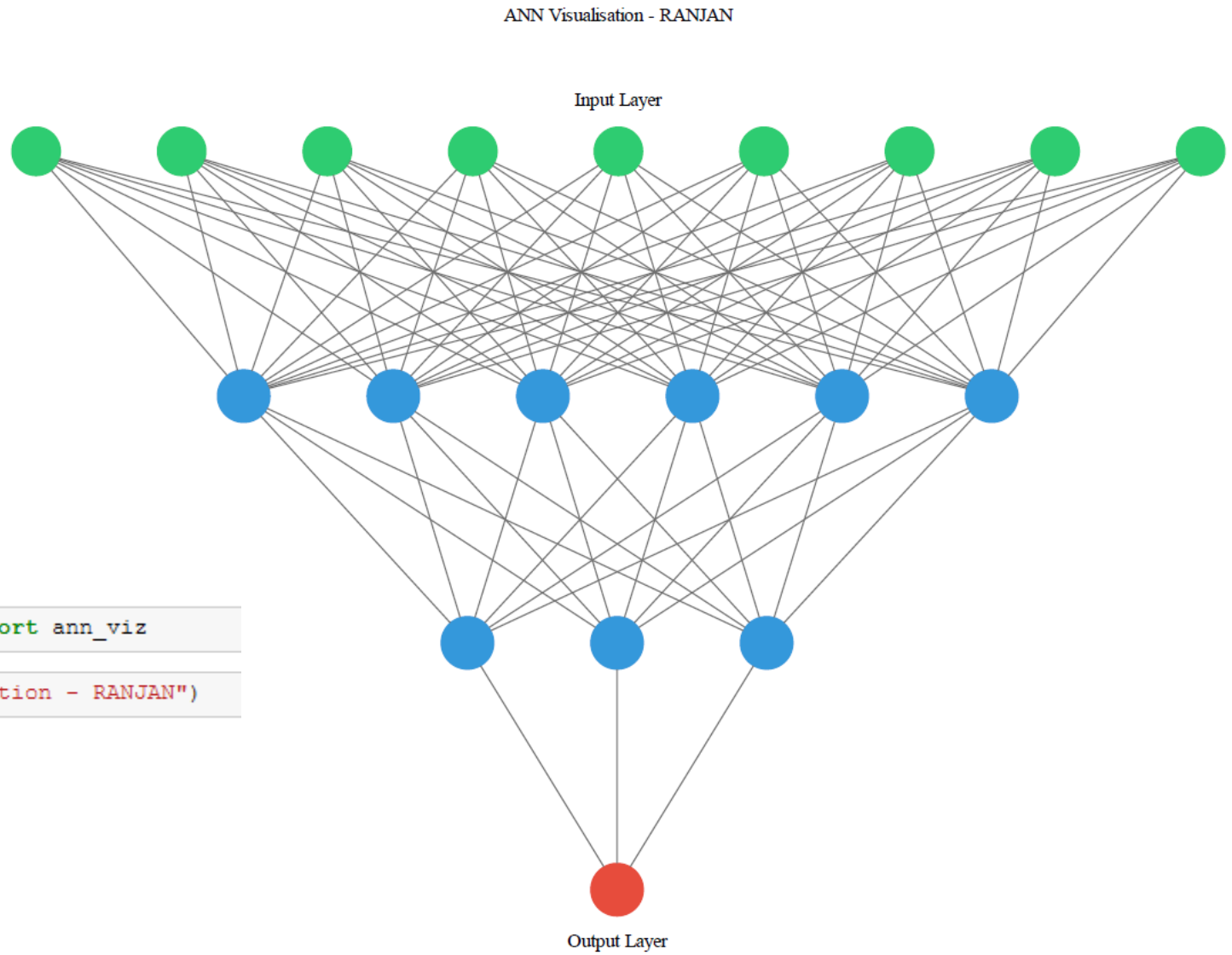
	0
0	6.899405
1	8.871464
2	7.913528
3	11.506614
4	7.451507
...	...
11428	5.736327
11429	7.877826
11430	5.308139
11431	9.081087
11432	9.929531

11433 rows × 1 columns

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Model Developed

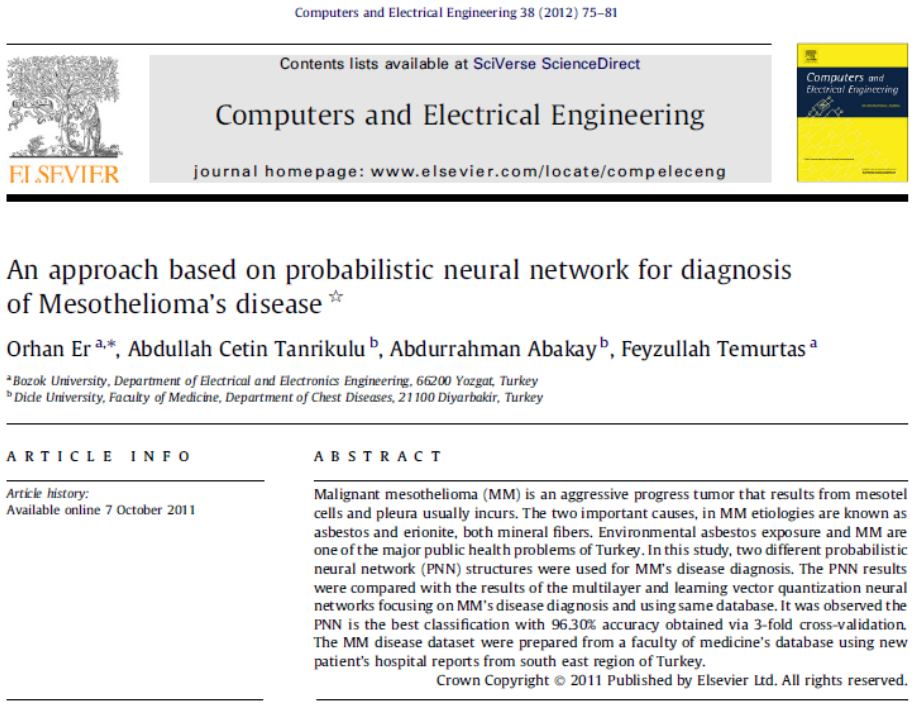
```
In [31]: 1 from ann_visualizer.visualize import ann_viz

In [33]: 1 ann_viz(ann, title="ANN Visualisation - RANJAN")
```



Dataset - Mesothelioma disease data set Data Set

Downloaded - UCL ML Repository (https://archive.ics.uci.edu/ml/datasets/Mesothelioma%C3%A2%E2%82%AC%E2%84%A2s+disease+data+set+)

Source: This data was prepared by;
Abdullah Cetin Tanrikulu from Dicle University, Faculty of Medicine, Department of Chest Diseases, 21100 Diyarbakir, Turkey
e-mail:cetintanrikulu '@' hotmail.com
Orhan Er from Bozok University, Faculty of Engineering, Department of Electrical and Electronics Eng., 66200 Yozgat, Turkey
e-mail:orhan.er@bozok.edu.tr



Computers and Electrical Engineering 38 (2012) 75–81

Contents lists available at SciVerse ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

An approach based on probabilistic neural network for diagnosis of Mesothelioma's disease ☆

Orhan Er [a,*], Abdullah Cetin Tanrikulu [b], Abdurrahman Abakay [b], Feyzullah Temurtas [a]

[a] Bozok University, Department of Electrical and Electronics Engineering, 66200 Yozgat, Turkey
[b] Dicle University, Faculty of Medicine, Department of Chest Diseases, 21100 Diyarbakir, Turkey

ARTICLE INFO

Article history:
Available online 7 October 2011

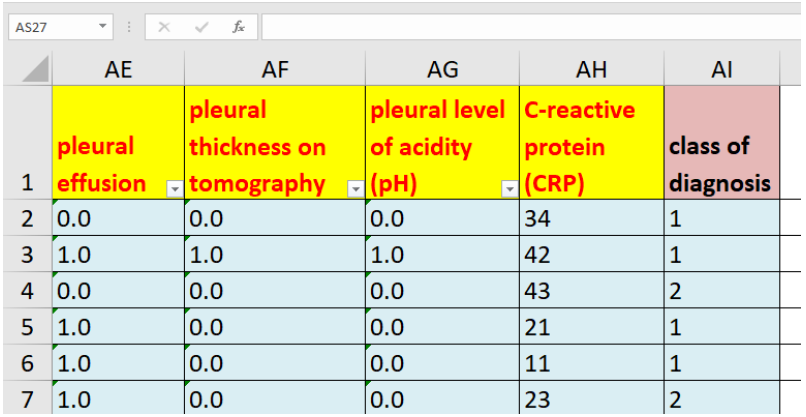
ABSTRACT

Malignant mesothelioma (MM) is an aggressive progress tumor that results from mesotel cells and pleura usually incurs. The two important causes, in MM etiologies are known as asbestos and erionite, both mineral fibers. Environmental asbestos exposure and MM are one of the major public health problems of Turkey. In this study, two different probabilistic neural network (PNN) structures were used for MM's disease diagnosis. The PNN results were compared with the results of the multilayer and learning vector quantization neural networks focusing on MM's disease diagnosis and using same database. It was observed the PNN is the best classification with 96.30% accuracy obtained via 3-fold cross-validation. The MM disease dataset were prepared from a faculty of medicine's database using new patient's hospital reports from south east region of Turkey.

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

Data Set Characteristics:	Multivariate	**Number of Instances:**	324	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	34	**Date Donated**	2016-01-11
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	16318

Malignant mesotheliomas (MM) are very aggressive tumors of the pleura. These tumors are connected to asbestos exposure, however it may also be related to previous simian virus 40 (SV40) infection and quite possible for genetic predisposition. Molecular mechanisms can also be implicated in the development of mesothelioma.



	AE	AF	AG	AH	AI
1	pleural effusion	pleural thickness on tomography	pleural level of acidity (pH)	C-reactive protein (CRP)	class of diagnosis
2	0.0	0.0	0.0	34	1
3	1.0	1.0	1.0	42	1
4	0.0	0.0	0.0	43	2
5	1.0	0.0	0.0	21	1
6	1.0	0.0	0.0	11	1
7	1.0	0.0	0.0	23	2

The features are; age, gender, city, asbestos exposure, type of MM, duration of asbestos exposure, diagnosis method, keep side, cytology, duration of symptoms, dyspnoea, ache on chest, weakness, habit of cigarette, performance status, White Blood cell count (WBC), hemoglobin (HGB), platelet count (PLT), sedimentation, blood lactic dehydrogenise (LDH), Alkaline phosphatise(ALP), total protein, albumin, glucose, pleural lactic dehydrogenise, pleural protein, pleural albumin, pleural glucose,dead or not, pleural effusion, pleural thickness on tomography, pleural level of acidity (pH), C-reactive protein (CRP), class of diagnosis.

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Load Dataset
- Classifying target variable

```
In [20]: 1 dataset
```

Out[20]:

type of MM	duration of asbestos exposure	diagnosis method	keep side	cytology	duration of symptoms	...	pleural lactic dehydrogenise	pleural protein	pleural albumin	pleural glucose	dead or not	pleural effusion	pleural thickness on tomography	pleural level of acidity (pH)	C-reactive protein (CRP)	class of diagnosis
0.0	20.0	1	0	1	24.0	...	289.0	0.0	0.00	79.0	1	0.0	0.0	0.0	34	1
0.0	45.0	1	0	0	1.0	...	7541.0	1.6	0.80	6.0	1	1.0	1.0	1.0	42	1
0.0	23.0	0	1	0	1.0	...	480.0	0.0	0.00	90.0	1	0.0	0.0	0.0	43	2
0.0	10.0	1	0	0	3.0	...	459.0	5.0	2.80	45.0	1	1.0	0.0	0.0	21	1
0.0	10.0	1	1	1	1.5	...	213.0	3.6	1.95	53.0	1	1.0	0.0	0.0	11	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.0	50.0	1	1	0	9.0	...	323.0	4.9	2.60	23.0	1	1.0	1.0	0.0	76	1
0.0	41.0	1	1	0	9.0	...	323.0	4.9	2.60	23.0	1	1.0	1.0	0.0	67	1
0.0	40.0	1	0	0	8.0	...	300.0	5.1	2.20	35.0	1	1.0	0.0	1.0	68	1
0.0	0.0	0	1	0	2.0	...	3000.0	2.4	1.20	2.0	1	1.0	1.0	0.0	78	2
0.0	40.0	1	1	0	3.0	...	2100.0	5.7	3.30	35.0	1	1.0	1.0	0.0	45	1

```
In [21]: 1 X = dataset.iloc[:, :-1].values
         2 y = dataset.iloc[:, -1].values

In [22]: 1 X
```

```
Out[22]: array([[47.,  1.,  0., ...,  0.,  0., 34.],
                [55.,  1.,  0., ...,  1.,  1., 42.],
                [29.,  1.,  1., ...,  0.,  0., 43.],
                ...,
                [58.,  1.,  6., ...,  0.,  1., 68.],
                [42.,  1.,  6., ...,  1.,  0., 78.],
                [54.,  1.,  0., ...,  1.,  0., 45.]])

In [23]: 1 y
```

```
Out[23]: array([1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
                2, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1,
                2, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Train and test data

In [24]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

print(X_train)
```

```
[[63.  0.  0. ...  0.  0. 89.]
 [65.  0.  4. ...  1.  0. 78.]
 [75.  1.  1. ...  1.  1. 44.]
 ...
 [52.  1.  1. ...  1.  0. 68.]
 [70.  0.  6. ...  1.  1. 79.]
 [44.  0.  4. ...  0.  0. 77.]]
```

In [25]:
```
print(X_test)
```

```
[[82.  1.  5. ...  1.  0. 79.]
 [25.  1.  6. ...  0.  0. 23.]
 [48.  0.  1. ...  0.  0. 34.]
 ...
 [66.  1.  2. ...  0.  0. 41.]
 [55.  0.  1. ...  1.  1. 67.]
 [47.  1.  0. ...  0.  0. 36.]]
```

In [26]:
```
print(y_train)
```

```
[1 2 1 2 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 2
 2 1 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 2 2 1 1 1 2 1 1 2 1 1 2 2 1 1 1 2 1 1
 1 1 1 1 1 1 2 2 1 2 1 2 1 2 2 1 1 1 2 1 1 2 1 1 1 2 2 2 1 2 1 1 1 1 2 2 1
 1 1 1 1 2 2 1 1 1 2 1 1 2 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1
 1 2 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1 1 1 2 2 2 1 2 2 1 2 1 1 1
 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 1 2 2 1 2 1 1 1 1 1 2 1 2 1 1 1 1 1 1
 1 1 1 2 1 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2]
```

In [27]:
```
print(y_test)
```

```
[1 1 1 1 1 2 1 2 1 2 1 2 1 1 1 2 2 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 2
 1 1 2 1 1 2 1 1 1 2 2 1 1 1 2 1 1 2 1 2 1 1 2 1 1 1 1 1 2 2 1 1 2 1 1 1 2
 1 1 2 1 2 1 1]
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Balancing Dataset

In [82]:
```
#Check balanced or not
dataset['class of diagnosis'].value_counts()
```

Out[82]:
```
1    228
2     96
Name: class of diagnosis, dtype: int64
```

In [83]:
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '2': {} \n".format(sum(y_train == 2)))
```

```
Before OverSampling, counts of label '1': 174
Before OverSampling, counts of label '2': 69
```

In [85]:
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=3)
X_train, y_train = sm.fit_resample(X_train, y_train)
pd.Series(y_train).value_counts()
```

Out[85]:
```
2    174
1    174
dtype: int64
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- ANN model

In [56]:
```
ann = Sequential()

ann.add(Dense(units = 18, kernel_initializer = 'uniform', activation = 'sigmoid', input_dim = 34))

ann.add(Dense(units = 9, kernel_initializer = 'uniform', activation = 'sigmoid'))

ann.add(Dense(units = 4, kernel_initializer = 'uniform', activation = 'sigmoid'))

ann.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'softmax'))
```

In [57]:
```
ann.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

ann.fit(x=X_train, y=y_train, batch_size = 32, epochs = 100, validation_data=(X_test, y_test), verbose=1)

ann.summary()
```

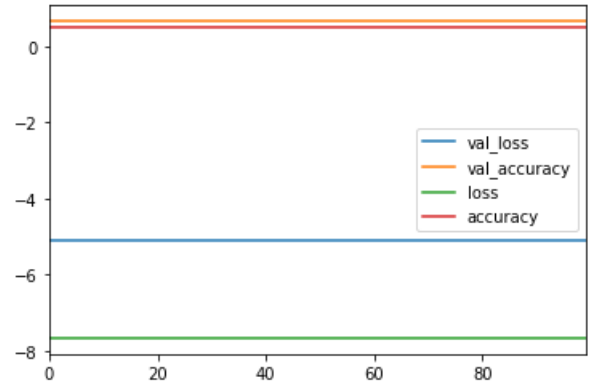
```
Epoch 100/100
348/348 [==============================] - 0s 75us/step - loss: -7.6666 - accuracy: 0.5000 - val_loss: -5.1111 - val_accu
racy: 0.6667
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_19 (Dense)             (None, 18)                630
_________________________________________________________________
dense_20 (Dense)             (None, 9)                 171
_________________________________________________________________
dense_21 (Dense)             (None, 4)                 40
_________________________________________________________________
dense_22 (Dense)             (None, 1)                 5
=================================================================
Total params: 846
Trainable params: 846
Non-trainable params: 0
_________________________________________________________________
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- ANN model summary

```
In [58]:
1 model_loss = pd.DataFrame(ann.history.history)
2 model_loss.plot()
```

```
Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x268efe9c148>
```




```
In [59]:
1 y_pred = ann.predict(X_test)
2 y_pred = (y_pred > 0.5)
3 y_pred
```

```
Out[59]: array([[ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
                [ True],
```

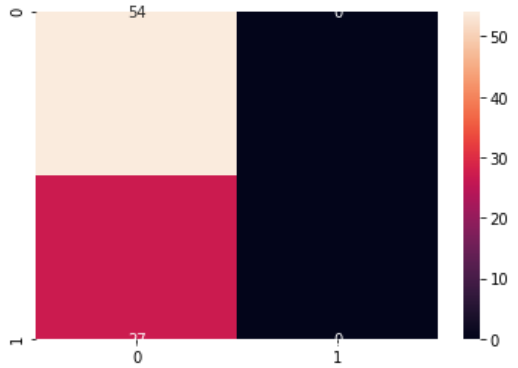
8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Evaluation

In [61]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print("Our accuracy is {}%".format((float(cm[0][0] + cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1]))*100))
```

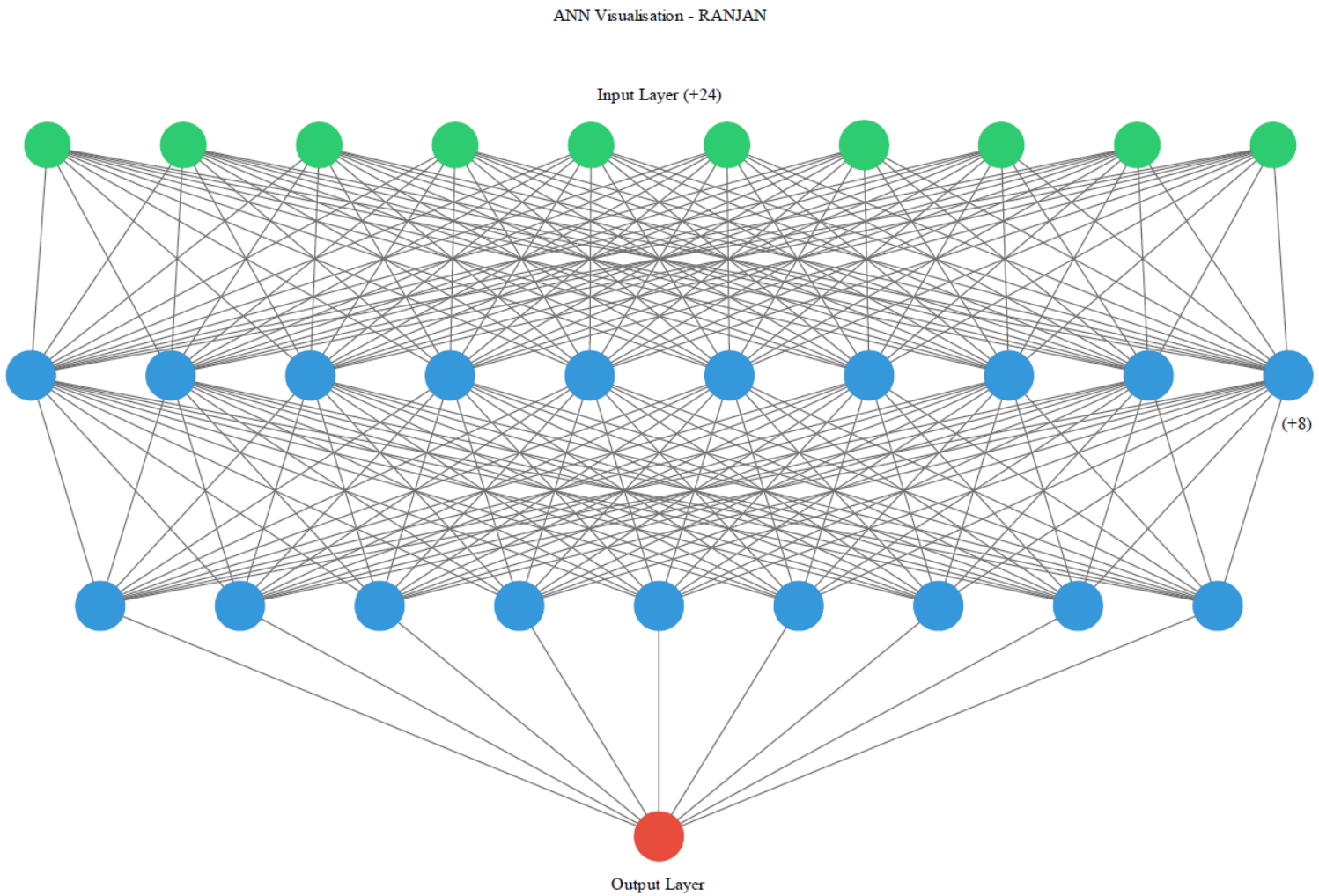
```
[[54  0]
 [27  0]]
0.6666666666666666
              precision    recall  f1-score   support

           1       0.67      1.00      0.80        54
           2       0.00      0.00      0.00        27

    accuracy                           0.67        81
   macro avg       0.33      0.50      0.40        81
weighted avg       0.44      0.67      0.53        81

Our accuracy is 66.66666666666666%
```

In [62]:
```
#Heatmap
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plt.savefig('h.png')
```



8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Model Developed




```
In [31]: 1 from ann_visualizer.visualize import ann_viz

In [33]: 1 ann_viz(ann, title="ANN Visualisation - RANJAN")
```

Dataset – Pneumonia Detection Data Set

Downloaded – Kaggle ML Repository (https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia)

Hindawi
Journal of Healthcare Engineering
Volume 2019, Article ID 4180949, 7 pages
https://doi.org/10.1155/2019/4180949

Hindawi

Research Article

An Efficient Deep Learning Approach to Pneumonia Classification in Healthcare

Okeke Stephen,[1] **Mangal Sain**,[2] **Uchenna Joseph Maduh**,[3] **and Do-Un Jeong**[2]

[1]*Department of Computer Engineering, Dongseo University, Busan, Republic of Korea*
[2]*Division of Computer Engineering, Dongseo University, Busan, Republic of Korea*
[3]*Department of Civil Engineering, Yeungnam University, Gyeongsan, Republic of Korea*

Correspondence should be addressed to Mangal Sain; mangalsain1@gmail.com and Do-Un Jeong; dujeong@dongseo.ac.kr

Received 19 December 2018; Accepted 24 February 2019; Published 27 March 2019

Guest Editor: Ahyoung Choi

This study proposes a convolutional neural network model trained from scratch to classify and detect the presence of pneumonia from a collection of chest X-ray image samples. Unlike other methods that rely solely on transfer learning approaches or traditional handcrafted techniques to achieve a remarkable classification performance, we constructed a convolutional neural network model from scratch to extract features from a given chest X-ray image and classify it to determine if a person is infected with pneumonia. This model could help mitigate the reliability and interpretability challenges often faced when dealing with medical imagery. Unlike other deep learning classification tasks with sufficient image repository, it is difficult to obtain a large amount of pneumonia dataset for this classification task; therefore, we deployed several data augmentation algorithms to improve the validation and classification accuracy of the CNN model and achieved remarkable validation accuracy.

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

Dataset arrangement

In [26]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import scipy as sc
%matplotlib inline

import tensorflow as tf
```

In [27]:
```
#Setting directory
training_path = 'D:/Ranjan_Dataset/chest_xray/train'
```

In [28]:
```
#Setting directory
test_path = 'D:/Ranjan_Dataset/chest_xray/test'
```

In [35]:
```
train_datagen = ImageDataGenerator( rotation_range=20,
                                    width_shift_range=0.10,
                                    height_shift_range=0.10,
                                    rescale = 1./255,
                                    shear_range = 0.2,
                                    zoom_range = 0.2,
                                    horizontal_flip = True
                                    )

```

In [37]:
```
training_set = train_datagen.flow_from_directory(training_path,
                                                 target_size = (64, 64),
                                                 #color_mode='rgb'
                                                 batch_size = 32,
                                                 class_mode = 'binary')


```

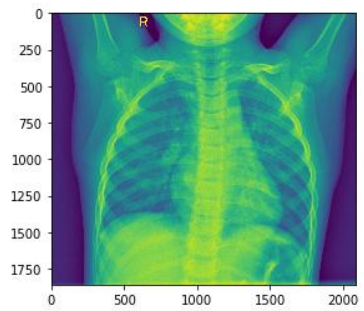
```
Found 5216 images belonging to 2 classes.
```

In [38]:
```
training_set.class_indices
```

Out[38]: {'NORMAL': 0, 'PNEUMONIA': 1}

In [39]:
```
from matplotlib.image import imread
single_training_image = imread('D:/Ranjan_Dataset/chest_xray/train/NORMAL/IM-0115-0001.jpeg')
plt.imshow(single_training_image)
single_training_image.shape
```

Out[39]: (1858, 2090)



8.5.1 Modeling CSV data with Multilayer Perceptron Networks

CNN model

In [48]:
```
#Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Activation, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

In [53]:
```
cnn = Sequential()
cnn.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation = 'relu', input_shape = (64, 64, 3)))
cnn.add(MaxPooling2D(pool_size = (2, 2), strides=2, padding='valid'))
cnn.add(Conv2D(filters=32, kernel_size=(3,3), padding="same", activation = 'relu'))
cnn.add(MaxPooling2D(pool_size = (2, 2), strides=2, padding='valid'))
cnn.add(Flatten())
cnn.add(Dense(units = 128, activation = 'relu'))
cnn.add(Dropout(0.5))      #randomly turn off 50% of neurons.
cnn.add(Dense(units = 1, activation = 'sigmoid'))
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
cnn.fit_generator(training_set,
        steps_per_epoch = 8000,
        epochs = 5,
        validation_data = test_set,
        validation_steps = 2000)

```

```
Epoch 1/5
8000/8000 [==============================] - 5329s 666ms/step - loss: 0.1135 - accuracy: 0.9582 - val_loss: 0.1171 - val_ac
curacy: 0.9087
Epoch 2/5
8000/8000 [==============================] - 5440s 680ms/step - loss: 0.0971 - accuracy: 0.9646 - val_loss: 1.2216 - val_ac
curacy: 0.8990
Epoch 3/5
8000/8000 [==============================] - 5409s 676ms/step - loss: 0.0885 - accuracy: 0.9676 - val_loss: 0.2101 - val_ac
curacy: 0.9103
Epoch 4/5
8000/8000 [==============================] - 5424s 678ms/step - loss: 0.0826 - accuracy: 0.9701 - val_loss: 0.0888 - val_ac
curacy: 0.9167
Epoch 5/5
8000/8000 [==============================] - 5435s 679ms/step - loss: 0.0765 - accuracy: 0.9723 - val_loss: 2.7228 - val_ac
curacy: 0.9103
```

Out[53]:
```
<keras.callbacks.callbacks.History at 0x2370cfd4e08>
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

CNN Summary

In [54]:
```
#Model summary
cnn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               1048704
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,058,977
Trainable params: 1,058,977
Non-trainable params: 0
_________________________________________________________________
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

Evaluation

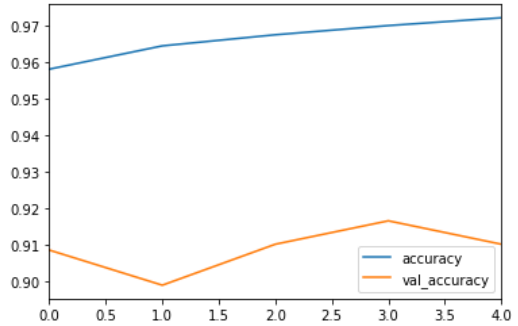
In [57]:
```
losses = pd.DataFrame(cnn.history.history)
losses.head()
```

Out[57]:

	val_loss	val_accuracy	loss	accuracy
0	0.117108	0.908654	0.113495	0.958188
1	1.221560	0.899038	0.097073	0.964562
2	0.210058	0.910256	0.088463	0.967602
3	0.088812	0.916667	0.082582	0.970117
4	2.722824	0.910256	0.076521	0.972254

In [58]:
```
losses[['accuracy','val_accuracy']].plot()          #val_accuracy means validation accuracy
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x2370d770108>



In [59]:
```
losses[['loss','val_loss']].plot()                  #val_loss means validation loss
```

Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x2370d62b648>

Evaluation

In [61]:
```python
#Numpy can treat this as True/False for us
predictions = pred_probabilities > 0.5
predictions
```

Out[61]:
```
array([[ True],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [False],
       [ True],
```

In [62]:
```python
#Classes to verify
test_set.classes
```

Out[62]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

Evaluation

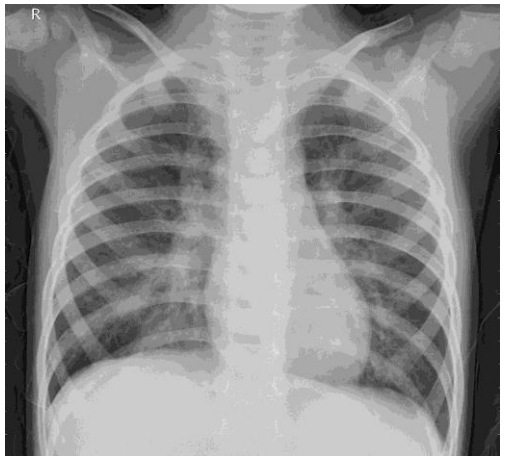
In [63]:
```
from sklearn.metrics import classification_report, confusion_matrix
confusion_matrix(test_set.classes, predictions)
print(classification_report(test_set.classes, predictions))
```

```
              precision    recall  f1-score   support

           0       0.37      0.32      0.35       234
           1       0.62      0.67      0.65       390

    accuracy                           0.54       624
   macro avg       0.50      0.50      0.50       624
weighted avg       0.53      0.54      0.53       624
```

In [64]:
```
from keras.preprocessing import image
test_image = image.load_img('D:/Ranjan_Dataset/chest_xray/A.jpeg', target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 0:
    prediction = 'Normal'
else:
    prediction = 'Pneumonia'
```

In [65]:
```
print(prediction)
```

```
Pneumonia
```



8.5.1 Modeling CSV data with Multilayer Perceptron Networks

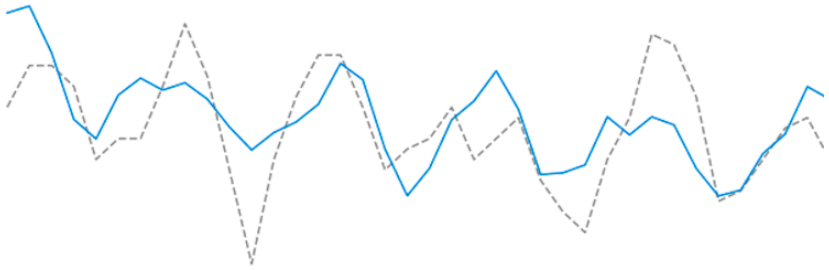
Dataset Detail: Stock exchange dataset [Time series dataset]

- Dataset Source: Kaggle ML Repository
- Domain: Management/Economics
- About Dataset: Time series data of 2018 us stock from Jan to Nov.

Objectives: Developed AI model to analyze and predict future trend of stack market without intervention of human expert.

Learning Outcome:

- Pre-processing – how to pre-process the image data
- Fit simple RNN (LSTM) to predict the result for a new image data
- Focus is on how to do it.
- Neither optimization nor constructing best model is done



Further Reading: Research Paper is not provided for now.

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Dataset

In [15]:
```
dataset_train = pd.read_csv('D:/Dataset/Google_Stock_Price_Train.csv')
training_set = dataset_train.iloc[:, 1:2].values
training_set
```

Out[15]:
```
array([[325.25],
       [331.27],
       [329.83],
       ...,
       [793.7 ],
       [783.33],
       [782.75]])
```

In [14]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set = sc.fit_transform(training_set)
training_set
```

Out[14]:
```
array([[0.08581368],
       [0.09701243],
       [0.09433366],
       ...,
       [0.95725128],
       [0.93796041],
       [0.93688146]])
```

8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Model

In [10]:

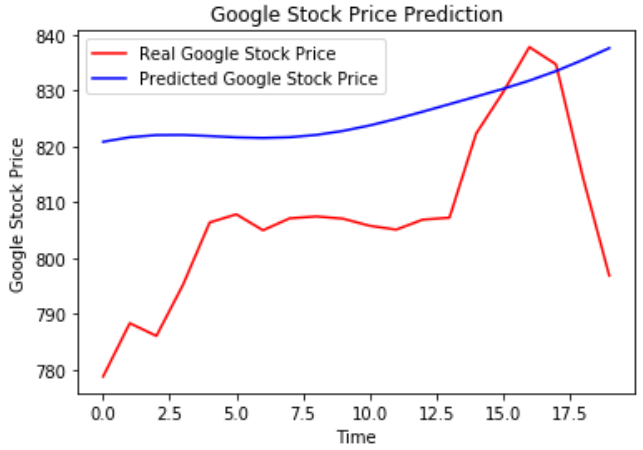
```
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.1))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.3))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

regressor.compile(optimizer = 'adam')

regressor.fit(X_train, y_train, epochs = 20, batch_size = 32)
```

```
Epoch 1/20
1198/1198 [==============================] - 11s 9ms/step - loss: 0.0439
Epoch 2/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0062
Epoch 3/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0057
Epoch 4/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0057
Epoch 5/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0050
Epoch 6/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0048
Epoch 7/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0049
Epoch 8/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0048
Epoch 9/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0047
Epoch 10/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0048
Epoch 11/20
1198/1198 [==============================] - 7s 6ms/step - loss: 0.0045
Epoch 12/20
```

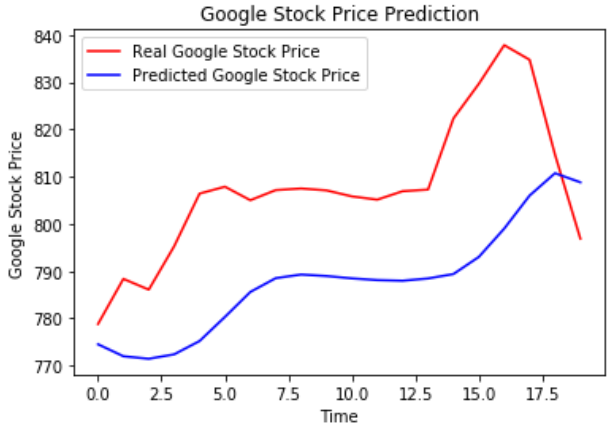
8.5.1 Modeling CSV data with Multilayer Perceptron Networks

- Result

In [12]:
```
# Visualising the results
plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```



In [4]:
```
# Visualising the results
plt.plot(real_stock_price, color = 'red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```



8.5.2 Setting up Input Data

Dataset - Physicochemical Properties of Protein Tertiary Structure Data Set

Downloaded - UCL ML Repository (https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure)

Source: Prashant Singh Rana, psrana '@' gmail.com, ABV - Indian Institute of Information Technology & Management, Gwalior, MP, India.

Int. Jour. of App. Sc. and Eng. (IJASE) 3(1) : June 2015, 1-11

DOI NO. 10.5958/2322-0465.2015.00002.7

Protein Tertiary Structure Classification based on its Physicochemical property using Neural Network and KPCA-SVM: A Comparative Study

Ashwini M. Jani[1] and Kalpit R. Chandpa[2]

[1]Department of Computer Science Engineering and Information Technology, SVM Institute of Technology, Bharuch, India
[2]Department of Computer Science Engineering and Information Technology, SVM Institute of Technology, Bharuch, India

*Corresponding author: ashwini_jani@yahoo.com

ABSTRACT

Proteins are one of the most important molecules in living organisms so they play a vital structural role in the cells of living organism. They are constructed of several polypeptide chains of amino acids, which fold into complex tertiary Structure. The knowledge of the protein function is directly dependent on its three dimensional (tertiary) structure. The Physicochemical properties of proteins always guide to determine the quality of the protein tertiary structure. Therefore it has been rigorously used to distinguish native or native like structure from other predicted structure. The experiments were conducted on the CASP dataset to classify RMSD (target class) near to native protein tertiary structure or not. Kernel principal component analysis (KPCA) is used for feature extraction since it performs better than PCA on protein tertiary structure dataset due to their nonlinear structures. The proposed model compare with neural network classification method. The experiments conducted shows that support vector machine combined with KPCA feature extraction performs better than neural network classifier. More than our results show better performance in Gaussian KPCA feature extraction with respect to other kernels.

Keywords: Protein Tertiary Structure, Physicochemical Property, Data Mining, Classification, Kernel-PCA, CASP dataset, Support Vector Machine, Neural network.

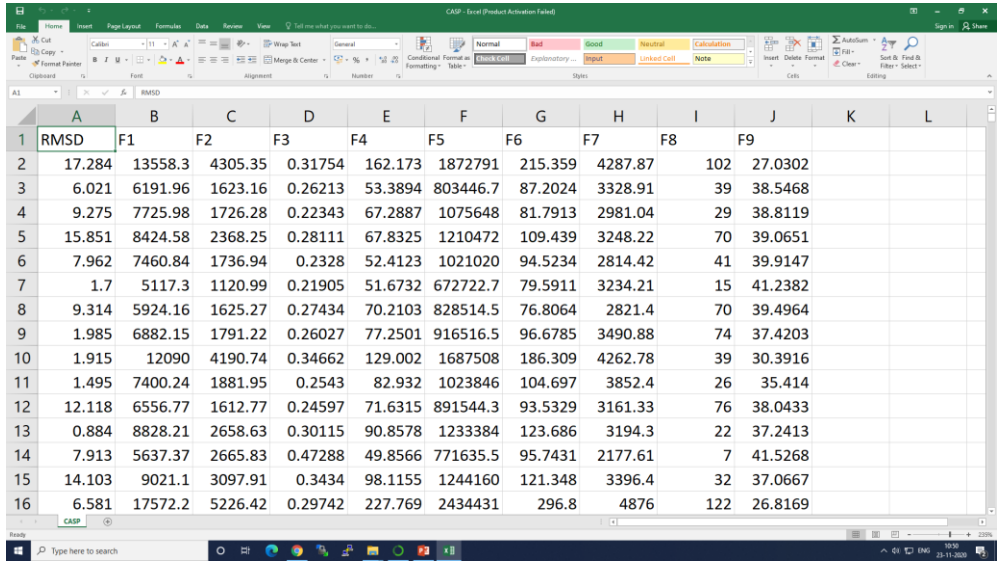
8.5.2 Setting up Input Data

Data Set Characteristics:	Multivariate	Number of Instances:	45730	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	9	Date Donated	2013-03-31
Associated Tasks:	Regression	Missing Values?	N/A	Number of Web Hits:	57788

The data set is taken from CASP 5-9. There are 45730 decoys and size varying from 0 to 21 armstrong.

Attribute Information:

RMSD-Size of the residue
F1 - Total surface area
F2 - Non polar exposed area
F3 - Fractional area of exposed non polar residue
F4 - Fractional area of exposed non polar part of residue
F5 - Molecular mass weighted exposed area
F6 - Average deviation from standard exposed area of residue
F7 - Euclidian distance
F8 - Secondary structure penalty
F9 - Spacial Distribution constraints (N,K Value)



	A	B	C	D	E	F	G	H	I	J	K	L
1	RMSD	F1	F2	F3	F4	F5	F6	F7	F8	F9		
2	17.284	13558.3	4305.35	0.31754	162.173	1872791	215.359	4287.87	102	27.0302		
3	6.021	6191.96	1623.16	0.26213	53.3894	803446.7	87.2024	3328.91	39	38.5468		
4	9.275	7725.98	1726.28	0.22343	67.2887	1075648	81.7913	2981.04	29	38.8119		
5	15.851	8424.58	2368.25	0.28111	67.8325	1210472	109.439	3248.22	70	39.0651		
6	7.962	7460.84	1736.94	0.2328	52.4123	1021020	94.5234	2814.42	41	39.9147		
7	1.7	5117.3	1120.99	0.21905	51.6732	672722.7	79.5911	3234.21	15	41.2382		
8	9.314	5924.16	1625.27	0.27434	70.2103	828514.5	76.8064	2821.4	70	39.4964		
9	1.985	6882.15	1791.22	0.26027	77.2501	916516.5	96.6785	3490.88	74	37.4203		
10	1.915	12090	4190.74	0.34662	129.002	1687508	186.309	4262.78	39	30.3916		
11	1.495	7400.24	1881.95	0.2543	82.932	1023846	104.697	3852.4	26	35.414		
12	12.118	6556.77	1612.77	0.24597	71.6315	891544.3	93.5329	3161.33	76	38.0433		
13	0.884	8828.21	2658.63	0.30115	90.8578	1233384	123.686	3194.3	22	37.2413		
14	7.913	5637.37	2665.83	0.47288	49.8566	771635.5	95.7431	2177.61	7	41.5268		
15	14.103	9021.1	3097.91	0.3434	98.1155	1244160	121.348	3396.4	32	37.0667		
16	6.581	17572.2	5226.42	0.29742	227.769	2434431	296.8	4876	122	26.8169		

8.5.2 Setting up Input Data

- Importing libraries
- Loading Dataset

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline

import tensorflow as tf
```

In [2]:
```
#Importing the tensorflow libraries and packages
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout

#Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

In [3]:
```
dataset = pd.read_csv('C:/Users/NEIST/Downloads/RANJAN/Regression/Physicochemical Properties of Protein Tertiary Structur
```

In [4]:
```
dataset
```

Out[4]:

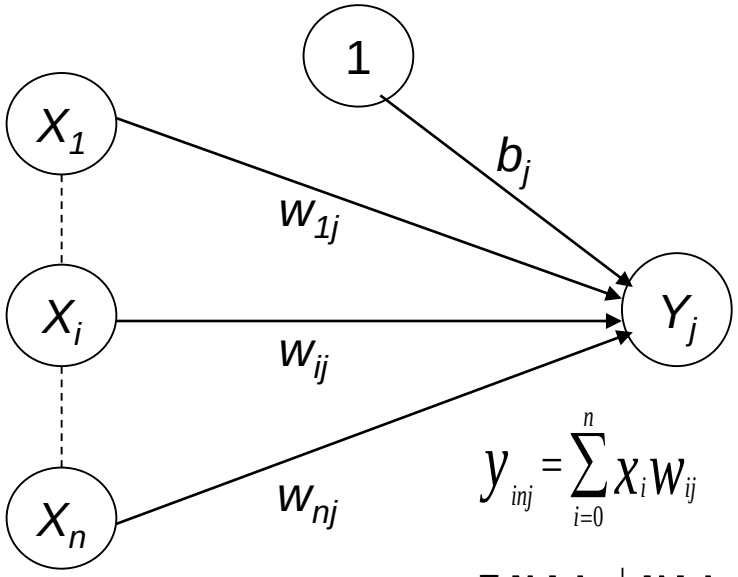
	RMSD	F1	F2	F3	F4	F5	F6	F7	F8	F9
0	17.284	13558.30	4305.35	0.31754	162.1730	1.872791e+06	215.3590	4287.87	102	27.0302
1	6.021	6191.96	1623.16	0.26213	53.3894	8.034467e+05	87.2024	3328.91	39	38.5468
2	9.275	7725.98	1726.28	0.22343	67.2887	1.075648e+06	81.7913	2981.04	29	38.8119
3	15.851	8424.58	2368.25	0.28111	67.8325	1.210472e+06	109.4390	3248.22	70	39.0651
4	7.962	7460.84	1736.94	0.23280	52.4123	1.021020e+06	94.5234	2814.42	41	39.9147
...	...	...	...	...	...	...	...	...	...	...
45725	3.762	8037.12	2777.68	0.34560	64.3390	1.105797e+06	112.7460	3384.21	84	36.8036
45726	6.521	7978.76	2508.57	0.31440	75.8654	1.116725e+06	102.2770	3974.52	54	36.0470
45727	10.356	7726.65	2489.58	0.32220	70.9903	1.076560e+06	103.6780	3290.46	46	37.4718
45728	9.791	8878.93	3055.78	0.34416	94.0314	1.242266e+06	115.1950	3421.79	41	35.6045
45729	18.827	12732.40	4444.36	0.34905	157.6300	1.788897e+06	229.4590	4626.85	141	29.8118

8.5.2 Setting up Input Data

- Splitting Target variable

```
In [9]:
1 X = dataset.iloc[:, 1:10].values
2 y = dataset.iloc[:, 0].values

In [10]:
1 X

Out[10]: array([[1.35583e+04, 4.30535e+03, 3.17540e-01, ..., 4.28787e+03,
                 1.02000e+02, 2.70302e+01],
                [6.19196e+03, 1.62316e+03, 2.62130e-01, ..., 3.32891e+03,
                 3.90000e+01, 3.85468e+01],
                [7.72598e+03, 1.72628e+03, 2.23430e-01, ..., 2.98104e+03,
                 2.90000e+01, 3.88119e+01],
                ...,
                [7.72665e+03, 2.48958e+03, 3.22200e-01, ..., 3.29046e+03,
                 4.60000e+01, 3.74718e+01],
                [8.87893e+03, 3.05578e+03, 3.44160e-01, ..., 3.42179e+03,
                 4.10000e+01, 3.56045e+01],
                [1.27324e+04, 4.44436e+03, 3.49050e-01, ..., 4.62685e+03,
                 1.41000e+02, 2.98118e+01]])

In [11]:
1 y

Out[11]: array([17.284,  6.021,  9.275, ..., 10.356,  9.791, 18.827])
```

8.5.3 Determining Network Architecture



$$y_{inj} = \sum_{i=0}^{n} x_i w_{ij}$$

$$= x_0 w_{0j} + x_1 w_{1j} + x_2 w_{2j} + ... + x_n w_{nj}$$

$$= w_{0j} + \sum_{i=1}^{n} x_i w_{ij}$$

$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

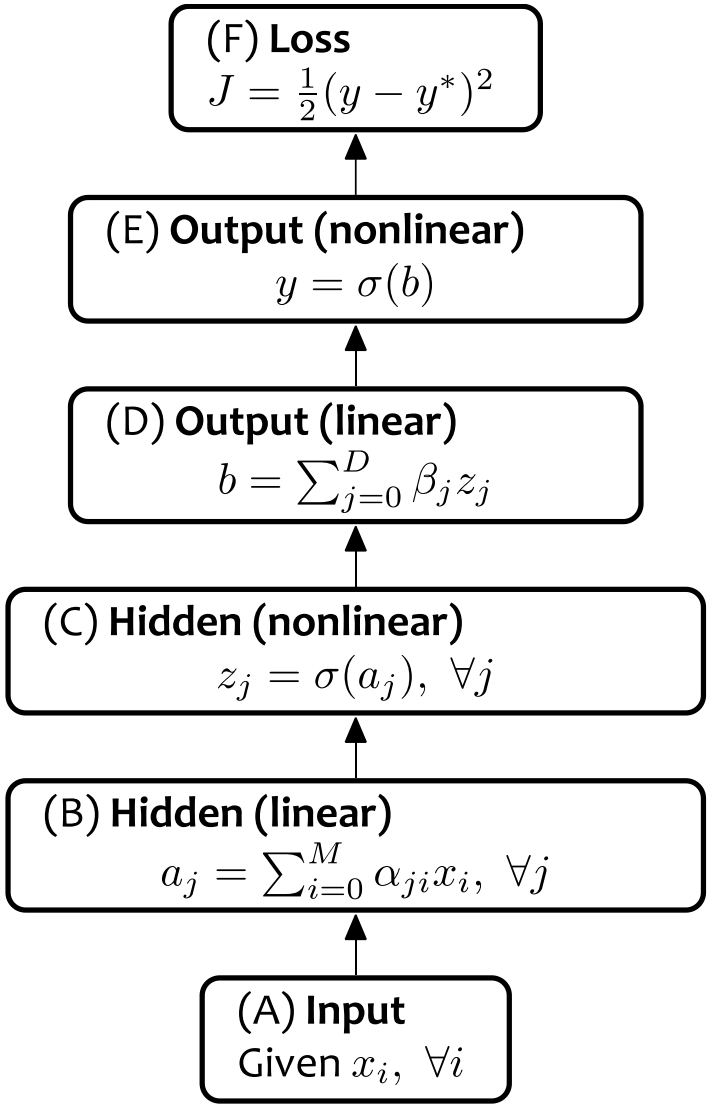
8.5.4 Model Training

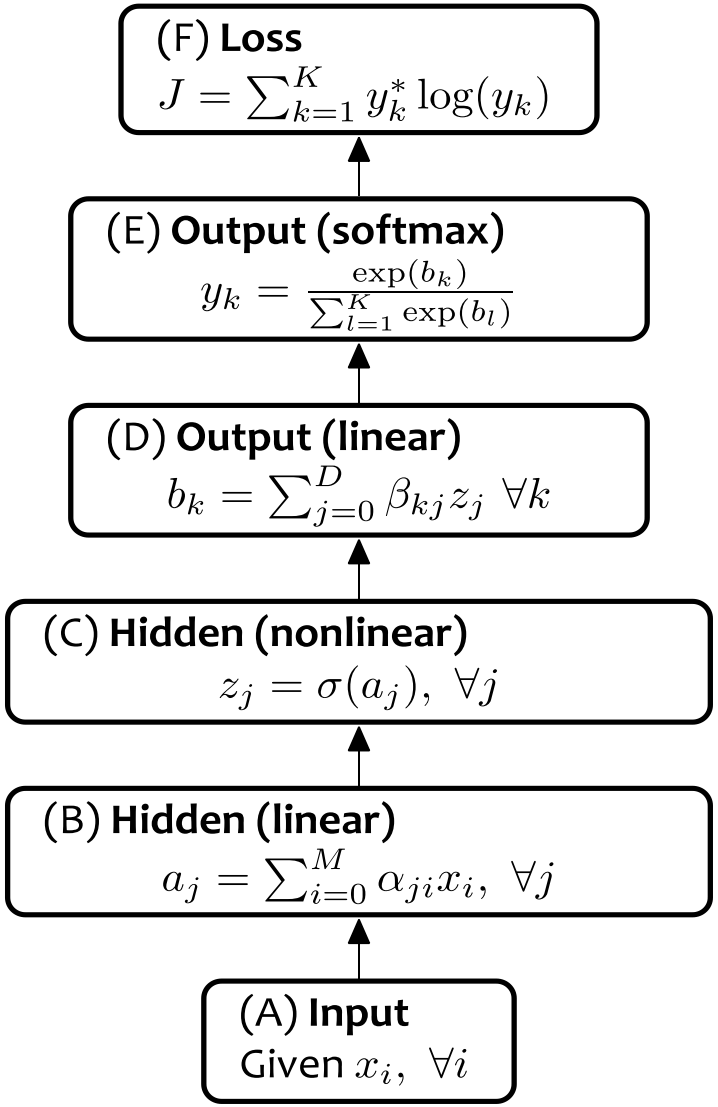
Learning Algorithm

- Forward propagation
- Backpropagation - to compute gradients of Error(loss) with respect to weights.

- Gradient Descent - accordingly optimize weights to reduce Error.
- Type of activation function

8.5.5 Model Evaluation





8.5.6 Concepts of Tuning Model Parameters

- A technique for improving the speed, performance and stability of ANN introduced in 2015.
- Proposed to solve internal covariate shift.

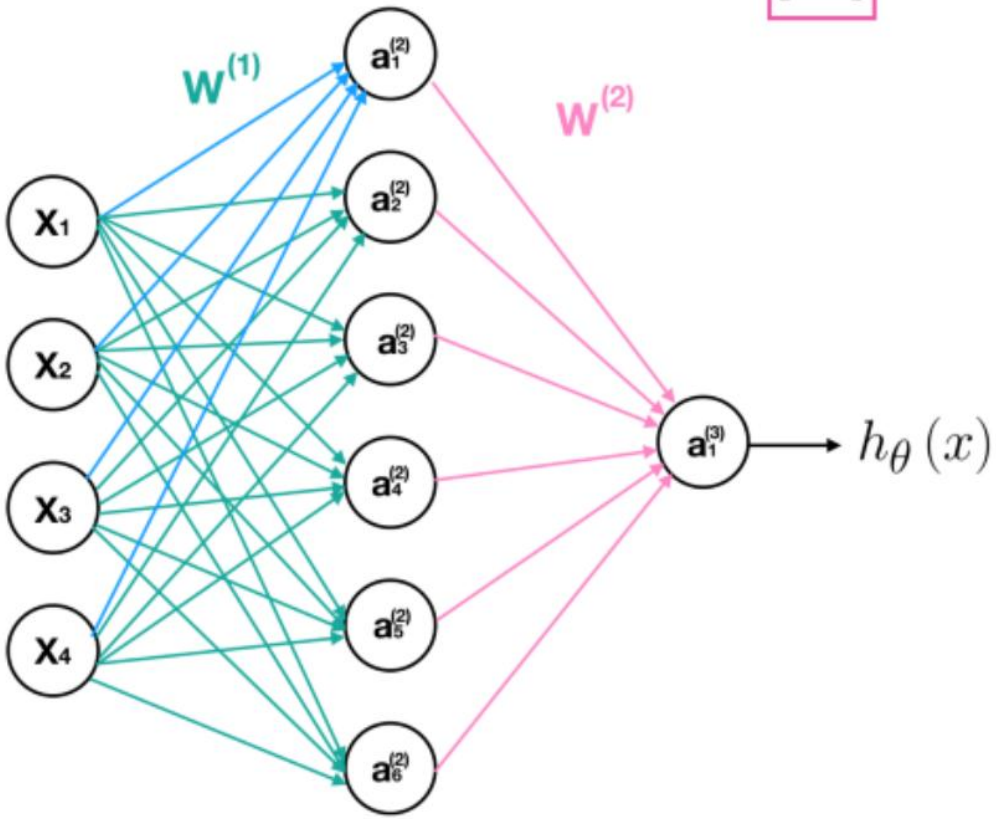
  "During training as the parameters of the preceding layers change, the distribution of inputs to the current layer changes accordingly, such that the current layer needs to constantly readjust to new distributions."
- It is used to normalize the input layer by adjusting and scaling the activations.
- The network can use higher learning rate without vanishing or exploding gradients.
- It regularizes the network such that it is easier to generalize and it is thus unnecessary to use dropout to mitigate overfitting.

Criticism

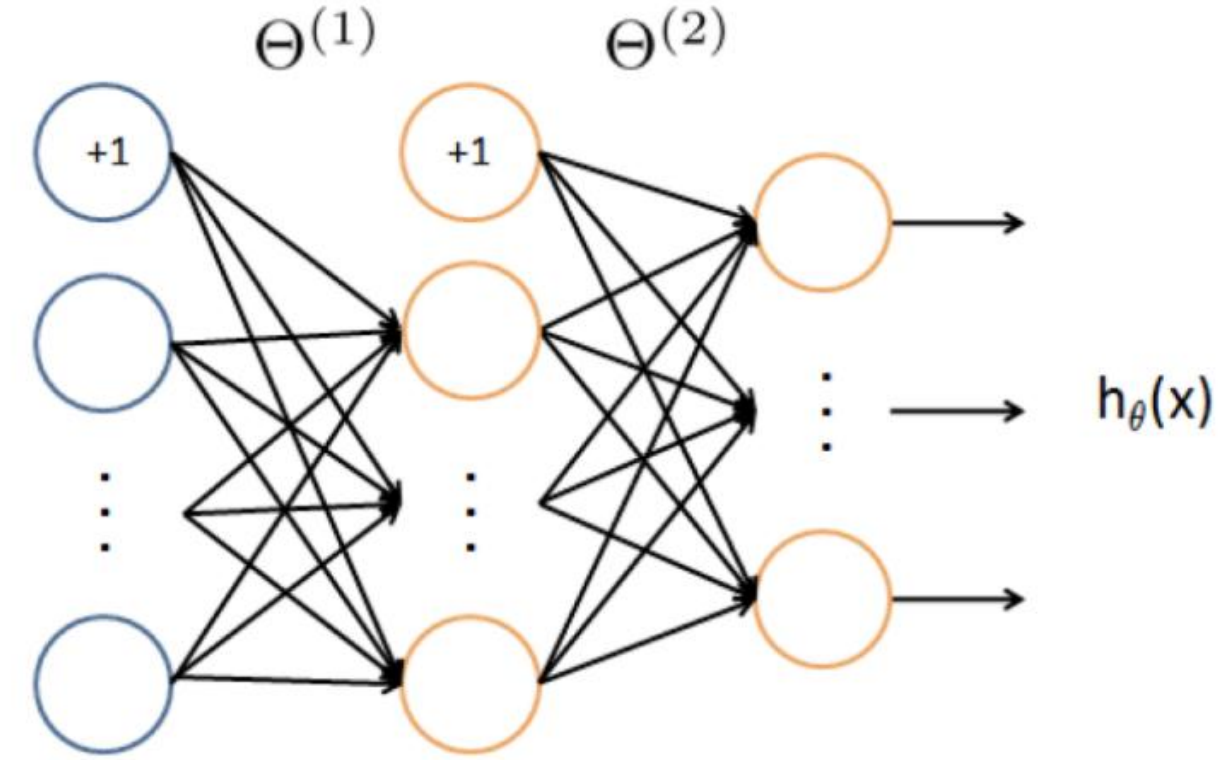
- While the effect of batch normalization is evident, the reasons remain under discussion.
- Recently, some study shown that batch normalization does not reduce internal covariate shift, but rather smooths the objective function to improve the performance.

8.5.7.1 Regression Model Output Layer

$$W^{(1)} = \begin{bmatrix} \theta_{11}^{(1)} & \theta_{21}^{(1)} & \theta_{31}^{(1)} & \theta_{41}^{(1)} & \theta_{51}^{(1)} & \theta_{61}^{(1)} \\ \theta_{12}^{(1)} & \theta_{22}^{(1)} & \theta_{32}^{(1)} & \theta_{42}^{(1)} & \theta_{52}^{(1)} & \theta_{62}^{(1)} \\ \theta_{13}^{(1)} & \theta_{23}^{(1)} & \theta_{33}^{(1)} & \theta_{43}^{(1)} & \theta_{53}^{(1)} & \theta_{63}^{(1)} \\ \theta_{14}^{(1)} & \theta_{24}^{(1)} & \theta_{34}^{(1)} & \theta_{44}^{(1)} & \theta_{54}^{(1)} & \theta_{64}^{(1)} \end{bmatrix} \qquad W^{(2)} = \begin{bmatrix} \theta_{11}^{(2)} \\ \theta_{12}^{(2)} \\ \theta_{13}^{(2)} \\ \theta_{14}^{(2)} \\ \theta_{15}^{(2)} \\ \theta_{16}^{(2)} \end{bmatrix}$$



8.5.7.2 Classification Model Output Layer



$a^{(1)} = x$ (add $a_0^{(1)}$)

$z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

$z^{(3)} = \Theta^{(2)} a^{(2)}$

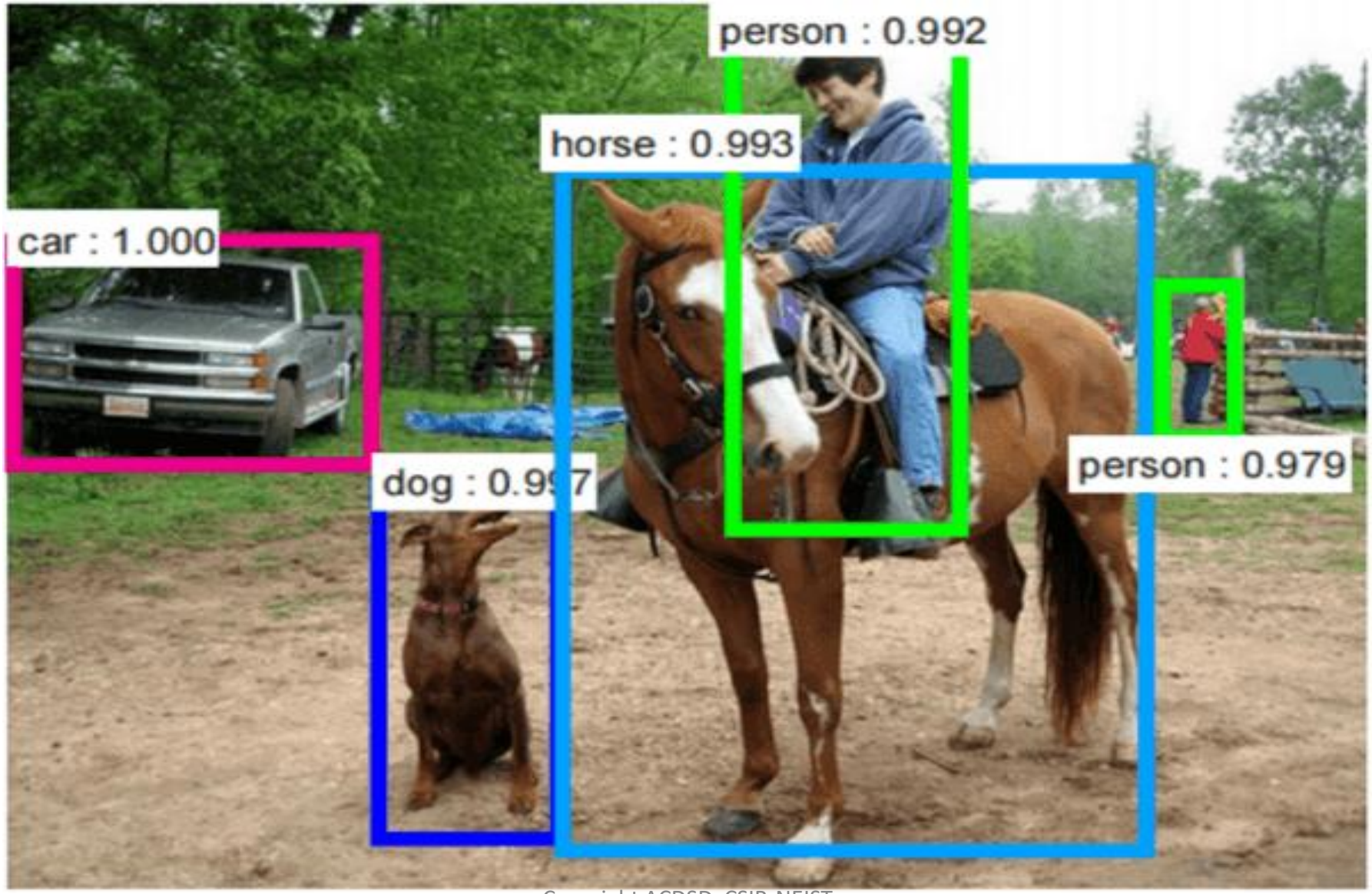
$a^{(3)} = g(z^{(3)}) = h_\theta(x)$

Input Layer Hidden Layer Output Layer

8.6 Applications

8.6.1 Computer vision
8.6.2 Speech recognition
8.6.3 NLP
8.6.4 Biology and Chemistry

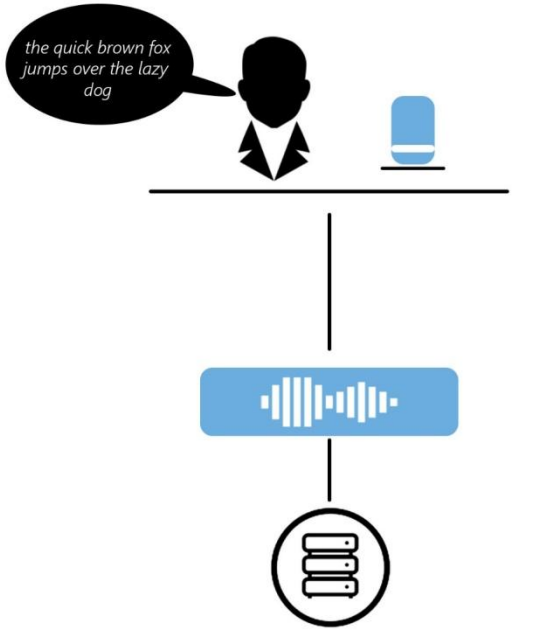
8.6.1 Computer Vision



8.6.2 Speech Recognition

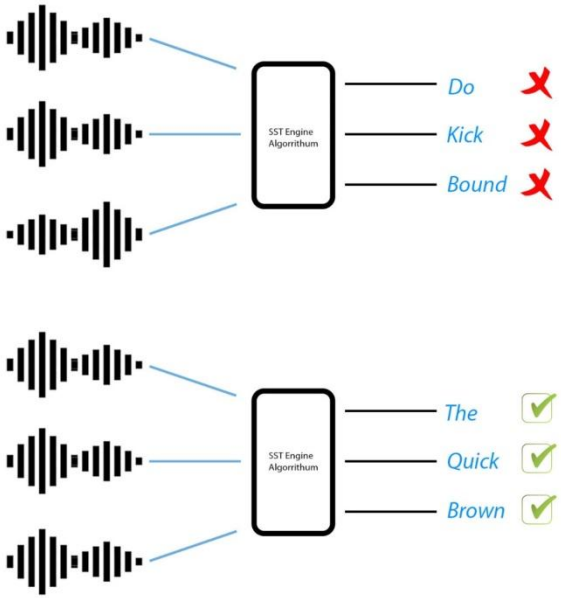
Common Voice Project

Step 1 **Record Voice**



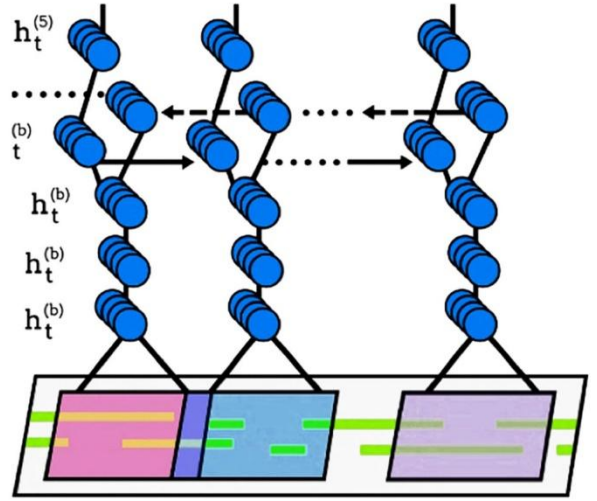
Open Sourse SST Engine

Step 2 **Voice Input Data**



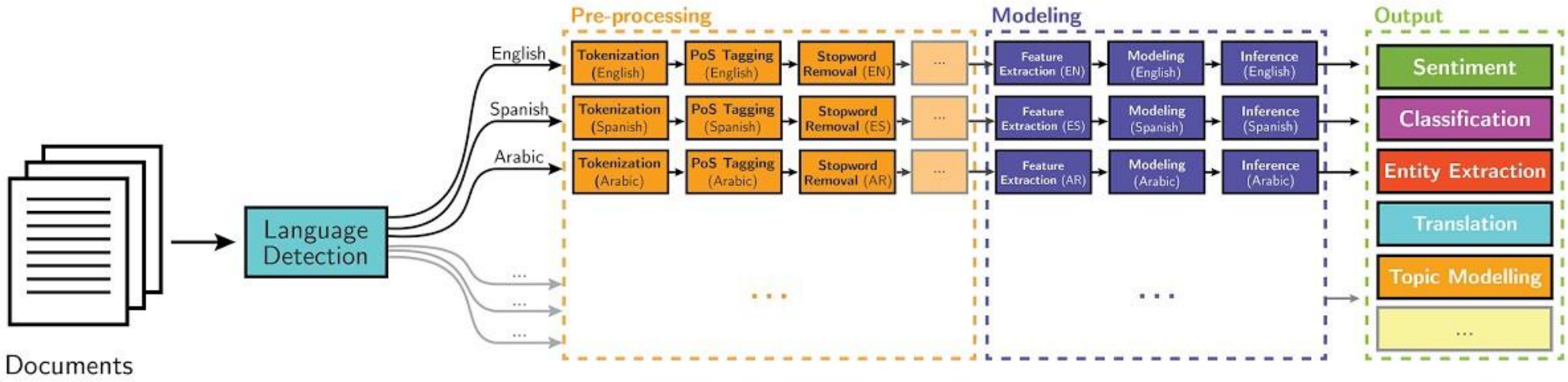
Deep Learning Architecture

Step 3 **Train the speech algorithm**

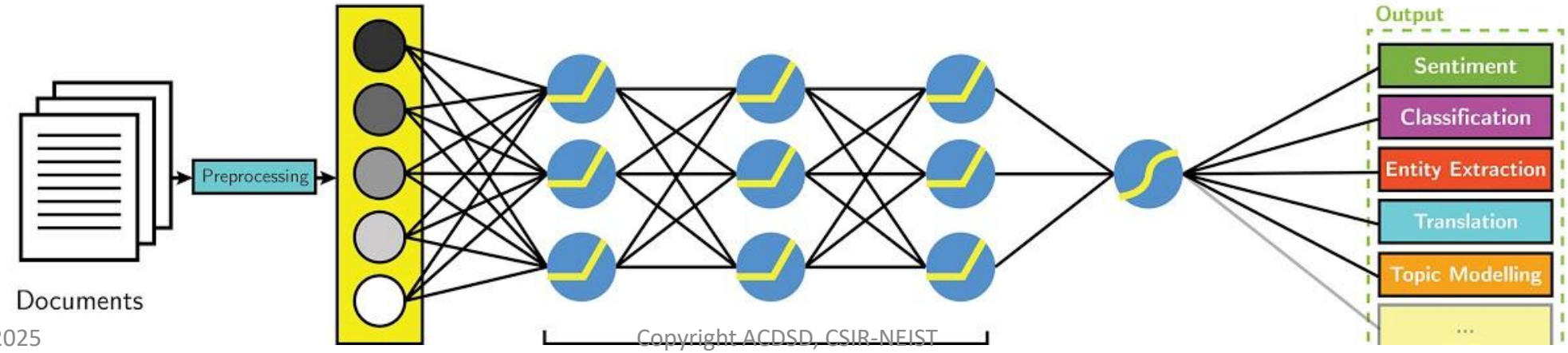


Generate **Check** **Correct** **Update** **Learn**

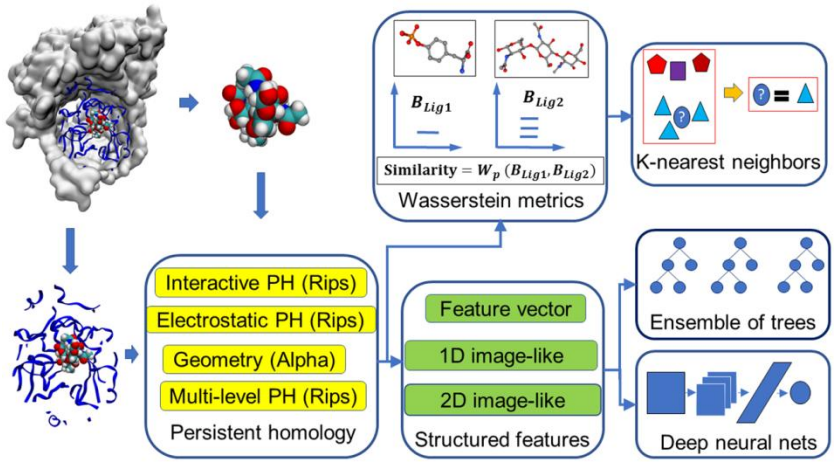
How a Speech Application Learns



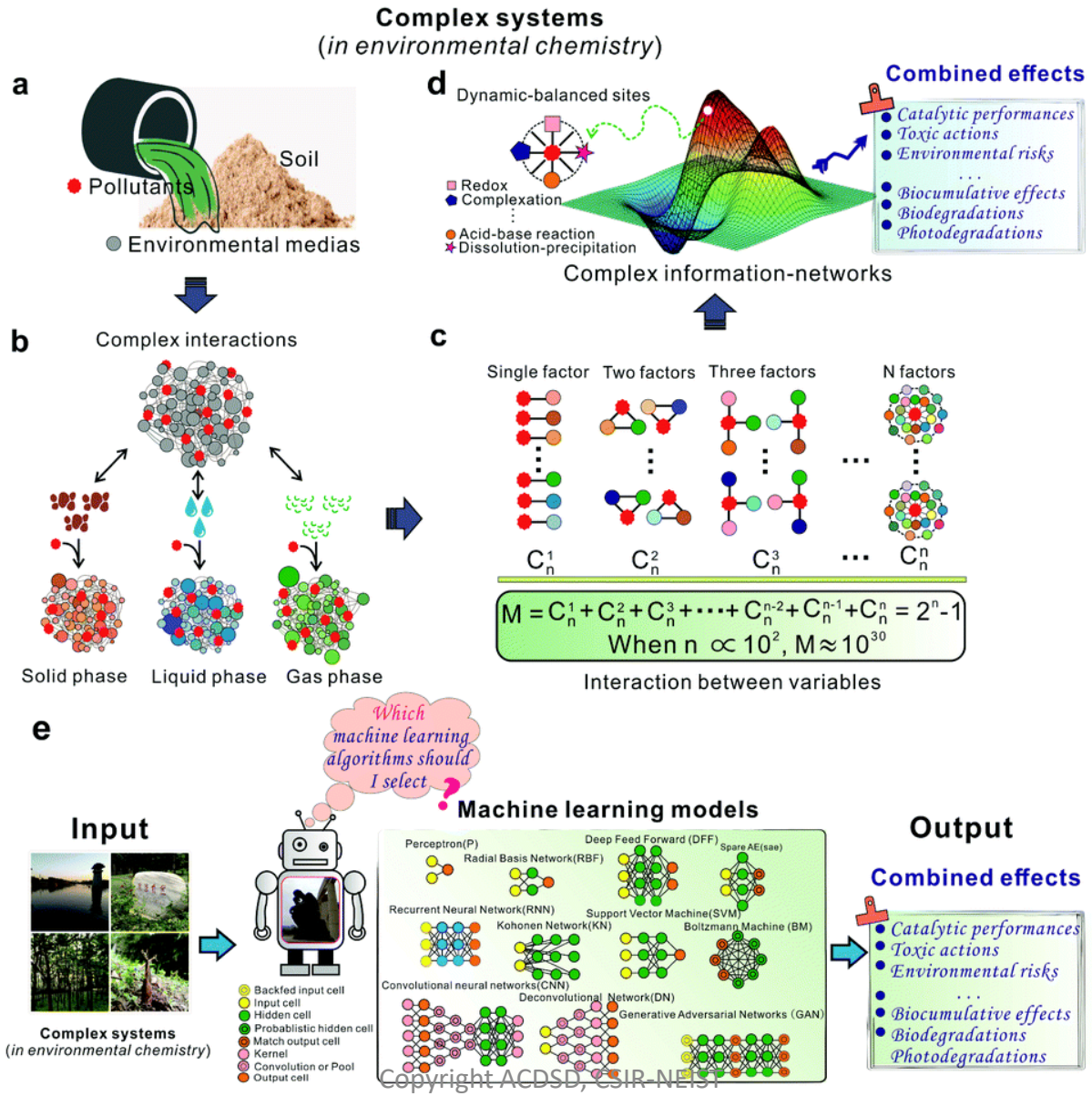
Deep Learning-based NLP



8.6.4 Biology and Chemistry



8.6.4 Biology and Chemistry



8.7 Exercises

Prof. S. wants to use an artificial neural network (ANN) to automatically determine the species of Gal´apagos finches (birds of the subfamily Geospizinae) in images using the following measurements: Beak length, beak height, eye diameter, head length, and body length. Given the location where the pictures were taken, the possible species are:

1. Large Ground Finch Geospiza magnirostris,
2. Medium Ground Finch Geospiza fortis,
3. Small Tree Finch Camarhynchus (formerly Geospiza) parvulus, and
4. Green Warbler-Finch Certhidea olivacea.

He has a database of a few hundred labeled images of individuals of these species on which to train his ANN.

Provide your input on the following design aspects for this ANN.

That is, give choices for these design aspects and justify your choices.

A. How many input units should the ANN have?

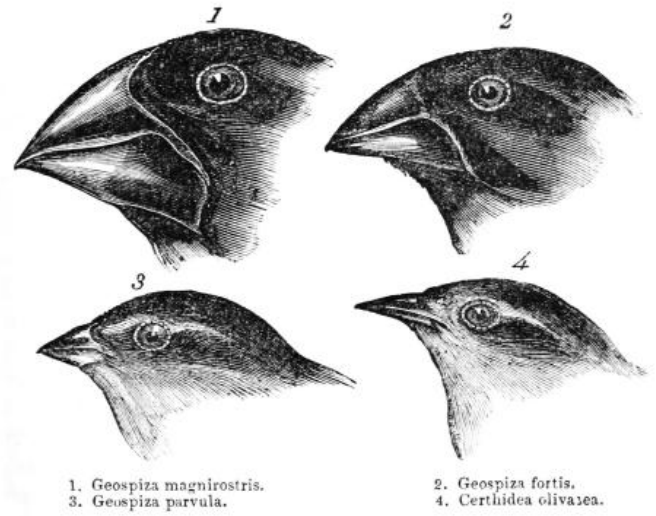
B. B. How many output units should the ANN have?

C. C. Should the ANN use hidden units or not?

D. Should the ANN use feedforward connections, recurrent connections, both, o

E. What activation function(s) should the neurons use?

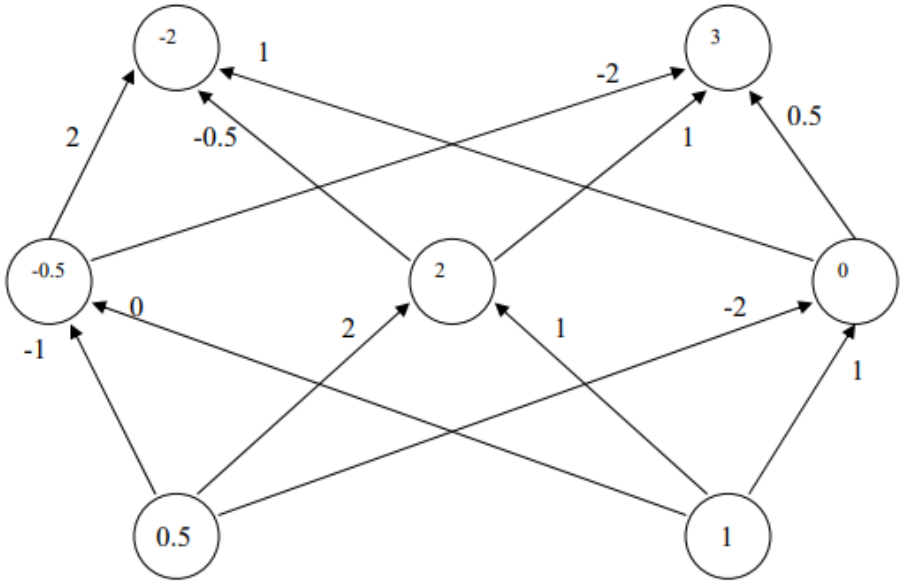
F. What learning mechanism(s) should the ANN use?



8.7 Exercises

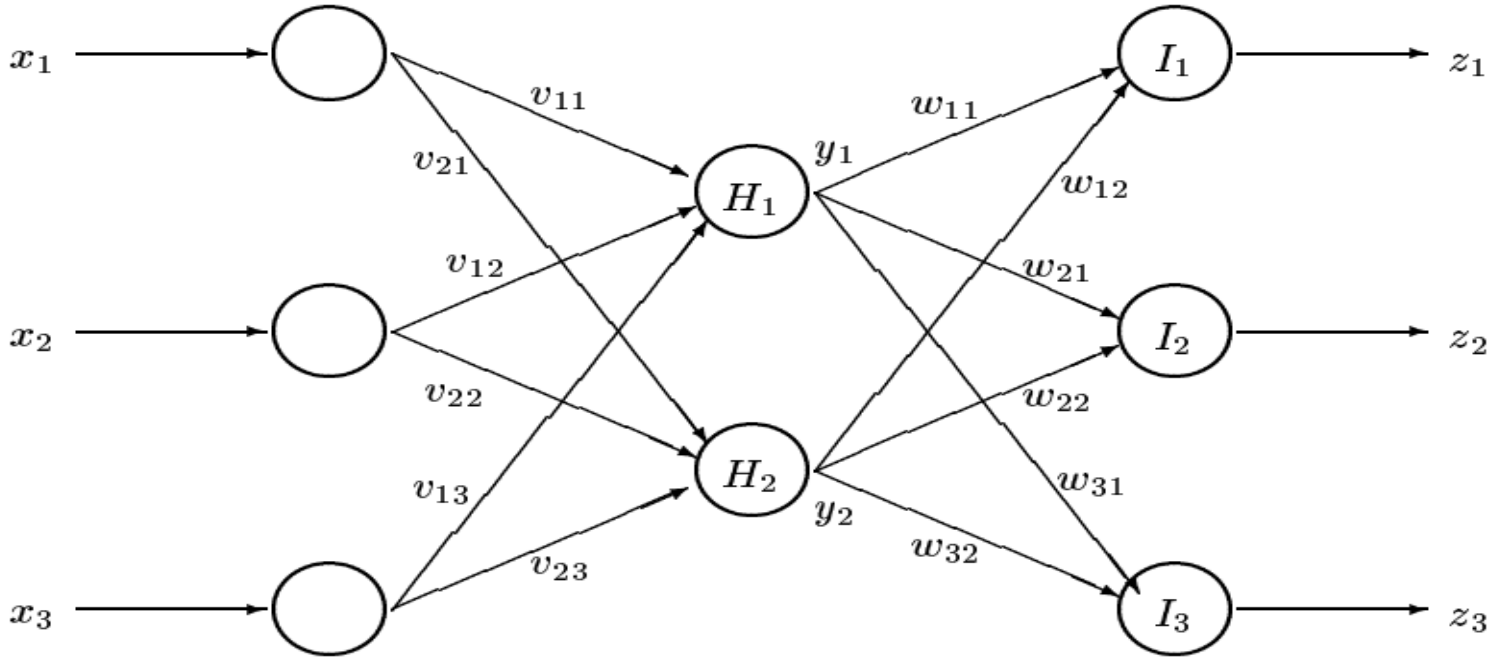
The following is a network of linear neurons, that is, neurons whose output is identical to their net input, $o_i = net_i$. The numbers in the circles indicate the output of a neuron, and the numbers at connections indicate the value of the corresponding weight.

- Compute the output of the hidden-layer and the output-layer neurons for the given input (0.5, 1) and enter those values into the corresponding circles.
- What is the output of the network for the input (1, 2), i.e. the left input neuron having the value 1 and the right one having the value 2? Do you have to do all the network computations once again in order to answer this question? Explain why you do or do not have to do this.



8.7 Exercises

A training pattern, consisting of an input vector $\mathbf{x} = [x_1, x_2, x_3]^T$ and desired outputs $\mathbf{t} = [t_1, t_2, t_3]^T$, is presented to the following neural network. What is the usual sequence of events for training the network using the backpropagation algorithm?



A. (1) calculate $z_k = f(I_k)$, (2) update w_{kj}, (3) calculate $y_j = f(H_j)$, (4) update v_{ji}.

B. (1) calculate $y_j = f(H_j)$, (2) update v_{ji}, (3) calculate $z_k = f(I_k)$, (4) update w_{kj}.

C. (1) calculate $y_j = f(H_j)$, (2) calculate $z_k = f(I_k)$, (3) update v_{ji}, (4) update w_{kj}.

D. (1) calculate $y_j = f(H_j)$, (2) calculate $z_k = f(I_k)$, (3) update w_{kj}, (4) update v_{ji}.

8.8 References

- Kelleher, J.D. (2019). *Deep Learning*. MIT Press.

➢ This book explains about deep learning and how it enables data-driven decisions by identifying and extracting patterns from large datasets. Also explains some of the basic concepts in deep learning, presents a history of advances in the field, and discusses the current state of the art. ★★★★

- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

➢ The text offers mathematical and conceptual background, covering relevant concepts in linear algebra, probability theory and information theory, numerical computation, and machine learning. ★★★★

- Chollet, F. (2017). *Deep Learning with Python*. Manning.

➢ The book introduces the field of deep learning using the Python language and the powerful Keras library. ★★★

- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media.

➢ Using concrete examples, minimal theory, and two production-ready Python frameworks—Scikit-Learn and TensorFlow, this book helps to gain an intuitive understanding of the concepts and tools for building intelligent systems. ★★★

8.8 References

- CS230: Deep Learning | Autumn 2018 | Stanford University

https://www.youtube.com/watch?v=PySo_6S4ZAg&list=PLoROMvodv4rOABXSygHTsbvUz4G_YQhOb

- MIT Introduction to Deep Learning | 6.S191 | Alexander Amini

https://www.youtube.com/watch?v=njKP3FqW3Sk&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI

- Carnegie Mellon University Deep Learning

https://www.youtube.com/watch?v=LmIjgmijyiI&list=PLp-0K3kfddPwz13VqV1PaMXF6V6dYdEsj

- Deep Learning UC Berkeley STAT-157 2019

https://www.youtube.com/watch?v=Va8WWRfw7Og&list=PLZSO_6-bSqHQHBCoGaObUljoXAyyqhpFW

- Deep Learning | IITM

https://www.youtube.com/watch?v=aPfkYu_qiF4&list=PL3pGy4HtqwD2kwldm81pszxZDJANK3uGV

- Machine Learning for Engineering and Science Applications | IITM

https://www.youtube.com/watch?v=_M-nDb0MIa4&list=PLyqSpQzTE6M-SISTunGRBRiZk7opYBf_K

- Deep Learning Lectures | Lex Fridman

https://www.youtube.com/watch?v=4zrU54VIK6k&list=PLrAXtmErZgOeiKm4sgNOknGvNjby9efdf&index=3

Thank You