INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Artificial Neural Network and Deep Learning

Dr. Animesh Chaturvedi

Assistant Professor: **IIIT Dharwad**

Young Researcher: **Heidelberg Laureate Forum**

and **Pingala Interaction in Computing**

Young Scientist: **Lindau Nobel Laureate Meetings**

Postdoc: **King's College London** & **The Alan Turing Institute**

PhD: **IIT Indore** MTech: **IIITDM Jabalpur**

LINDAU NOBEL LAUREATE MEETINGS

HEIDELBERG LAUREATE FORUM

KING'S College LONDON

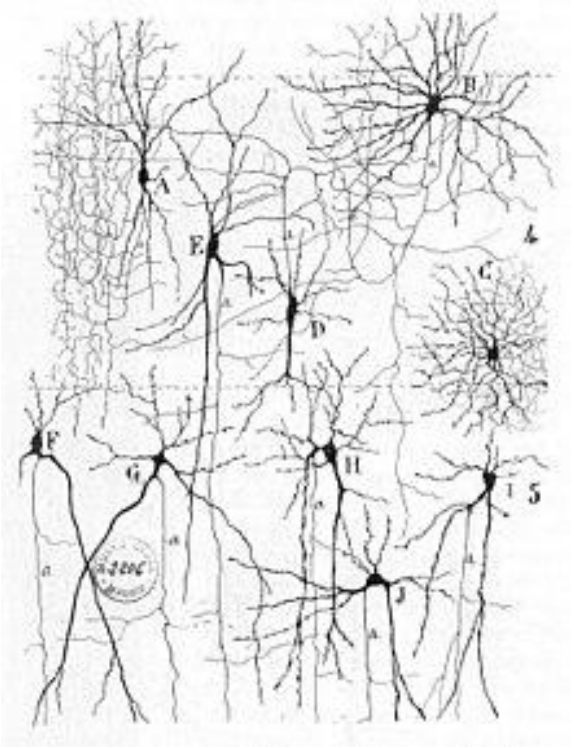
The Alan Turing Institute

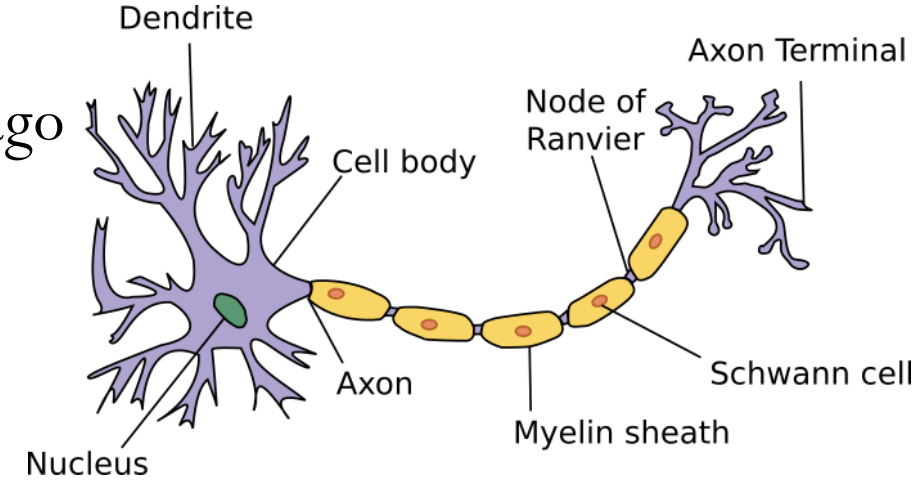
Indian Institute of Technology Indore
भारतीय प्रौद्योगिकी संस्थान इंदौर

PDPM
Indian Institute of Information Technology, Design and Manufacturing, Jabalpur

Neural Network (Anatomy)

- **1870s:** Reticular Theory "nervous system and brain is a single continuous network, all nerve cells in the nervous system constituted a continuous, interconnected network"
- **1888:** Neurons "fundamental units of the brain and nervous system, each nerve cell is an independent entity and nerve synapses transfer nerve impulses from one cell to another"
- **1906:** Nobel prize to both Camillo Golgi and Santiago Ramón y Cajal "in recognition of their work on the structure of the nervous system."

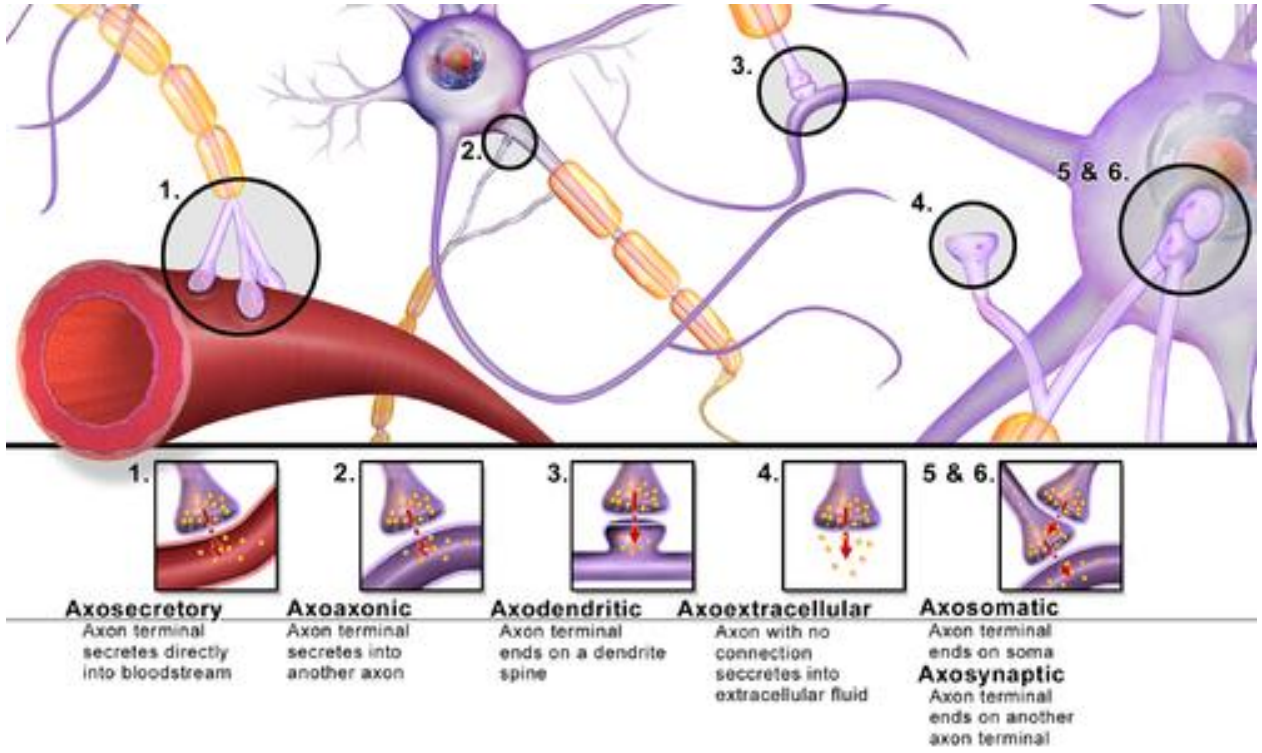




https://www.nobelprize.org/prizes/medicine/1906/summary/

Neural Network (Anatomy)

- Synapse is a structure that permits a neuron (or nerve cell) to pass an electrical or chemical signal to another neuron or to the target effector cell.
- Synapses are essential to the transmission of nervous impulses from one neuron to another.
- Different types of synapses



Perceptron

Perceptron

- **1943 Perceptron** was invented by **McCulloch and Pitts**
- **1958 Mark I Perceptron** hardware developed and constructed by **Frank Rosenblatt**
- Rosenblatt proved that a perceptron will learn any linearly separable function.

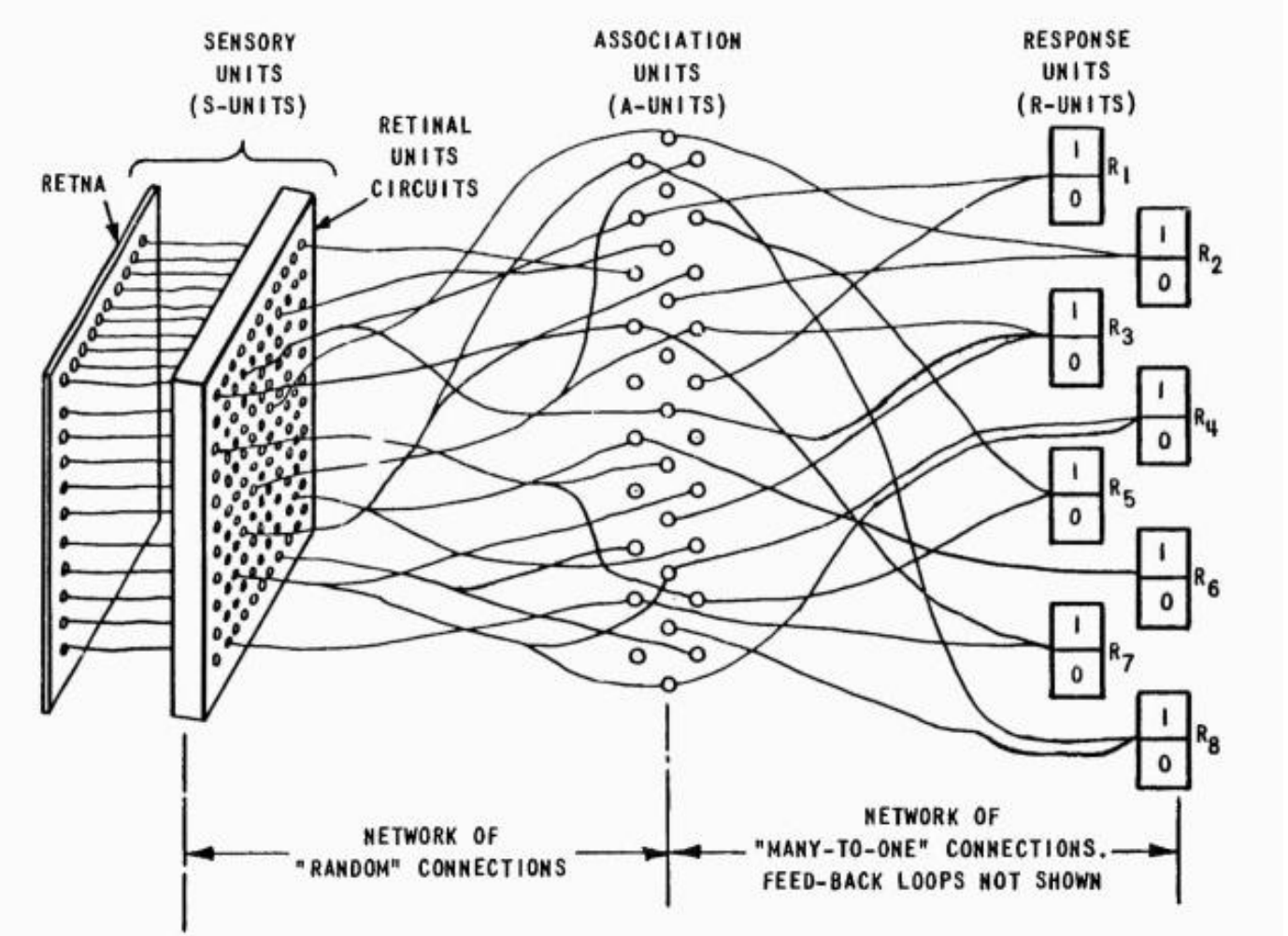
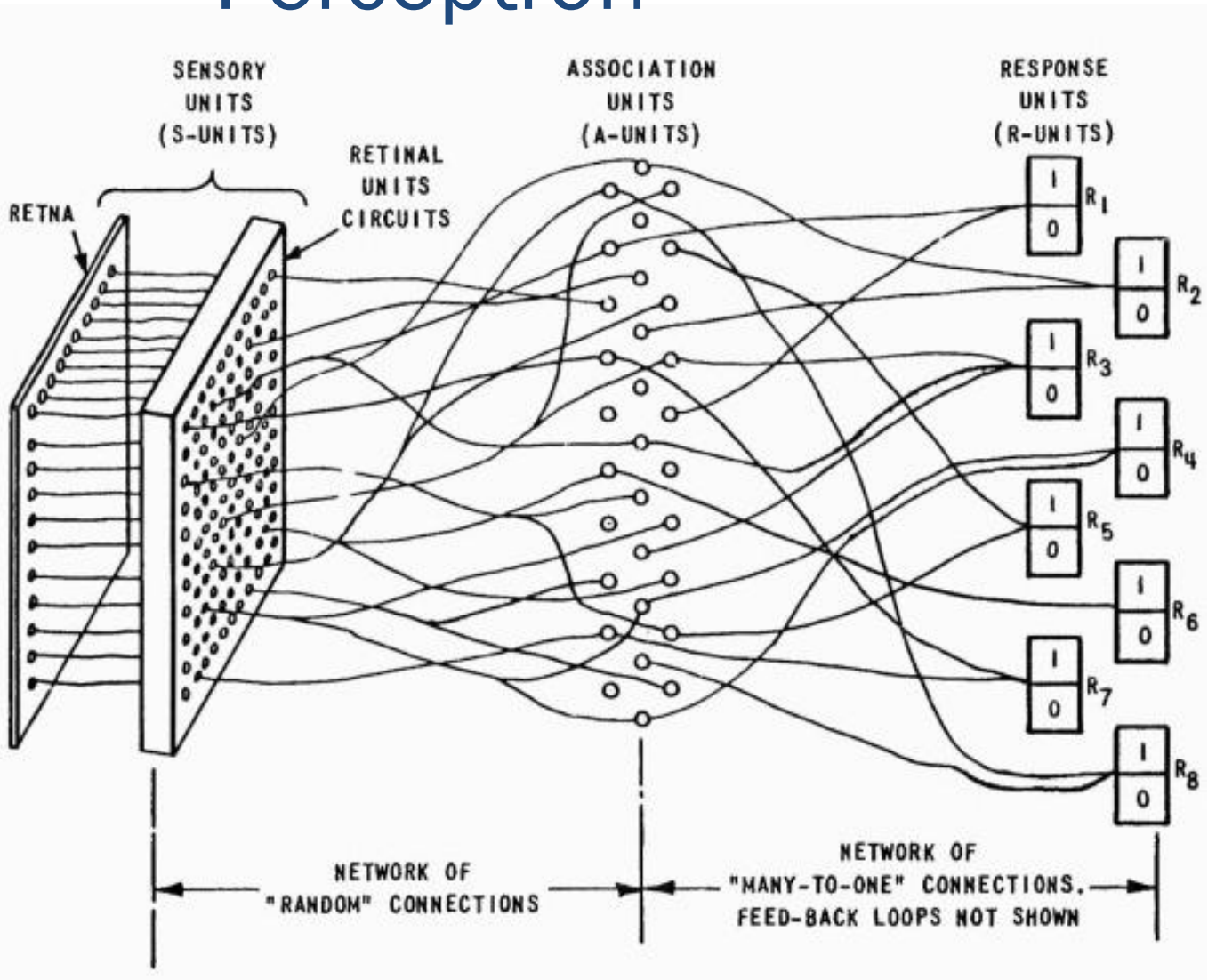
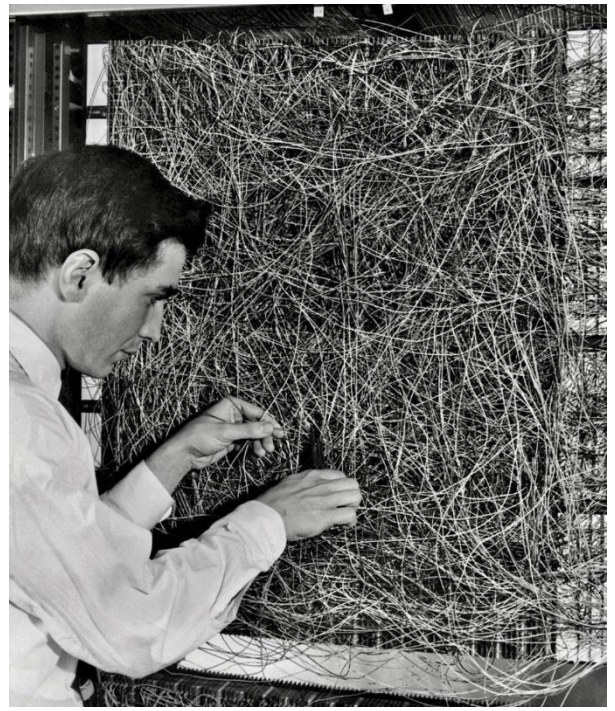
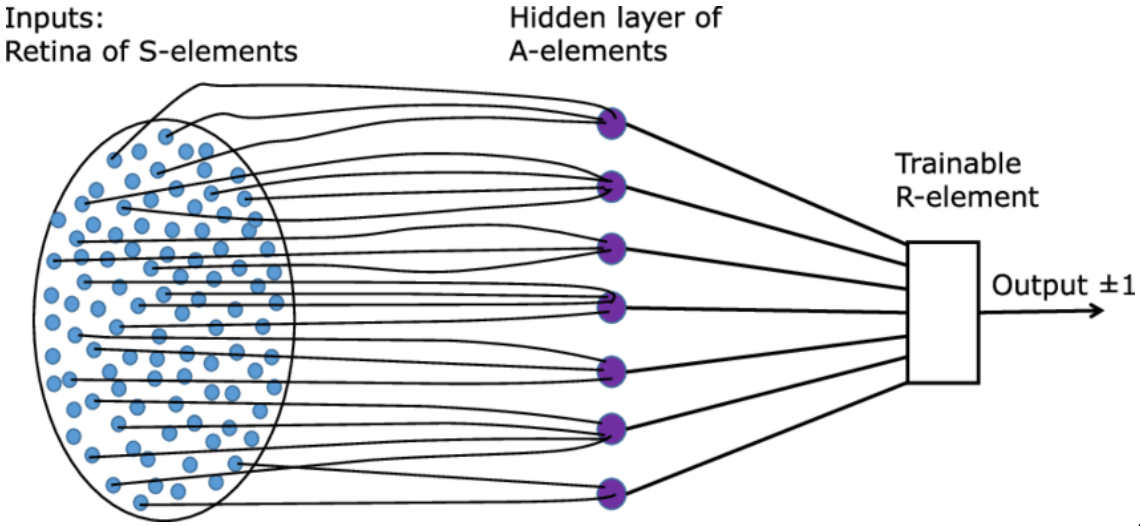


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

Perceptron


SENSORY UNITS (S-UNITS)
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
RETINAL UNITS CIRCUITS
RETNA
R1
R2
R3
R4
R5
R6
R7
R8
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS. FEED-BACK LOOPS NOT SHOWN



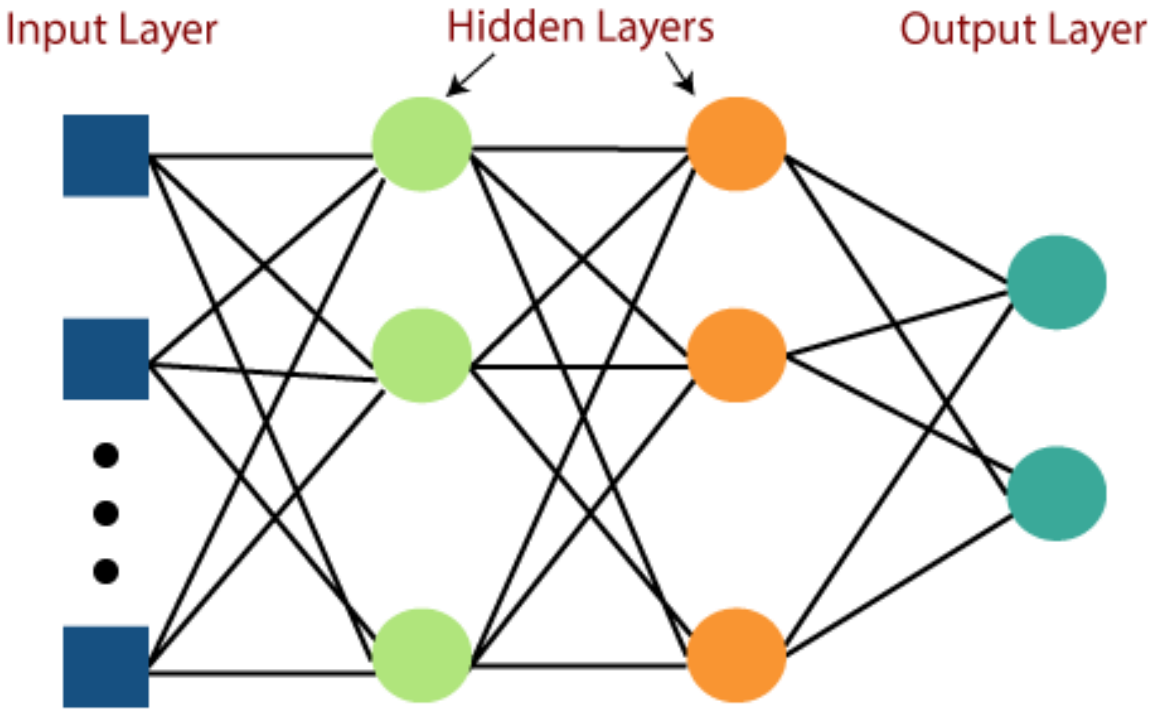

Inputs:
Retina of S-elements
Hidden layer of
A-elements
Trainable
R-element
Output ±1

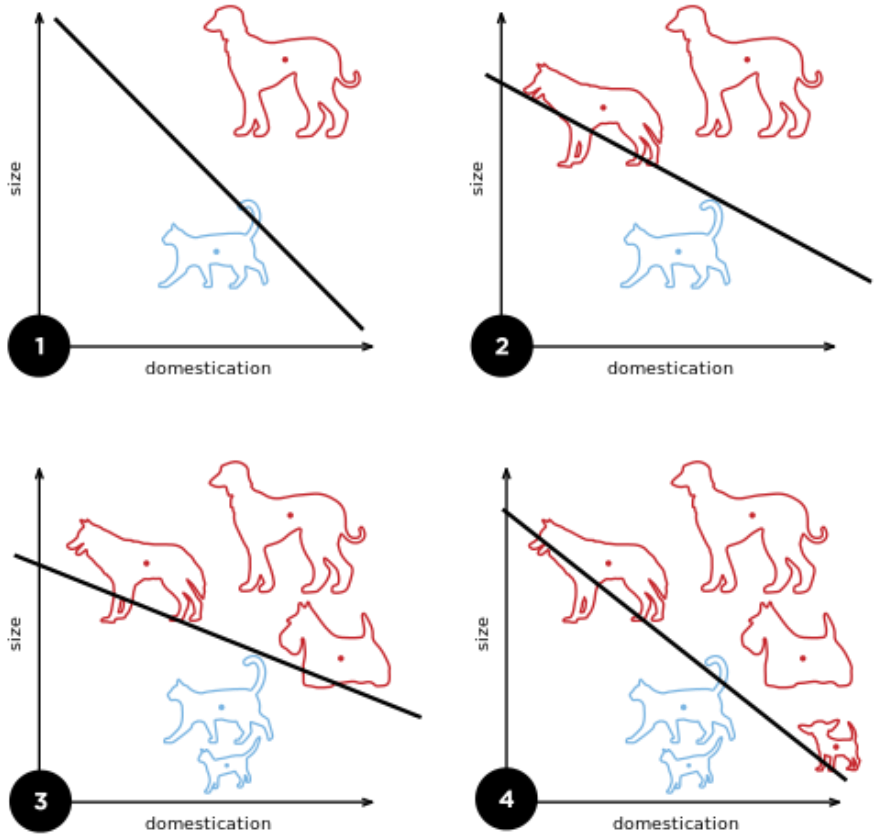
Multi-layer Perceptron (MLP)

- **Alexey Ivakhnenko** et al. introduced the first multilayer neural networks; a method called the Group Method of Data Handling (GMDH).
- First system that could train a feedforward multilayer perceptron with multiple hidden layers.
- The 1965 GMDH method was the first learning algorithm that could train a multilayer neural network with multiple hidden layers.
- Learning method employed supervised machine learning techniques, including least squares analysis, to train the models layer by layer.
- Significance: development of a multilayer perceptron (MLP).

Multi-layer Perceptron (MLP)

- Ivakhnenko et. al. introduced Multilayer Perceptrons
- **1960s** Limitation of Perceptron: Failed to classify basic problems





Artificial Neural Network

Artificial Neural Network

- Feedforward Neural Networks,
  Back-propagation, and
  Stochastic Gradient Descent
- Recursive relations with back-propagation algorithm to compute the gradient of the loss function with respect to the weight parameters.
- Use the stochastic descent algorithm to train a feedforward neural network.
- It is not guaranteed to reach global (only local) optimum with SGD to minimize the training loss.
- Recognize when a network has overcapacity .

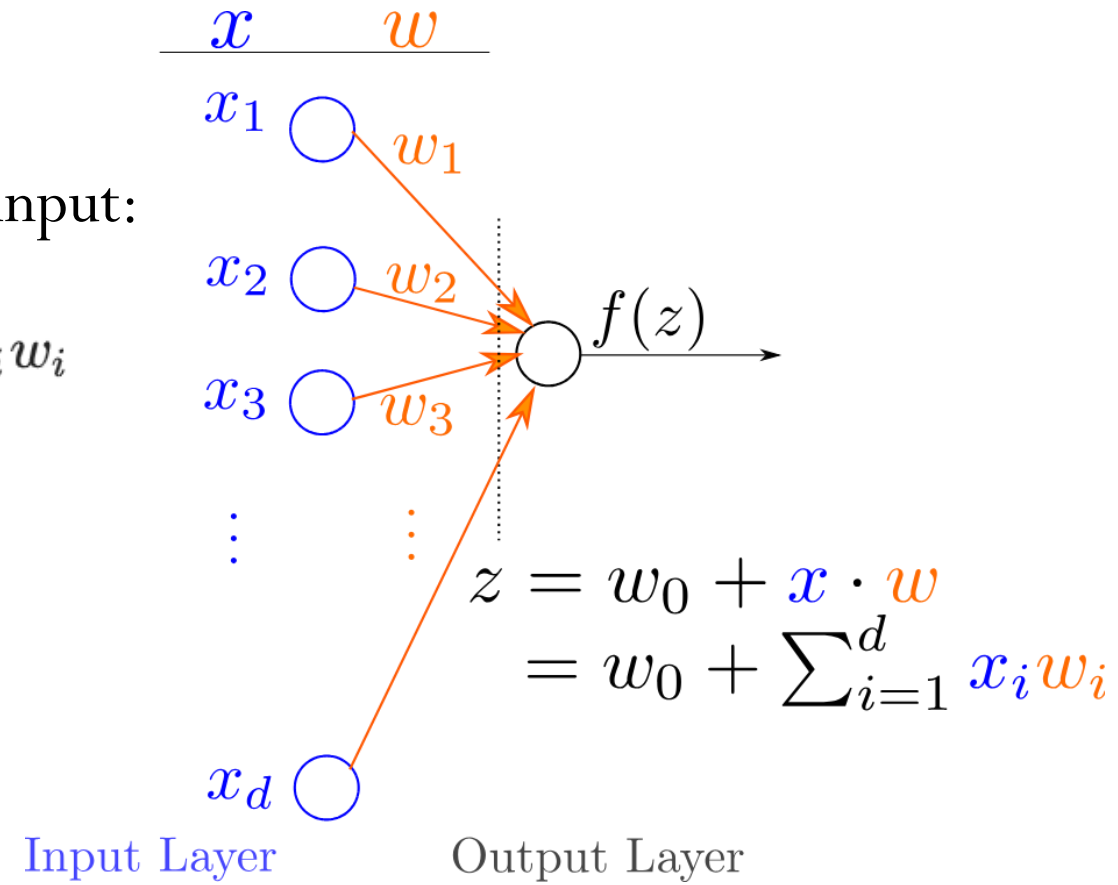
Feedforward Neural Networks

- Recognize different layers in a feedforward neural network and the number of units in each layer.
- Activation functions such as the
  - hyperbolic tangent function , and
  - rectified linear function (ReLU) .
- Compute the output of a simple neural network possibly with hidden layers given the weights and activation functions .
- Determine whether data after transformation by some layers is linearly separable, draw decision boundaries given by the weight vectors and use them to help understand the behavior of the network.

Neural Network Units

- A **neural network unit** is a primitive neural network that consists of only the "input layer", and an output layer with only one output. It is represented pictorially as follows:
- A neural network unit computes a non-linear weighted combination of its input:

$$\hat{y} = f(z) \quad \text{where } z = w_0 + \sum_{i=1}^{d} x_i w_i$$



Neural Network Linear Combination Example

$$z = w_0 + \sum_{i=1}^{d} x_i w_i$$

$$x = [0, 1], \quad w = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad w_0 = -3$$

Step 1: Multiply each x_i and w_i

$$x_1 w_1 = 0 \times 1 = 0$$

$$x_2 w_2 = 1 \times (-1) = -1$$

Step 2: Sum the products

$$\sum_{i=1}^{2} x_i w_i = 0 + (-1) = -1$$

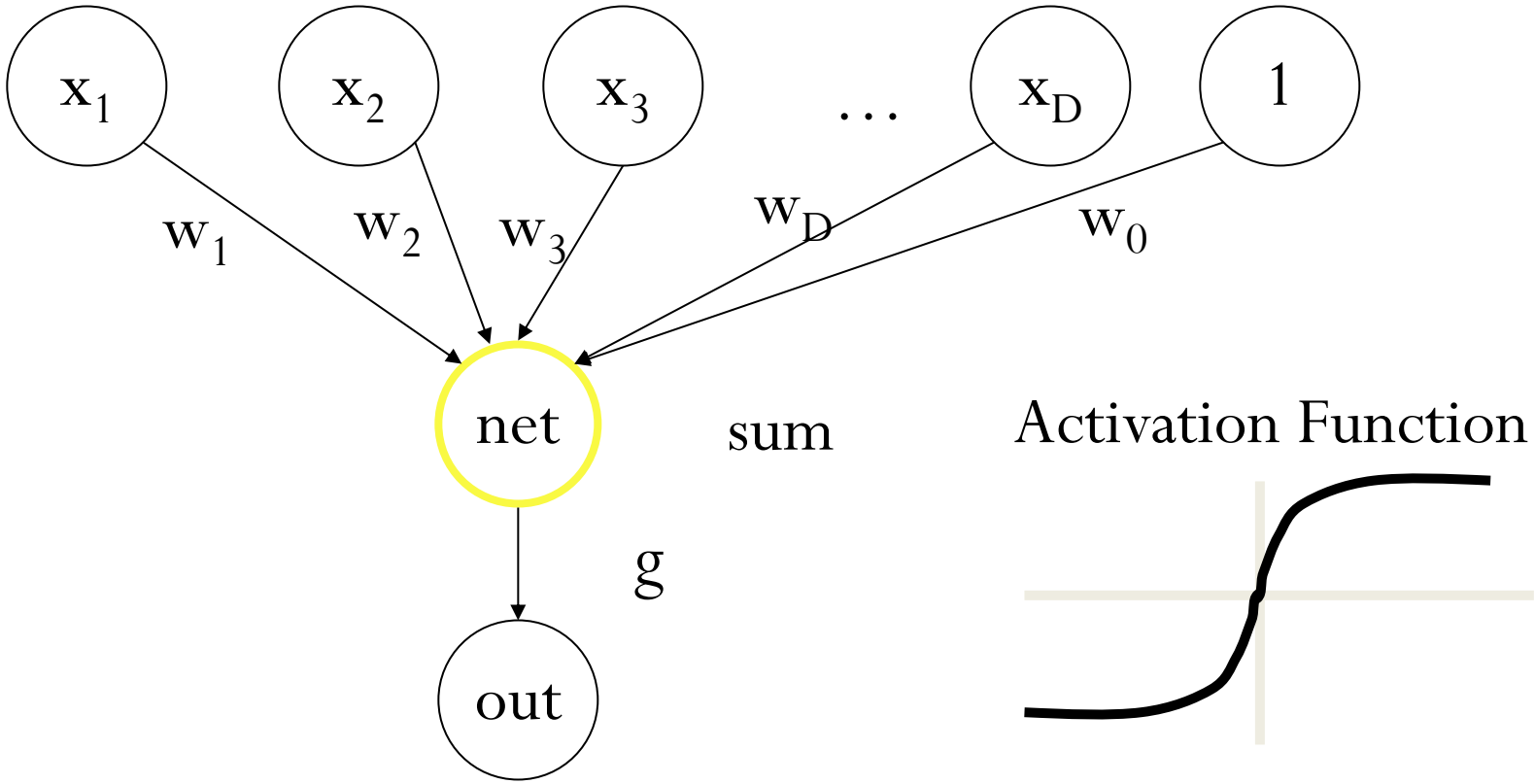
Step 3: Add the bias

$$z = w_0 + (-1) = -3 + (-1) = -4$$

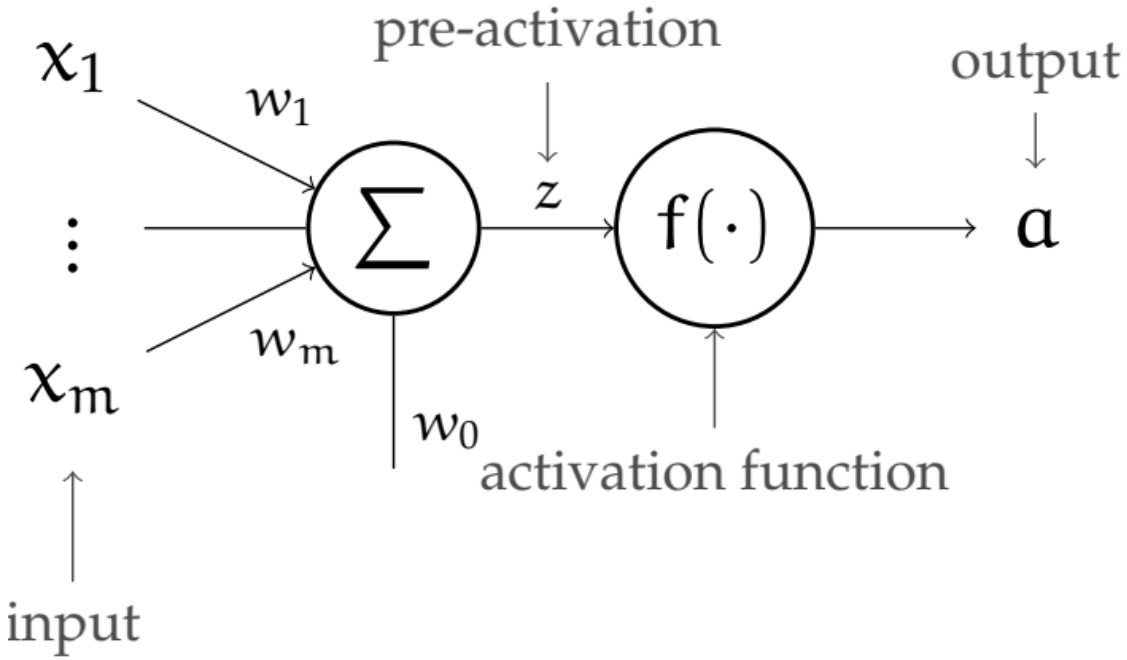
Activation Function and Loss functions

Classification Neural Network

- Activation function makes the output look more like bits: 0 or 1 decisions.



Neural Network Units



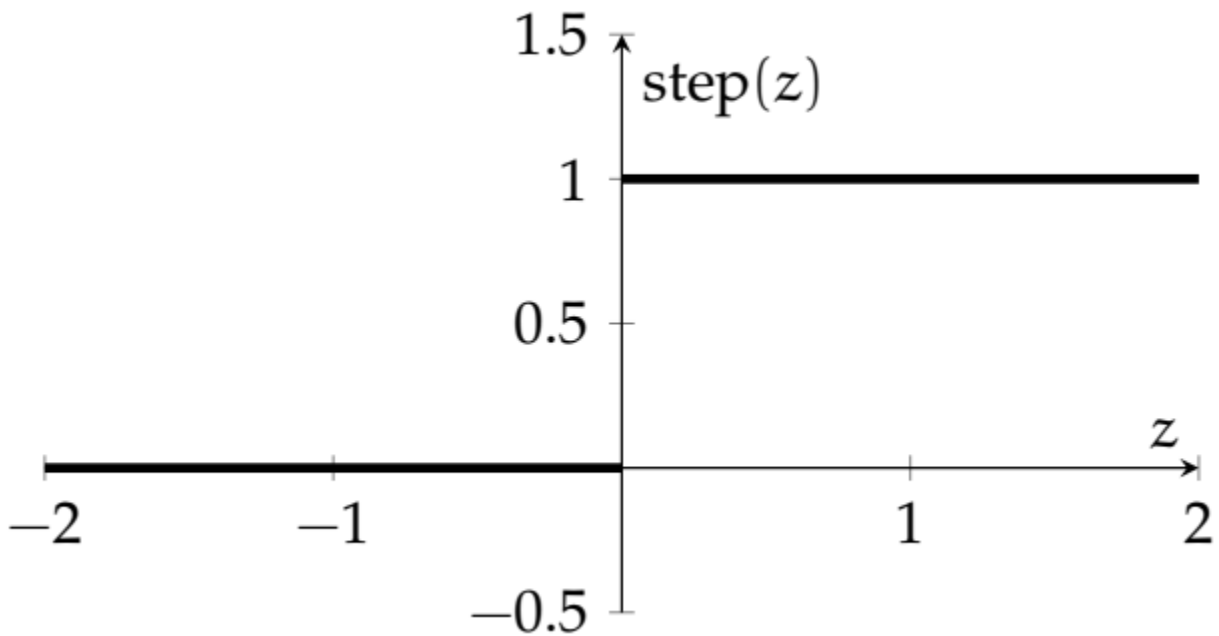
$$a = f(z) = f\left(\sum_{j=1}^{m} x_j w_j + w_0\right) = f(w^T x + w_0)$$

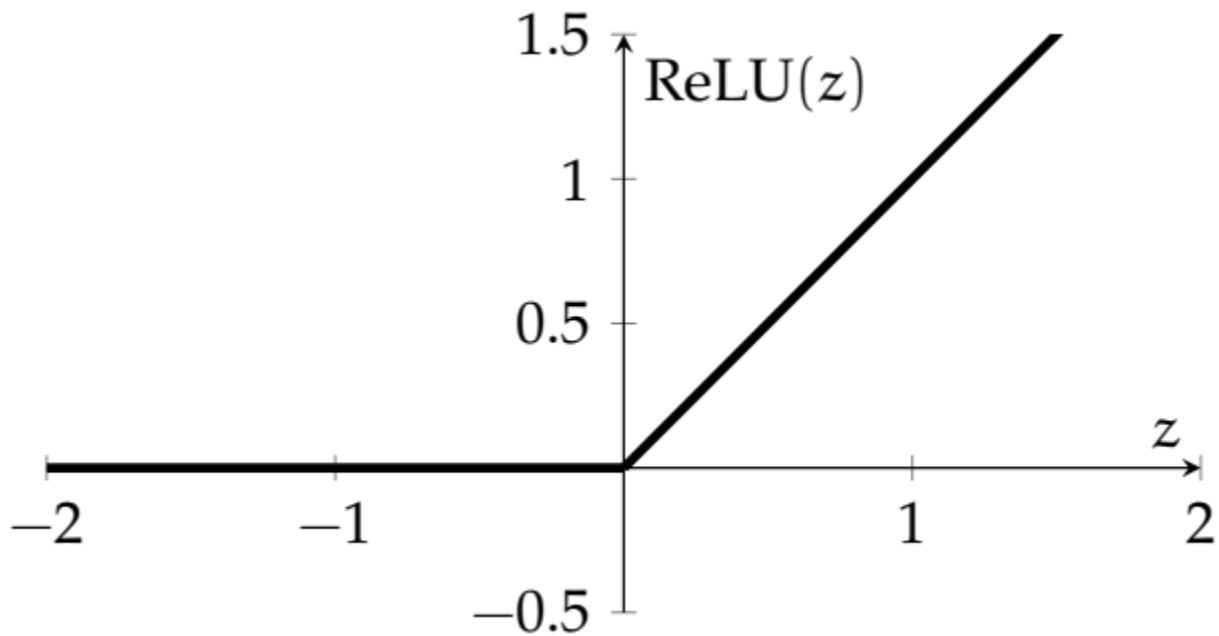
Step function

$$\text{step}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{otherwise} \end{cases}$$

Rectified linear unit

$$\text{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} = \max(0, z)$$



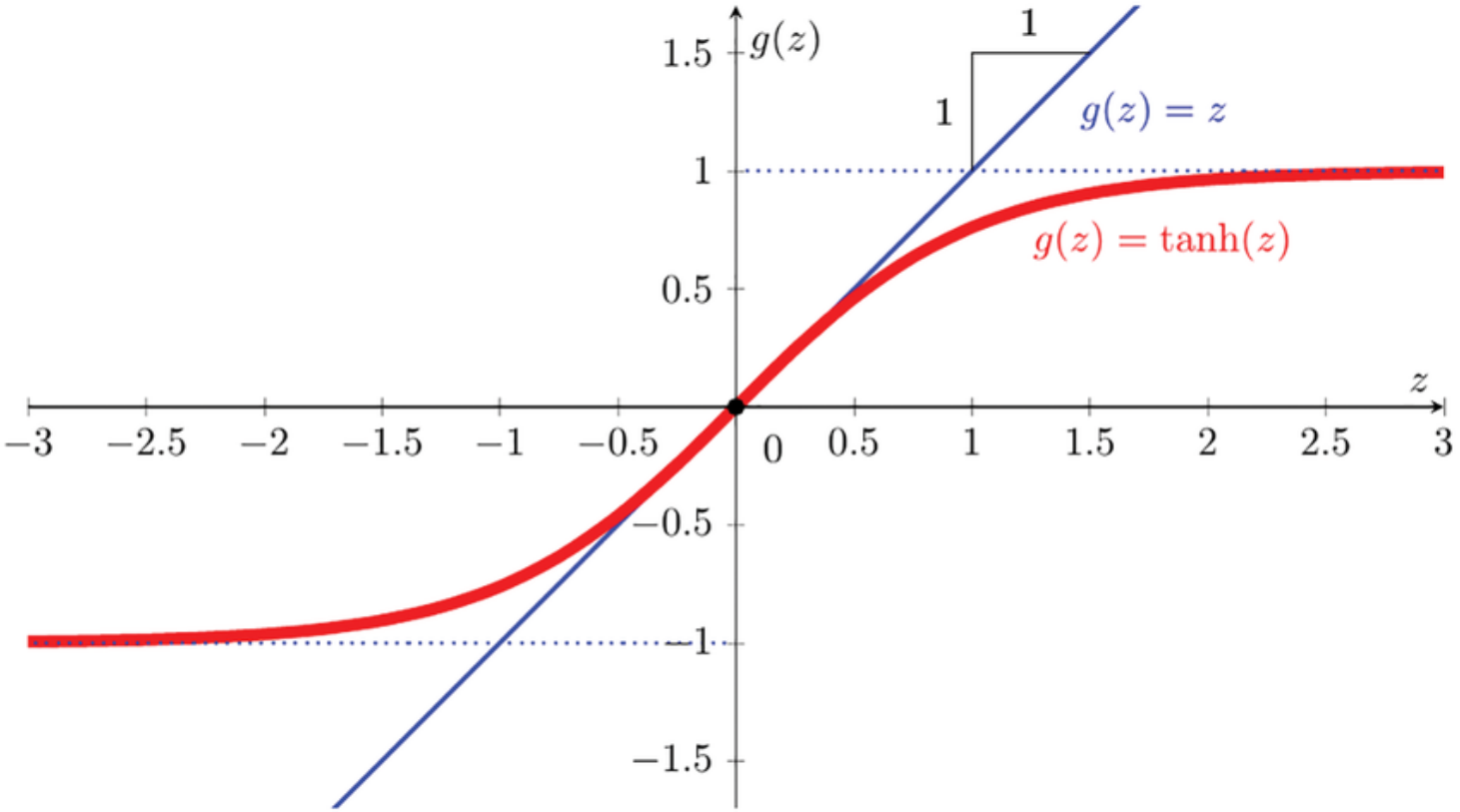


Activation Function (Hyperbolic Tangent Function)

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 1 - \frac{2}{e^{2z} + 1}$$

- tanh is defined for **all real z**, because the denominator is never zero.

$$\tanh(0) = \frac{e^0 - e^{-0}}{e^0 + e^{-0}} = \frac{1 - 1}{1 + 1} = 0$$



Activation Function

- Is tanh(z) odd
- Range

A function f is **odd** if $f(-z) = -f(z)$, and **even** if $f(-z) = f(z)$.

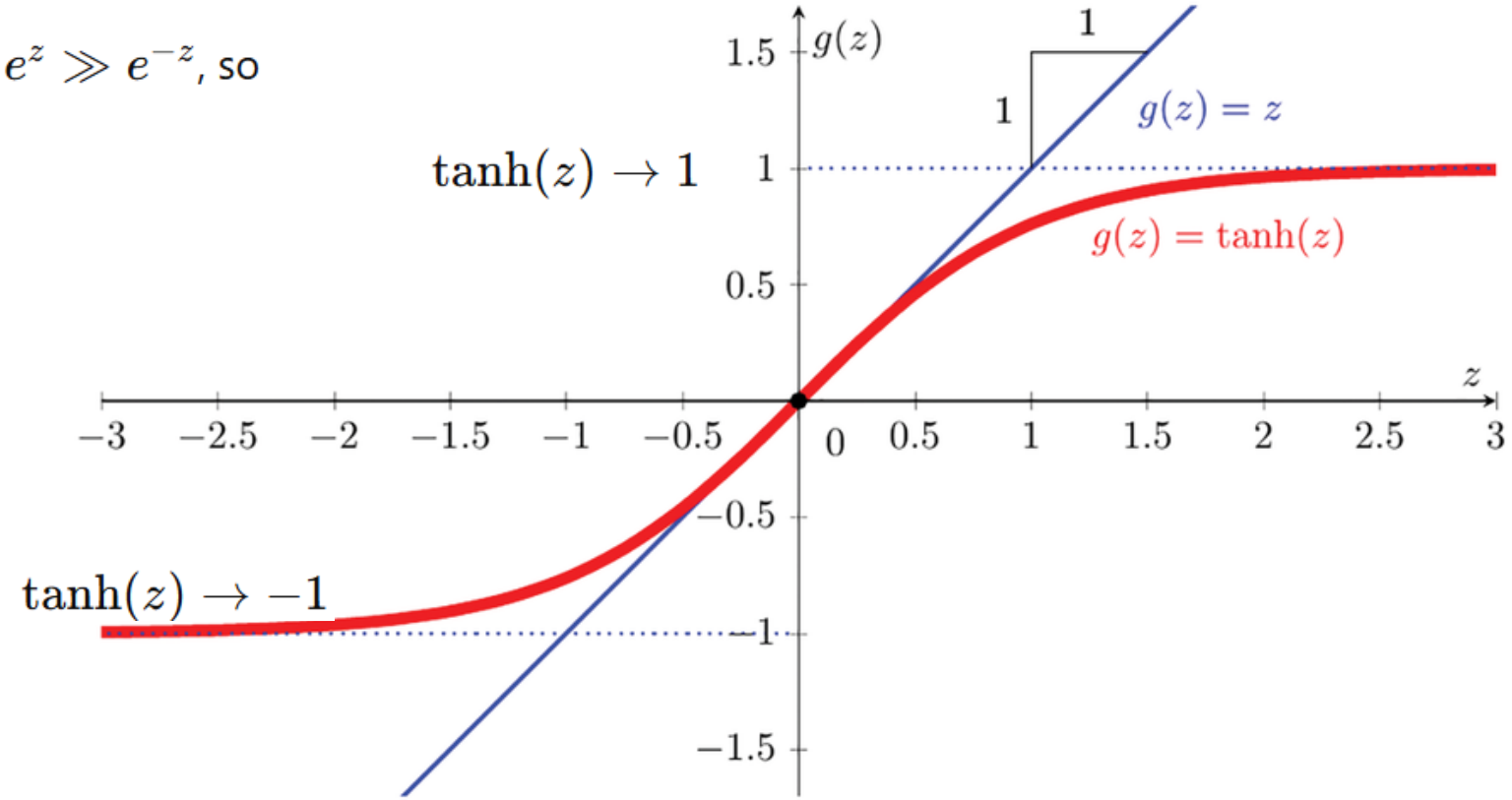
$$\tanh(-z) = \frac{e^{-z} - e^{z}}{e^{-z} + e^{z}} = -\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} = -\tanh(z)$$

As $z \to +\infty$, $e^{z} \gg e^{-z}$, so

$\tanh(z) \to 1$

As $z \to -\infty$, $e^{-z} \gg e^{z}$, so

$\tanh(z) \to -1$

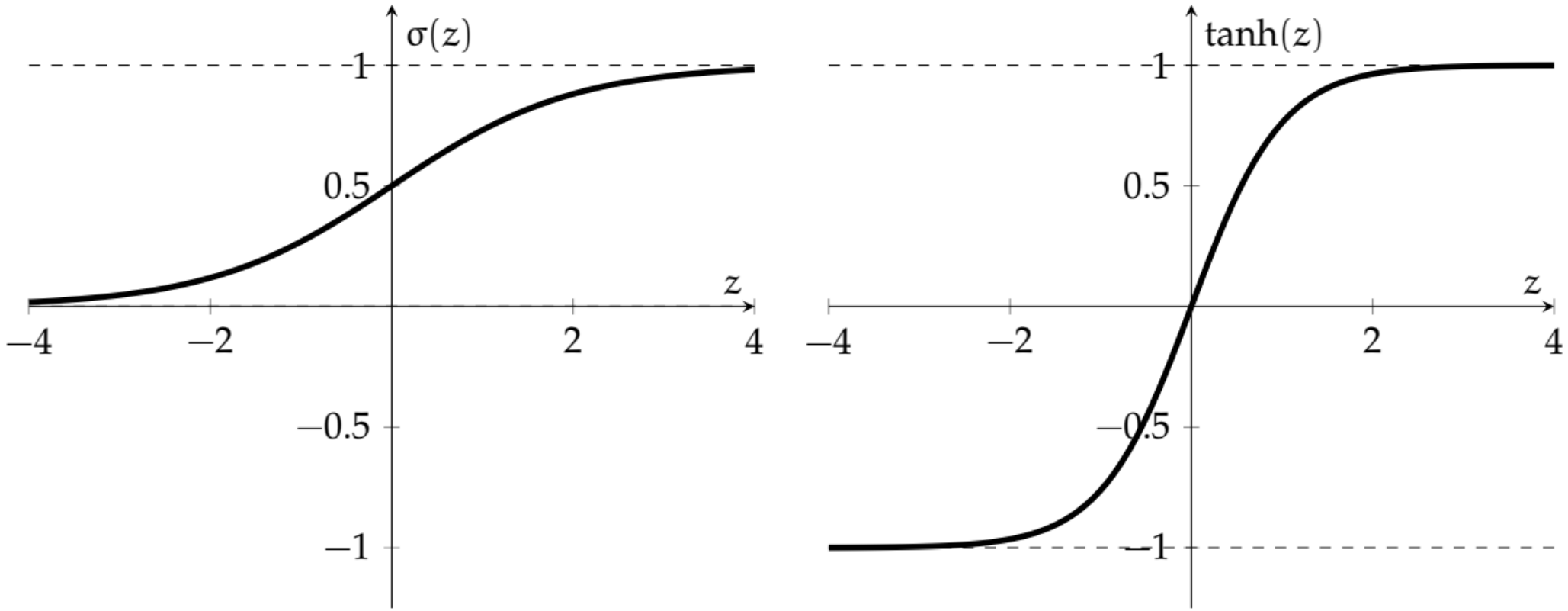


Sigmoid function Also known as a *logistic* function, can be interpreted as probability, because for any value of z the output is in $[0, 1]$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

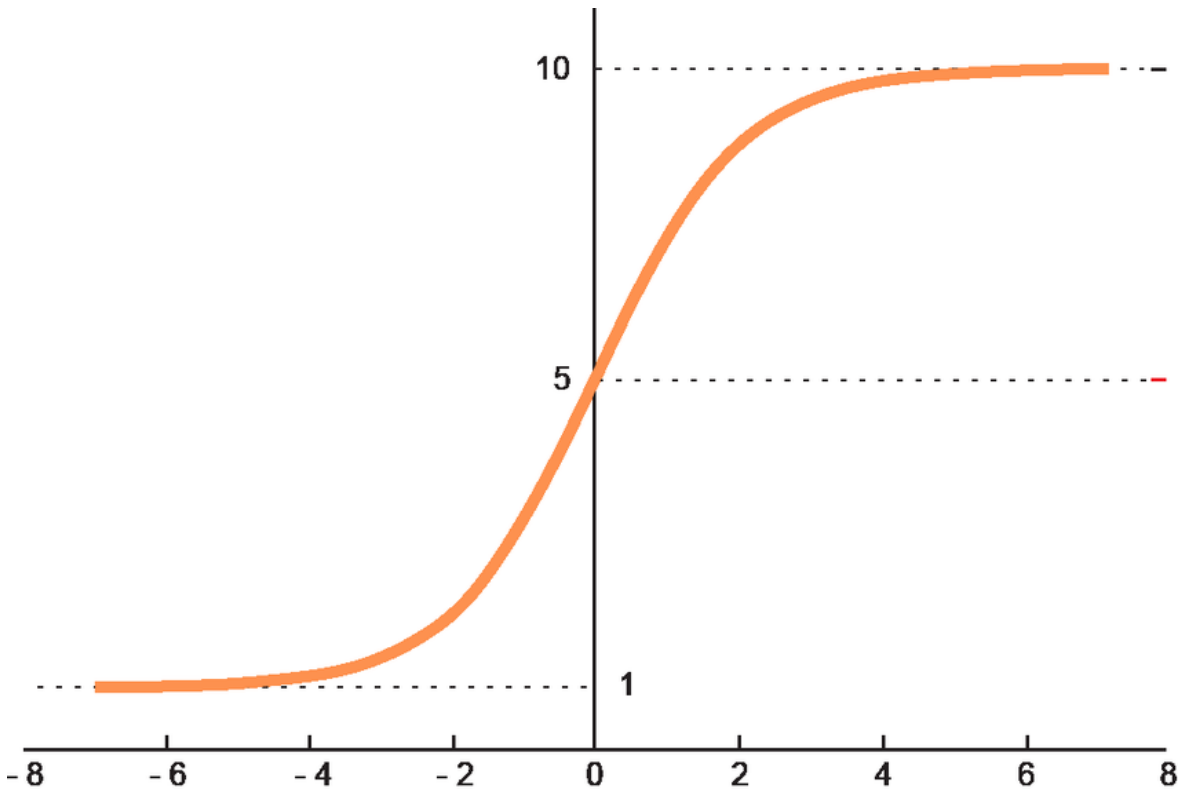
Hyperbolic tangent Always in the range $[-1, 1]$

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Softmax function Takes a whole vector $Z \in \mathbb{R}^n$ and generates as output a vector $A \in [0,1]^n$ with the property that $\sum_{i=1}^{n} A_i = 1$, which means we can interpret it as a probability distribution over n items:

$$\text{softmax}(z) = \begin{bmatrix} \exp(z_1)/\sum_i \exp(z_i) \\ \vdots \\ \exp(z_n)/\sum_i \exp(z_i) \end{bmatrix}$$



Loss functions

- Table of loss functions and activations that make sense

Loss	f^L
squared	linear
hinge	linear
NLL	sigmoid
NLLM	softmax

- **Two-class classification**

$$\mathcal{L}_h(\text{guess}, \text{actual}) = \max(1 - \text{guess} \cdot \text{actual}, 0)$$

- **Multi-class classification and log likelihood**

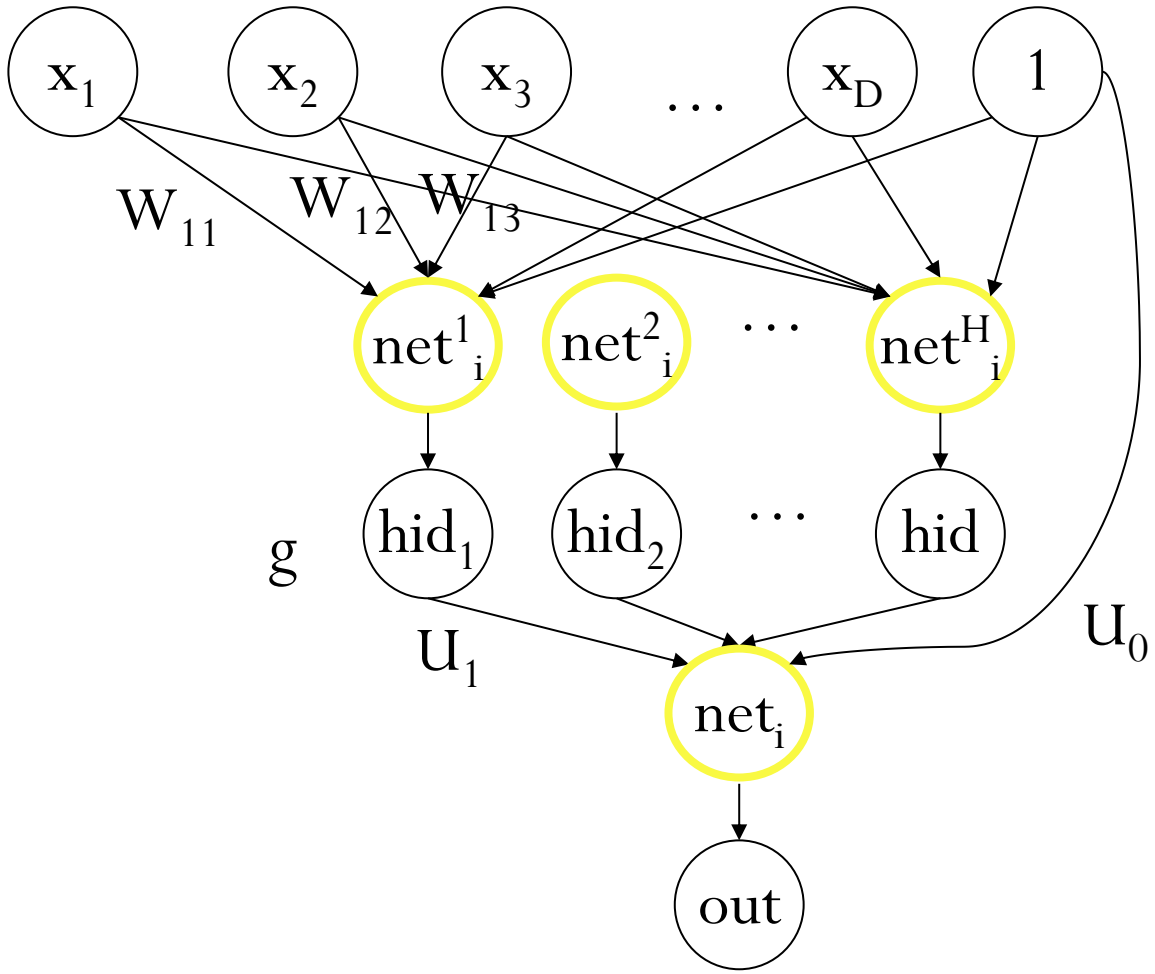
$$\mathcal{L}_{\text{nllm}}(\text{guess}, \text{actual}) = -\sum_{k=1}^{K} \text{actual}_k \cdot \log(\text{guess}_k)$$

We'll call this NLLM for *negative log likelihood multiclass.*

Backpropagation

Backpropagation

- Bryson and Ho (1969, same year) described a training procedure for multilayer networks. Went unnoticed.
- Multiply rediscovered in the 1980s.
- Multilayer Network



Training

Find a set of weights U, W

that minimize

$$\text{sum}_{(\underline{x},\underline{y})}\ \text{sum}_i\ (y_i - \text{out}_i(\underline{x}))^2$$

using gradient descent.

Incremental version (vs. batch):

Move weights a small amount for each training example

Updating Weights Multilayer Network

1. Feed-forward to hidden:

$$net_j = sum_k W_{kj} x_k; hid_j = g(net_j)$$

2. Feed-forward to output:

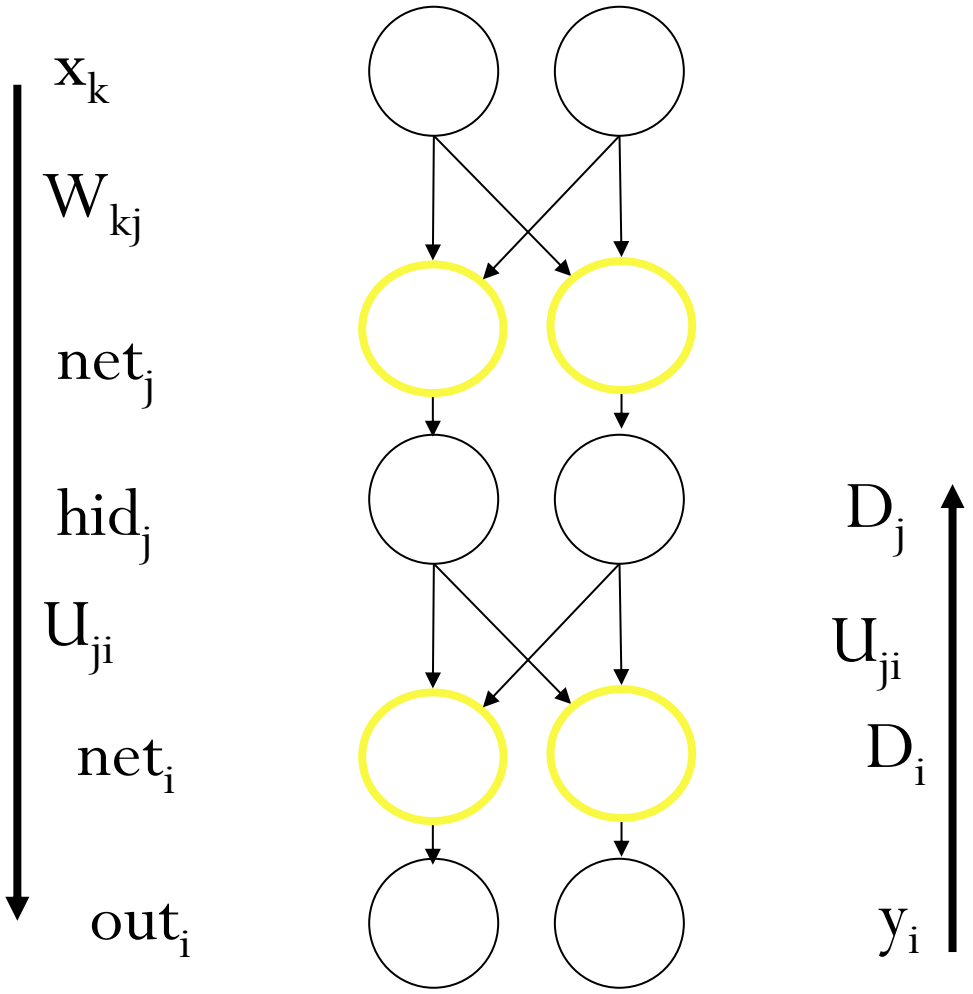
$$net_i = sum_j U_{ji} hid_j; out_i = g(net_i)$$

3. Update output weights:

$$D_i = g'(net_i) (y_i - out_i); U_{ji} \mathrel{+}= \eta\ hid_j\ D_i$$

4. Update hidden weights:

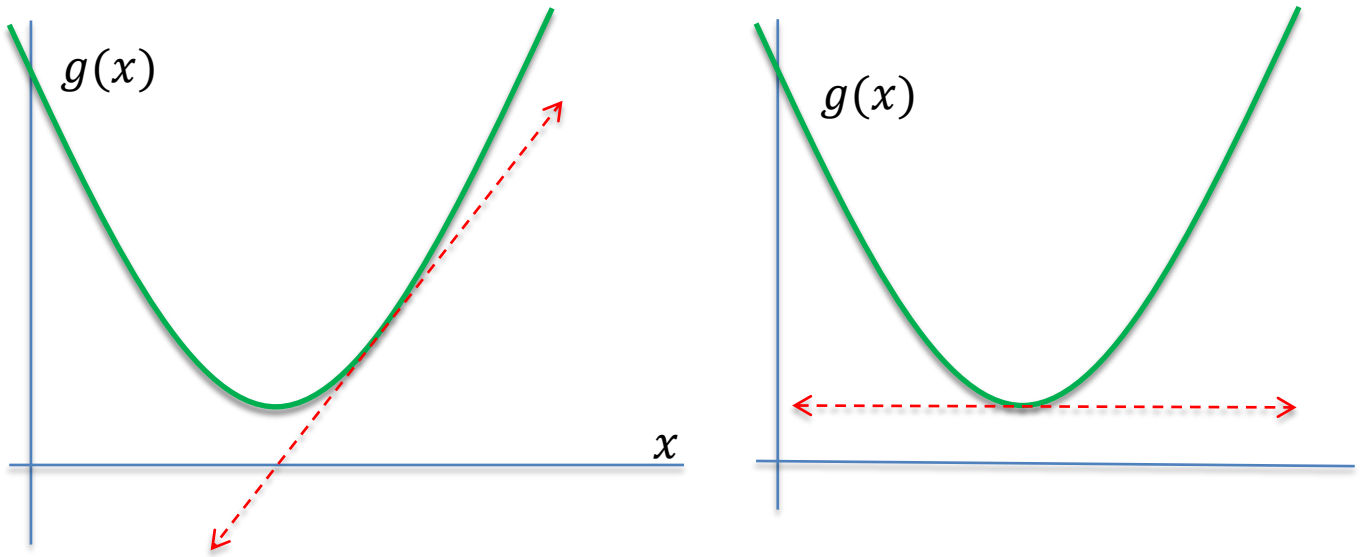
$$D_j = g'(net_j)\ sum_i U_{jj} D_i; W_{kj} \mathrel{+}= \eta\ x_k\ D_j$$

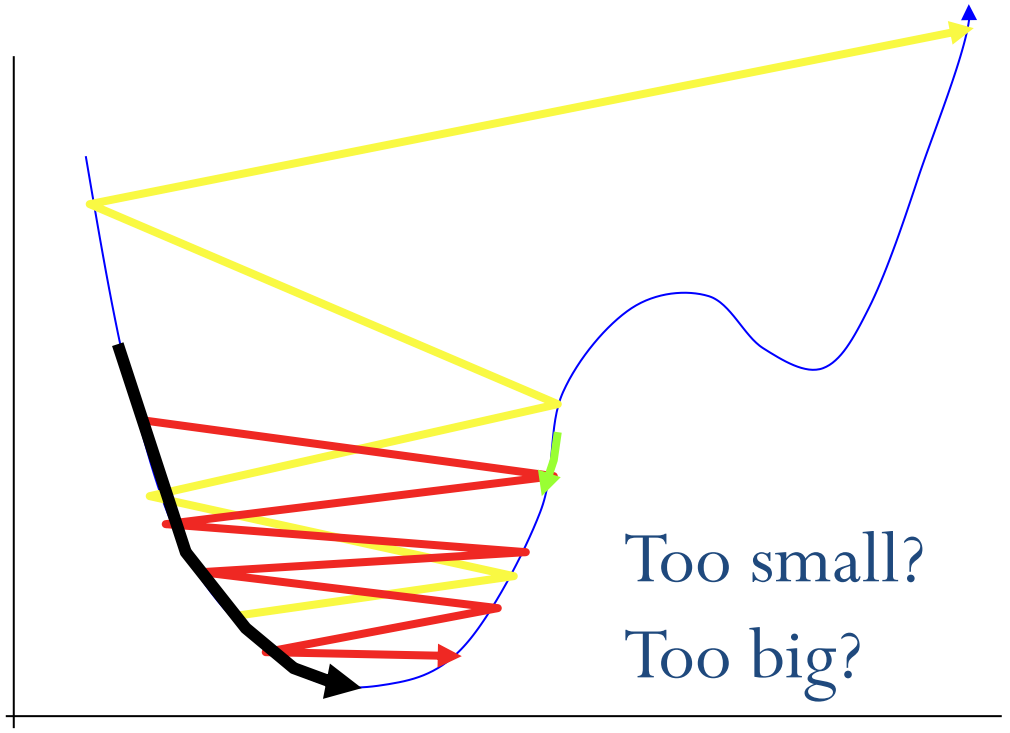


Gradients of Convex Functions

- For a differentiable convex function $g(x)$ its gradients yield linear **approximations**: zero gradient corresponds to a global optimum

- Lots of practical applications.





Issues

Representation Issues

- Any continuous function can be represented by a one hidden layer net with sufficient hidden nodes.
- Any function at all can be represented by a two hidden layer net with a sufficient number of hidden nodes.

Generalization Issues

- Pruning weights: "optimal brain damage"
- Cross validation
- Much, much more to this. Take a class on machine learning.

Back-propagation Issues

- It requires labeled training data.
  - Almost all data is unlabeled.

- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.

  - It can get stuck in poor local optima.

Stochastic Gradient Descent (SGD)

- For multi-layer neural networks, stochastic gradient descent (SGD) is not guaranteed to reach a global optimum.
- Multi-layer networks have non-convex loss surfaces.
- SGD can get stuck in local minima, saddle points, or flat regions.
- Larger models tend to be easier to learn because their units are collectively sufficient to solve the task.
- Bigger models have more representational power and redundancy, so many configurations of weights can achieve good performance.

Stochastic Gradient Descent (SGD)

- Neural network training = non-convex optimization → multiple local optima.
- SGD finds good enough minima, not necessarily global ones.
- Model size and initialization crucially affect training dynamics and results.

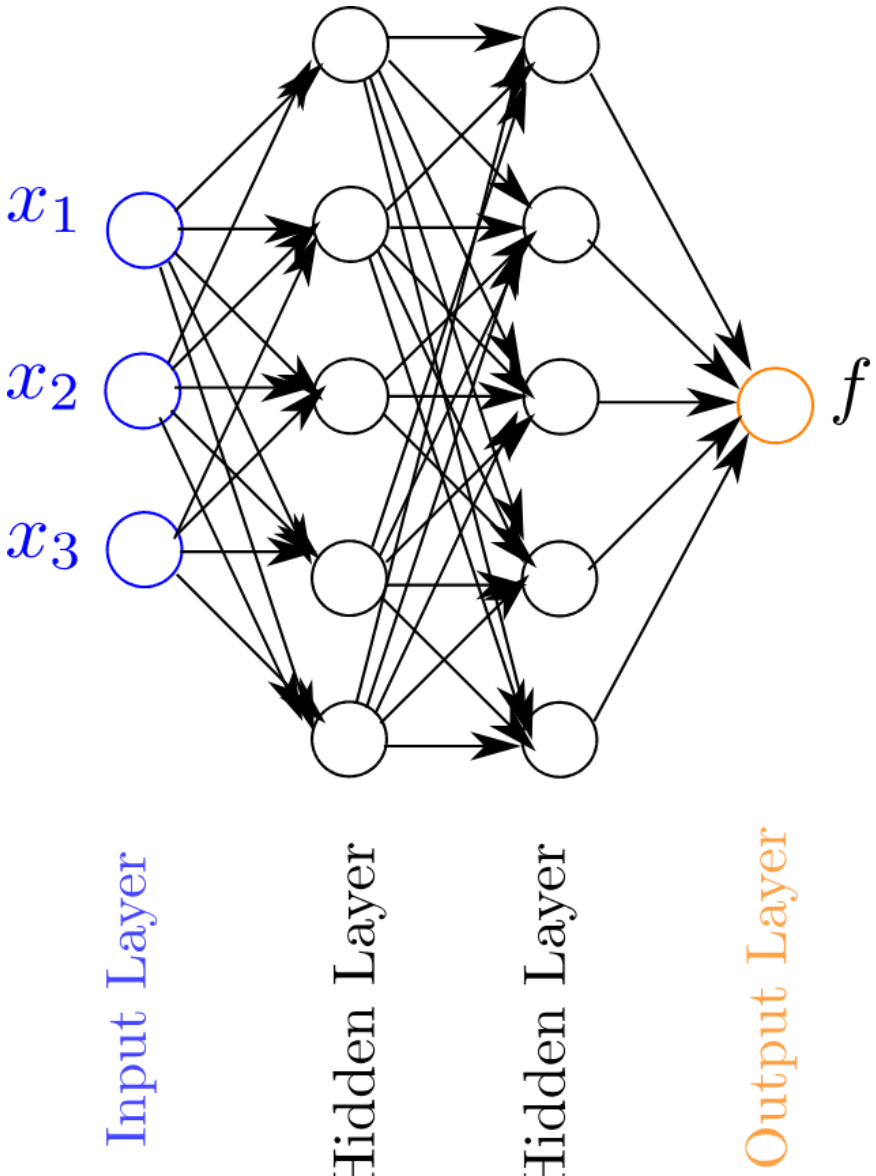
Deep Learning

Deep Neural Network (DNN)

- Feedforward DNN contains
  - input, output, and hidden layers.

- DNN can learn to extract complex and sophisticated features from the raw features presented to them as their input.

- Image recognition, ANNs can extract the features that differentiate a cat from a dog based only on the raw pixel data presented to them from images.



Reason for Deep Learning

- Lots of data:
  - many significant problems can only be solved at scale
- Computational resources (esp. GPUs):
  - platforms/systems that support running deep (machine) learning algorithms at scale
- Large models are easier to train:
  - large models can be successfully estimated with gradient based learning algorithms
- Flexible neural "pieces"
  - common representations, diversity of architectural choices

Hyper-parameters

- Parameters that affect the learning process and the performance of the model
  - Learning rate

- Parameters that control the network architecture
  - number of hidden units/layers

Deep Learning

- Deep learning form a Deep Neural Network (DNN)

- DNN is good for deeper level learning based on classification of data with small error rate.

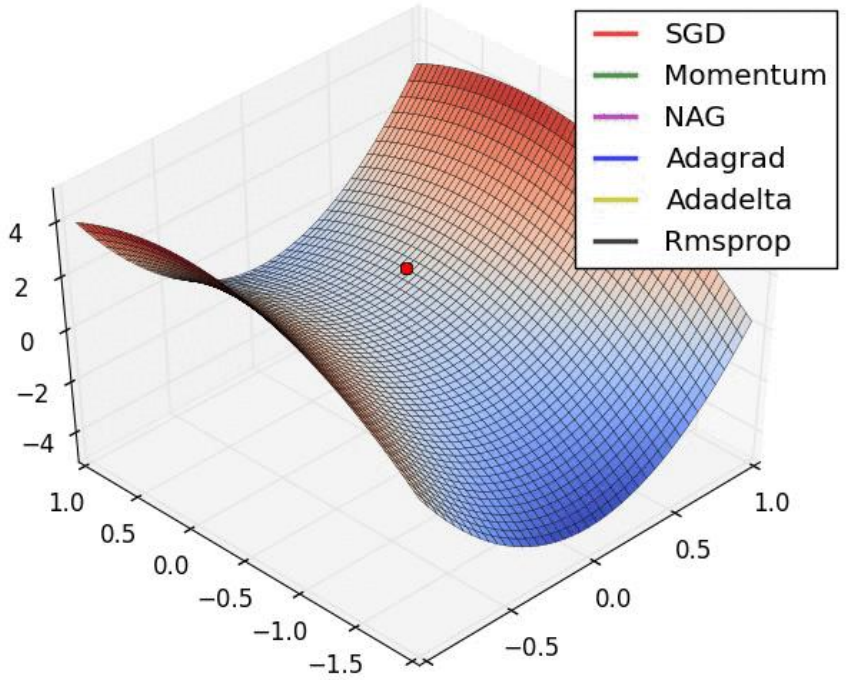
- We used three famous techniques:
  - *Restricted Boltzmann Machines* (*RBM*)
  - *Deep Belief Networks* (*DBN*)
  - *denoising Autoencoders* (*dA*)

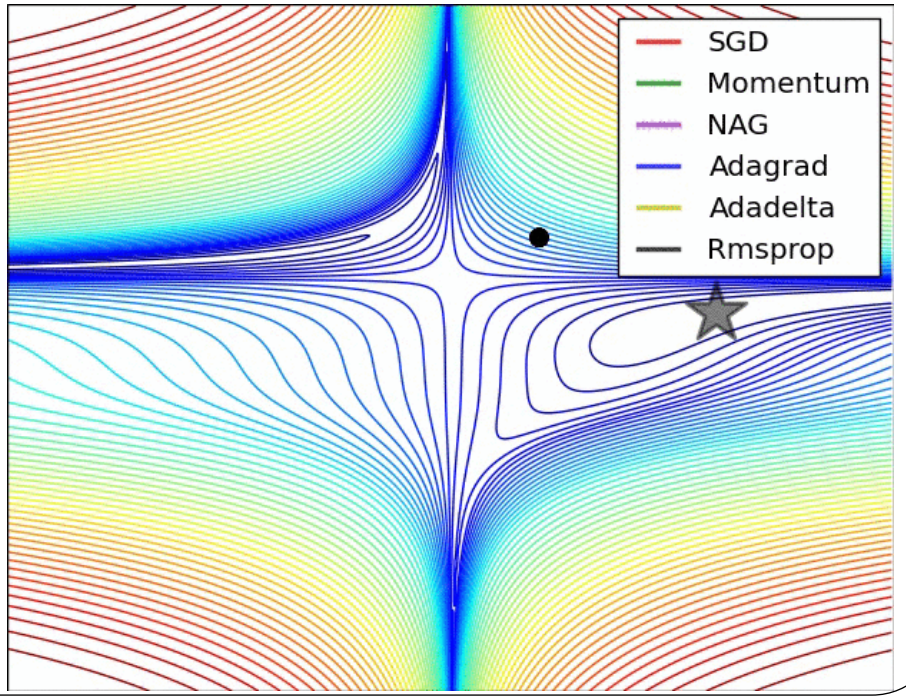
These techniques learn from the input matrices to reconstruct an output matrix.

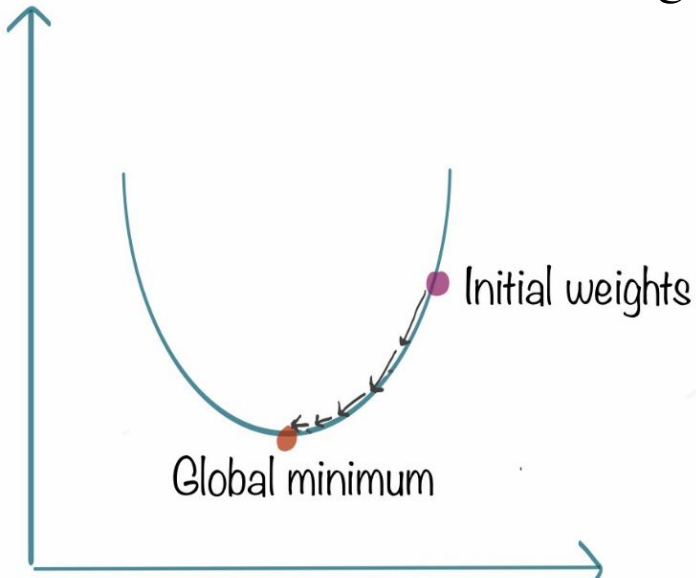
Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.

Maximum Likelihood Estimation

- For the Probabilities and Expectation equations
  - an optimization algorithm finds the value of the parameters (weights) that minimize the error when mapping inputs to outputs.
  - Optimization algorithms affect the accuracy and the speed of the training models of Machine Learning and Deep learning.







https://awesomeopensource.com/project/Jaewan-Yun/optimizer-visualization

MLE for the Deep Learning

- RBM: Focuses on energy function and partition function for MLE.
- DBN: Uses a stack of RBMs for layer-wise pre-training and joint probability modeling.
- DA: Minimizes the expected reconstruction error after a corruption process for robust representation learning.
- The key equations and principles for MLE in RBM, DBN, and DA, providing a clear and concise overview for each model.

The Energy of a joint configuration

- ignoring terms to do with biases

binary state of visible unit i

binary state of hidden unit j

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

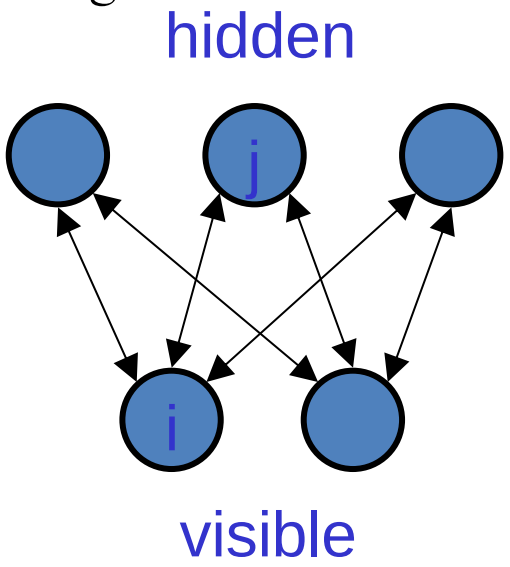
weight between units i and j

$$-\frac{\partial E(v,h)}{\partial w_{ij}} = v_i h_j$$

Restricted Boltzmann Machines (Smolensky, 1986, called them "harmoniums")

- Restrict the connectivity to make learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.

- In an RBM, the hidden units are conditionally independent given the visible states.
  - Quickly get an unbiased sample from the posterior distribution when given a data-vector.
  - This is a big advantage over directed belief nets



MLE for the Restricted Boltzmann Machine (RBM)

Energy Function: $E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i W_{ij} h_j$

Probability Distribution: $P(v,h) = \frac{e^{-E(v,h)}}{Z}$

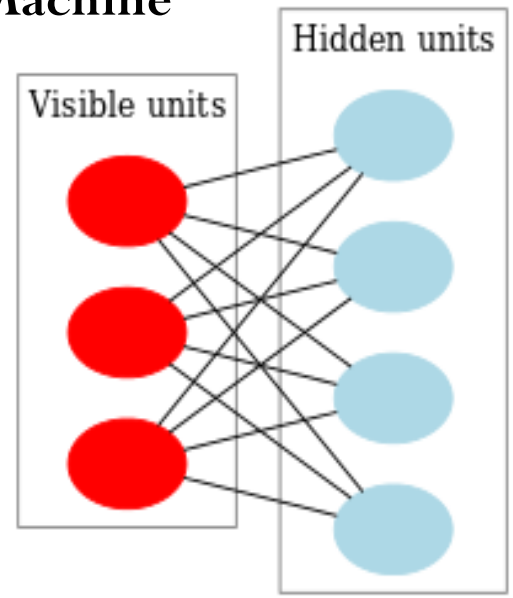
where $Z = \sum_{v,h} e^{-E(v,h)}$ is the partition function.

Maximum Likelihood Estimation:

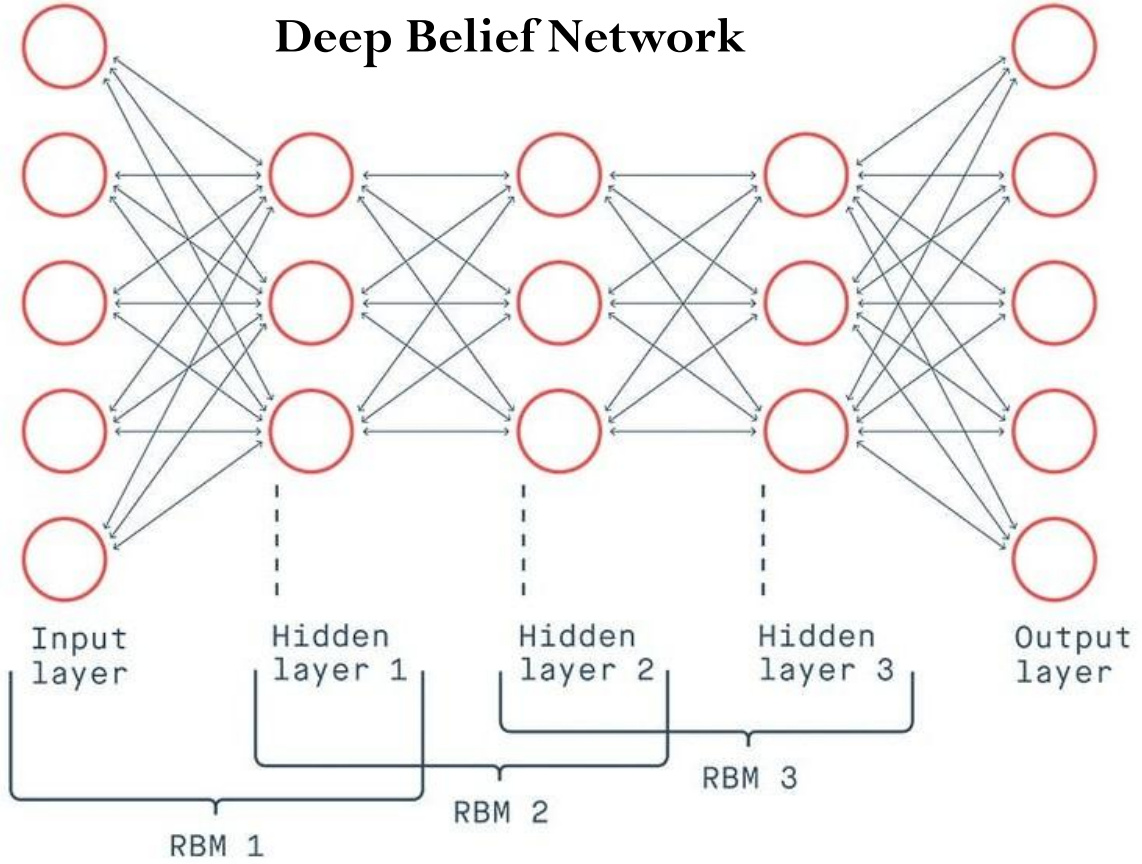
$$\frac{\partial \log P(v)}{\partial \theta} = \langle \frac{\partial E(v,h)}{\partial \theta} \rangle_{\text{data}} - \langle \frac{\partial E(v,h)}{\partial \theta} \rangle_{\text{model}}$$

Restricted Boltzmann Machine and Deep Belief Network

Restricted Boltzmann Machine



Deep Belief Network



Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann Machines for Collaborative Filtering." Proceedings of the 24th International Conference on Machine learning. 2007.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for Deep Belief Nets." Neural computation 18.7 (2006): 1527-1554.

MLE for the Deep Belief Network (DBN)

- Layer-wise Pre-training:
  - Each layer is trained as an RBM.
- Joint Probability of Visible and Hidden Units:

$$P(v, h^{(1)}, \ldots, h^{(L)}) = P(v|h^{(1)}) \prod_{l=1}^{L-1} P(h^{(l)}|h^{(l+1)}) P(h^{(L)})$$

- **Maximum Likelihood Estimation**
  - Layer-wise MLE similar to RBM

$$\frac{\partial \log P(v)}{\partial \theta} = \langle \frac{\partial E(v,h)}{\partial \theta} \rangle_{\text{data}} - \langle \frac{\partial E(v,h)}{\partial \theta} \rangle_{\text{model}}$$

Analogies: Autoencoders vs. Zip/Unzip

The Analogy: Compression and Decompression

- **Encoder (Zip):** Takes input data and compresses it into a smaller, dense representation (the **latent space** or "bottleneck").
  - Analogous to a zip program creating a smaller archive file.

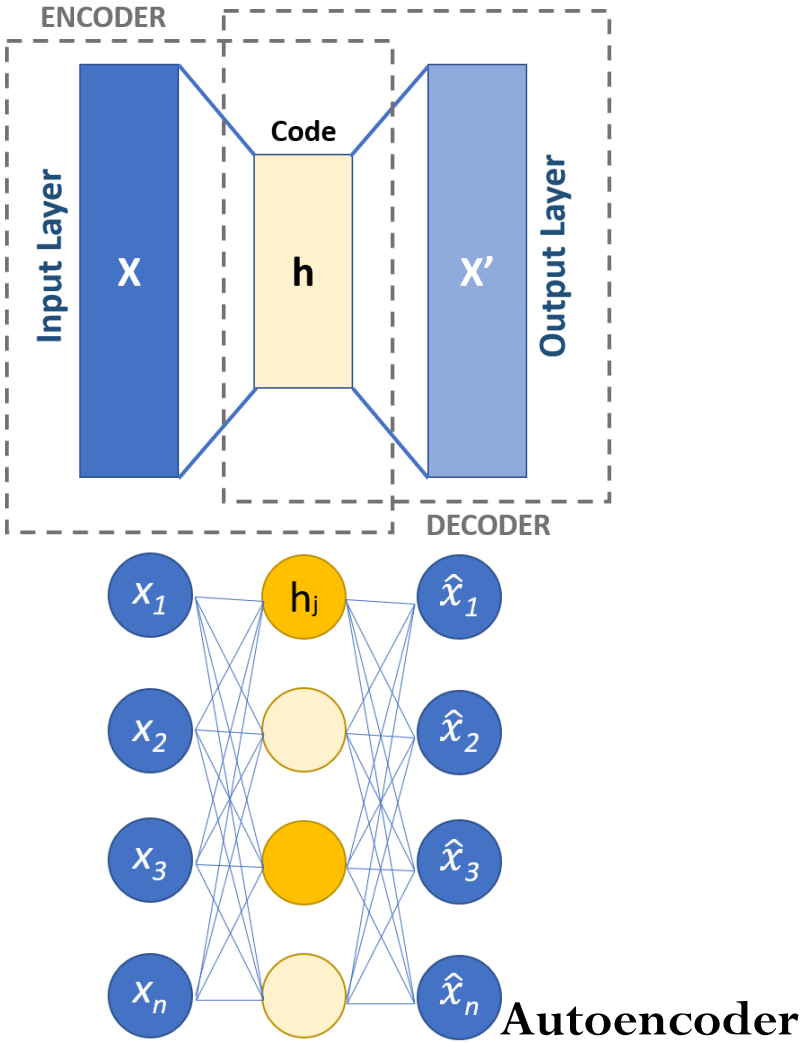
- **Decoder (Unzip):** Takes the compressed representation and reconstructs the data to its original size.
  - Analogous to an unzip program expanding an archive back into a usable file.

- **Shared Goal:** Efficient representation and data reduction.

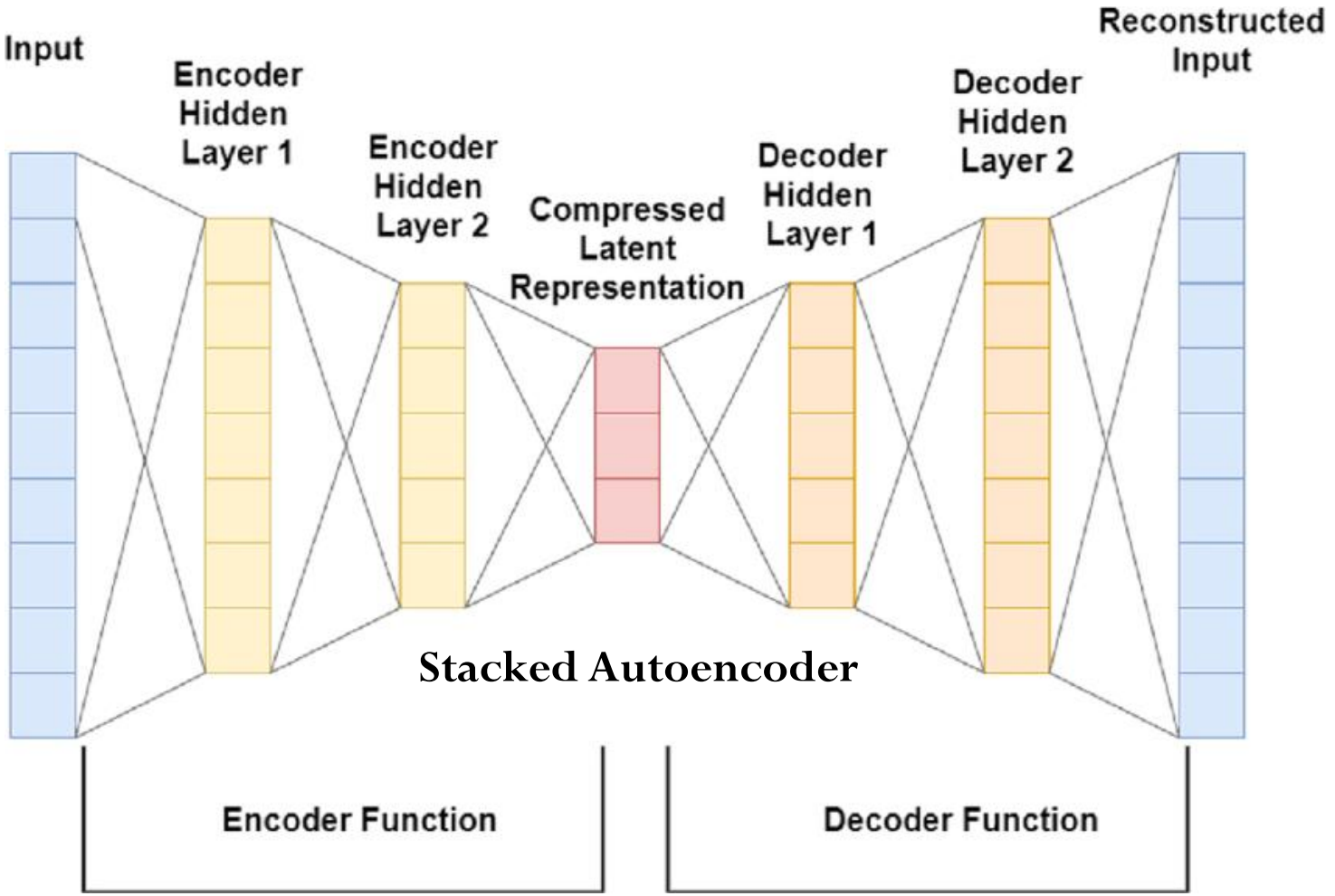
Key Differences: Autoencoders vs. Zip/Unzip

Feature	Autoencoders	Zip/Unzip (e.g., Gzip)
Compression Type	**Lossy:** Reconstructed data is a close approximation, not an exact copy.	**Lossless:** Original data is perfectly restored.
Method	**Learned:** Discovers a data-specific, non-linear function through training.	**Fixed Algorithm:** Uses general-purpose methods (e.g., Huffman encoding) to remove redundancy.
Generality	**Data-specific:** Trained on faces, it's useless for compressing trees.	**General-purpose:** Can compress any type of file effectively.
Primary Use	Feature extraction, anomaly detection, denoising, data generation.	Efficient storage and transmission of existing files.

Autoencoder and Stacked Autoencoder



Autoencoder



Stacked Autoencoder

Pascal Vincent, et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." Journal of Machine Learning Research 11.12 (2010).

MLE for the Denoising Autoencoder (DA)

- Reconstruction Error:

$$L(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2$$

where x is the original input and $\hat{x}$ is the reconstructed input.

- **Corruption Process**:

$$\tilde{x} = q_D(\tilde{x}|x)$$

where $\tilde{x}$ is the corrupted version of x

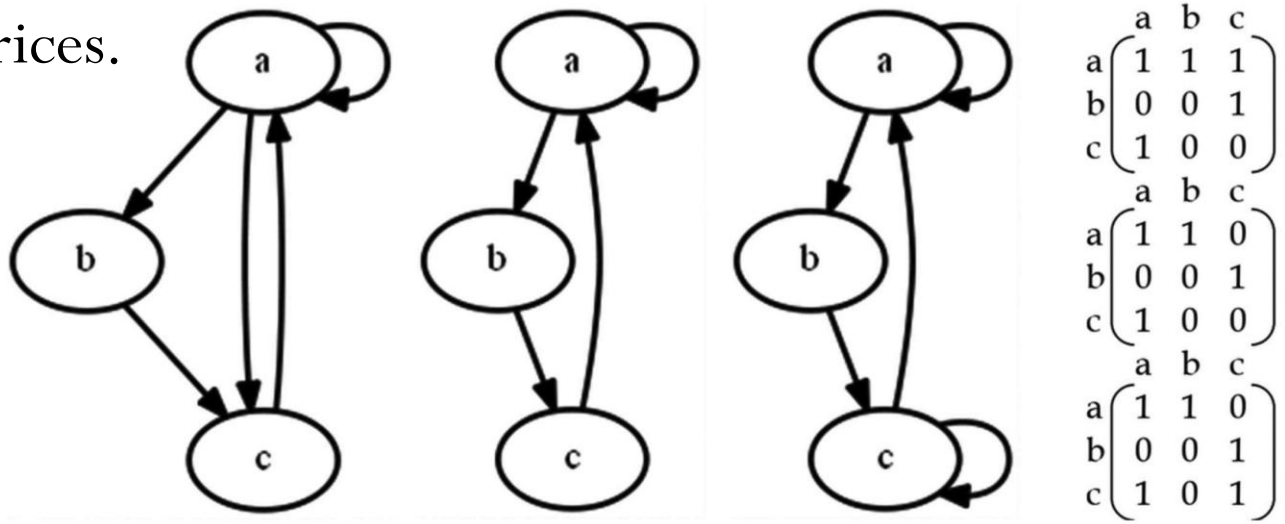
- **Maximum Likelihood Estimation**
- Minimize the expected reconstruction error:

$$\mathcal{L} = \mathbb{E}_{q_D(\tilde{x}|x)}[L(x, \hat{x})]$$

System Neural Network for Evolution Learning

System Graph Evolution Learning

- Graph evolution example, when it is evolved in three incremental states with three similar temporal patterns and matrices.



- To do System Graph Evolution Learning, we extended 3 deep learning (RBM, DBN, and dA) as three different deep evolution learning approaches.
- For each evolving system, we used the 3 *deep evolution learning* approaches separately, which resulted in three different ***System Neural Networks***.

Remodeling of Equations

$$\mathrm{g}(\mathbf{ER}) = P(\boldsymbol{h}|\mathbf{EDSM}) = \prod_{l=1}^{L} P(\boldsymbol{h_l}|\mathbf{EDSM}).$$

Restricted Boltzmann Machines (RBM)

$$= \sum_i (a_i\, ec_{jk}) + \sum_l (b_l\, h_l) + \sum_i \sum_l (ec_{jk}\, w_{il}\, h_l)$$

$$\frac{d\log P(\boldsymbol{h}|\mathbf{EDSM})}{d\boldsymbol{W}_{jkl}} \approx < ec_{jk} h_l >_{\text{data}} - < ec_{jk} h_l >_{\text{reconstruct}}$$

ec_{jk} means entity connection between jth row and kth column of Evolving Design Structured Matrix (EDSM)

Deep Belief Networks (DBN)

$$\mathrm{g}(\mathbf{ER}) = P\left(\mathbf{EDSM}, h^1,\ h^2 \ldots h^L\right) = \left(\prod_{l=0}^{L-2} P(h^l|h^{l+1})\right) P\left(h^{L-1}, h^L\right)$$

Autoencoder

$$\mathrm{RE} = -\log P\left(\mathrm{EDSM}|c\left(\boldsymbol{EDSM}\right)\right)$$

$$\mathrm{RE} = -\sum_i \mathrm{ec}_{jk} \log f_i\left(c\left(\boldsymbol{EDSM}\right)\right) + (1 - \mathrm{ec}_{jk}) \log\left(1 - f_i\left(c\left(\boldsymbol{EDSM}\right)\right)\right)$$

Denoising Autoencoder dA

$$RE = -\log\left(\mathrm{g}\left(\mathbf{ER}\right)\right) = -\log P(\mathbf{EDSM}|c(\widetilde{\mathbf{EDSM}}))$$

System Structure Learning (SSL) that uses **Deep Evolution Learner** (DEL)

Evolution Representor

Evolving Design Structure Matrix (EDSM), where $vector_i$ represents i^{th} state

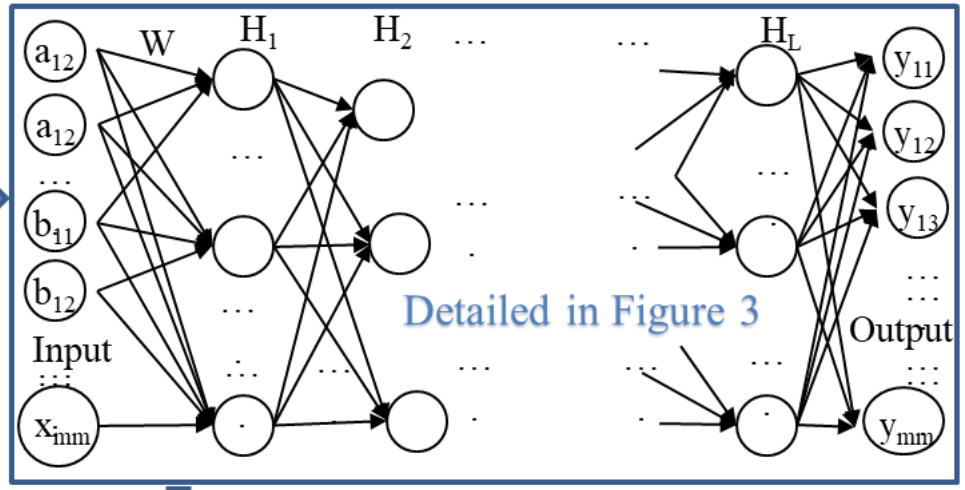
$vector_1$ $[\{a_{11}.. a_{jk} \dots a_{1m}\}, \quad \{a_{21}.. a_{jk} \dots a_{2m}\} \dots \quad \{a_{m1}.. a_{jk} \dots a_{mm}\}]$

$vector_2$ $[\{b_{11}.. b_{jk} \dots b_{1m}\}, \quad (b_{21}.. b_{jk} \dots b_{2m}) \dots \quad (b_{m1},.. b_{jk} \dots b_{mm})]$

$vector_N$ $[(x_{11}.. x_{jk} \dots x_{1m}), \quad (x_{21}.. x_{jk} \dots x_{2m}) \dots \quad (x_{m1}.. x_{jk} \dots x_{mm})]$

DEL

Deep System Neural Network (Deep SysNN)



DEL

0	0	….	0

zero_vector (m × m)

Conversion

0	0	….	0
….	….	….	….
….	….	….	….
….	….	….	….
0	….	….	0

Zero Matrix of size (m × m)

y_{11}	y_{12}	….	y_{mm}

output_vector (m × m)

Conversion

y_{11} , y_{12} …	y_{1k}	…	y_{1m}
….	….	….	….
y_{j1} , y_{j2} , …	y_{jk}	…	y_{jm}
….	….	….	….
y_{m1} , y_{m2} …	y_{mk}	…	y_{mm}

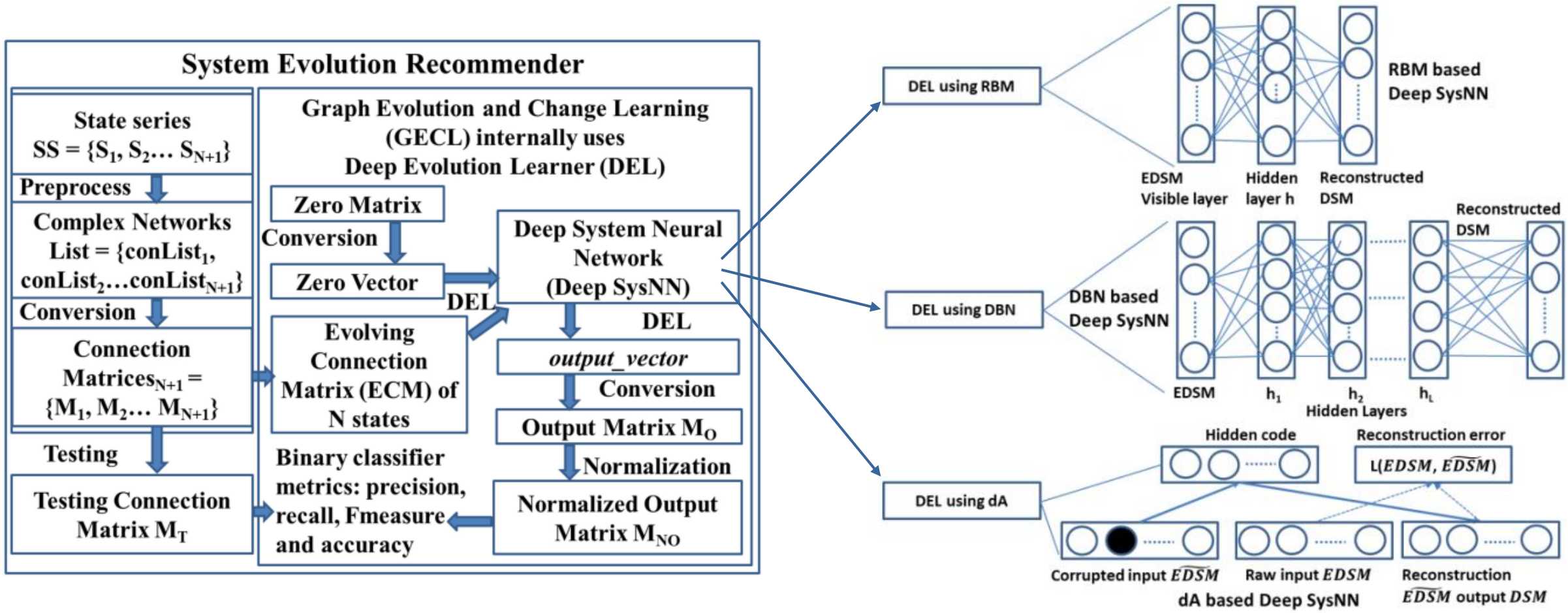
Output Matrix M_O (m × m)

Normalization

n_{11} , n_{12} …	n_{1k}	…	n_{1m}
….	….	….	….
n_{j1} , n_{j2} , …	n_{jk}	…	n_{jm}
….	….	….	….
n_{m1} , n_{m2} …	n_{mk}	…	n_{mm}

Symbol	Description
SS	($vector_1$, $vector_2$ …. $vector_N$)
ER	EDSM = f(SS)
$\mathbf{a_{jk}}$ $\mathbf{b_{jk}}$ $\mathbf{x_{jk}}$	Elements of EDSM
input	*zero_vector*
g(**ER**)	H_1, H_2, …. H_L
Y	*output_vector* (m × m)
$\mathbf{M_{NO}}$	binary output matrix (m × m)

System Graph Evolution Learning



System Structure Learning

$$\mathbf{ER} = \mathrm{f}(\mathbf{SS})$$

$$\mathbf{ER} = \mathrm{f}(\{a_{11}..\, a_{jk}..\, a_{mm}\}, \{b_{11}..\, b_{mm}\}\ \ldots\ \{x_{11}..\, x_{jk}..\, x_{mm}\})$$

$$\mathrm{EDSM} = \mathrm{f}(\mathbf{SS}) = \boldsymbol{evolutionRepresentor}\,(N_{DSMs})$$

Algorithm ***SSL***(*N_DSMs*)

Initialize a zero matrix as M_Z
zero_vector = ***matrixToRowVector***(M_Z)
EDSM = ***evolutionRepresentor***(*N_DSMs*)
output_vector = ***deep_evolution_learner*** (*EDSM*, *zero_vector*)
M_O = ***rowVectorToMatrix***(*output_vector*)
M_{NO} = ***normalize***(M_O)
Return M_{NO}

Algorithm ***evolutionRepresentor***(*N_DSMs*)

For each *DSM_i* in *N_DSMs* where i ∈ state number
vector_i = ***matrixToRowVector***(*DSM_i*)

$$EDSM = \left\{ \begin{array}{c} vector\_i \\ + \\ EDSM \end{array} \right\}$$

End for
Return *EDSM*

Algorithm ***deep_evolution_learner*** (*EDSM*, *zero_vector*)

Initialize List *trainParameters*< LR, Ep, L >
Initialize a matrix *Deep_SysNN*
Deep_SysNN = ***delTrain***(*EDSM*, *trainParameters*)
// three variant of Deep SysNN based on Equation 4 to 10
output_vector = ***delReconstruct***(*Deep_SysNN*, *zero_vector*)
Return *output_vector*

$$\mathbf{Y} = \mathrm{g}(\mathbf{ER}) = \mathrm{g}(\mathrm{f}(\mathbf{SS}))$$

$$\boldsymbol{Y} = (\{y_{11}..\, y_{jk}..\, y_{1\,m}\}, \{y_{21}..\, y_{2\,m}\} \ldots \{y_{m1}..\, y_{jk}\ \ldots y_{mm}\})$$

Experiments on Evolving Systems

INFORMATION ABOUT IMBALANCED DATASET USED

Evolving Systems	Testing matrix (M_T)	Number of 1s	Number of 0s
Hadoop HDFS	Version 2.7.2	2938	9787703
List of Bible Translation	20th Century	46	579
List of Multi-sport Events	201 i.e. 2010-2017 decade	8	188
Frequent Market Basket	1211 i.e. Dec. 2011	617	13307
Positive sentiment of movie genres	201 i.e. 2010-2019 decade	47	578
Negative sentiment of movie genres	201 i.e. 2010-2019 decade	177	1848

Binary Metrics

# denotes "count of"

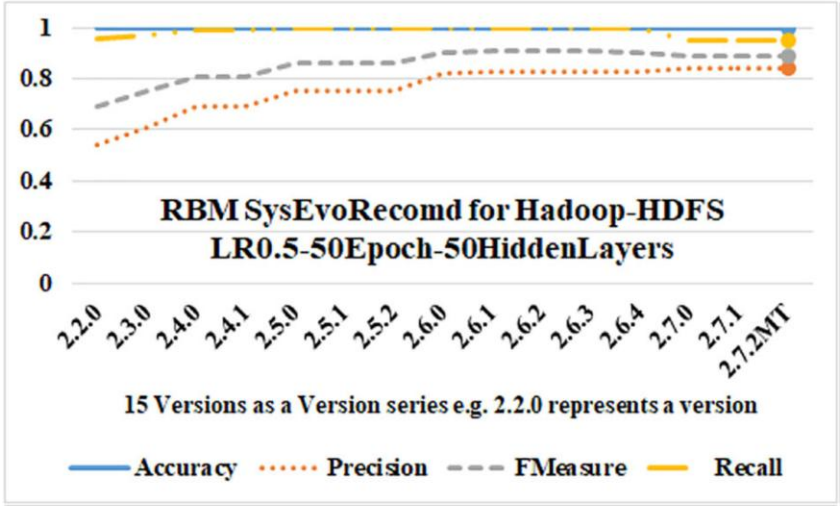
$$\text{accuracy} = \frac{\left(\#\begin{array}{c}\text{correctly recommended connections +}\\ \text{correctly recommended no connections}\end{array}\right)}{\#\text{all possible entity connections i.e. matrix size}}$$

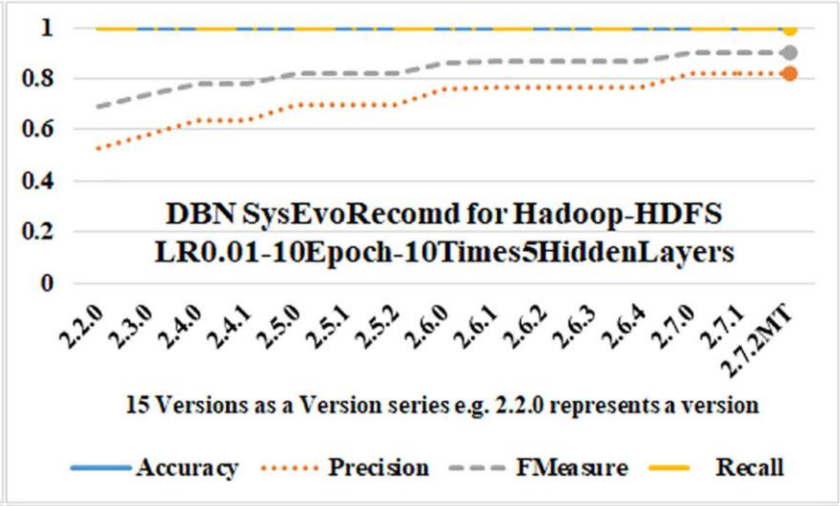
$$\text{precision} = \frac{\#\text{correctly recommended connections}}{\#\text{all connections recommended by the tool}}$$

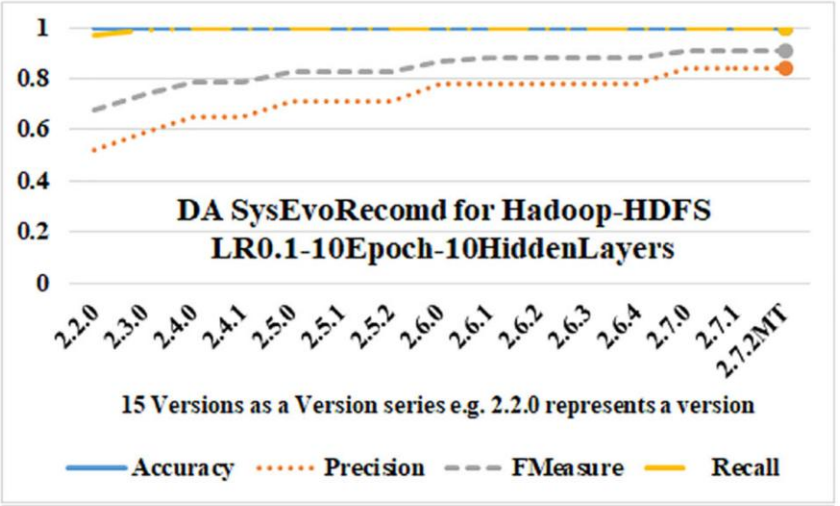
$$\text{recall} = \frac{\#\text{correctly recommended connections}}{\left(\#\begin{array}{c}\text{correctly recomended connections + incorrectly}\\ \text{recommended connections as no connections}\end{array}\right)}$$

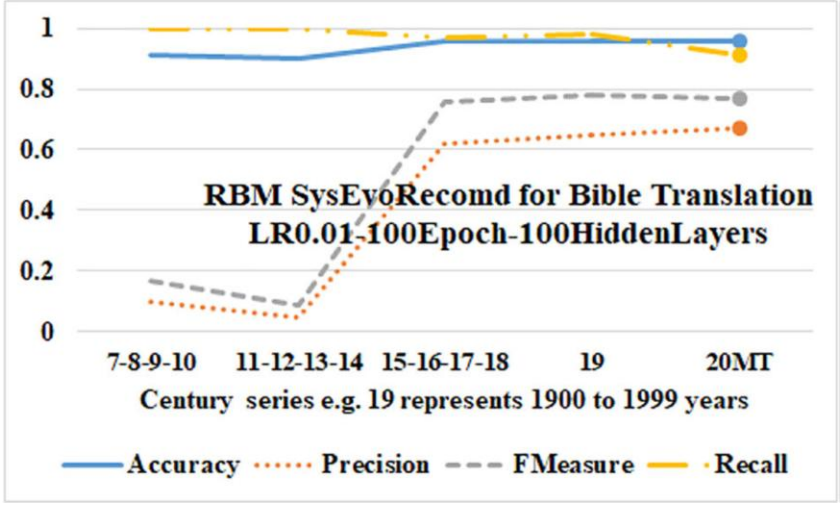
$$F\text{-}Measure = 2 \times \frac{precision \times recall}{precision + recall}$$

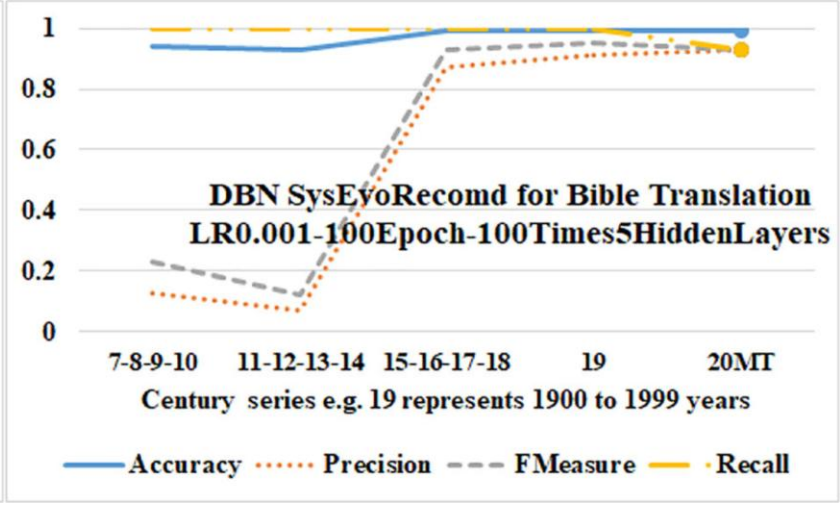
System Evolution Recommendations

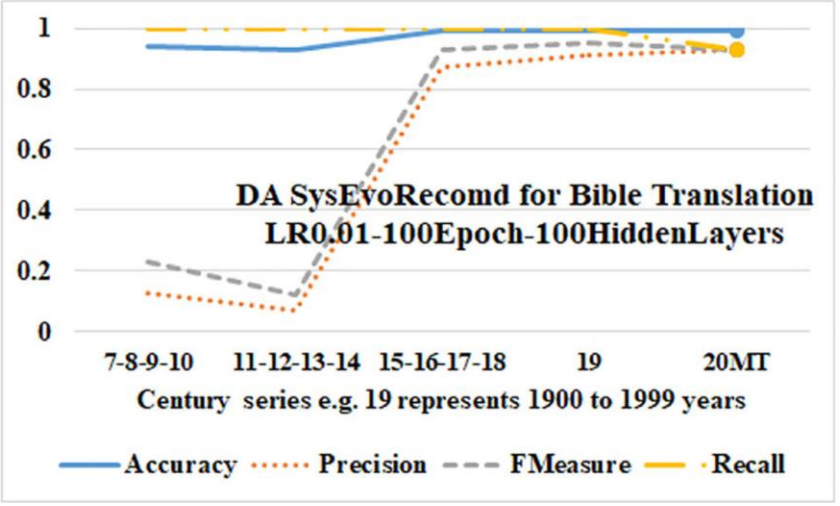




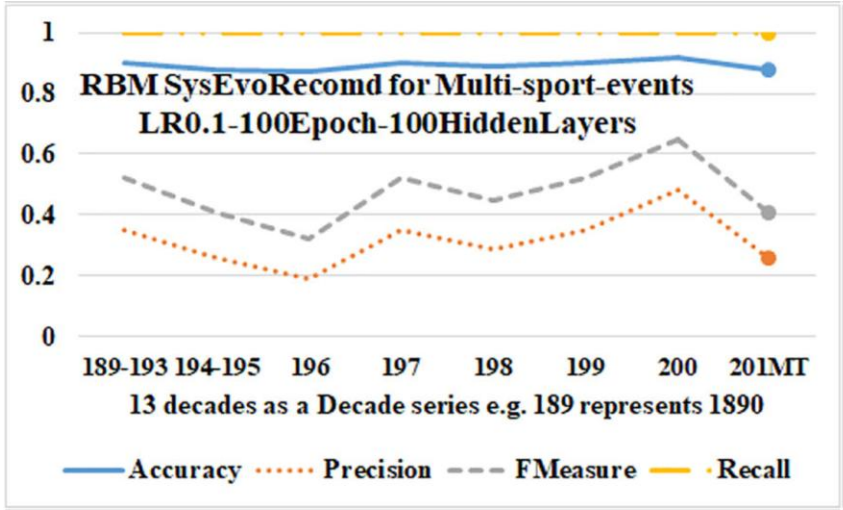


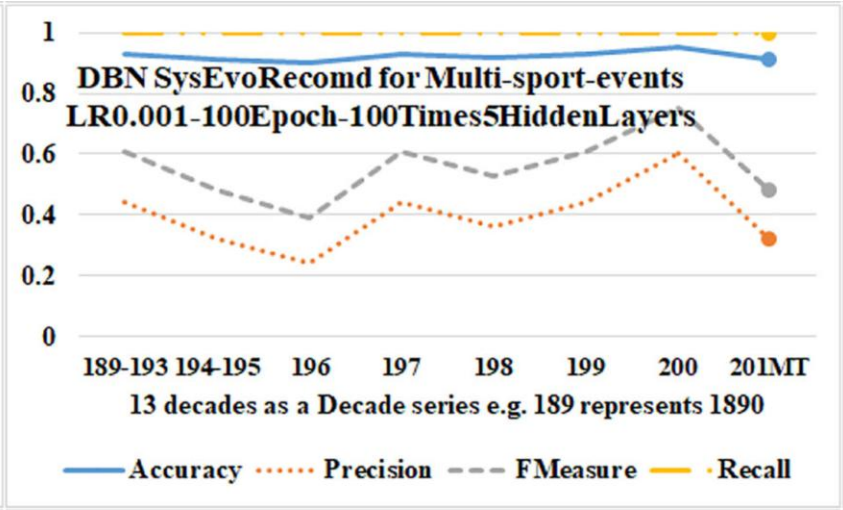


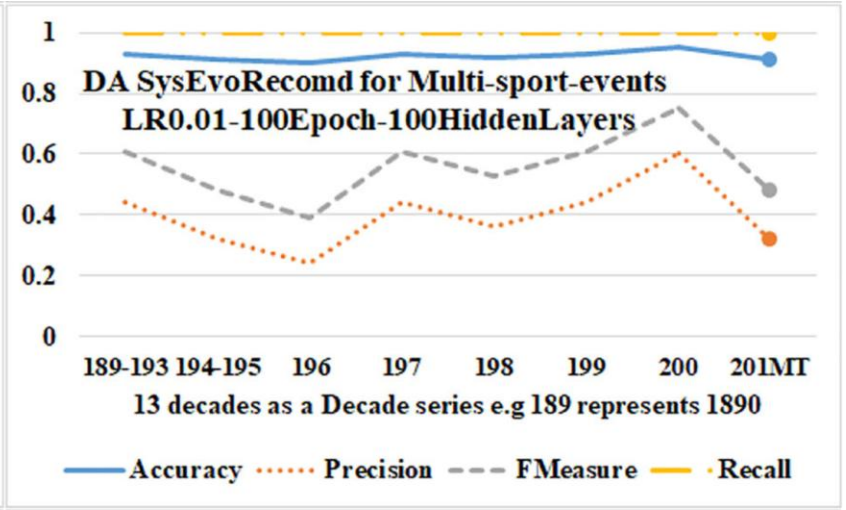


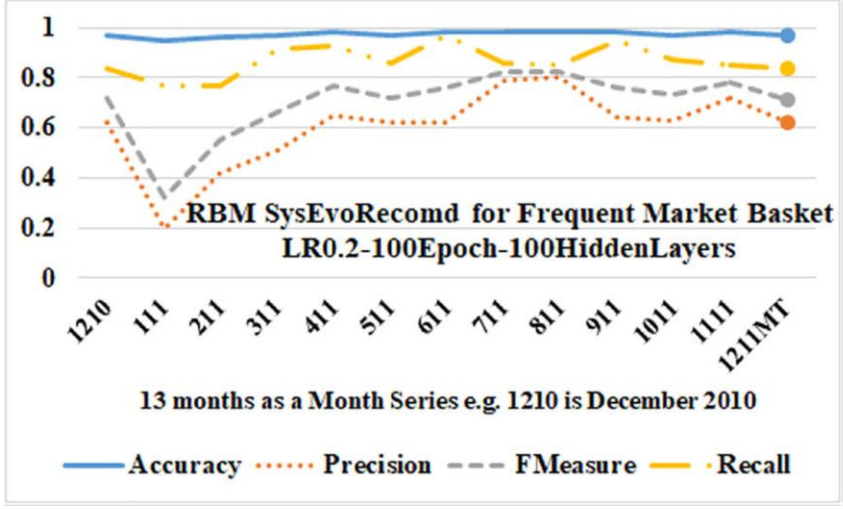


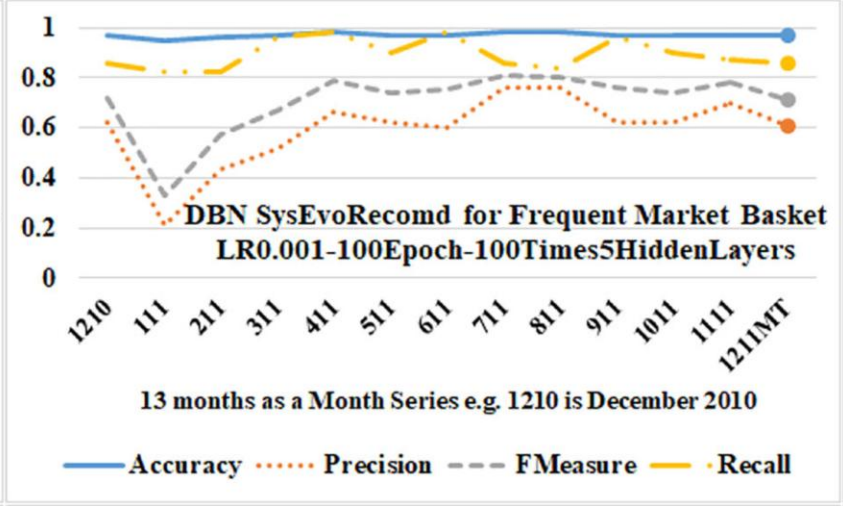
Recommendation

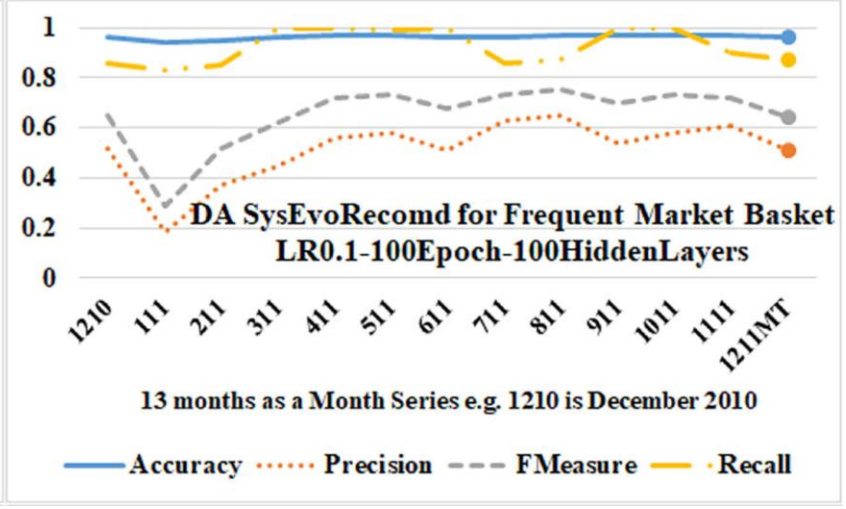




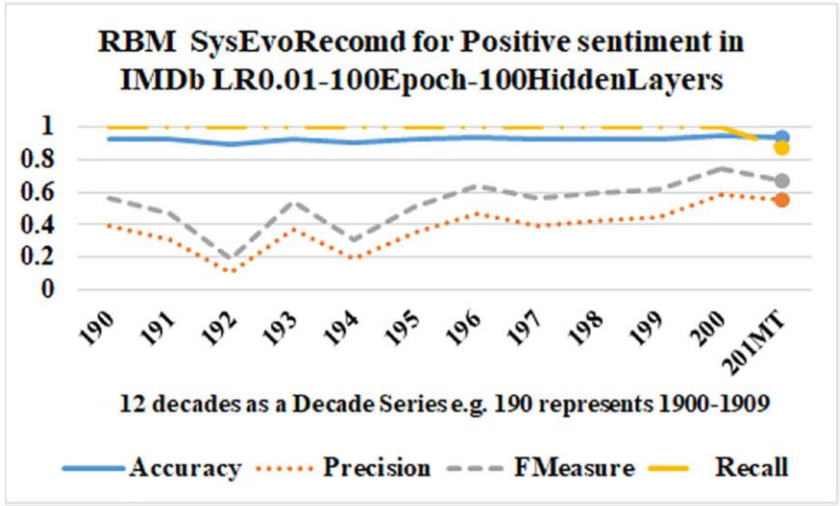


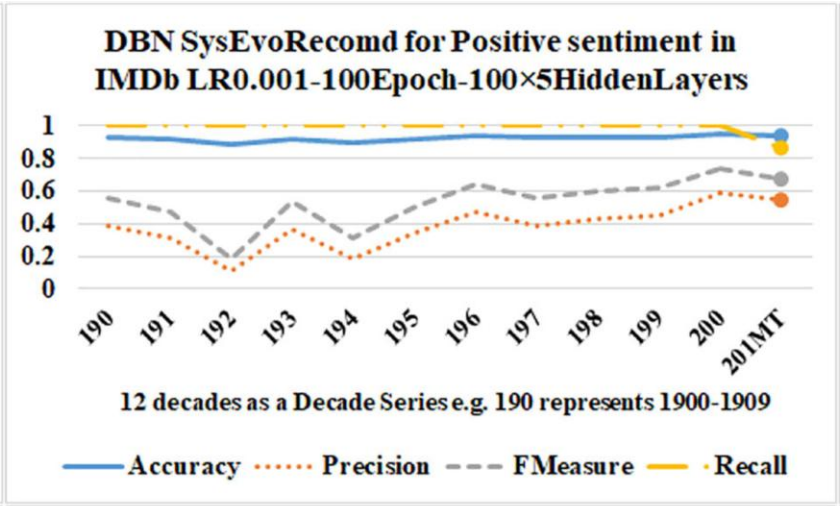


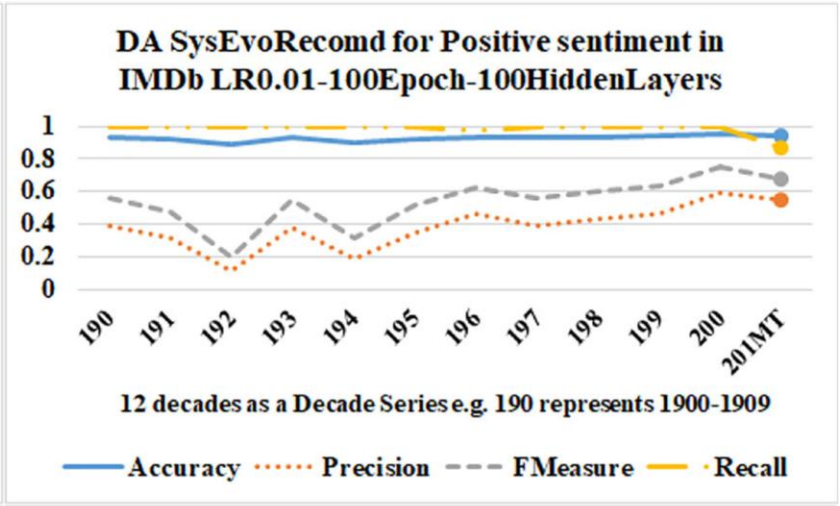


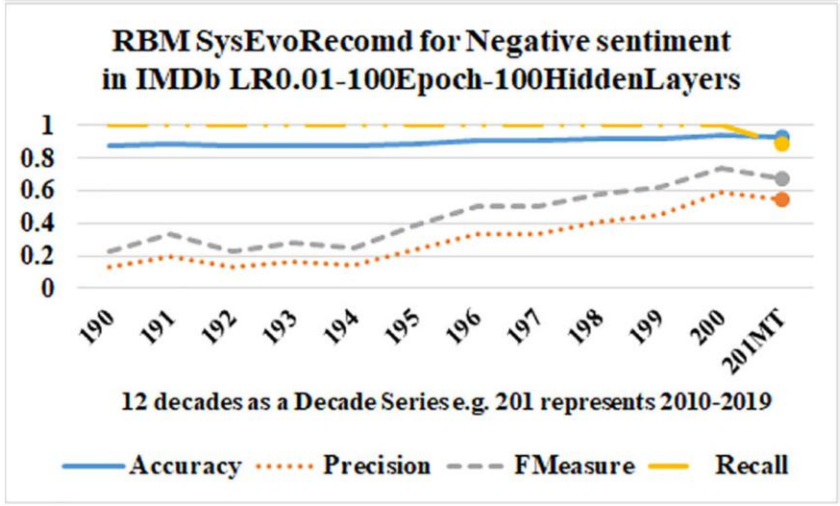


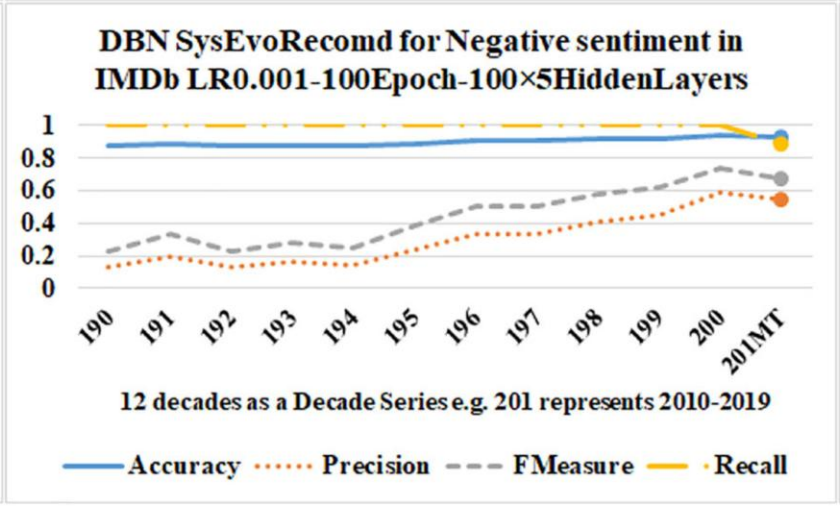
Recommendation

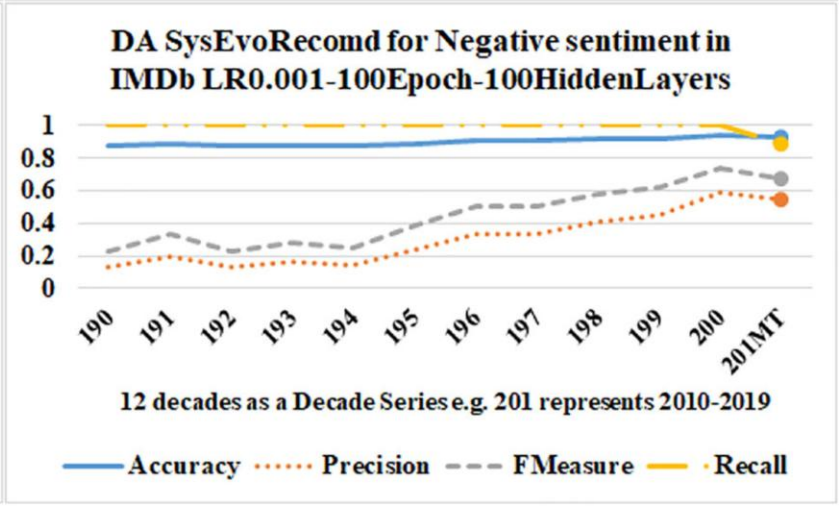








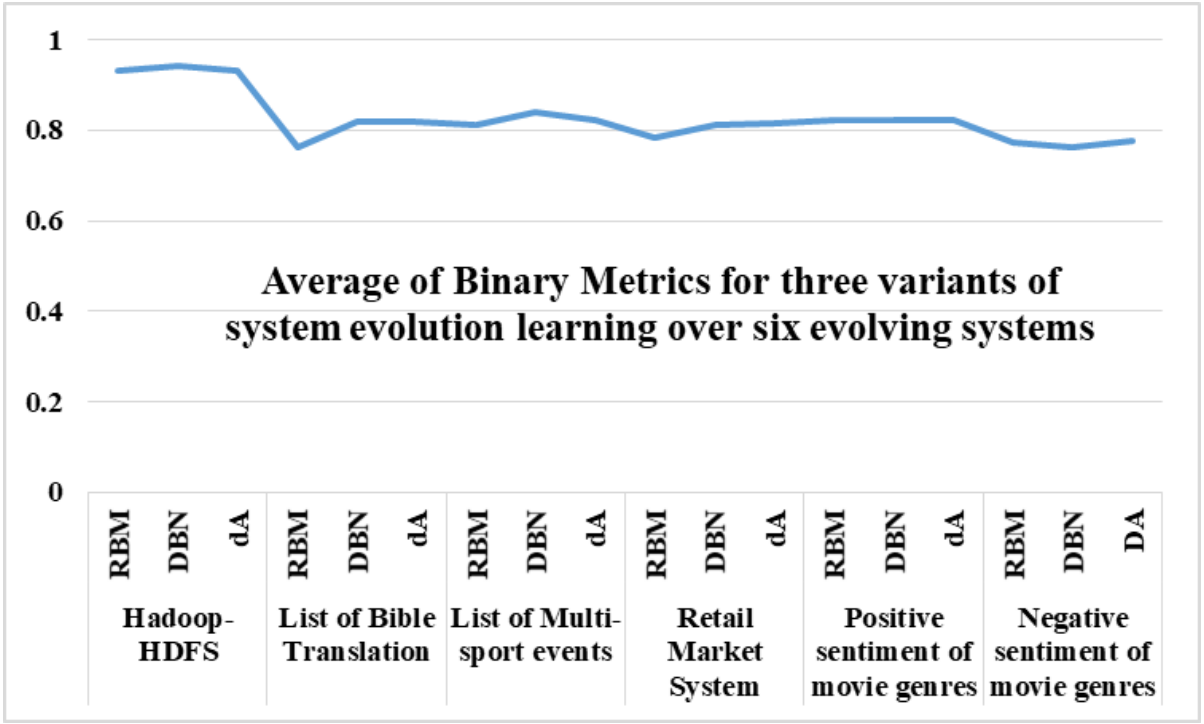




Experiments on Evolving Systems

Evolving systems	Training States (N)	Deep Evolution learning	ABM
Hadoop HDFS-Core	15	RBM	0.888
		DBN	0.886
		dA	0.883
Bible Translation	13	RBM	0.649
		DBN	0.749
		dA	0.749
Multi-sport Events	13	RBM	0.662
		DBN	0.714
		dA	0.696
Retail market system	13	RBM	0.755
		DBN	0.760
		dA	0.737
Positive sentiment of movie genres	16	RBM	0.68
		DBN	0.68
		dA	0.680
Negative sentiment of movie genres	16	RBM	0.623
		DBN	0.639
		dA	0.631

- **Average of Binary Metrics** (ABM)
- $\text{ABM} = \frac{(\text{accuracy} + \text{precision} + \text{fmeasure} + \text{recall})}{4}$



References

- Michael L. Littman, Introduction to Artificial Intelligence, Princeton University.
- Introduction to Machine Learning, MIT.
- Machine Learning with Python, MIT.
- **Animesh Chaturvedi**, Aruna Tiwari, Shubhangi Chaturvedi, and Pietro Lio'. "System Neural Network: Evolution and Change based Structure Learning". *IEEE Transactions on Artificial Intelligence*, Vol. 3.3, pp. 426 - 435, June 2022. **DOI:** 10.1109/TAI.2022.3143778. PDF
- **Animesh Chaturvedi**, Aruna Tiwari, and Shubhangi Chaturvedi "SysEvoRecomd: Network Reconstruction by Graph Evolution and Change Learning". *IEEE Systems Journal*, Vol. 14.3, pp. 4007 - 4014, Sept. 2020. **DOI:** 10.1109/JSYST.2020.2988037. PDF, Video, IF: 4.463, Q1.

תודה רבה
Hebrew

Ευχαριστώ
Greek

Спасибо
Russian

Danke
German

धन्यवादः
Sanskrit

Merci
French

ধন্যবাদ
Bangla

நன்றி
Tamil

شكراً
Arabic

ಧನ್ಯವಾದಗಳು
Kannada

Thank You
English

നന്ദി
Malayalam

多謝
Traditional Chinese

Grazie
Italian

ధన్యవాదాలు
Telugu

આભાર
Gujarati

ਧੰਨਵਾਦ
Punjabi

धन्यवाद
Hindi & Marathi

Gracias
Spanish

多谢
Simplified Chinese

https://sites.google.com/site/animeshchaturvedi07

Obrigado
Portuguese

ありがとうございました
Japanese

ขอบคุณ
Thai

감사합니다
Korean