DEEP LEARNING FOR SIGNAL & IMAGE PROCESSING

Dr. Mithun Kumar Kar

School of Artificial Intelligence

Amrita Vishwa Vidyapeetham Coimbatore

Recurrent Neural Network

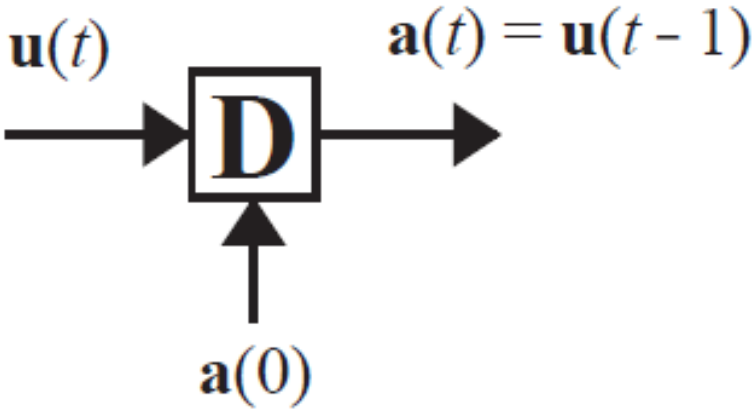
- Artificial neural networks that do not have looping nodes are called feed forward neural networks. Because all information is only passed forward, this kind of neural network is also referred to as a multi-layer neural network.
- Recurrent Neural Network(RNN) is a type of neural network where the output from the previous step is fed as input to the current step.

Recurrent Neural Network

- A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data.
- Recurrent Neural Network(RNN) is a type of neural network where the output from the previous step is fed as input to the current step.
- For sequential data analysis RNN is used.

Recurrent Neural Nets

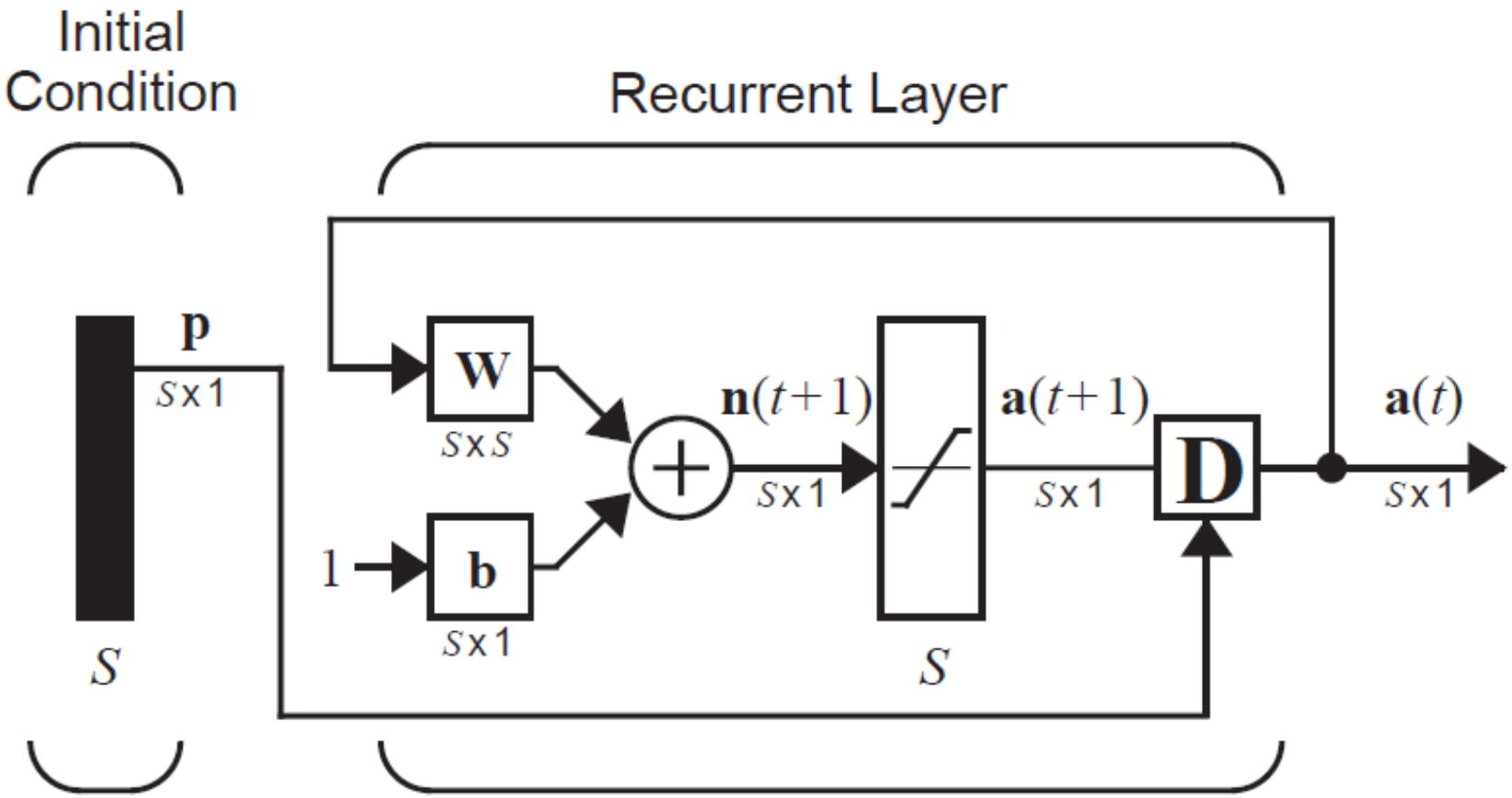
- **Delay block:**



- The delay output a(t) is computed from its input u(t) as a(t)=u(t-1). Thus the output is the input delayed by one time step.
- The initial condition of output is given by a(0).

Recurrent Neural Nets

- A recurrent network is a network with feedback; some of its outputs are connected to its inputs.



Recurrent Neural Nets

- In this particular network the vector **p** supplies the initial conditions that is **a(0)= p.**

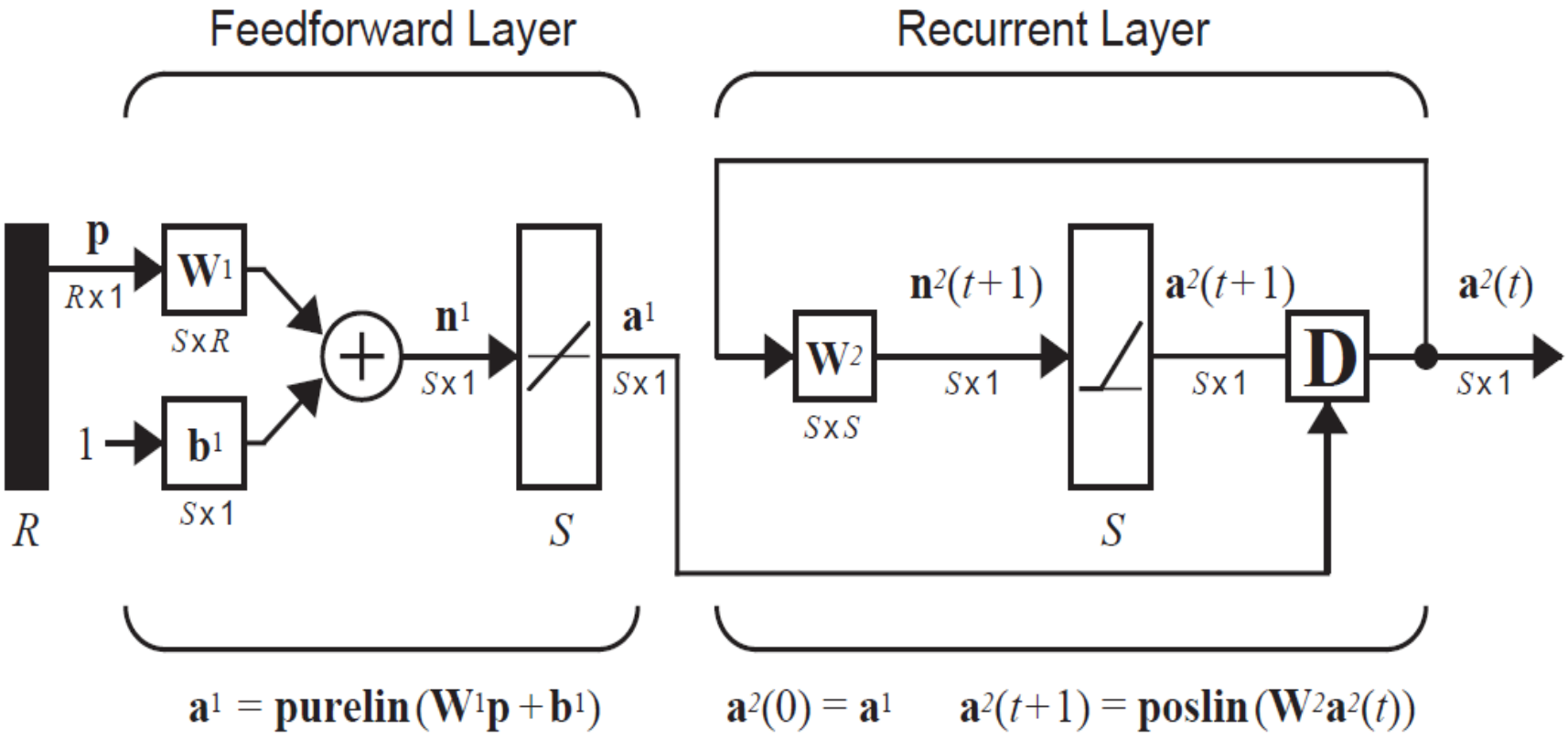
$$\mathbf{a}(1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(0) + \mathbf{b})$$

$$\mathbf{a}(2) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(1) + \mathbf{b})$$

- Recurrent networks can exhibit temporal behavior.
- where **satlins** is the transfer function that is linear in the range [-1, 1].

Hamming Network

- The objective of the Hamming network is to decide which prototype vector is closest to the input vector.



Recurrent Neural Network

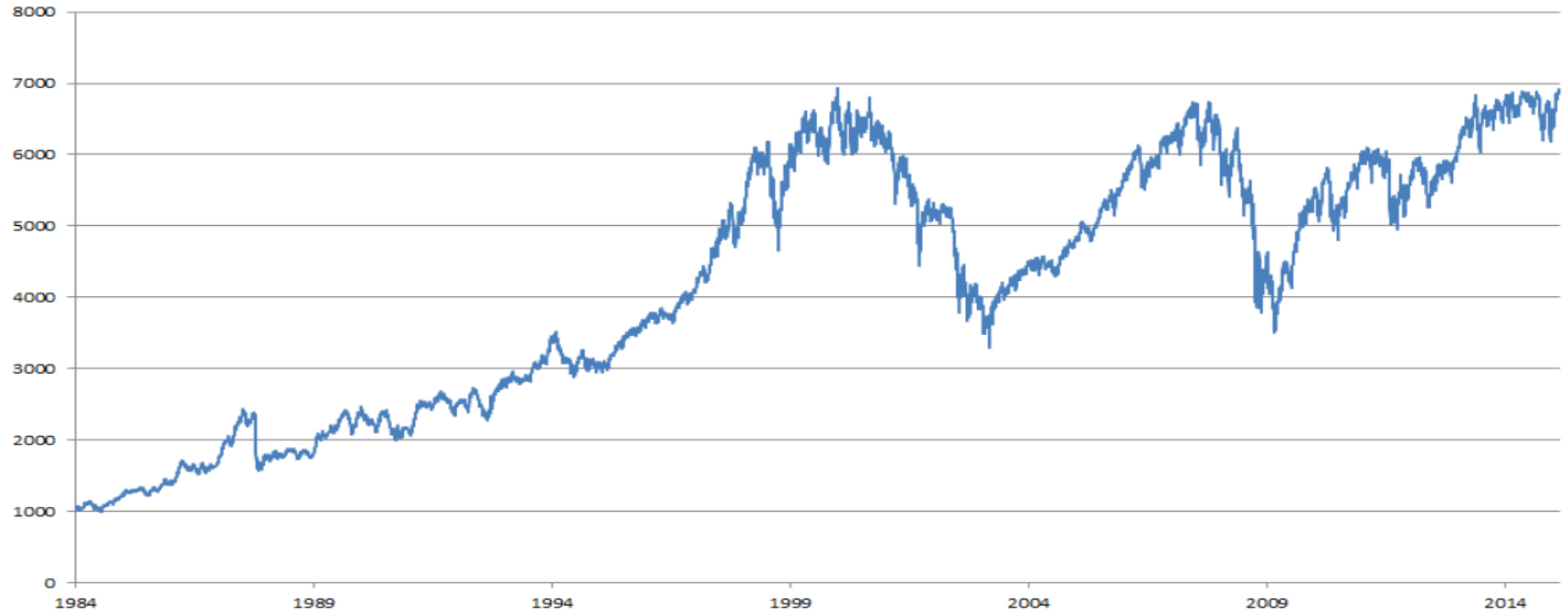
- **Sequential data:** the words in a paragraph, Image frames in a video, the audio signal in a conversation, the browsing behavior on a website, all follow sequential order.
- In short, while CNNs can efficiently process spatial information, recurrent neural networks (RNNs) are designed to better handle sequential information.
- Music, speech, text, and videos are all sequential in nature. If we were to permute them they would make little sense.
- **dog bites man/man bites dog**

Recurrent Neural Network

- Problems such as Speech Recognition or Time-series Prediction require a system to store and use context information.
- **R**ecurrent **N**eural **N**etwork**s** take the previous output or hidden states as inputs. The composite input at time *t* has some historical information about the happenings at time T < *t.*

Statistical Tools

- We need statistical tools and new deep neural network architectures to deal with sequence data.



- Price of a share in different years in stock market.

Statistical Tools

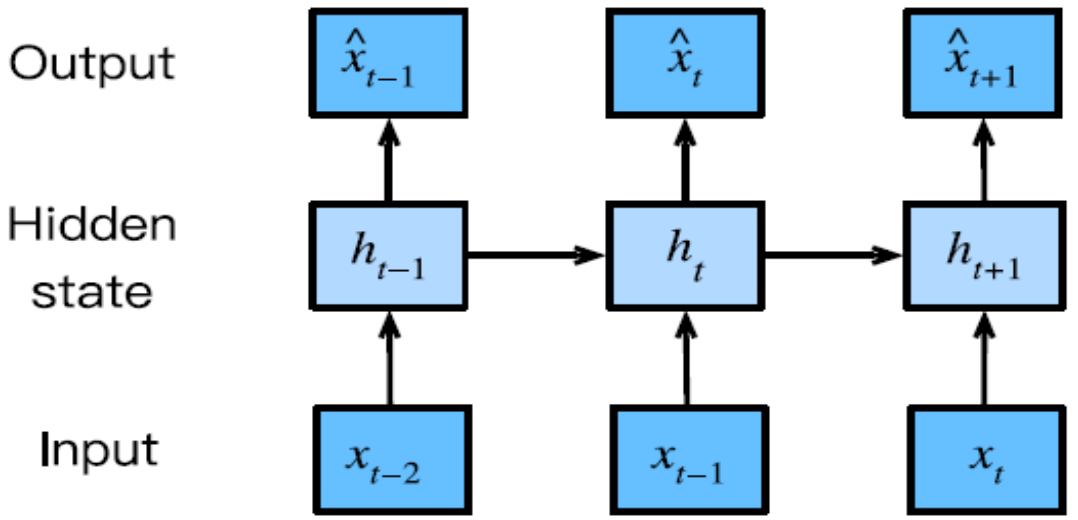
- Let us denote the prices by x_t i.e., at time step t.
- Suppose that a trader who wants to do well in the stock market on day t predicts x_t via

$$x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1).$$

- In order to achieve this, our trader could use a regression model.
- How to estimate $x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1).$ efficiently?

Statistical Tools

- First we can use autoregressive model of limited time span.
- The second strategy, shown in Figure, is to keep some summary h_t of the past observations and at the same time update h_t in addition to the prediction $\hat{x}_t$.



A latent autoregressive model.

Statistical Tools

- This leads to models that estimate x_t by

$$\hat{x}_t = p(x_t \mid h_t)$$

and moreover updates of the form $h_t = g(h_{t-1}, x_{t-1})$

- We can estimate the entire sequence by

$$p(x_1, x_{2,\ldots} x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \ldots., x_1)$$

- We need to use a neural based classifier to estimate

$$P(x_t \mid x_{t-1}, \ldots., x_1)$$

Statistical Tools

- The probability of a text sequence containing four words

P(deep, learning, is, fun) =

P(deep)P(learning | deep)P(is | deep, learning) P(fun | deep, learning, is)

- In order to compute the language model, we need to calculate the probability of words and the conditional probability of a word given the previous few words.

$$P(x_t \mid x_{t-1}, \ldots, x_1) \approx P(x_t \mid h_{t-1})$$

RNN

- RNN uses a latent variable to predict x_t as

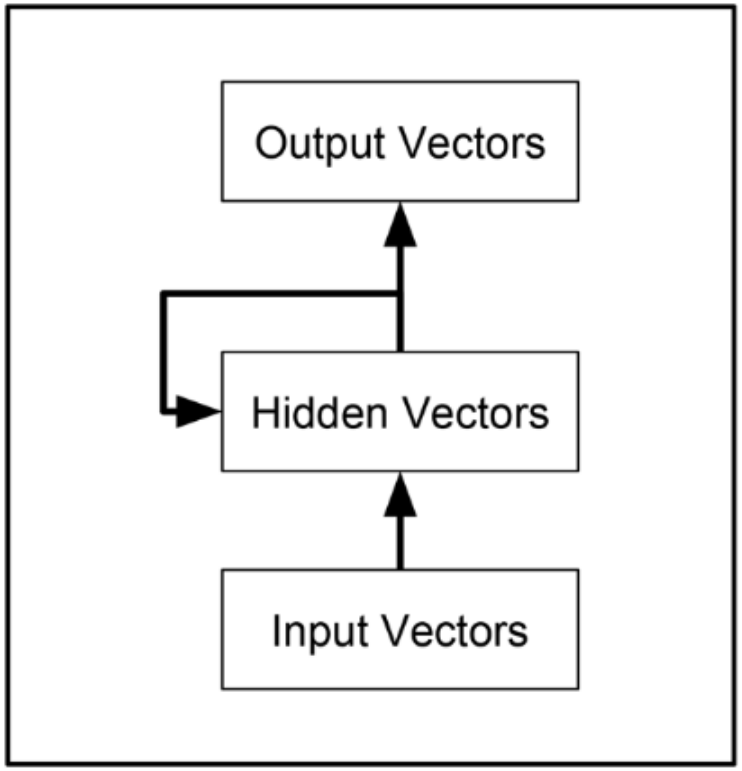
$$P(x_t \mid x_{t-1}, ..., x_1) \approx P(x_t \mid h_{t-1})$$

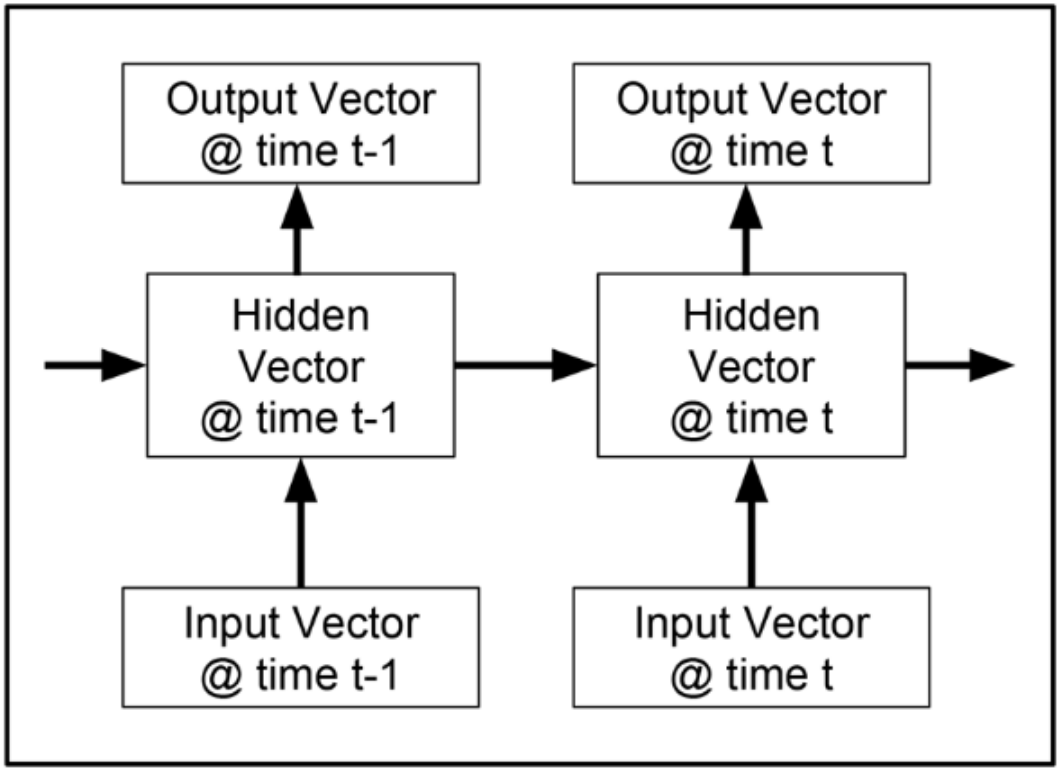
- Where h_{t-1} is a hidden state (also known as a hidden variable) that stores the sequence information up to time step (t-1).
- In general, the hidden state at any time step t could be computed based on both the current input x_t and the previous hidden state h_{t-1}.

$$h_t = f(x_t, h_{t-1})$$

- Recurrent neural networks (RNNs) are neural networks with hidden states.

RNN


Output Vectors
Hidden Vectors
Input Vectors


Output Vector
@ time t-1
Output Vector
@ time t
Hidden
Vector
@ time t-1
Hidden
Vector
@ time t
Input Vector
@ time t-1
Input Vector
@ time t

Neural Networks without Hidden States

- Let the hidden layer's activation function be Φ.
- Given a minibatch of examples $X \in \mathbb{R}^{n \times d}$ with batch size n and d inputs, the hidden layer's output $H \in \mathbb{R}^{n \times h}$ is calculated as

$$H = \Phi(XW_{xh} + b_h)$$

- The hidden variable H is used as the input of the output layer. The output layer is given by

$$O = \Phi(HW_{hq} + b_q)$$

- Where $O \in \mathbb{R}^{n \times q}$ is the output variable, $W_{hq} \in \mathbb{R}^{h \times q}$ is the weight parameter a $b_q \in \mathbb{R}^{1 \times q}$ the bias parameter of the output layer.

RNN with Hidden States

- Assume that we have a minibatch of inputs $X_t \in \mathbb{R}^{n \times d}$ at time step **t**.
- For a minibatch of **n** sequence examples, each row of X_t corresponds to one example at time step **t** from the sequence. The weight parameter for input is given by $W_{xh} \in \mathbb{R}^{d \times h}$
- The hidden variable of time step t is denote by

$$H_t \in \mathbb{R}^{n \times h}$$

- Here we save the hidden variable H_{t-1} from the previous time step and introduce a new weight parameter $W_{hh} \in \mathbb{R}^{h \times h}$ to describe how to use the hidden variable of the previous time step in the current time step.

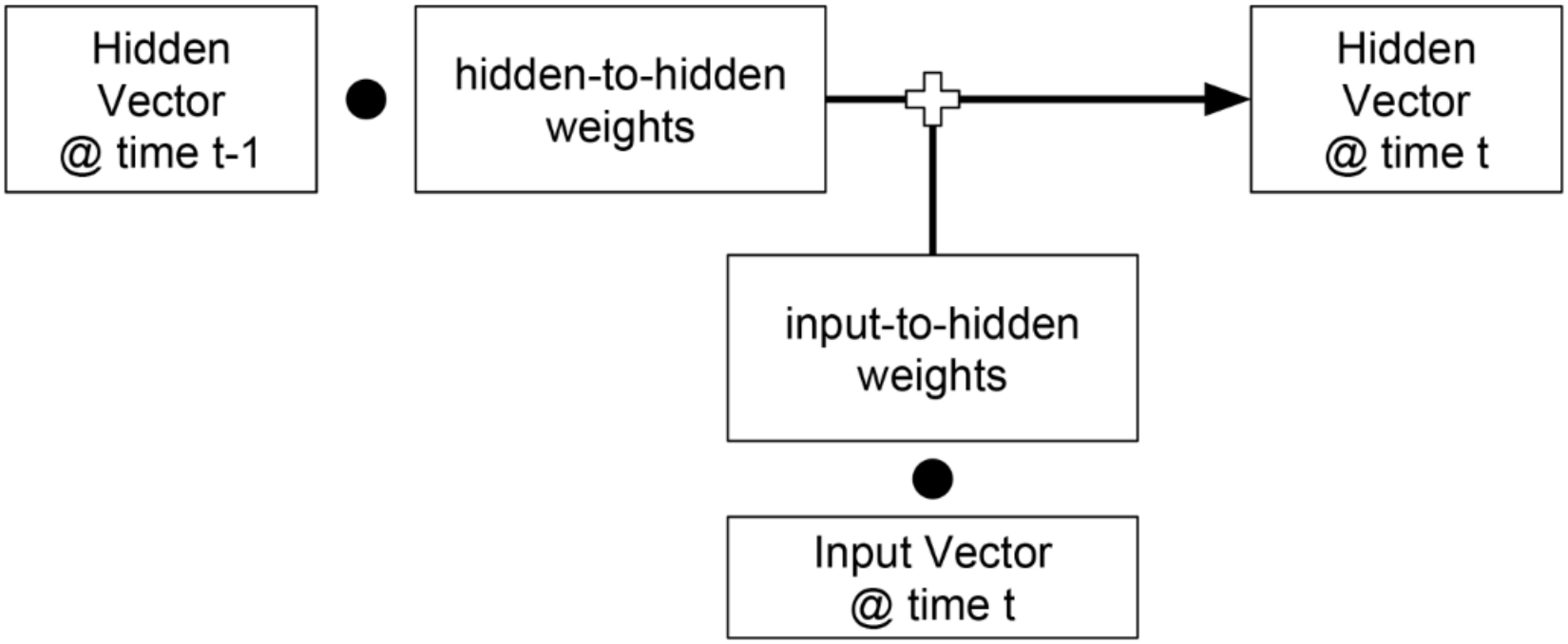
RNN with Hidden States

- Here, the calculation of the hidden variable of the current time step is determined by the input of the current time step together with the hidden variable of the previous time step:

$$H_t = \Phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$

- These variables captured and retained the sequence's historical information up to their current time step. Therefore, such a hidden variable is called a hidden state.
- Since the hidden state uses the same definition of the previous time step in the current time step, the computation of is recurrent. Hence, neural networks with hidden states based on recurrent computation are named recurrent neural networks.

RNN with Hidden States



RNN with Hidden States

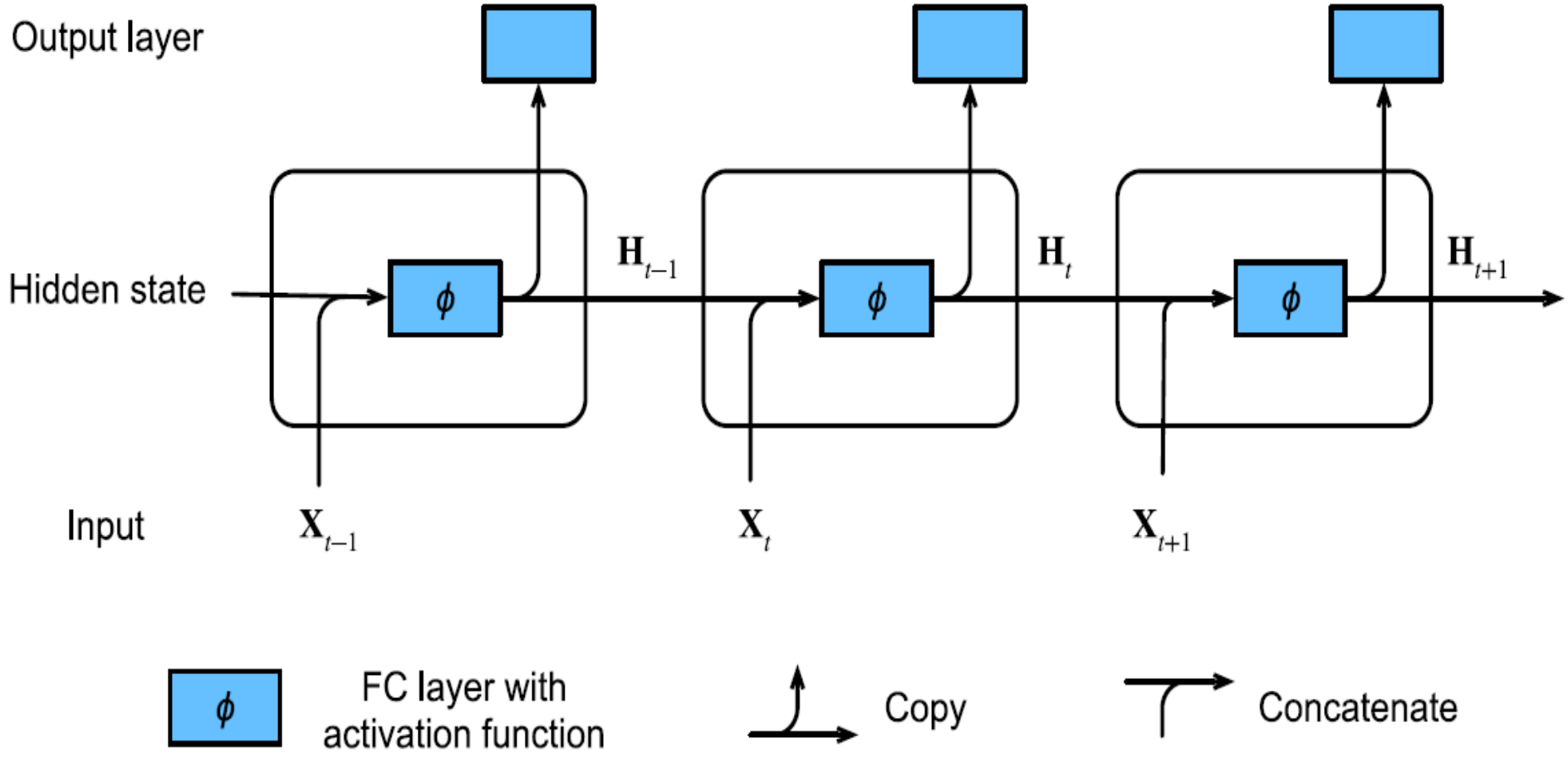
- For time step t, the output of the output layer is similar to the computation in the MLP:

$$O_t = H_t W_{hq} + b_q$$

- Parameters of the RNN include the weights $W_{xh} \in \mathbb{R}^{d \times h}$, $W_{hh} \in \mathbb{R}^{h \times h}$ and the bias $b_h \in \mathbb{R}^{1 \times h}$ of the hidden layer, together with the weights $W_{hq} \in \mathbb{R}^{h \times q}$ and the bias $b_q \in \mathbb{R}^{1 \times q}$ of the output layer.
- Even at different time steps, RNNs always use these model parameters. Therefore, the parameterization cost of an RNN does not grow as the number of time steps increases.

RNN with Hidden States

- Computational logic of an RNN at three adjacent time steps.



RNN with Hidden States

- At any time step t, the computation of the hidden state can be treated as:

1) Concatenating the input X_t at the current time step t and the hidden state H_{t-1} at the previous time step t-1;

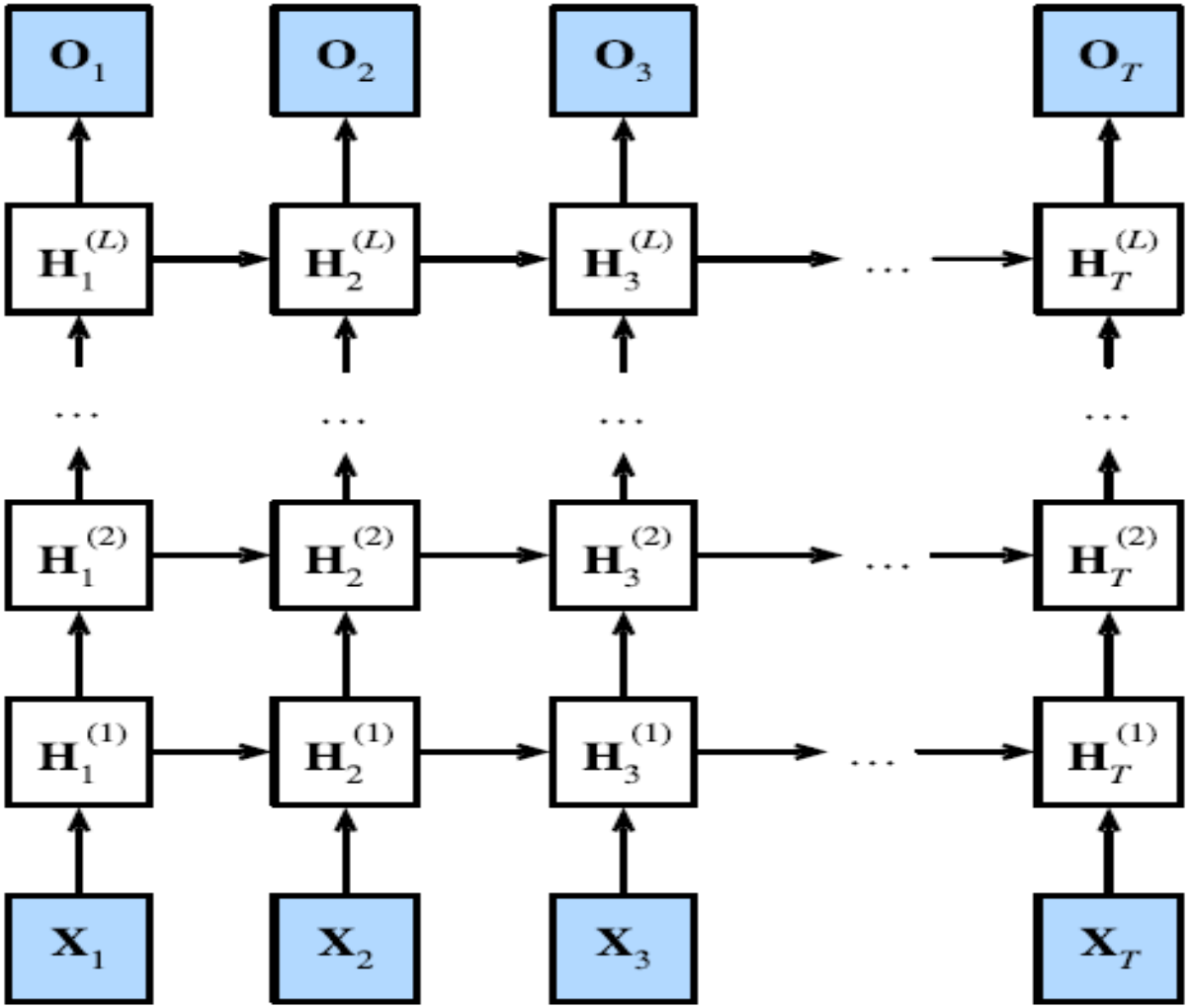
2) feeding the concatenation result into a fully connected layer with the activation function φ.

- The output of such a fully-connected layer is the hidden state H_t of the current time step t.
- The hidden state of the current time step t, H_t, will participate in computing the hidden state H_{t+1} of the next time step t + 1.

RNN with Hidden States

- H_t will also be fed into the fully-connected output layer to compute the output O_t of the current time step t.
- The calculation of $X_t W_{xh} + H_{t-1} W_{hh}$ for the hidden state is equivalent to matrix multiplication of concatenation of X_t and H_{t-1} and concatenation of W_{xh} and W_{hh}.

Architecture of a deep RNN.



Architecture of a deep RNN

- Suppose that we have a mini batch input $X \in \mathbb{R}^{n \times d}$ at time step t.
- At the same time step, let the hidden state of the l^{th} hidden layer (l = 1,2, …L) be $\mathbf{H}_t^{(l)} \in \mathbb{R}^{n \times h}$ and the output layer variable be $\mathbf{O}_t \in \mathbb{R}^{n \times q}$.
- If the hidden state of the l^{th} hidden layer that uses the activation function ϕ_l , then

$$\mathbf{H}_t^{(l)} = \phi_l(\mathbf{H}_t^{(l-1)}\mathbf{W}_{xh}^{(l)} + \mathbf{H}_{t-1}^{(l)}\mathbf{W}_{hh}^{(l)} + \mathbf{b}_h^{(l)})$$

$$\mathbf{O}_t = \mathbf{H}_t^{(L)}\mathbf{W}_{hq} + \mathbf{b}_q,$$

RNN

- RNNs are among the most powerful models that enable us to take on applications such as classification, labeling of sequential data, generating sequences of text (such as with the SwiftKey Keyboard app, which predicts the next word), and converting one sequence to another, such as when translating a language (for example, from French to English).
- As RNNs can process data in sequence, we can use vectors of different lengths and generate outputs of different lengths.

Implementation

- Assume we have the following one dimension array input data (row = 7 , columns=1).
- [1,2,3,4,5,6,7]
- we created sequential data and the label as shown below.
- [1,2,3],[2,3,4], [3,4,5],[4,5,6],[5,6,7]
- Now we need to break this one into batches. Let's say we take batch size = 2.

Inputs			label
1	2	3	4
2	3	4	5

Batch 1

Inputs			label
3	4	5	6
4	5	6	7

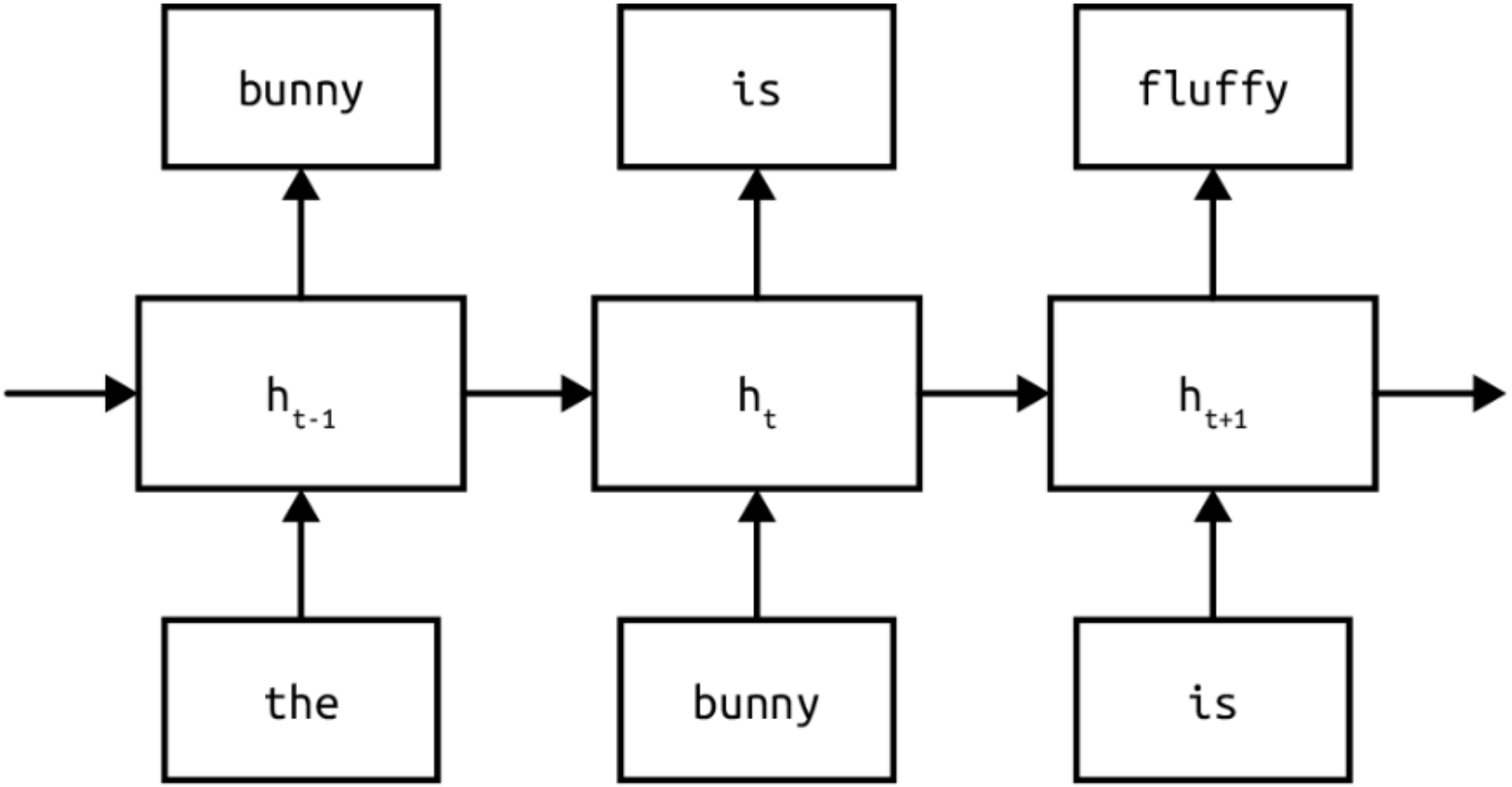
Batch 2

Implementation

- Input data: RNN should have 3 dimensions. (Batch Size, Sequence Length and Input Dimension).
- Batch Size is the number of samples we send to the model at a time.
- We have batch size = 2 but you can take it 4, 8,16, 32, 64 etc. depends on the memory (basically in 2's power).
- Sequence Length is the length of the sequence of input data (time step:0,1,2…N), the RNN learn the sequential pattern in the dataset.
- Our sequence length = 3.

Implementation

- We can predict the next character based on the current and previous characters via an RNN for character-level language modeling.



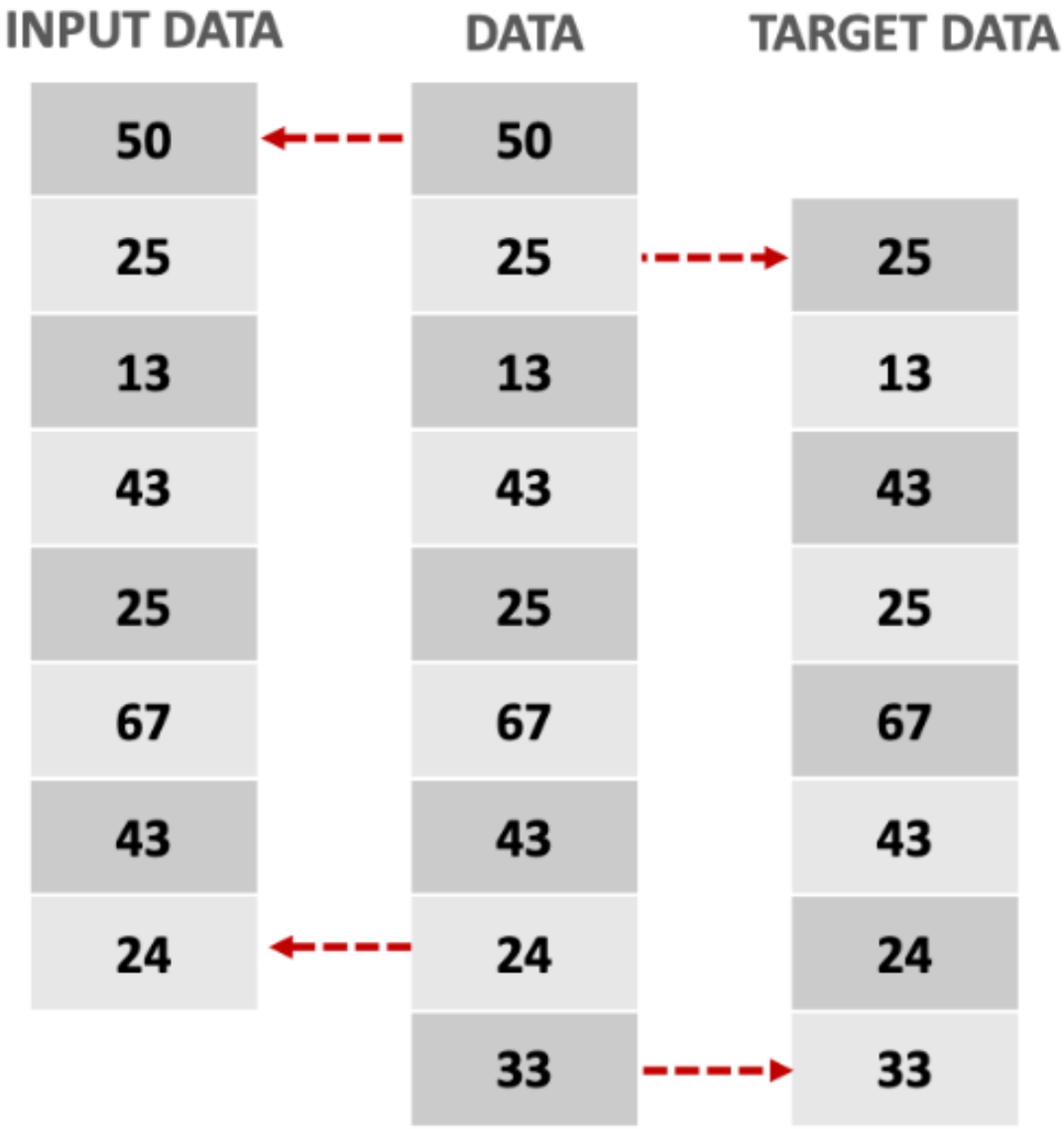
Implementation

- Create random dataset of 10 weekly records representing 5 weeks of data, i.e., shape (10,5), then the first 4 columns are data and last 4 columns are targets.
- | | week1 | week2 | week3 | week4 | week5 |
- | --- | --- | --- | --- | --- | --- |
- | 0 | 44 | 47 | 64 | 67 | 67 |
- | 1 | 9 | 83 | 21 | 36 | 87 |
- | 2 | 70 | 88 | 88 | 12 | 58 |
- | 3 | 65 | 39 | 87 | 46 | 88 |
- | 4 | 81 | 37 | 25 | 77 | 72 |

Implementation

- Considering that the objective is to predict the following element in a sequence, the target matrix is typically the same information as the input data, with the target being one step ahead.
- This means that the input variable should contain all the data points of the sequence, except for the last value. while the target variable should contain all the data points of the sequence, except for the first value – that is, the first value of the target variable should be the second one of the input variable, and so on,

Implementation



Input and target variables for a sequenced data problem

Implementation

- Import the required libraries.
- Load the dataset and slice it so that it contains all the rows but only the columns from index 1 to 52.
- Plot the weekly sales transactions of five randomly chosen products from the entire dataset. Use a random seed of 0 when doing random sampling in order to achieve the same results as in the current activity.
- Create the inputs and targets variables, which will be fed to the network to create the model. These variables should be of the same shape and be converted into PyTorch tensors.

Implementation

- The inputs variable should contain the data for all the products for all the weeks, except the last week because the idea of the model is to predict this final week.
- The targets variable should be one step ahead of the inputs variable; that is, the first value of the targets variable should be the second one of the inputs variable, and so on, until the last value of the targets variable (which should be the last week that was left outside of the inputs variable)

Implementation

- Create a class containing the architecture of the network; note that the output size of the fully connected layer should be 1.
- Instantiate the class function containing the model. Feed the input size, the number of neurons in each recurrent layer (10), and the number of recurrent layers (1).
- Define a loss function, an optimization algorithm, and the number of epochs to train the network.
- Use the Mean Squared Error loss function, the Adam optimizer, and 10,000 epochs for this.

Implementation

- Use a for loop to perform the training process by going through all the epochs.
- In each epoch, a prediction must be made, along with the subsequent calculationof the loss function and the optimization of the parameters of the network.
- Then, save the loss of each of the epochs.
- Plot the losses of all the epochs.
- Using a scatter plot, display the predictions that were obtained in the last epoch of the training process against the ground truth values (that is, the sales transactions of the last week).

Gated recurrent units (GRUs)

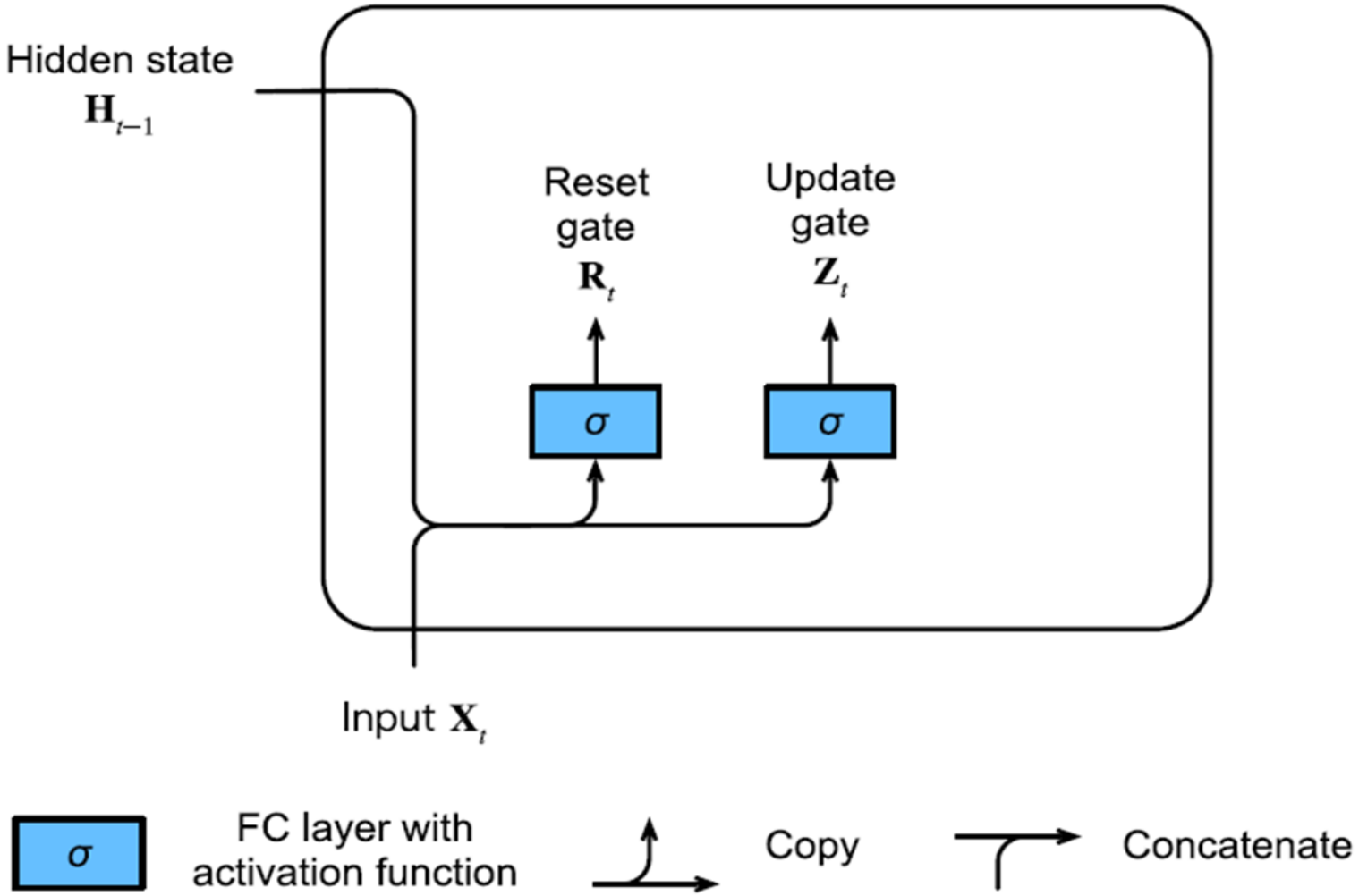
- We found that long products of matrices can lead to vanishing or exploding gradients in RNN.
- We might encounter situations where an early observation is highly significant for predicting all future observations.
- We might encounter situations where some tokens carry no pertinent observation.
- We might encounter situations where there is a logical break between parts of a sequence.
- To better capture dependencies for sequences with large time step distances, GRUs and LSTM are used.

Gated Hidden State

- **Gating of the hidden state**. This means that we have dedicated mechanisms for when a hidden state should be updated and also when it should be reset.
- We will learn how to update the hidden state according to the requirement. We need to introduce the **reset gate** and the **update gate.**
- For instance, a reset gate would allow us to control how much of the previous state we might still want to remember.
- An update gate would allow us to control how much of the new state is just a copy of the old state.

Reset and update gates in a GRU

- Figure shows the inputs for both the reset and update gates in a GRU



Reset and update gates in a GRU

- The outputs of two gates are given by two fully-connected layers with a sigmoid activation function.
- For a given time step t, suppose that the input is a minibatch $X_t \in \mathbb{R}^{n \times d}$ (number of examples: *n*, number of inputs: *d*), and the hidden state of the previous time step is $H_{t-1} \in \mathbb{R}^{n \times h}$ (number of hidden units: *h*).
- Then, the reset gate $R_t \in \mathbb{R}^{n \times h}$ and update gate $Z_t \in \mathbb{R}^{n \times h}$ are computed as follows:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z),$$

Reset and update gates in a GRU

- Where $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h \times h}$ are weight parameters and $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1 \times h}$ are biases.
- **Candidate Hidden State:**
- The candidate hidden state $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ at time step *t* is given by

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h),$$

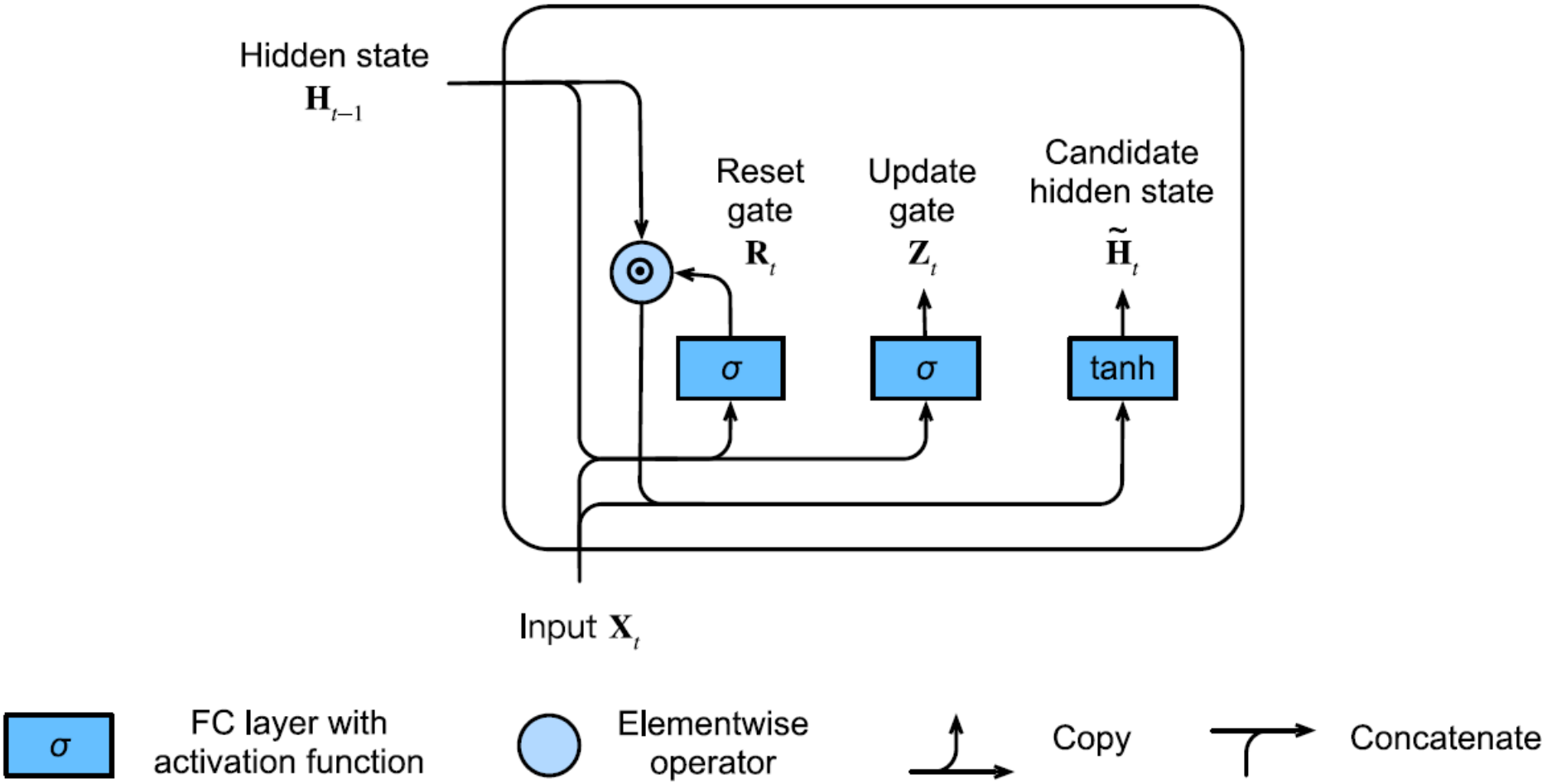
- Where $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ and $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ is the bias.
- The values in the candidate hidden state remain in the interval (-1, 1).

Reset and update gates in a GRU

- The result is a candidate since we still need to incorporate the action of the update gate.
- For a simple RNN $H_t = \Phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$
- Now $\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h),$
- Now the influence of the previous states can be reduced with the element wise multiplication of **R**t and H_{t-1}.
- Whenever the entries in the reset gate Rt are close to 1, we recover a simple RNN, entries of the reset gate Rt that are close to 0, the candidate hidden state is the result of an MLP with Xt as the input.

Gated Recurrent Units (GRU)

- Figure shows the computational flow after applying the reset gate.



Gated Recurrent Units (GRU)

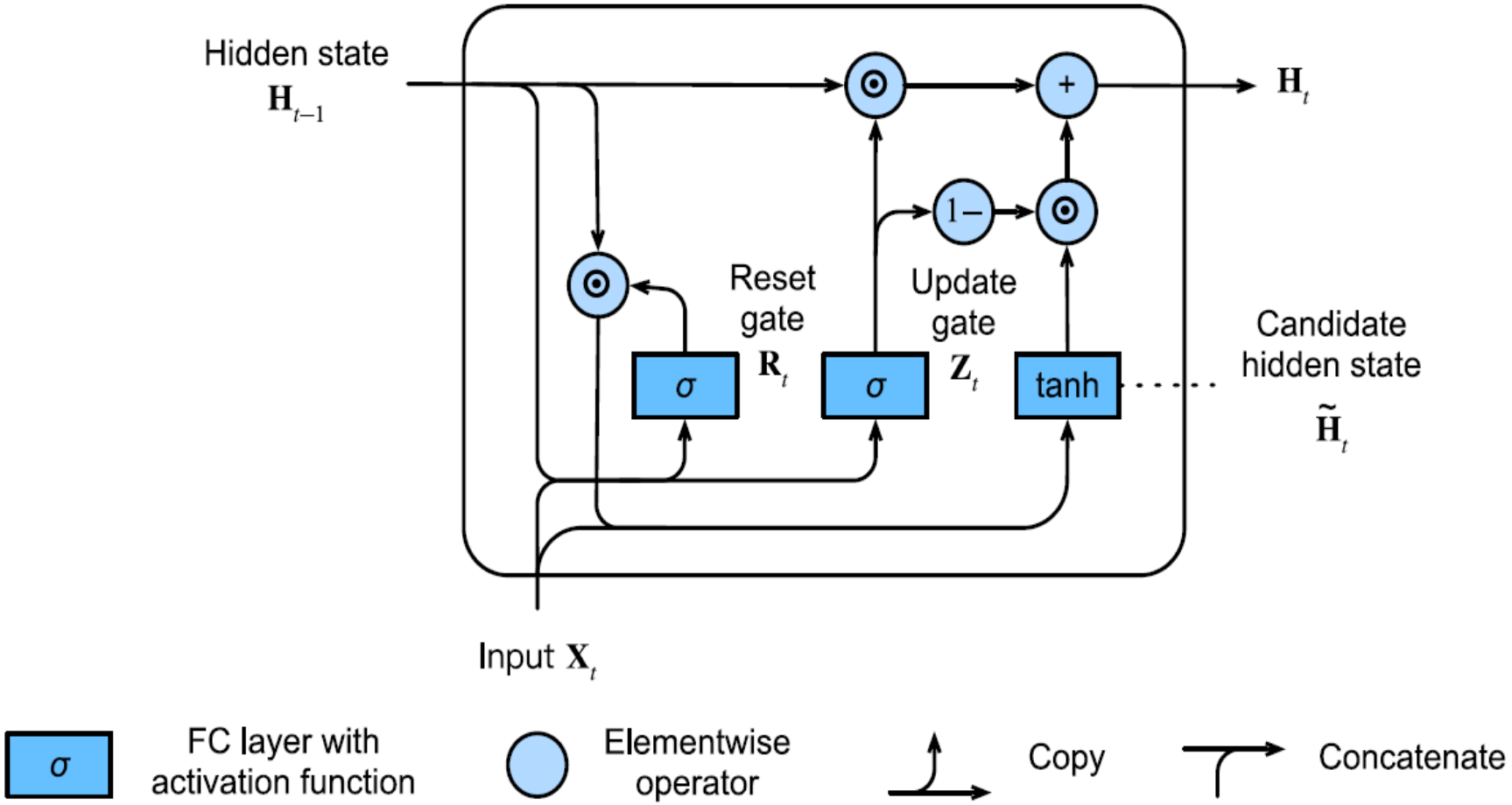
- We need to incorporate the effect of the update gate Z_t.
- This determines the extent to which the new hidden state $H_t \in \mathbb{R}^{n \times h}$ is just the old state H_{t-1} and by how much the new candidate state $\tilde{H}_t$ is used.
- This leads to the final update equation for the GRU

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t.$$

- Whenever the update gate Z_t is close to 1, we simply retain the old state and whenever $\mathbf{Z}_t$ is close to 0, the new latent state $\mathbf{H}_t$ approaches the candidate latent state $\tilde{H}_t$.

Gated Recurrent Units (GRU)

- Figure illustrates the computational flow after the update gate is in action.



Gated Recurrent Units (GRU)

- These designs can help us cope with the vanishing gradient problem in RNNs and better capture dependencies for sequences with large time step distances.
- For instance, if the update gate has been close to 1 for all the time steps of an entire subsequence, the old hidden state at the time step of its beginning will be easily retained and passed to its end, regardless of the length of the subsequence.
- Reset gates help capture short-term dependencies in sequences.
- Update gates help capture long-term dependencies in sequences.

Long Short-Term Memory (LSTM)

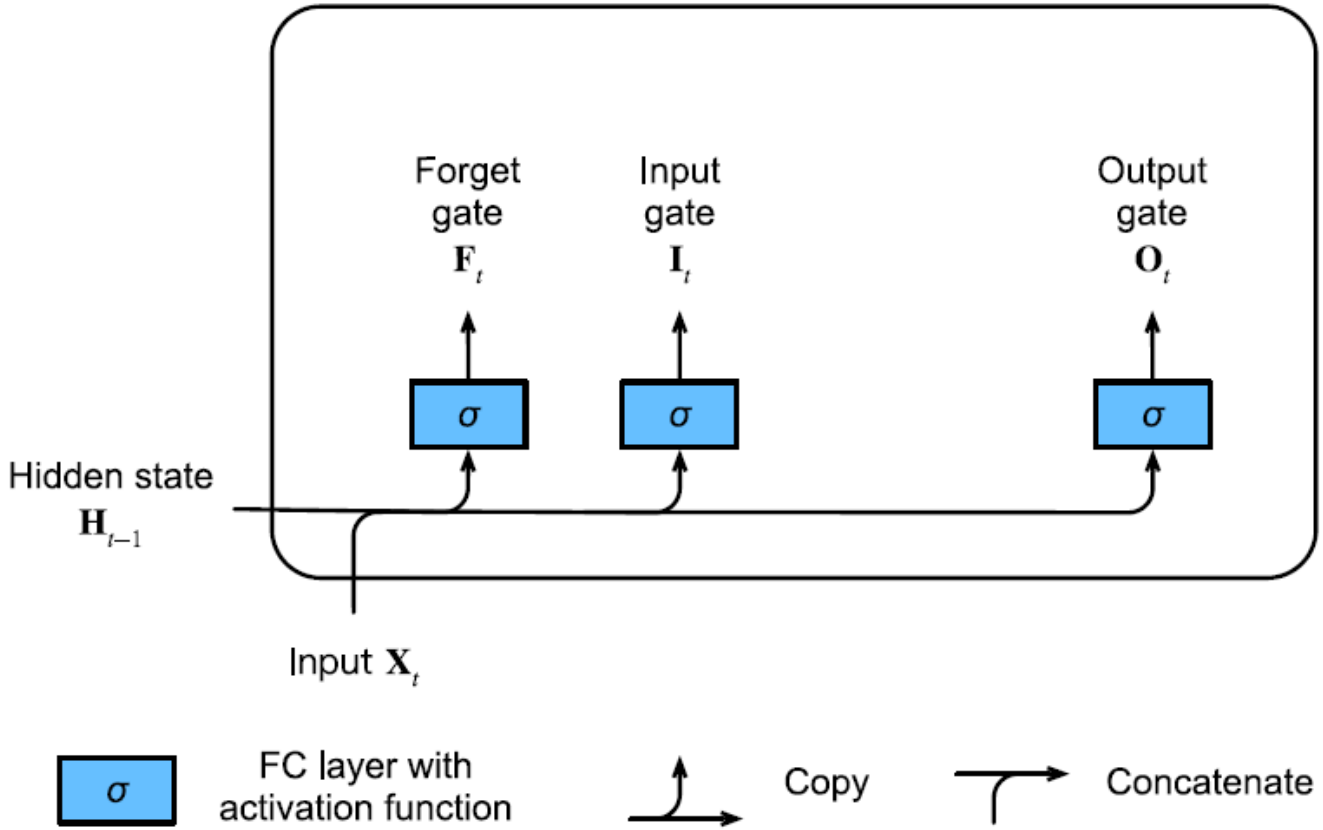
- The challenge to address long-term information preservation and short-term input skipping in latent variable models has existed for a long time.
- The motivation for LSTM design is the same as that of GRUs, namely to be able to decide when to remember and when to ignore inputs in the hidden state via a dedicated mechanism.
- LSTM introduces **a memory cell (or cell for short)** that has the same shape as the hidden state, engineered to record additional information.

LSTM

- **Input Gate, Forget Gate, and Output Gate:**
- Input gate is needed to decide when to read data into the cell.
- Output gate is needed to read out the entries from the cell.
- Forget gate is needed to reset the content of the cell.
- The data feeding into the LSTM gates are the input at the current time step and the hidden state of the previous time step.

LSTM

- They are processed by three fully connected layers with a sigmoid activation function to compute the values of the input, forget, and output gates. As a result, values of the three gates are in the range of (0,1).



LSTM

- Suppose that the number of inputs is d, the batch size is n, and there are h hidden units.
- Thus, the input is $\mathbf{X}_t \in \mathbb{R}^{n \times d}$
- The hidden state of the previous time step is $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$
- Correspondingly, the gates at time step t are defined as follows:
- The dimension of input gate is $\mathbf{I}_t \in \mathbb{R}^{n \times h}$, the forget gate is $\mathbf{F}_t \in \mathbb{R}^{n \times h}$, and the output gate is $\mathbf{O}_t \in \mathbb{R}^{n \times h}$.

LSTM

- They are calculated as follows

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$
$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$

- Where $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho} \in \mathbb{R}^{h \times h}$ are weight parameters and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{1 \times h}$ are bias parameters.
- Next we design the memory cell.

LSTM

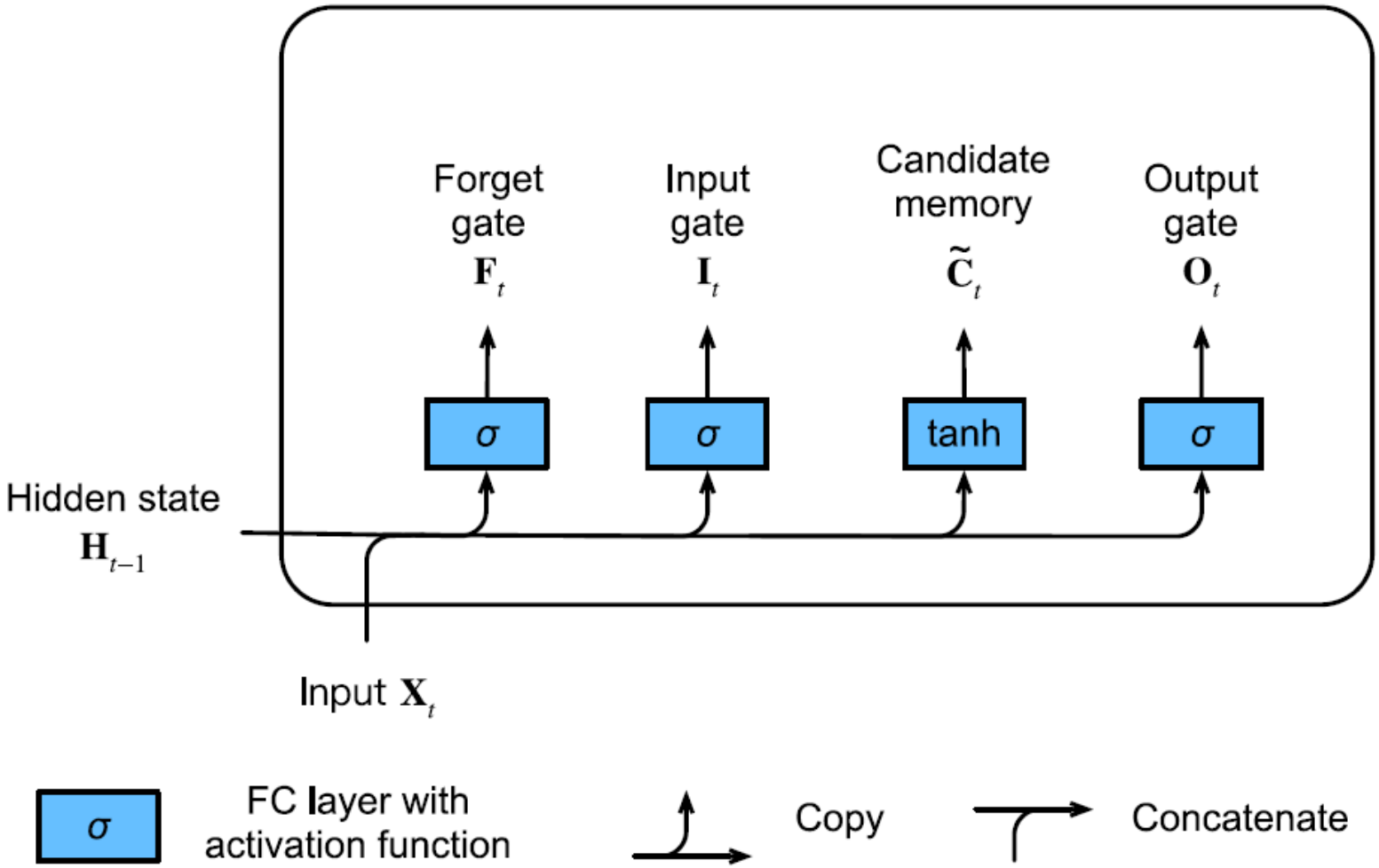
- **Candidate Memory Cell:**
- The candidate memory cell $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n \times h}$ is similar to that of the three gates, but using a tanh activation function with a value range for (-1; 1).
- At time step t, it is given by

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c),$$

where $\mathbf{W}_{xc} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hc} \in \mathbb{R}^{h \times h}$ are weight parameters and $\mathbf{b}_c \in \mathbb{R}^{1 \times h}$ is a bias parameter.

LSTM

- The candidate memory cell is shown in Figure



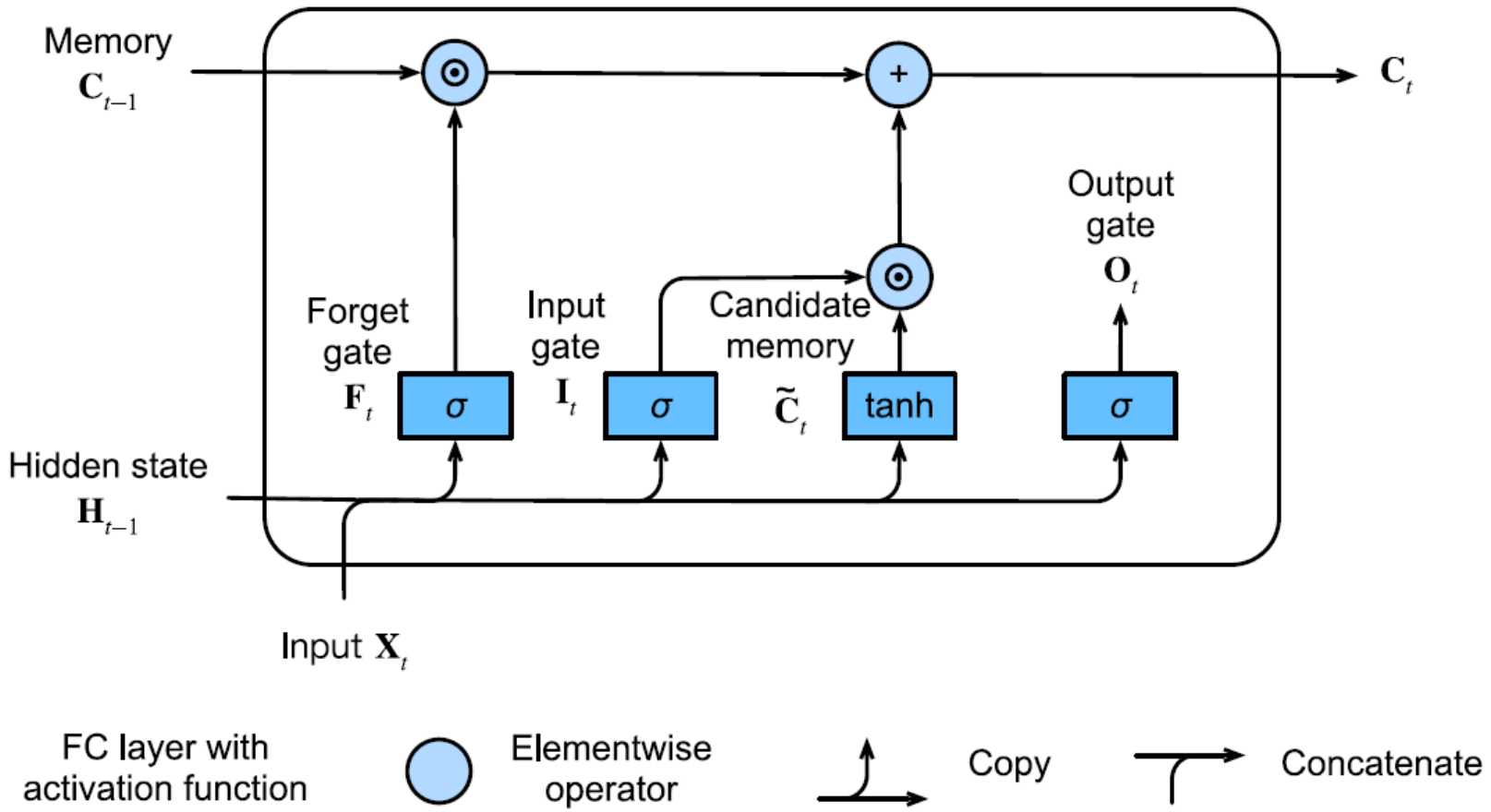
LSTM

- **Memory Cell:**
- In LSTMs we have two dedicated gates for governing input and forgetting (or skipping).
- The input gate $\mathbf{I}_t$ governs how much we take new data into account via $\tilde{\mathbf{C}}_t$.
- The forget gate $\mathbf{F}_t$ addresses how much of the old memory cell content $\mathbf{C}_{t-1} \in \mathbb{R}^{n \times h}$ we retain.
- The updating equation for memory cell is

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

LSTM

- If the forget gate is always approximately 1 and the input gate is always approximately 0, the past memory cells $\mathbf{C}_{t-1}$ will be saved over time and passed to the current time step.



LSTM

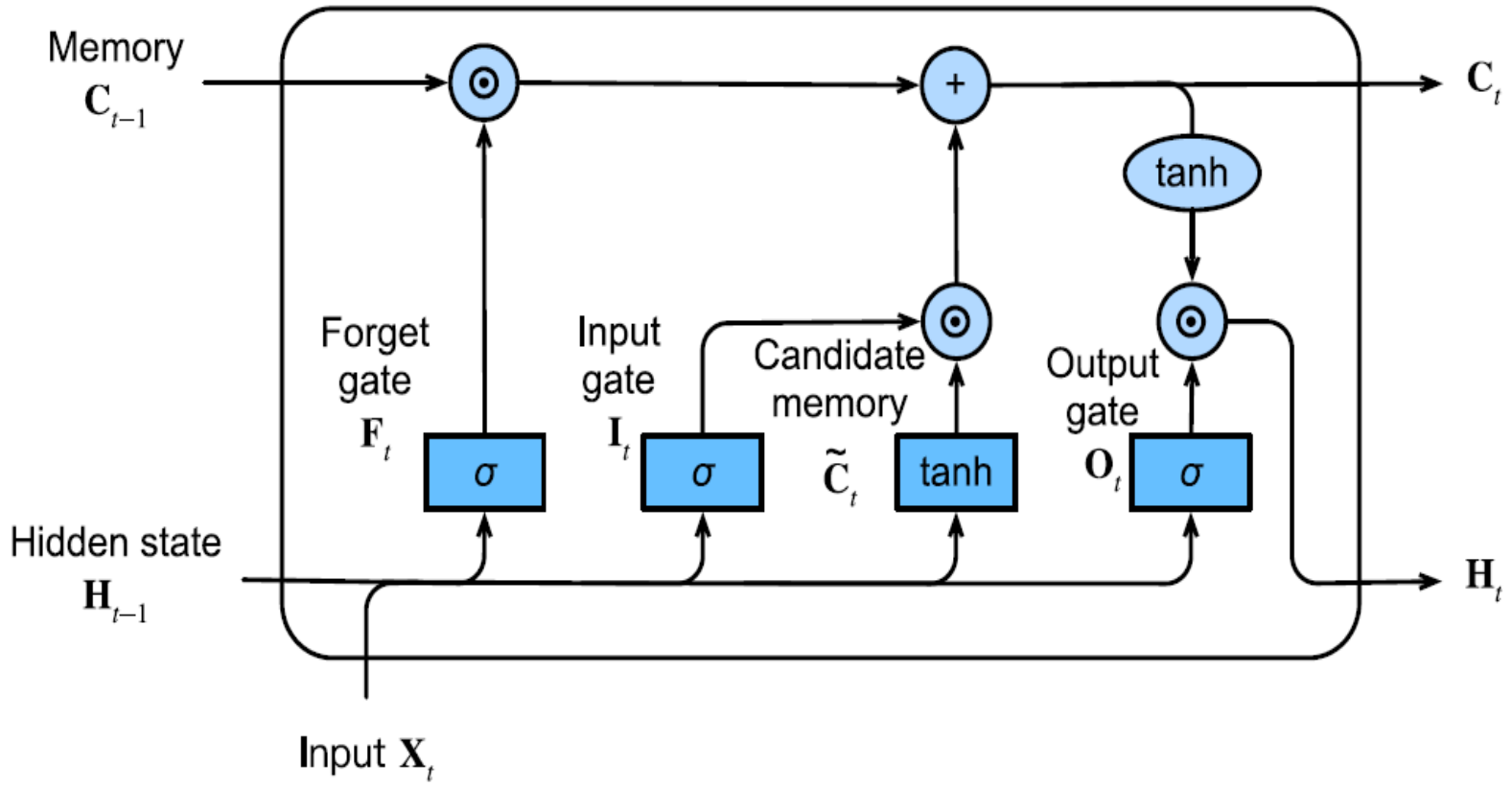
- **Hidden State:**
- Now we need to define how to compute the hidden state $\mathbf{H}_t \in \mathbb{R}^{n \times h}$. In LSTM it is simply a gated version of the tanh of the memory cell.

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t).$$

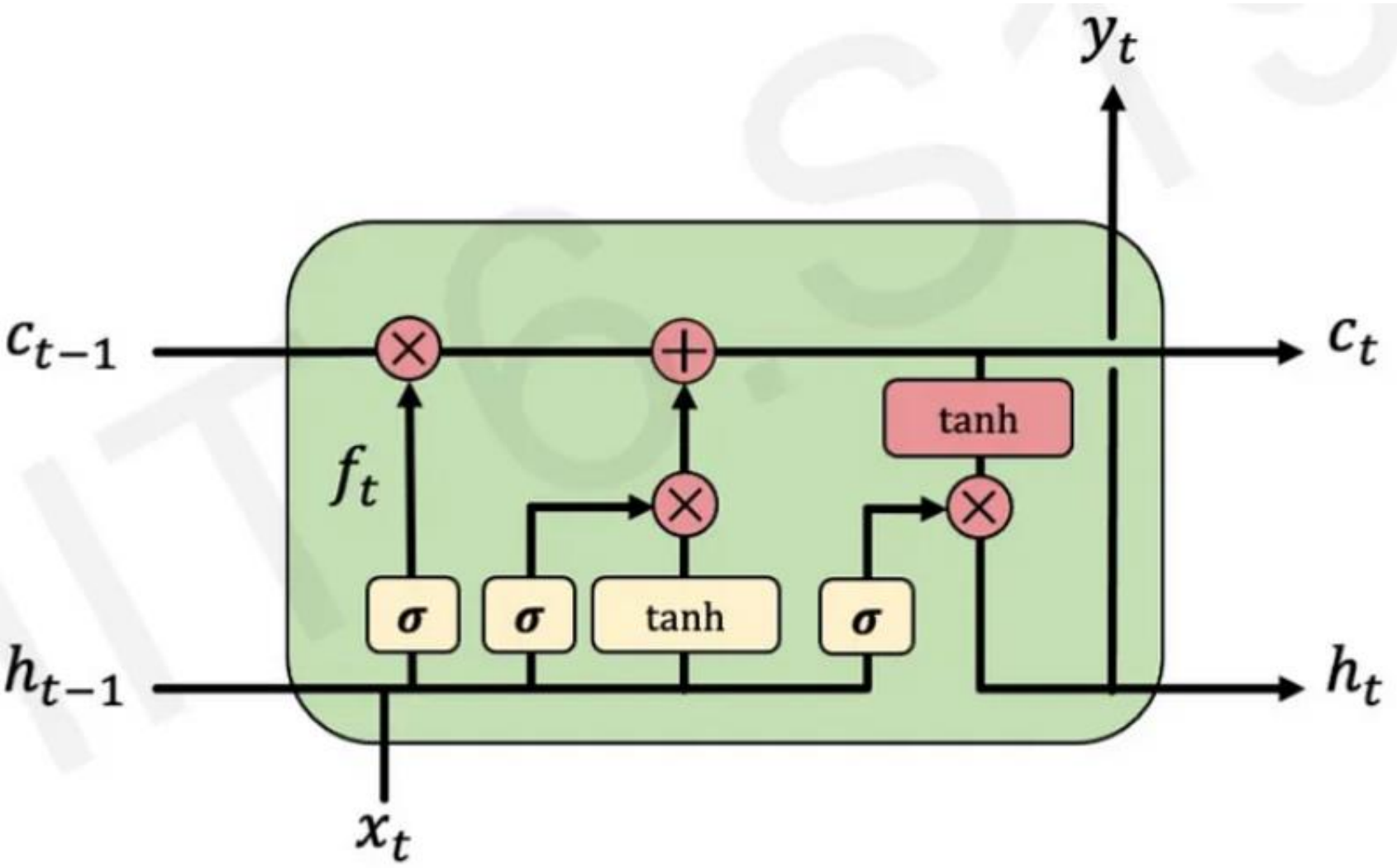
- Whenever the output gate approximates 1 we effectively pass all memory information through to the predictor, whereas for the output gate close to 0 we retain all the information only within the memory cell and perform no further processing.

LSTM

- Figure shows a graphical illustration of the data flow.



LSTM



Text classification

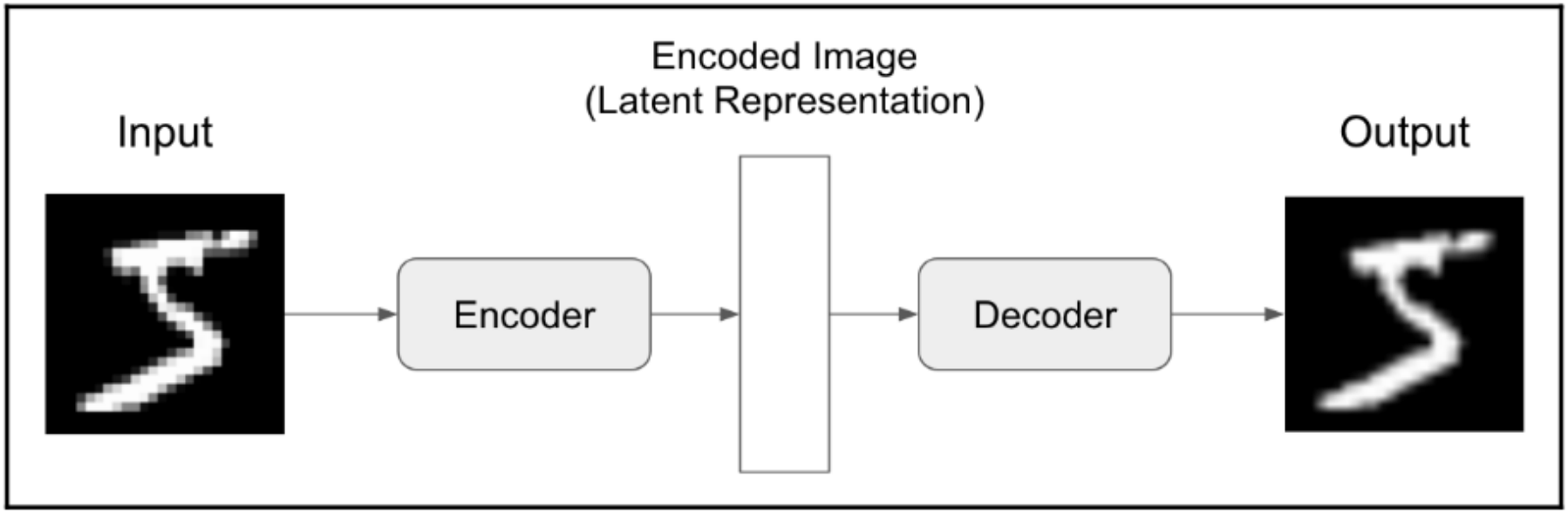
- How to represent text as vectors or tensors?
- The simplest approach is still one that you'll find in many approaches to NLP, and it's called one-hot encoding.
- The cat sat on the mat.
- We have a tensor of [the, cat, sat, on, mat].
- One-hot encoding simply means that we create a vector that is the size of the vocabulary, and for each word in it, we allocate a vector with one parameter set to 1 and the rest to 0:

Text classification

- [the, cat, sat, on, mat]
- the — [1 0 0 0 0]
- cat — [0 1 0 0 0]
- sat — [0 0 1 0 0]
- on — [0 0 0 1 0]
- mat — [0 0 0 0 1]
- We've now converted the words into vectors, and we can feed them into our network.

Auto encoders

- Auto encoders are an unsupervised learning technique which fall under representational learning and are used to find a compressed representation of the inputs.
- They are composed of an encoder and a decoder.



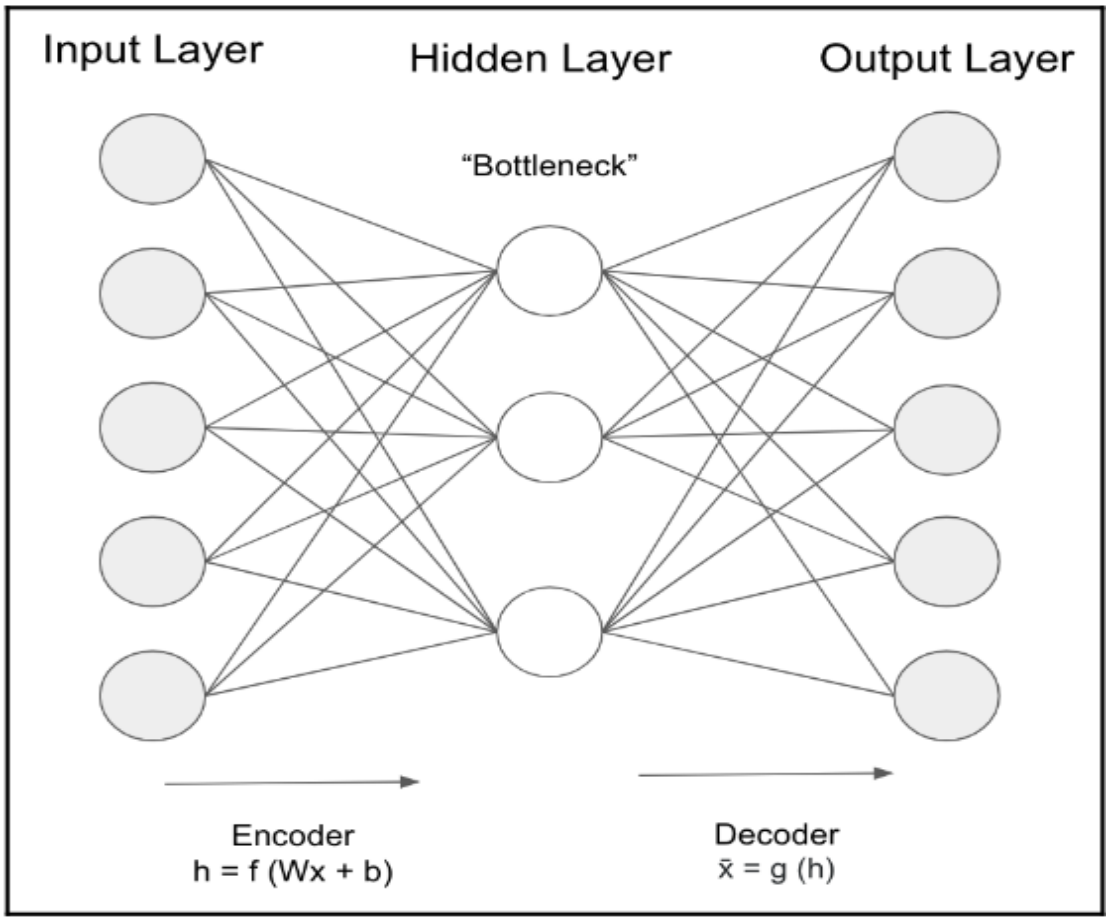
Applications of autoencoders

- Examples of applications of autoencoders include the following:

1) Data denoising.
2) Dimensionality reduction for data visualization.
3) Image generation.
4) Interpolating text.

Autoencoders

- Autoencoders impose a bottleneck on the network, enforcing a compressed knowledge representation of the original input.

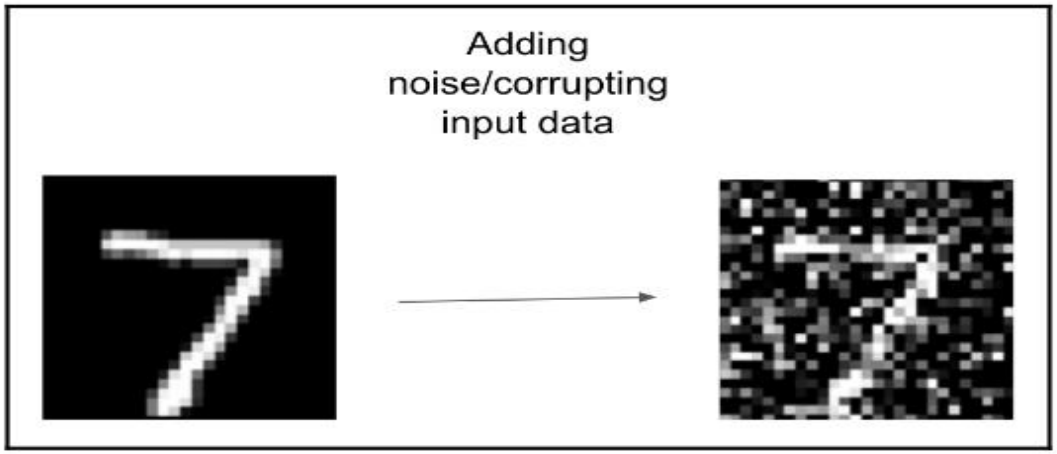


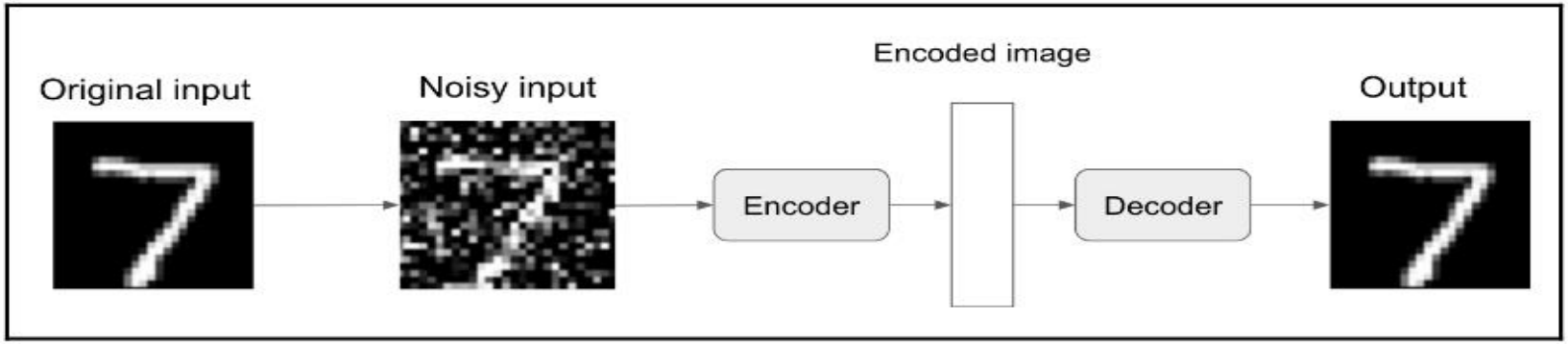
Autoencoders

- **Convolutional autoencoders**: Autoencoders can be used with convolutions instead of fully connected layers. This can be done using 3D vectors instead of 1D vectors. In the context of images, downsampling the image forces the autoencoder to learn a compressed version of it.
- **Denoising autoencoders:** Denoising encoders deliberately add noise to the input of the network. These autoencoders essentially create a corrupted copy of the data. In doing so, this helps the encoder to learn the latent representation in the input data.

Denoising autoencoders

- Noises were added in the original input and the encoder encodes the input and sends it to the decoder, which then decodes the noisy input into the cleaned output.



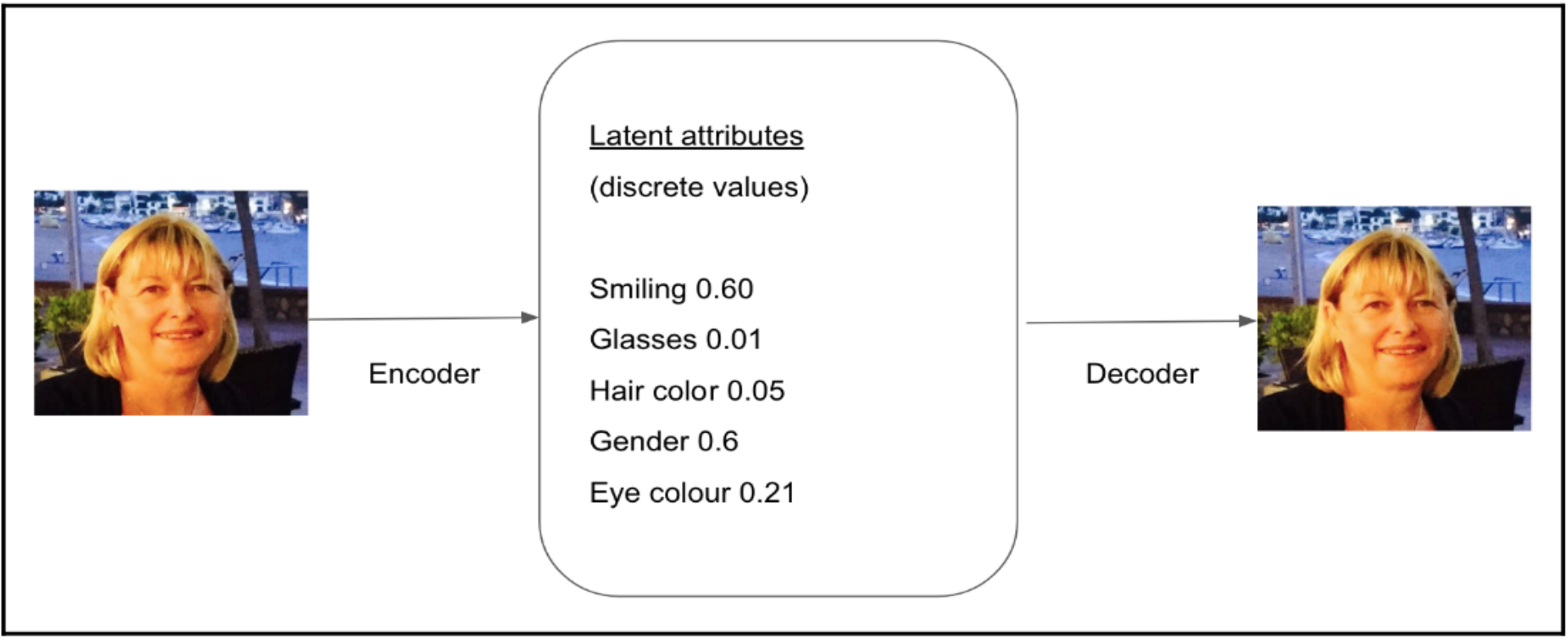


Variational autoencoders (VAEs)

- VAEs describe an observation in latent space in a probabilistic manner rather than deterministic.
- A probability distribution for each latent attribute is output, rather than a single value.
- If we allow each feature to be within a range of possible values rather than a single value, VAEs can be used to describe the attributes in probabilistic terms.
- The distribution of each latent attribute in VAEs is sampled from the image in order to generate the vector that is used as the input for the decoder model.

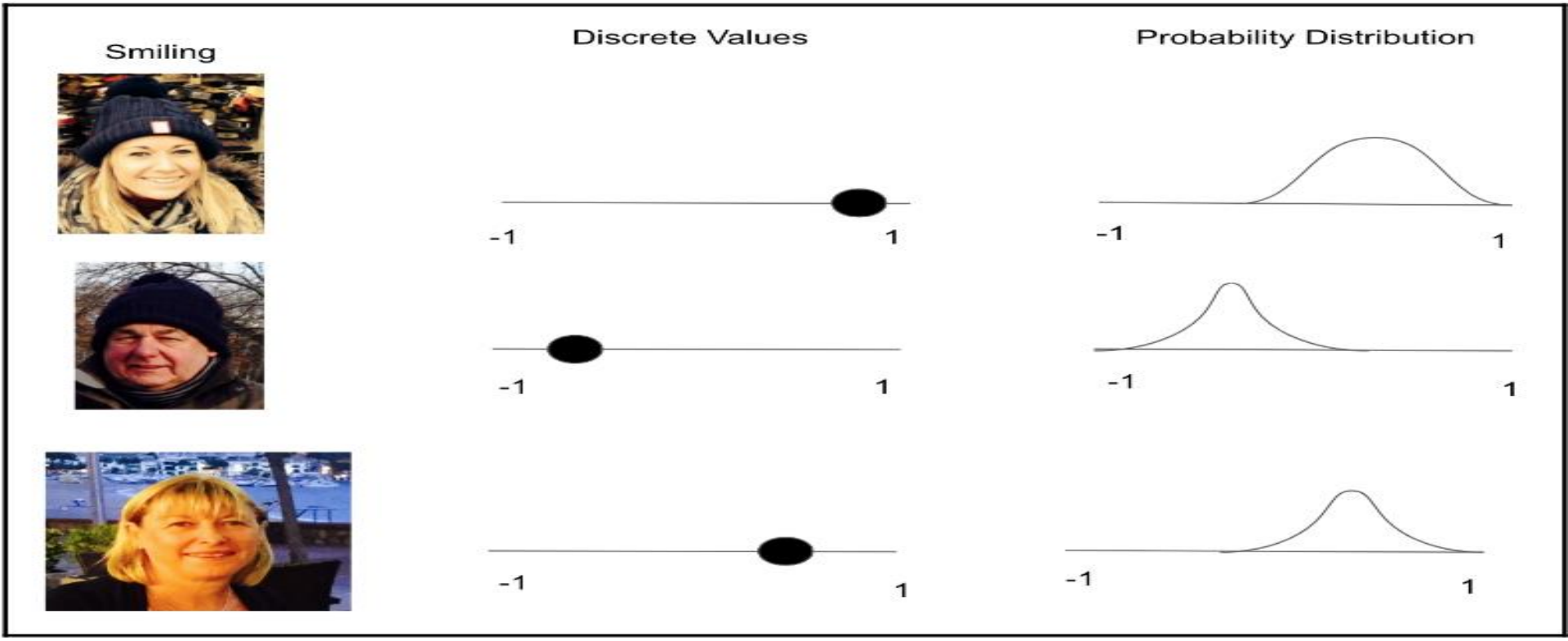
Variational autoencoders

- When training an autoencoder model on a dataset of face, it would learn latent attributes, such as whether the person is smiling, their skin tone, whether they are wearing glasses, and more.



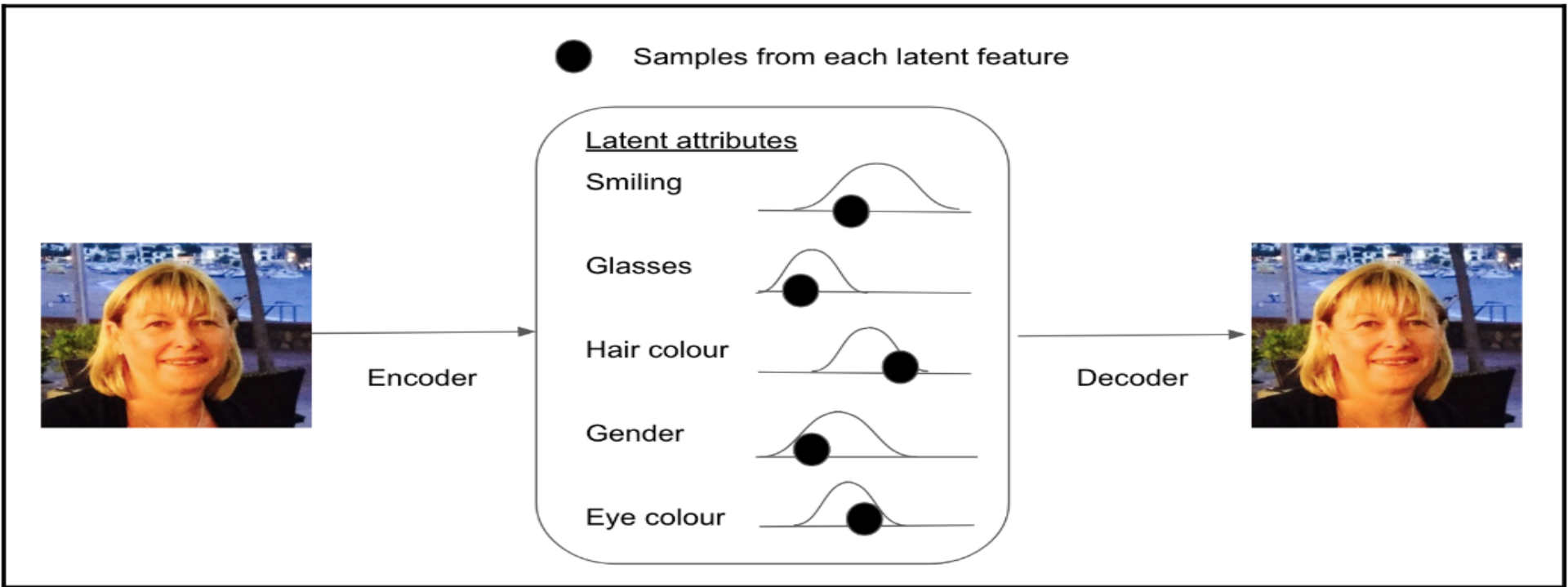
Variational autoencoders

- Standard autoencoders represent these latent attributes as discrete values.
- If we allow each feature to be within a range of possible values rather than a single value. we can use VAEs to describe the attributes in probabilistic terms.

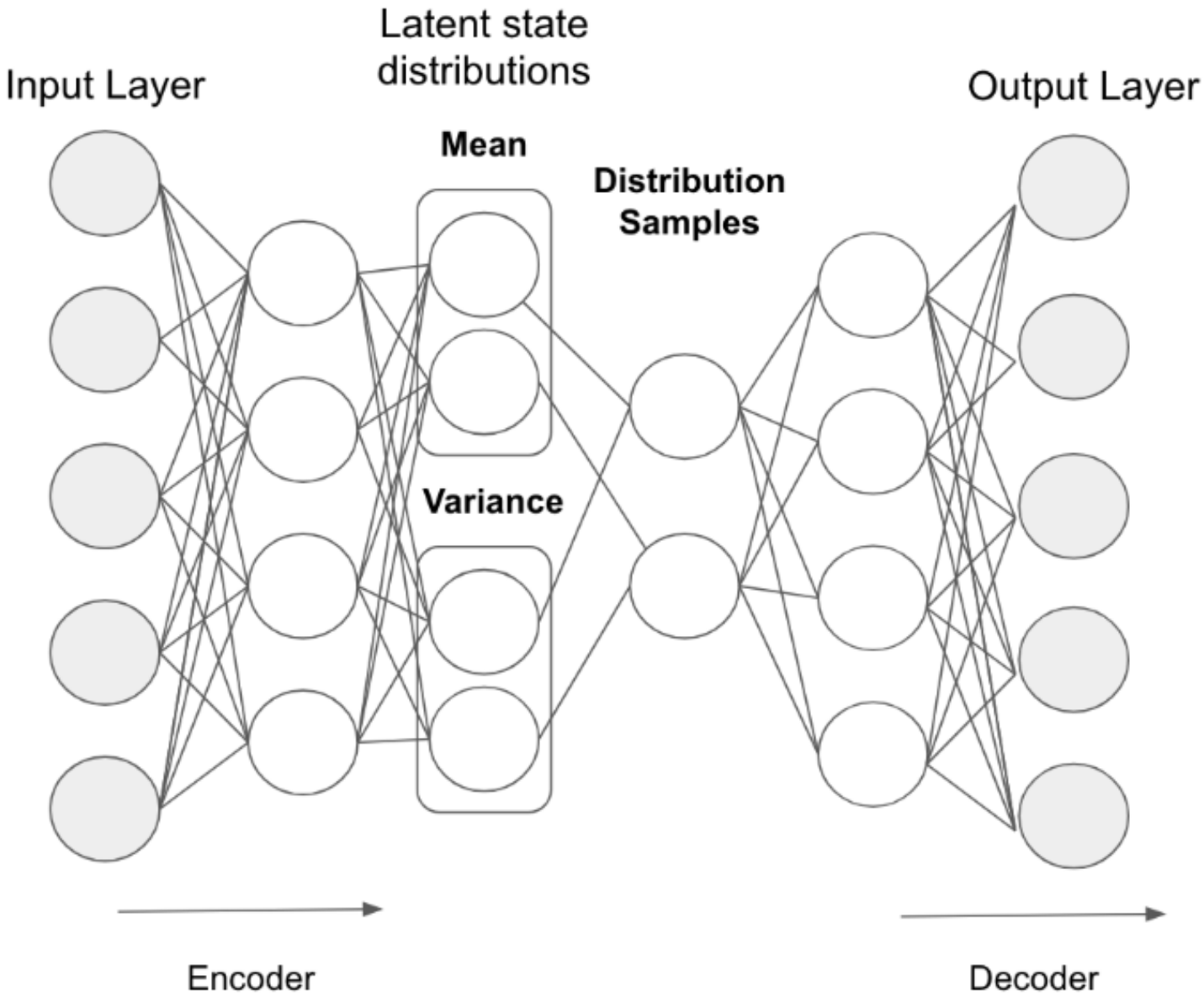


Variational autoencoders

- We can use VAEs to describe the attributes in probabilistic terms.
- The distribution of each latent attribute is sampled from the image in order to generate the vector that is used as the input for the decoder model.

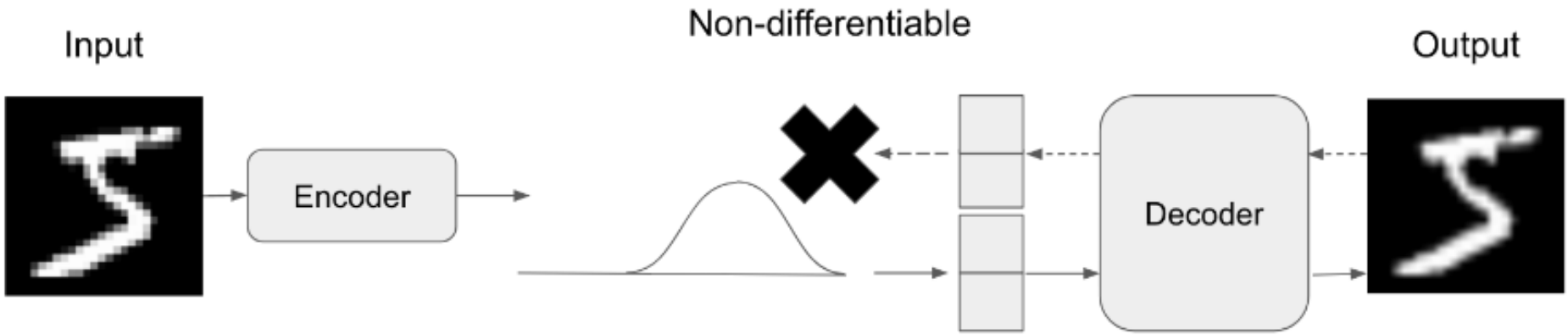


Variational autoencoders



Training VAEs

- Standard autoencoders use back propagation in order to reconstruct the loss value across the weights of the network.
- As the sampling operation in VAEs is not differentiable, the gradients cannot be propagated from the reconstruction error.

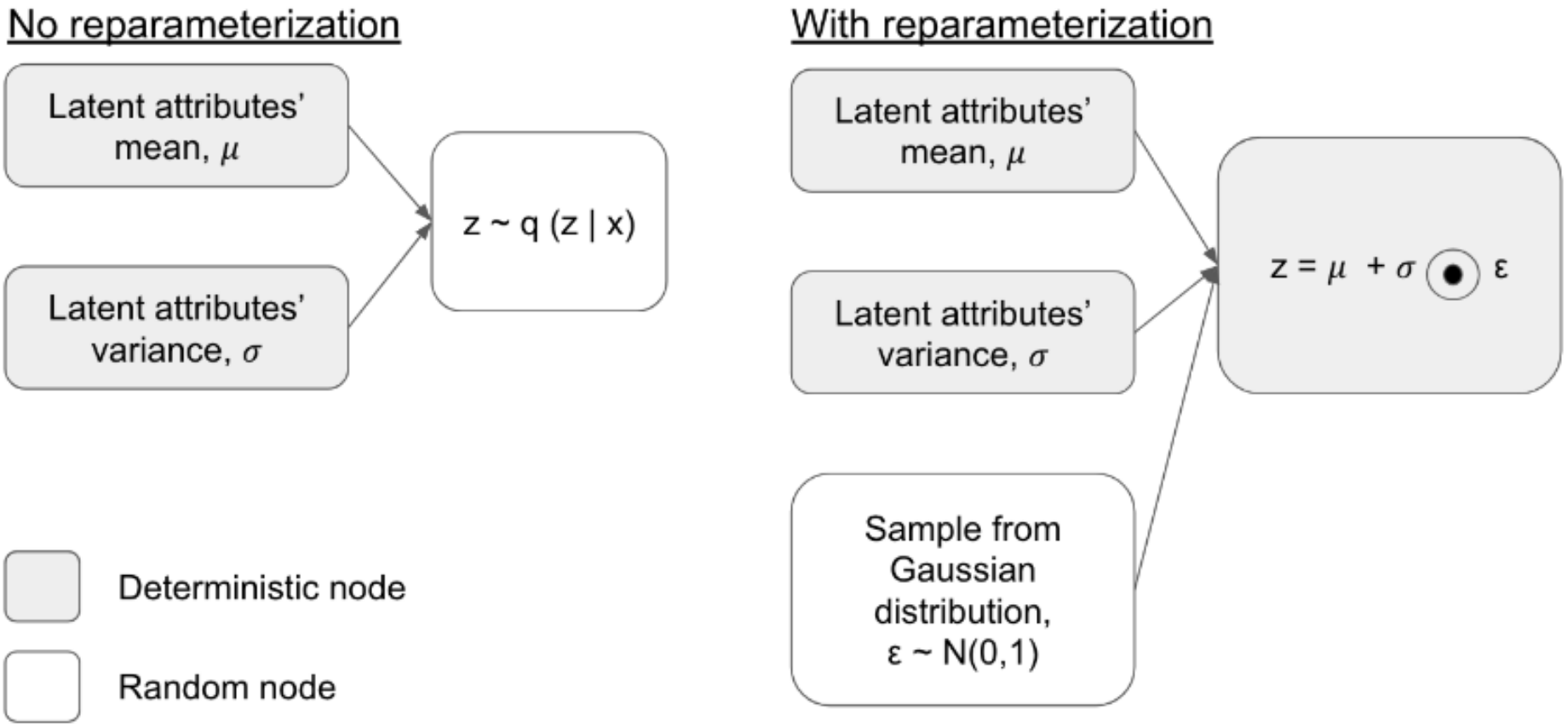


Training variational autoencoders

- Standard autoencoders use backpropagation in order to reconstruct the loss value across the weights of the network.
- As the sampling operation in VAEs is not differentiable, the gradients cannot be propagated from the reconstruction error.
- Hence reparameterization trick is used where samples are taken from Normal distribution($\varepsilon \sim N(0,1)$)with latent variable's mean μ and latent variable's variance σ as $z = \mu + \sigma\varepsilon$.

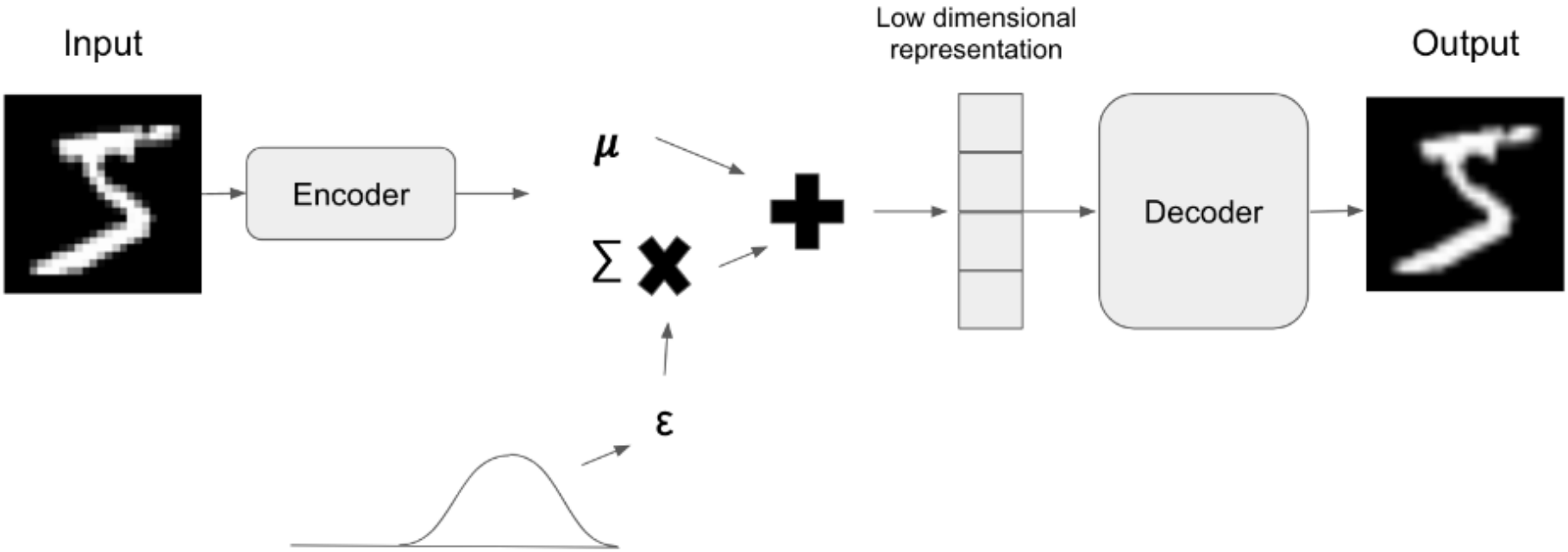
Variational autoencoders

- We can now optimize the parameters of the distribution while maintaining the ability to randomly sample from it.



Variational autoencoders

- We can now train the model using simple backpropagation with the introduction of the reparameterization trick.



Objective function of VAE

- **VAEs optimize two primary loss functions:**
- **Reconstruction Loss**: This loss ensures that the images generated by the decoder closely resemble the input images.
- It's typically computed using the Mean Squared Error (MSE) between the original and reconstructed images.

$$L_{\text{MSE}}(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} (x_i - f_\theta(g_\phi(x_i)))^2$$

- $L_{\text{MSE}}(\theta, \phi)$ represents the reconstruction loss
- θ and ϕ are the parameters of the decoder and encoder, respectively.

Objective function of VAE

- N is the number of samples.
- X_i is the original input image.
- f_θ is the decoder function, and g_Φ is the encoder function.
- The formula calculates the squared difference between each original image x_i and its corresponding reconstructed image $f_\theta(g_\Phi(x_i))$, then averages these squared differences over all samples.

Objective function of VAE

- **KL Divergence**: This measures the difference between the encoder's distribution and a standard normal distribution.
- It is a regularizer, ensuring the latent variables are close to a standard normal distribution.
- It encourages the model to maintain a structured and continuous latent space, which is particularly beneficial for generative tasks. It is given by

$$L_{\mathrm{KL}}[G(Z_\mu, Z_\sigma)|\mathcal{N}(0,1)] = -0.5 * \sum_{i=1}^{N} 1 + \log(Z_{\sigma_i^2}) - Z_{\mu_i}^2 - Z_{\sigma_i}^2$$

Objective function of VAE

- L_{KL} represents the KL divergence loss.
- $G(Z_\mu, Z_\sigma)$ is the Gaussian distribution defined by the encoder's outputs (mean) and (standard deviation).
- $\mathcal{N}(0, 1)$ is the standard normal distribution.
- The formula calculates the difference between the encoder's distribution and the standard normal distribution for each sample and sums these differences.

Restricted Boltzmann machines(RBM)

- An RBM is an algorithm that has been widely used for tasks such as collaborative filtering, feature extraction, topic modeling, and dimensionality reduction.
- They can learn patterns in a dataset in an unsupervised fashion.
- For example, if you watch a movie and say whether you liked it or not, we could use an RBM to help us determine the reason why you made this decision.

Restricted Boltzmann machines(RBM)

- The goal of RBM is to minimize energy defined by the following formula, which depends on the configurations of visible/input states, hidden states, weights, and biases:

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{ij} v_i h_j w_{ij}$$

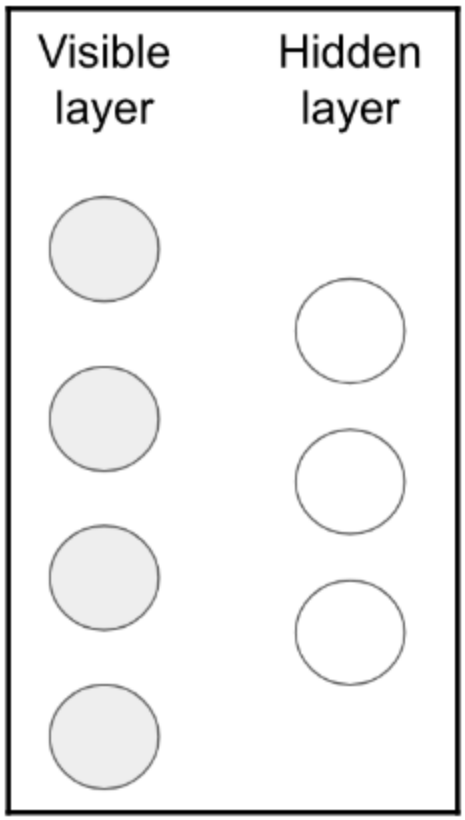
- RBMs are two-layer networks that are the fundamental building blocks of a Deep Belief Network (DBN).
- The first layer of an RBM is a visible/input layer of neurons and the second is the hidden layer of neurons:

Restricted Boltzmann machines(RBM)

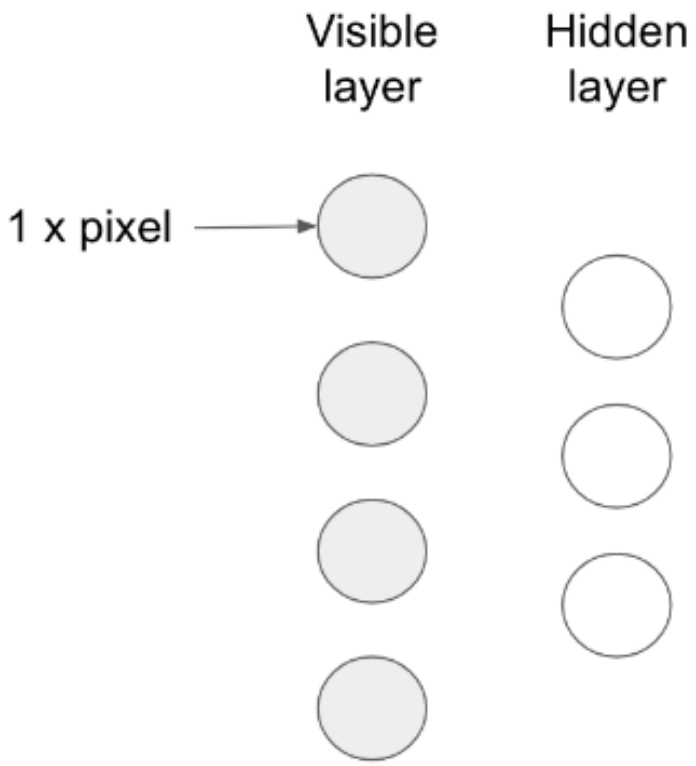
- The RBM translates the inputs from the visible layer and translates them into a set of numbers. Through several forward and backward passes, the number is then translated back to reconstruct the inputs.
- The restriction in the RBM is such that nodes in the same layer are not connected.
- A low-level feature is fed into each node of the visible layer from the training dataset.

Restricted Boltzmann machines(RBM)

- In the case of image classification, each node would receive one pixel value for each pixel in an image.

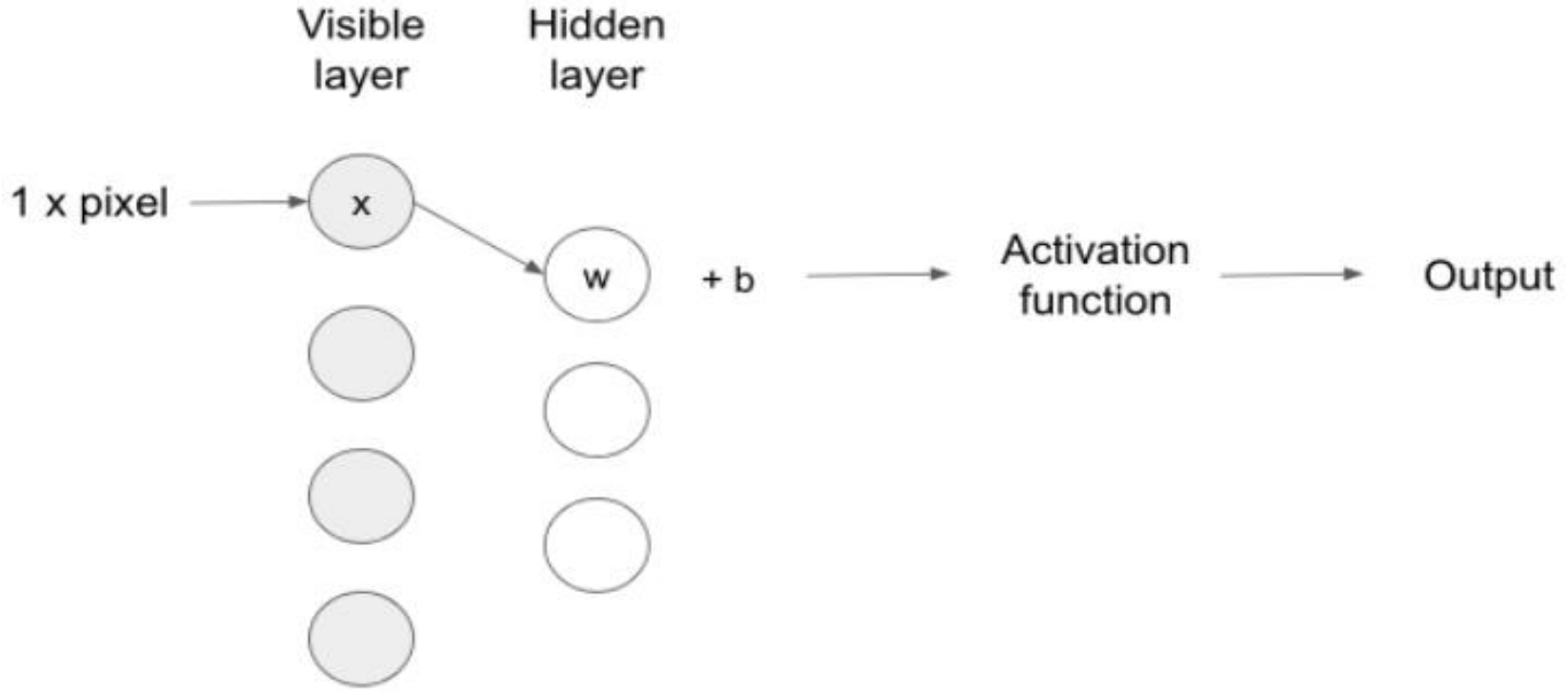


RBM



Restricted Boltzmann machines(RBM)

- Following one pixel through the network, the input x is multiplied by the weight from the hidden layer and a bias is then added. This is then fed into an activation function, which produces the output.



Activation Function ((input x * weight w) + bias b) = Output

Restricted Boltzmann machines(RBM)

- At each node in the hidden layer, x from each pixel value is multiplied by a separate weight.
- The products are then summed, and a bias is added. The output of this is then passed through an activation function, producing the output at that single node.
- At each point in time, the RBM is in a certain state, which refers to the values of the neurons in the visible v and hidden h layers.

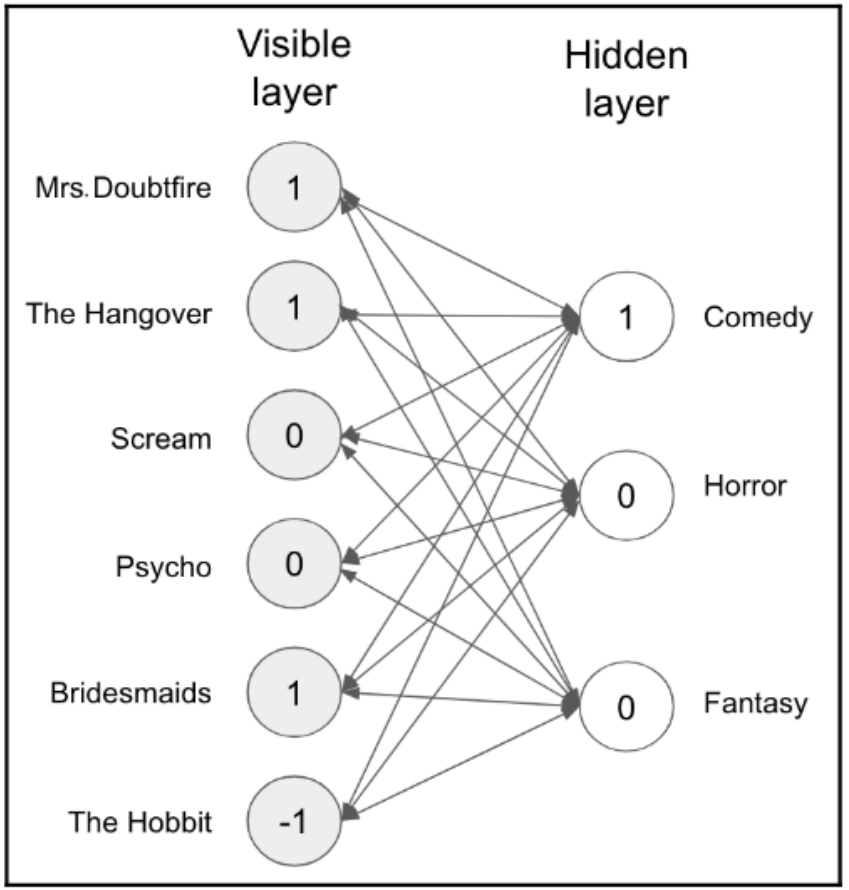
Restricted Boltzmann machines(RBM)

- The probability of such a state can be given by the following joint distribution function:

$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}}$$

RBM recommender system

- In the context of movies, we can use RBMs to uncover a set of latent factors that represent their genre, and consequently determine which genre of film a person likes.
- For each hidden neuron, the RBM assigns a probability of the hidden neuron given the input neuron.



Movie Recommender System using RBM

- The objective here is to create a Recommender System with a binary output YES/NO to predict whether or not the viewer will like a particular movie.
- GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org).
- It has 1 million ratings from 6000 users on 4000 movies. Released in FEB/2003.

RBM recommender system

- Movie data:

	0	1	2
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

- Users data:

	0	1	2	3	4
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

Ratings:

	0	1	2	3
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

RBM recommender system

- Importing the Training and Testing set.
- Converting the data frame to array.
- Getting total number of Users and Movies.
- Convert the training and test set into a matrix where the lines will be the Users and the columns are going to be Movies and cells are going to be Ratings.
- In each of these matrices, each cell of index Ui (U is the user and i is the movie) will get the rating for the movie i by the user U.

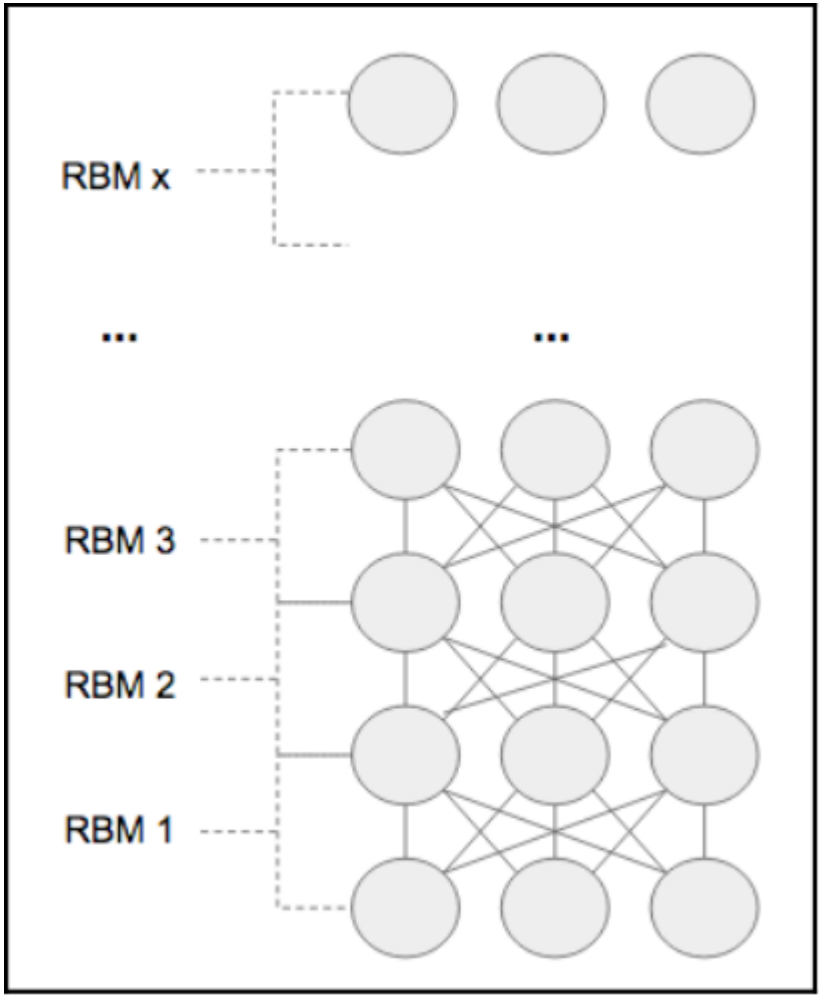
RBM recommender system

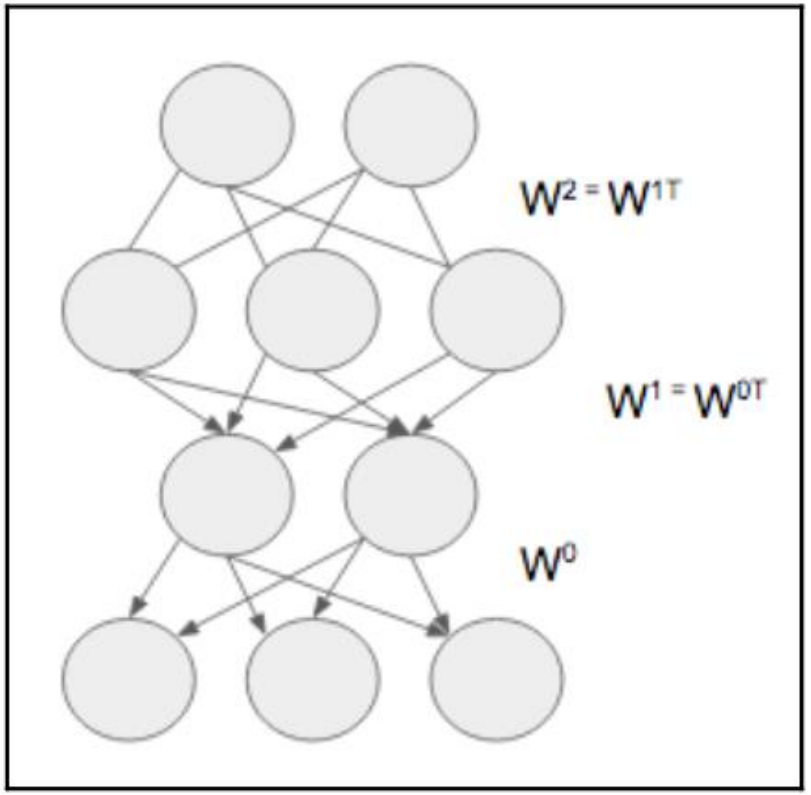
- We are building a recommender system to predict whether a person will like the movie or not.

Deep belief network (DBN) architecture

- A DBN is a multilayer belief network where each layer is an RBM stacked against one another.
- Apart from the first and final layers of the DBN, each layer serves as both a hidden layer to the nodes before it and as the input layer to the nodes that come after it.
- Two layers in the DBN are connected by a matrix of weights.

Deep belief network (DBN) architecture

- 



Deep belief network (DBN) architecture

- The top two layers of a DBN are undirected, which gives a symmetric connection between them, forming an associative memory.
- The lower two layers have direct connections to the layers above. The sense of direction converts associative memory into observed variables.

Deep belief network (DBN) architecture

- The two most significant properties of DBNs:

1] A DBN learns top-down, generative weights via an efficient, layer-by-layer procedure. These weights determine how the variables in one layer depend on the layer above.

2] Once training is complete, the values of the hidden variables in each layer can be inferred by a single bottom-up pass. The pass begins with a visible data vector in the lower layer and uses its generative weights in the opposite direction.

Deep belief network (DBN) architecture

- Once the pretraining phase of the DBN has been completed by the RBM stack, a feed forward network can then be used for the fine-tuning phase in order to create a classifier or simply help cluster unlabeled data in an unsupervised learning scenario.