# Natural Language Processing

## Chapter 8 Transfer Learning and Transformer Technology

DR RAYMOND LEE
ASSOCIATE PROFESSOR, DST
BNU-HKBU UNITED INTERNATIONAL COLLEGE

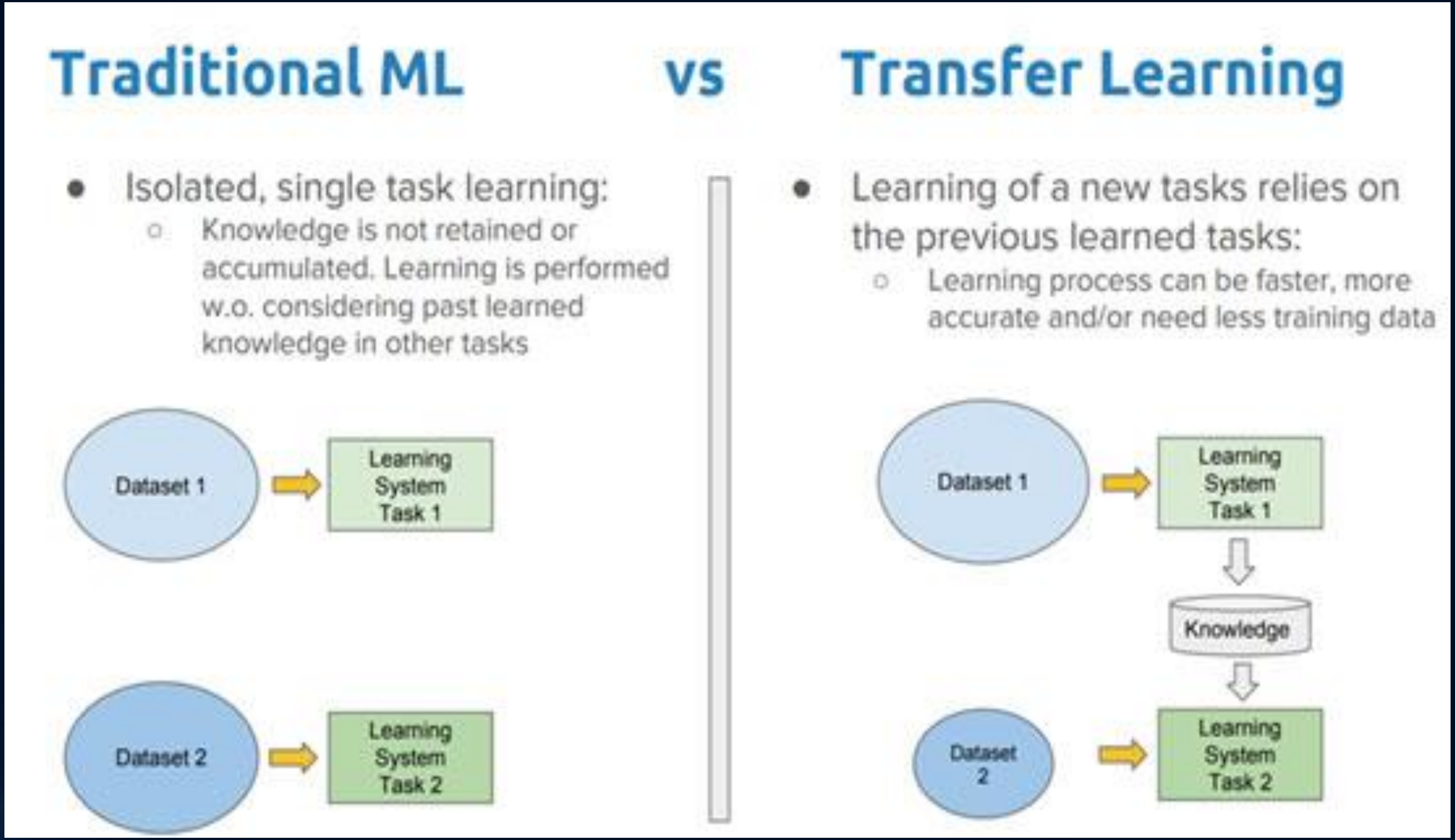
## Transfer Learning and Transformer Technology

## 8.1 What is Transfer Learning?

## What is Transfer Learning?

- Transfer learning focuses in solving a problem from acquired knowledge and applying such knowledge to solve another related problem(s).
- It is like two students learn to play guitar.
- One has musical knowledge and the other has not.
- It is natural for the one to transfer background knowledge to the learning process.
- Every task has its isolated datasets and trained model in traditional ML, whereas learning a new task in TL relies on previous learned tasks to acquire knowledge with larger datasets as shown in Fig 8.1.

## What is Transfer Learning?


Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data

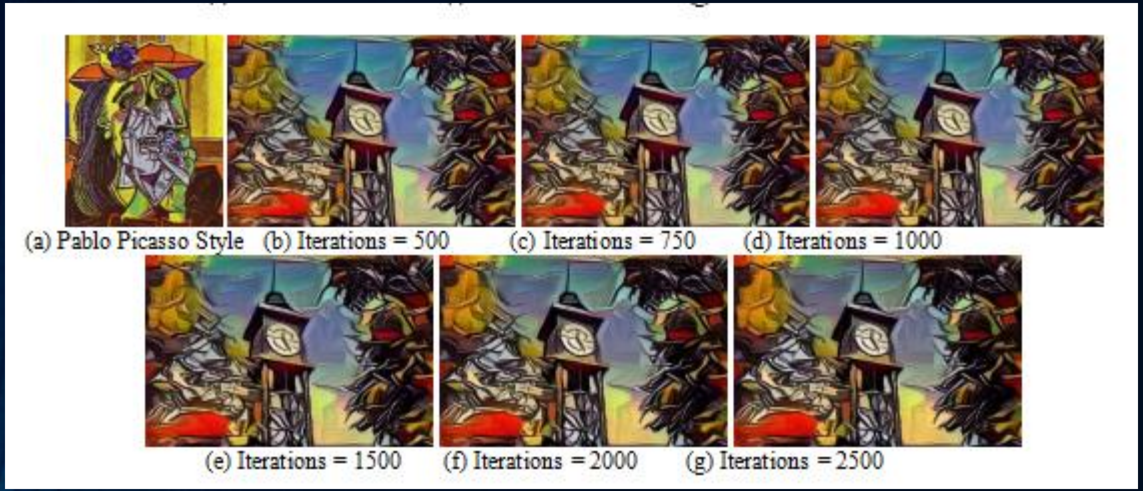
**Fig. 8.1** Traditional machine learning vs Transfer learning

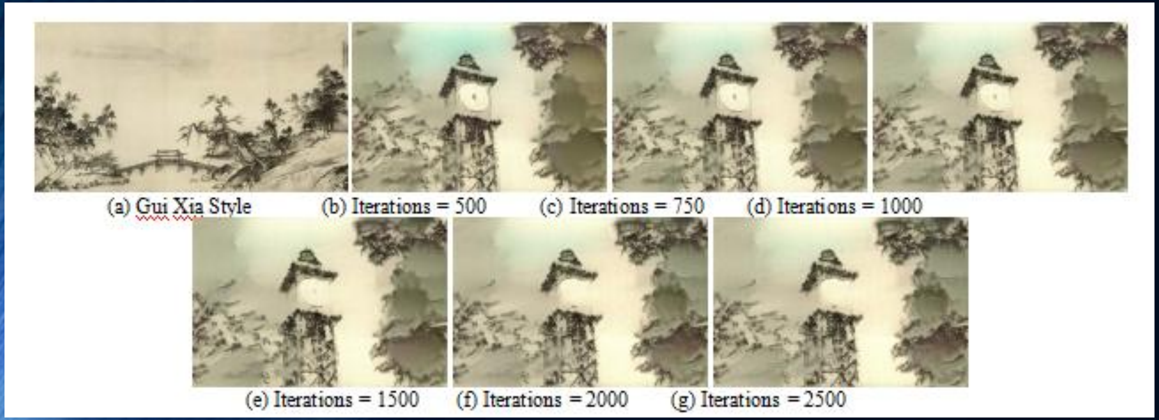
## 8.2 Motivation of Transfer Learning

## Motivation of Transfer Learning

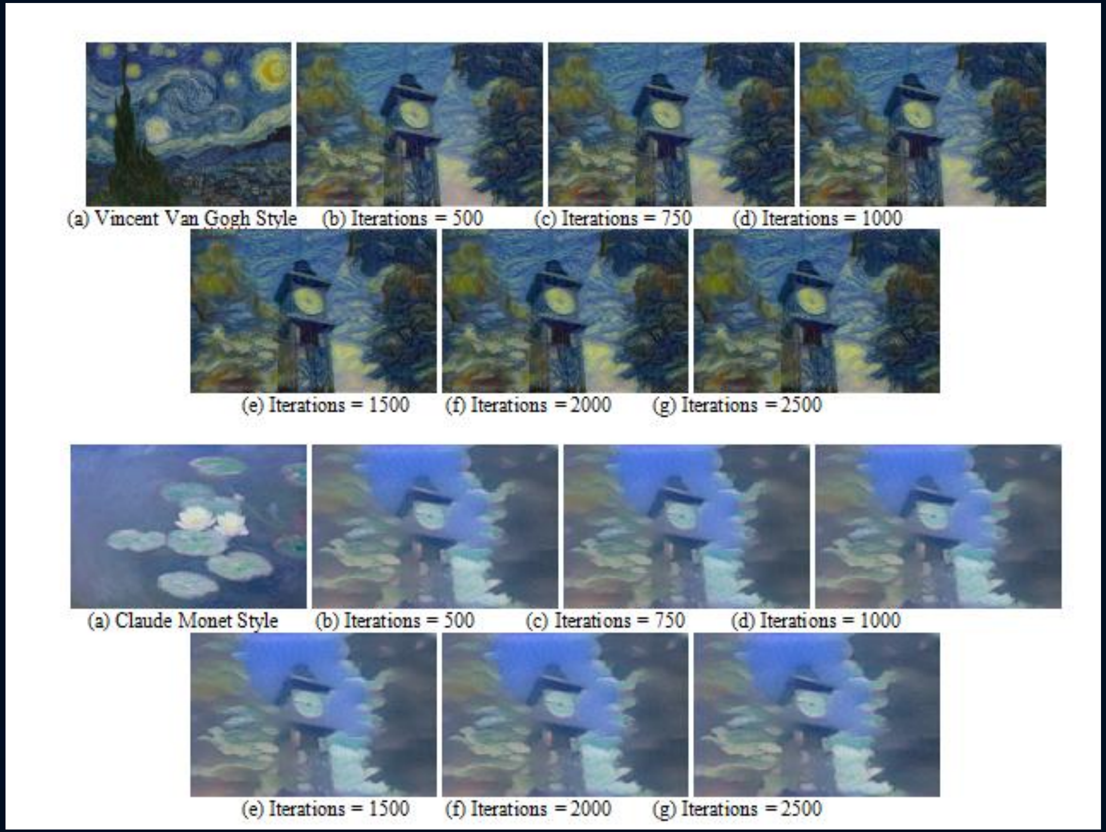
- Traditional ML datasets and trained model parameters cannot reuse. They involve enormous, rare, inaccessible, time-consuming, and costly training process in NLP tasks and computer vision.
- For example, if a task is text sentiment reviews predictions on laptops, there are large amounts of labeled data, target data and training data from these reviews.
- Traditional ML can work well on correlated domains, but when there are large amounts of target data like food reviews, the inference results will be unsatisfactory due to domains differences.
- Nevertheless, these domains are correlated in some sense to bear same domain reviews as language characteristics and terminology expressions, which makes TL possible to apply in a high-level approach on prediction task.
- This approach enables source domains to become a target domain and determine its sub-domains correlations as shown in Fig. 8.2.
- TL has been implemented to several machine learning applications such as image and text sentiment classifications.

# Transfer Learning in Painting





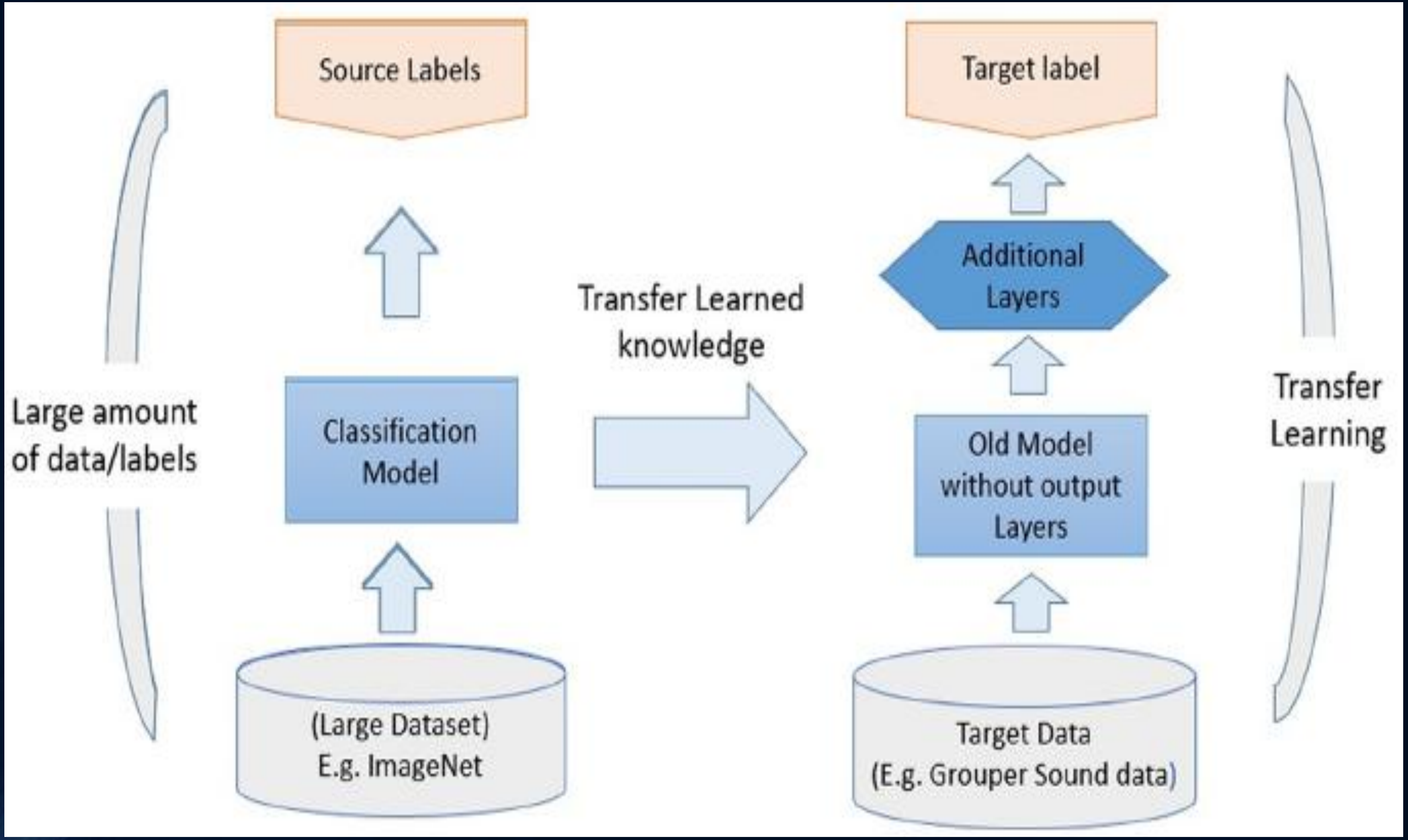
Transfer learning from Chinese painting style



Transfer learning from Van Gogh (star night) style

Nuobei Shi, Zhuohui Chen, Ling Chen, Raymond S.T. Lee* (2024), ReLU-oscillator: Chaotic VGG10 model for real-time neural style transfer on painting authentication, Expert Systems with Applications, Volume 255, Part A, 2024, 124510, https://doi.org/10.1016/j.eswa.2024.124510.

# Motivation of Transfer Learning



Source Labels
Large amount of data/labels
Classification Model
(Large Dataset) E.g. ImageNet
Transfer Learned knowledge
Target label
Additional Layers
Old Model without output Layers
Target Data (E.g. Grouper Sound data)
Transfer Learning

**Fig. 8.2** Transfer learning

## Categories of Transfer Learning

- Domain is to be assigned with a definition by feature space *X* and marginal probability distribution *P(X)* where $X=\{x_1, x_2, x_3, \dots, x_n\} \in X$.
- If a feature space *X* and distribution *P(X)* between two domains are different, they are different domains.
- If a task is defined by a label space *Y* with a predictive function *f( ·)*, *f( ·)* is represented by a conditional probability distribution given by (8.1):

$$f(x_i) = P(y_i|x_i) \tag{8.1}$$

- If a function *f( ·)* and label space Y between two tasks are different, they are different tasks.
- Now TL can give a new representation by above definitions that have $D_s$ as source domain and $T_s$ as source learning task. $D_t$ represents target domain and $T_t$ represents target learning task.
- Given two domains are unidentical or has two different tasks, TL aim is to improve the results $P(Y_t | X_t)$ of $D_t$ when $T_s$ and $D_s$ knowledge can be obtained.

## Heterogeneous vs Homogeneous Transfer Learning

- There are two types of TL 1) heterogeneous and 2) homogeneous as shown in Fig. 8.3.
- **Heterogeneous transfer learning:** When source feature space and feature space are different which means that $Y_t \neq Y_s$ and/or $X_t \neq X_s$. Under the condition of same domain distributions, the strategy of resolution is to adjust feature space smaller and transform it to homogeneous so that the differences between marginal or conditional of source, and target domains will be reduced.
- **Homogeneous transfer learning:** When there are conditions $X_t = X_s$ and $Y_t = Y_s$, the difference of two domains lies on data distributions. Three strategies are commonly used to tackle homogenous TL problems: 1) reduction the differences of $P(X_t) \neq P(X_s)$, 2) reduction the differences of $P(Y_t|X_t) \neq P(Y_s|X_s)$ and 3) the combination of strategies 1) and 2).

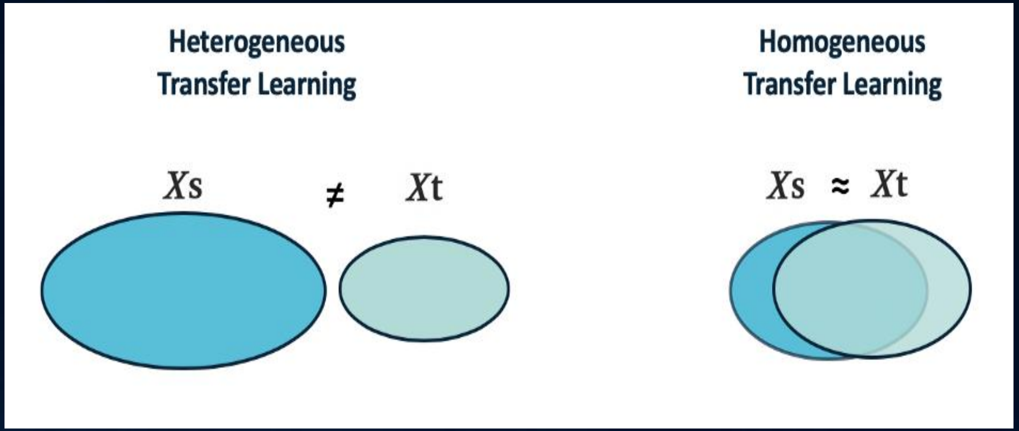

Heterogeneous Transfer Learning
Xs
≠
Xt
Homogeneous Transfer Learning
Xs
≈
Xt

**Fig. 8.3** Two categories of transfer learning

## 8.3 Solutions of Transfer Learning

## Solutions of Transfer Learning

- There are four methods to solve problems produced by homogeneous and heterogeneous TL: 1) instance-based, 2) feature-based, 3) parameter-based and 4) relational-based.
- **Instance-based**
  This method reweights samples from source domains and uses them as target domain data to bridge the gap of marginal distribution differences which works best when conditional distributions of two tasks are equal.
- **Feature-based**
  This method works for both heterogeneous and homogeneous TL problems. For homogeneous types is to bridge the gap between conditional and marginal distributions of target and source domains. For heterogeneous types is to reduce the differences between source and target features spaces.
- It has two approaches a) asymmetric and b) symmetric.

## Feature-based Transfer Learning

a) Asymmetric feature transformation aims to modify the source domain and reduce the gap between source and target instances by transforming one of the source and target domains to the other as shown in Fig. 8.4. It can be applied when $Y_s$ and $Y_t$ are identical.

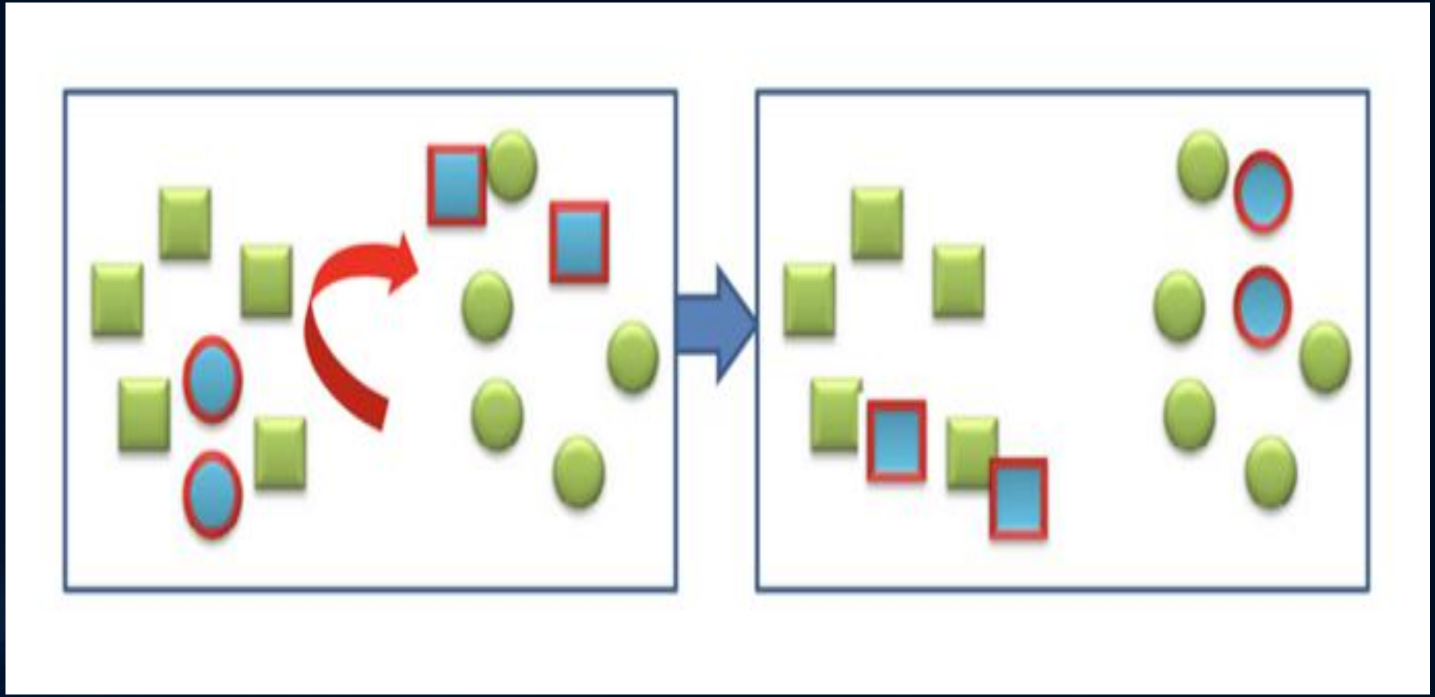


**Fig. 8.4** Asymmetric feature transformation

# Feature-based Transfer Learning

b) Symmetric feature transformation aims to transform source and target domains into their shared feature space, starting from the idea of discovering meaningful structures between domains. The feature space they share is usually low-dimensional. The purpose of this approach is to reduce the marginal distribution distance between destination and source. The difference between symmetric and symmetric feature transformation is shown in Fig. 8.5.

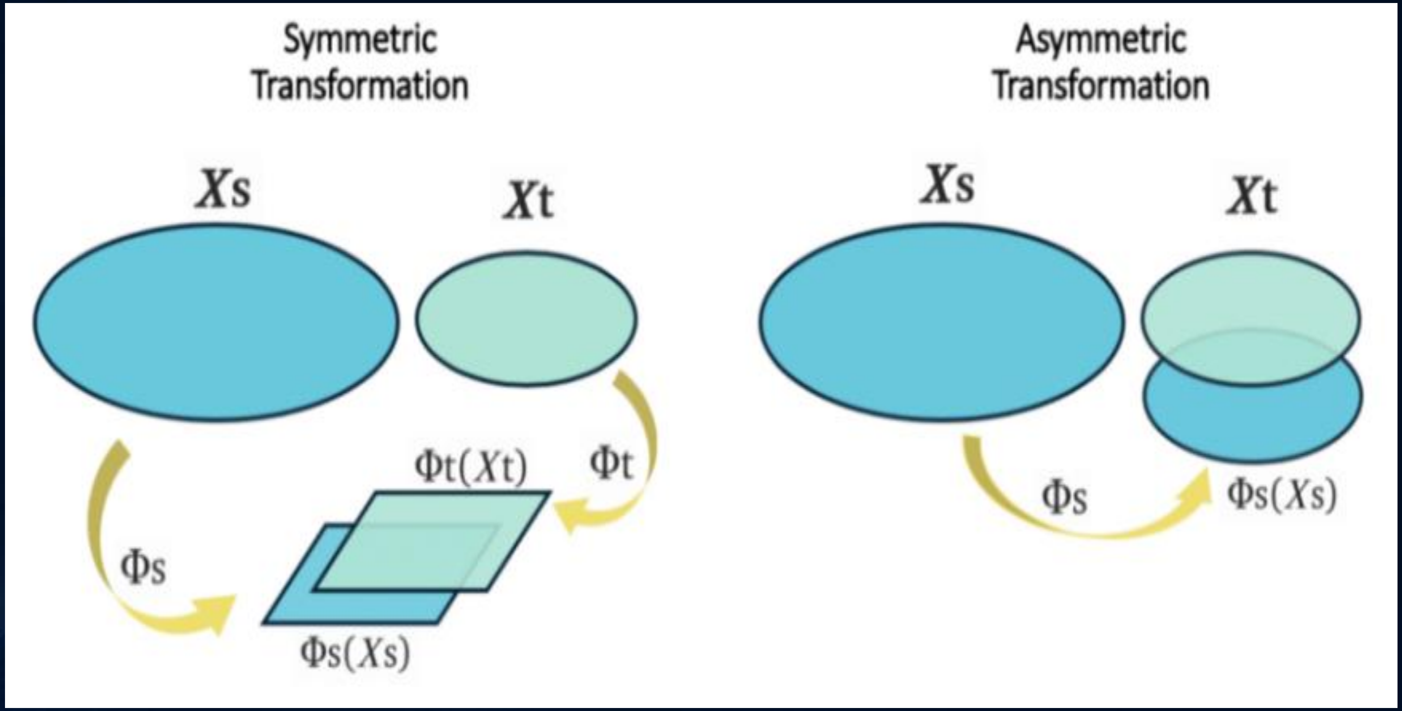


**Fig. 8.5** Symmetric Feature Transformation (left) and Asymmetric feature transformation (right)

## Parameter-based Transfer Learning

This method transfers learnt knowledge by sharing parameters common to the models of source and target learners. It applies to the idea that two related tasks have similarity in model structure. The trained model is transferred from source domain to target domain with parameters. This approach has a huge advantage because the parameters are usually trained from randomly initialized parameters as training process can be time-consuming for models trained from the beginning. This approach can train more than one model on the source data and combine parameters learnt from all models to improve results of the target learner. It is often used in deep learning applications as shown in Fig. 8.6.

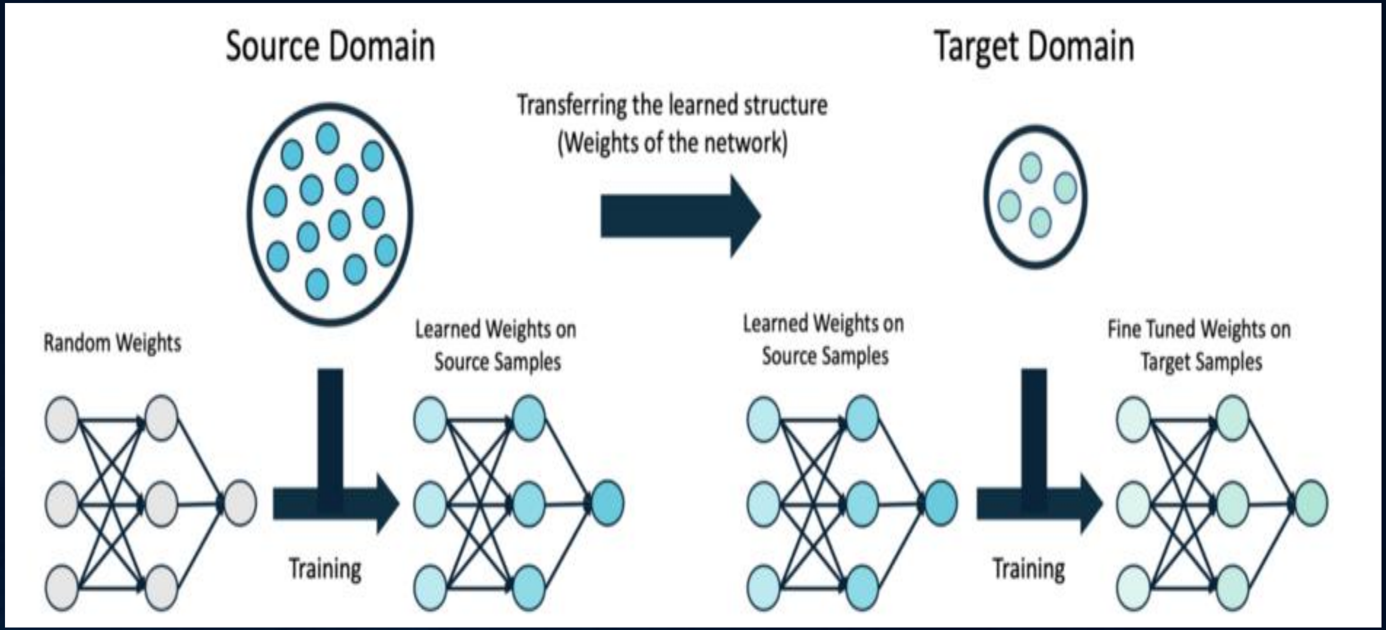
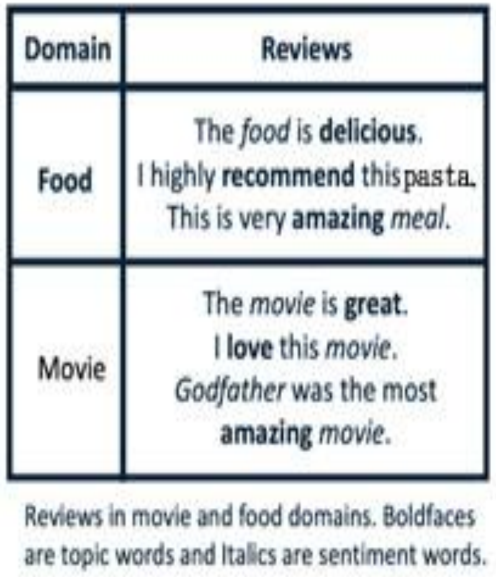

Source Domain
Target Domain
Transferring the learned structure (Weights of the network)
Random Weights
Training
Learned Weights on Source Samples
Learned Weights on Source Samples
Training
Fine Tuned Weights on Target Samples

**Fig. 8.6** Parameter-based methods

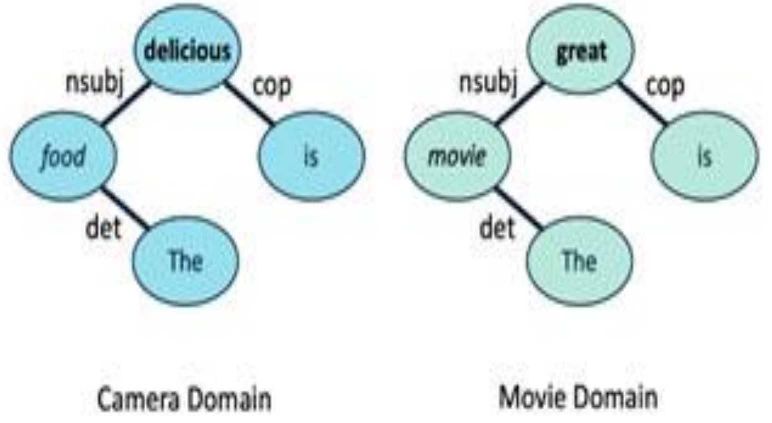
## Relational-based Transfer Learning

This method transfers learnt knowledge by sharing its learnt relations between different samples parts of source and target domains as shown in Fig. 8.7. Food and movie domains are a related domain example. Although the reviews texts are different, but sentence structures are similar. It aims to transfer learnt relations of different review sentences parts from these domains to improve text sentiment analysis results.



| Domain | Reviews |
|---|---|
| Food | The *food* is **delicious**. I highly **recommend** this pasta. This is very **amazing** *meal*. |
| Movie | The *movie* is **great**. I **love** this *movie*. *Godfather* was the most **amazing** *movie*. |

Reviews in movie and food domains. Boldfaces are topic words and Italics are sentiment words.



Dependency tree structure.

**Fig. 8.7** Relational-based approaches: an example of learning sentence structure of food reviews to help with movie reviews' sentiment analysis

## 8.4 Recurrent Neural Networks (RNN)

## Recurrent Neural Networks
### What is RNN?

- *Recurrent neural network (RNN)* is a class of artificial neural networks (ANNs) to consider time series or sequential data as input and use them prior inputs to produce current input and output.
- RNN has *memory* which means its output is influenced by prior elements of the sequence against traditional Feedforward Neural Network (FNN) with independent inputs and outputs as shown in Fig. 8.8.

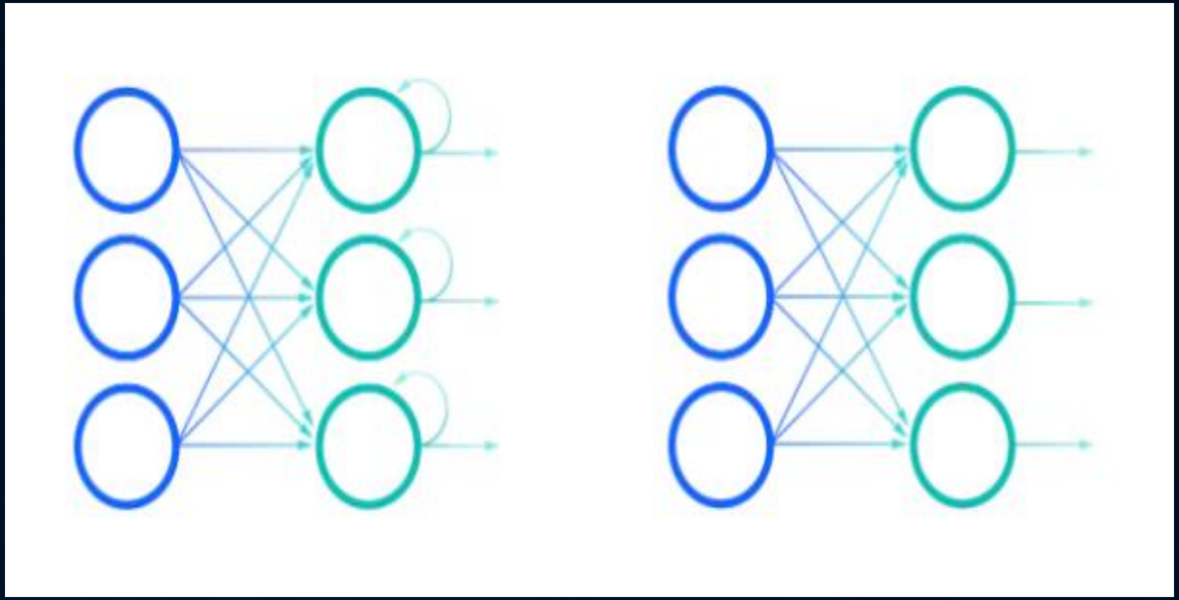


**Fig. 8.8** Recurrent Neural Network(left) vs. Feedforward Neural Network(right)

# Recurrent Neural Networks
## Motivation of RNNs

- There are many learning tasks required sequential data processing which include speech recognition, image captioning and synced sequence in video classification. Sentiment analysis and machine translation model outputs are sequences, but tasks inputs are time or space related that cannot be modeled by traditional neural networks to assume that test and training data are independent.
- For example, a language translation task aims to translate a phrase *feel under the weather* means *unwell*. This phrase makes sense only when it is expressed in that specific order. Thus, the positions of each word in sentence must be considered when model predicts the next word.
- There are five major categories of RNN architecture corresponding to different tasks:
  1) simple one to one model for image classification task,
  2) one to many for image captioning tasks,
  3) many to one model for sentiment analysis tasks,
  4) many to many models for machine translation, and
  5) complex many to many models for video classification tasks as shown in Fig. 8.9.

# Recurrent Neural Networks
## Major types of RNNs

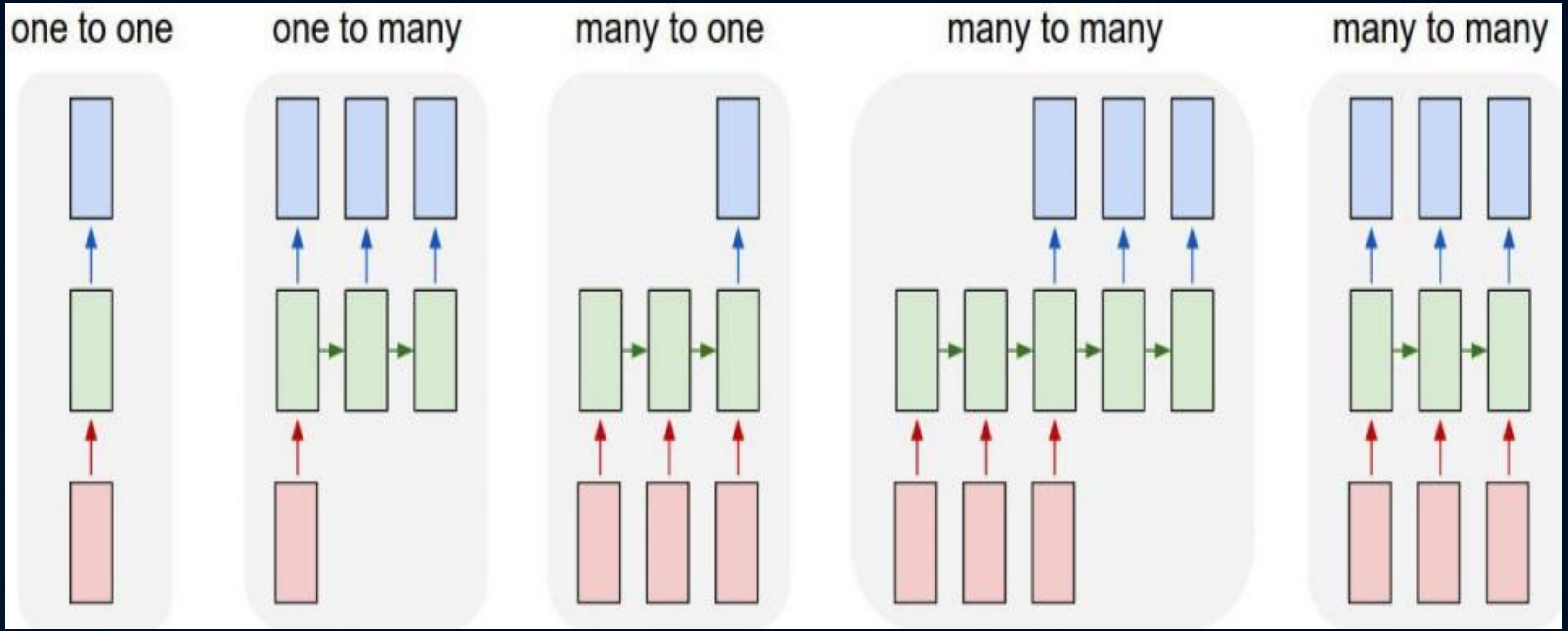


**Fig. 8.9** 5 major types of RNNs

## Recurrent Neural Networks
### RNN Architecture

- RNN is like standard neural networks consists of input, hidden and output layers as shown in Fig. 8.10.

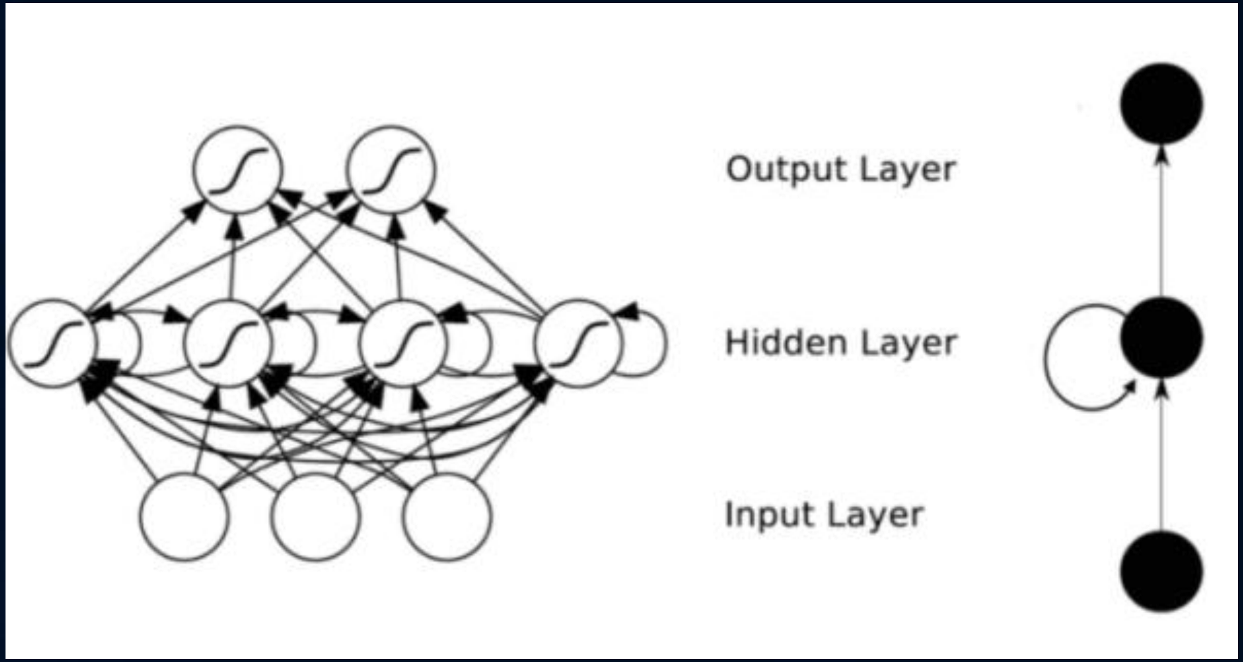


**Fig. 8.10** Basic architecture of RNN

## Recurrent Neural Networks
### RNN Architecture

- An unfolded RNN architecture is narrated by $x_t$ as the input at time-step $t$, $s_t$ stores the values of hidden units/states at time $t$ and $o_t$ is the output of network at time-step $t$. $U$ are weights of inputs, $W$s are weights of hidden units, $V$ is bias as shown in Fig. 8.11.

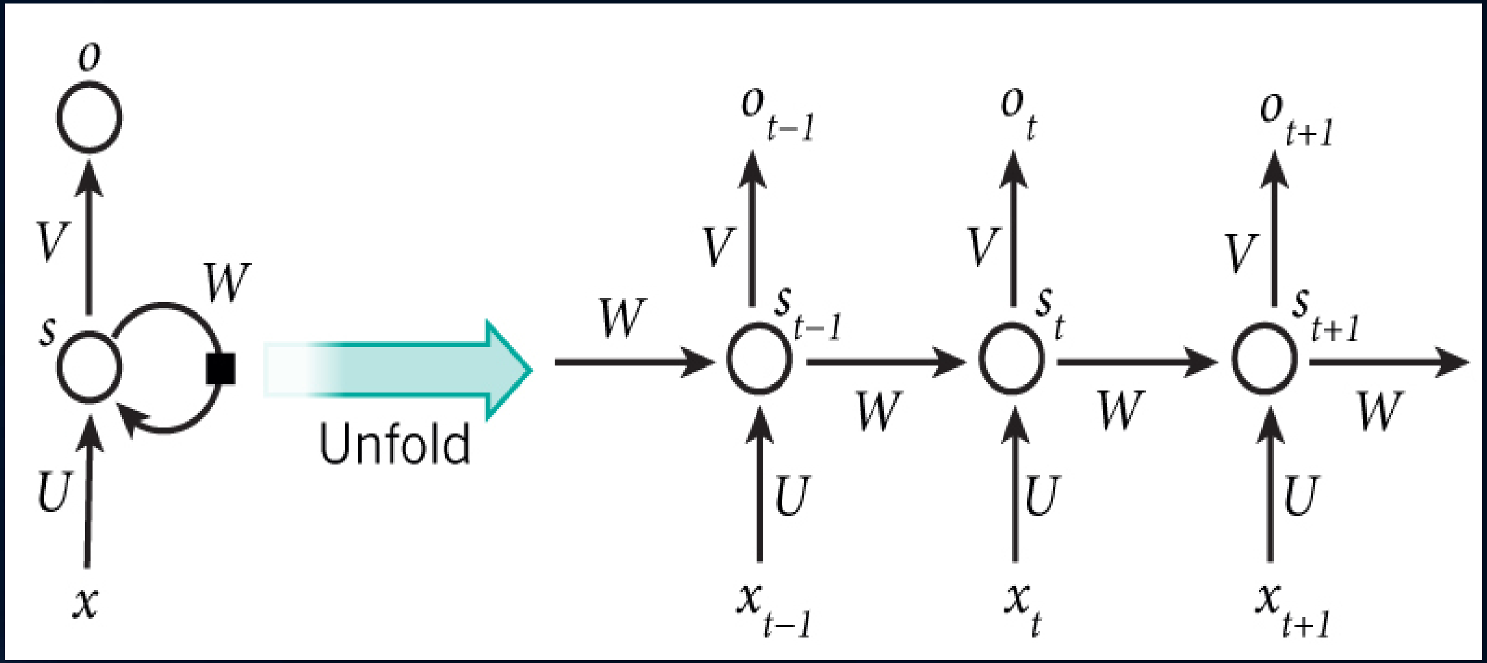


**Fig. 8.11** Unfolded RNN architecture

# Recurrent Neural Networks
## RNN Architecture

- with the activation function *f*, the hidden states $s_t$ is calculated by equation:

$$s_t = f(Ux_t + Ws_{t-1}) \tag{8.2}$$

- the output of each recurrent layer $o_t$ is calculated by equation:

$$o_t = softmax(Vs_t) \tag{8.3}$$

- The hidden states $s_t$ are considered as network memory units which consists of hidden states from several former layers. Each layer's output is only related to hidden states of the current layer. A significant difference between RNN and traditional neural networks is that weights and bias *U*, *W*, and *V* are shared among layers.
- There will be an output at each step of the network but unnecessary. For instance, if inference is applied for sentiment expressed by a sentence, only an output is required when the last word is input, and none after each word for input. The key to RNNs is the hidden layer to capture sequence information.
- For RNN feedforward process, if the number of time steps is *k*, then hidden unit values and output will be computed after *k+1* time steps. For backward process, RNN applies an algorithm called backpropagation through time (BPTT).

# Recurrent Neural Networks
## RNN Architecture

- RNN topologies range from partly to fully recurrent. Partly recurrent is a layered network with distinct output and input layers where recurrence is limited to the hidden layer. Fully recurrent (FRNN) connect all neurons' outputs to inputs as shown in Fig. 8.12.

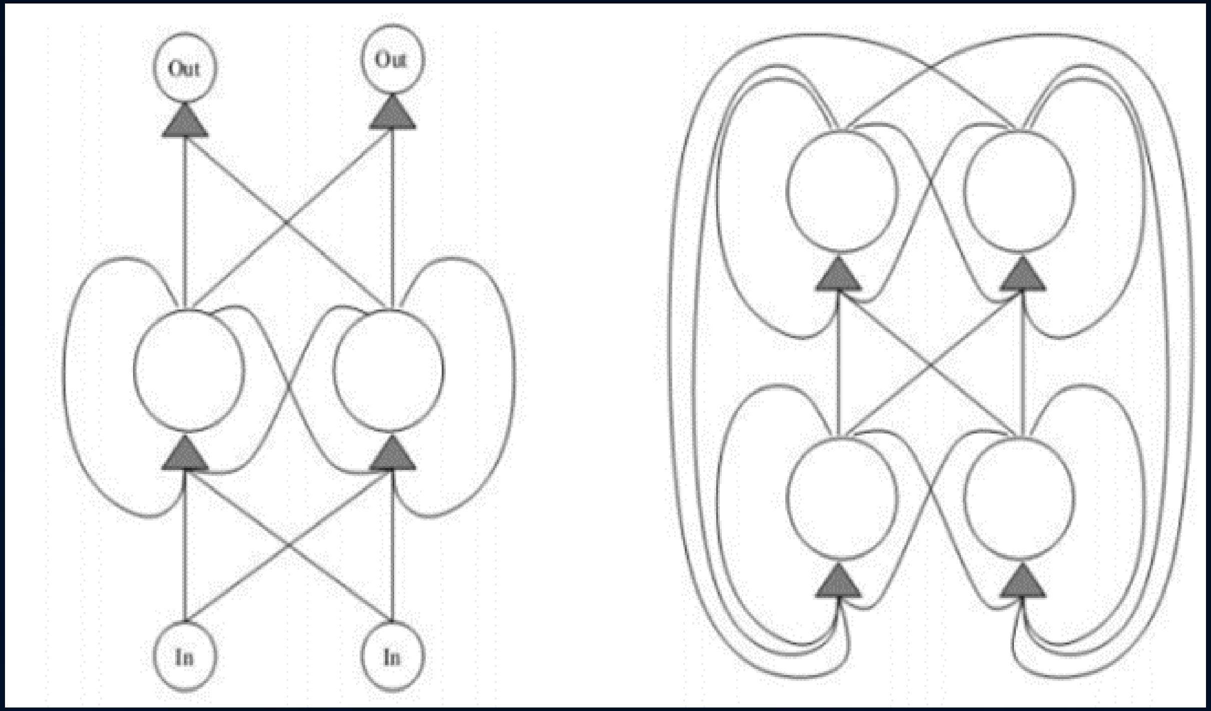


**Fig. 8.12** A simple recurrent neural network Right (left) and fully connected recurrent neural network (right)

## Long-Short Term Memory (LSTM) Network

### What is LSTM?

- Long short-term memory (LSTM) network (Staudemeyer and Morris, 2019; Yu et al., 2019) is a type of RNN with special hidden layers to deal with gradient explosion and disappearance problems during long sequence training process proposed by Hochreiter and Schmidhuber (1997). LSTM has better performance with training longer sequences against Naïve RNNs. Structure frameworks of LSTM and naïve RNN are shown in Fig. 8.13.
- LSTM has two hidden layers as RNN where a memory cell in the layer is to replace hidden node. RNN has only one transfer state $h^t$ as compared with RNN. There are two transfer states, $c^t$ (cell state) and $h^t$ (hidden state) in LSTM. RNN's $h^t$ corresponds to LSTM's $c^t$. $c^t$ passed down information among them, output $c^t$ is produced by adding $c^{t-1}$ passed from state and values of previous step. RNN's $h^t$ has larger difference among nodes usually.

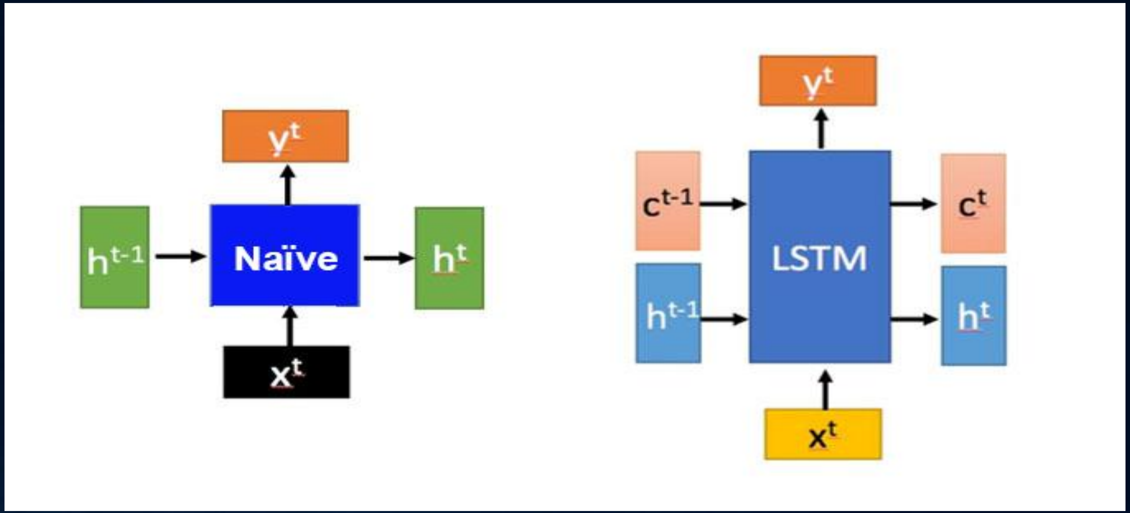


**Fig 8.13** Standard RNNs (left) and LSTM (right)

# Long-Short Term Memory (LSTM) Network
## LSTM Architecture

- $x^t$ and $h^{t-1}$ are concatenated inputs from the state of previous step to train with activations for four states as shown in Fig. 8.14.

- ***z*** is input calculated by multiplying the concatenate vector with weights ***w*** and converted into values *0-1* through activation function *tanh*. $\mathbf{z^f}$, $\mathbf{z^i}$, $\mathbf{z^o}$ are calculated by multiplying the concatenate vector with corresponding weights and converting to values *0-1* by a sigmoid function σ to generate gate states. $\boldsymbol{z^f}$ represents forget gate, $\boldsymbol{z^i}$ represents input gate, $\boldsymbol{z^o}$ represents output gate. A memory cell of LSTM calculation is shown in Fig. 8.15.

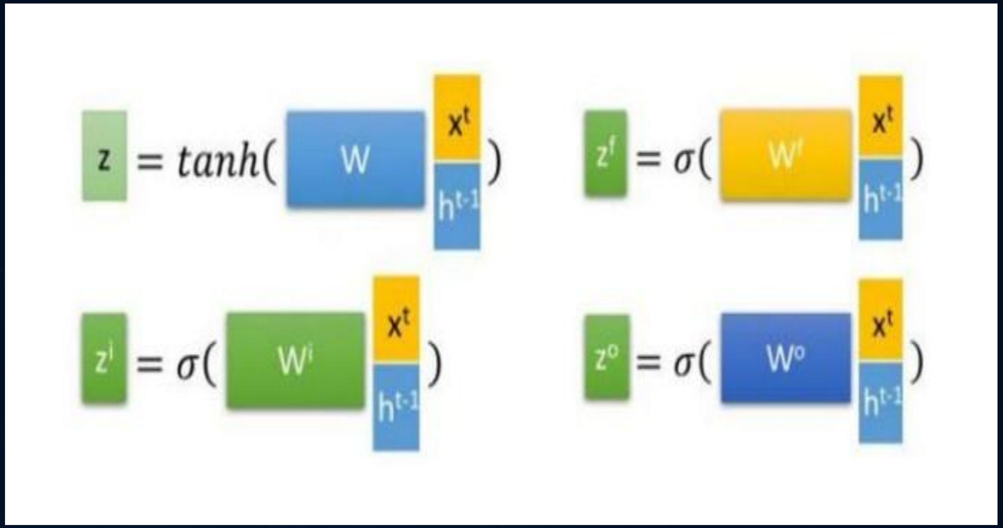


**Fig 8.14** Four states of LSTM

# Long-Short Term Memory (LSTM) Network

## LSTM Architecture

- Memory cells $c^t$, $h^t$, $y^t$ are calculated by gate states as equations below: ($\oplus$ is matrix addition, $\odot$ is Hadamard Product)

- LSTM has: 1) forget, 2) memory select and 3) output stages.
- **Forget stage** - This stage retains important information passed in by previous node $c^{t-1}$ (the previous cell state) and discards unimportant ones. The calculated $z^f$ is used as a forget gate to control what type of $c^{t-1}$ information should be retained or discarded.
- **Memory select stage -** This stage *remembers* input $x^t$ selectively to record important information. $z$ refers to present input. $z^i$ is the input gate to control gating signals.
- **Output stage** - This stage determines what is considered as $h^t$ (the current state) to be passed down to the next layer. $z^o$ is output gate to control this process prior $c^t$ is scaled from memory select stage (convert through a *tanh* function).
- Each layer output $y^t$ is calculated by multiplying weights with $h^t$ and converted the product through an activation function like RNN, the cell state $c^t$ is passed to next layer at the end of each layer.

$$c^t = z^f \odot c^{t-1} + z^i \odot z$$

$$h^t = z^o \odot tanh(c^t)$$

$$y^t = \sigma(W' h^t)$$

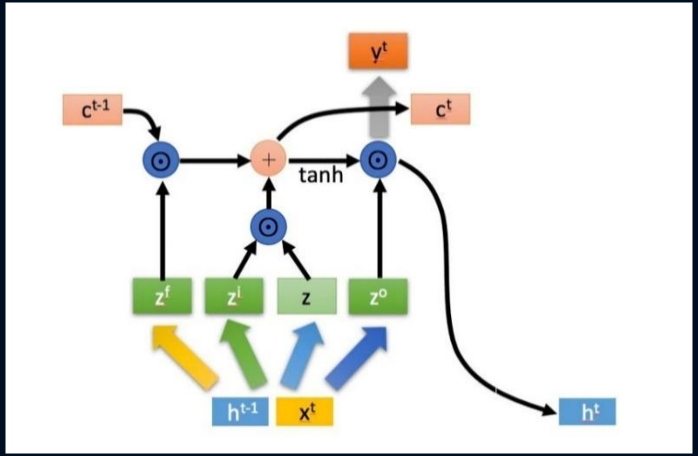


**Fig 8.15** Calculations in memory cell of LSTM

## Gate Recurrent Unit (GRU)
## What is GRU?

- Gate Recurrent Unit (GRU) can be considered as a kind of RNN like LSTM but to manage back propagation gradients problems (Chung et al., 2014; Dey et al., 2017). GRU proposed in 2014 and LSTM proposed in 1997 had similar performances in many cases but the former is often exercised due to simple calculation with comparable results than the latter.

- GRU's input and output structures are like RNN. There are inputs $\boldsymbol{x^t}$, and $\boldsymbol{h^{t-1}}$ to contain relevant information of the prior node. Current outputs $\boldsymbol{y^t}$ and $\boldsymbol{h^t}$ are calculated by combining $\boldsymbol{x^t}$ and $\boldsymbol{h^{t-1}}$. A GRU architecture is shown in Fig. 8.16.

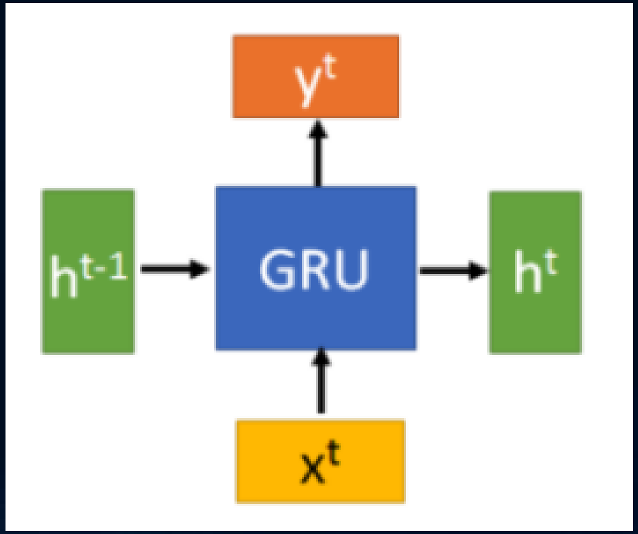
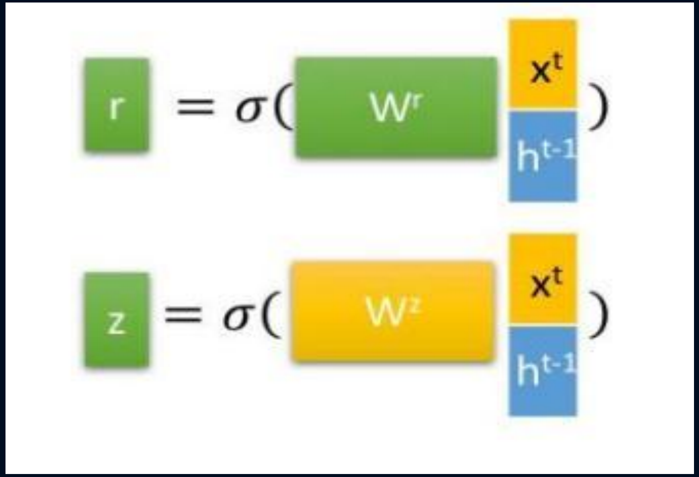


**Fig. 8.16** General architecture of GRU

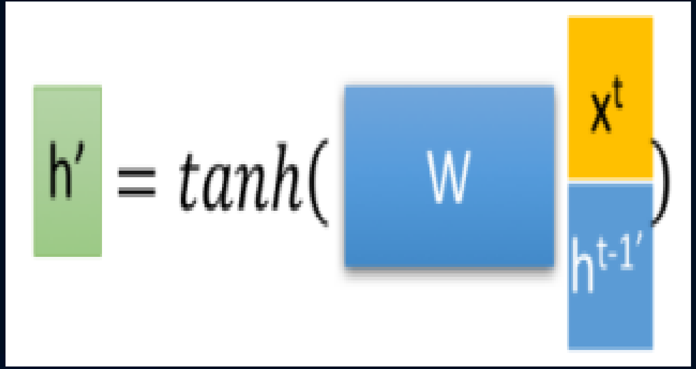
# Gate Recurrent Unit (GRU)
## GRU Inner Architecture

- ***r*** is reset gate and ***z*** is update gate. They are concatenated with input ***x*** $^{t}$ and hidden state ***h*** $^{t-1}$ from the prior node and multiply results with weights as shown in Fig. 8.17.
- When a gate control signal is available, apply ***r*** reset gate to obtain data ***h*** $^{t-1}$ = ***h*** $^{t-1}\odot$ ***r*** after reset, ***h*** $^{t-1}$ are concatenated with ***x*** $^{t}$ and apply a tanh function to generate data lie within range (-1,1) as shown in Fig. 8.18.
- Finally, update memory stage is the most critical step where forget and remember steps are performed simultaneously. The gate **z** obtained earlier is applied as:
- $h_t = (1 - z) \odot h_{t-1} + z \odot h'$ (8.5)
- where ***z*** (gate signal) is within range 0~1. If it is close to 1 or 0, it signifies more data is remained or forgotten respectively.



$r = \sigma(W^r \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$
$z = \sigma(W^z \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$

**Fig. 8.17** Reset and Update Gates of GRU



$h' = tanh(W \begin{bmatrix} x^t \\ h^{t-1'} \end{bmatrix})$

**Fig. 8.18** Computation of **h**

## Gate Recurrent Unit (GRU)
### GRU Inner Architecture

- It is noted that forget ***z*** and select *(1-z)* factors are linked, which means it will forget the passed in information selectively. When weights *(**z**)* are forgotten, it will apply weights in ***h*′** to configurate *(1-z)* at a constant state.

- GRU's input and output structures are like RNN, its internal concept is like LSTM. GRU has one less internal gate as compared with LSTM and fewer parameters but can achieve comparable satisfactory results with reduced time and computational resources. A GRU computation module is shown in Fig. 8.19.

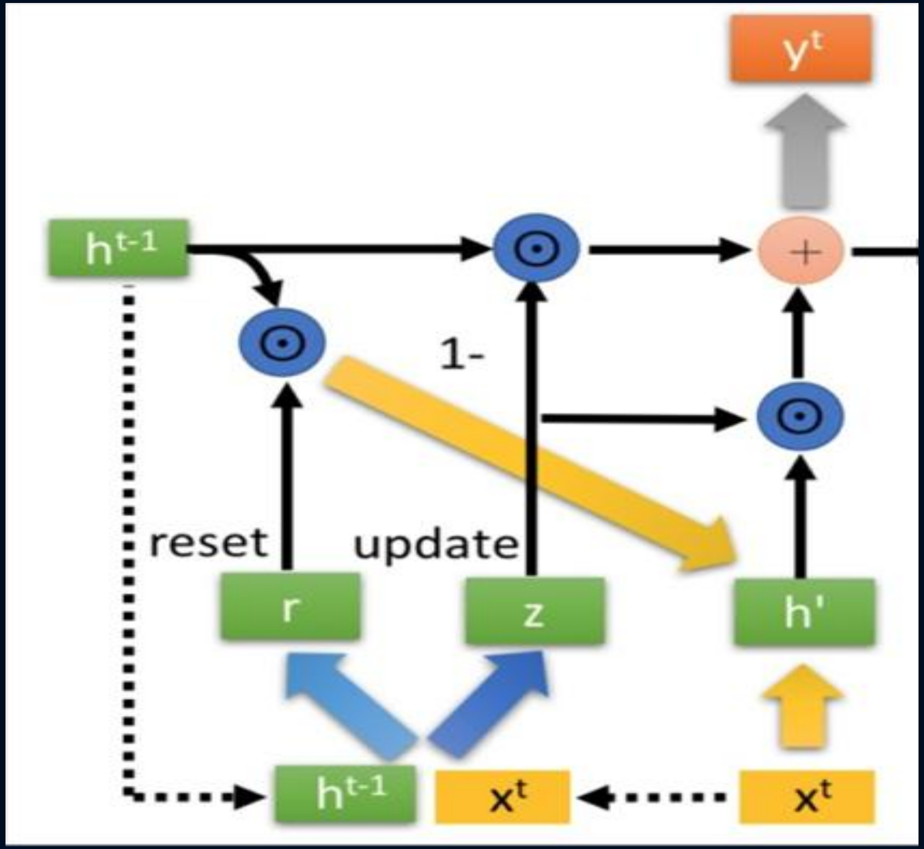


**Fig. 8.19** Computation module of GRU

# Bidirectional Recurrent Neural Networks (BRNNs)

## What is BRNN?

- Bidirectional Recurrent Neural Network (BRNN) is a type with RNN layers in two directions (Singh et al., 2016). It links with previous and subsequent information outputs to perform inference against both RNN and LSTM to possess information from previous one. For example, in text summarization, it is insufficient to consider the information from previous content, sometimes it also requires subsequent text information for words prediction of a sentence. BRNN is proposed to deal with these circumstances.
- BRNN consists of two RNNs superimposed on top of each other. The output is mutually generated by two RNNs states. A BRNN structure is shown in Fig. 8.20.
- BRNN training process is as follows:
  - begin forward propagation from time step 1 to time step *T* to calculate hidden layer's output and save at each time step.
  - proceed from time step *T* to time step *t* to calculate backward hidden layer output and save at each time step.
  - obtain each moment final output according to forward and backward hidden layers after calculating all input moments from both forward and backward directions.

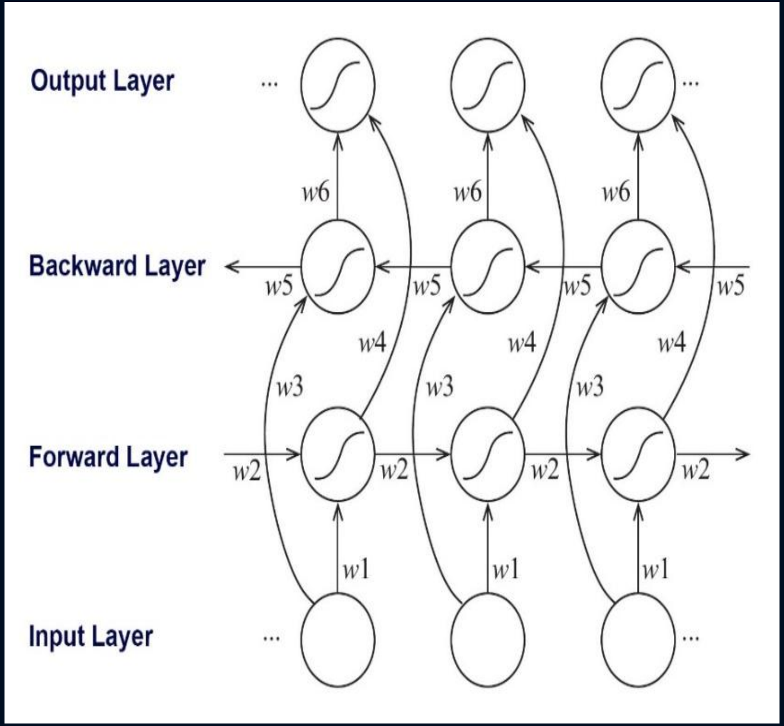


**Fig. 8.20** Structure of BRNN

## 8.5 Transformer Technology

## Transformer Technology
## What is Transformer

- *Transformer* is a network based on attention mechanism without recurrent and convolution units.
- Transformer and LSTM have different training processes. LSTM is serial and iterative; it cannot proceed until the word before is processed against transformer is parallel which means that all words are trained simultaneously to improve computational efficiency.
- A transformer system structure is shown in Fig. 8.21.

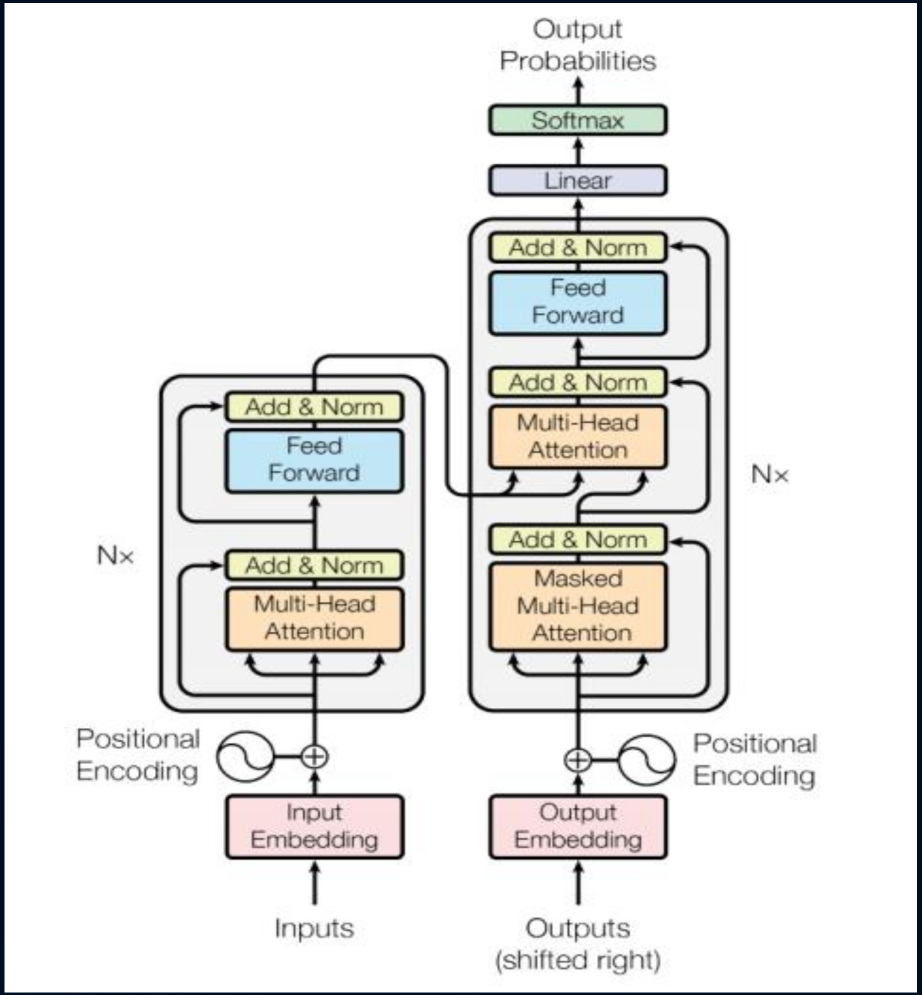


**Fig 8.21** Transformer architecture

# Transformer Technology
## Transformer Architecture

- A transformer model has two parts 1) encoder and 2) decoder. Language sequence extracts as input, encoder maps it into a hidden layer, and decoder maps the hidden layer inversely to a sequence as output.

**Encoder**

- There are six identical encoder layers in the transformer with two sub-layers 1) self-attention and 2) feed forward in each encoder layer.
- Self-attention layer is the first sub-layer to exercise attention mechanism, and a simple fully connected feedforward network is the second sub-layer.
- There follows a residual connection and layer normalization from each of the sub-layers.
- An encoder layer architecture is shown in Fig. 8.22.

**Decoder**

- There are 6 identical encoder layers in the transformer.
- In addition to identical two sublayers as each encoding layer, a third sublayer is added to the decoder to perform multi-head attention, taking the output of last encoder layer as input.
- Residual connections and layer normalization are used sequentially for all sublayers, which is the same as the encoder.
- The decoder's self-awareness is modified by the mask to ensure that inference of the position can only use information from a known position, or in other words, its previous position.

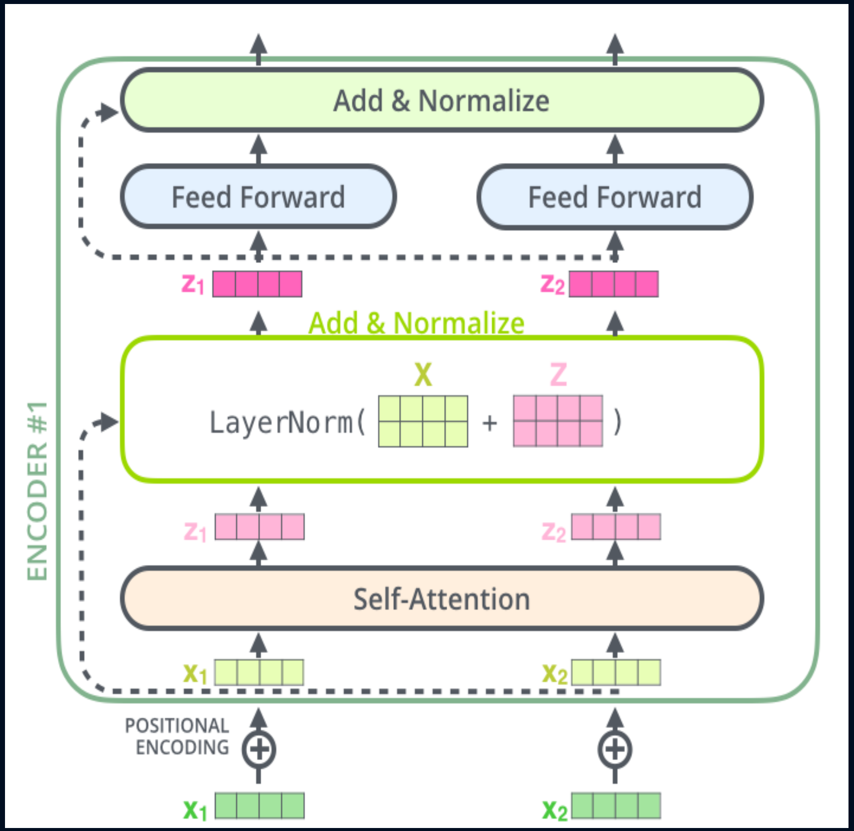


**Fig 8.22** Architecture of an encoder layer

# Transformer Technology
## Positional Encoding

- Since transformer has no iterative process, each word's position information must be provided to ensure that it can recognize the position relationship in language.
- Linear transformation of *sin* and *cos* functions are applied to provide model position information as equation:
- $PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$
- $PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$ (8.6)
- where *pos* represents to a word's position in a sentence, *i* represents to word vector's dimension number, $d_{model}$ represents to embedded dimension's value.
- There is a set of formulas such as sets of 0, 1, or 2, 3 processed with the above sum function respectively.
- As the dimension number increases, the periodic changes moderately to generate a texture containing position information.

# Transformer Technology
## Self-Attention Mechanism

- For input sentence, the word vector of each word is obtained through word embedding, and the position vector of all words is obtained in same dimensions through positional encoding that can be added directly to obtain the true vector representation. $i_{th}$ word's vector is written as $\boldsymbol{x}_i$, $X$ is input matrix combined by all word vectors. $i_{\text{th}}$ row refers to $i_{\text{th}}$ word vector.
- $W_Q$, $W_K$, $W_V$ are matrices defined to perform three linear transformations with $\boldsymbol{X}$ to generate three matrices $Q$ (queries), $K$ (keys), and $V$ (values) respectively.

$$Q = X \cdot W_Q$$
$$K = X \cdot W_K$$
$$V = X \cdot W_V \qquad (8.7)$$

- Attention mechanism computation can be described as:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \qquad (8.8)$$

- The dot products are calculated by multiplying query Q by keys K, dividing the result by $\sqrt{d_k}$ , and applying a softmax function to obtain value scores V.

# Transformer Technology
## Multi-Head Attention Mechanism

- The previously defined set of Q, K, V allows a word uses the information of related words. Multiple Q, K,V defined groups can enable a word to represent sub-spaces at different positions with identical calculation process, except that the matrix of linear transformation has changed from one group ($W_Q$, $W_K$, $W_V$ ) to multiple groups ($W_Q^0$, $W_K^0$, $W_V^0$ ) , ($W_Q^1$, $W_K^1$, $W_V^1$ ) … as equation:

$$MultiHead\ (Q,K,V)\ =\ Concat\ (head_1,\ldots,headh_{)} \cdot W_O$$

$$where\ head\ _i\ =\ Attention(XW_Q^i, XW_K^i, XW_V^i) \qquad (8.9)$$

where $W_O$ is the weights of concatenated results.

- Adding input with a sub-layer (self-attention layer for example) to generate residual connections as equation:

$$X_{attention} = X_{embedding} + \text{Attention}\ (Q,K,V) \qquad (8.10)$$

# Transformer Technology
## Attention Sub-layer

**Layer Normalization of attention sub-layer**

- Layer normalization is to standardize the distribution of hidden layers independently to improve convergence and training processes effectively.

$$X_{attention} = \text{LayerNorm}(X_{attention}) \qquad (8.11)$$

**Feed Forward layer**

- It is a two-layer linear map with an activation function i.e. *ReLU*.

$$X_{hidden} = \text{Linear}(\text{ ReLU}(\text{ Linear }( X_{attention})))$$

followed by residual connection and layer normalization scheme:

$$X_{hidden} = X_{attention} + X_{hidden}$$

$$X_{hidden} = \text{LayerNorm}(X_{hidden}) \qquad (8.12)$$

## 8.6 BERT

# BERT
## What is BERT?

- BERT is a pre-trained model of language representation called Bidirectional Encoder Representation from Transformers (Devlin et al., 2018). It uses MLM (masked language model) to generate deep bidirectional linguistic representation instead of traditional one-direction model or concatenate two one-direction models to pre-train language.

### *Architecture of BERT*

- BERT models are pre-trained either by left-to-right or right-to-left language models previously, this unidirectional property restricts model structure to obtain unidirectional context information only and propensity for representation. BERT adopted MLM in pre-training stage and a bidirectional transformer with deep layers to build the entire model, the representation generated integrates both left and right content information. A BERT system architecture is shown in Fig. 8.23.

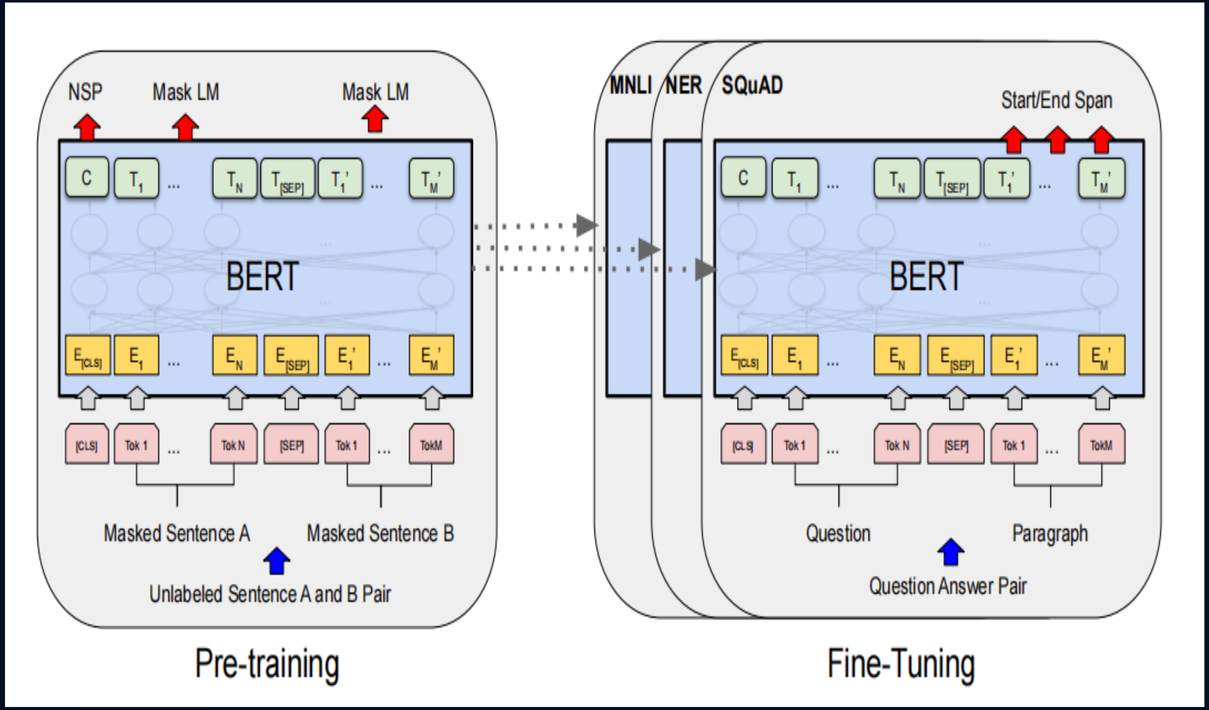


**Fig 8.23** System Architecture of BERT

# BERT
## Training of BERT

- BERT has two training process steps: 1) pre-training and 2) fine-tuning.

**Pre-training BERT**

- BERT is not constrained by a one-way language model because it randomly replaces tokens in each training sequence with mask tokens ([MASK]) with 15% probability to predict the original word at position [MASK]. [MASK] does not appear in fine-tuning of downstream tasks, leading to differences in pre-training and fine-tuning stages, because the pre-training objective improves language representation, being sensitive to [MASK] and to other insensitive tokens. BERT applies the following strategies:
- First, in each training sequence, a token position is randomly selected for prediction with a probability of 15%. If ith token is selected, it will be replaced by one of the following tokens:
  1. 80% is [MASK]. For instance, the cat is **adorable** —> the cat is **[MASK].**
  2. 10% is a random token. For instance, the cat is **adorable** —> the cat is ginger.
  3. 10% is the original token (no change). For instance, his cat is **adorable** —> his cat is adorable.
- Second, apply $T_i$ corresponding to the position, predict the original token through full connection, then apply softmax to output the probability of each token, and finally apply cross entropy to evaluate loss.
- This method causes BERT sensitive to [MASK] and all tokens to extract representative information.

# BERT
## Training of BERT

**Next Sentence Prediction (NSP)**

- There are tasks such as question answering and natural language reasoning to understand the relationship between two sentences.
- Sentence-level representations cannot be captured directly, as MLM tasks tend to extract token-level representations.
- BERT applies NSP pretraining task to let the model understands the relationships between sentences and predict whether they are connected.
- For every training sample, select Set A and B from corpus to create a sample, where Set A is 50% of Set B (labeled "IsNext"), and Set B is 50% random.
- Next, training examples are put into BERT model to generate binary classification predictions.

**Fine-tuning BERT**

- It is necessary to add an additional output layer to fine-tune downstream tasks for satisfactory performance.
- It does not require task-specific structural modification in this process.

## 8.7 Other Related Transformer Technology

# Other Related Transformer Technology

## Transformer-XL

- Transformers are widely used as a feature extractor in NLP but required to set a fixed length input sequence i.e. the default length for BERT is 512. If text sequence length is shorter than fixed length, it must be solved by padding. If text sequence length exceeds fixed length, it can be divided into multiple segments. Each segment is processed at training separately as shown in Fig. 8.24.
- Nevertheless, there are two problems: 1) segments are trained independently, the largest dependency between different tokens depends on the segment length; 2) segments are separated according to a fixed length without sentences' natural boundaries consideration to produce semantically incomplete segments. Thus, transformer-XL (Dai et al., 2019) is proposed.

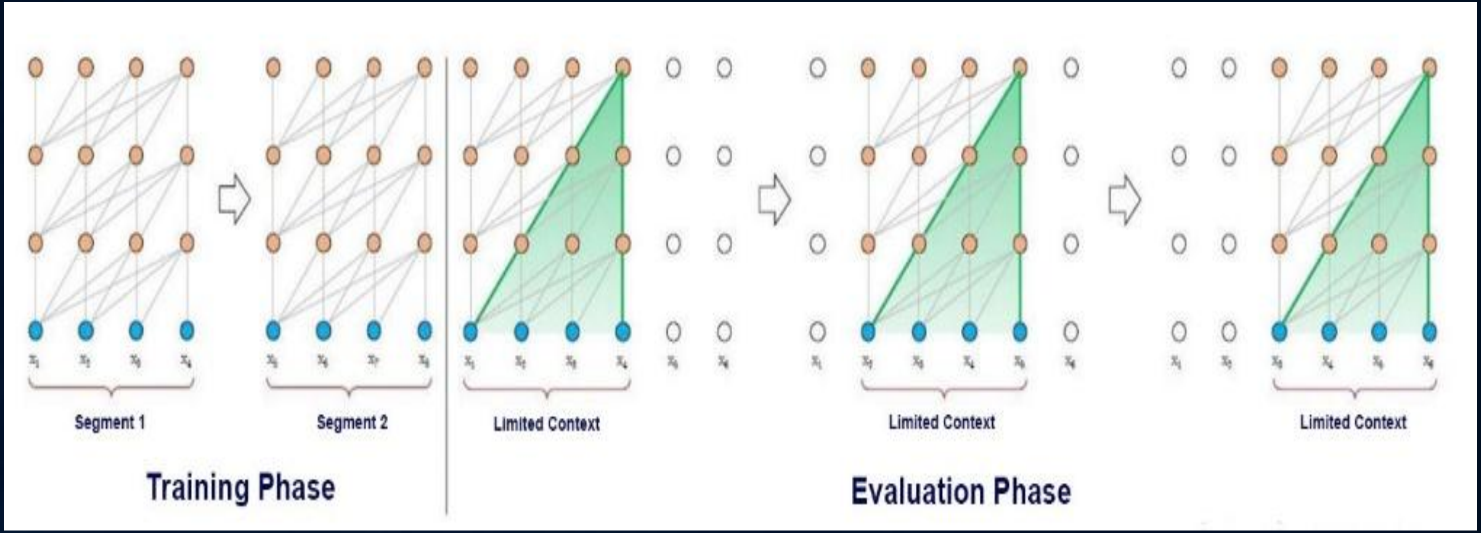


**Fig. 8.24** Segment training of standard transformer

# Other Related Transformer Technology
## Transformer-XL

1) **Segment-level Recurrence -** When processing the current segment, Transformer-XL caches and applies hidden vector sequence to all layers from previous segment. These sequences only participate in forward calculation without back propagation called segment-level recurrence. Fig. 8.25 shows the segment training of Transformer-XL.
2) **Relative Position Encodings:** Each token has an embedding position to represent positions relationship in standard transformer. This embedding position encoding is either generated by sin/cos function or learning, but it is impractical in Transformer-XL because positional relationship of different segments is unidentified if the same positional code is added to each segment. Transformer-XL applies relative position encoding instead of absolute position encoding, so when calculating the hidden vector of current position, it considers tokens' relative position relationships to calculate attention score.

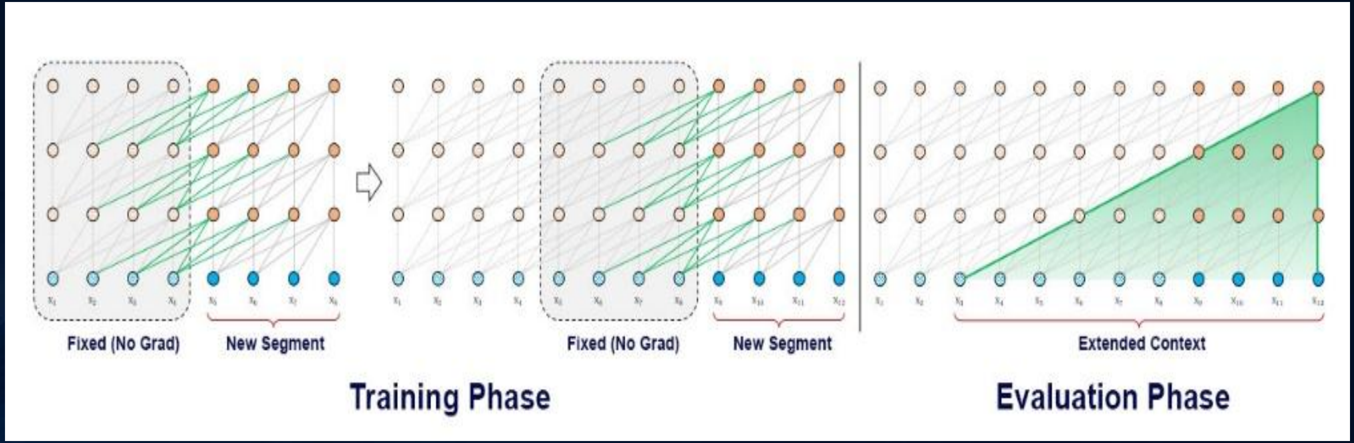


**Fig 8.25** Segment training of Transformer-XL

## Other Related Transformer Technology
### ALBERT

- BERT model has many parameters, but it is limited by GPU/TPU memory size as model size increases. Google proposed *A Lite BERT (ALBERT)* to solve this problem.
- ALBERT applies two techniques to reduce parameters and improve NSP pre-training task, which include:

1. parameter sharing -- apply same weights to all 12-layers
2. factorize embeddings -- shorten initial embeddings to 128 features
3. pretrain by LAMB Optimizer -- replace ADAM Optimizer
4. Sentence Order Prediction (SOP) -- replace BERT's Next Sentence Prediction (NSP) task.
5. N-gram masking -- modify Masked Language Model (MLM) task to mask out words' N-grams instead of single words.

## 8.8 Summary

## Summary

- This chapter studied Transfer Learning, a technique in deep learning and parameter-based transfer learning is the most used strategy amongst others.
- RNNs architectures and related networks include LSTMs, GRUs, and BRNNs.
- LSTMs enhance "short memory" of RNNs. GRU is simpler than LSTM, its idea is like LSTM. BRNN consists of two RNNs in different directions.
- Further, Transformer models include standard Transformer, BERT, Transformer-XL, and ALBERT.
- All these networks have been widely implemented in NLP tasks such as sentiment analysis and machine translation, and even some of them (such as translators) can be used for computer vision tasks.

## Next

CH9 NLPApplications