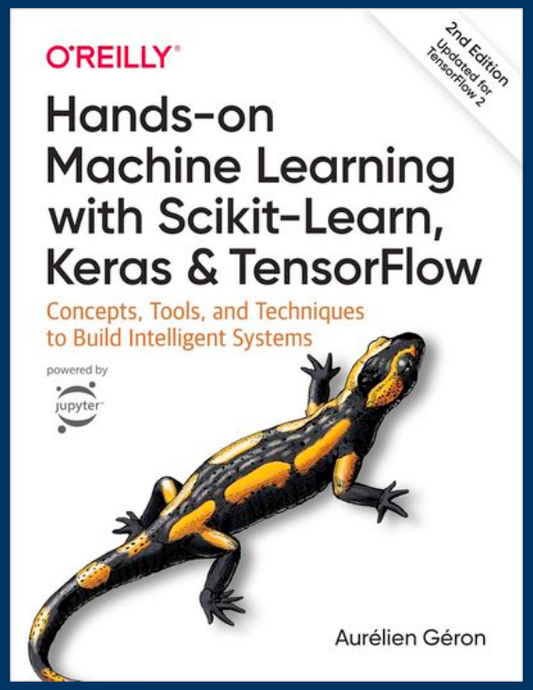
AI Bootcamp day#2

Session#3 -
RNN(Recurrent Neural Network)의 이해와 실습

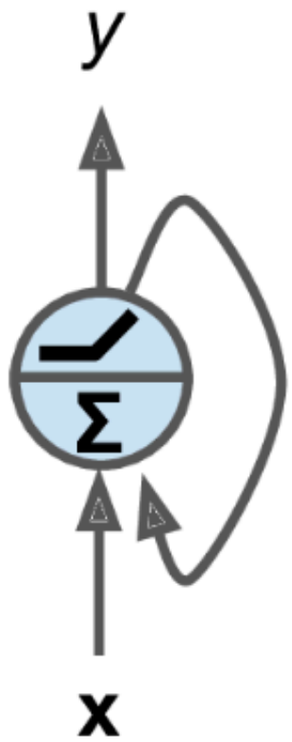


Ch 15. Processing Sequences Using RNNs and CNNs
Ch 16. Natural Language Processing with RNNs and Attention
Ch 17. Representation Learning Using Autoencoders

Example#1: random number prediction using RNN
Example#2: stock price prediction using LSTM
Example#3: dimensionality reduction using autoencoder

Anmo Kim (Hanyang University, AI department)

Processing sequences using RNNs



simplist RNN

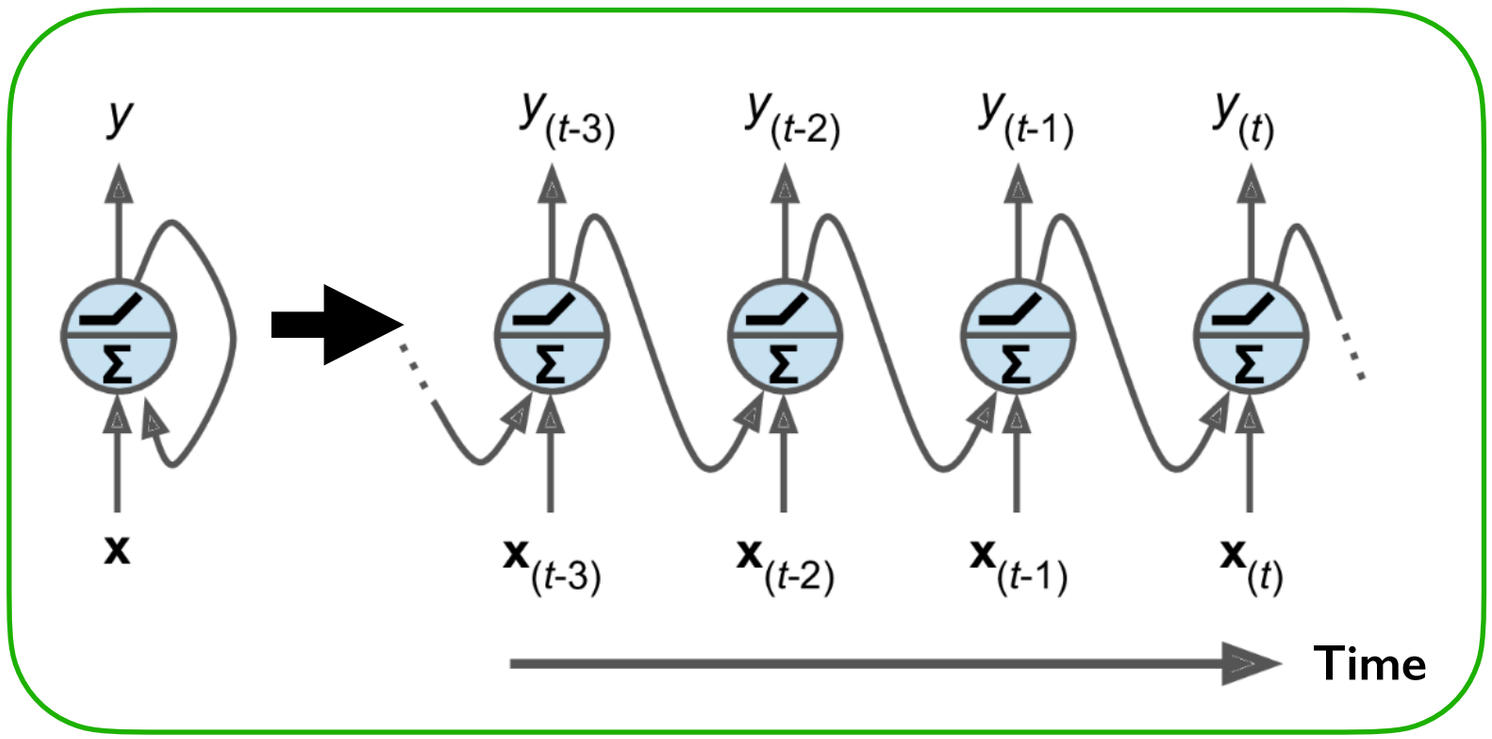
- **RNNs employs “recurrent” connections**
  - signaling in the brain actually has significant recurren
- **RNN is designed for <u>“prediction of the future based on the past”</u>**
  - e.g., stock prices, autopilot (anticipating trajectories of other cars), language processing (automatic translation, speech-to-text)
- **Two main difficulties of RNNs in training them using backpropagation through time**
  - <u>unstable gradients</u> → recurrent dropouts, recurrent layer normalization
  - <u>limited short-term memory</u> → LSTM and GRU cells
- a regular dense network can deal with short sequences (without recurrency)
  - CNNs can work well for relatively long audio samples or text

Recurrent neurons

- **recurrent NN has connections pointing backward**
  - feedforward NN → activations flow only in one direction
  - RNN is typically represented by **unrolling** the network through time

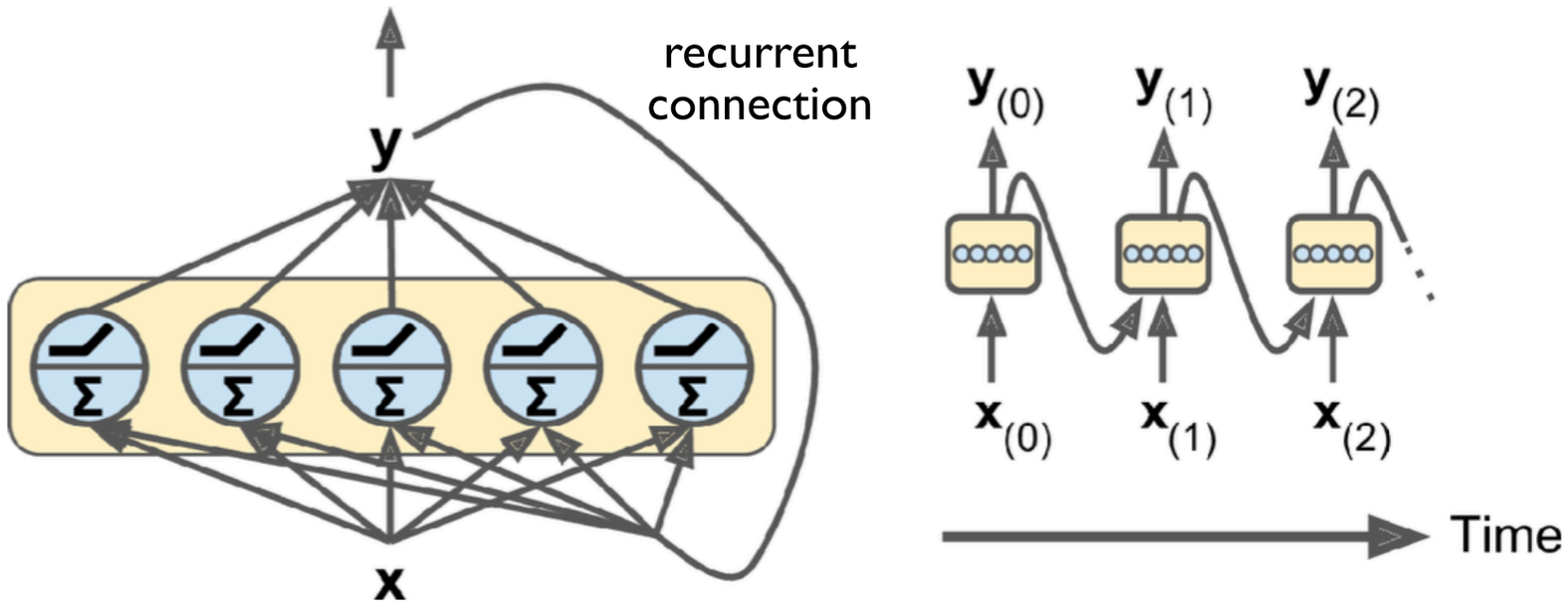
$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_y{}^{\intercal}\mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

output · activation function · input · recurrent term · bias



Recurrent neurons and layers

- **recurrent NN has connections pointing backward**
  - RNN is typically represented by **unrolling** the network through time



a layer of recurrent neurons unrolled & simplified representation

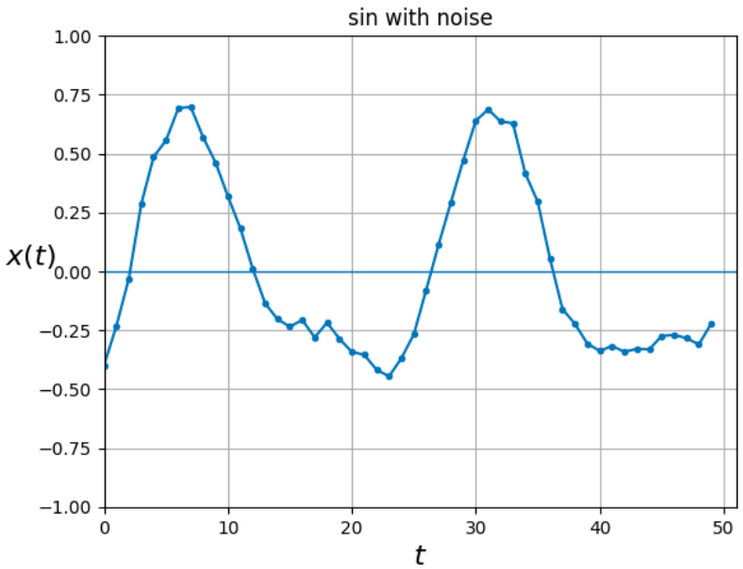
$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \mathbf{Y}_{(t-1)}\mathbf{W}_y + \mathbf{b}\right)$$

$$= \phi\left(\begin{bmatrix}\mathbf{X}_{(t)} & \mathbf{Y}_{(t-1)}\end{bmatrix}\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_y\end{bmatrix}$$

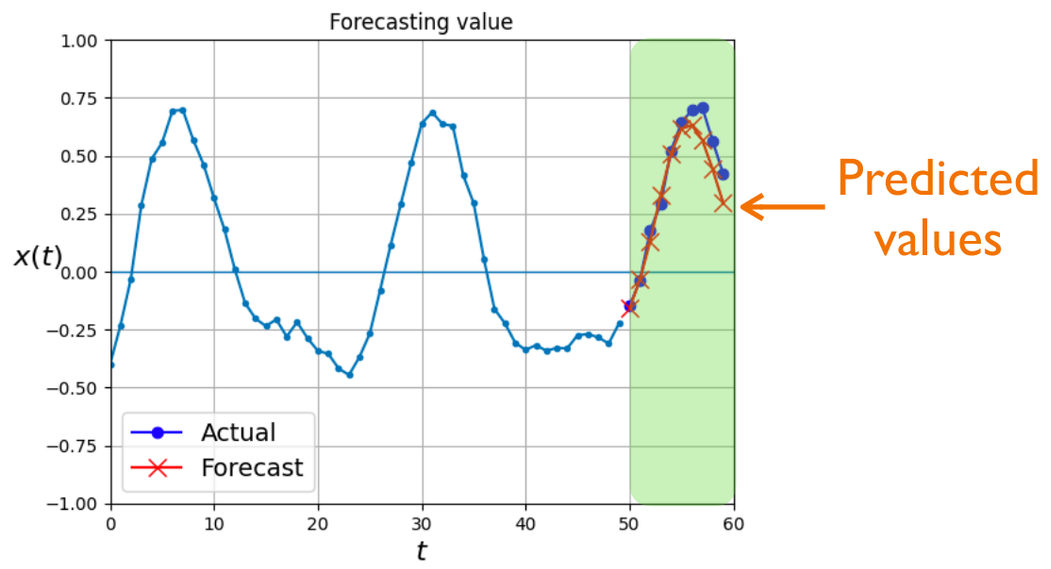
← 한번에 계산 가능

Example#7 — forecasting a time series using RNN

Input

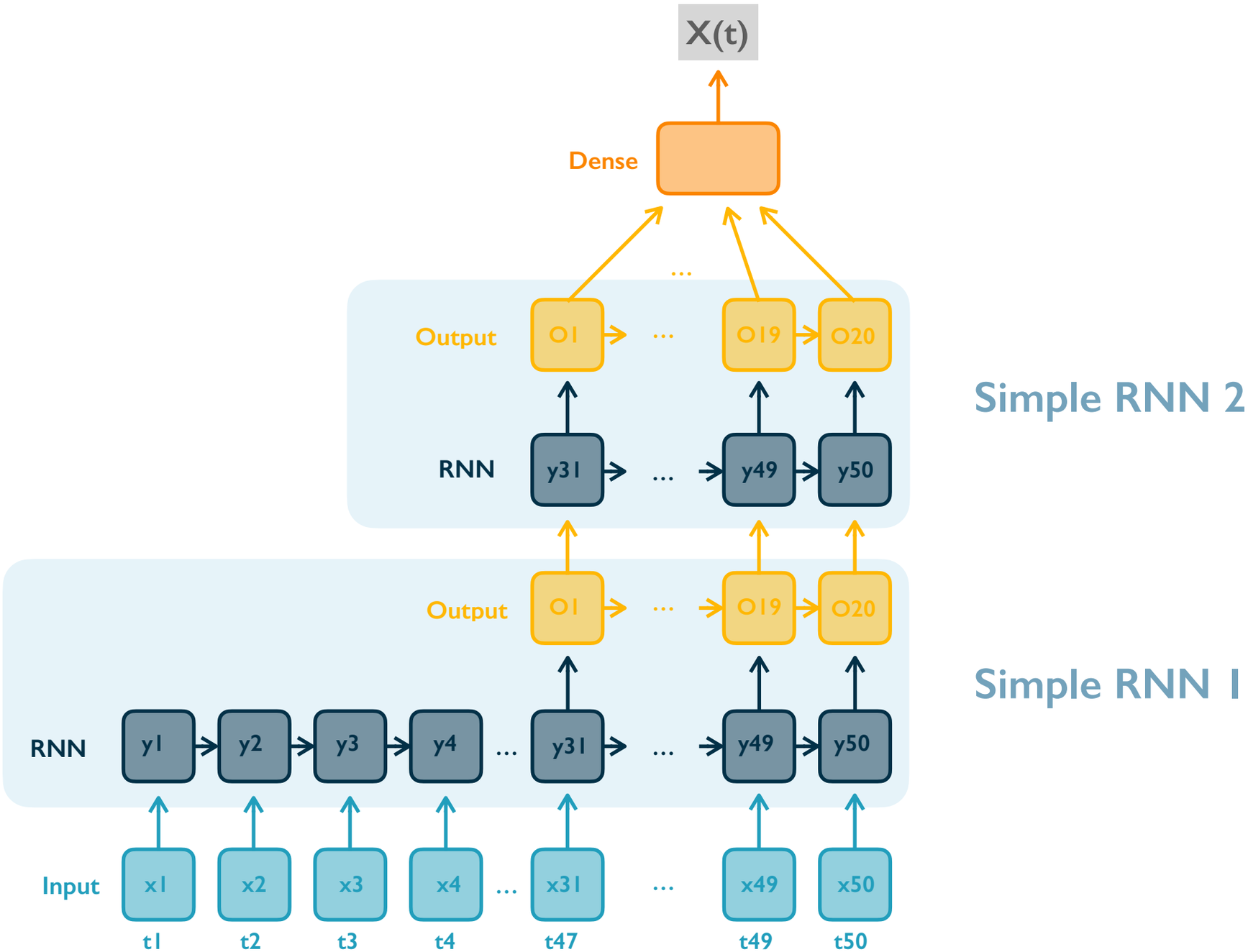


Output



- 두 Sin 값을 더한 후 noise를 추가시킨 시계열 데이터
- 총 50개의 time step을 학습 후 10개의 time step을 예측

Example#7 — forecasting a time series using RNN



Ex#7 — forecasting a time series using RNN

Make and Split dataset

```
np.random.seed(42)
n_steps = 50
series = generate_time_series(10000, n_steps + 1)
X_train, y_train = series[:7000, :n_steps], series[:7000, -1]
X_valid, y_valid = series[7000: :n_steps], series[7000:, -1]
```

Build a model

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)])
model.compile(loss="mse", optimizer="adam")
```

Train the model

```
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid, y_valid))
```

Test the model

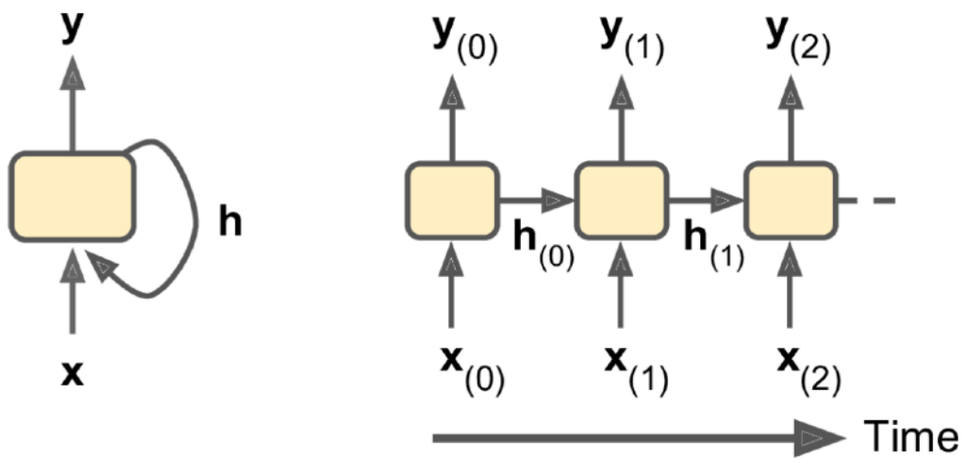
```
n_steps = 50
pred_steps = 10
series = generate_time_series(1, n_steps + pred_steps)
X,Y = series[:, :n_steps], series[:, n_steps:]

for step_ahead in range(pred_steps):
    y_pred_one = model.predict(X[:, step_ahead:])[:, np.newaxis, :]
    X = np.concatenate([X, y_pred_one], axis=1)

Y_pred = X[:, n_steps:]
plt.title("Forecasting value")
plot_multiple_forecasts(X[:, :n_steps], Y, Y_pred)
plt.show()
```

RNN neurons are *memory cells*

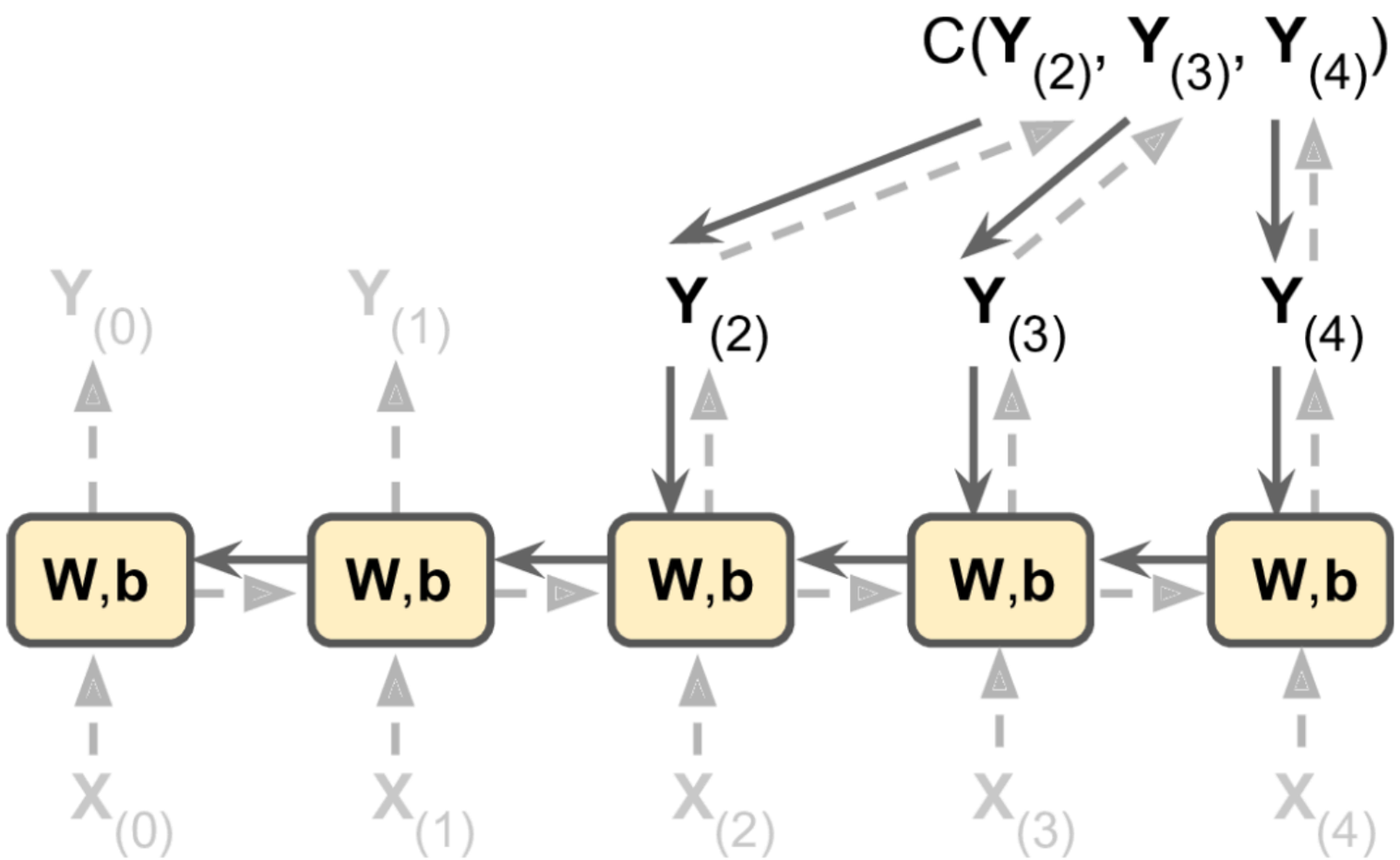
- **Memory cell: a part of a neural network that preserves some state across the steps**
  - RNN has a memory because $y_{(t)}$ is a function of all the past inputs
  - a single recurrent neuron is a very **basic memory cell**
    - practically, capable of learning only short patterns (less than 10 steps)



  - there are other cells that can have a longer memory (10x longer than a basic cell)

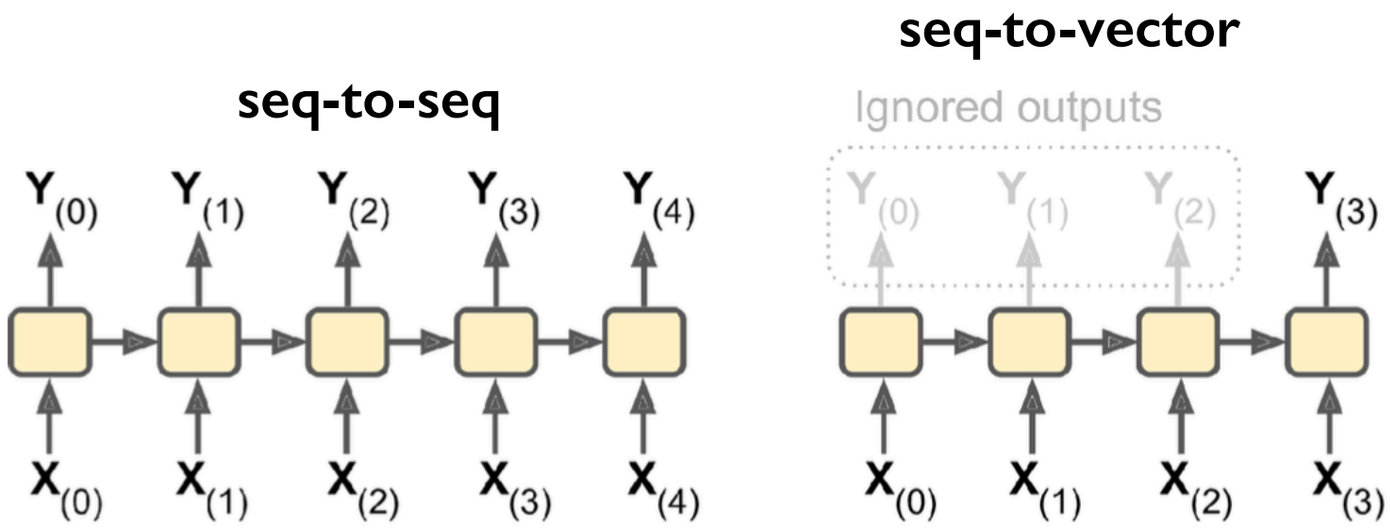
How can we train an RNN?

- **backpropagation through time (BPTT)**
  - unroll it though time and use regular backpropagation technique
  - a cost function:



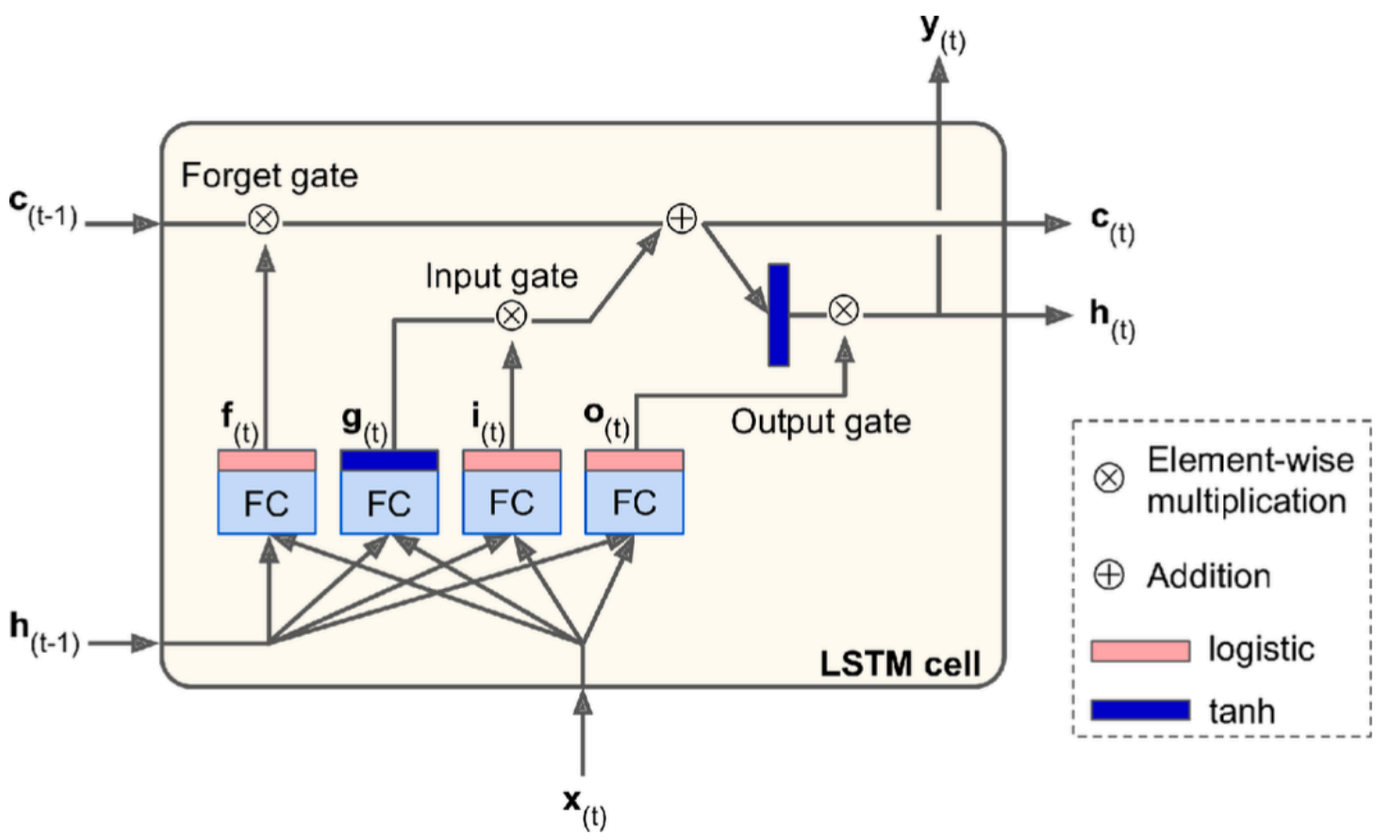
Different input/output pairs in RNNs

- **seq-to-seq: e.g., stock prices**
  - input: stock prices from N days ago through today
  - output: stock prices from N-1 days ago through tomorrow
  - **"encoder-decoder structure"** generally performs better than an RNN
    - encoder for seq-to-vector; decoder for vector-to-seq
- **seq-to-vector:** e.g., movie review
  - input: all the comments
  - output: sentiment score (별점)
- **vector-to-seq:** e.g., image to caption



Long Short-Term Memory (LSTM) network

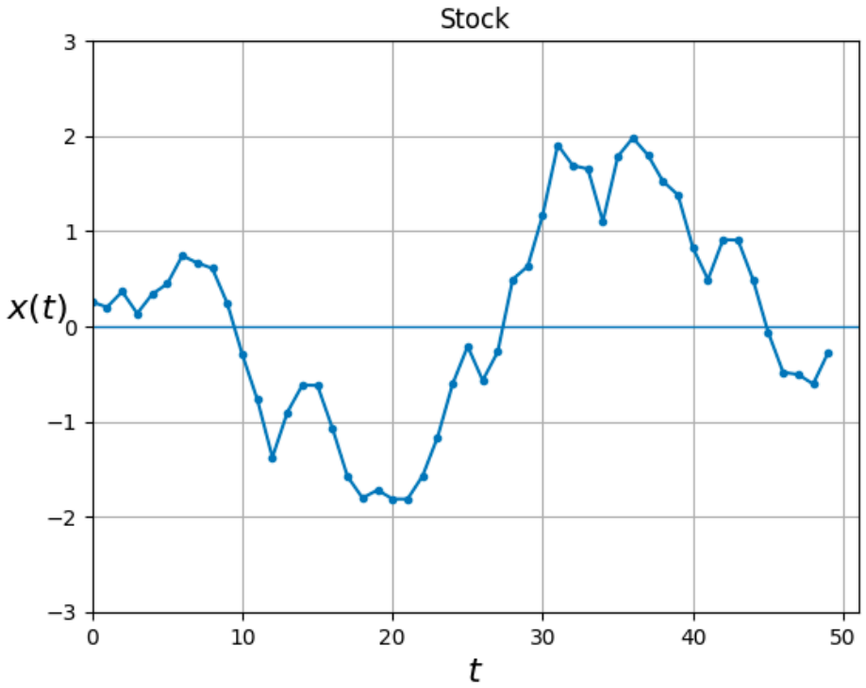
- **LSTM extended the memory length significantly**
- **LSTM cell (1997) is designed to learn**
  - what to forget, f(t)
  - what to store in the long term state, o(t)
  - what to read from the input and the hidden state, i(t) & g(t)



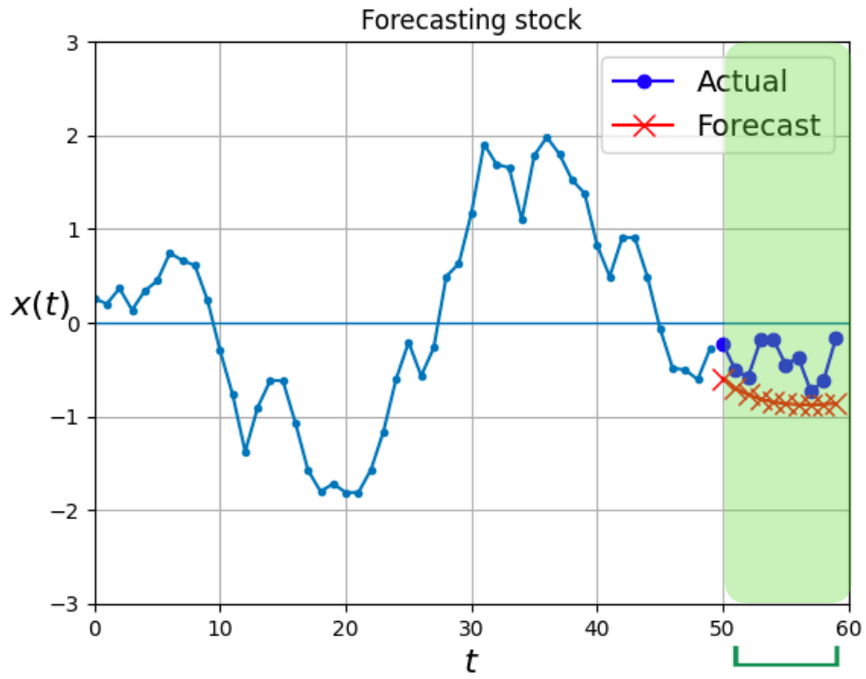
Example#8: predicting the stock price using LSTM

	Code	ISU_CD	Name	Market	Dept	Close	ChangeCode	Changes	ChagesRatio	Amount	Marcap	Stocks	MarketId
0	005930	KR7005930003	삼성전자	KOSPI		70500	2	-600	-0.84	361168557100	420869669775000	5969782550	STK
1	373220	KR7373220003	LG에너지솔루션	KOSPI		551000	2	-6000	-1.08	46412915000	128934000000000	234000000	STK
2	000660	KR7000660001	SK하이닉스	KOSPI		122400	2	-2700	-2.16	149607884800	89107489476000	728002365	STK
3	207940	KR7207940008	삼성바이오로직스	KOSPI		793000	2	-9000	-1.12	24323038000	56440982000000	71174000	STK
4	005490	KR7005490008	POSCO홀딩스	KOSPI		599000	2	-22000	-3.54	702103136000	50658166770000	84571230	STK
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2734	001525	KR7001521004	동양우	KOSPI		5010	2	-20	-0.40	4313370	3094832310	617731	STK
2735	348840	KR7348840000	데이드림엔터	KONEX	일반기업부	3205	2	-900	-21.92	3205	2700212500	842500	KNX
2736	245450	KR7245450002	씨앤에스링크	KONEX	일반기업부	1600	1	37	2.37	1600	2527936000	1579960	KNX
2737	322190	KR7322190000	베른	KONEX	일반기업부	165	1	15	10.00	463	1472657505	8925197	KNX
2738	308700	KR7308700004	테크엔	KONEX	일반기업부	253	4	33	15.00	227000	1012000000	4000000	KNX

- 한국 증권 시장 정보 중 랜덤하게 불러온 종목 코드의 월별 평균 종가인 시계열 데이터
- 총 50개의 time step을 학습 후 10개의 time step을 예측

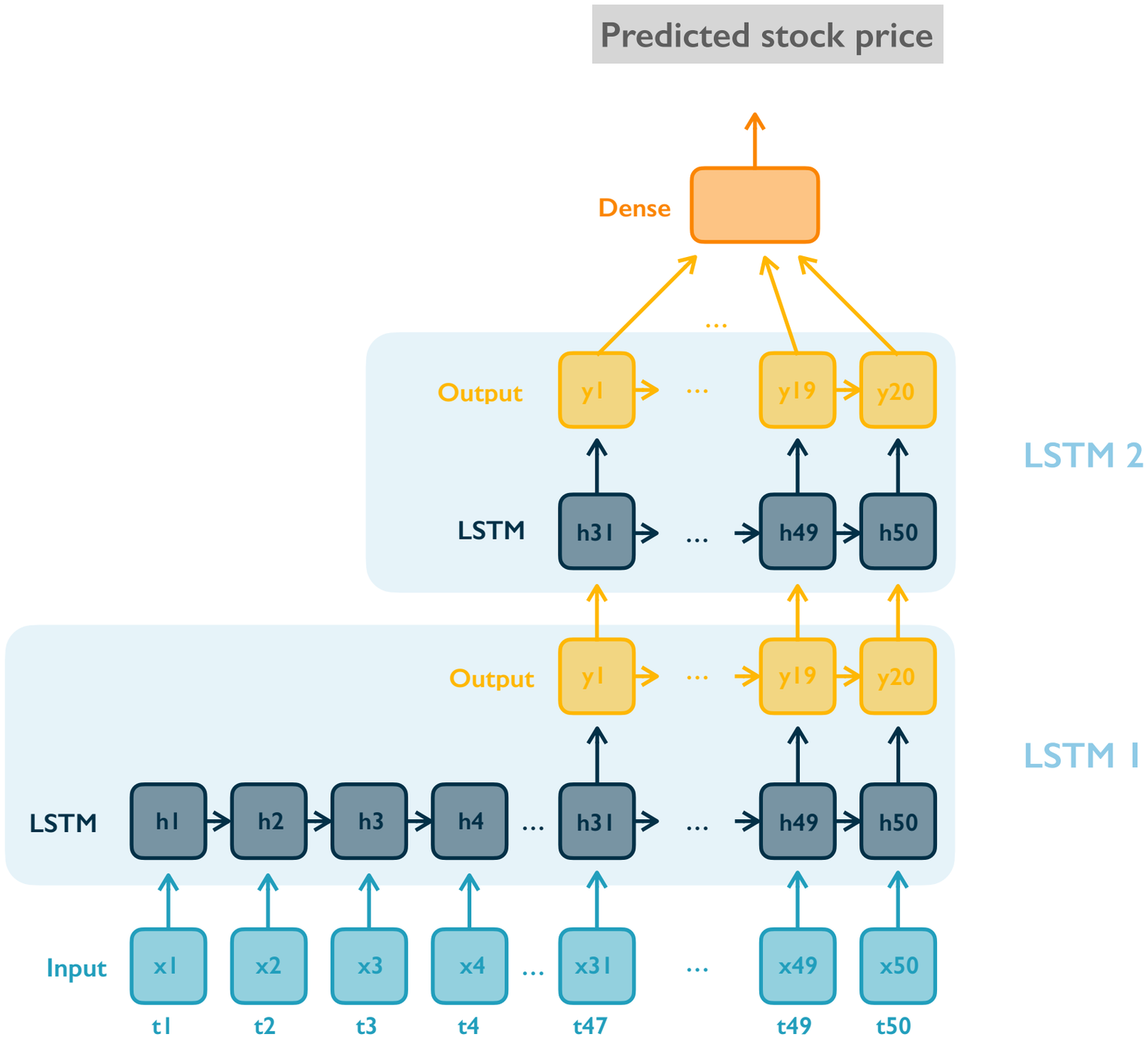


Input



Output

Example#8: predicting the stock price using LSTM



Example#8: predicting the stock price using LSTM

Import library

```
import FinanceDataReader as fdr
```

Load dataset

```
n_steps = 50
series = generate_stock_data(100, n_steps + 1)
```

Split the data

```
X_train, y_train = series[:70, :n_steps], series[:70, -1]
X_valid, y_valid = series[70:90, :n_steps], series[70:90, -1]
X_test, y_test = series[90:, :n_steps], series[90:, -1]
```

Build a model

```
model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.LSTM(20),
    keras.layers.Dense(1)])
model.compile(loss="mse", optimizer="adam")
```

Train the model

```
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_valid, y_valid))
```

Test the model

```
n_steps = 50
pred_steps = 10
series = generate_stock_data(1, n_steps + pred_steps, kospi=True)
X,Y = series[:, :n_steps], series[:, n_steps:]
plt.title(“Stock”); plot_series(X[0, :, 0])

for step_ahead in range(pred_steps):
    y_pred_one = model.predict(X[:, step_ahead:])[:, np.newaxis, :]
    X = np.concatenate([X, y_pred_one], axis=1)
Y_pred = X[:, n_steps:]

plot_multiple_forecasts(X[:, :n_steps],Y,Y_pred)
plt.title("Forecasting stock")
```

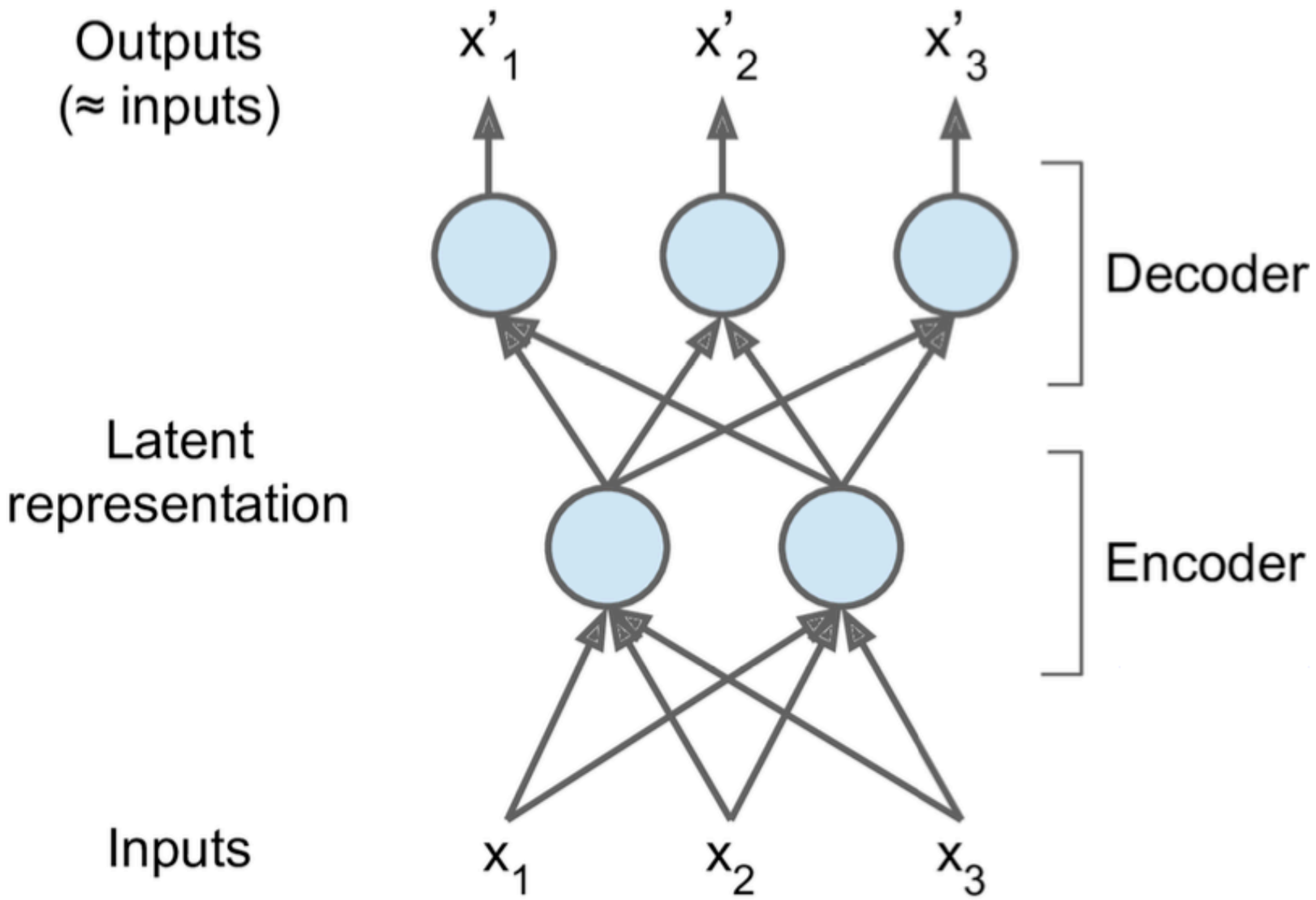
NLP with RNNs and attention

- **Turing test (1950) was for a linguistic task (e.g., chatbot)**
  - instead of recognizing cats in a picture, playing chess, comparing music, or escaping a maze
  - mastering language is arguably Homo Sapiens's greatest cognitive ability
  - can we build a machine that can read/write natural language?
  - character RNN (predicting the next character)
    - stateless RNN
    - statful RNN
    - KNN for sentiment analysis
    - RNNs for encoder-decoder architecture
  - **attention mechanisms** + RNN
  - Transformer: attention only (with no recurrent connections)
    - GPT-2, BERT

Autoencoder for dimensional reduction

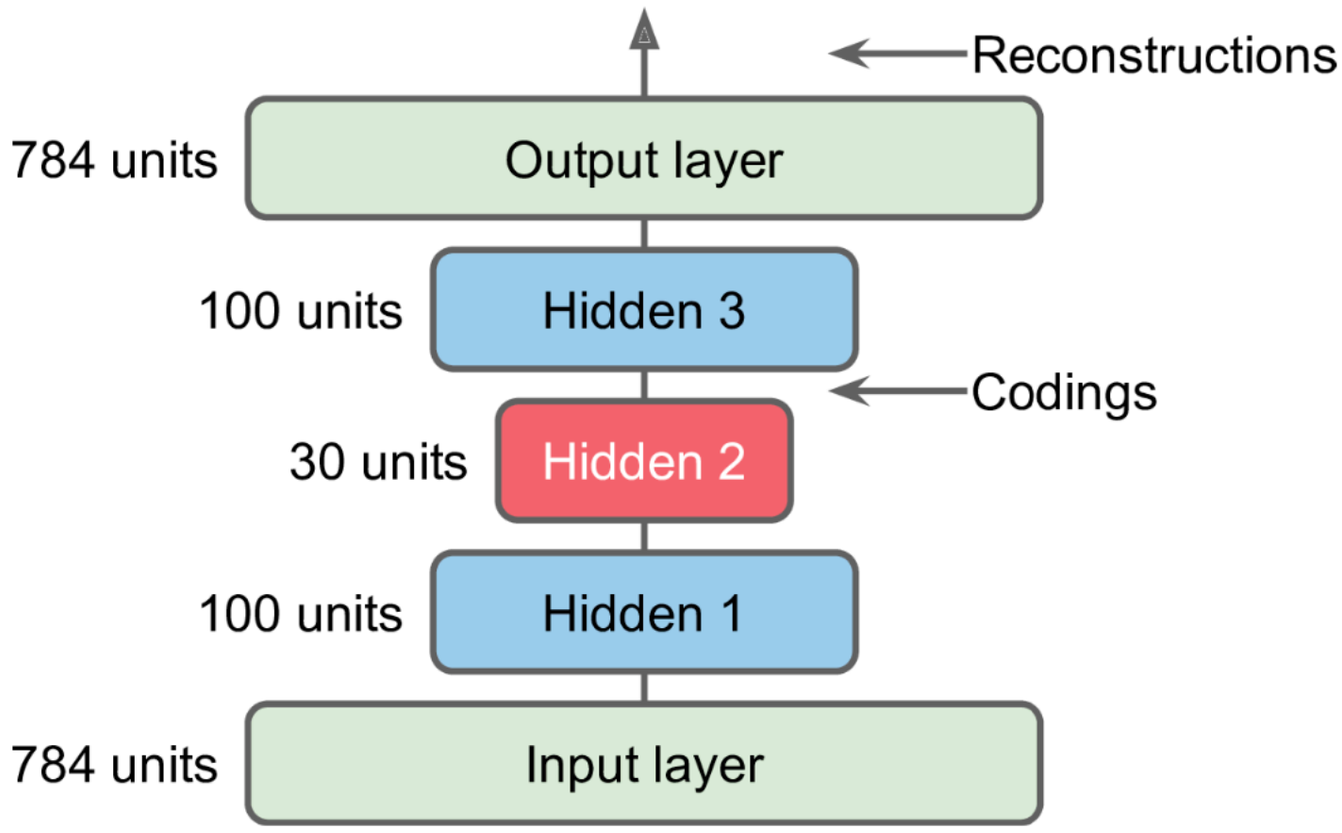
- **Autoencoder — ANN capable of learning dense represenations of the input data, without any supervision**
  - "dense representation" is also known as "latent representations" or "codings"
  - these *codings* have much lower dimensionality than the input data
    - autoencoder is useful for dimensionality reduction
    - also useful for
      - feature detectors
      - unsupervisded pretraining of a DNN
      - generative models (but it's fuzzier than GAN)
  - autoencoder = encoder + decoder
    - encoder is a recognition network
    - decoder is a generative network
    - autoencoder is "undercomplete" and thus is forced to learn most important features, while discarding less important ones

Architecture of an Autoencoder



Stacked autoencoders (or deep autoencoders)

- **The architecture is symmetrical w.r.t. the central hidden layer**
- **How to determine the number of units?**
  - if too many, an encoder may learn to map each input to assign an arbitrary number (overfitting)
  - an autoencoder for MNIST
    - 784 inputs,

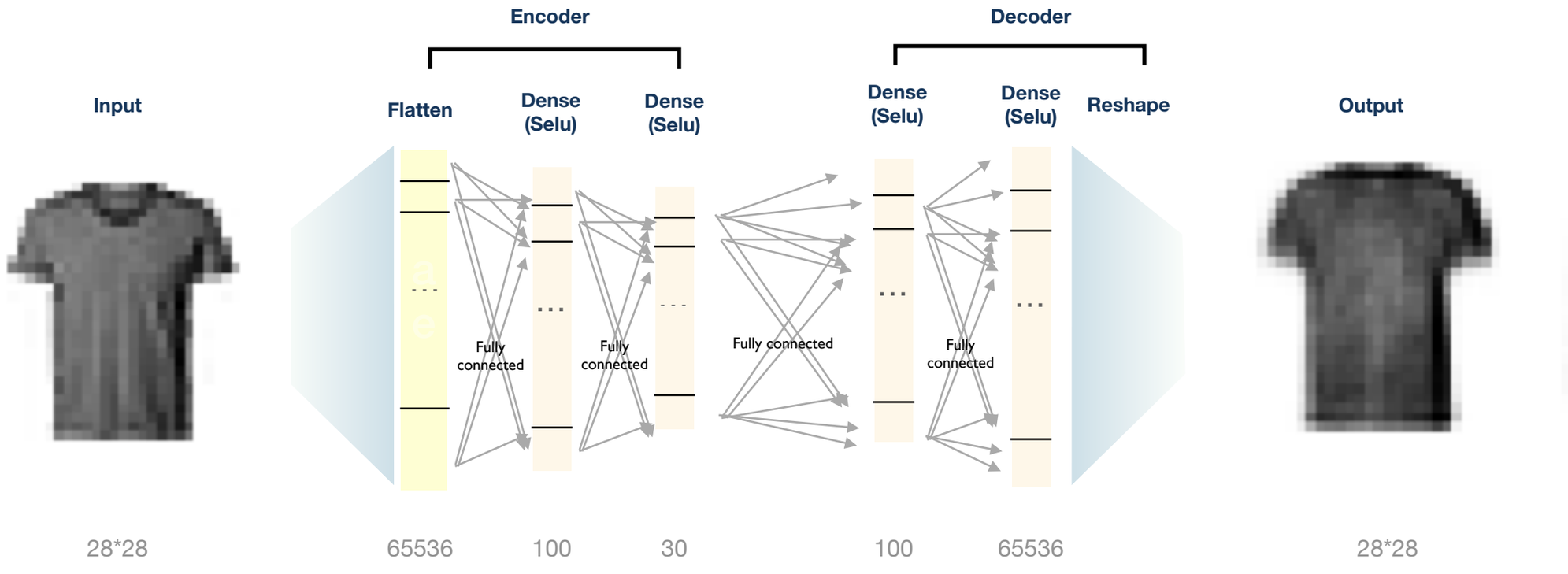


How to train autoencoders?

- **tying weights**
- **training one autoencoder at a time**

Example#9— Dimensional reduction by Autoencoder

- fasion MNIST 데이터 셋을 활용
- Encoder와 Decoder로 구성된 모델을 학습
- Input 으로 넣어준 이미지를 Decoder 가 똑같이 재현하도록 학습
- Encoder 의 학습 정보를 활용 30차원의 features 로 재생성



Ex#9— Dimensional reduction by Autoencoder

Import library

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow import keras
from sklearn.manifold import TSNE
```

Load & Split the dataset

```
(X_train_full, y_train_full), (X_test, y_test) = keras.datasets.fashion_mnist.load_data()
X_train_full = X_train_full.astype(np.float32) / 255
X_test = X_test.astype(np.float32) / 255
X_train, X_valid = X_train_full[:-5000], X_train_full[-5000:]
y_train, y_valid = y_train_full[:-5000], y_train_full[-5000:]
```

Build the model

```
tf.random.set_seed(42)
np.random.seed(42)
stacked_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu"),
])
stacked_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
```

Ex#9— Autoencoder for dimensional reduction

Train the model

```
stacked_ae.compile(loss="binary_crossentropy",
                   optimizer=keras.optimizers.SGD(learning_rate=1.5),
                   metrics=[rounded_accuracy])
history = stacked_ae.fit(X_train, X_train, epochs=20,
                         validation_data=(X_valid, X_valid))
```

Validation & preprocessing

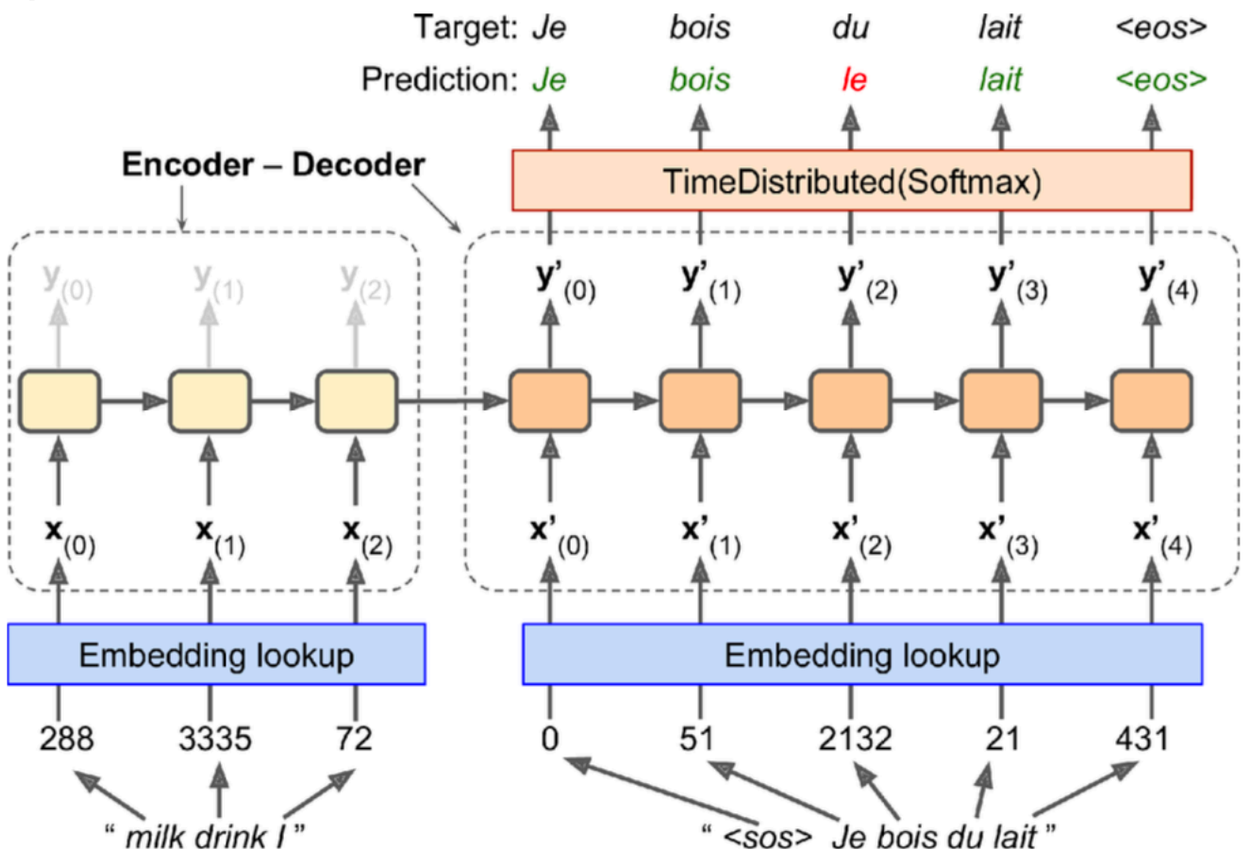
```
X_valid_compressed = stacked_encoder.predict(X_valid)
tsne = TSNE()
X_valid_2D = tsne.fit_transform(X_valid_compressed)
X_valid_2D = (X_valid_2D - X_valid_2D.min()) / (X_valid_2D.max() - X_valid_2D.min
```

Plot the encoder output

```
plt.figure(figsize=(10, 8))
cmap = plt.cm.tab10
plt.scatter(X_valid_2D[:, 0], X_valid_2D[:, 1], c=y_valid, s=10, cmap=cmap)
image_positions = np.array([[1., 1.]])
for index, position in enumerate(X_valid_2D):
    dist = np.sum((position - image_positions) ** 2, axis=1)
    if np.min(dist) > 0.02: # if far enough from other images
        image_positions = np.r_[image_positions, [position]]
        imagebox = mpl.offsetbox.AnnotationBbox(
            mpl.offsetbox.OffsetImage(X_valid[index], cmap="binary"),
            position, bboxprops={"edgecolor": cmap(y_valid[index]), "lw": 2})
        plt.gca().add_artist(imagebox)
plt.axis("off")
plt.show()
```

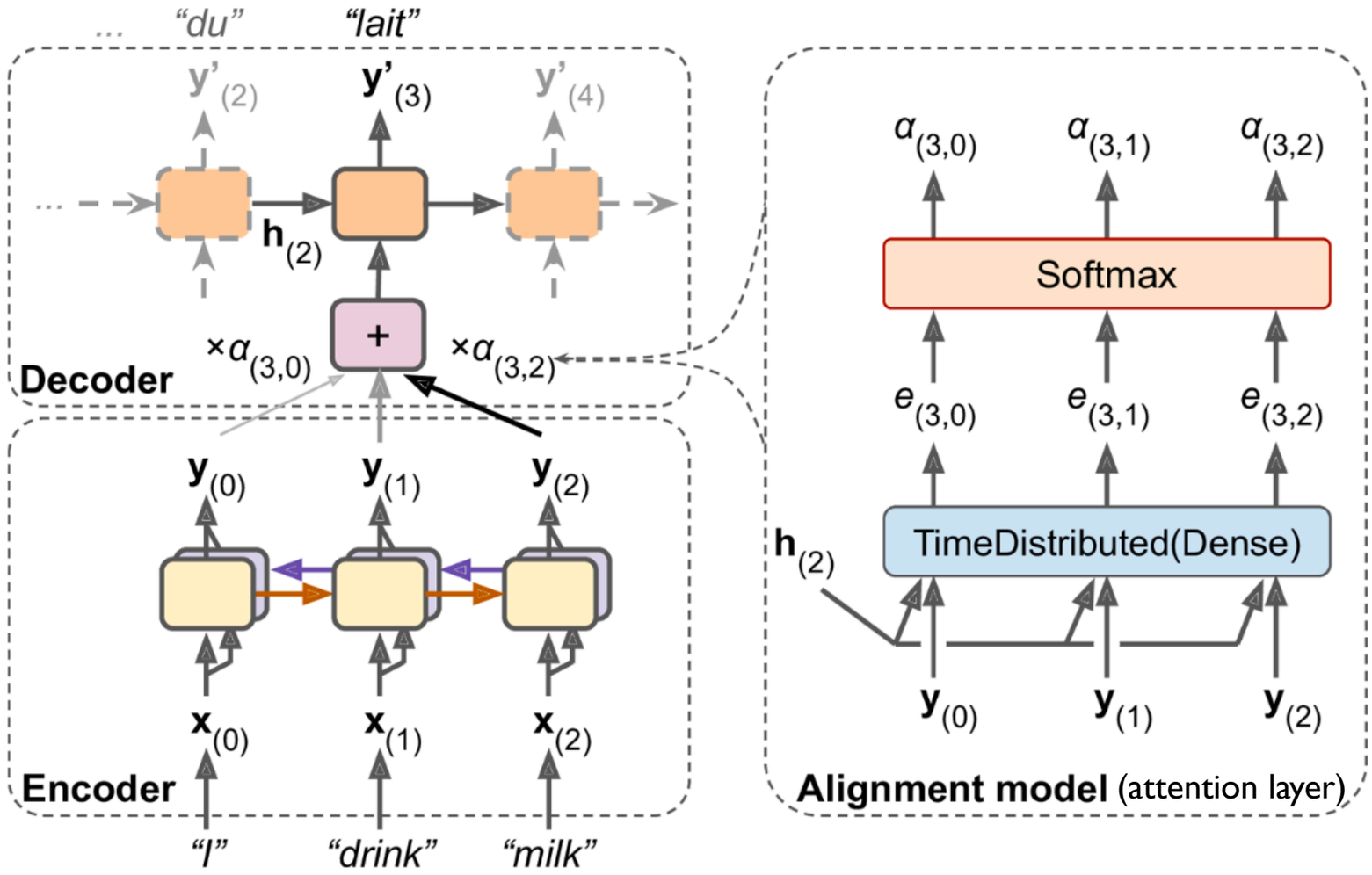
An encoder-decoder network for Neural machine translation (English to French)

- **English is fed to the encoder network**
  - French is also provided as input to the decoder network, but with a step of delay
    - the decoder is given as input the word that it should have output at the previous step



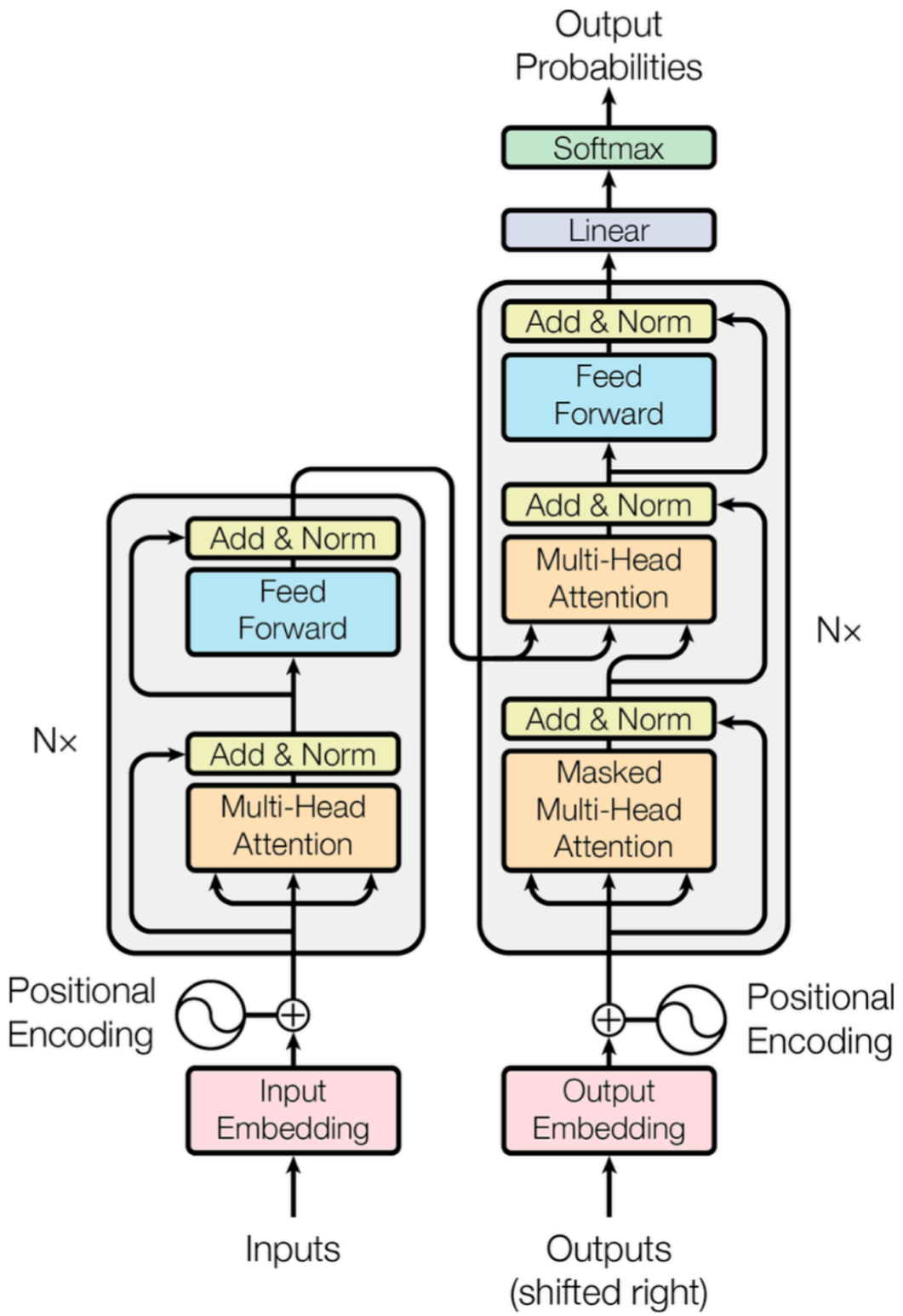
Attention mechanism for Neural machine translation

- **Attention model (2014)**
  - e.g., 앞의 예에서 프랑스어 단어 “lait”는 영어 단어 “milk”와 거리가 멀어 학습이 어렵다
  - Attention model은 decoder가 encoder측 입력 중 어떤 단어에 “집중”해야할 지 학습하게 한다
  - 긴 문장 (>30단어)의 번역을 가능하게 한 핵심 기술



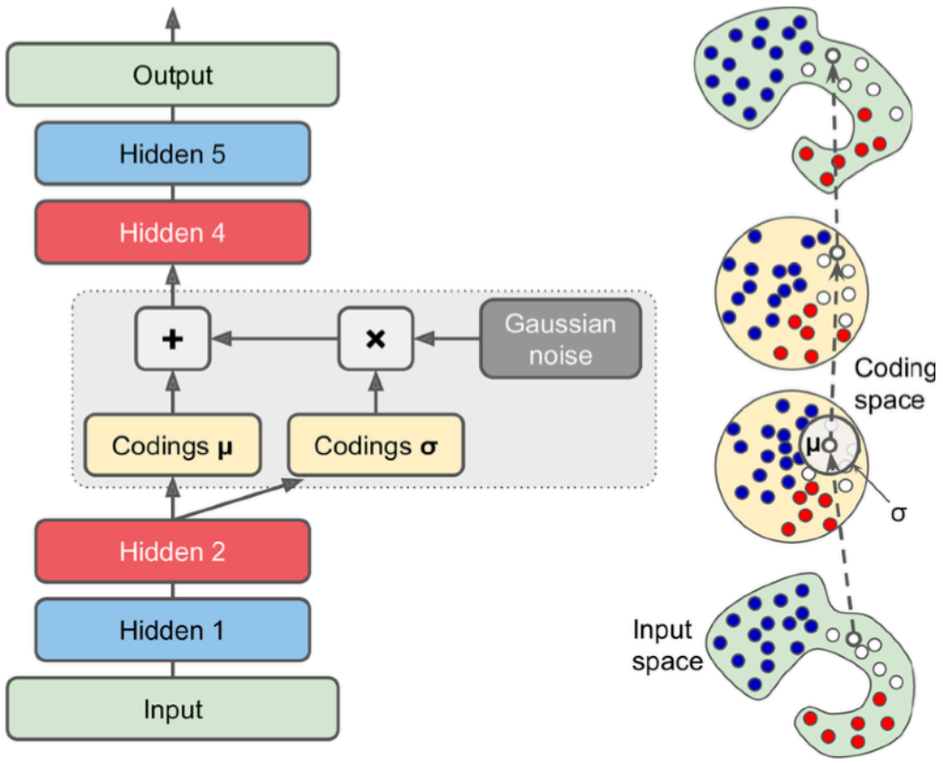
Transformer (2017) — “Attention is All You Need”

- **Transformer**
  - neural machine translation is possible only with attention mechanism, with no recurrent or convolution layers
  - much faster to train and easier to parallelize
  - self-attention was used
    - in encoder side
    - in decoder’s “masked multi-head attention”
  - A positional embedding is a dense vector that encodes the position of a word within a sentence
- **In 2018, GPT, BERT, and other LSTM-based models were reported to advance NMT significantly**
  - leading to the revolutionary chatGPT

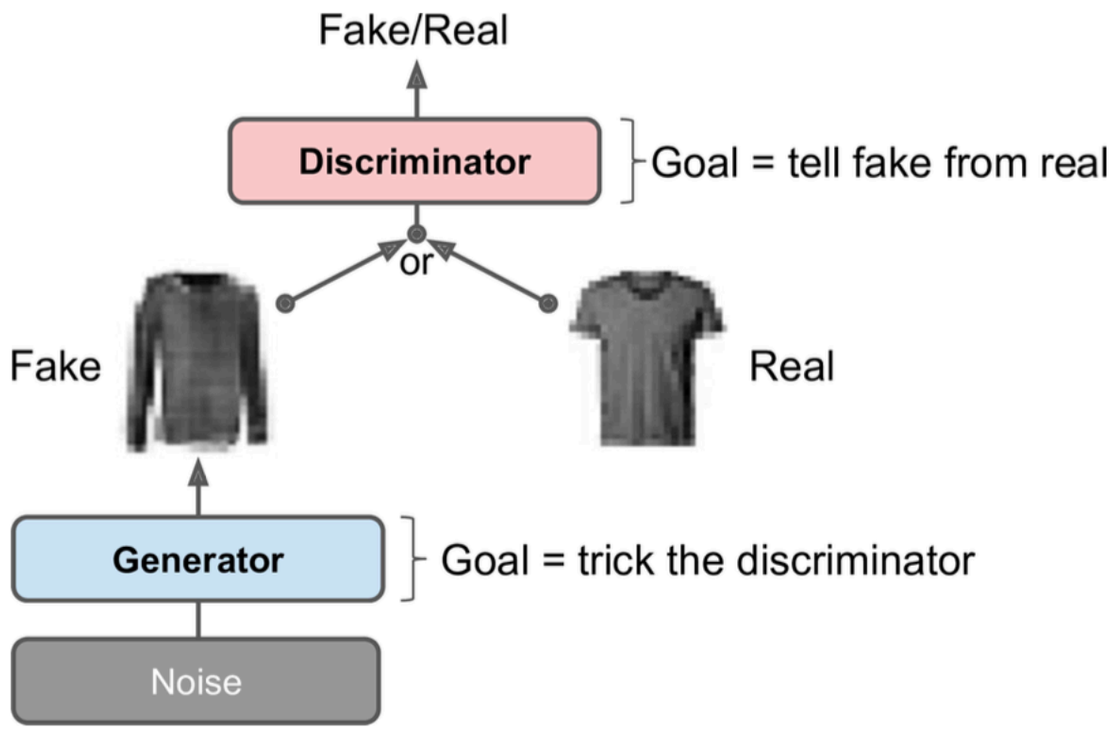


VAE & GAN — generative models

- **VAE**: Autoencoder with stochasticity to the latent variables



- **GAN**: make neural networks compete against each other in the hope that this competition will push them to excel



Summary of session#3

- **RNN (Recurrent neural network)**
  - use “feedback” connections to use past data to predict a future value
- **LSTM (Long Short Term Memory)**
  - LSTM 셀을 활용하여 메모리 길이를 혁신적으로 늘린 RNN
- **Autoencoder**
  - encoder-decoder로 구성되어 있고, 학습 후 encoder는 PCA와 같이 차원 축소에 활용 가능하다
  - decoder는 생성모델에 활용 가능하다
- **Transformer**
  - Recurrent connection없이 Attention 기능만으로 기존의 RNN보다 높은 seq-to-seq 작업 성능을 보인다