MODULE 9: INTRODUCTION TO DAE

BA713 - Machine Learning & AI

CONTENTS

GENERATIVE MODELING

INTRODUCTION TO DAE

ANOMALY DETECTION

SUPERVISED VERSUS UNSUPERVISED LEARNING-Recap

Supervised Learning

- **Data:** features + labels
- **Objective:** Learn to match the features with labels
- **Example:** Classification, regression, Object detection, etc.

Unsupervised Learning

- **Data:** only features (no labels)
- **Objective:** Learn underlying data structure and relationships
- **Example:** Clustering, Feature selection, dimensionality reduction, etc.

GENERATIVE VERSUS DISCRIMINATIVE MODELS

Generative models:

- can generate new data instances.
- learn the data distribution using **unsupervised learning**
- aim at learning the true data distribution of the training observations to generate new observations with some variations
- Exact distribution→ not always possible
- given a set of data instances X and a set of labels Y
- capture the joint probability p(X, Y), or just p(X) if no labels are available
- For example, model that can predict the next word in a sequence
- **Example models**→ GAN and its variants, DAE and its variants

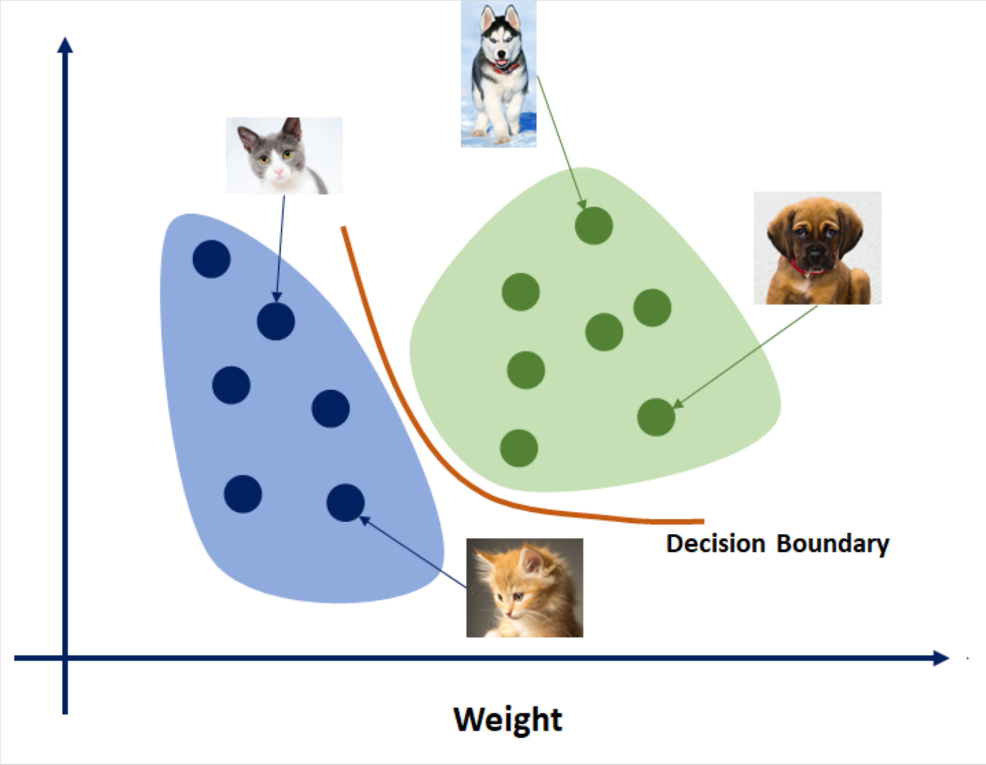
Discriminative models:

- discriminate between different kinds of data instances
- **supervised learning**: e.g., classification
- capture the conditional probability p(Y | X)
- For example, how likely a particular label can apply to a specific observation/instance

Source: https://developers.google.com

GENERATIVE VERSUS DISCRIMINATIVE MODELS

Generative Model



Discriminative Model

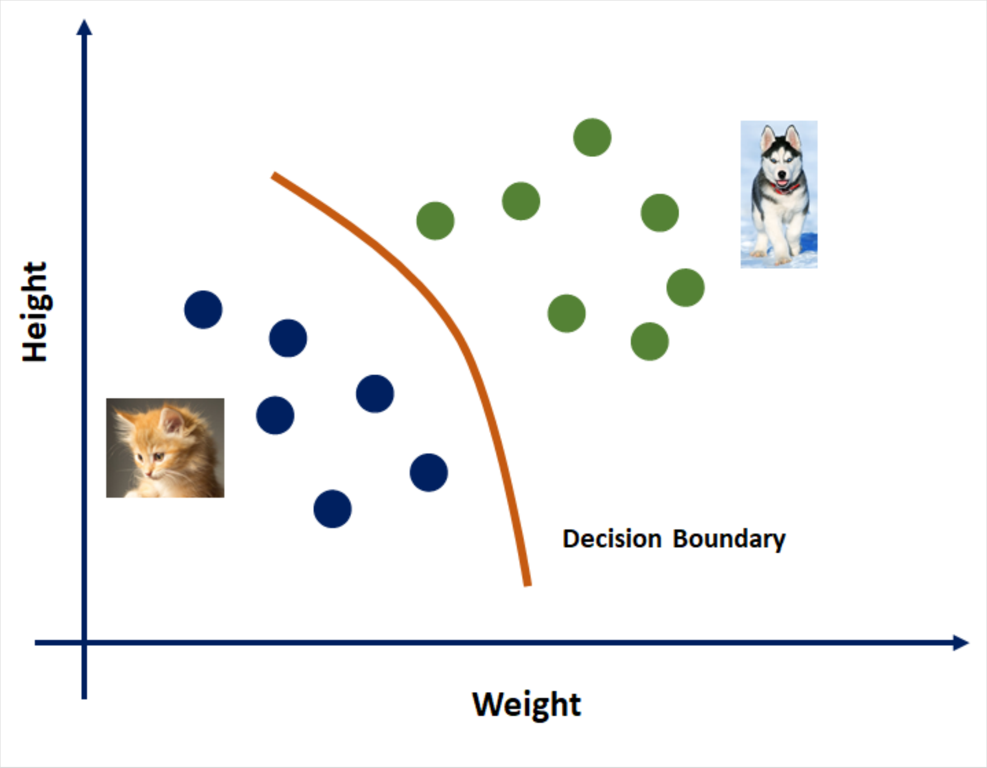
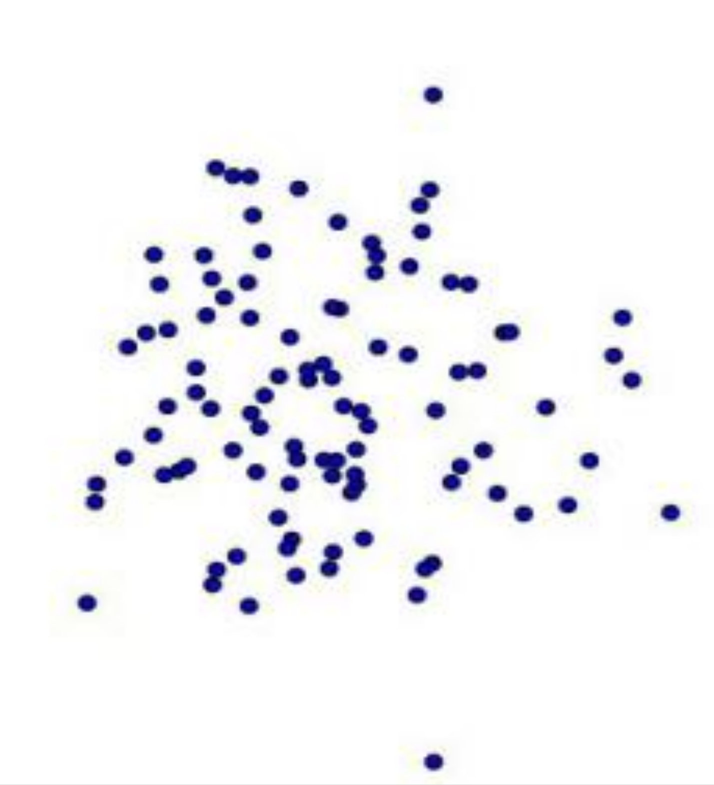


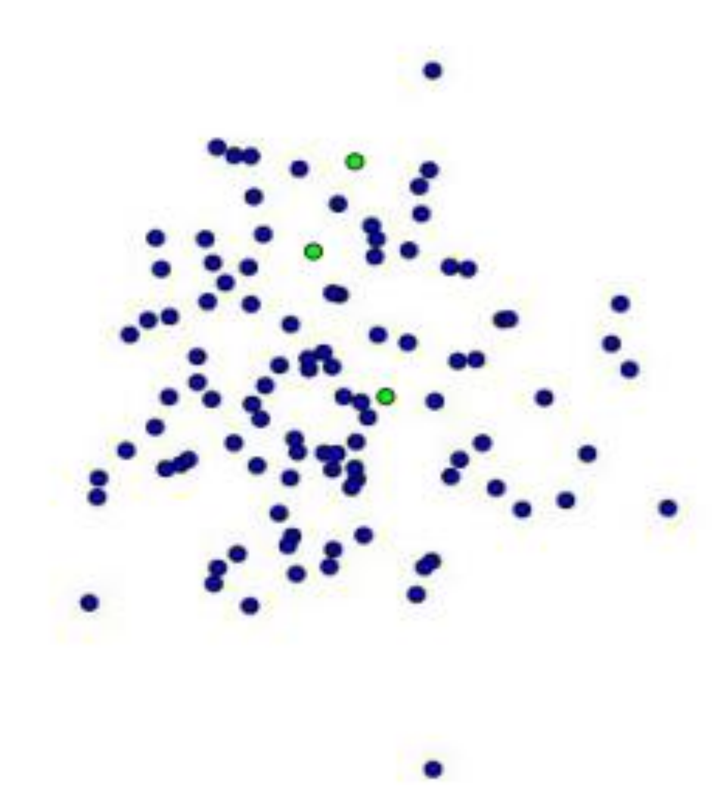
Image Source: https://medium.com/@jordi299/about-generative-and-discriminative-models-d8958b67ad32

GENERATIVE MODELING

- Generate samples with the same probabilistic distribution as input samples



Input data samples in blue color



Generated samples in green color

Source: https://www.cs.toronto.edu/~slwang/

NEED FOR GENERATIVE MODELING

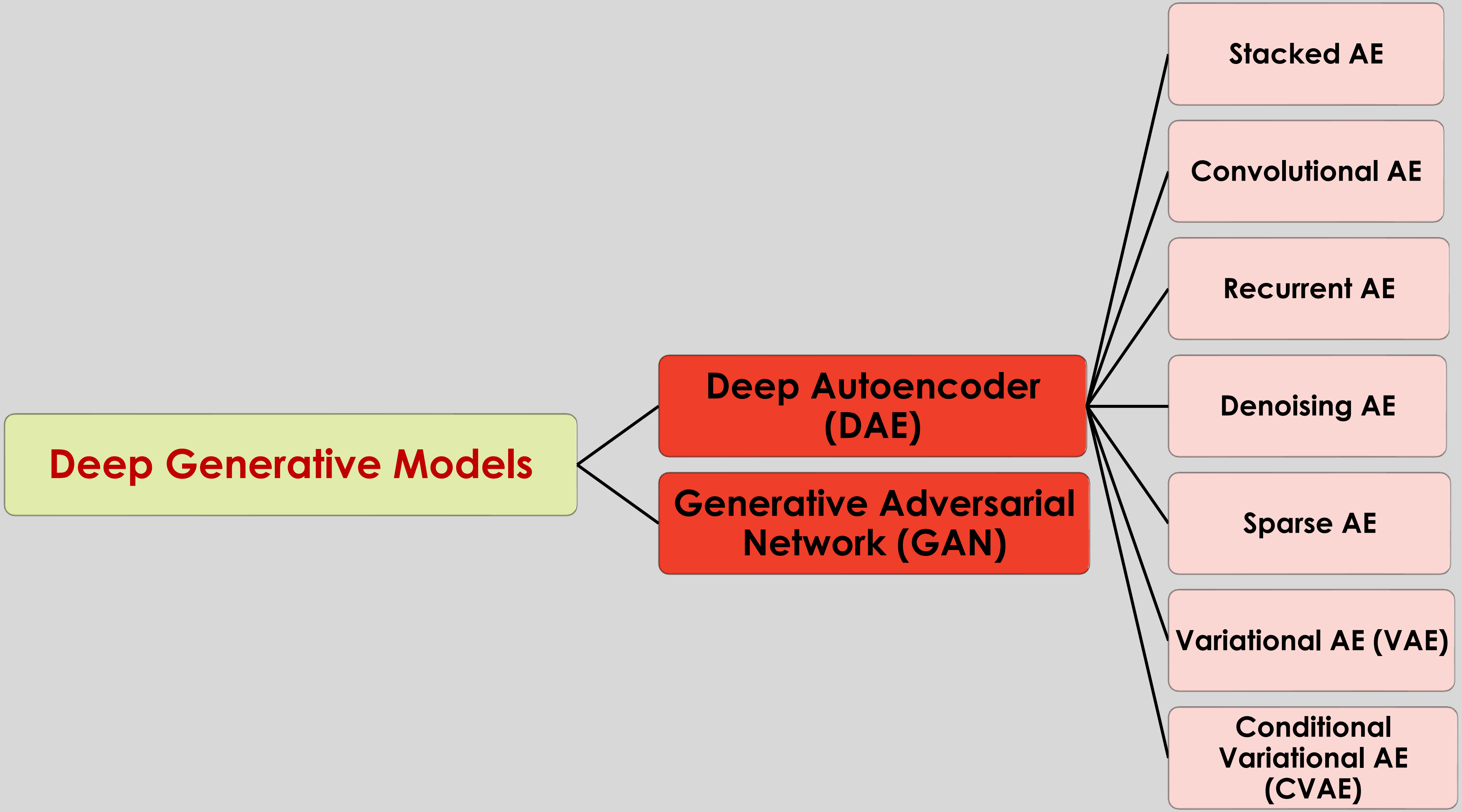
- Learn underlying features in the dataset without the need for labels
- Representative datasets
- More learning samples
- Solve the class imbalance problem

Homogenous skin color and pose

Diverse skin color, pose, and illumination

Source: introtodeeplearning.com

DEEP GENERATIVE MODELS CLASSIFICATION



Deep Generative Models

Deep Autoencoder (DAE)

Generative Adversarial Network (GAN)

Stacked AE

Convolutional AE

Recurrent AE

Denoising AE

Sparse AE

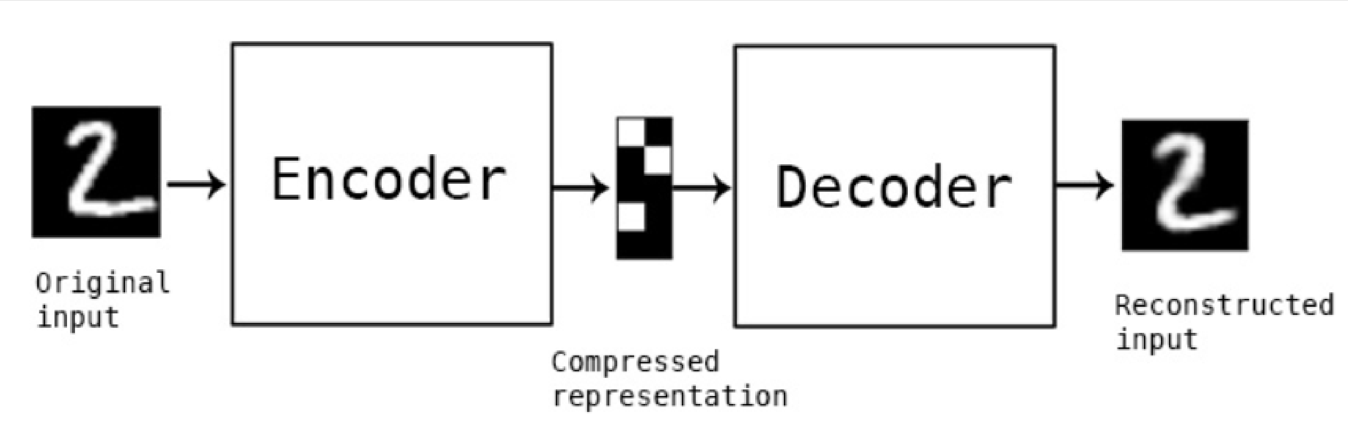
Variational AE (VAE)

Conditional Variational AE (CVAE)

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

AUTOENCODER MODEL

- data compression algorithm
- Implemented with neural networks
- **Self-supervised learning**→ Supervised Learning with automatically generated labels
- **Data-specific**→ compress data similar to what they have been trained on
- **Lossy**→ decompressed outputs will be degraded compared to the original inputs
- **Learn automatically**→ easy to train, no feature engineering required



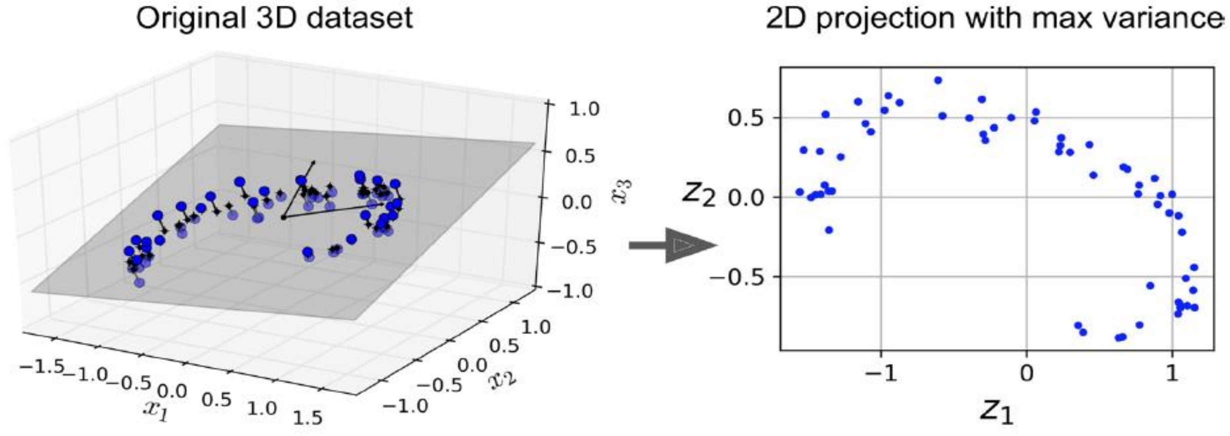
Three important components of an AE model

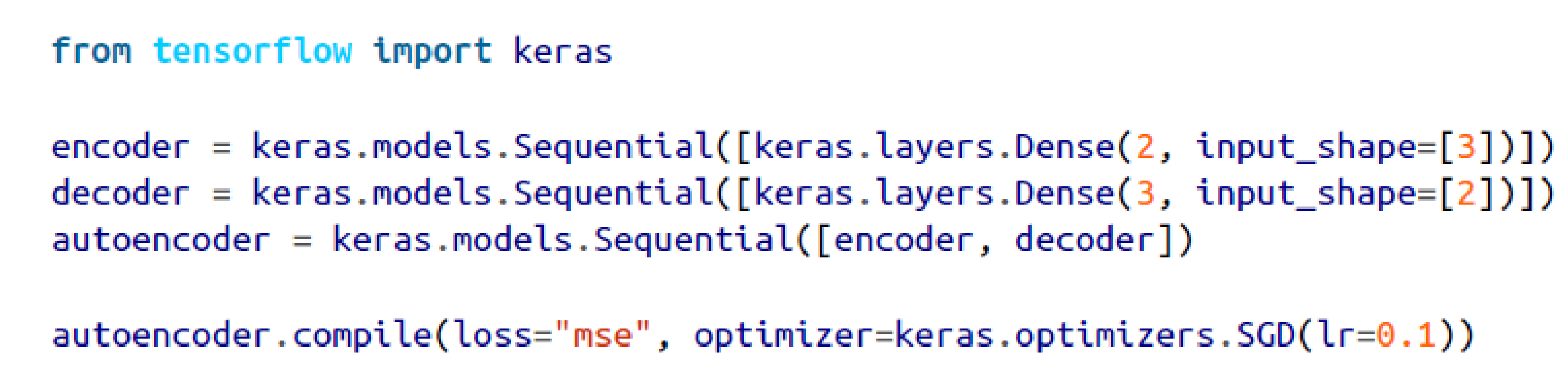
- **Encoding function→ Neural Network**
- **Decoding function→ Neural Network**
- **Loss function→ distance between input and generated output**

Source: https://blog.keras.io/building-autoencoders-in-keras.html

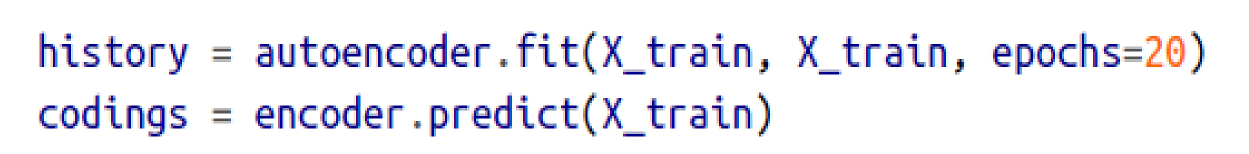
UNDERCOMPLETE LINEAR AUTOENCODER (AE)

- Shallow Machine Learning
- uses only linear activations
- cost function is the mean squared error
- Similar to PCA





```
from tensorflow import keras

encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])
decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])
autoencoder = keras.models.Sequential([encoder, decoder])

autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=0.1))
```

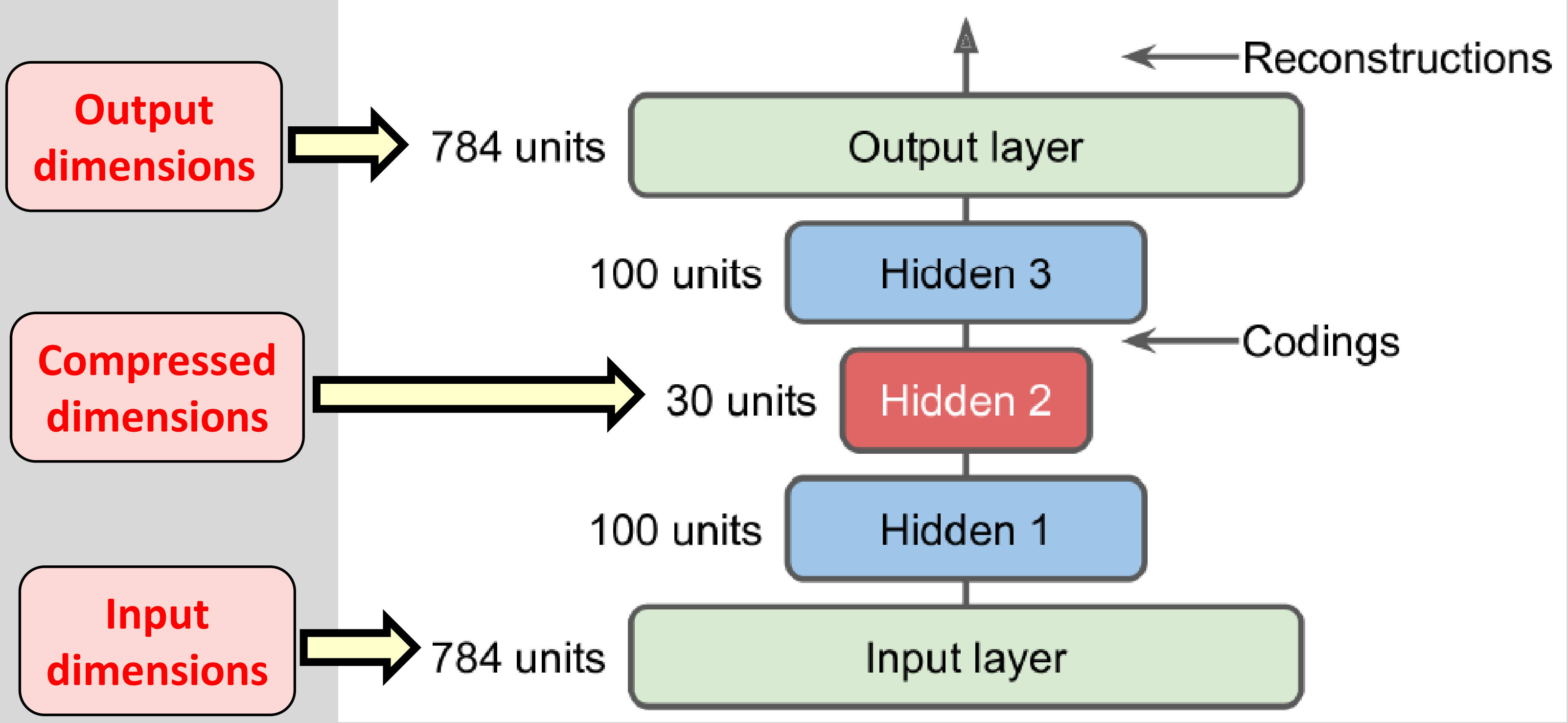


```
history = autoencoder.fit(X_train, X_train, epochs=20)
codings = encoder.predict(X_train)
```

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

STACKED/DEEP AUTOENCODER (SAE/DAE)

- have multiple hidden layers
- learn more complex codings
- Symmetric architecture→ central hidden layer or coding layer
- sandwich



Output dimensions

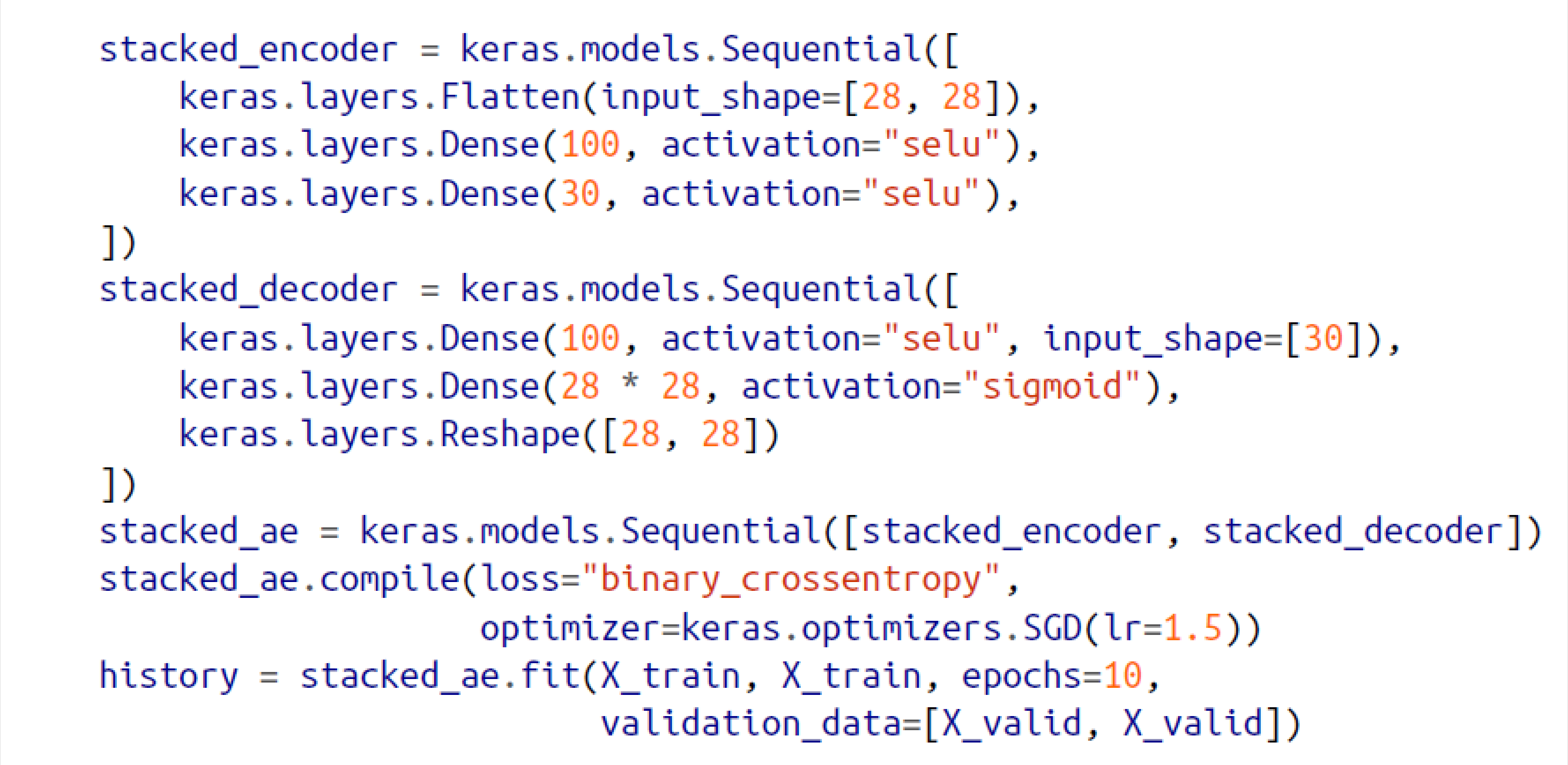
Compressed dimensions

Input dimensions

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

STACKED/DEEP AUTOENCODER EXAMPLE

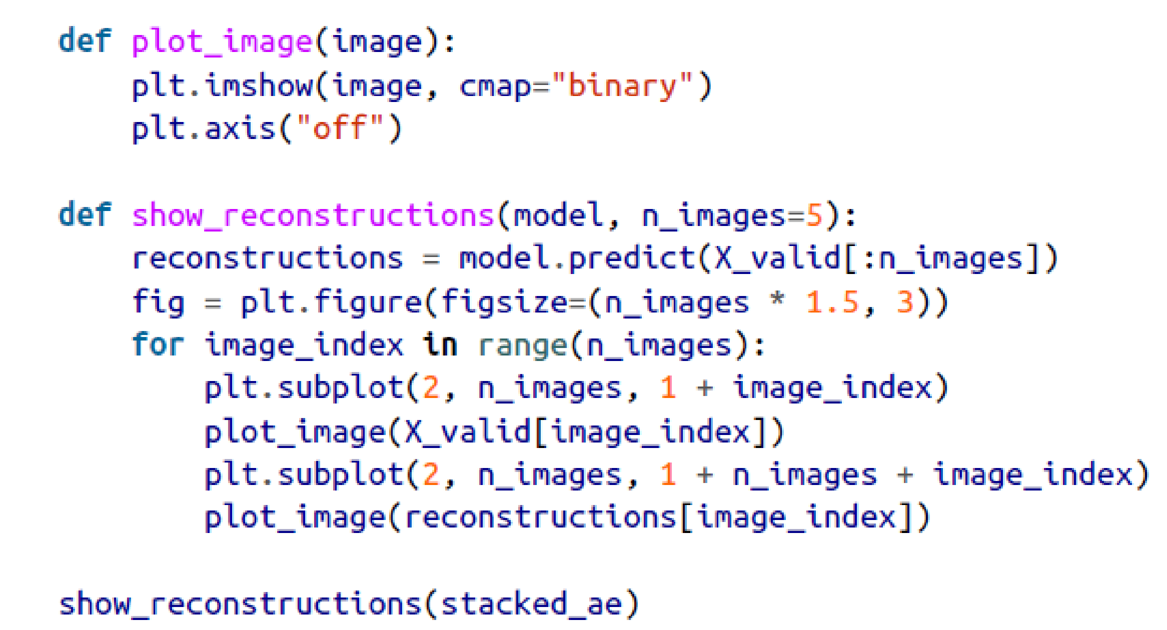
- Two Deep Neural Networks→ Encoder and Decoder
- Multiple hidden layers for each network



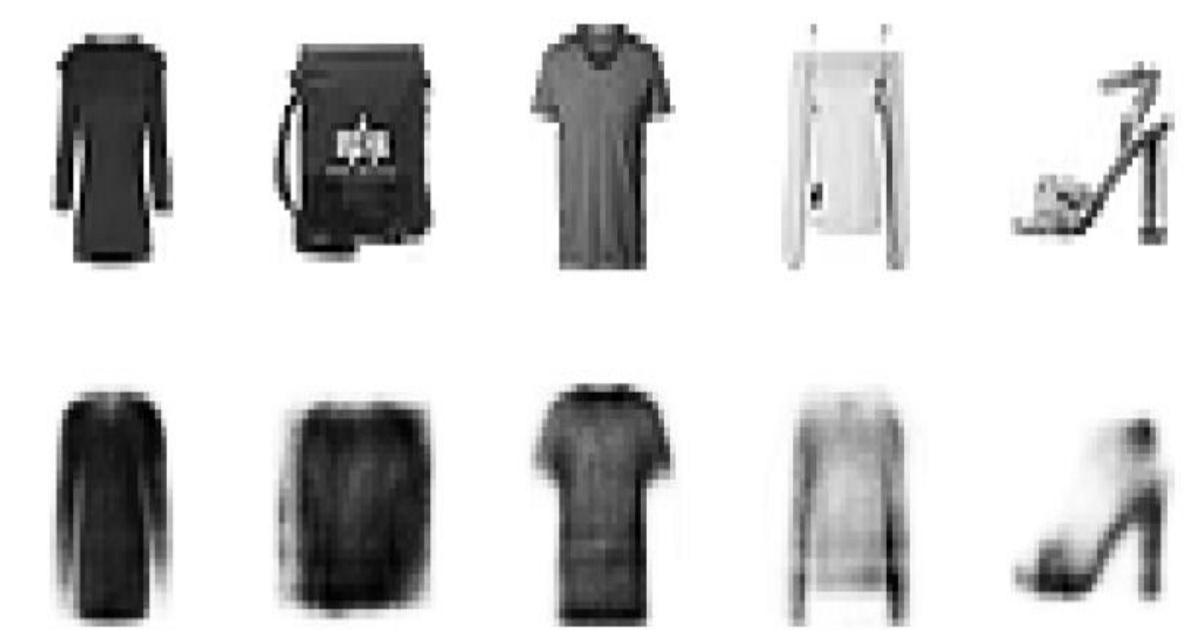
```
stacked_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu"),
])
stacked_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
stacked_ae.compile(loss="binary_crossentropy",
                   optimizer=keras.optimizers.SGD(lr=1.5))
history = stacked_ae.fit(X_train, X_train, epochs=10,
                         validation_data=[X_valid, X_valid])
```

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

VISUALIZING THE RECONSRUCTION OF DAE MODEL



```
def plot_image(image):
    plt.imshow(image, cmap="binary")
    plt.axis("off")

def show_reconstructions(model, n_images=5):
    reconstructions = model.predict(X_valid[:n_images])
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plot_image(X_valid[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])

show_reconstructions(stacked_ae)
```



Original images (top) and their reconstructions (bottom)

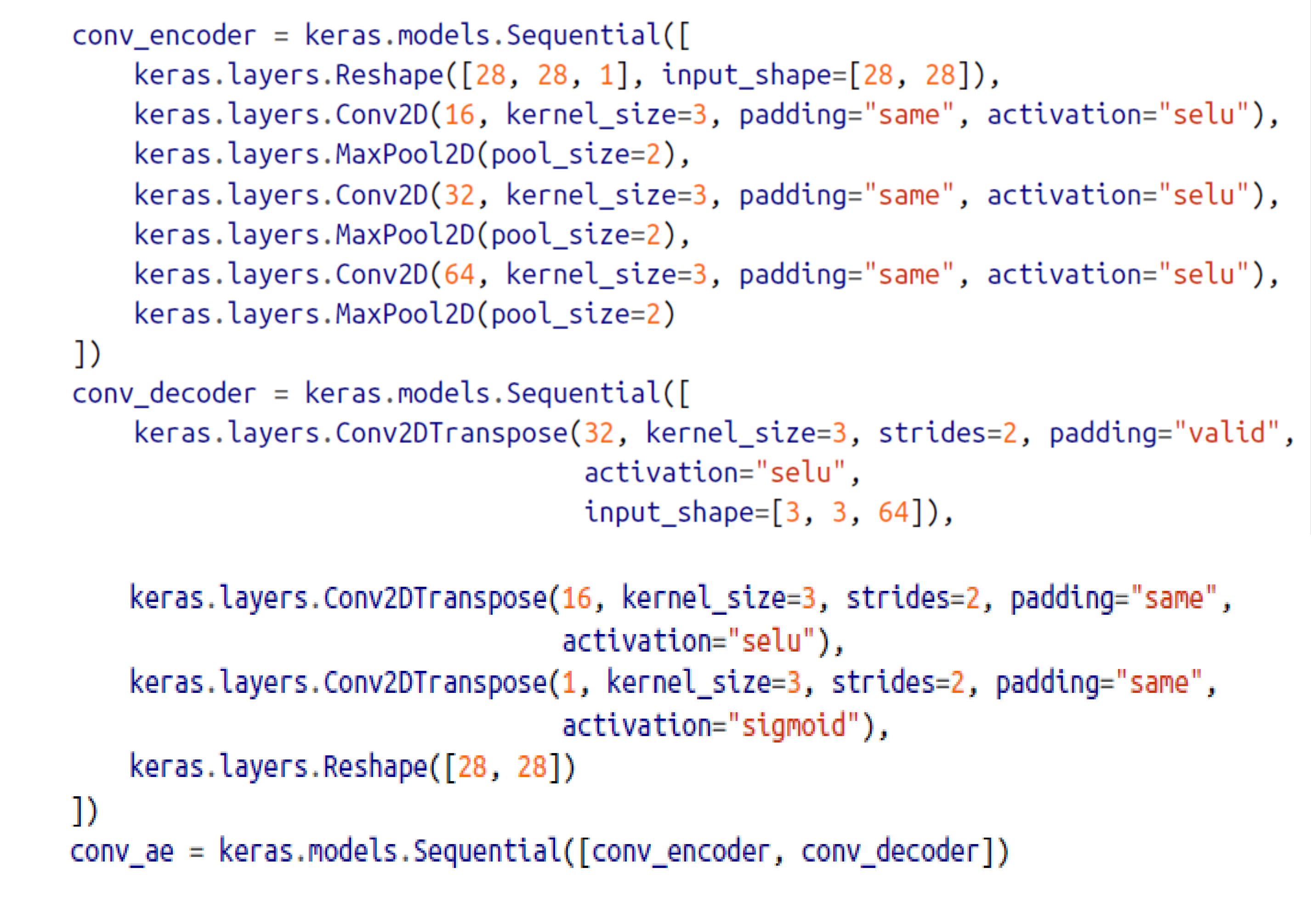
Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

CONVOLUTIONAL AUTOENCODER

- work with huge image datasets
- unsupervised pretraining or dimensionality reduction
- encoder is a regular CNN composed of convolutional layers and pooling layers

- **Encoder→** reduces the inputs (height and width), increases the depth (feature maps)

- **Decoder→** reverse (transpose) of encoder, upscale image and convert to original dimensions



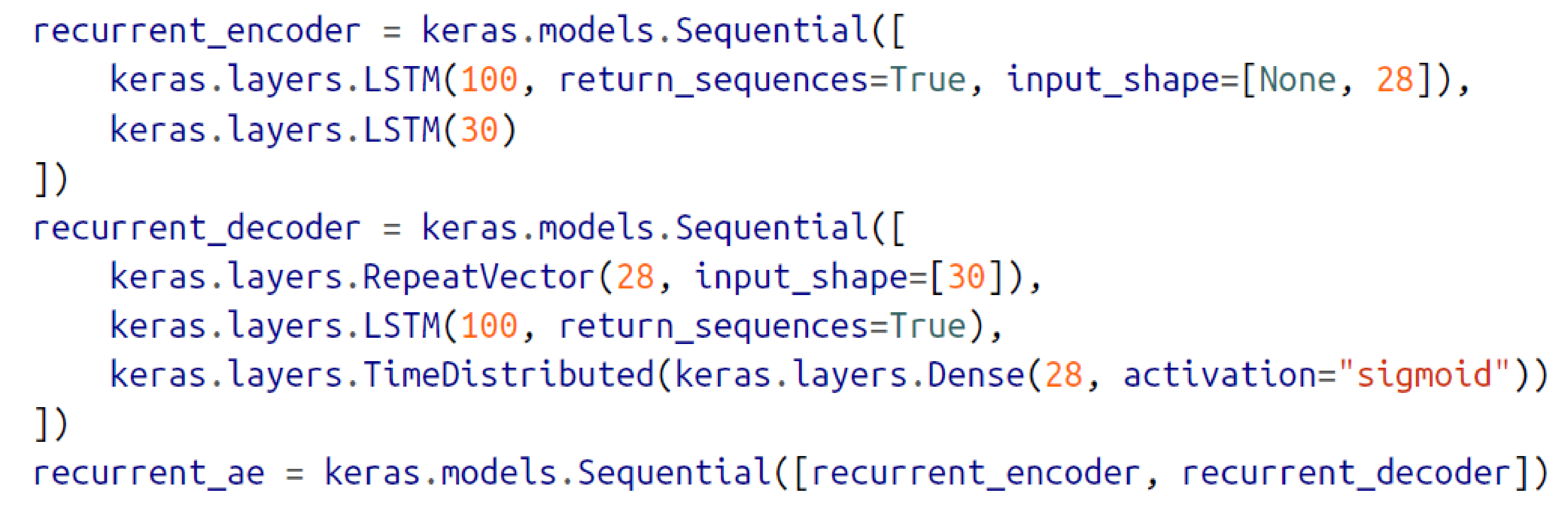
```
conv_encoder = keras.models.Sequential([
    keras.layers.Reshape([28, 28, 1], input_shape=[28, 28]),
    keras.layers.Conv2D(16, kernel_size=3, padding="same", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(32, kernel_size=3, padding="same", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(64, kernel_size=3, padding="same", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2)
])
conv_decoder = keras.models.Sequential([
    keras.layers.Conv2DTranspose(32, kernel_size=3, strides=2, padding="valid",
                                 activation="selu",
                                 input_shape=[3, 3, 64]),

    keras.layers.Conv2DTranspose(16, kernel_size=3, strides=2, padding="same",
                                 activation="selu"),
    keras.layers.Conv2DTranspose(1, kernel_size=3, strides=2, padding="same",
                                 activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
conv_ae = keras.models.Sequential([conv_encoder, conv_decoder])
```

Simple Conv-AE for Fashion MNIST dataset

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

RECURRENT (SEQUENCE-TO-SEQUENCE) AUTOENCODER

- autoencoder for sequences, such as time series or textual information
- unsupervised learning or dimensionality reduction
- **Encoder→** a sequence-to-vector RNN which compresses the input sequence down to a single vector
- **Decoder→** vector-to-sequence RNN that does the reverse



```
recurrent_encoder = keras.models.Sequential([
    keras.layers.LSTM(100, return_sequences=True, input_shape=[None, 28]),
    keras.layers.LSTM(30)
])
recurrent_decoder = keras.models.Sequential([
    keras.layers.RepeatVector(28, input_shape=[30]),
    keras.layers.LSTM(100, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(28, activation="sigmoid"))
])
recurrent_ae = keras.models.Sequential([recurrent_encoder, recurrent_decoder])
```

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

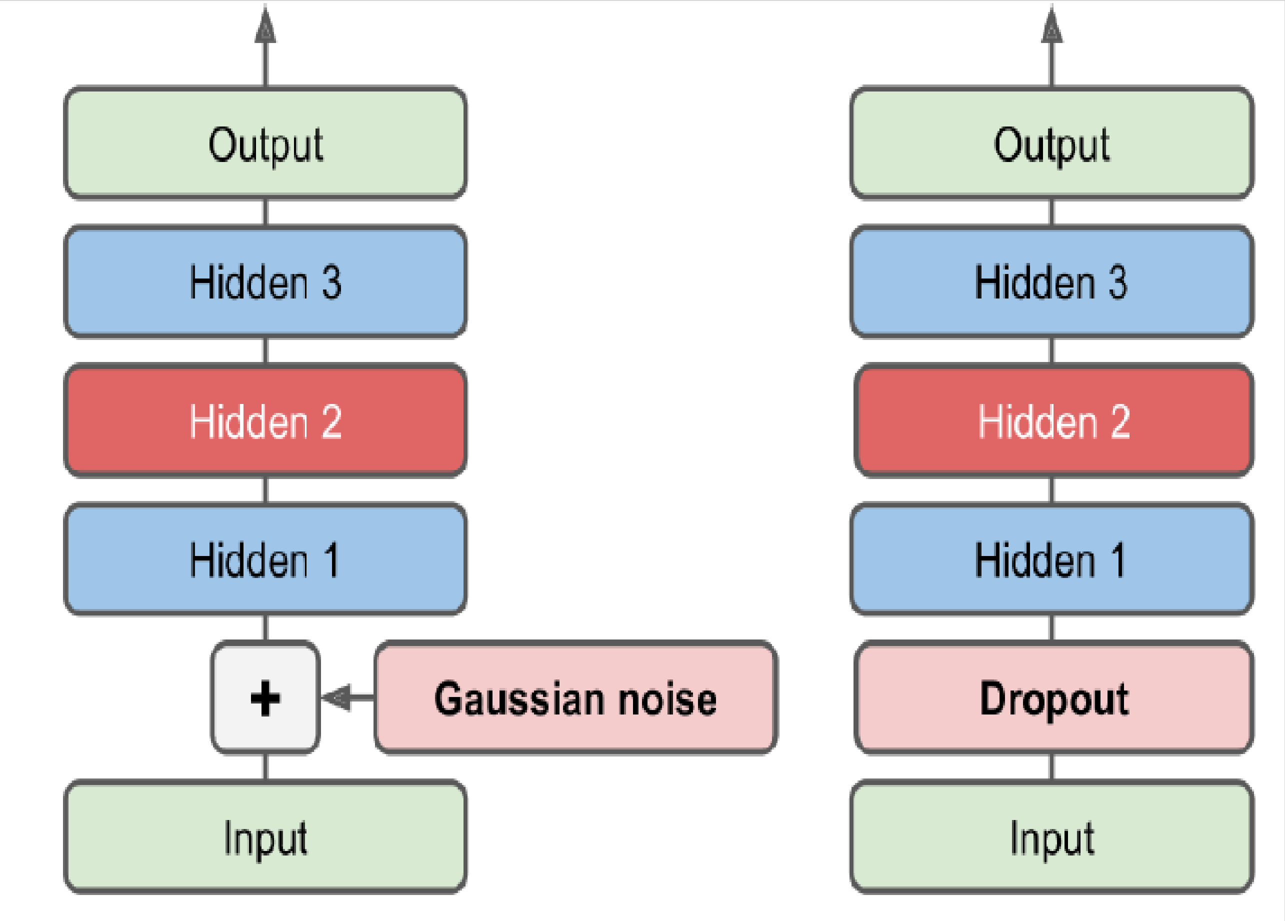
DENOISING AUTOENCODER

- learn useful features by adding noise to its inputs

- training it to recover the original, noise-free inputs

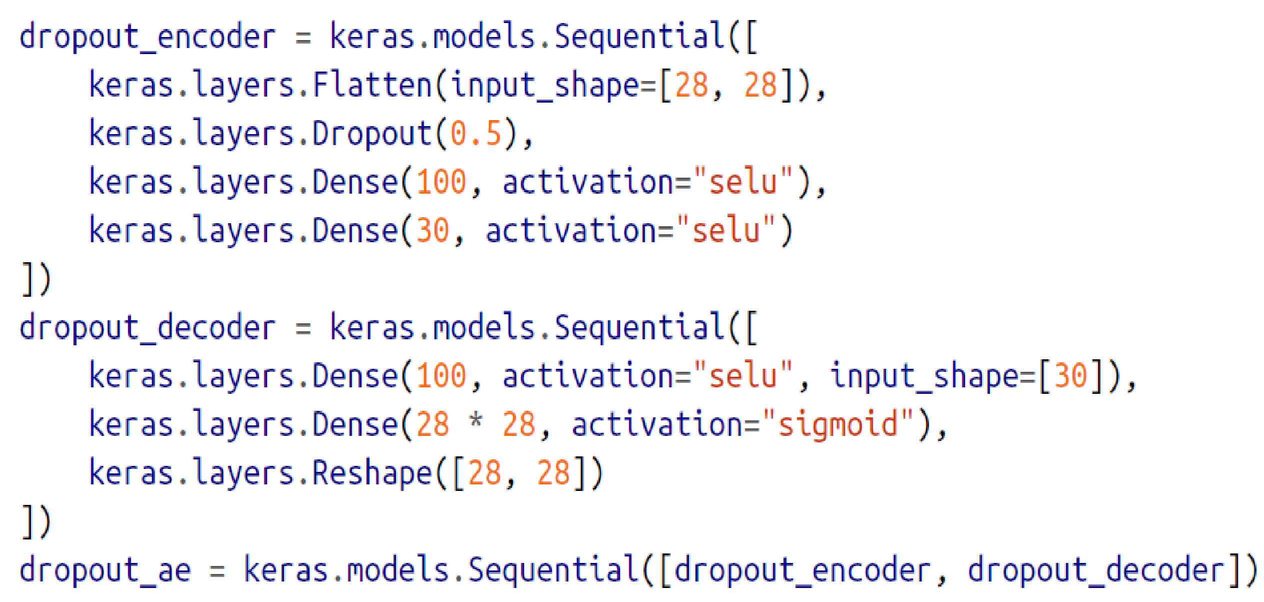
- Noise→ pure Gaussian Noise or randomly adding dropout

- Regular stacked autoencoder→ additional Dropout layer or GaussianNoise layer in encoder

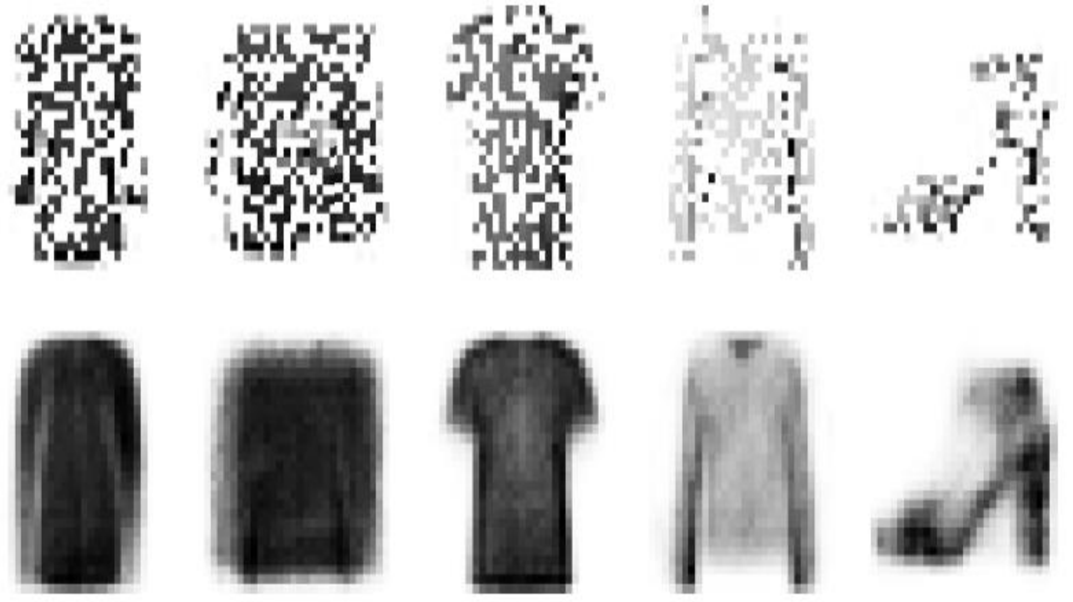


Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

DENOISING AUTOENCODER EXAMPLE



```
dropout_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu")
])
dropout_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])
])
dropout_ae = keras.models.Sequential([dropout_encoder, dropout_decoder])
```

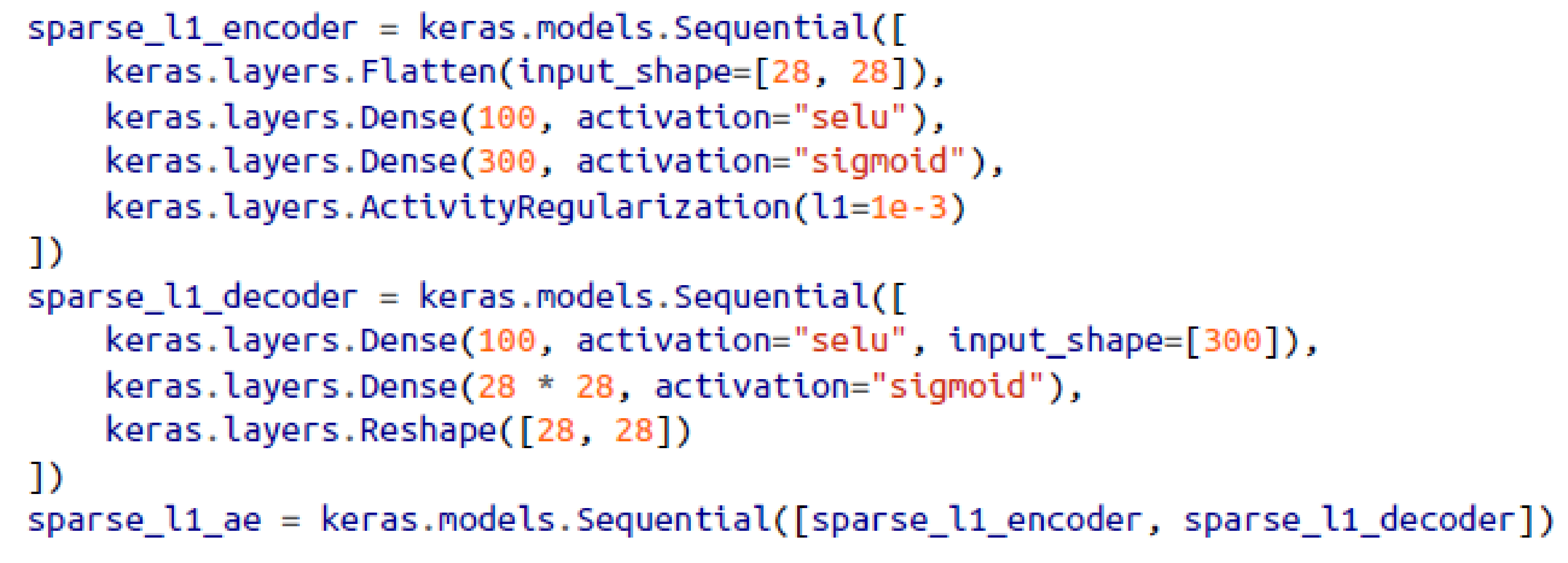


Noisy images (top) and their reconstructions (bottom)

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

SPARSE AUTOENCODER

- **Sparsity**: adding an appropriate term to the cost function
- reduce the number of active neurons in the coding layer
- For example, AE may be pushed to have on average only 5% significantly active neurons in the coding layer
- Small number of activations
- Each neuron in coding layer→ represents a useful feature
- Use **Sigmoid activation** (between 0 and 1) in coding layer
- Add **L1 Regularization** to coding layer's activations
- Decoder→ regular decoder



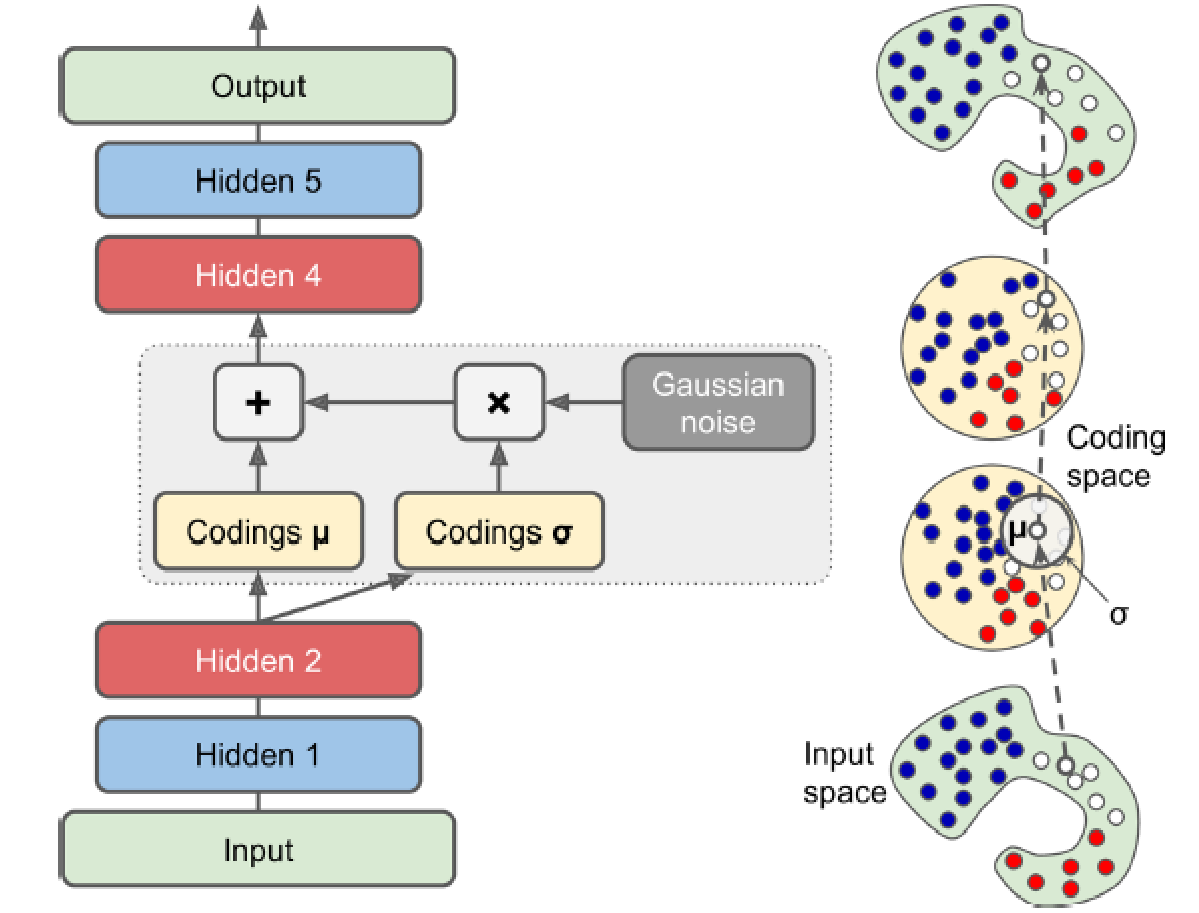
```
sparse_l1_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(300, activation="sigmoid"),
    keras.layers.ActivityRegularization(l1=1e-3)
])
sparse_l1_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[300]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
sparse_l1_ae = keras.models.Sequential([sparse_l1_encoder, sparse_l1_decoder])
```

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

VARIATIONAL AUTOENCODER (VAE)

- Introduced in **2013**
- **Diederik Kingma** and **Max Welling**

- most popular generative AE model
- probabilistic autoencoders
- their outputs are partly determined by chance, even after training
- can generate new instances that look like they were sampled from the training set
- instead of directly producing a coding for a given input, the encoder produces a mean coding **μ** and a standard deviation **σ**
- Actual coding→ sampled randomly→ Gaussian distribution (**μ,σ**)
- **Latent space/Sampling Layer**→ cloud of Gaussian points
- very easily generate a new instance
- sample a random coding from the Gaussian distribution and then decode it



Variational autoencoder (left) and an instance going through it (right)

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

VAE LOSS FUNCTION

Composed of two parts:-

- **Reconstruction Loss:** Loss between the inputs and reconstructions (e.g., *cross entropy*)
- **Latent Loss:** *KL Divergence* between the target distribution (i.e., the Gaussian distribution) and the actual distribution of the codings

The objective function for a VAE model is given below:

$$\log P(X) - D_{KL}[Q(z|X) \| P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X) \| P(z)]$$

- **log P(X) → model our data**
- **Q(z|X) is the encoder network**
- **z is the encoded representation**
- **P(X|z) is the decoder network**
- **DKL[Q(z|X)||P(z|X)] is the overall error**
- **E[log P(X|z)]→ Maximum Likelihood Estimation→ given an input z and an output X→ maximize the conditional distribution P(X|z)**
- **Q(z|X)→ simple distribution**
- **P(z)→true latent distribution**

Source: https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

CONDITIONAL VARIATIONAL AUTOENCODER (CVAE)

- Enhanced VAE model with a conditional variable
- **Kihyuk Sohn, Xinchen Yan**, and **Honglak Lee** in 2015
- VAE→ no control on the data generation process
- problematic if we want to generate some specific data
- For example, generate multiple versions of the handwritten MNIST digit “5”

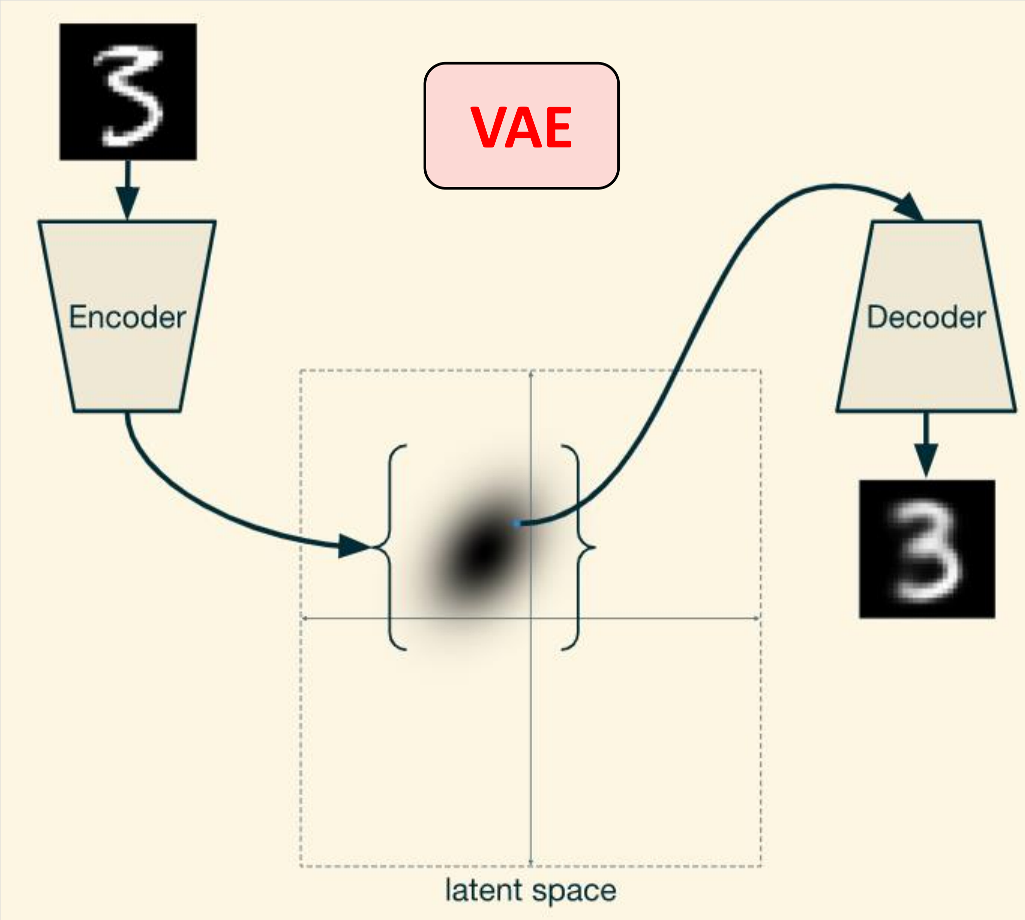
- The **objective function** for a CVAE model is similar to VAE but is conditioned using a variable “c”

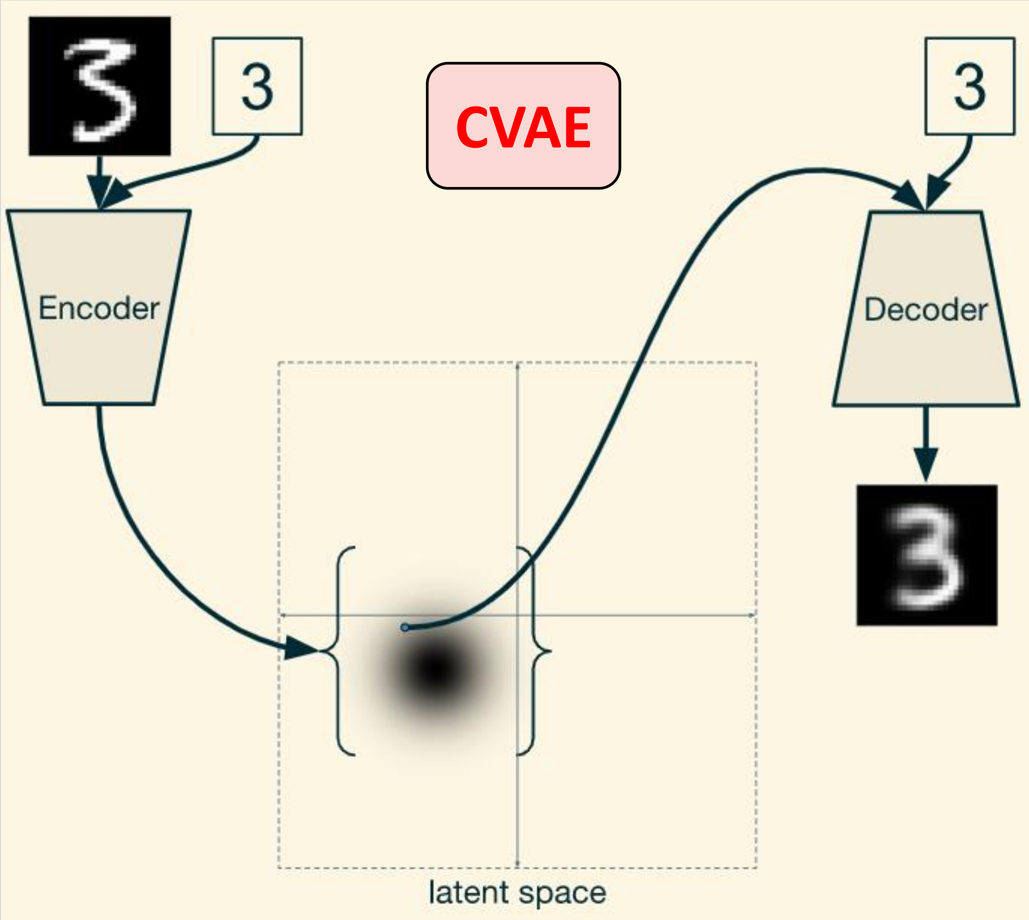
$$\log P(X|c) - D_{KL}[Q(z|X,c) \| P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c) \| P(z|c)]$$

  - we just conditioned all the distributions with a variable “c”
  - Conditional probability distribution (CPD)

Source: https://wiseodd.github.io/techblog/2016/12/17/conditional-vae/

VAE VERSUS CVAE

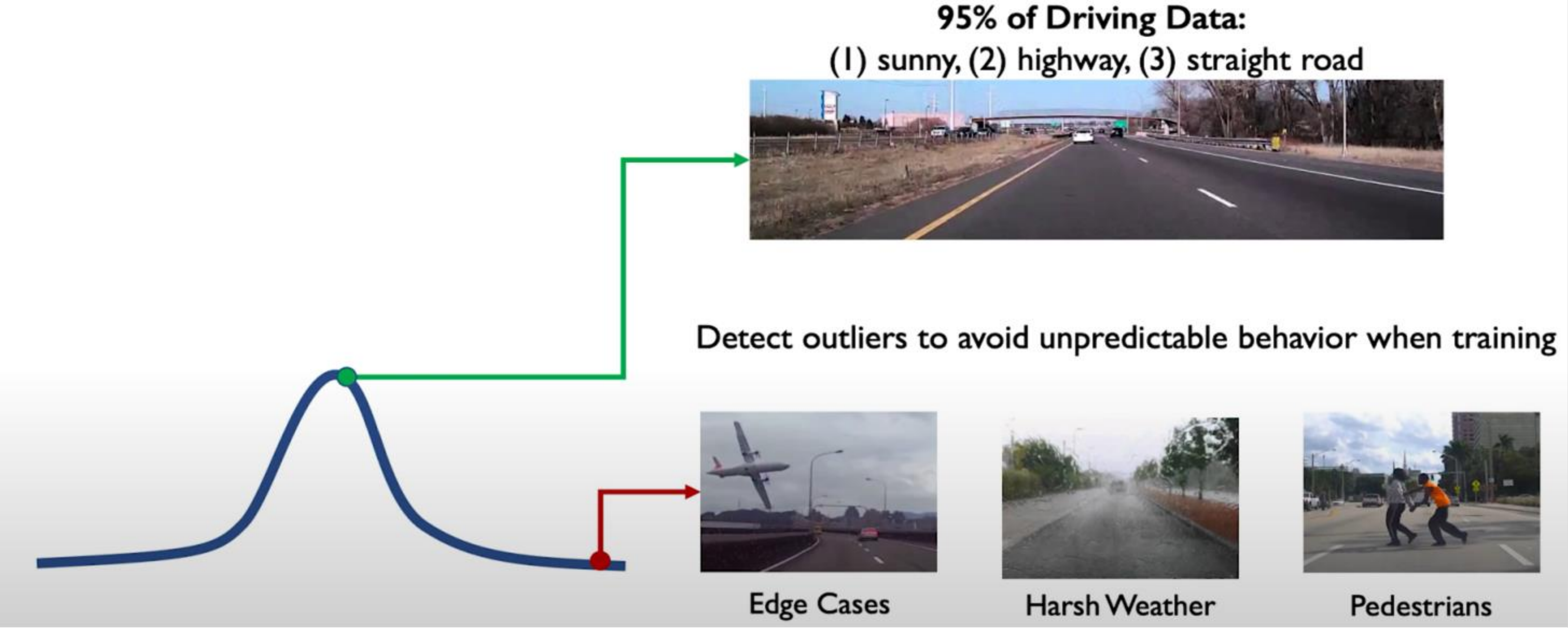




Source: https://ijdykeman.github.io/ml/2016/12/21/cvae.html

ANOMALY/ OUTLIER DETECTION

- DAE models → detect rare anomalies or outliers in the data distribution



95% of Driving Data:
(1) sunny, (2) highway, (3) straight road

Detect outliers to avoid unpredictable behavior when training

Source: introtodeeplearning.com

VAE PROGRAMMING EXERCISE

(Anomaly Detection)

THANKS!

Do you have any questions?