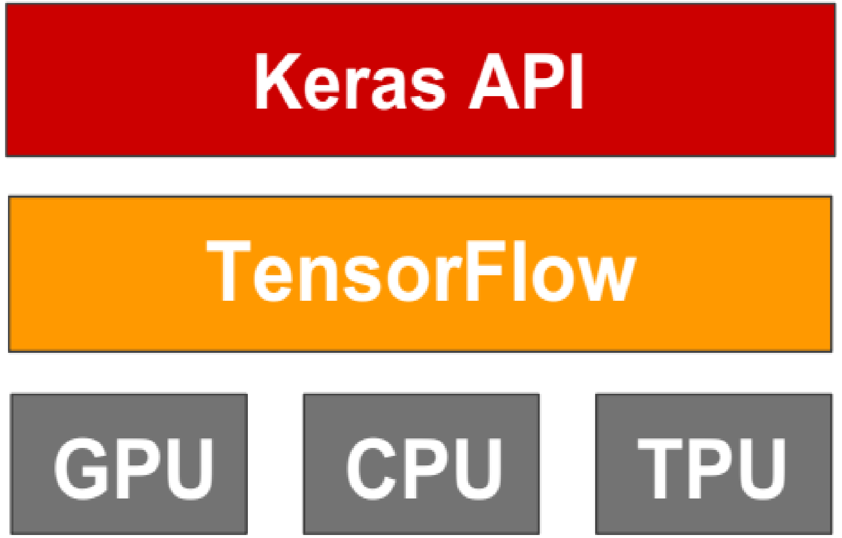
Data Analysis

Practice 5: Intro to TF2&Keras

Dr. Nataliya K. Sakhnenko

Deep Learning Frameworks



Keras API

- TensorFlow is an industry-strength numerical computing framework that can run on CPU, GPU, or TPU. It can automatically compute the gradient of any differentiable expression, it can be distributed to many devices, and it can export programs to various external runtimes.

- Keras is the standard API to do deep learning with TensorFlow.

- The central class of Keras is the Layer. A layer encapsulates some weights and some Layer computation. Layers are assembled into models.

- Before you start training a model, you need to pick an optimizer, a loss, and some metrics, which you specify via the model.compile() method.

- To train a model, you can use the method, which runs mini-batch gradient descent fit() for you. You can also use it to monitor your loss and metrics on "validation data", a set of inputs that the model doesn’t see during training.

- Once your model is trained, use the model.predict() method to generate predictions on new inputs.

Keras API

The Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

https://keras.io/

Recap: softmax

$$T:\ X \to Y,\ Y = \{1, 2, \ldots K\}$$

$$h_\theta(x) = \begin{Bmatrix} P(y=1 \mid x;\ \theta) \\ P(y=2 \mid x;\ \theta) \\ \ldots \\ P(y=K \mid x;\ \theta) \end{Bmatrix} = \frac{1}{\sum_{i=1}^{K} \exp(\theta_i^T x)} \begin{pmatrix} \exp(\theta_1^T x) \\ \exp(\theta_2^T x) \\ \ldots \\ \exp(\theta_K^T x) \end{pmatrix}$$

One-hot encoding

$$\begin{array}{ccccc} \begin{bmatrix} 1 \\ 0 \\ \ldots \\ 0 \end{bmatrix}, & \begin{bmatrix} 0 \\ 1 \\ \ldots \\ 0 \end{bmatrix}, \ldots & \begin{bmatrix} 0 \\ 0 \\ \ldots \\ 1 \end{bmatrix}, & Y \in R^K \\ 1, & 2, \ldots & K & \end{array}$$

The Sequential model, the most approachable API—it's basically a Python list. As such, it's limited to simple stacks of layers

The functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
```

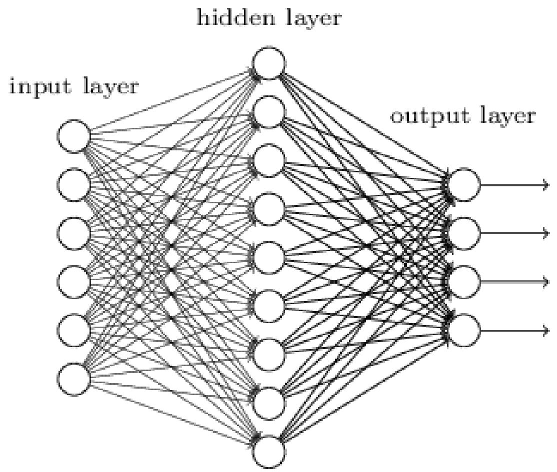
Model Training APIs

https://keras.io/api/models/model_training_apis/	
Compile method	Model.compile(optimizer="rmsprop", loss=None, metrics=None, …)
Fit method	Model.fit(x=None, y=None, batch_size=None, epochs=1, verbose="auto", callbacks=None, validation_split=0.0, validation_data=None, …)
Evaluate method	Model.evaluate(x=None, y=None, batch_size=None, verbose=1, …)
Predict method	Model.predict(x, …)

- **x**: Input data. It could be e.g. a numpy array

Last-layer activation and Loss function

Problem type	Last-activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass classification	softmax	categorical_crossentropy
Regression	None	mse



Installation & compatibility

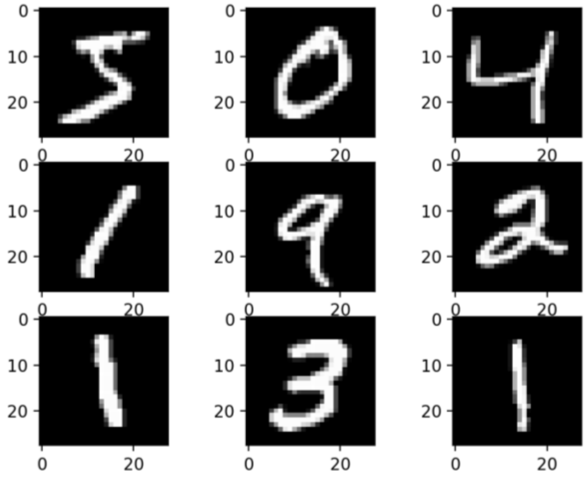
Keras comes packaged with TensorFlow 2 as tensorflow.keras. To start using Keras, simply install TensorFlow 2 (https://www.tensorflow.org/install)

Keras/TensorFlow are compatible with:

- Python 3.7–3.10
- Ubuntu 16.04 or later
- Windows 7 or later
- macOS 10.12.6 (Sierra) or later (no GPU support)

MNIST - Logistic Regression

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

input_dim = 28 * 28 #784

x_train = x_train.reshape(60000, input_dim)
x_test = x_test.reshape(10000, input_dim)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```



One-hot encoding

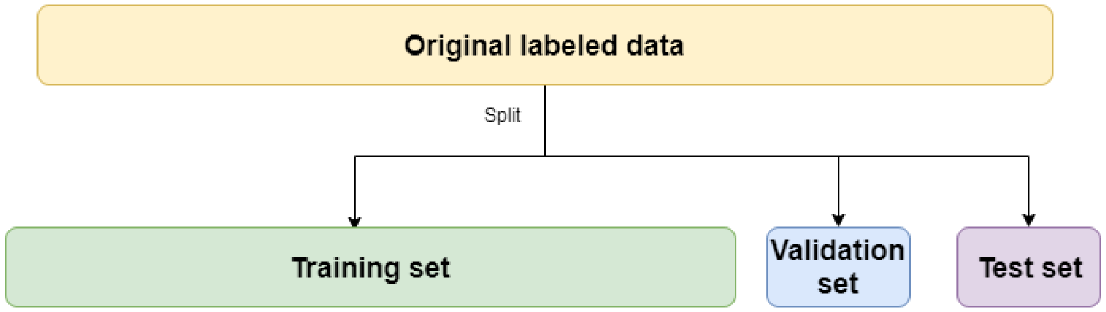
MNIST - Logistic Regression

```
model = keras.Sequential([layers.Dense(num_classes, activation="softmax")])
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2, verbose = 2)
```

Model summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
```

```
score = model.evaluate(x_test, y_test)
```



MNIST - Multilayer FC ANN

```
model =  keras.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',   optimizer='adam', metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_13 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_14 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

Callbacks `my_callbacks = [tf.keras.callbacks.EarlyStopping(patience=2)]`

Using Dropout

```
model =  keras.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

Using Batch Normalization

```
model =  keras.Sequential()

model.add(layers.Dense(512, input_shape=(784,)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(512))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

MLP for text classification: Sentiment analysis, IMDb

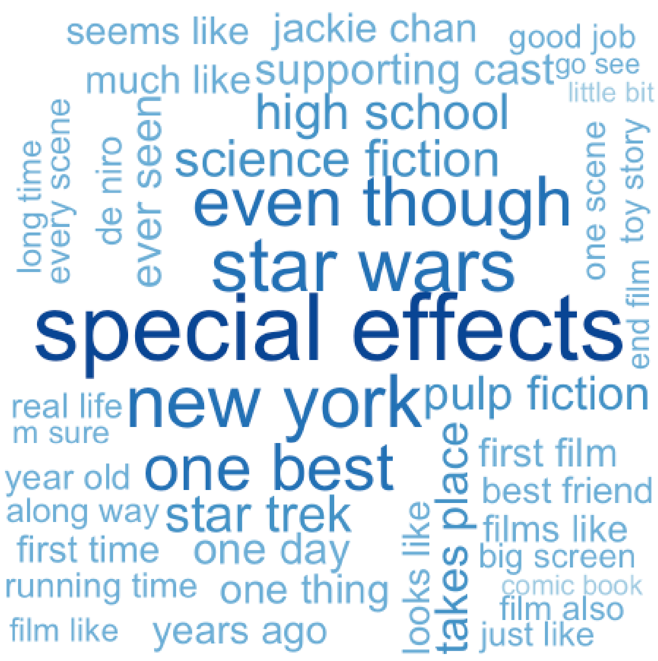
http://ai.stanford.edu/~amaas/data/sentiment/

- ✓ We'll be using a dataset of 50,000 movie reviews taken from IMDb.
- ✓ The data is split evenly with 25k reviews intended for training and 25k for testing your classifier.
- ✓ Each set has 12.5k positive and 12.5k negative reviews.
- ✓ IMDb lets users rate movies on a scale from 1 to 10. To label these reviews the curator of the data labeled anything with ≤ 4 stars as negative and anything with ≥ 7 stars as positive. Reviews with 5 or 6 stars were left out.

Sentiment analysis (also known as **opinion mining** or **emotion AI**) refers to the use of NLP to systematically identify, extract, quantify, and study affective states and subjective information.

Sentiment Analysis, IMDB movies

```
from tensorflow.keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data()

x_train = vectorize_sequences(train_data)

model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")])

model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history = model.fit()
```

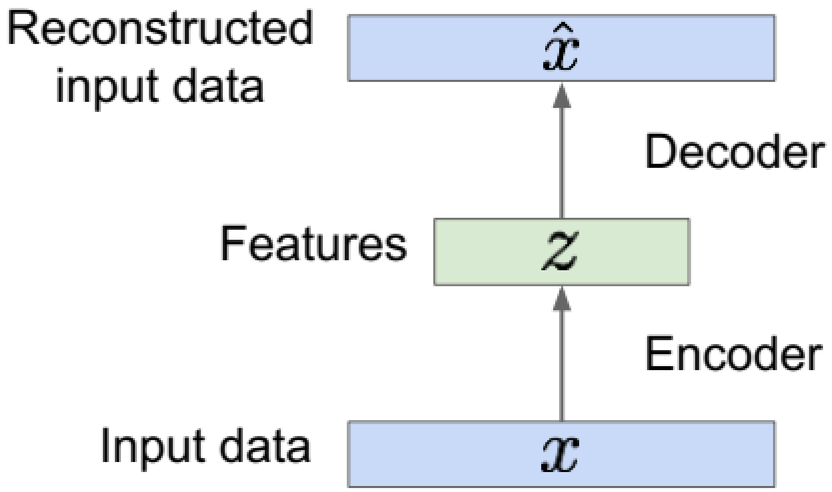


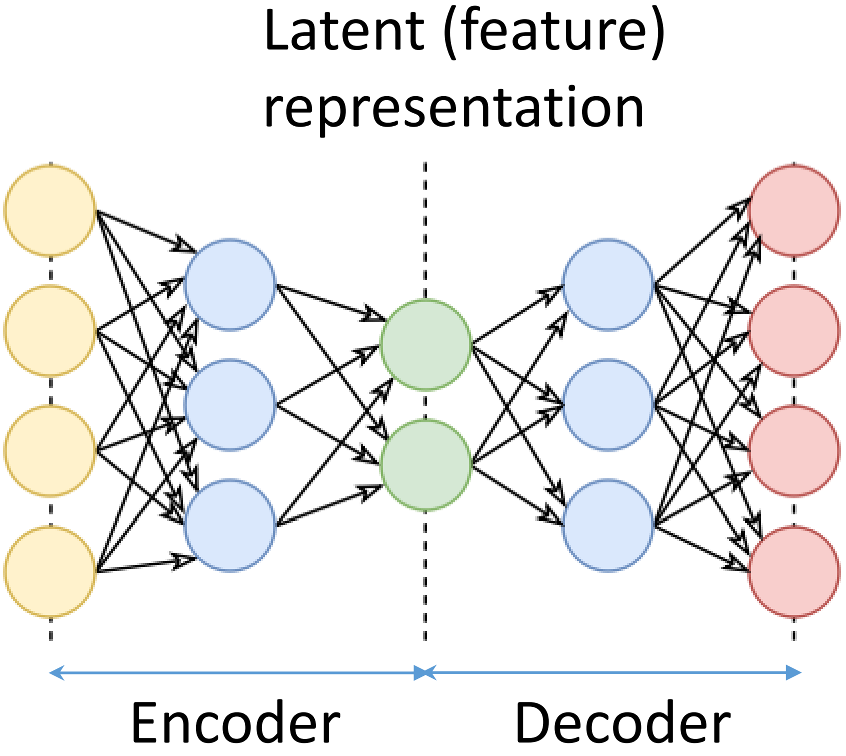
Bi-grams

Autoencoders

AE for dimensionality reduction

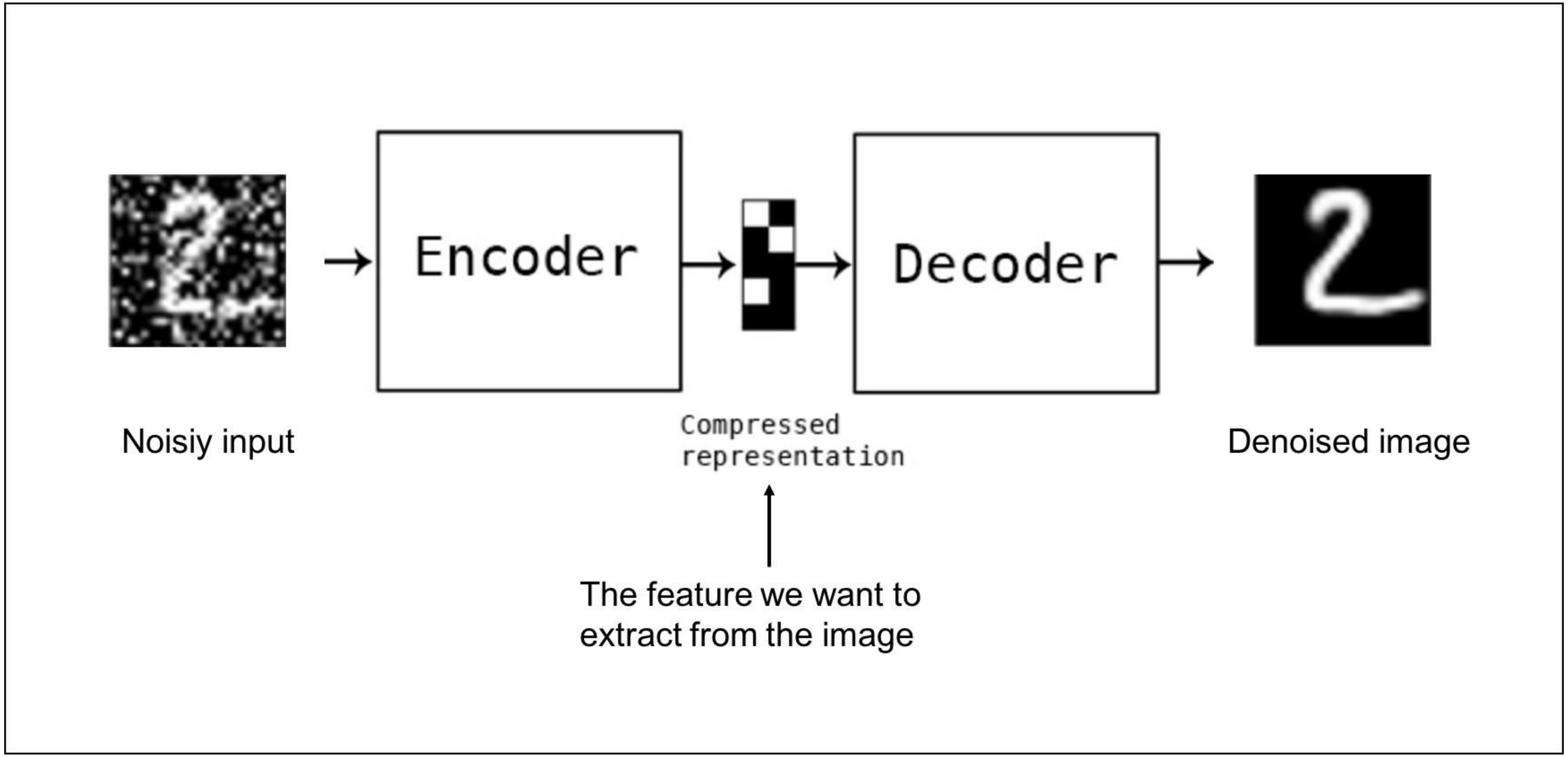
Train to reconstruct original data





An **autoencoder (AE)** is a type of ANN used to learn efficient data coding in an unsupervised manner. The goal of any AE is to reconstruct its own input. Usually, the AE first compresses the input into a smaller form, then transforms it back into an approximation of the input.

Denoising AE



- Reconstruction X' computed from the corrupted input X_noise
- Loss function compares X' reconstruction with the noiseless input X

AE: MNIST example

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
```

Load MNIST data

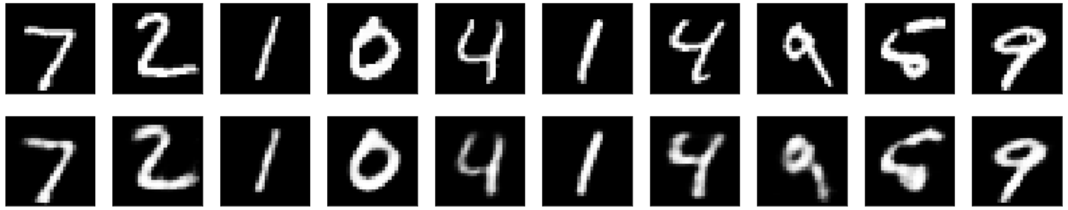
```
Input_dim = 784
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.reshape(60000, input_dim)
x_test = x_test.reshape(10000, input_dim)
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
```

```
def build_encoder():
        input_img = keras.Input(shape=(input_dim,))
        encoding_layer1 = layers.Dense(128, activation='relu')(input_img)
        encoding_layer2 = layers.Dense(64, activation='relu')(encoding_layer1)
        encoded = layers.Dense(latent_dim, activation='relu')(encoding_layer2)
        return keras.Model(input_img, encoded)

def build_decoder(): decoder_input = keras.Input(shape=(latent_dim,))
        decoding_layer1 = layers.Dense(64, activation='relu')(decoder_input)
        decoding_layer2 = layers.Dense(128, activation='relu')(decoding_layer1)
        decoded = layers.Dense(input_dim, activation='sigmoid')(decoding_layer2)
        return keras.Model(decoder_input, decoded)
```

```
latent_dim = 32
encoder = build_encoder()
dencoder = build_dencoder()
```

```
input_images = keras.Input(shape=(input_dim,))
encoded_repr = encoder(input_images)
reconstructed_output = decoder(encoded_repr)
autoencoder = keras.Model(input_images, reconstructed_output)

autoencoder.compile(optimizer = ‘adam’, loss = ‘binary_crossentropy’)

autoencoder.fit(x_train, x_train)
```

```
# Encode and decode some digits
 encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```



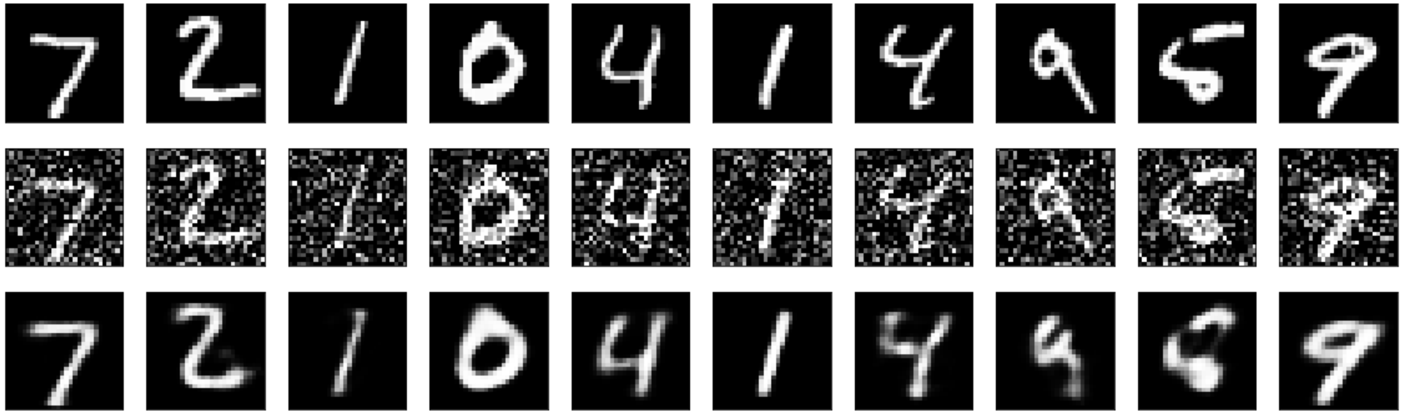
https://blog.keras.io/building-autoencoders-in-keras.html

The standard workflow: compile(), fit(), evaluate(), predict()

AE for image denoising (MNIST example)

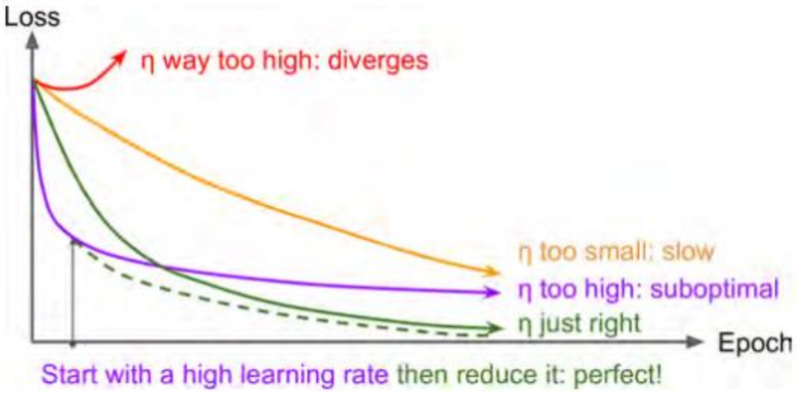
```
# Build the Model
input_layer = keras.Input(shape=(input_dim,))
encoding_layer1 = layers.Dense(128, activation='relu')(input_layer)
encoding_layer2 = layers.Dense(64, activation='relu')(encoding_layer1)
encoding_layer3 = layers.Dense(latent_dim, activation='relu')(encoding_layer2)
decoding_layer1 = layers.Dense(64, activation='relu')(encoding_layer2)
decoding_layer2 = layers.Dense(128, activation='relu')(decoding_layer1)
output_images = layers.Dense(input_dim, activation='sigmoid')(decoding_layer2)
autoencoder = keras.Model(input_layer, output_images)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

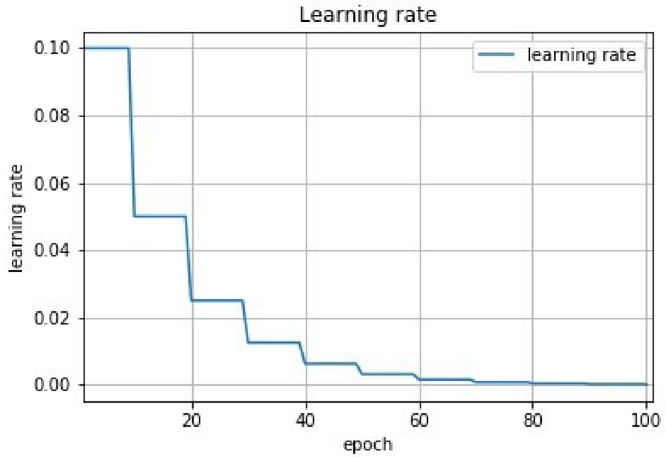
```
noise_factor = 0.4
x_train_noise  = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noise = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

x_train_noise = np.clip(x_train_noise, 0., 1.)
x_test_noise = np.clip(x_test_noise, 0., 1.)
```

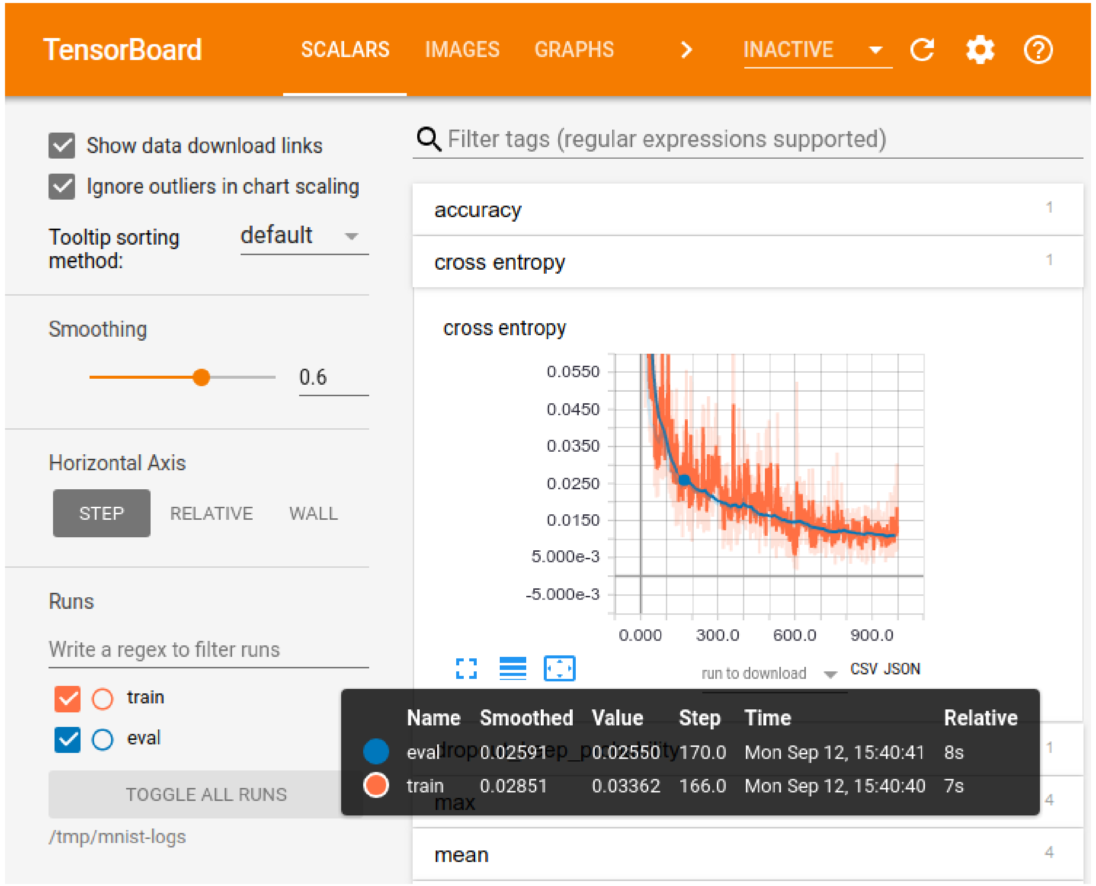


Keras: using callbacks

keras.callbacks.ModelCheckpoint
keras.callbacks.EarlyStopping
keras.callbacks.LearningRateScheduler
keras.callbacks.CSVLogger
keras.callbacks.TensorBoard







Start with high learning rate and then reduce it

- Predetermined piecewise constant learning rate
- Exponential scheduling

https://keras.io/api/callbacks/

TensorBoard: Playground

https://playground.tensorflow.org/