In [1]:
```
import sklearn
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [2]:
```
np.random.seed(20)
tf.random.set_seed(20)

%matplotlib inline

mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

In [3]:
```
(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:]
```

In [4]:
```
def rounded_accuracy(y_true, y_pred):
    return keras.metrics.binary_accuracy(tf.round(y_true), tf.round(y_pred))
```

In [5]:
```
# Construction d'un encodeur empilé "stacked" qui prend en entrée des images
# en niveaux de gris de 28*28 pixels
# On utilise la fonction Flatten pour applatir l'image
# Ensuite, nous avons deux couches denses de taille (100,30)
# Les deux couches utilisent la fonction d'activation selu
# Nous pouvons également tilisé la normalisation he

stacked_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu", kernel_initializer='he_normal'),
    keras.layers.Dense(30, activation="selu"),
])
```

In [6]:
```
# Pour la construction du décodeur
# À la dernière couche, nous remettons les vecteurs des données
# sous la forme de tableaux 28×28 afin que ses sorties soient de
# la même dimension que les entrées de l'encodeur.
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])
])
```

In [7]:
```
# Pour la compilation de l'autoencodeur, nous considérons
# la perte d'entropie croisée binaire.
# Nous considérons la reconstruction comme un
# problème de classification binaire multiétiquette
# L'entrainement du modèle est fait en utilisant
# X_train à la fois pour les entrées et pour les cibles (recontruction)
# De la même manière, nous utilisons X_valid pour les entrées et les
# cibles de validation (recontruction).
stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
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))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/20
55000/55000 [==============================] - 21s 380us/sample - loss: 0.3363 - rounded_accuracy: 0.8893 - val_loss: 0.3129 - val_rounded_accuracy: 0
.9063
Epoch 2/20
55000/55000 [==============================] - 19s 353us/sample - loss: 0.3048 - rounded_accuracy: 0.9161 - val_loss: 0.3015 - val_rounded_accuracy: 0
.9221
Epoch 3/20
55000/55000 [==============================] - 19s 352us/sample - loss: 0.2973 - rounded_accuracy: 0.9230 - val_loss: 0.2959 - val_rounded_accuracy: 0
.9268
Epoch 4/20
55000/55000 [==============================] - 19s 354us/sample - loss: 0.2932 - rounded_accuracy: 0.9267 - val_loss: 0.2924 - val_rounded_accuracy: 0
.9286
Epoch 5/20
55000/55000 [==============================] - 20s 355us/sample - loss: 0.2905 - rounded_accuracy: 0.9292 - val_loss: 0.2912 - val_rounded_accuracy: 0
.9282
Epoch 6/20
55000/55000 [==============================] - 20s 357us/sample - loss: 0.2884 - rounded_accuracy: 0.9311 - val_loss: 0.2891 - val_rounded_accuracy: 0
.9337
Epoch 7/20
55000/55000 [==============================] - 20s 360us/sample - loss: 0.2868 - rounded_accuracy: 0.9325 - val_loss: 0.2890 - val_rounded_accuracy: 0
.9295
Epoch 8/20
55000/55000 [==============================] - 20s 359us/sample - loss: 0.2857 - rounded_accuracy: 0.9335 - val_loss: 0.2879 - val_rounded_accuracy: 0
.9301
Epoch 9/20
55000/55000 [==============================] - 20s 359us/sample - loss: 0.2847 - rounded_accuracy: 0.9344 - val_loss: 0.2854 - val_rounded_accuracy: 0
.9350
Epoch 10/20
55000/55000 [==============================] - 20s 359us/sample - loss: 0.2839 - rounded_accuracy: 0.9350 - val_loss: 0.2854 - val_rounded_accuracy: 0
.9330
Epoch 11/20
55000/55000 [==============================] - 20s 362us/sample - loss: 0.2833 - rounded_accuracy: 0.9355 - val_loss: 0.2867 - val_rounded_accuracy: 0
.9356
Epoch 12/20
55000/55000 [==============================] - 20s 358us/sample - loss: 0.2826 - rounded_accuracy: 0.9361 - val_loss: 0.2839 - val_rounded_accuracy: 0
.9367
Epoch 13/20
55000/55000 [==============================] - 20s 357us/sample - loss: 0.2821 - rounded_accuracy: 0.9365 - val_loss: 0.2836 - val_rounded_accuracy: 0
.9363
Epoch 14/20
55000/55000 [==============================] - 20s 363us/sample - loss: 0.2818 - rounded_accuracy: 0.9367 - val_loss: 0.2833 - val_rounded_accuracy: 0
.9378
Epoch 15/20
55000/55000 [==============================] - 20s 362us/sample - loss: 0.2813 - rounded_accuracy: 0.9371 - val_loss: 0.2823 - val_rounded_accuracy: 0
.9385
Epoch 16/20
55000/55000 [==============================] - 20s 362us/sample - loss: 0.2810 - rounded_accuracy: 0.9373 - val_loss: 0.2826 - val_rounded_accuracy: 0
.9383
Epoch 17/20
55000/55000 [==============================] - 21s 382us/sample - loss: 0.2807 - rounded_accuracy: 0.9376 - val_loss: 0.2827 - val_rounded_accuracy: 0
.9386
Epoch 18/20
55000/55000 [==============================] - 20s 368us/sample - loss: 0.2804 - rounded_accuracy: 0.9378 - val_loss: 0.2992 - val_rounded_accuracy: 0
.9057
Epoch 19/20
55000/55000 [==============================] - 20s 361us/sample - loss: 0.2801 - rounded_accuracy: 0.9380 - val_loss: 0.2813 - val_rounded_accuracy: 0
.9383
Epoch 20/20
55000/55000 [==============================] - 20s 372us/sample - loss: 0.2798 - rounded_accuracy: 0.9382 - val_loss: 0.2812 - val_rounded_accuracy: 0
.9385
```

This function processes a few test images through the autoencoder and displays the original images and their reconstructions:

In [8]:
```
# Fonctions pour afficher des images à niveaux de gris
def plot_image(image):
    plt.imshow(image, cmap="binary")
    plt.axis("off")
```

In [9]:
```
# Affichage de quelques images de validation et leur reconstruction
def show_reconstructions(model, images=X_valid, n_images=5):
    reconstructions = model.predict(images[: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(images[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])
```

In [10]:
```
# Affichage des images originales et reconstruites
show_reconstructions(stacked_ae)
```