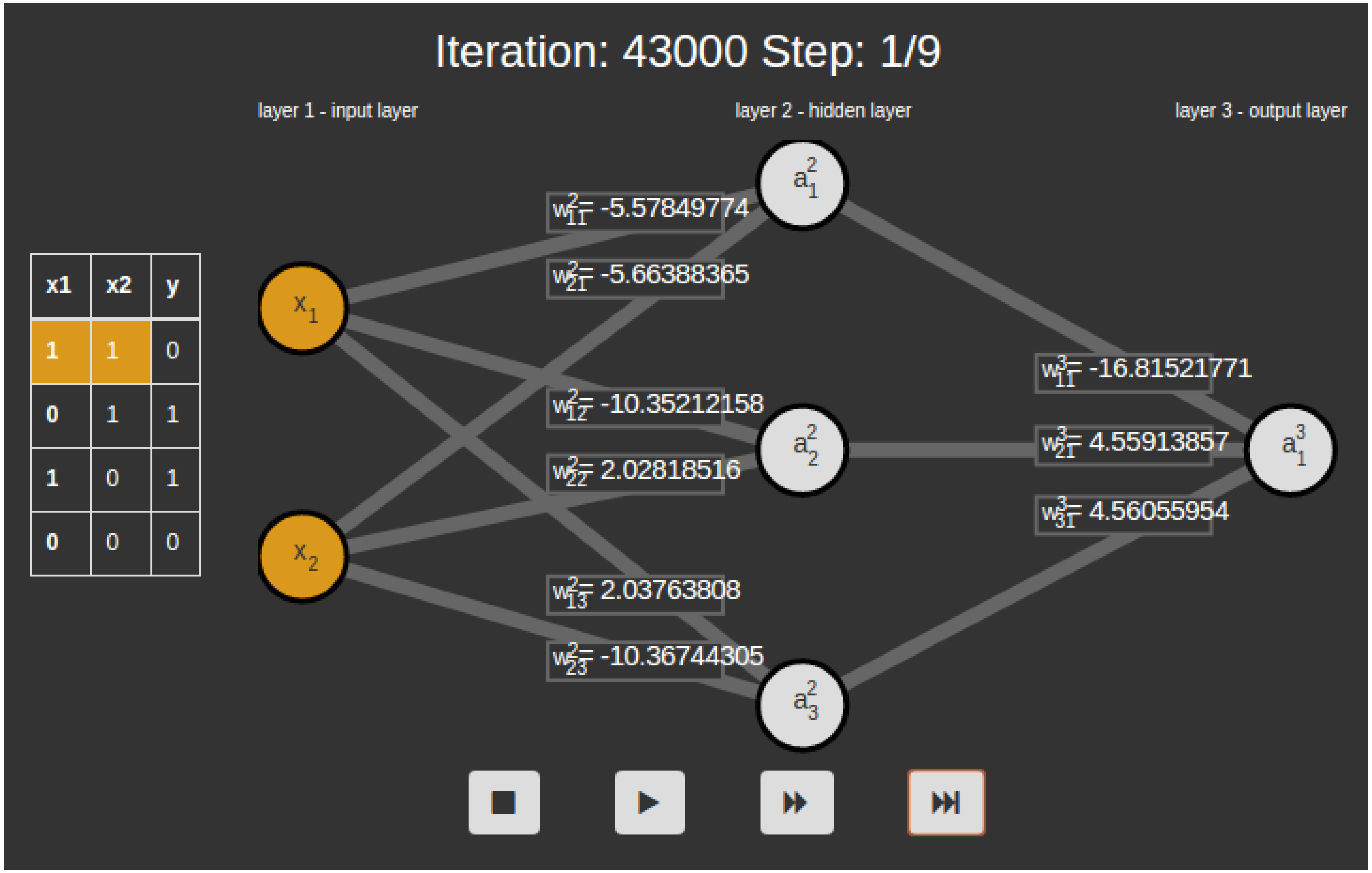
Autoencoders

Seoul AI Meetup, July 8

Martin Kersner, m.kersner@gmail.com

Visualized training of Multilayer Perceptron (https://www.mladdict.com/).



Structure of this presentation is largely based on chapter *15: Autoencoders* from book Hands-On Machine Learning with Scikit-Learn & Tensorflow.

Some examples are modified version of https://github.com/ageron/handson-ml.

Content

Efficient Data Representation

- Number sequences
  - 56, 46, 8, 56, 7, 6, 8, 52,...
  - 5, 16, 8, 4, 2, 1, 4, 2, 1,...

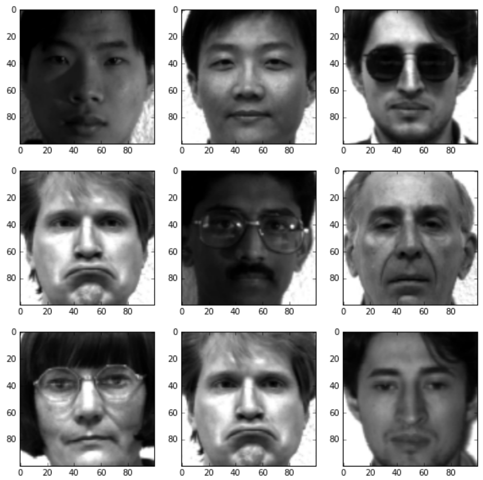
- Lower Data Dimensionality
  - Reduced computational cost
  - Easier to train (Curse of dimensionality)
  - Easier to visualize
    - *N*D -> 3D
    - *N*D -> 2D
- Information retrieval tasks
  - Semantic hashing
- Other methods
  - Factor Analysis
  - Independent Component Analysis
  - t-SNE

Principal Component Analysis

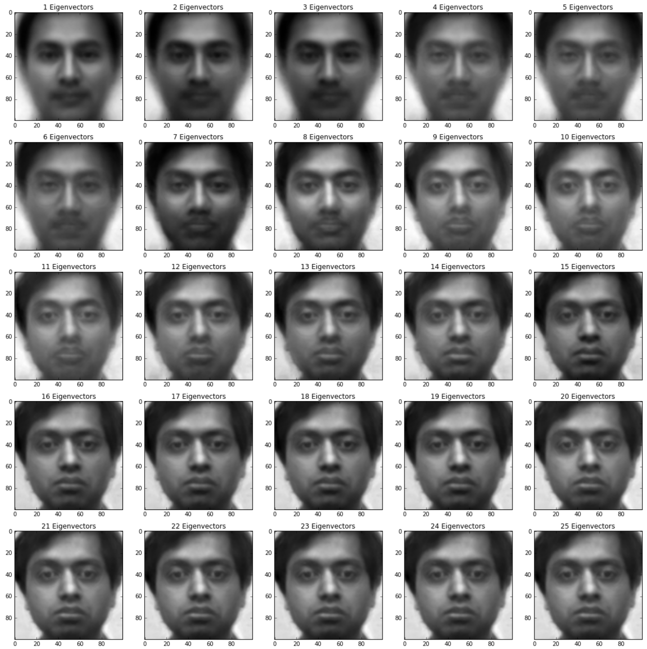
- For unlabeled data
- Transformation from original coordinate system to the new one
- Orthogonal linear transformation
- Used for dimensionality reduction
- Principal components represent directions along which the data has the largest variations
- sklearn.decomposition.PCA

Yale Face Database

- 15 people
- 11 images per subject one per different facial expression or configuration
- (center-light w/glasses, happy, left-light, w/no glasses, normal, right-light, sad, sleepy, surprised, and wink)



Eigenfaces (Jupyter Notebook)



In [2]:

```python
# scikit-learn: Principal Component Analysis
import numpy as np
from sklearn.decomposition import PCA

X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
pca = PCA(n_components=2)
pca.fit(X)

print(pca.explained_variance_ratio_)
```

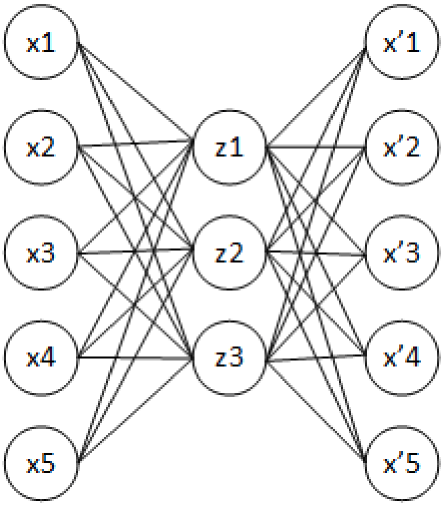
```
[ 0.99244289  0.00755711]
```

Autoencoders

- Artificial Neural Networks
- Same architecture as Multi-Layer Perceptron
- Number of input neurons = Number of output neurons
- Trained to efficiently encode (**codings**) input information

- Purposes
  - Decrease dimensionality
  - Feature detectors (unsupervised pretraining for deep neural networks)
  - Randomly generate new data

- **Encoder** (Recognition network)
- **Decoder** (Generative network)

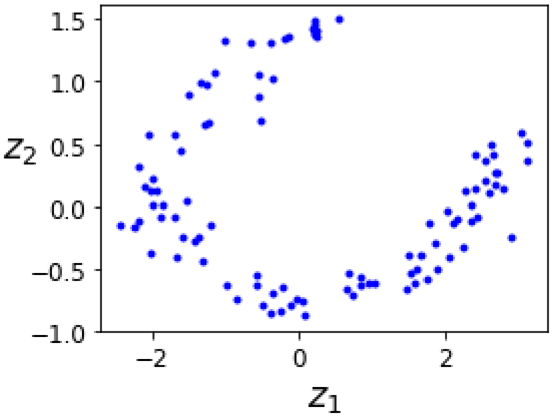


Example of **undercomplete** autoencoder.

Autoencoder as PCA

- Linear activations
- Cost function Mean Squared Error

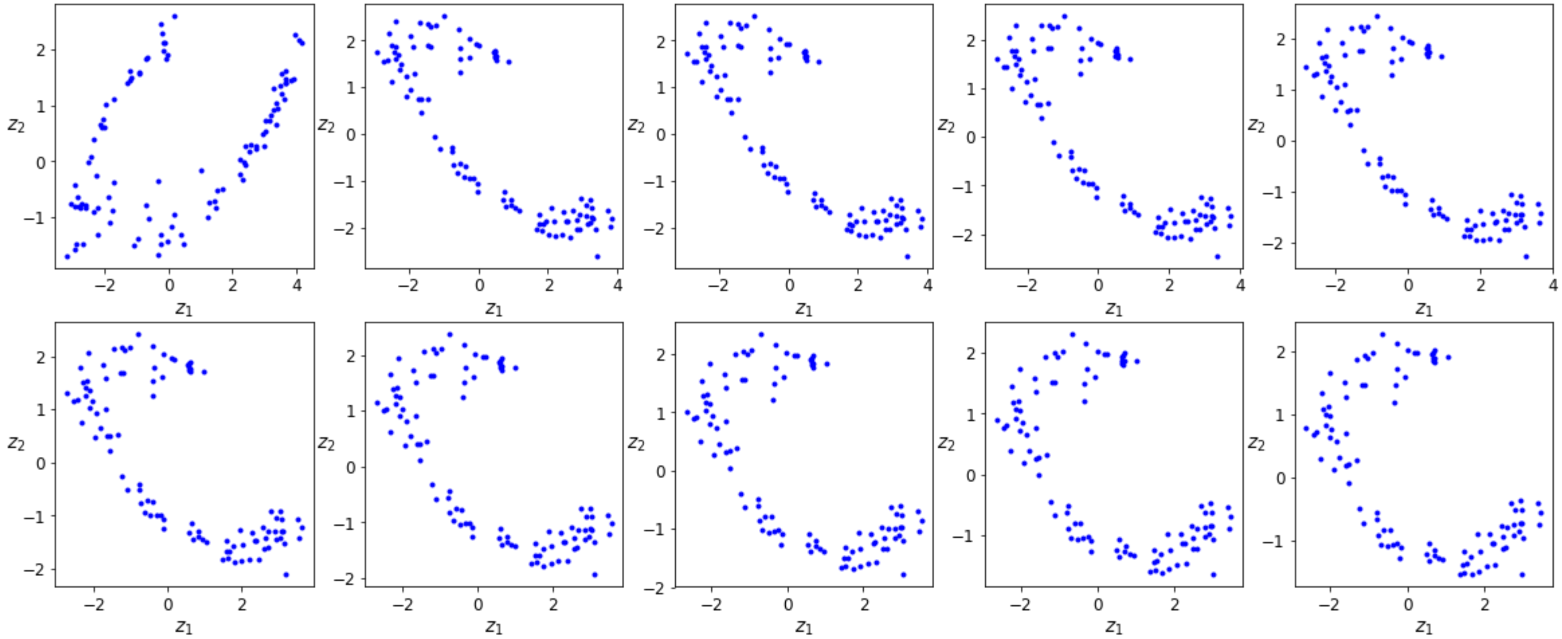
In [4]:
```
# PCA
pca = PCA(n_components=2)
pca.fit(X_train)
pca_codings = pca.transform(X_test)

# Encodings created using PCA
plot_coding(pca_codings)
```



Autoencoder Temporary Results

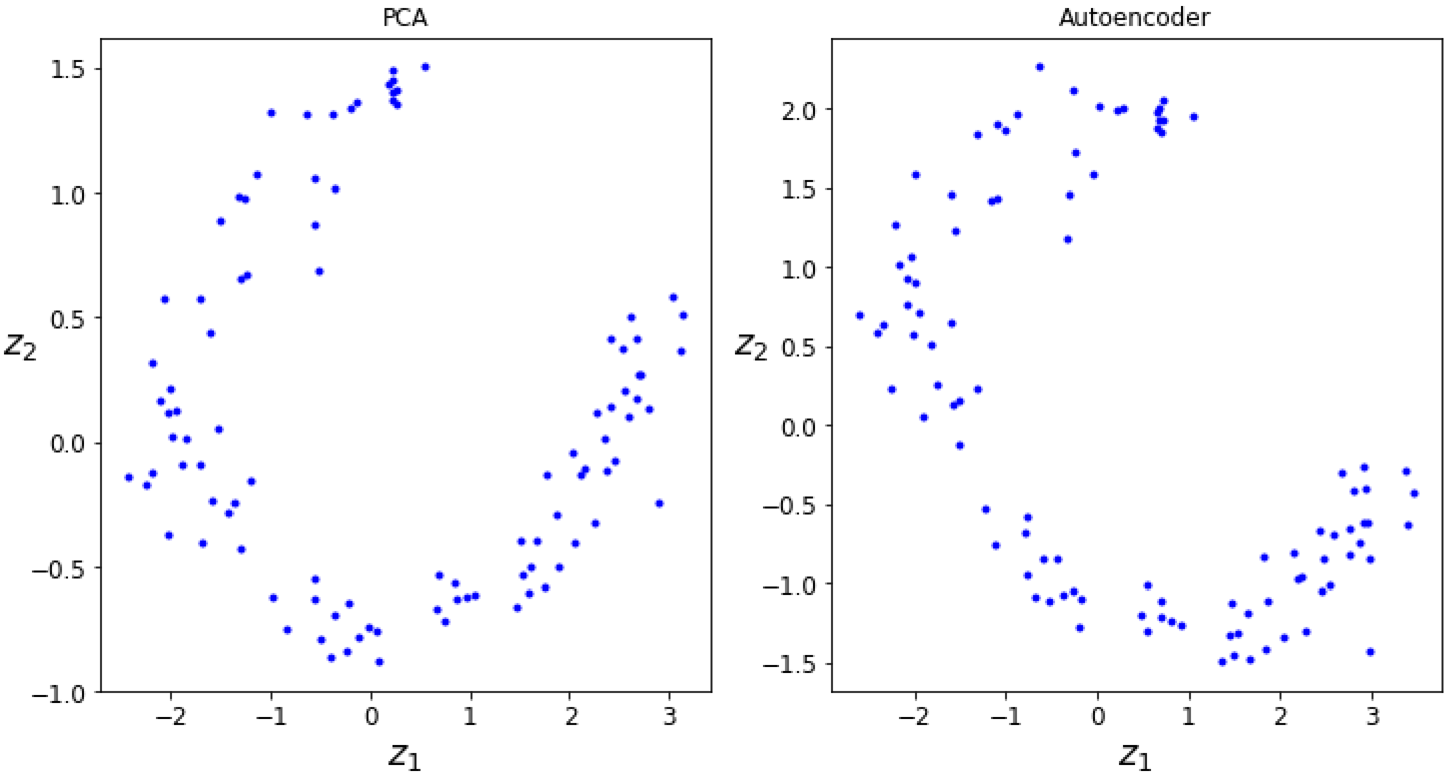
In [6]:

```
plot_many_codings(codings_val_progress)
```



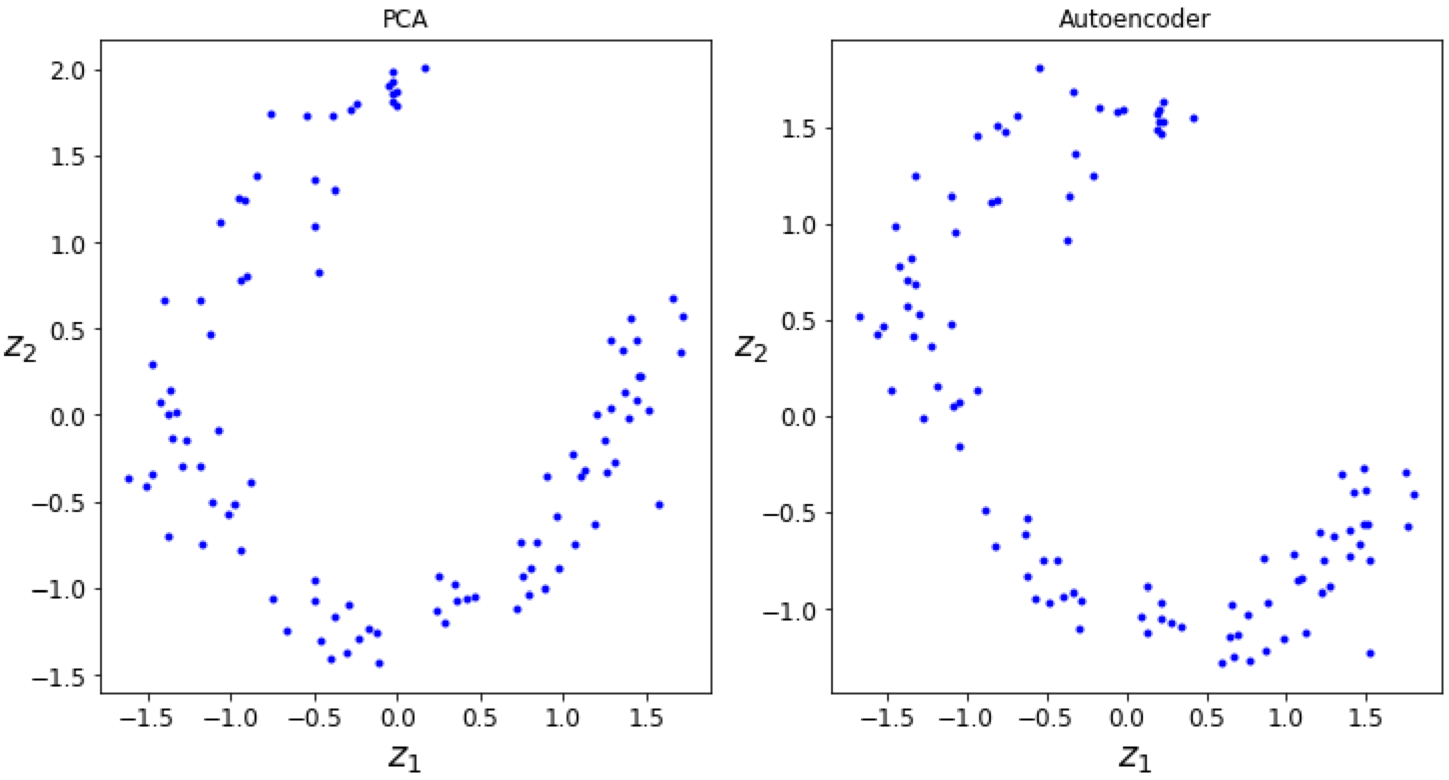
PCA vs Autoencoder

In [7]:
```
plot_codings(pca_codings, codings_val)
```



sklearn.preprocessing.StandardScaler

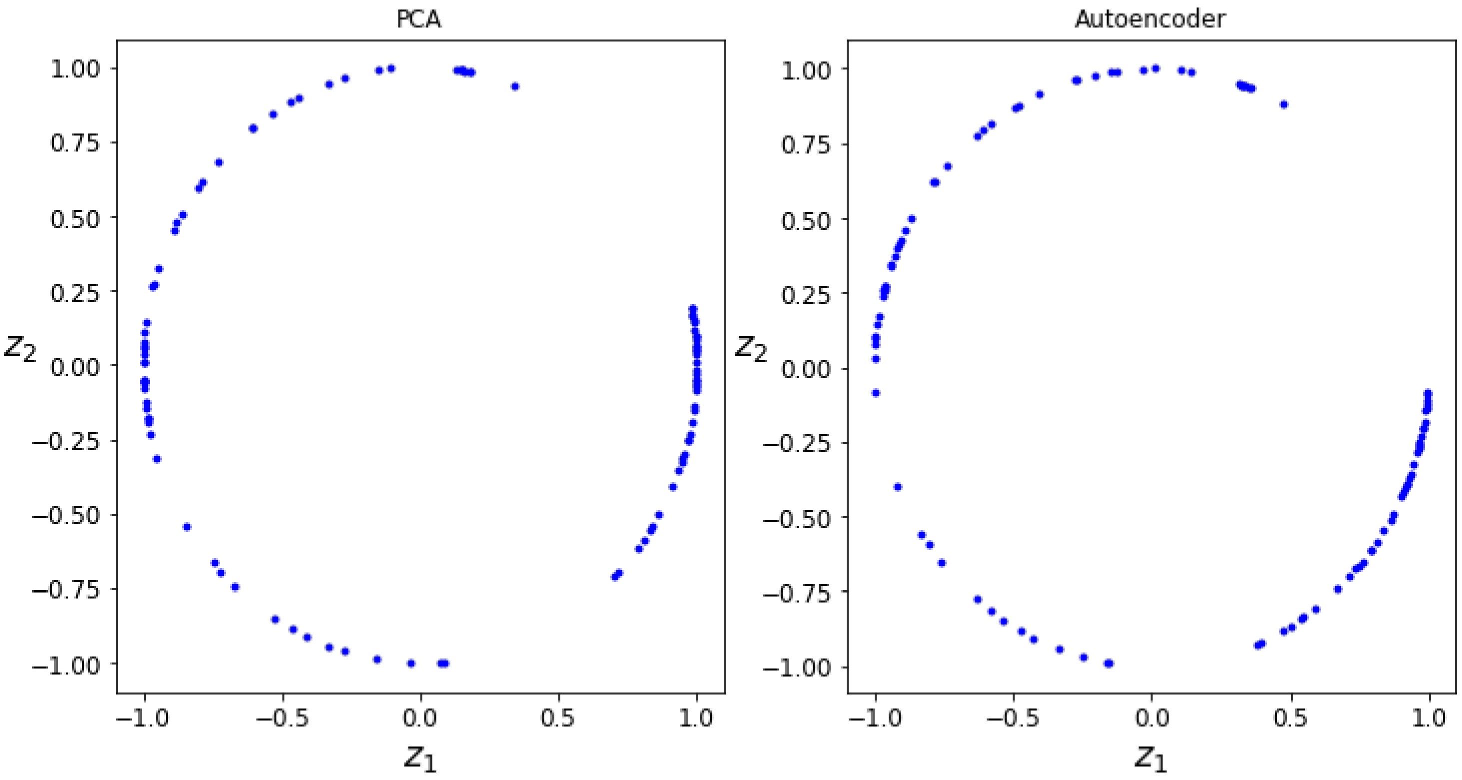
In [8]:
```
scaler1 = StandardScaler()
pca_norm = scaler.fit_transform(pca_codings)
scaler2 = StandardScaler()
autoencoder_norm = scaler.fit_transform(codings_val)
plot_codings(pca_norm, autoencoder_norm)
```



sklearn.preprocessing.normalize

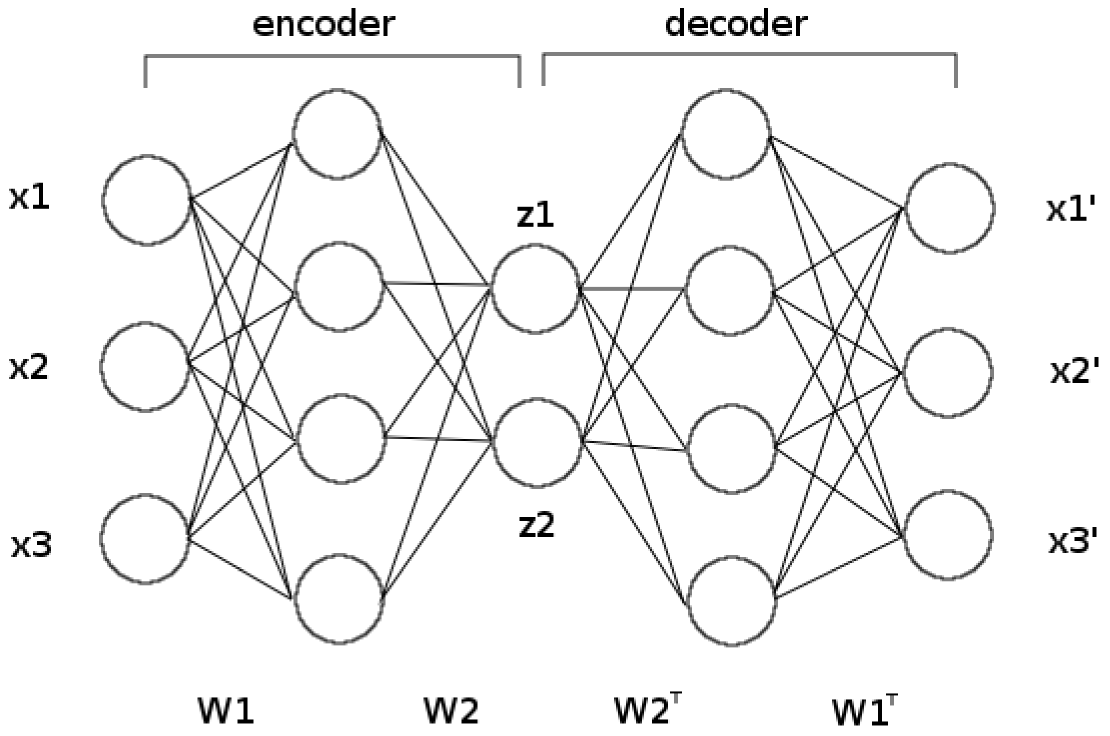
In [9]:

```
from sklearn.preprocessing import normalize

pca_norm = normalize(pca_codings)
autoencoder_norm = normalize(codings_val)
plot_codings(pca_norm, autoencoder_norm)
```



Stacked Autoencoders

- Multiple hidden layers => Stacked Autoencoders, Deep Autoencoders
- Learn more complex codings
- Typically symmetrical architecture



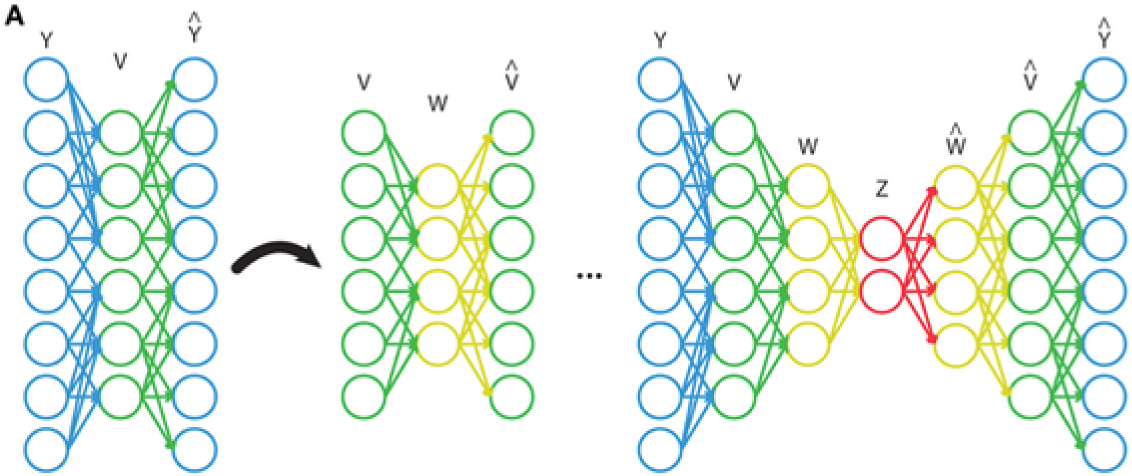
Tying Weights

- When layers of **Encoder** are symmetrical to **Decoder**, weights can be shared => Tying Weights
- Half of the weights
  - Speed up training
  - Limiting risk of overfitting

$$W_{N-L+1} = W_L^T \text{ for } L = 1, 2, 3, \ldots, N/2$$

Training One Autoencoder at a Time

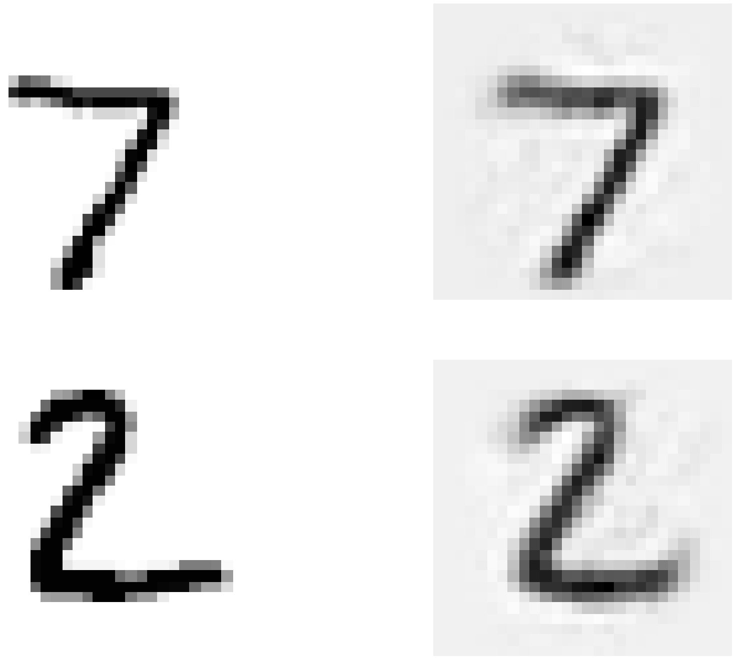
- Training of shallow autoencoder is faster than training stacked autoencoders at once.
- Training is performed in phases.



Visualizing the Reconstructions

In [18]:
```
show_reconstructed_digits(X, outputs, model_path="my_model_one_at_a_time.ckpt")
```

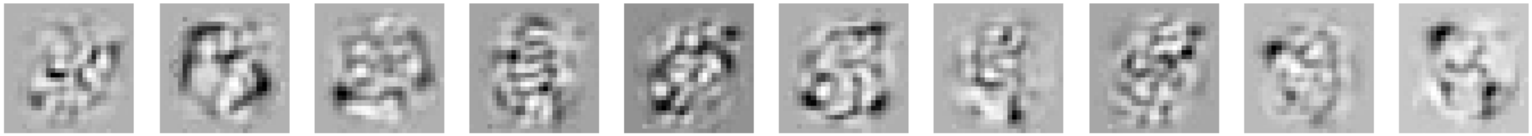
```
INFO:tensorflow:Restoring parameters from my_model_one_at_a_time.ckpt
```



Techniques of Visualizing Features

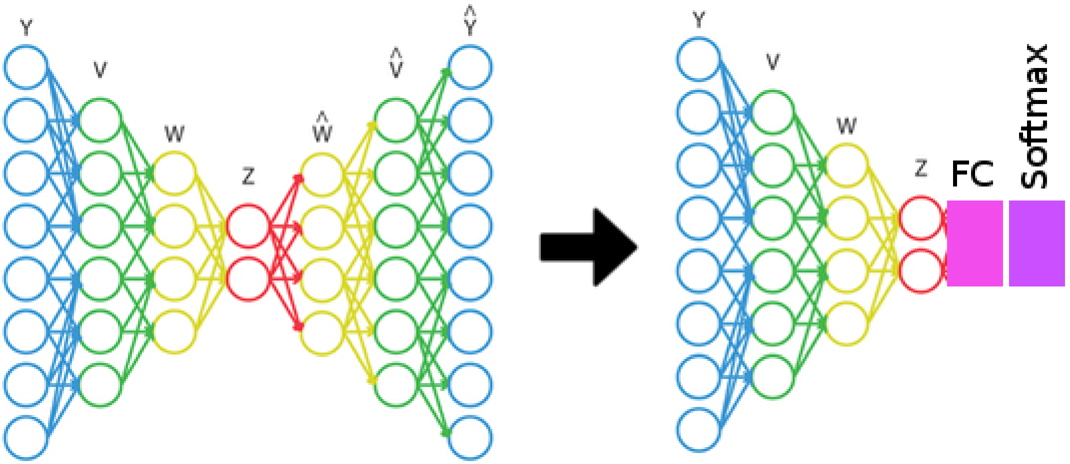
1. Examine each neuron in every layer independently
2. Display weights of each neuron in the first layer

In [21]:
```
plot_features_from_first_hidden_layer(weights1_val)
```



Unsupervised Pretraining Using Stacked Autoencoders

1. Train autoencoder
2. Remove Decoder
3. If not enough training data, freeze **Encoder** weights
4. Train classifier on top of network



Overcomplete Autoencoders

Autoencoders with the same size (or even larger) of codings as the input layer.

1. Denoising Autoencoders
2. Sparse Autoencoders

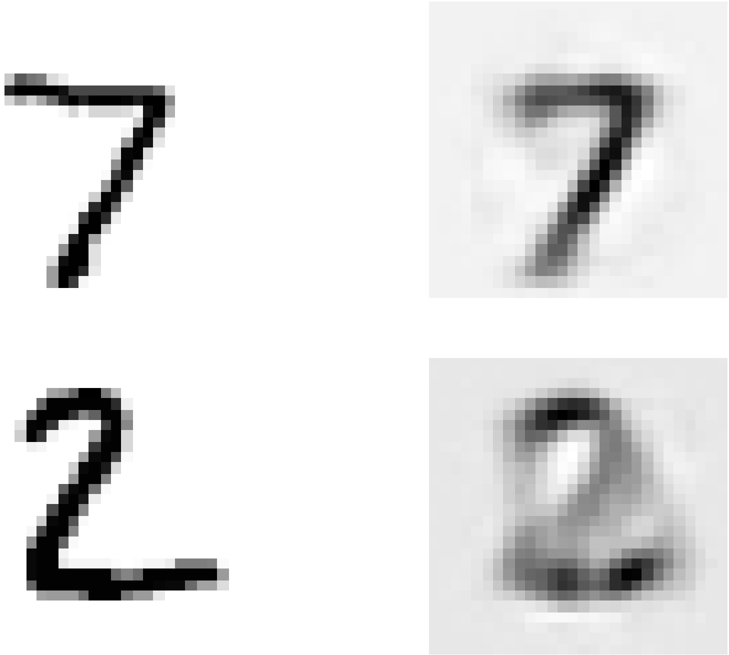
Denoising Autoencoders

- Added noise to its inputs.
    - Gaussian nose
    - Dropout layer
- Train to recover the original, noise-free inputs.
- Find patterns in data.

Reconstructions From Denoising Autoencoder

In [27]:
```
show_denoising_autoencoder_results(X, outputs)
```

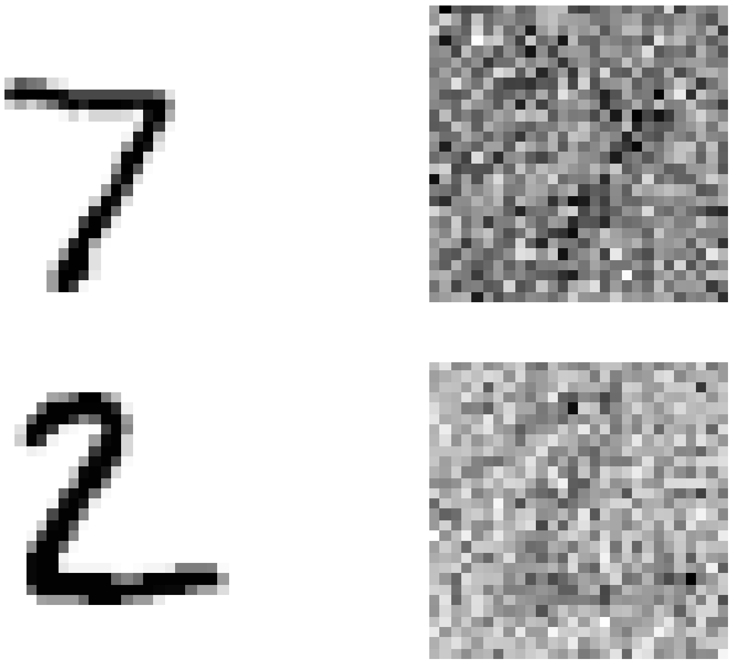
```
INFO:tensorflow:Restoring parameters from ./my_model_stacked_denoising_gaussia
n.ckpt
```



Noisy Inputs

In [29]:
```
plot_noisy_inputs(X, X_noisy)
```

```
INFO:tensorflow:Restoring parameters from ./my_model_stacked_denoising_gaussia
n.ckpt
```



Sparse Autoencoders

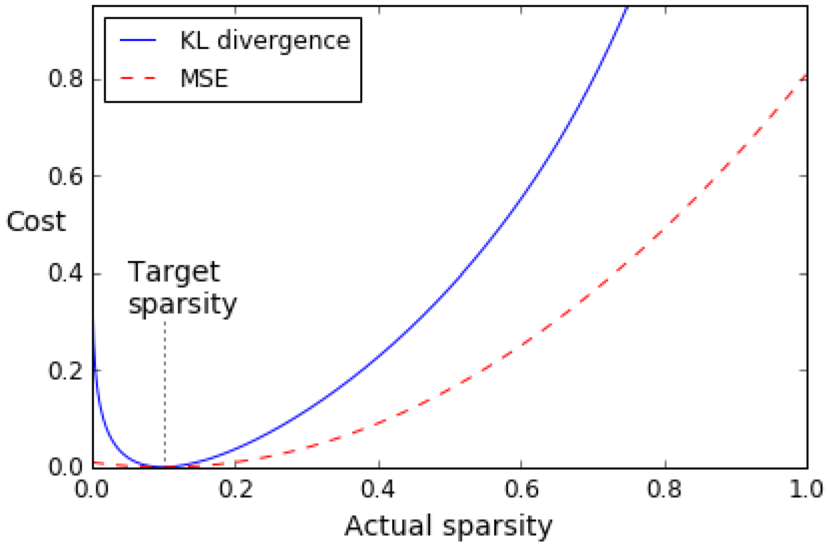
- Loss function involves **sparsity penalty** on the coding layer.
- Used to learn features for another task such as classification.
- Represent each input as a combination of **small** number of activations.

$$L(\boldsymbol{x}, g(f(\widetilde{\boldsymbol{x}}))) + \lambda\Omega(\boldsymbol{h}), \text{ where } \boldsymbol{h} = f(\widetilde{\boldsymbol{x}})$$

Sparsity of Coding Layer

1. Decide target sparsity (e.g. 0.1).
2. Compute average activation of each neuron in the coding layer, over the whole training batch.
   - *sigmoid* activation
   - `tf.reduce_mean(hidden1, axis=0)`
3. Compute Kullback-Leibler divergence (stronger gradients than for example MSE).

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$



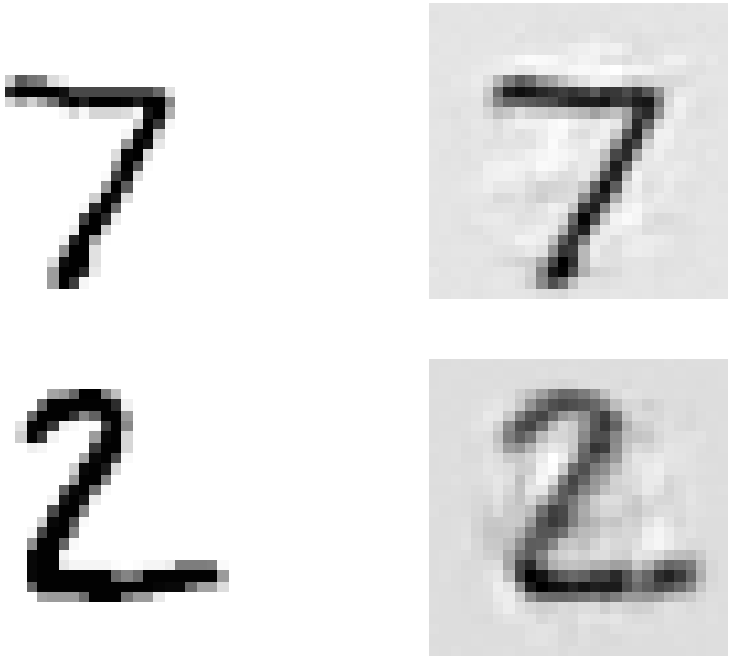
Reconstructions From Sparse Autoencoder

- 3 layers
- 1,000 coding neurons

```
sparsity_target = 0.1
sparsity_weight = 0.2
n_epochs = 100
batch_size = 1000
```

In [35]:
```
show_sparse_autoencoder_results(X, outputs)
```

```
INFO:tensorflow:Restoring parameters from ./my_model_sparse.ckpt
```



Variational Autoencoders

- Probabilistic
  - Outputs are partly determined by chance.
- Generative
  - Generate new instances that are similar to the one in the training dataset.

Similar to Restricted Boltzmann Machines.

Coding Generation

1. Encoder produces mean μ and standard deviation σ of coding.
2. Coding is then sampled from Gaussian distribution.
3. `Loss = Reconstruction loss + Latent loss`

```
hidden3_mean = my_dense_layer(hidden2, n_hidden3, activation=None)
hidden3_gamma = my_dense_layer(hidden2, n_hidden3, activation=None)
noise = tf.random_normal(tf.shape(hidden3_gamma), dtype=tf.float32)
#hidden3 = hidden3_mean + hidden3_gamma * noise
hidden3 = hidden3_mean + tf.exp(0.5 * hidden3_gamma) * noise
```

Generated Digits

In [39]:
```
generate_digits()
```

Other Autoencoders

- Contractive Autoencoders
  - two similar inputs have similar condings
- Stacked Convolutional Autoencoders
- Generative Stochastic Network
  - denoising autoencoders with added capability to generate data
- Winner-take-all Autoencoder
  - only top k activations is preserved, leads to sparse coding
- Adversial Autoencoders
  - two networks
  - one is trained to reproduce its inputs
  - the other one find inputs that the first network cannot reconstruct properly