CS5489 - Machine Learning

Lecture 11a - Auto-Encoders and Deep Generative Models

Prof. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

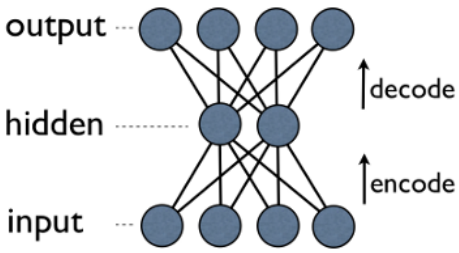
- Unsupervised models
  - Autoencoders
  - Denoising Autoencoders
- Deep generative models
  - Variational Autoencoders
  - GANs
  - Diffusion models

Neural Networks and Unsupervised Learning

- How to use NN for dimensionality reduction or clustering?

Autoencoder

- learn a latent representation of the data.
- Use the hidden layer as the lower-dimensional representation ("code", "bottleneck" layer)
- Train the network to "encode" and "decode"
  - run it through the encoding-decoding network
  - minimize the difference between the output and the original input



Autoencoder

- encode and decode with NN:
  - encoder: $\mathbf{z} = f(\mathbf{x})$
  - decoder: $\hat{\mathbf{x}} = g(\mathbf{z})$
- reconstruction: $\hat{\mathbf{x}} = g(f(\mathbf{x}))$
- Goal: minimize the reconstruction loss.
  - MSE (real inputs): $||\mathbf{x} - \hat{\mathbf{x}}||^2$
  - binary cross-entropy (inputs bounded in [0,1]): $\mathbf{x} \log \hat{\mathbf{x}} + (1 - \mathbf{x}) \log(1 - \hat{\mathbf{x}})$

Weight Sharing

- to reduce the number of parameters, share weights between the two layers
- D's weights are the transpose of E's weights
  - encoder: $\mathbf{z} = f(\mathbf{x}) = h_1(\mathbf{A}^T\mathbf{x})$
  - decoder: $\hat{\mathbf{x}} = g(\mathbf{z}) = h_2(\mathbf{A}\mathbf{z})$
- if h_1 and h_2 are linear activations, then it's similar to PCA.
  - except the orthogonal constraint on $\mathbf{A}$
- use more layers/non-linearities gives more general latent representations.

Example on MNIST

- Reshape the images into vectors, and scale to [0,1]

In [11]:
```
# Reshape the images and map the data to [0,1]
trainXraw = trainimg.reshape((len(trainimg), -1), order='C') / 255.0
testXraw = testimg.reshape((len(testimg), -1), order='C') / 255.0
```

In [12]:
```
# generate a fixed validation set using 10% of the training set
vtrainXraw, validXraw = \
  model_selection.train_test_split(trainXraw,
  train_size=0.9, test_size=0.1, random_state=4487)
```

- Define the Dense layer using transposed weights from another layer.
  - has its own bias term

In [13]:
```
class DenseTranspose(keras.layers.Layer):
    def __init__(self, dense, activation=None, **kwargs):
        self.dense = dense
        self.activation = keras.activations.get(activation)
        super().__init__(**kwargs)
    def build(self, batch_input_shape):
        self.biases = self.add_weight(name="bias", initializer="zeros",
                                      shape=[self.dense.input_shape[-1]])
        super().build(batch_input_shape)
    def call(self, inputs):
        z = tf.matmul(inputs, self.dense.weights[0], transpose_b=True)
        return self.activation(z + self.biases)
    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.dense.input_shape[-1])
```

- Use `Model` class to define encoder, decoder, and the combination

In [14]:
```
K.clear_session()
# initialize random seed
random.seed(4487); tf.random.set_seed(4487)

# Build the Encoder model
input_img = Input(shape=(784,))                                    # input layer
encoder_dense_1 = Dense(10, activation='relu', name="encoder_1") # create dense layer
encoded = encoder_dense_1(input_img)                               # connect input
encoder = Model(input_img, encoded, name="encoder")                # create model

# Build the Decoder model
encoded_input = Input(shape=(10,))
decoder_dense_1 = DenseTranspose(encoder_dense_1, activation='sigmoid', name="decoder_1")
decoded = decoder_dense_1(encoded_input)
decoder = Model(encoded_input, decoded, name="decoder")

# build the full autoencoder model
autoencoder = Model(input_img, decoder(encoder(input_img)))
```

```
Metal device set to: Apple M1 Max
```

```
2023-01-24 21:22:47.025307: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-24 21:22:47.025439: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
```

- Encoder and decoder subnetworks

In [15]:
```
encoder.summary()
```

```
Model: "encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 encoder_1 (Dense)           (None, 10)                7850

=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
decoder.summary()
```

```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 10)]              0

 decoder_1 (DenseTranspose)  (None, 784)               8634

=================================================================
Total params: 8,634
Trainable params: 8,634
Non-trainable params: 0
_________________________________________________________________
```

- Full auto-encoder network
  - Composed of the encoder and decoder models

In [17]:
```
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 encoder (Functional)        (None, 10)                7850

 decoder (Functional)        (None, 784)               8634

=================================================================
Total params: 8,634
Trainable params: 8,634
Non-trainable params: 0
_________________________________________________________________
```

- compile the model
  - outputs are in [0,1], so use cross-entropy

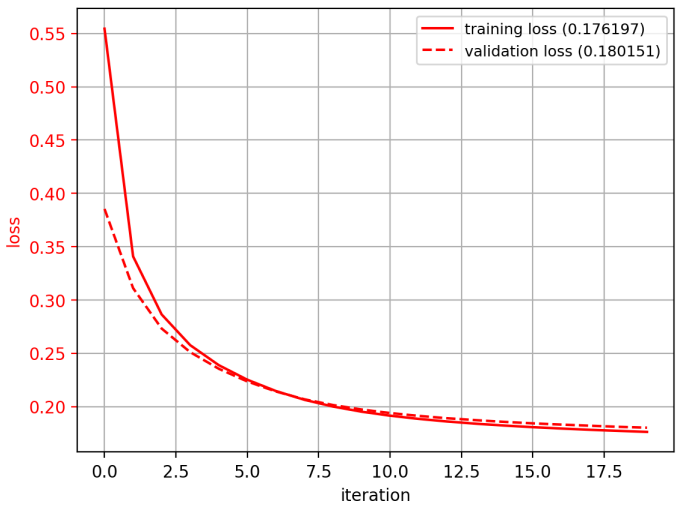
In [20]:
```
 # early stopping criteria
earlystop = keras.callbacks.EarlyStopping(monitor='val_loss',
                           min_delta=0.0001, patience=10,
                           verbose=1, mode='auto')

# compile and fit the network
autoencoder.compile(loss=keras.losses.binary_crossentropy,
           optimizer=keras.optimizers.SGD(learning_rate=0.2, momentum=0.9, nesterov=True))
```

- fit the model
  - the target is the same as the input

In [21]:
```
# fit the model: the input and output are the same
# write to a log directory to see training process
history = autoencoder.fit(vtrainXraw, vtrainXraw,
                 epochs=20, batch_size=50,
                 callbacks=[earlystop],
                 validation_data=(validXraw, validXraw), verbose=False)
```

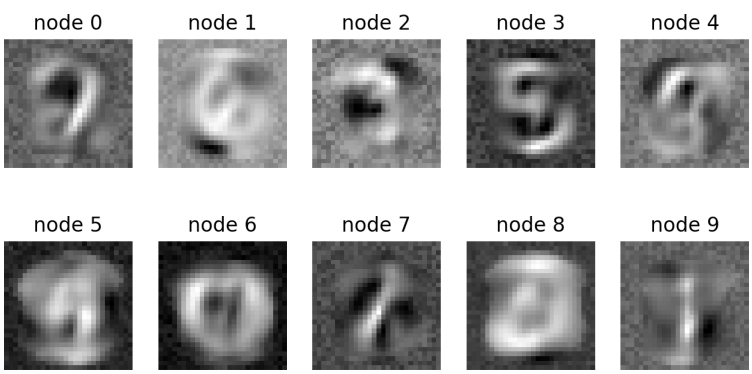
```
2023-01-24 21:23:07.446860: W tensorflow/core/platform/profile_utils/cpu_utils.cc:
128] Failed to get CPU frequency: 0 Hz
2023-01-24 21:23:07.533990: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-24 21:23:08.340640: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

In [22]:
```
plot_history(history)
```



- Visualize the weights of the hidden layer in E
  - each hidden node activates on a particular structure

In [23]:
```
W = encoder.get_layer(index=1).get_weights()[0]
filter_list = [W[:,i].reshape((28,28)) for i in range(W.shape[1])]
plt.figure(figsize=(8,4))
show_imgs(filter_list, nc=5, titles="node %d")
```



- Encode images into low-dim representation.

In [24]:
```
Z = encoder.predict(trainXraw)
Z.shape
```

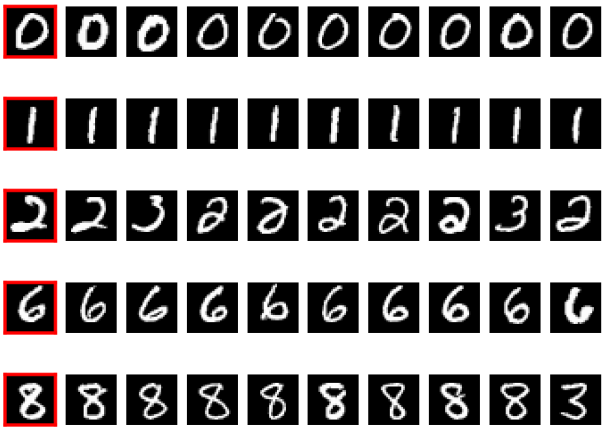
```
2023-01-24 21:23:20.977829: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

Out[24]:
```
(6000, 10)
```

- for the given highlighted red digit
  - show the 9 nearest neighbors in the latent space

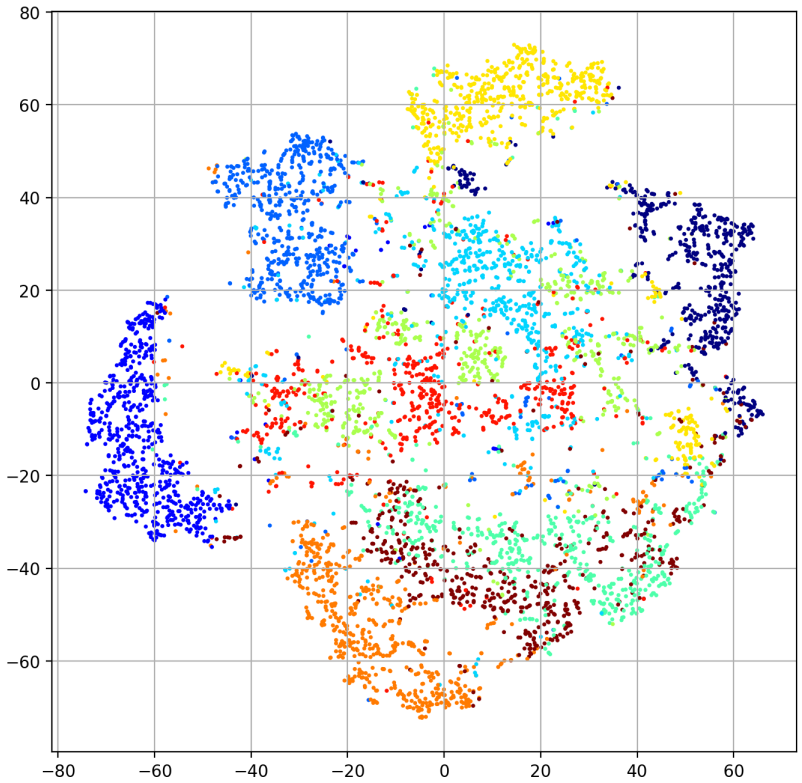
In [26]:
```
Zfig
```

Out[26]:



- Visualize a t-SNE embedding of the latent-space

In [30]:
```
plot_embedding(Z, trainY, perplexity=30.)
```



- Visualize the reconstruction of the input image

In [31]:
```
testXrecon = decoder.predict(encoder.predict(testXraw))
```

```
2023-01-24 21:24:08.003866: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

In [33]:
```
rfig
```

Out[33]:

Better Representations

- How to encourage better representations in the latent space?
- Two approaches:
  1. Regularize the latent space
     - e.g., using a sparisty-inducing regularizer of the latent vector
     - $L = ||\mathbf{x} - g(f(\mathbf{x}))||^2 + \lambda||f(\mathbf{x})||_1$
  2. Denoising auto-encoder (DAE)
     - add noise to make the encoder learn more about the data manifold.

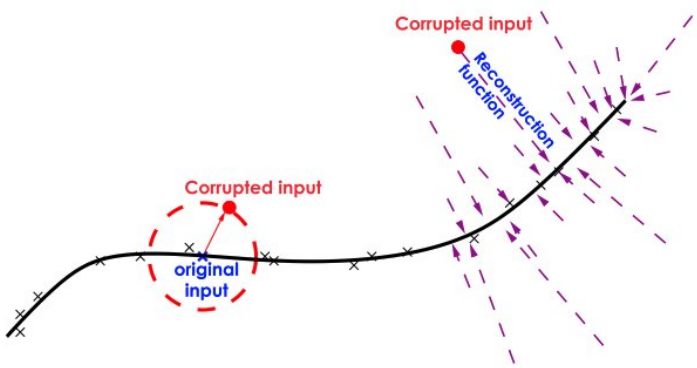
Denoising Autoencoder

- **Idea**: the AE's goal is to minimize $||\mathbf{x} - g(f(\mathbf{x}))||^2$.
  - the minimum occurs when $g(f(\mathbf{x})) = \mathbf{x}$, i.e., the identity function.
- **Problem**: The network can obtain this arbitrariliy if enough capacity.
  - but without learning anything about the data distribution.

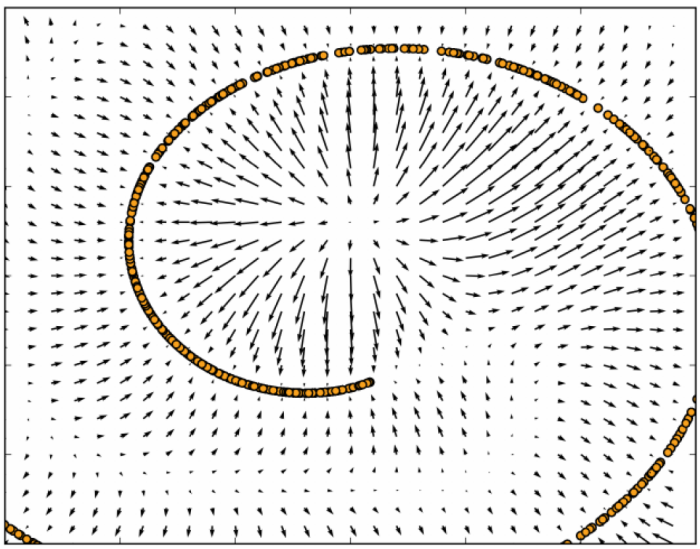
- **Solution**:
  - Force the AE to learn more about the data by corrupting the inputs.
    - minimize $||\mathbf{x} - g(f(\tilde{\mathbf{x}}))||^2$
    - $\tilde{\mathbf{x}}$ is a corrupted input
      - e.g., adding Gaussian noise, or zeroing some values
  - AE will learn how to "undo" the corruption, which requires learning the structure of the data manifold.
    - allows "code" size to be larger than the input size

DAE Interpretation (1)

- DAE is learning a vector field from the corrupted $\tilde{\mathbf{x}}$ to the clean $\mathbf{x}$ (the data manifold).

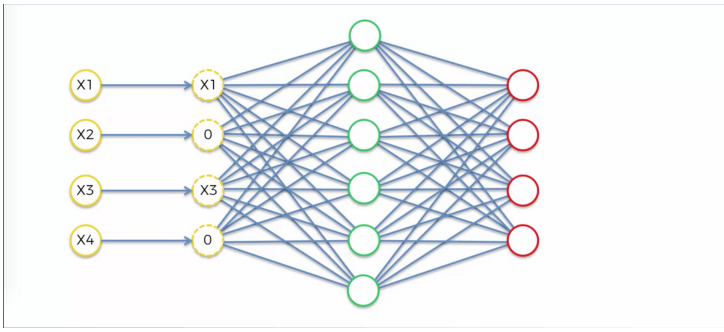


Example



Denoising Autoencoder

- randomly corrupt the input (by setting values to 0)
- implemented by applying Dropout on the inputs.



In [34]:
```
K.clear_session()
random.seed(4487); tf.random.set_seed(4487) # initialize random seed

# Build the Encoder model
input_img = Input(shape=(784,))
corrupted_img = Dropout(rate=0.3)(input_img)
encoded = Dense(10, activation='relu')(corrupted_img)
encoder = Model(input_img, encoded, name='DAE-encoder')

# Build the Decoder model (here we don't share weights)
encoded_input = Input(shape=(10,))
decoded = Dense(784, activation='sigmoid')(encoded_input)
decoder = Model(encoded_input, decoded, name='DAE-decoder')

# build the full autoencoder model
autoencoder = Model(input_img, decoder(encoder(input_img)))
```

- Encoder and decoder subnetworks

In [35]:
```
encoder.summary()
```

```
Model: "DAE-encoder"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

```
 input_1 (InputLayer)        [(None, 784)]             0

 dropout (Dropout)           (None, 784)               0

 dense (Dense)               (None, 10)                7850

=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
decoder.summary()
```

```
Model: "DAE-decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 10)]              0

 dense_1 (Dense)             (None, 784)               8624

=================================================================
Total params: 8,624
Trainable params: 8,624
Non-trainable params: 0
_________________________________________________________________
```

- Full auto-encoder network
  - Composed of the encoder and decoder models

In [37]:
```
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 DAE-encoder (Functional)    (None, 10)                7850

 DAE-decoder (Functional)    (None, 784)               8624

=================================================================
Total params: 16,474
Trainable params: 16,474
Non-trainable params: 0
_________________________________________________________________
```

- Compile the model

In [38]:
```
# compile and fit the network
autoencoder.compile(loss=keras.losses.binary_crossentropy,
           optimizer=keras.optimizers.SGD(lr=0.2, momentum=0.9, nesterov=True))
```

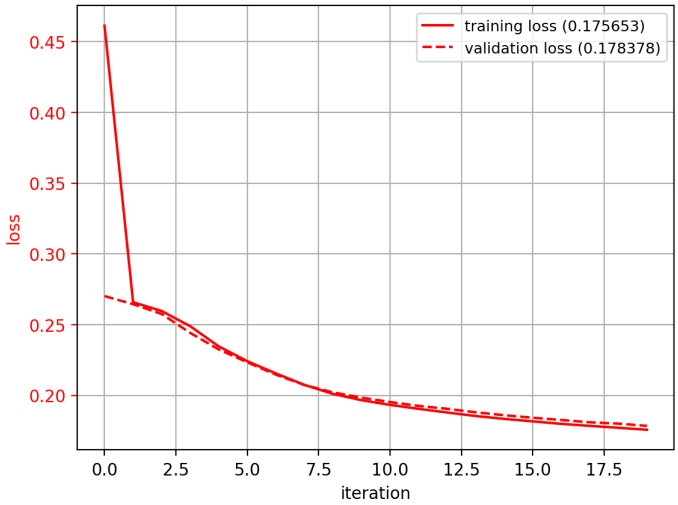
```
/Users/abc/miniforge3/envs/py38tfm28/lib/python3.8/site-packages/keras/optimizer_v
2/gradient_descent.py:102: UserWarning: The `lr` argument is deprecated, use `lear
ning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

- Fit the model

In [39]:
```
# fit the model: the input and output are the same
history = autoencoder.fit(vtrainXraw, vtrainXraw,
                 epochs=20, batch_size=50,
                 callbacks=[earlystop],
                 validation_data=(validXraw, validXraw), verbose=False)
```

```
2023-01-24 21:24:22.900510: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-24 21:24:23.529734: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

In [40]:
```
plot_history(history)
```



- Encode images into low-dim representation.
  - Note: at test time, Dropout layers do not add noise.

In [41]:
```
Z = encoder.predict(trainXraw)
Z.shape
```

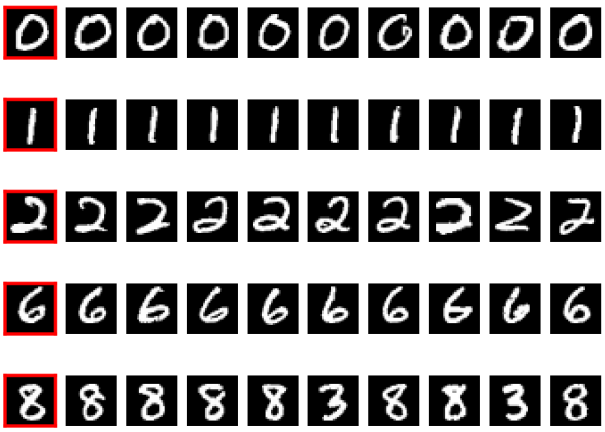
```
2023-01-24 21:24:39.460109: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

Out[41]:
```
(6000, 10)
```

- Visualize the nearby neighbors in the low-dim representation.
  - each row represents one set of neighbors

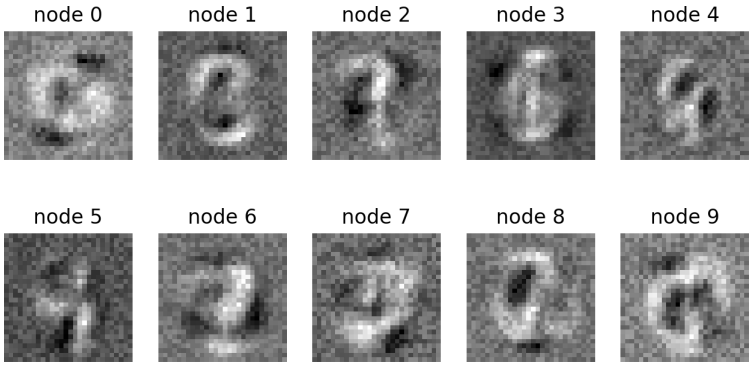
In [43]:
```
Zfig
```

Out[43]:



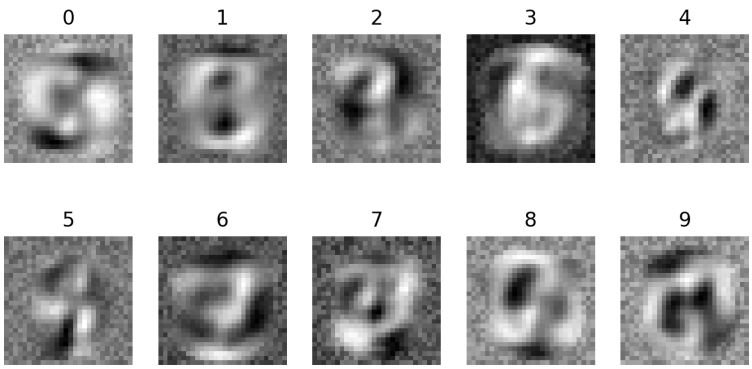
- Visualize the weights of the hidden layer that generate codes
  - each hidden node activates on a particular structure

In [44]:
```
W = encoder.get_layer(index=2).get_weights()[0]
filter_list = [W[:,i].reshape((28,28)) for i in range(W.shape[1])]
plt.figure(figsize=(8,4))
show_imgs(filter_list, nc=5, titles="node %d")
```



- Visualize the weights that project the code into an image
  - the image structures match those of the encoder, but are smoother.

In [45]:
```
W = decoder.get_layer(index=1).get_weights()[0]
filter_list = [W[i,:].reshape((28,28)) for i in range(W.shape[0])]
plt.figure(figsize=(8,4))
show_imgs(filter_list, nc=5, titles="%d")
```



- Visualize the reconstruction of the input image

In [46]:
```
testXrecon = decoder.predict(encoder.predict(testXraw))
```

```
2023-01-24 21:24:48.268968: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

In [48]:
```
rfig
```

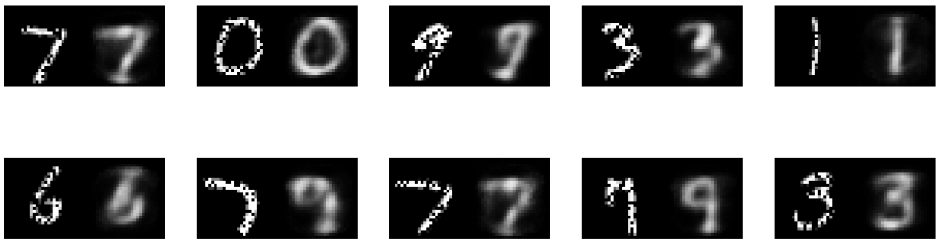
Out[48]:

- Corrupt the input image and encode-decode
  - performs "denoising" of the input

In [49]:
```
noisytest = testXraw * random.binomial(n=1,p=1-0.3,size=testXraw.shape)
testXrecon = decoder.predict(encoder.predict(noisytest))
```

In [51]:
```
dfig
```

Out[51]:



DAE Interpretation 2

- Learning a reconstruction distribution $p_{\text{recon}}(\mathbf{x}|\tilde{\mathbf{x}})$
- Generative process:
  1. sample training example $\mathbf{x}$
  2. corrupt $\tilde{\mathbf{x}} \sim C(\tilde{\mathbf{x}}|\mathbf{x})$
  3. use $(\tilde{\mathbf{x}}, \mathbf{x})$ as a training example to estimate

$$p_{\text{recon}}(\mathbf{x}|\tilde{\mathbf{x}}) = p_{\text{decoder}}(\mathbf{x}|\mathbf{z}),$$
$$\mathbf{z} = f(\tilde{\mathbf{x}})$$

- Assume $p_{\text{decoder}}(\mathbf{x}|\mathbf{z})$ is Gaussian:
  - Note that $\log p_{\text{decoder}}(\mathbf{x}|\mathbf{z}) = -||\mathbf{x} - g(\mathbf{z})||^2$ (negative MSE loss)
- Minimize the loss over the training distribution
  - $\sum_i ||\mathbf{x}_i - g(f(\tilde{\mathbf{x}}_i))||^2$
- is equivalent to maximizing the expected log probability

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}}|\mathbf{x})}[\log p_{\text{decoder}}(\mathbf{x}|\mathbf{z} = f(\tilde{\mathbf{x}}))]$$

- expectation is over the data and the corruption.

Convolutional Auto-Encoder

- Encoder - a standard CNN (w/o classifier)
  - Extract a feature map

In [52]:
```
K.clear_session()
random.seed(4487); tf.random.set_seed(4487)

# the Conv2D encoder
input_img2 = Input(shape=(28, 28, 1))
x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img2)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded2 = MaxPooling2D((2, 2), padding='same')(x)
encoder2 = Model(input_img2, encoded2, name='ConvEncoder')
# the representation is (4, 4, 8) i.e. 128-dimensional
```

- Decoder
  - the opposite architecture
  - Replace maxpooling with upsampling

In [53]:
```
# the Conv2D decoder
encoded_input2 = Input(shape=(4,4,8))
x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded_input2)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu')(x)
```

```
x = UpSampling2D((2, 2))(x)
decoded2 = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
decoder2 = Model(encoded_input2, decoded2, name='ConvDecoder')
```

- Connect the two to form the autoencoder

In [54]:
```
# connect the encoder to the decoder
autoencoder2 = Model(input_img2, decoder2(encoder2(input_img2)))
autoencoder2.compile(optimizer='adam', loss='binary_crossentropy')
```

- Encoder and Decoders

In [55]:
```
encoder2.summary()
```

```
Model: "ConvEncoder"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_1 (InputLayer)         [(None, 28, 28, 1)]       0

 conv2d (Conv2D)              (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2D  (None, 14, 14, 16)       0
 )

 conv2d_1 (Conv2D)            (None, 14, 14, 8)         1160

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 8)          0
 2D)

 conv2d_2 (Conv2D)            (None, 7, 7, 8)           584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 8)          0
 2D)

=================================================================
Total params: 1,904
Trainable params: 1,904
Non-trainable params: 0
_________________________________________________________________
```

In [56]:
```
decoder2.summary()
```

```
Model: "ConvDecoder"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_2 (InputLayer)         [(None, 4, 4, 8)]         0

 conv2d_3 (Conv2D)            (None, 4, 4, 8)           584

 up_sampling2d (UpSampling2D  (None, 8, 8, 8)          0
 )

 conv2d_4 (Conv2D)            (None, 8, 8, 8)           584

 up_sampling2d_1 (UpSampling  (None, 16, 16, 8)        0
 2D)

 conv2d_5 (Conv2D)            (None, 14, 14, 16)        1168

 up_sampling2d_2 (UpSampling  (None, 28, 28, 16)       0
 2D)

 conv2d_6 (Conv2D)            (None, 28, 28, 1)         145

=================================================================
Total params: 2,481
Trainable params: 2,481
```

```
Non-trainable params: 0
_________________________________________________________________
```

- The whole autoencoder

In [57]:
```
autoencoder2.summary()
```

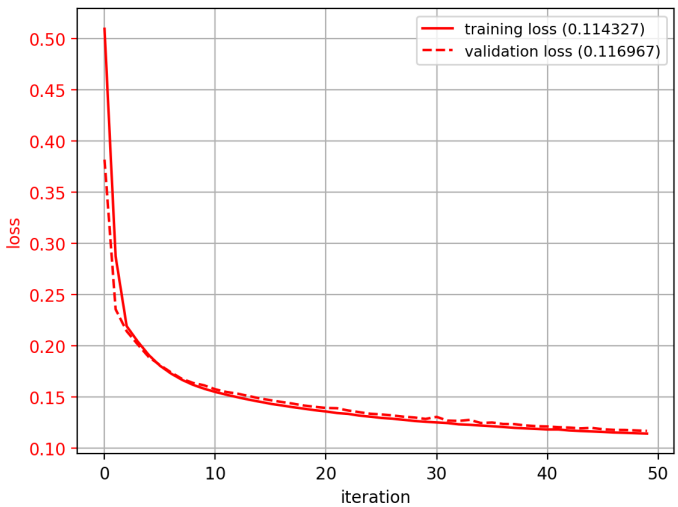
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 ConvEncoder (Functional)    (None, 4, 4, 8)           1904

 ConvDecoder (Functional)    (None, 28, 28, 1)         2481

=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
```

- Now fit the model

In [58]:
```
# training with images
history = autoencoder2.fit(vtrainI, vtrainI,
                           epochs=50,
                           batch_size=128,
                           shuffle=True,
                           callbacks=[earlystop, TensorBoard(log_dir='./logs/ae2')],
                           validation_data=(validI, validI), verbose=False)
```

```
2023-01-24 21:25:04.880868: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-24 21:25:08.306738: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

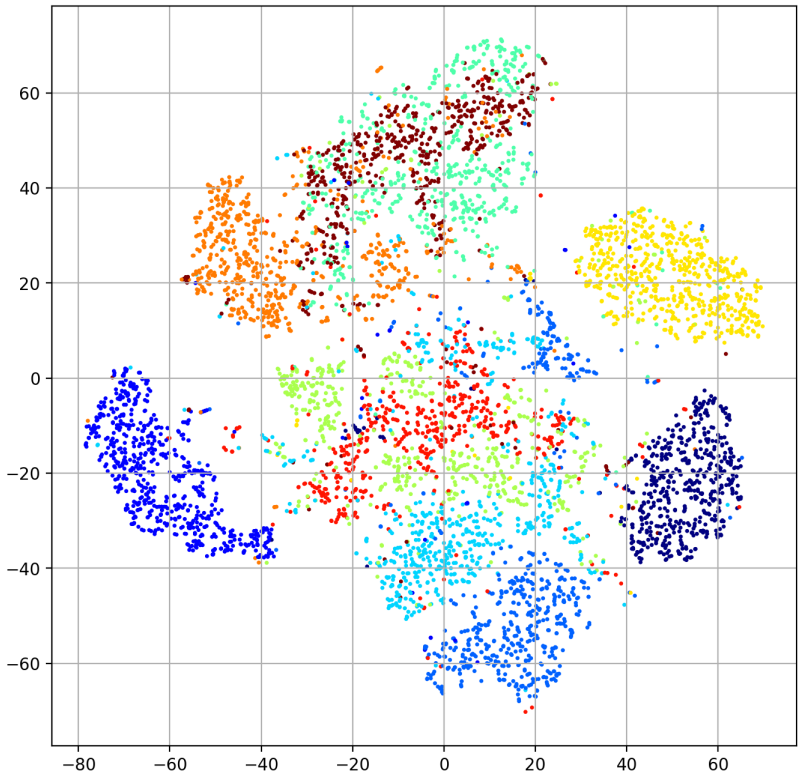
In [59]:
```
plot_history(history)
```



- Visualize the latent space

In [60]:
```
Z = encoder2.predict(trainI)
plot_embedding(Z.reshape((Z.shape[0],-1)), trainY, perplexity=30.)
```

```
2023-01-24 21:25:51.273105: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```



- Encode and reconstruct an image

In [61]:
```
testIrecon = decoder2.predict(encoder2.predict(testI))
```

```
2023-01-24 21:26:14.840807: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

- Better visualization

In [63]:
```
rfig
```

Out[63]:

- Traverse the latent space
  - Sample points:

$$\mathbf{z}_1 = f(\mathbf{x}_1), \mathbf{z}_2 = f(\mathbf{x}_2)$$

  - interpolate:

$$\hat{\mathbf{z}} = (1-\alpha)\mathbf{z}_1 + \alpha\mathbf{z}_2, 0 \leq \alpha \leq 1$$

  - reconstruct:

$$\hat{\mathbf{x}} = g(\hat{\mathbf{z}})$$

- Change a 7 into a 1

In [65]:
```
inds = [0,40]
X = encoder2.predict(testI[inds,:])
Xd = decoder2.predict(Xinterp(X))
```

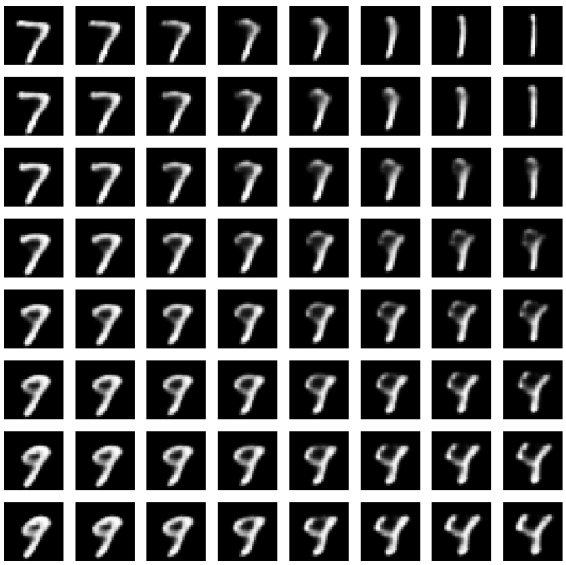
In [67]:
```
rfig
```

Out[67]:

- Traverse the latent space between a 7, 1, 9, and 4
  - captures shapes in between

In [70]:
```
rfig
```

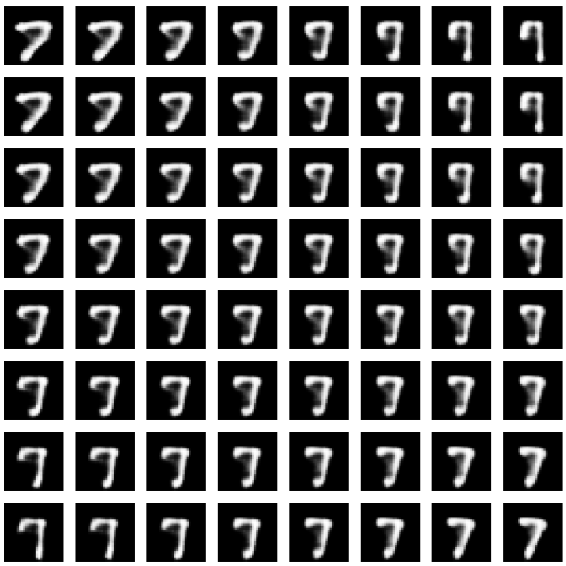
Out[70]:



- Traverse the latent space between different 7s
  - captures different shapes of 7

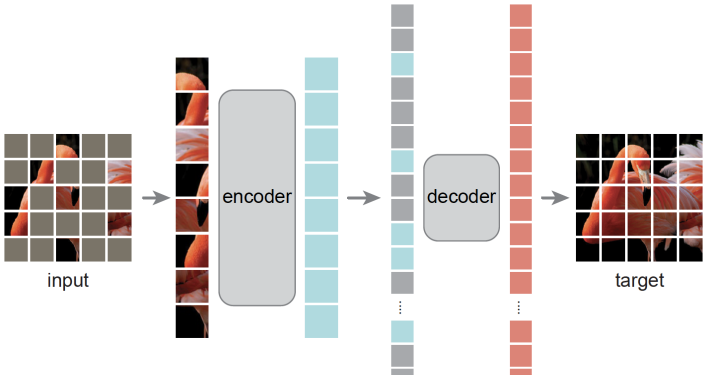
In [72]:
```
rfig
```

Out[72]:



Masked Autoencoder (MAE)

- Combine ViT with auto-encoder for self-supervised representation learning.
  - encode subset of input patches (mask out 75%)
  - decode encoded & masked patches to reconstruct the image
- Pre-train on IN1k (w/o labels)



- Fine-tune on IN1k classification task
  - better than supervised pre-training of ViT with IN1k
  - but worse than supervised pre-training with large-scale data (JFT)

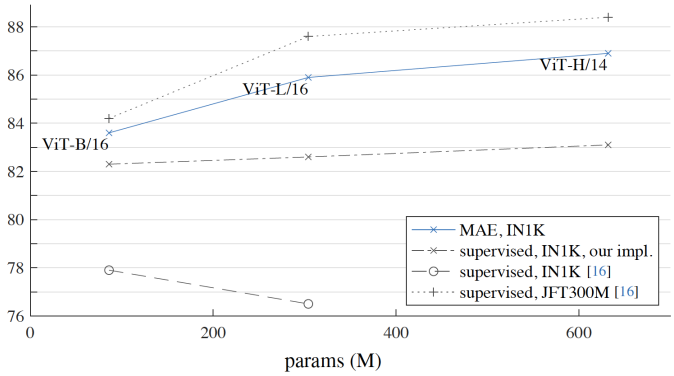
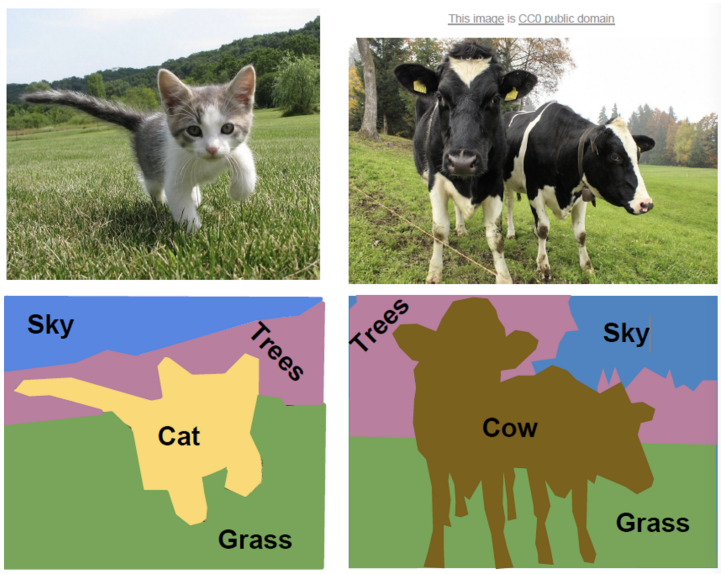


Figure 8. **MAE pre-training *vs*. supervised pre-training**, evaluated by fine-tuning in ImageNet-1K (224 size). We compare with the original ViT results [16] trained in IN1K or JFT300M.

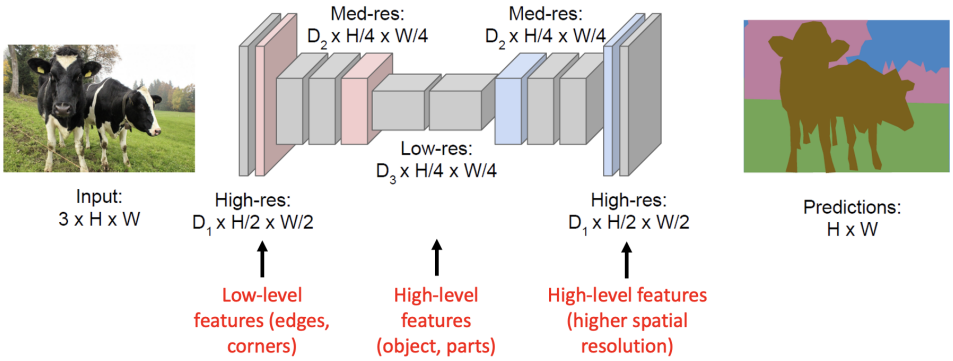
Semantic Segmentation

- Label each pixel in the image with a category label.



Fully-convolutional network (FCN)

- learns mapping from input image to output segmentation
  - encoder/decoder structure similar to AE.
  - middle contains the low-dim representation.



U-net: More sophisticated FCN

- Idea: concatenate higher-resolution features when upsampling
  - Better localize boundaries.
  - Mix high-level and low-level features.