7. Autoencoders: Q&A Session

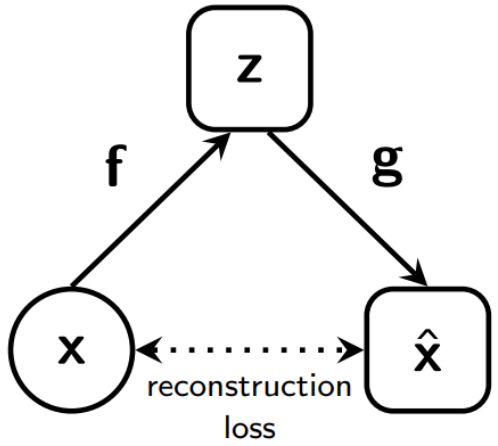
CS-E4890 Deep Learning

Aalto University
School of Science

Anni Niskanen
28.4.2023

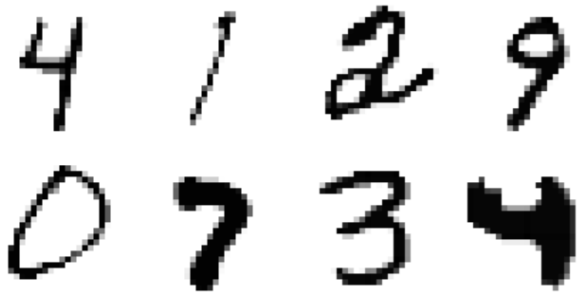
Autoencoders: Short recap

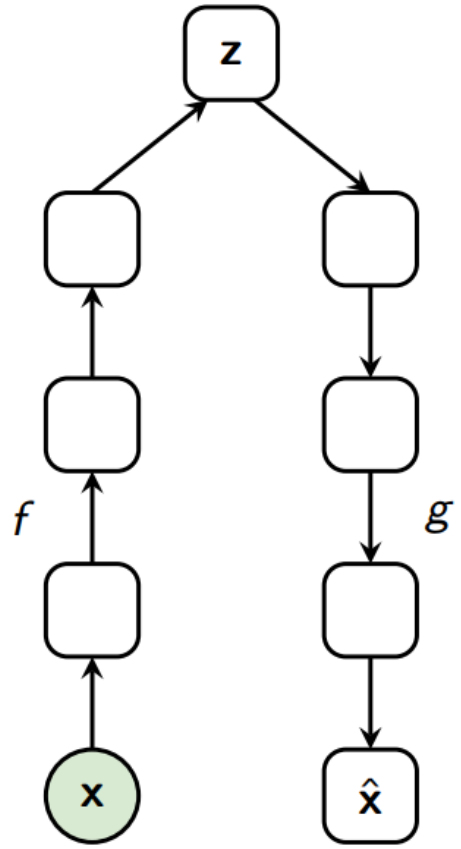
- Unsupervised Learning
- Basic idea:
  1. Encode data with the encoder
     - Goal is to compress data and find new features
  2. Decode with the decoder
- Uses:
  - Data compression
  - Feature extraction
  - Sample generation



Task 1: Vanilla bottleneck autoencoder (AE)

- Both the encoder and decoder are deep neural networks
- MNIST dataset
- Fairly straightforward





Task 1: Vanilla bottleneck autoencoder (AE)

Step 1: Encoder

```
class Encoder(nn.Module):
    def __init__(self, n_components):
        """
        Args:
          n_components (int): Number of elements in produced encodings.
        """
        # YOUR CODE HERE
        raise NotImplementedError()

    def forward(self, x):
        """
        Args:
          x of shape (batch_size, n_channels=1, width, height): Examples to encode.

        Returns:
          z of shape (batch_size, n_components): Produced encodings.
        """
        # YOUR CODE HERE
        raise NotImplementedError()
```

Task 1: Vanilla bottleneck autoencoder (AE)

Step 2: Decoder

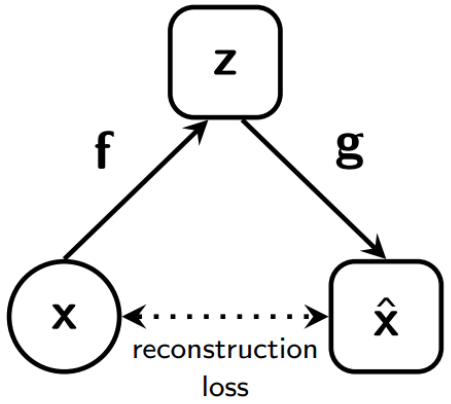
```
class Decoder(nn.Module):
    def __init__(self, n_components):
        """
        Args:
          n_components (int): Number of elements in input codes.
        """
        # YOUR CODE HERE
        raise NotImplementedError()

    def forward(self, z):
        """
        Args:
          z of shape (batch_size, n_components): Codes to decode.

        Returns:
          xrec of shape (batch_size, n_channels=1, width, height): Reconstructions computed from the given codes.
        """
        # YOUR CODE HERE
        raise NotImplementedError()
```

Task 1: Vanilla bottleneck autoencoder (AE)

Step 3: Training loop

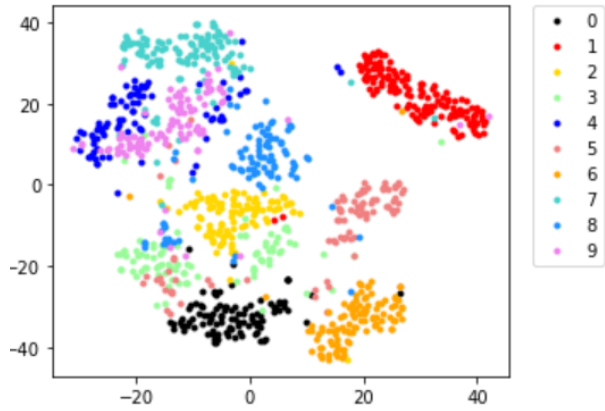
```
# Training loop
if not skip_training:
    # YOUR CODE HERE
    raise NotImplementedError()
```

- MSELoss for reconstruction loss
- No need for a GPU



Step 4: Evaluate embeddings

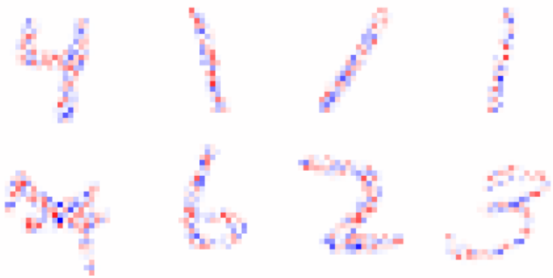
- Visualise embeddings with t-SNE
- Train a classifier on the embeddings
  - If the embeddings are meaningful, the classifier can achieve a high accuracy

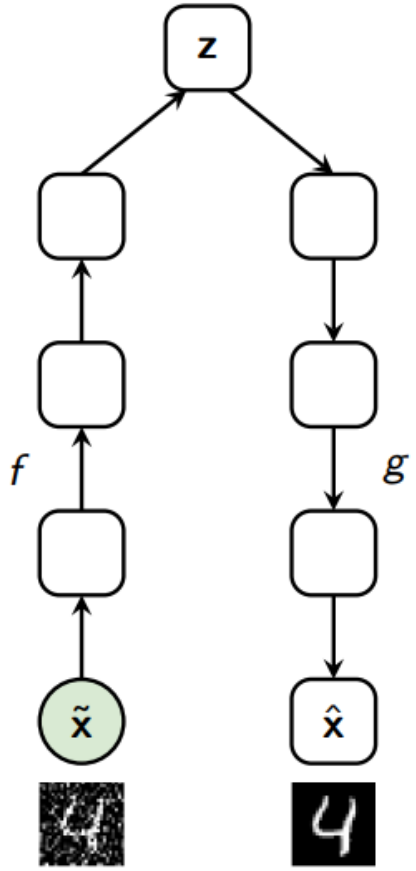


Aalto University
School of Science

Task 2: Denoising autoencoder (DAE)

- varianceMNIST dataset: information is in pixel variances, not pixel intensities
  - Vanilla autoencoder cannot encode the information!
- Denoising autoencoder:
  - The encoder is given inputs corrupted with (Gaussian) noise
  - DAE aims to reconstruct the original, noiseless data
- Do not confuse the varianceMNIST generation and noise corruption processes!





Task 2: Denoising autoencoder (DAE)

Step 1: DAE (encoder + decoder)

```
class DAE(nn.Module):
    def __init__(self, n_components=10):
        """
        Args:
          n_components (int): Number of outputs in the bottleneck layer.
        """
        # YOUR CODE HERE
        raise NotImplementedError()

    def forward(self, x):
        """
        Args:
          x of shape (batch_size, n_channels=1, width, height): Examples corrupted with noise.

        Returns:
          z of shape (batch_size, n_components): Outputs of the bottleneck layer.
          denoised_x of shape (batch_size, n_channels=1, width, height): Denoised examples.
        """
        # YOUR CODE HERE
        raise NotImplementedError()
```

Task 2: Denoising autoencoder (DAE)

Step 1: DAE (encoder + decoder)

- Keeping track of the dimensions before and after each layer is very helpful for (un)flattening the data
- Optimal denoising function:

$$g(\tilde{x}) = \tilde{x}\,\mathrm{sigmoid}(f(\sigma_x^2, \sigma_n^2))$$

- We do not know the parameters of f, but f must be some function of the input data

Task 2: Denoising autoencoder (DAE)

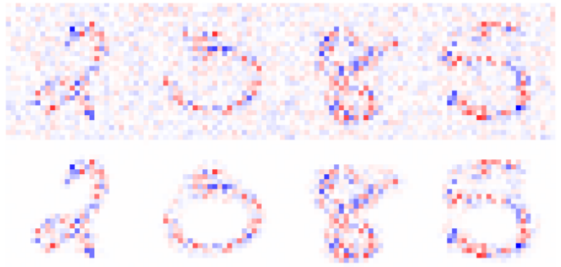
Step 2: Training loop

```
# Training loop
if not skip_training:
    # YOUR CODE HERE
    raise NotImplementedError()
```

- First corrupt the training data with Gaussian noise
- No need for a GPU

Step 3: Evaluate embeddings

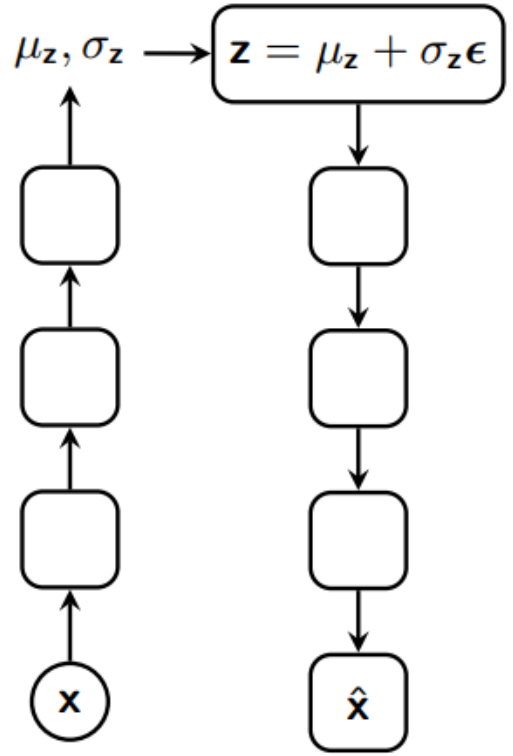
- Visualise embeddings with t-SNE
- Denoise some test images
- Train a classifier on the embeddings
    - If the embeddings are meaningful, the classifier can achieve a high accuracy

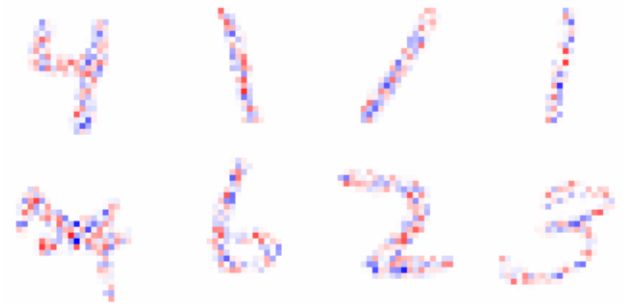


Aalto University
School of Science

Task 3: Variational autoencoder (VAE)

- Variational autoencoder:
  - A generative model: learns the data distribution and can be used to generate new samples
  - Encoder outputs means $\mu_{\mathbf{z}}$ and variances $\sigma_{\mathbf{z}}$ of the code $\mathbf{z}$
  - $\mathbf{z}$ are then sampled using $\mu_{\mathbf{z}}$ and $\sigma_{\mathbf{z}}$ and given to the decoder
- Decoder outputs $\mu_{\boldsymbol{x}}$ and $\sigma_{\boldsymbol{x}}$ (= predictive distribution of the data) instead of reconstructing the data
- varianceMNIST dataset





Task 3: Variational autoencoder (VAE)

Step 1: Encoder

```
class Encoder(nn.Module):
    def __init__(self, n_components):
        """
        Args:
          n_components (int): Number of elements in produced codes.
        """
        # YOUR CODE HERE
        raise NotImplementedError()

    def forward(self, x):
        """
        Args:
          x of shape (batch_size, n_channels=1, width, height): Examples to encode.

        Returns:
          z_mean of shape (batch_size, n_components): Means of the approximate distributions of the codes.
          z_logvar of shape (batch_size, n_components): Log-variances of the approximate distributions of the codes.
        """
        # YOUR CODE HERE
        raise NotImplementedError()
```

Task 3: Variational autoencoder (VAE)

Step 1: Encoder

```
    def sample(self, z_mean, z_logvar):
        """Draw one sample from the posterior of the latent codes described by given parameters.
        This is needed for the re-parameterization trick.

        Args:
          z_mean of shape (batch_size, n_components): Means of the approximate distributions of the codes.
          z_logvar of shape (batch_size, n_components): Log-variance of the approximate distributions of the codes.

        Returns:
          z of shape (batch_size, n_components): Drawn samples.
        """
        # YOUR CODE HERE
        raise NotImplementedError()
```

- Sample with

$$\mathbf{z} = \mu_{\mathbf{z}} + \sigma_{\mathbf{z}}\epsilon$$

Task 3: Variational autoencoder (VAE)

Step 2: KL divergence loss

```
def loss_kl(z_mean, z_logvar):
    """
    Args:
      z_mean of shape (batch_size, n_components): Means of the approximate distributions of the codes.
      z_logvar of shape (batch_size, n_components): Log-variance of the approximate distributions of the codes.

    Returns:
      loss (torch scalar): Kullback-Leibler divergence.
    """
    # YOUR CODE HERE
    raise NotImplementedError()
```

- The loss

$$\frac{1}{N}\sum_{i=1}^{N}\int q(z_i)\log\frac{q(z_i)}{p(z_i)}dz_i$$

  can be simplified to a form without integration, check the original VAE paper if stuck

Task 3: Variational autoencoder (VAE)

Step 3: Decoder

```
class Decoder(nn.Module):
    def __init__(self, n_components):
        """
        Args:
          n_components (int): Number of elements in input codes.
        """
        # YOUR CODE HERE
        raise NotImplementedError()

    def forward(self, z):
        """
        Args:
          z of shape (batch_size, n_components): Input codes.

        Returns:
          y_mean of shape (batch_size, n_channels=1, width, height): Means of the probability distributions
              describing the data examples that correspond to the given codes.
          y_logvar of shape (batch_size, n_channels=1, width, height): Log-variances of the probability
              distributions describing the data examples that correspond to the given codes.
        """
        # YOUR CODE HERE
        raise NotImplementedError()
```

Task 3: Variational autoencoder (VAE)

Step 4: Expected log-likelihood loss

```
def loss_loglik(y_mean, y_logvar, x):
    """
    Args:
      y_mean of shape (batch_size, 1, 28, 28): Predictive mean of the VAE reconstruction of x.
      y_logvar of shape (batch_size, 1, 28, 28): Predictive log-variance of the VAE reconstruction of x.
      x of shape (batch_size, 1, 28, 28): Training samples.

    Returns:
      loss (torch scalar): Expected log-likelihood loss.
    """
    # YOUR CODE HERE
    raise NotImplementedError()
```

- The loss

$$-\int q(z_i) \log \mathcal{N}\left(x_i \mid \mu_x(z_i),\ \mathrm{diag}(\sigma_x^2(z_i))\right) dz_i$$

  can be simplified to a form without integration, check the lecture slides if stuck

Task 3: Variational autoencoder (VAE)

Step 5: Training loop

```
# Training loop
if not skip_training:
    # YOUR CODE HERE
    raise NotImplementedError()
```

- Sample after encoding, give samples to decoder
- Takes a little longer to train, but no GPU needed

Step 6: Evaluate embeddings

- Visualise embeddings with t-SNE
- Visualise predictive variances
- Generate some samples
- Train a classifier on the embeddings

Aalto University
School of Science

Questions?