AI504: Programming for Artificial Intelligence

Week 4: Autoencoders

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr

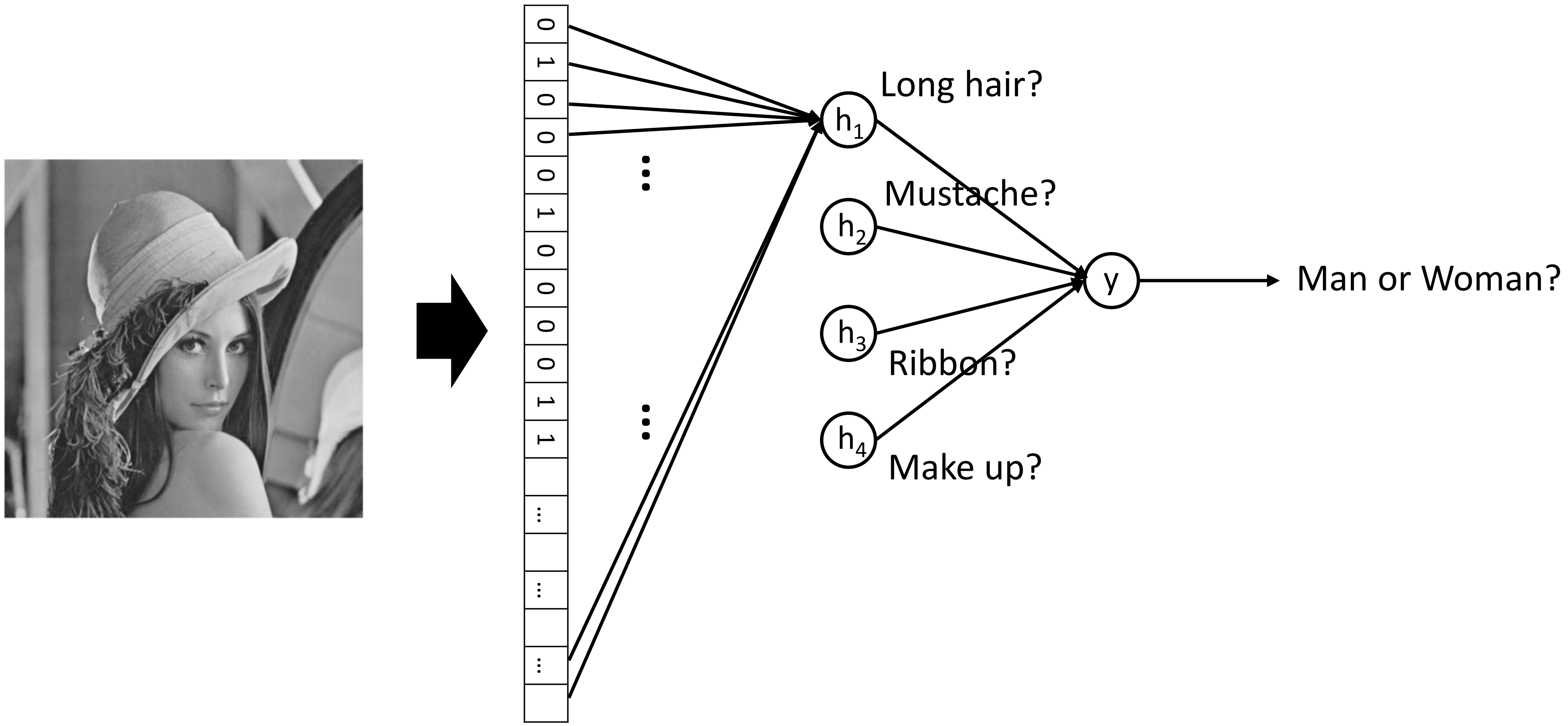
Today’s Topic

- Latent representation
- Autoencoders
- Training Process (not just for autoencoders)
- Visualization
- Autoencoder variants
  - Denoising autoencoder
  - Sparse autoencoder

Latent Representation

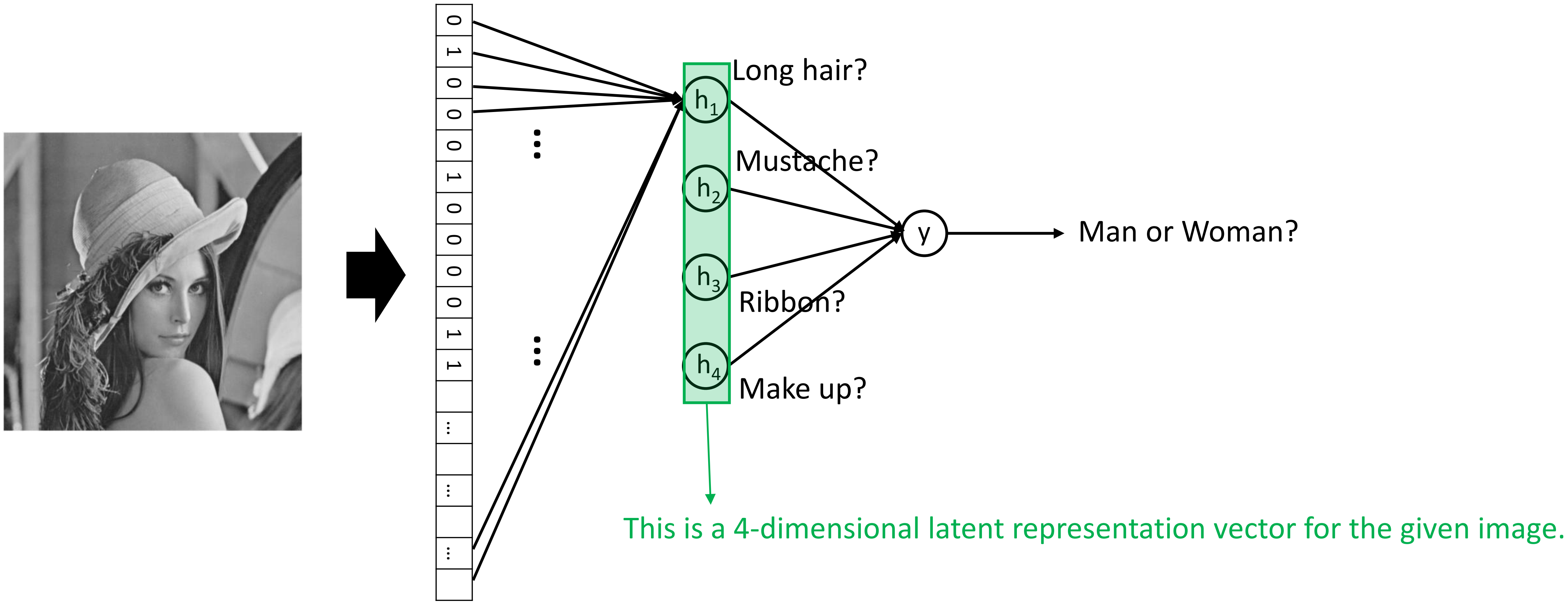
Latent Representation

- Man/Woman classification with an MLP.



Latent Representation

- 4-dimensional hidden representation



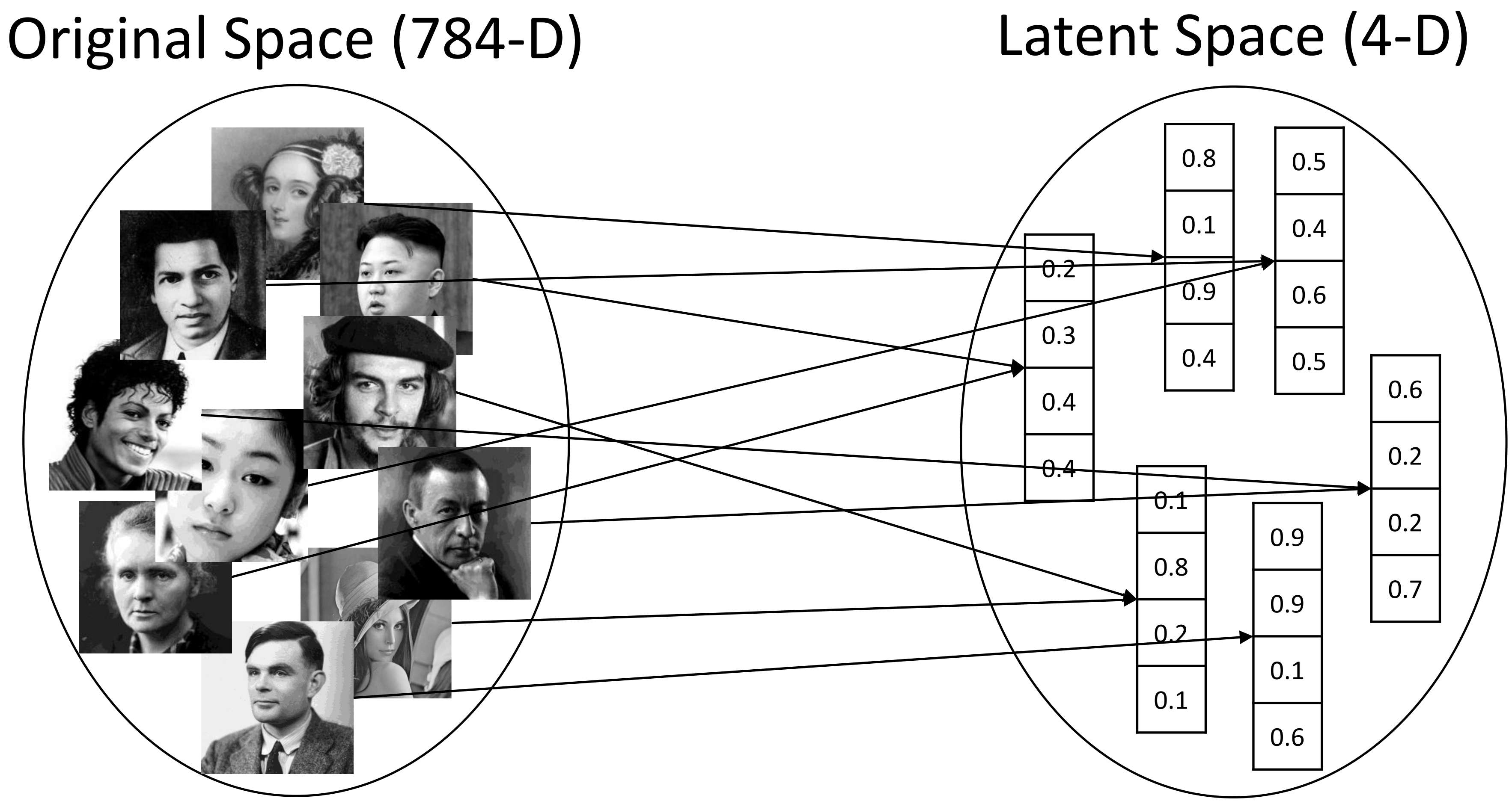
Latent Representation

Linear?/Non-linear? Transformation

0.8
0.1
0.9
0.4

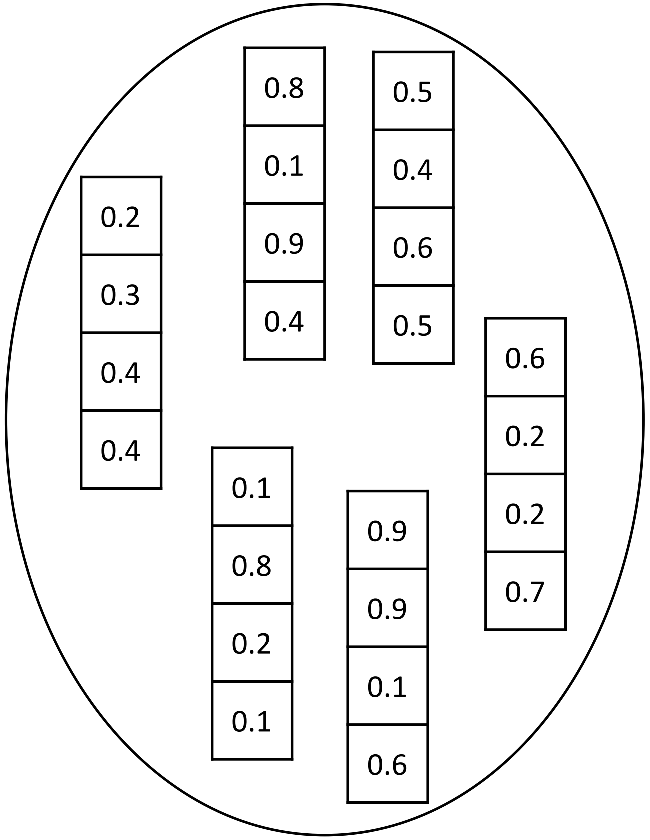
0.1
0.2
0.1
0.1

Latent Representation



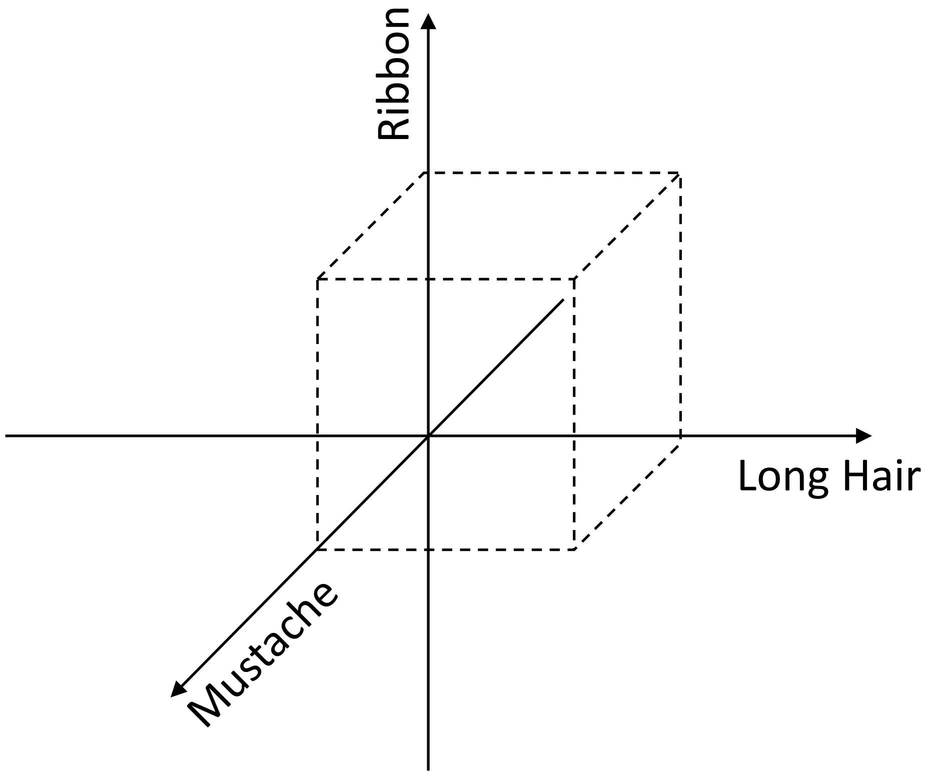
Latent Representation

Latent Space (4-D)



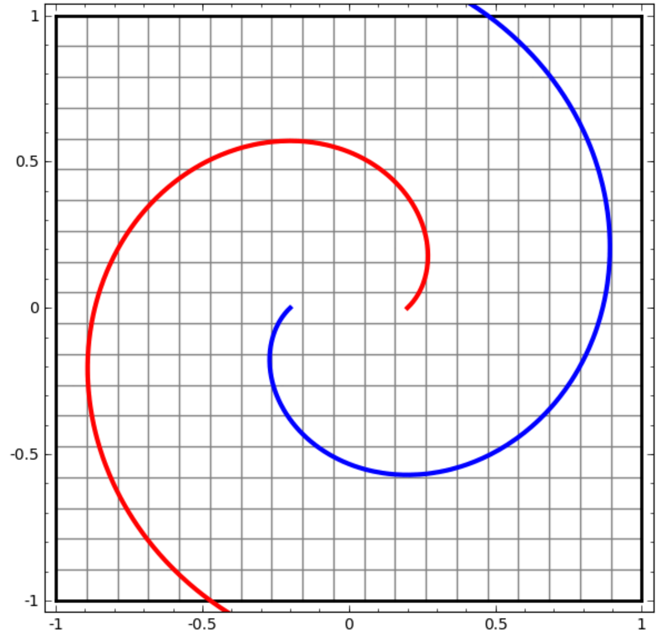
Consists of 4 axes:

- Long Hair?
- Mustache?
- Ribbon?
- Make up?



Latent Space

- How did we learn this space?
  - By minimizing the cross entropy loss!



https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Learning the 2D space for classification

Latent Space

- Your latent space is shaped by your loss function (your task)
  - Man/Woman classification
  - Iris classification
  - House price regression
  - Word embedding
  - French-English translation

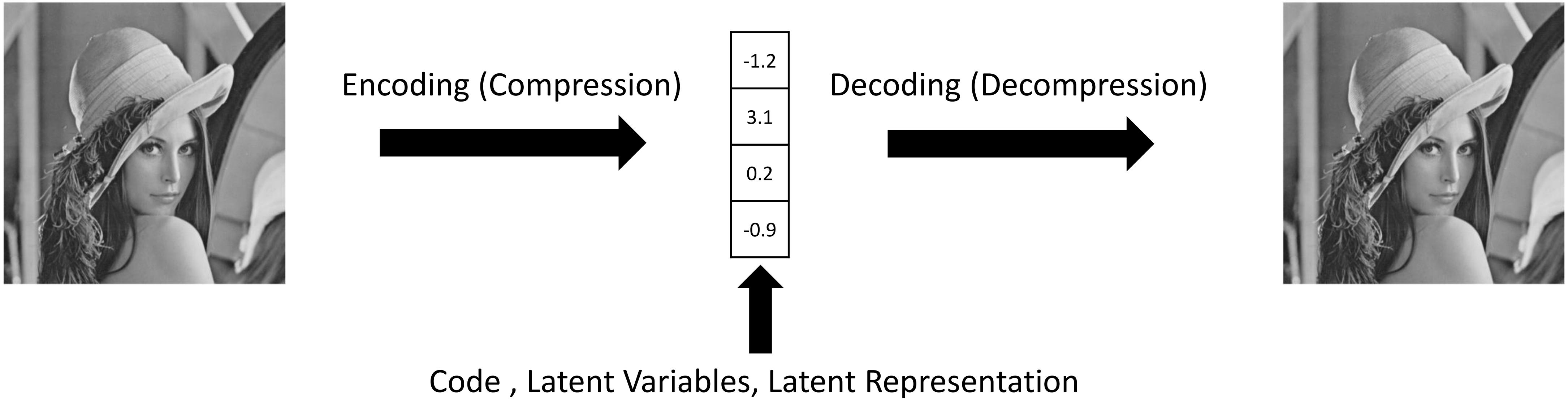
Latent Space

- Your latent space is shaped by your loss function (your task)
  - Man/Woman classification
  - Iris classification
  - House price regression
  - Word embedding
  - French-English translation
  - **Image compression**

Autoencoders

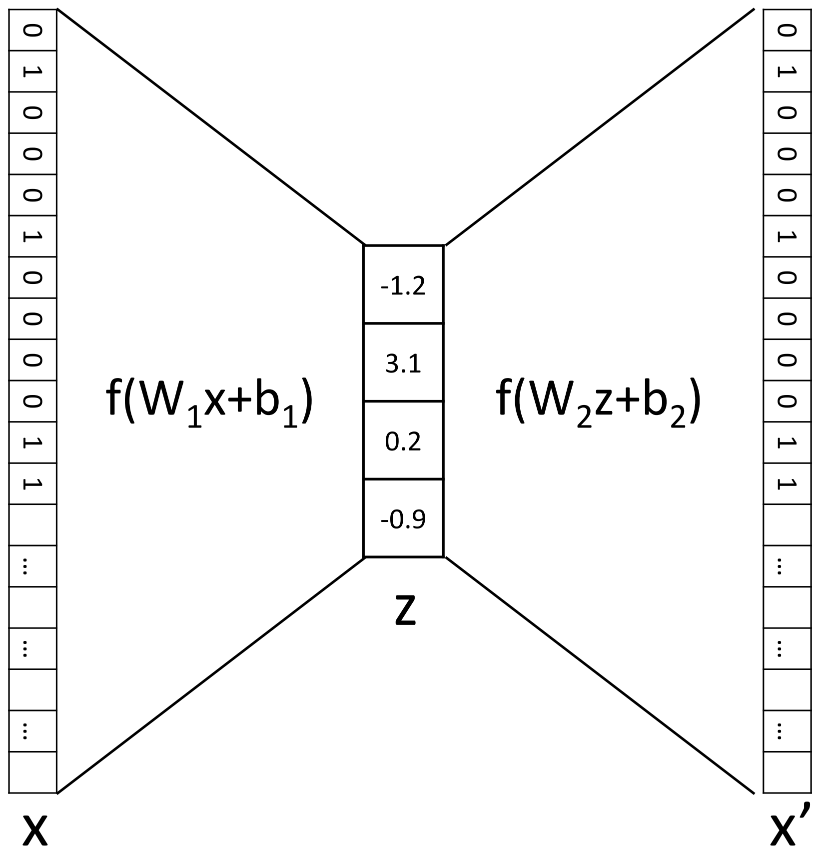
Autoencoder

- Consists of Encoder and Decoder



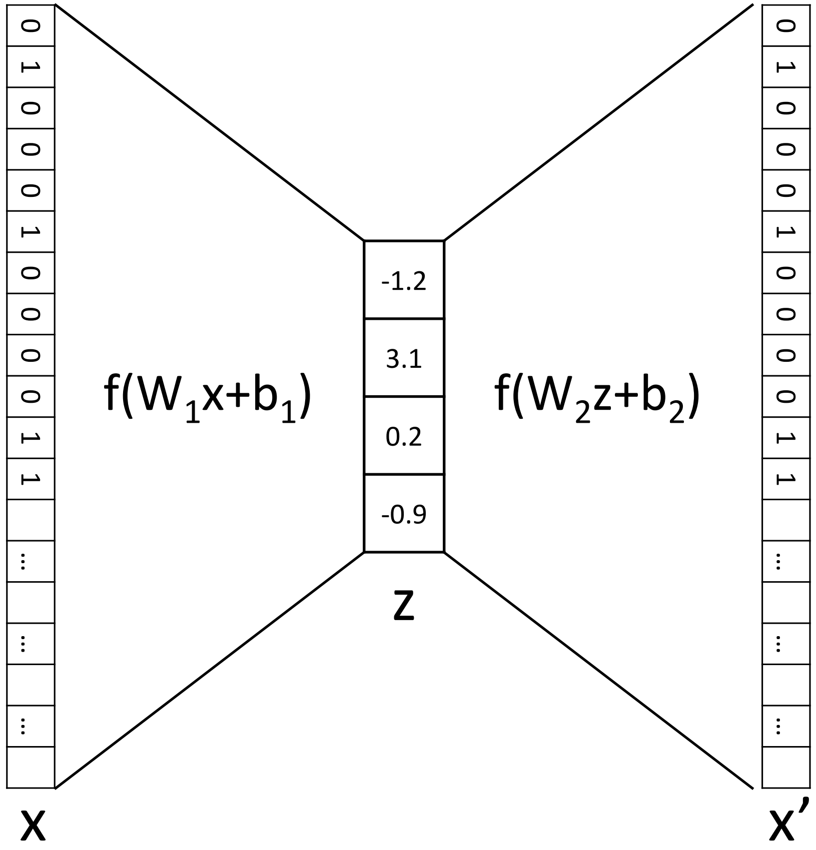
Autoencoder

- Consists of Encoder and Decoder



Autoencoder

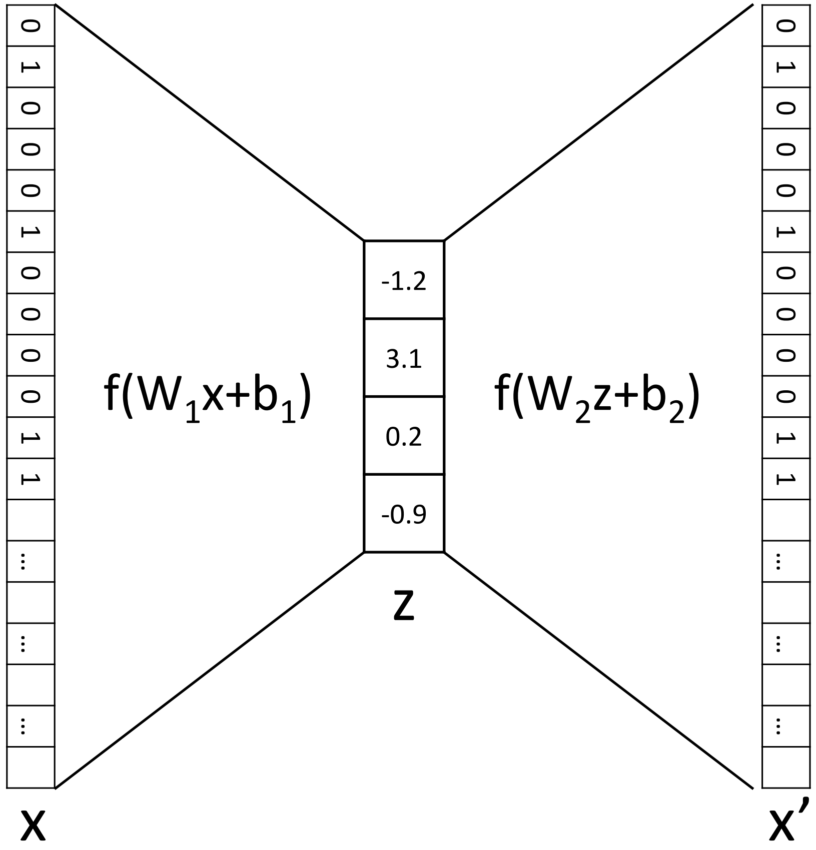
- Consists of Encoder and Decoder



- Encoding
  - $z = f(W_1x+b_1)$
- Decoding
  - $x' = f(W_2z+b_2)$
- Loss
  - $\mathcal{L}(\mathrm{x}, \mathrm{x}') = ??$

Autoencoder

- Mean Squared Error (MSE) loss

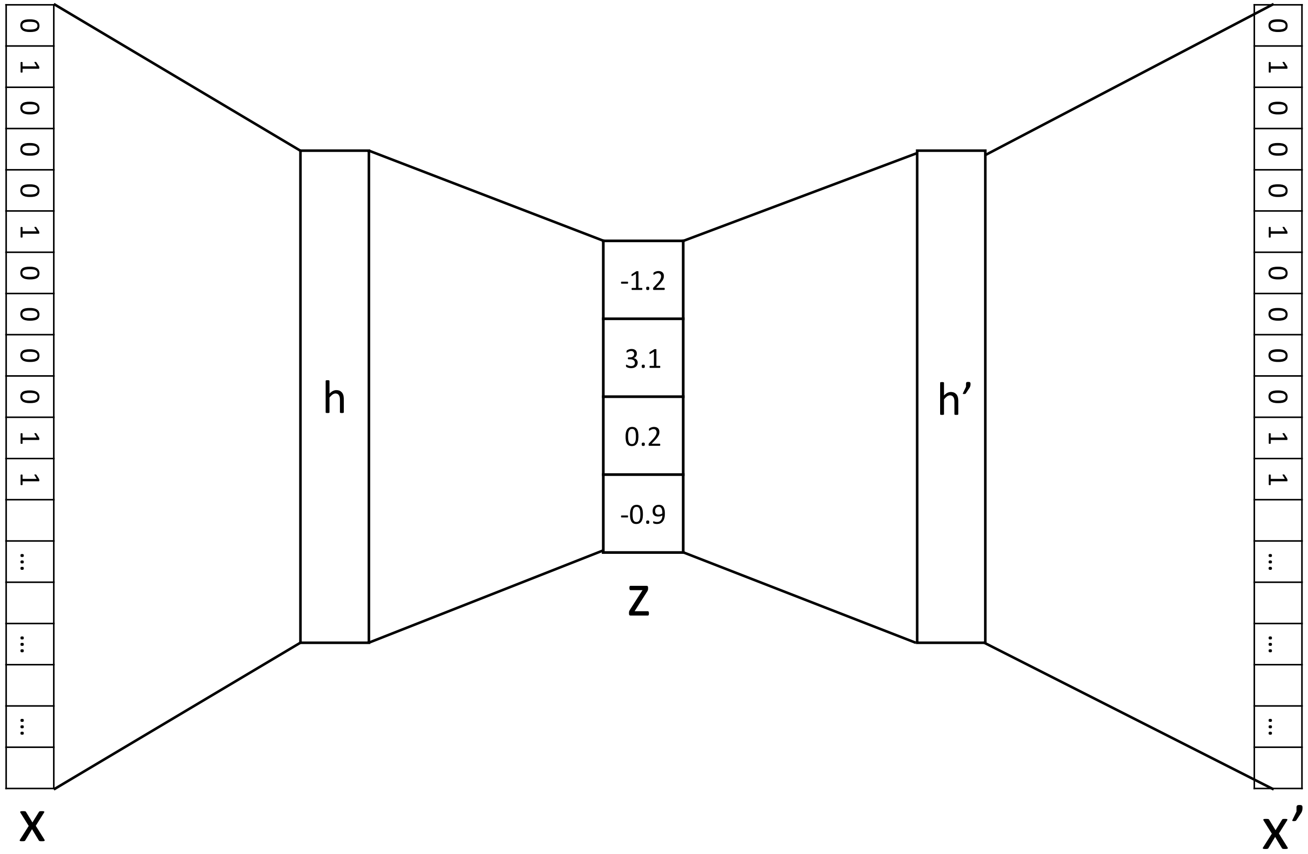


- Encoding
  - z = f(W_1x+b_1)
- Decoding
  - x’= f(W_2z+b_2)
- Loss
  - $\mathcal{L}(\mathrm{x}, \mathrm{x}') = \|\mathrm{x} - \mathrm{x}'\|_2^2$

    (Squared Error)

Autoencoder

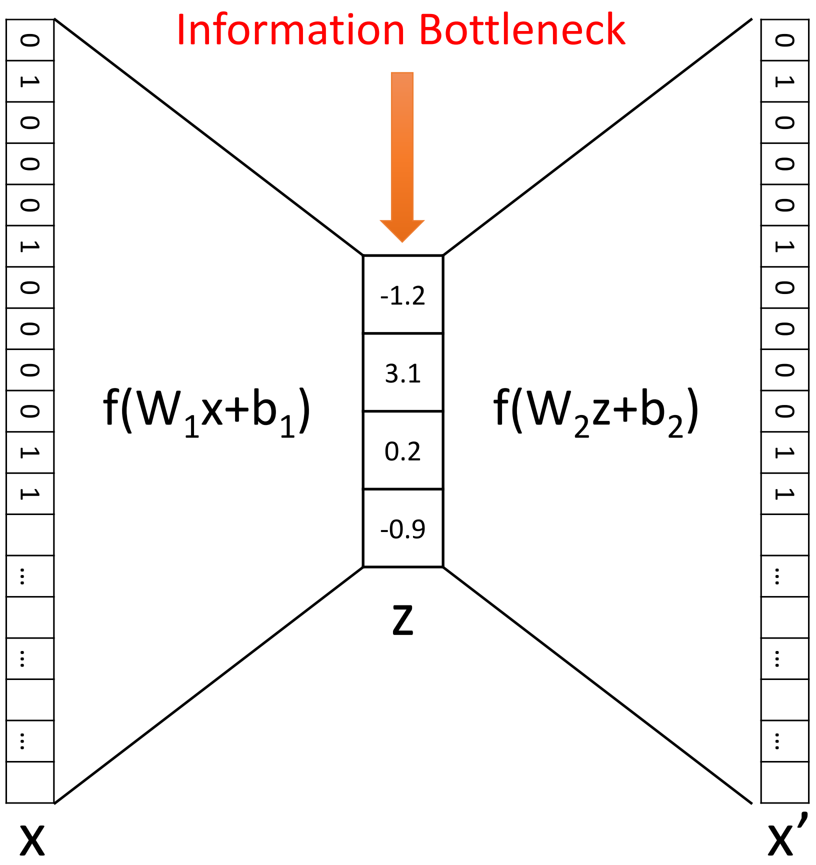
- Using muliple layers



- Encoding
  - $h = f(W_1x+b_1)$
  - $z = f(W_2h+b_2)$
- Decoding
  - $h' = f(W_3z+b_3)$
  - $x' = f(W_4h'+b_4)$

Autoencoder

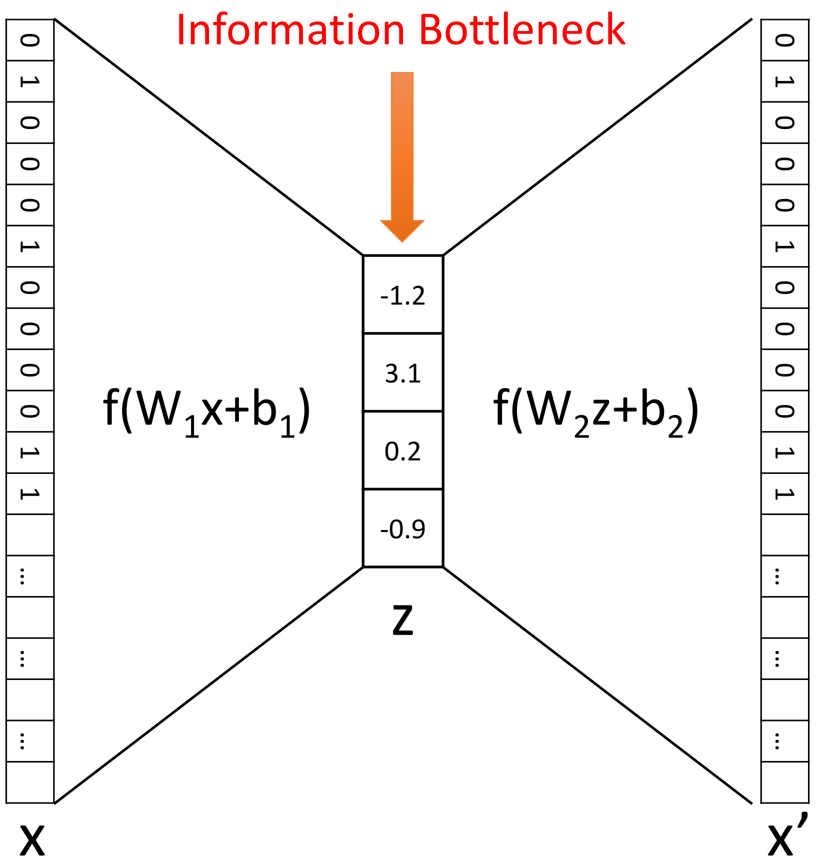
- Compression



Need to pack all information in 4-D
➔ Need to learn some useful hidden features

Autoencoder

- Compression

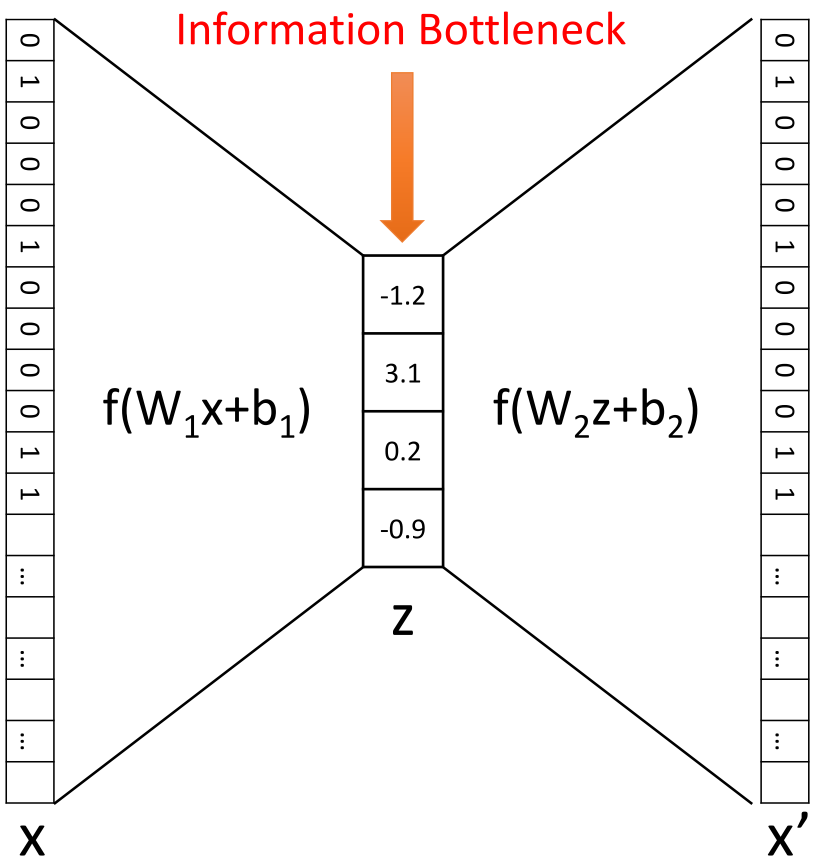


Still consists of 4 axes?

- Long Hair?
- Mustache?
- Ribbon?
- Make up?

Autoencoder

- Compression

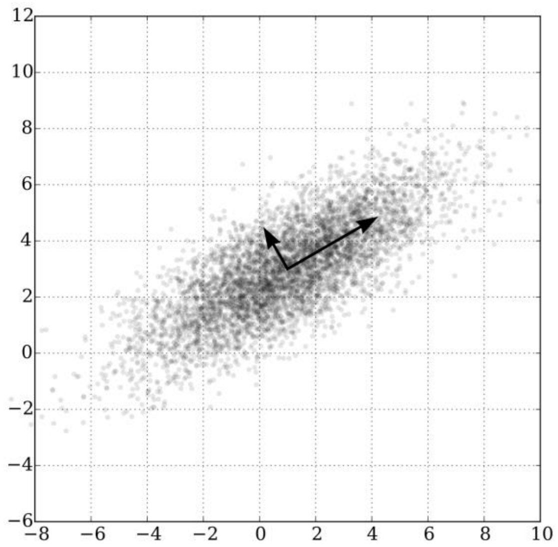


Still consists of 4 axes?

- Long Hair?
- Mustache?
- Ribbon?
- Make up?

Probably Not!

- Think about PCA



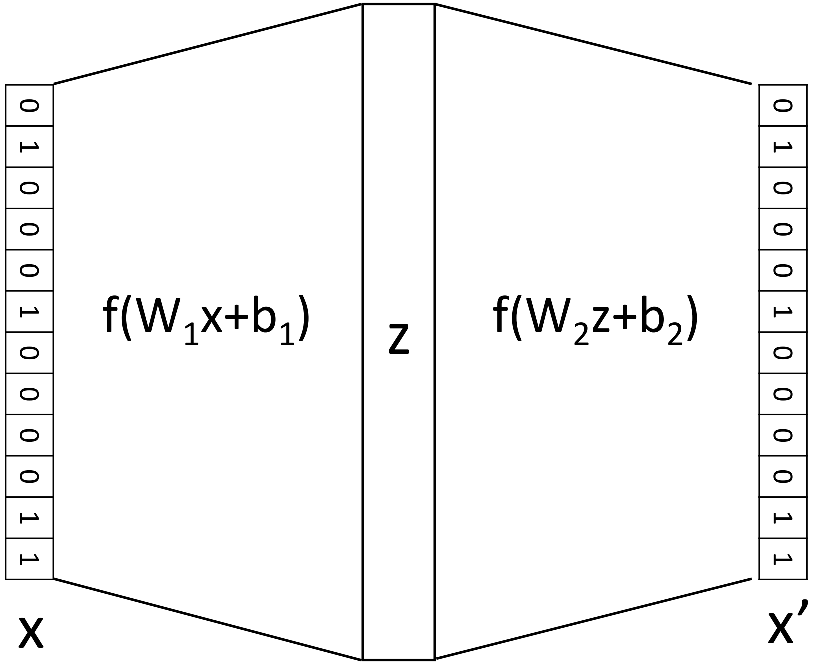
Autoencoders VS PCA

- PCA
  - Minimize reconstruction error
    - == Maximize variance
  - Linear transformation
  - Select k var-maximizing bases
    - Ignore small variance
  - Use kernels to map to higher dim space

- Autoencoder
  - MInimize reconstruction error
    - == Maximize variance
  - Non-Linear transformation
  - Use k dims to compress
    - Implicitly ignore small variance
  - Can easily map to higher dim space

Autoencoder

- Encoding to a higher dimensional space



If dim(z) > dim(x),
what would happen to the MSE loss?

Training Process

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

Training Neural Networks

- Split into train/validation/test
  - 8:1:1, 6:2:2 depending on real-world use case
  - Consider N-fold cross validation

➔ Can use scikit-learn train_test_split

Training Neural Networks

- Split into train/validation/test
- Load data
  - Entire data into the memory
    - Handling small datasets
  - Stream data from the disk
    - When data doesn't fit in memory
    - When streaming from DB, video, audio

→ Use torch.utils.data.Dataset & torch.utils.data.DataLoader

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M* ➔ Use torch.nn.Module
  - Define loss *L*
    - Cross entropy, MSE, L_2 Regularization

  ➔ Use torch.nn.* (or define your own)

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
  - SGD, Adagrad, Adam, Adamax, etc.

→ Use torch.optim.*

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
    - Fetch a random minibatch (*X*, *y*)
    - Push *X* through *M*
      - Obtain *y'*
    - Calculate *L*(*y*, *y'*)
    - Use *O* to update *M*'s params
  - Evaluate *M* on validation set
    - If best validation perf, save *M*

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
    - Fetch a random minibatch (*X*, *y*)

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model M
  - Define loss L
- Define optimizer O
- Training Loop
- Evaluate best M on test set

- Repeat for N epochs
  - Repeat for K batches
    - Fetch a random minibatch (X, y)
      - X: B number of samples
      - y: B number of labels

➔ Use torch.utils.data.DataLoader

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
    - Fetch a random minibatch (*X*, *y*)
    - Push *X* through *M*

    ➔ *y’ = M(X)*

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
    - Fetch a random minibatch (*X*, *y*)
    - Push *X* through *M*
    - Calculate *L*(*y*, *y'*)

    ➔ *c = L(y, y')*

    (Also print c every once in a while)

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
    - Fetch a random minibatch (*X*, *y*)
    - Push *X* through *M*
      - Obtain *y'*
    - Calculate *L*(*y*, *y'*)
    - Use *O* to update *M*'s params

    ➔ *O.zero_grad()*
    c.backward()
    O.step()

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
  - Evaluate *M* on validation set

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
  - Repeat for *J* batches of valid. set
    - Fetch next minibatch (*X*, *y*)
      - Use another DataLoader
    - Push *X* through *M*
      - Obtain *y'*
    - Save batch performance
      - $y == y'$, $(y - y')^2$
  - Calculate validation performance

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Repeat for *N* epochs
  - Repeat for *K* batches
  - Repeat for *J* batches of valid. set
  - Calculate validation performance
    - Accuracy, AUROC, MSE, etc.
    - If best validation perf so far, save *M*

    ➔copy.deepcopy(*M*.state_dict())

    or torch.save(*M*.state_dict(), path)

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Load best *M*
- Evaluate best *M* on test set
- Calculate test performance

Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
  - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Load best *M*

  ➔*M.load_state_dict(model_state_dict)*

  or M.load_state_dict(torch.load(path))

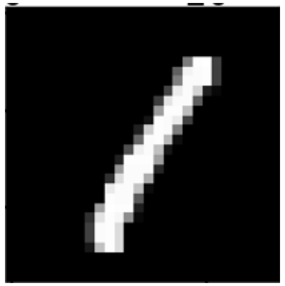
Training Neural Networks

- Split into train/validation/test
- Load data
- Define a model *M*
    - Define loss *L*
- Define optimizer *O*
- Training Loop
- Evaluate best *M* on test set

- Load best *M*
- Repeat for *T* batches of test set
    - Fetch next minibatch (*X*, *y*)
        - Use another DataLoader
    - Push *X* through *M*
        - Obtain *y'*
    - Save batch performance
        - $y == y'$, $(y - y')^2$
- Calculate test performance
    - Accuracy, AUROC, RMSE, etc.

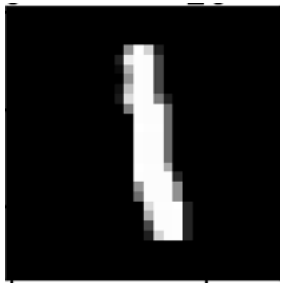
Visualization

Visualizing Latent Representations



0.5
-2.1
-1.8
1.7

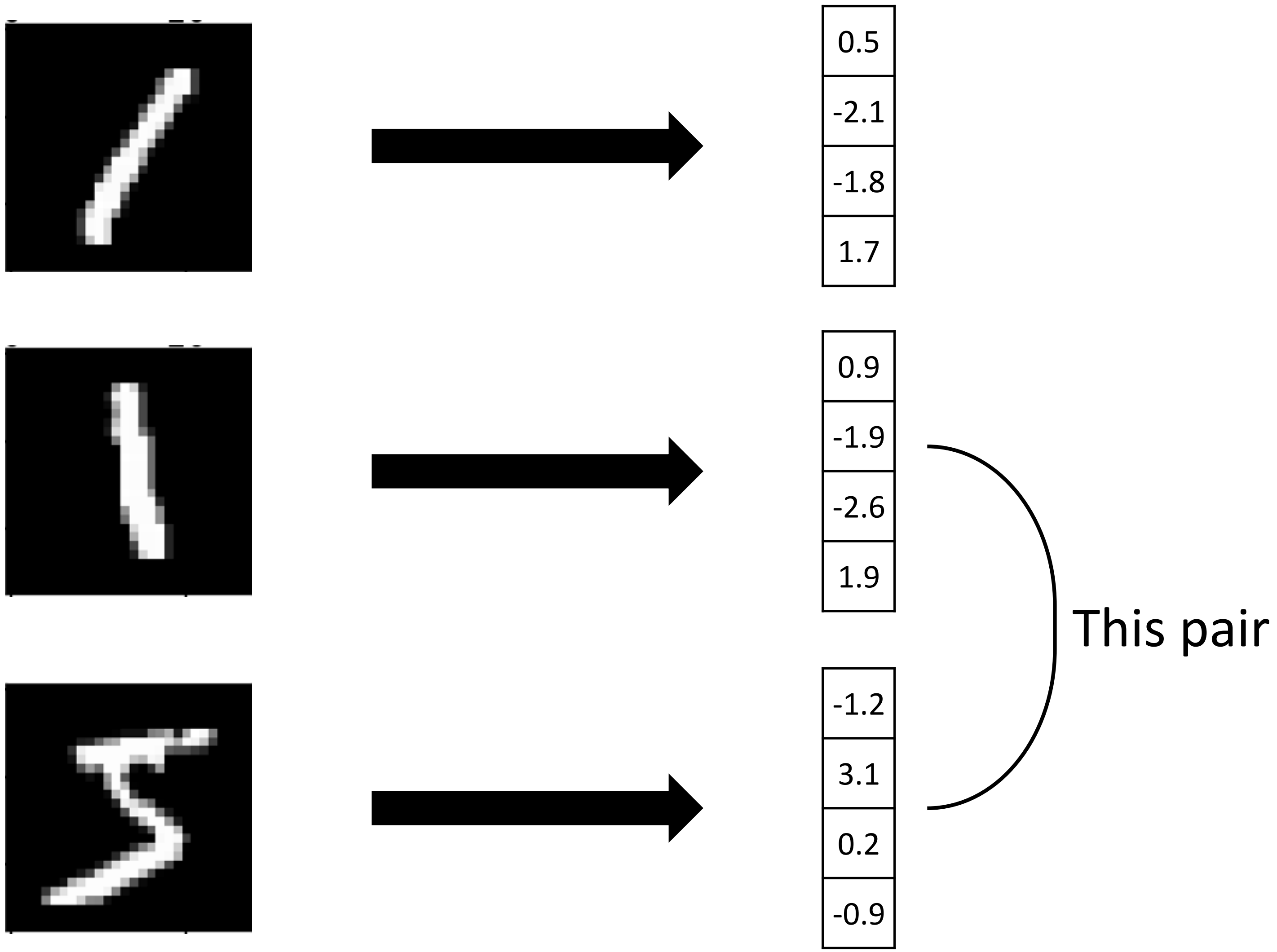
This pair should be closer than



0.9
-1.9
-2.6
1.9

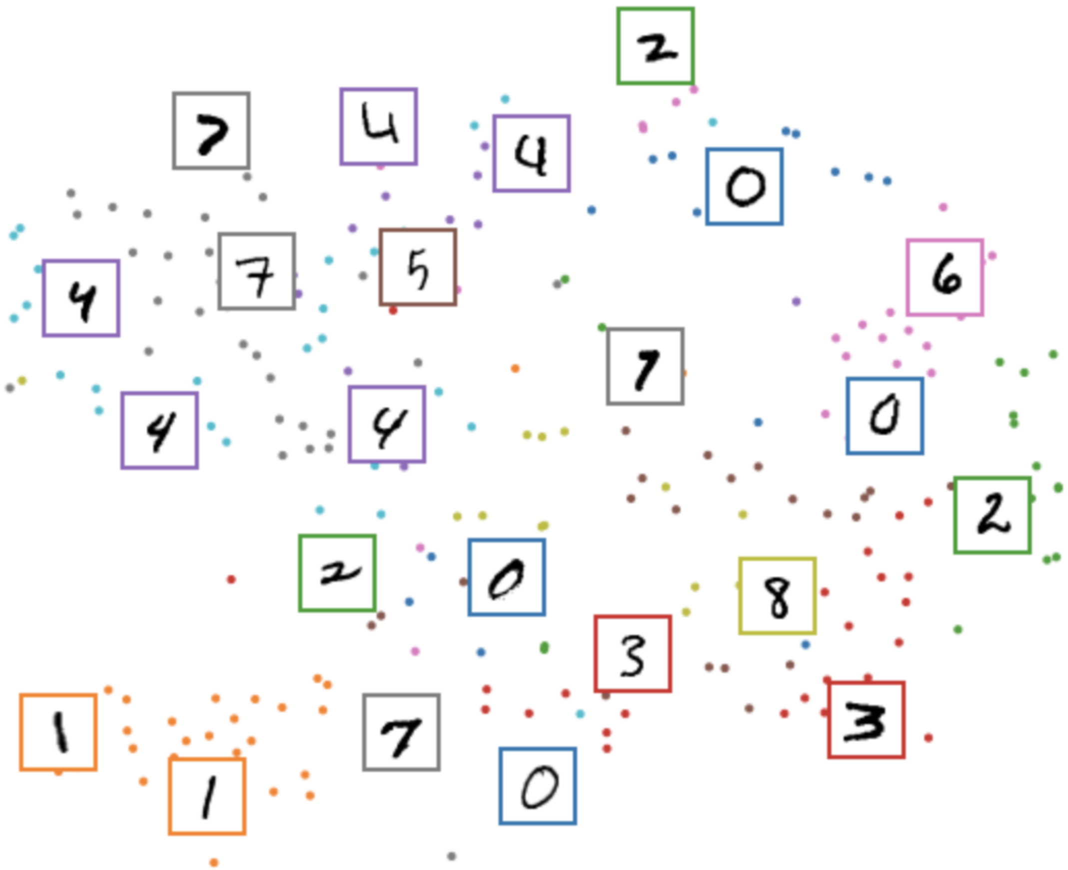
-1.2
3.1
0.2
-0.9

Visualizing Latent Representations



Visualizing Latent Representation

- It’s easier to draw (or scatter) on a 2D plane



Visualizing Latent Representations

- We need an additional dimension reduction
  - From dim($\mathbf{z}$) to 2
  - So that we can plot on a 2D plane.

Visualizing Latent Representations

- We need an additional dimension reduction
  - From dim($\mathbf{z}$) to 2
  - So that we can plot on a 2D plane.
- Popular algorithms
  - PCA
  - t-SNE
  - UMAP

Visualizing Latent Representations

- Popular algorithms
  - PCA
    - Variance-based
    - In scikit-learn
  - t-SNE
    - Distance-based
    - In scikit-learn
  - UMAP
    - Distance-based
    - Needs explicit installation
      - Depends on scikit-learn

Visualizing Latent Representations

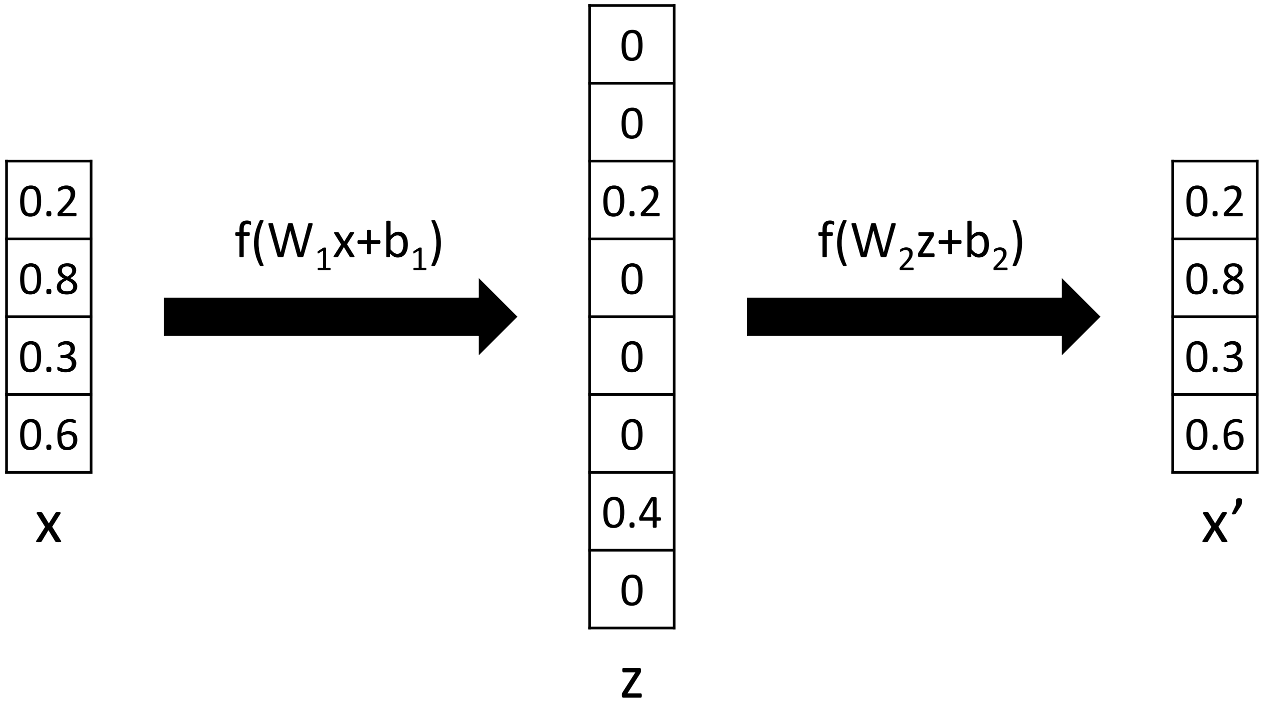
- Popular algorithms
  - PCA
    - Variance-based
    - In scikit-learn
  - t-SNE
    - Distance-based
    - In scikit-learn
  - UMAP
    - Distance-based
    - Needs explicit installation
      - Depends on scikit-learn

See https://pair-code.github.io/understanding-umap/
for a comparison between t-SNE and UMAP

Autoencoder Variants

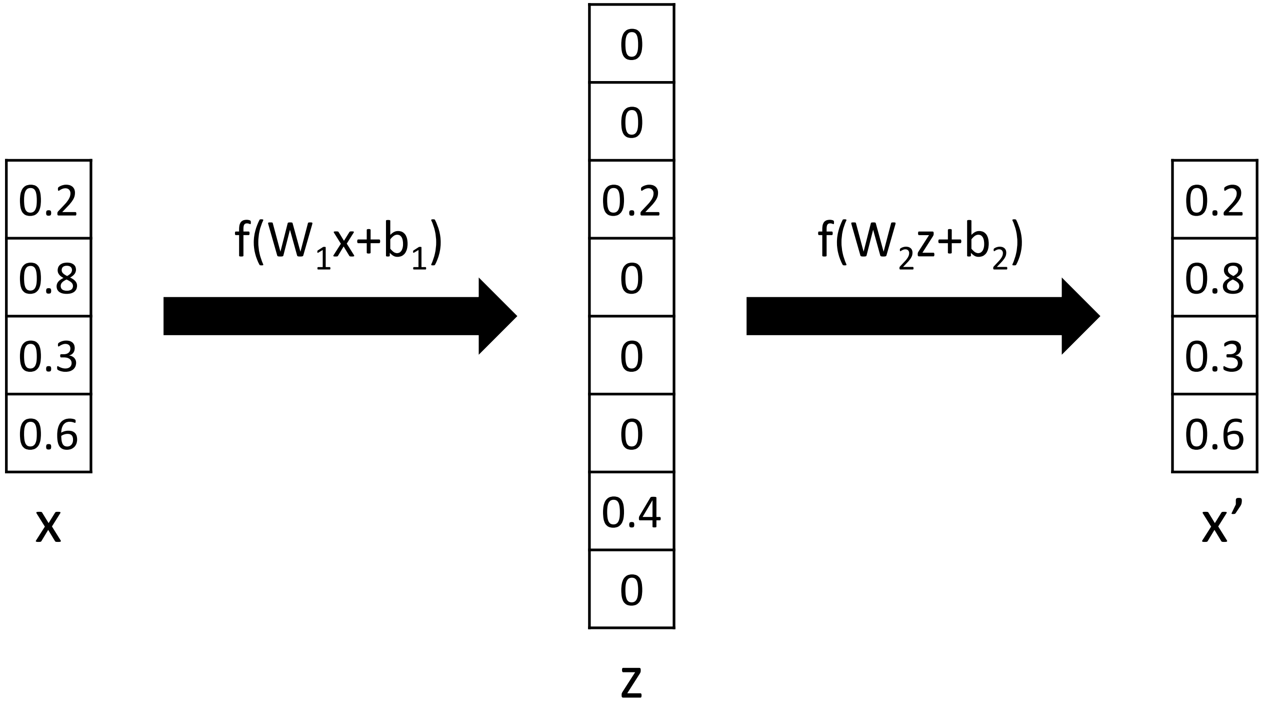
Sparse Autoencoder

- Induce sparse latent representations
  - For better performance in downstream classification



Sparse Autoencoder

- Induce sparse latent representations
  - For better performance in downstream classification

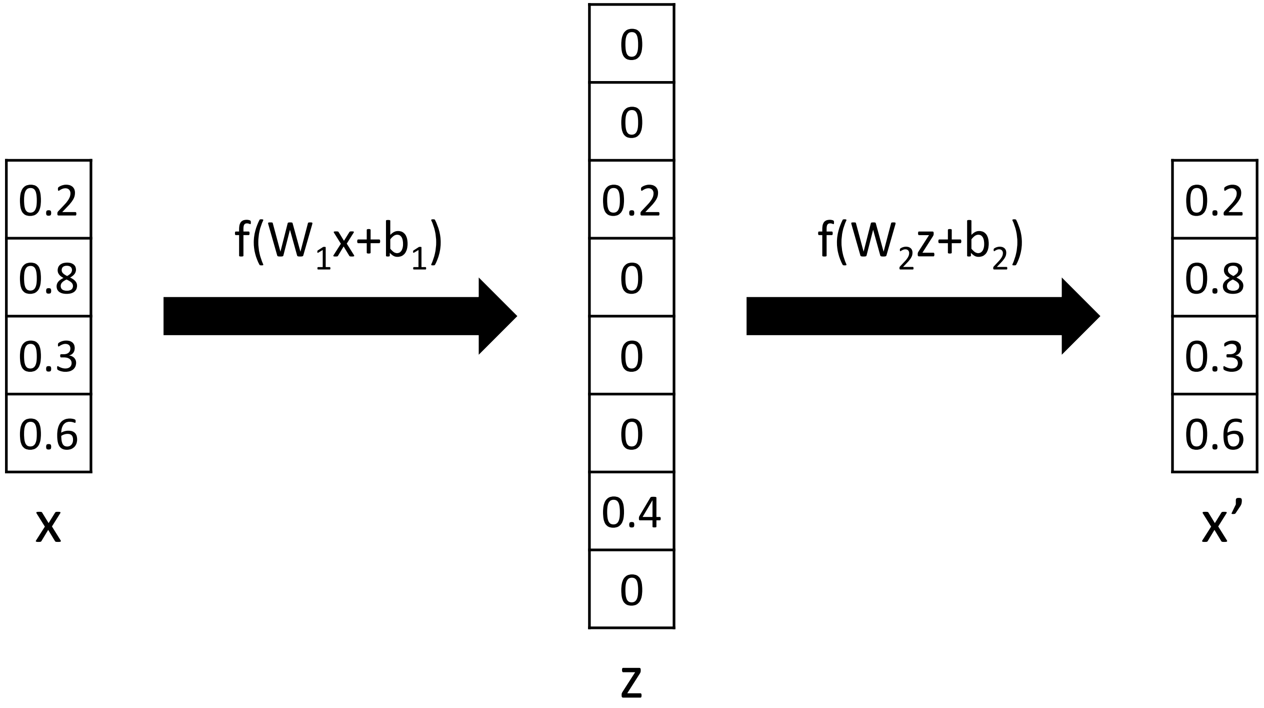


Loss function = $L(\mathbf{x}, \mathbf{x'}) + \Omega(\mathbf{z})$

- Reconstruction $L(\mathbf{x}, \mathbf{x'})$
- Sparsity inducing term $\Omega(\mathbf{z})$

Sparse Autoencoder

- Induce sparse latent representations
  - For better performance in downstream classification



Loss function = $L(\mathbf{x}, \mathbf{x'}) + \Omega(\mathbf{z})$

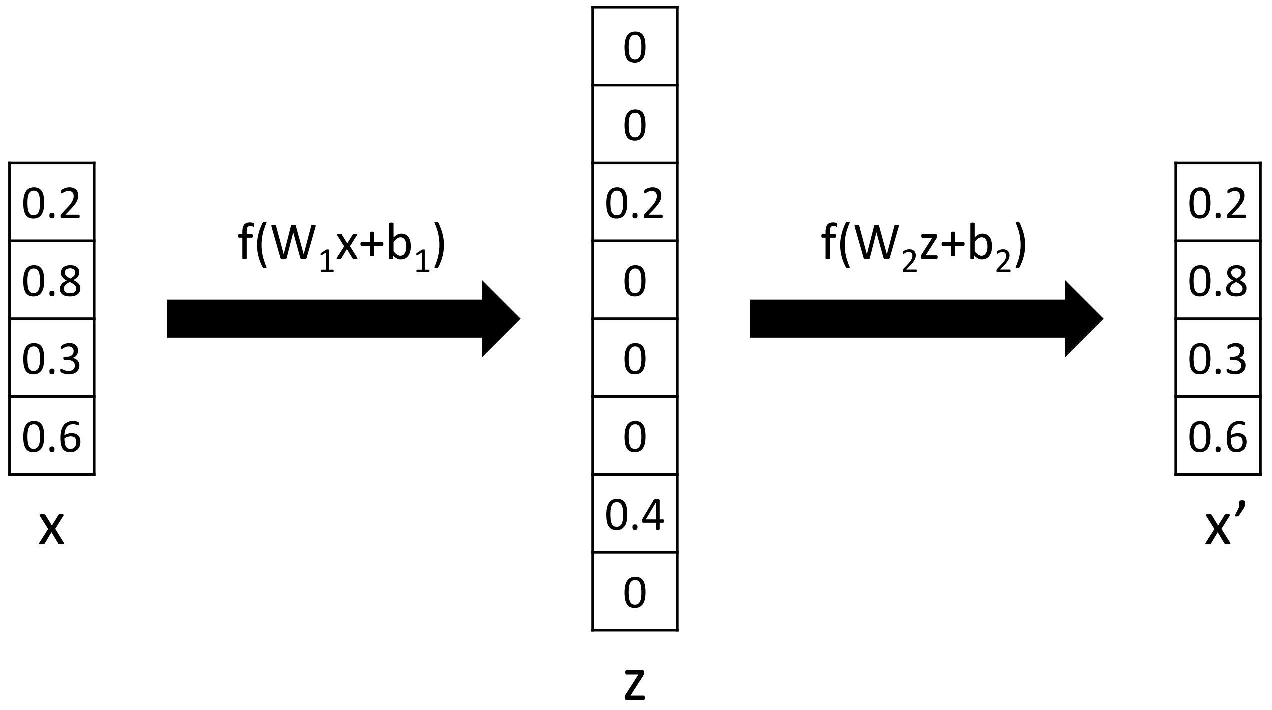
- Reconstruction $L(\mathbf{x}, \mathbf{x'})$
- Sparsity inducing term $\Omega(\mathbf{z})$

Using KL-divergence

- $\Omega(\mathbf{z}) = \sum_i KL(\rho || \hat{\rho}_i)$
- $\hat{\rho}_i$ = Average activation of z_j
- ρ = Target Bernoulli dist. mean

Sparse Autoencoder

- Induce sparse latent representations
  - For better performance in downstream classification



Loss function = $L(\mathbf{x}, \mathbf{x'}) + \Omega(\mathbf{z})$

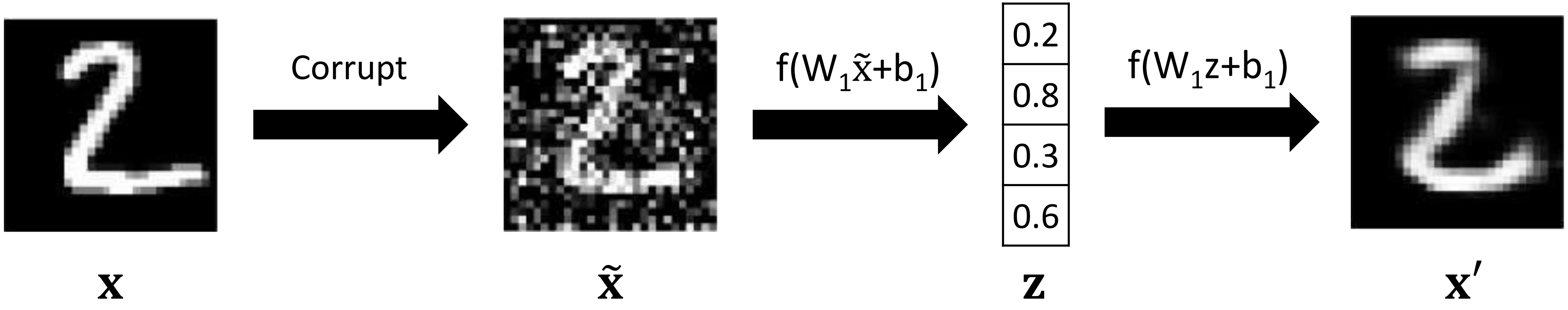
- Reconstruction $L(\mathbf{x}, \mathbf{x'})$
- Sparsity inducing term $\Omega(\mathbf{z})$

Using L_1 Regularization

- $\Omega(\mathbf{z}) = \lambda \sum_i |z_i|$

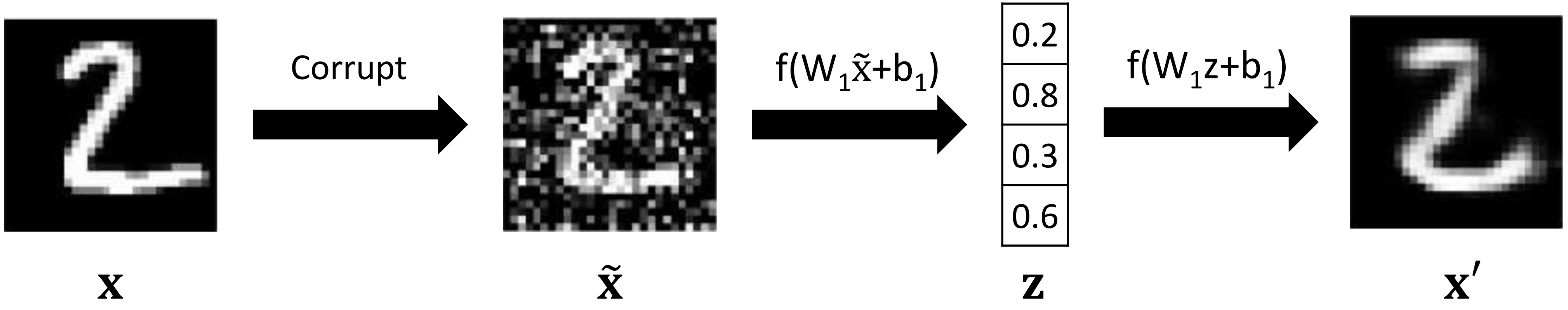
Denoising Autoencoder

- Reconstruct an input with random noise (i.e. corrupted input)
  - Small noise doesn’t affect higher-level representation (i.e. raw data)
  - Autoencoder must learn useful patterns to perform denoising



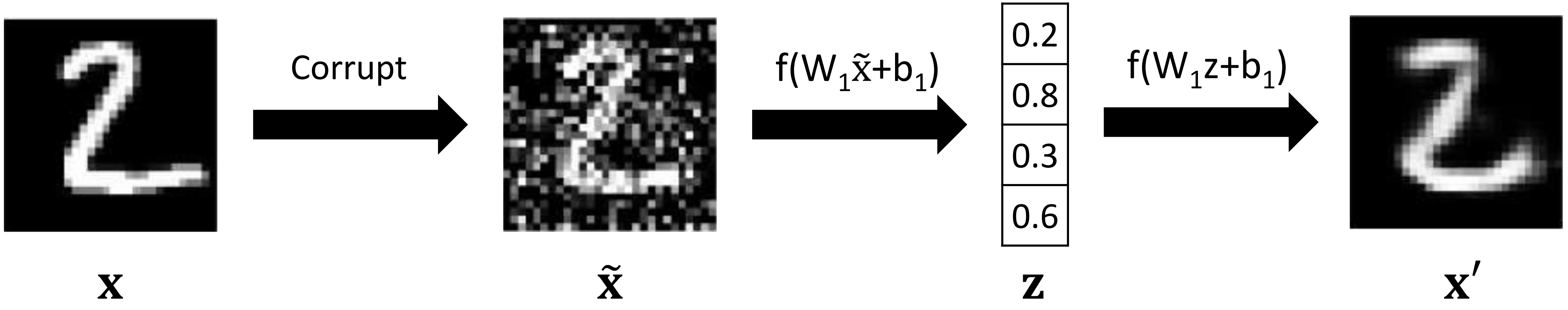
Denoising Autoencoder

- Reconstruct an input with random noise (i.e. corrupted input)
  - Small noise doesn't affect higher-level representation (i.e. raw data)
    - Reminds you of something?
  - Autoencoder must learn useful patterns to perform denoising



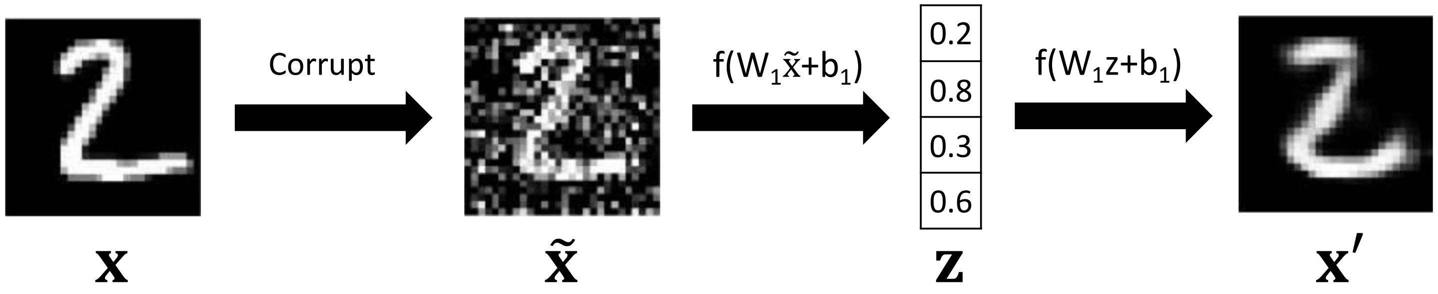
Denoising Autoencoder

- Reconstruct an input with random noise (i.e. corrupted input)
  - Small noise doesn't affect higher-level representation (i.e. raw data)
    - Reminds you of something? ➔ PCA
  - Autoencoder must learn useful patterns to perform denoising



Denoising Autoencoder

- Reconstruct an input with random noise (i.e. corrupted input)
  - Small noise doesn't affect higher-level representation (i.e. raw data)
  - Autoencoder must learn useful patterns to perform denoising



Loss function = $L(\mathbf{x}, \mathbf{x}')$

- NOT $L(\tilde{\mathbf{x}}, \mathbf{x}')$

Corrupt

- Usually Gaussian noise

AI504: Programming for Artificial Intelligence

Week 4: Autoencoders

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr