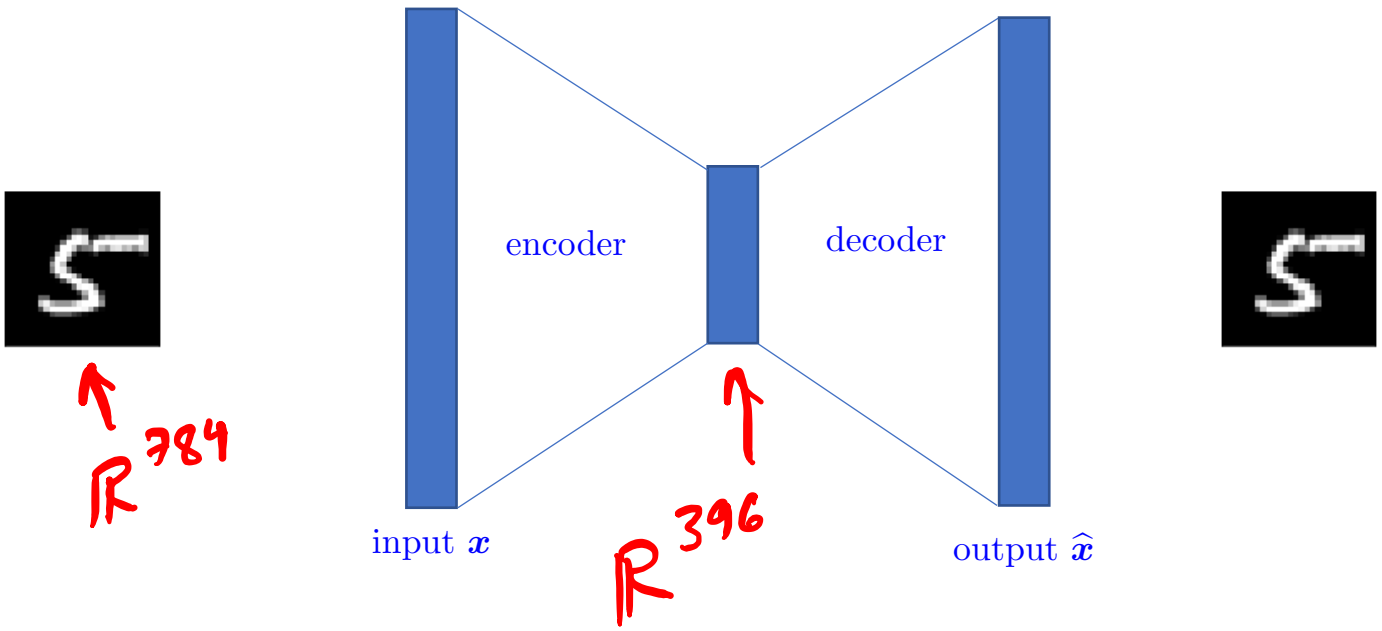
Neural Networks, Autoencoders

- neural network basics
- autoencoders

Autoencoders

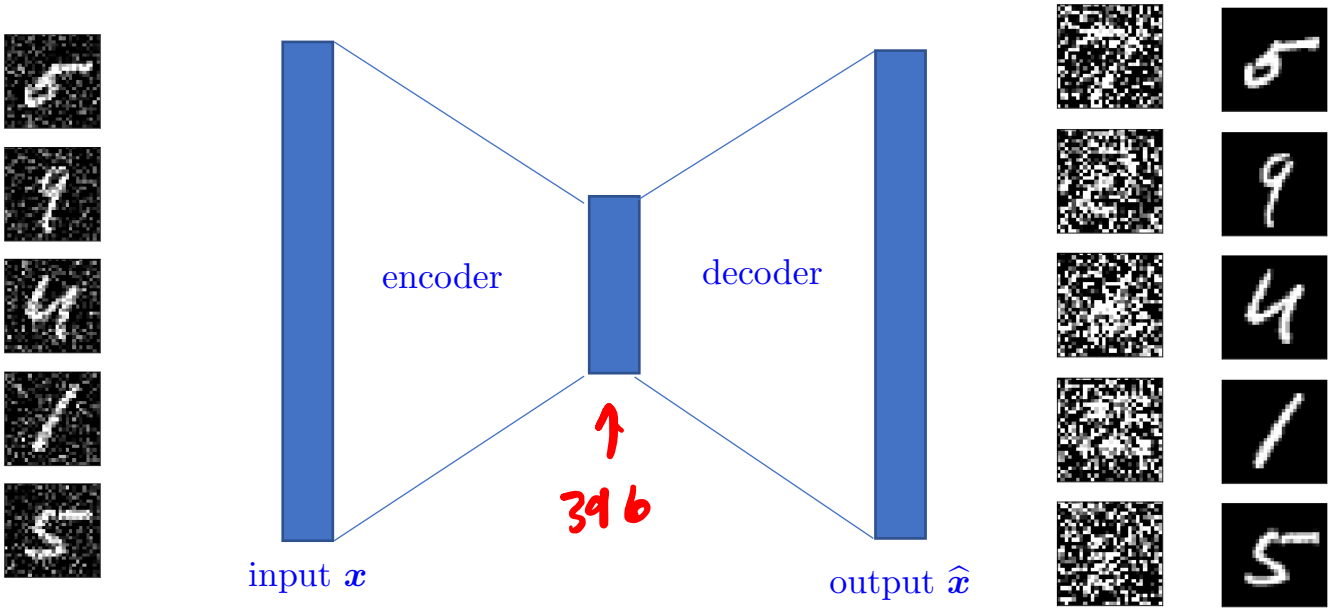


- main applications: denoising, data visualization, manifold learning

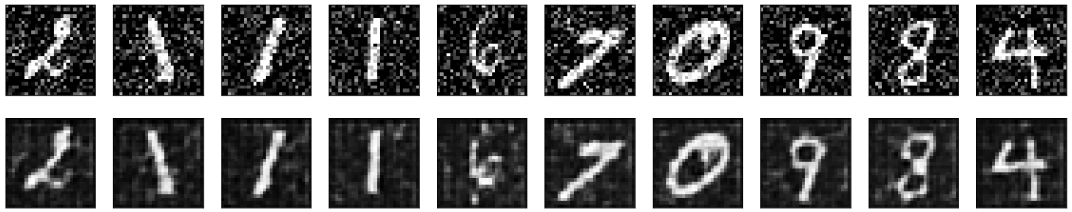
- autoencoders are (in general) **not** better than engineered compression algorithms

De-noising Autoencoder

$$\mathcal{D} = \{(\; , \;)\}$$



Test



Multinomial Logistic Regression, Single Layer

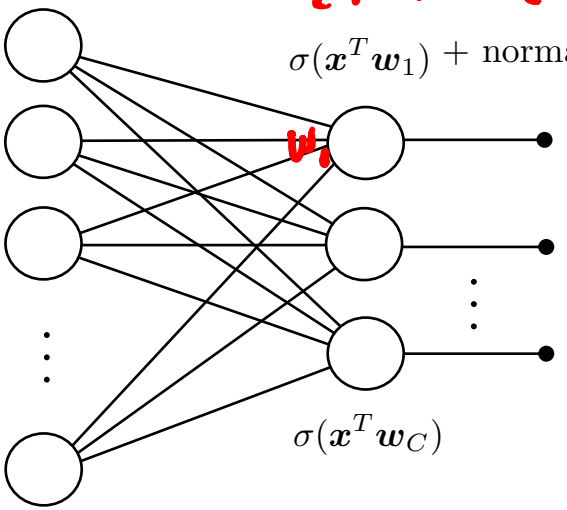
- logistic regression is a single *single layer of artificial neural network*

multiple neurons, single layer: $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{W}^T \boldsymbol{x})$

$W = [w_1, w_2, \cdots]$

$\boldsymbol{x}_i =$ 

$\boldsymbol{y}_i = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$



$\widehat{\boldsymbol{y}}_i = \begin{bmatrix} 0.92 \\ 0.01 \\ \vdots \\ 0.03 \end{bmatrix}$

- loss

$$\ell(\widehat{\boldsymbol{y}}_i, \boldsymbol{y}_i) = D(\boldsymbol{y}_i || \widehat{\boldsymbol{y}}_i)$$

$$\min_W \sum_{i=1}^{n} \ell(\hat{y}_i, y)$$

- training: gradient descent/Newton's method
- objective is convex

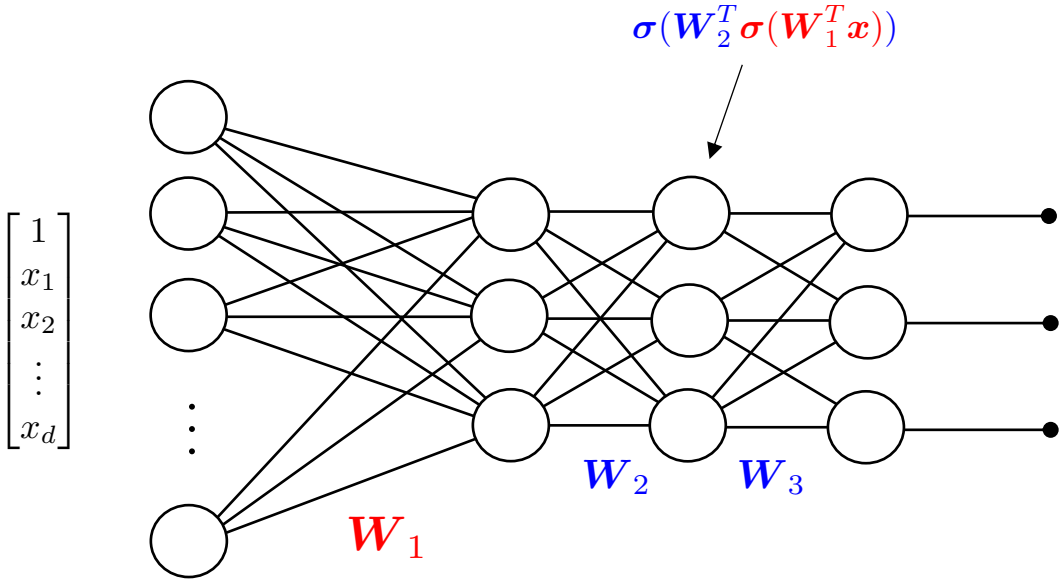
Artificial Neural Networks and Logistic Regression

- binary logistic regression:
  - single *artificial neuron*
- multiclass logistic regression:
  - single layer *artificial neural network*
- *feedforward* neural network:

single layer: $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{W}^T \boldsymbol{x})$

two layers: $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{W}_2^T \boldsymbol{\sigma}(\boldsymbol{W}_1^T \boldsymbol{x}))$

ℓ-layers: $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{W}_\ell^T \dots \boldsymbol{\sigma}(\boldsymbol{W}_2^T \boldsymbol{\sigma}(\boldsymbol{W}_1^T \boldsymbol{x})) \dots)$



- softmax:

$$[\boldsymbol{\sigma}(\boldsymbol{z})]_i = \frac{e_i^z}{\sum_j e^{z_j}}$$

$$\boldsymbol{\sigma}(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\sum_j e^{z_j}}$$

- logistic (unnormalized):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Training

- output

ℓ-layers: $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{W}_\ell^T \dots \boldsymbol{\sigma}(\boldsymbol{W}_2^T \boldsymbol{\sigma}(\boldsymbol{W}_1^T \boldsymbol{x})) \dots)$

- objective

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2, \dots} \sum_i \ell(\widehat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$$

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2, \dots} \sum_i D(\boldsymbol{y}_i || \boldsymbol{\sigma}(\boldsymbol{W}_\ell^T \dots \boldsymbol{\sigma}(\boldsymbol{W}_2^T \boldsymbol{\sigma}(\boldsymbol{W}_1^T \boldsymbol{x}_i)) \dots))$$

- no longer convex
- we can still compute the gradient and head downhill
- stochastic gradient descent:

$\sum_{i=1}^{100,000} \ell(y_i, \hat{y}_i)$ → 1) $\sum_{i=1}^{100} \ell(y_i, \hat{y}_i)$ 2) $\sum_{i=101}^{200} \ell(y_i, \hat{y}_i)$ ...

Training

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

- output

  m-layers: $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{W}_m^T \dots \boldsymbol{\sigma}(\boldsymbol{W}_2^T \boldsymbol{\sigma}(\boldsymbol{W}_1^T \boldsymbol{x})) \dots)$

- objective

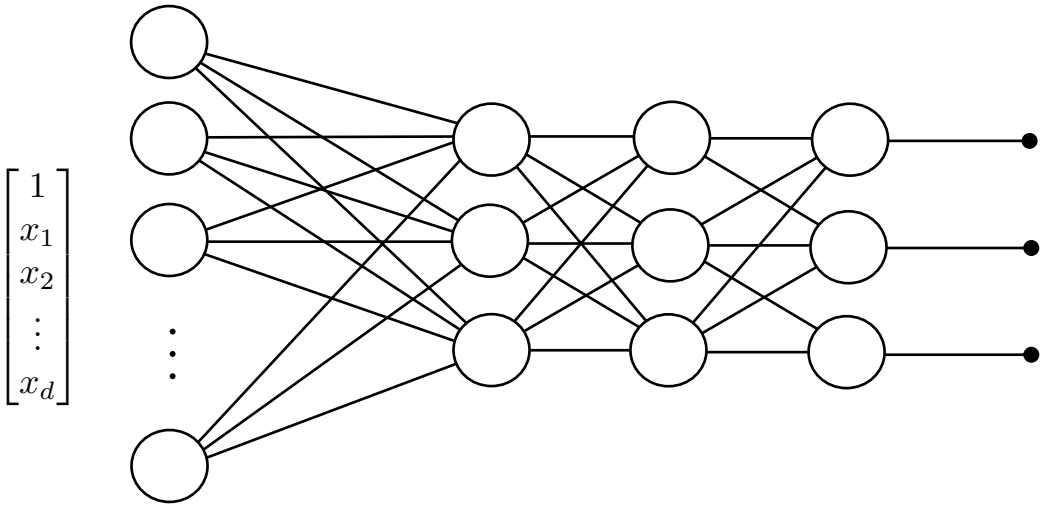
$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2, \dots} \sum_i \ell(\widehat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$$

- *backpropagation* - smart way to compute gradient

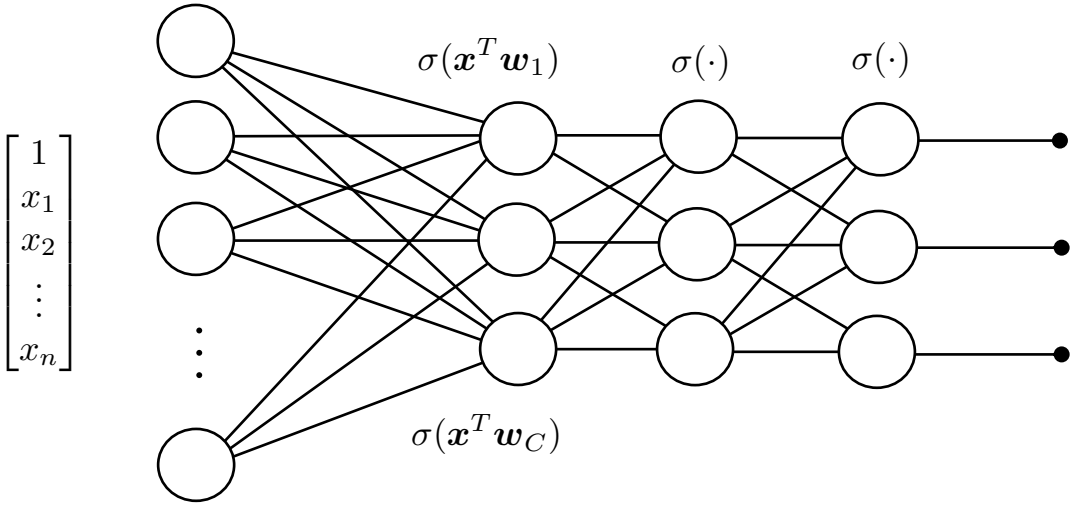
  1) run $\boldsymbol{x}_i$ through network

  2) compute gradient with respect to weights in last layer

  $\vdots$



Activation Functions, Loss functions



- activation functions
  - logistic
  - ReLU
  - linear
  - ...
- loss functions
  - cross entropy
  - KL divergence
  - squared error
  - ...

▻ Available activations

- `relu` function
- `sigmoid` function
- `softmax` function
- `softplus` function
- `softsign` function
- `tanh` function
- `selu` function
- `elu` function
- `exponential` function

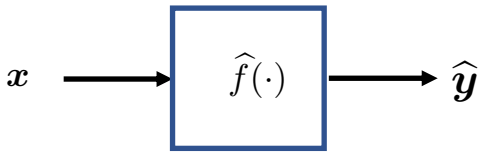
Probabilistic losses

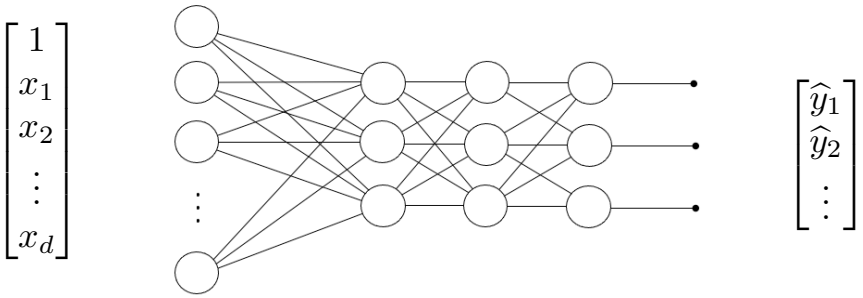
- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

Function Approximation





- universal approximation theorem

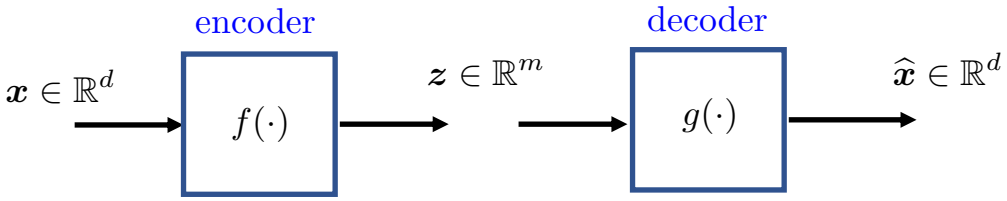
Consider a single layer neural network: $\widehat{f}(\boldsymbol{x}) = \boldsymbol{W}_2^T \boldsymbol{\sigma}(\boldsymbol{W}_1^T \boldsymbol{x})$.

For every continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$ there exists a set of weights $\boldsymbol{W}_1, \boldsymbol{W}_2$ *(for some arbitrarily wide layer)* such that

$$\sup_{\boldsymbol{x}} ||f(\boldsymbol{x}) - \widehat{f}(\boldsymbol{x})|| \leq \epsilon$$

for any $\epsilon > 0$.

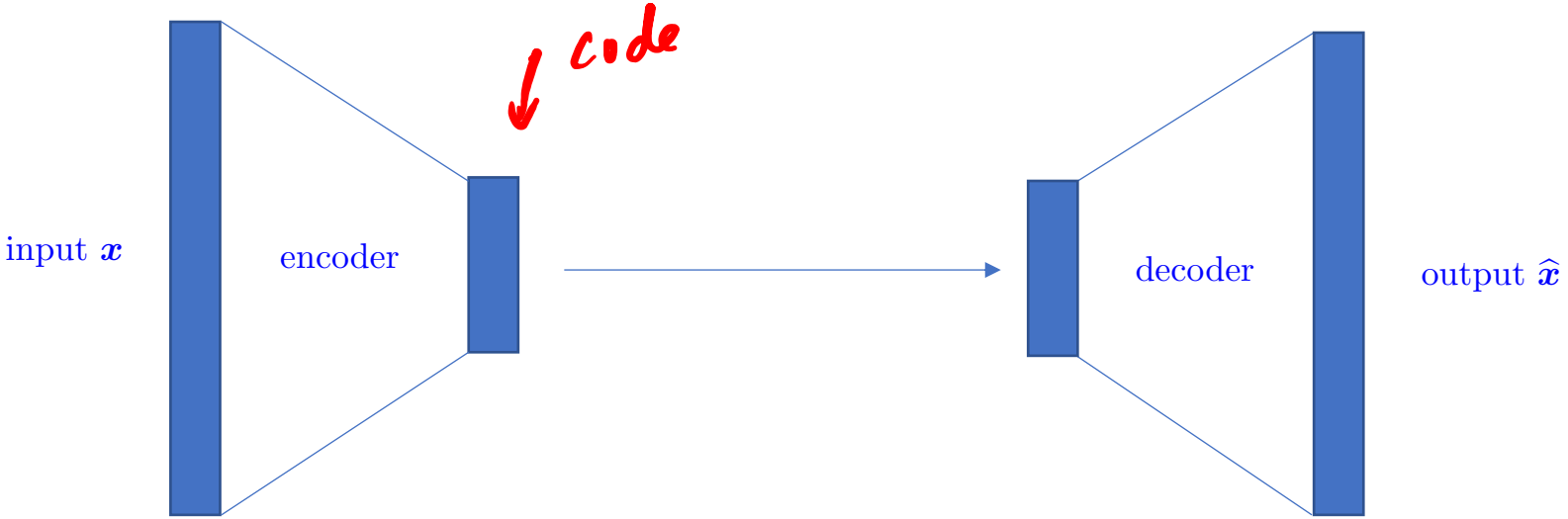
Autoencoders



- basic idea: output should equal input: $g(f(\boldsymbol{x})) \approx \boldsymbol{x}$
- *dimensionality reduction*
- *undercomplete* auto-encoder

$m < d$

code

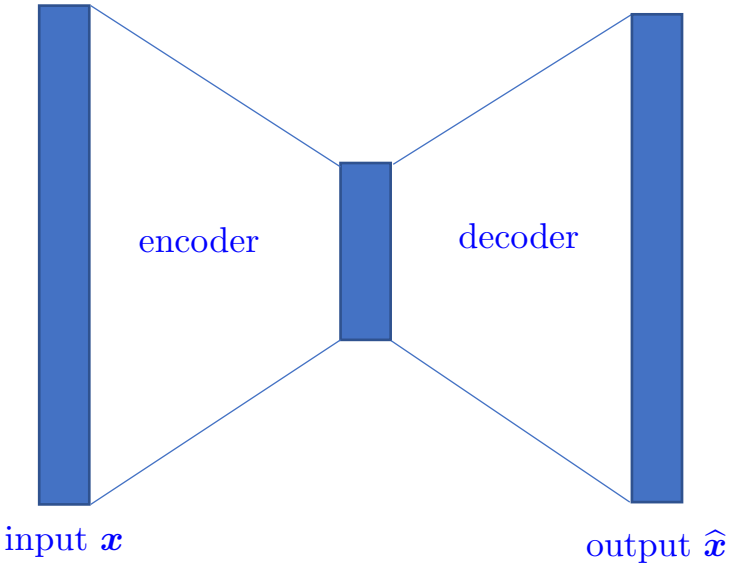


Autoencoders

- autoencoders are lossy

- autoencoders are specific to the types of examples they are trained on

- trained from data as opposed to engineered by a human



- autoencoders are (in general) **not** better than engineered compression algorithms

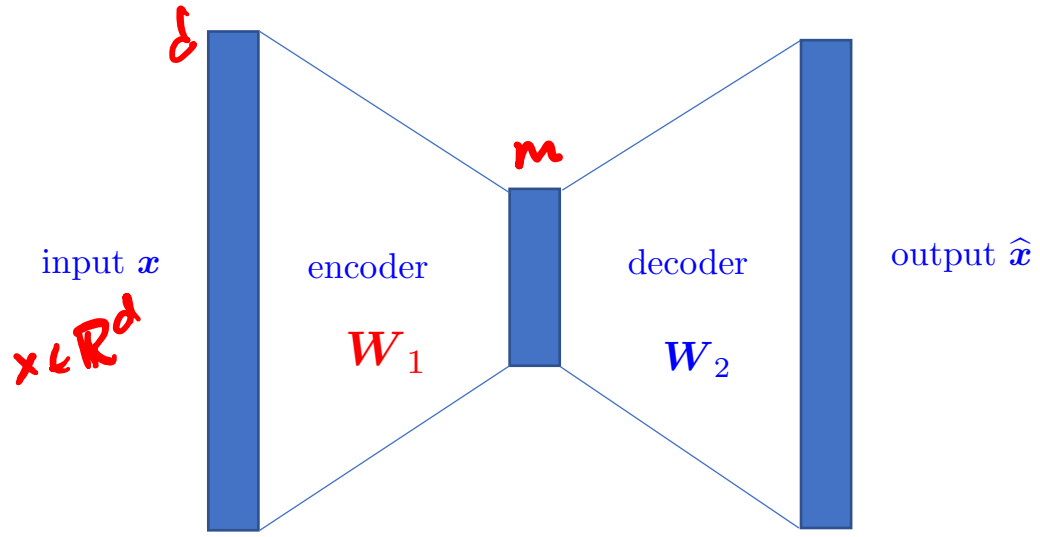
- main applications: denoising, data visualization, manifold learning

Autoencoders

- training undercomplete autoencoders:

$$\min \sum_i \ell(\boldsymbol{x}_i, \widehat{\boldsymbol{x}}_i)$$

- use SGD and backpropagation



- example: linear activation functions + mean square error loss

$$\widehat{\boldsymbol{x}} = \boldsymbol{\sigma}(\boldsymbol{W}_2^T \boldsymbol{\sigma}(\boldsymbol{W}_1^T \boldsymbol{x}))$$

$$\widehat{\boldsymbol{x}} = \boldsymbol{W}_2^T \boldsymbol{W}_1^T \boldsymbol{x}$$

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2} \sum_i \ell(\boldsymbol{x}_i, \widehat{\boldsymbol{x}}_i) = \min_{\boldsymbol{W}_1, \boldsymbol{W}_2} \sum_i ||\boldsymbol{x}_i - \boldsymbol{W}_2^T \boldsymbol{W}_1^T \boldsymbol{x}_i||_2^2$$

$$= \min_{\boldsymbol{W}_1, \boldsymbol{W}_2} ||\boldsymbol{X} - \boldsymbol{W}_2^T \boldsymbol{W}_1^T \boldsymbol{X}||_F^2$$

$$= \min_{\boldsymbol{A}: \text{rank}(\boldsymbol{A}) \le m} ||\boldsymbol{X} - \boldsymbol{A}||_F^2$$

$$\boldsymbol{A} = \sum_{i=1}^{m} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \quad \text{PCA}$$

Autoencoders

- undercomplete autoencoders
- regularized autoencoders
- sparse autoencoders

$$\min \sum_i \ell(\boldsymbol{x}_i, \widehat{\boldsymbol{x}}_i) + \lambda ||\boldsymbol{z}||_1$$

- de-noising autoencoders

input $\boldsymbol{x}$ + noise