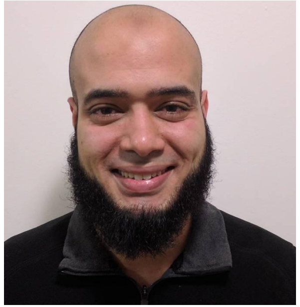
Machine Learning
AutoEncoder

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Dimensionality Reduction

- **Dimensionality reduction** is the process of reducing the number of features to a smaller set of features
- Main reasons
  - **Noise Reduction**: Eliminating less important features
  - **Speeding up models**: Faster with fewer input features (dimensions).
  - **Data Visualization**: visualize data in 2d / 3d
  - **Storage efficiency**: Less dimensional data requires less storage space.
  - Avoiding the Curse of Dimensionality: Higher-dimensional data may lead to overfitting

Dimensionality Reduction: Techniques

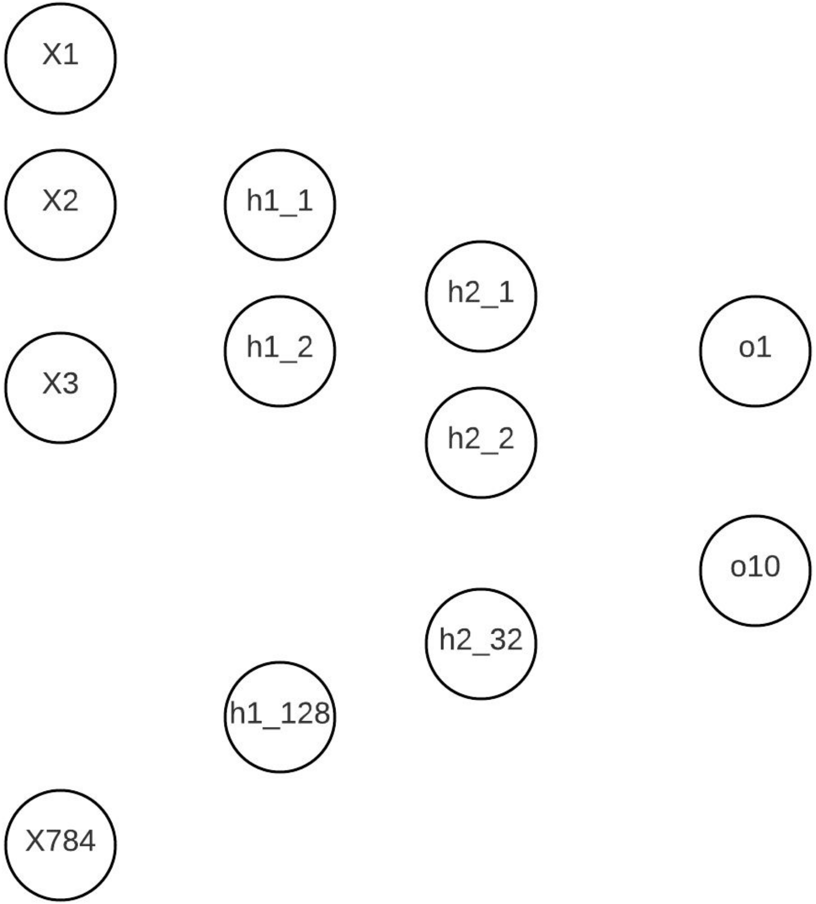
- Principal Component Analysis (PCA)
  - Reduce but retain as much of the **original variance** as possible
  - Keywords: Covariance Matrix, Eigen Decomposition, Eigenvalues, Principal Components
  - Limitations of PCA: **Linearity** - assumes that the data's principal components are a linear combination of the original features
- t-Distributed Stochastic Neighbor Embedding (**t-SNE**)
  - Reduces dimensionality while trying to keep similar instances close and dissimilar apart
    - For 2D and 3D visualization
- Autoencoders
  - Neural Network: a non-linear reduction technique

AutoEncoder

- An AutoEncoder is a popular neural network structure that plays role in deep learning
    - Similar to PCA, but are more flexible and can capture **non-linear** transformations
        - It can be used like other techniques to reduce the number of features
    - Feature learning (representation): The encoded representations can be **used as features** for other machine learning tasks. Due to DL nature, this can be stronger representation
- Part of many DNN work
    - Variational AutoEncoders (VAEs) are probabilistic AutoEncoder
    - Self-supervised networks, e.g. image inpainting
    - Semantic Segmentation
    - Transformers (e.g. used in LLMs) use the encoder

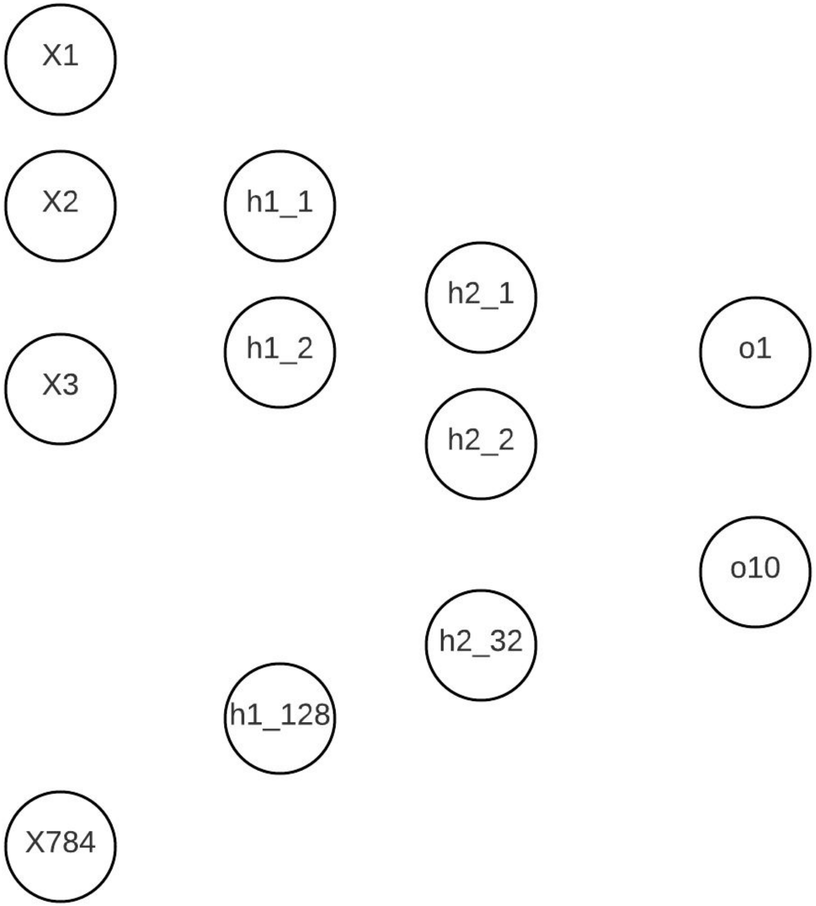
Recall: Classification

- Assume you want to classify MNIST images to 10 classes
  - Formulate input image 28x28 into 784 features
  - Learn NN that ends with 10 output nodes with softmax
    - Intermediate layers decrease toward 10 from 784



Encoder

- In this network, we have 3 generated layers: H1, H2, O1
- Each layer is a new representation for the input X
  - All layers are **optimized toward** the ability to classify X
- These intermediate representations as **encoded** (latent) versions of the input

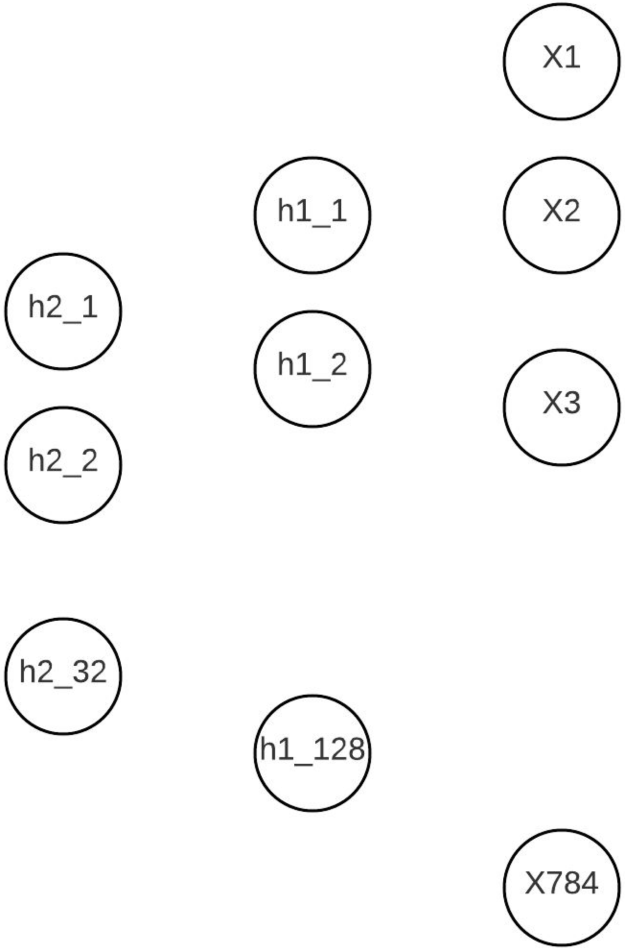


Encoder

- Latent representation (space/variables): lower-dimensional space representing the data and captures the most important features
    - We can use as a compressed version
    - GANs and VAEs use latent spaces to generate new images, often by sampling random points in the latent space and then decoding these points into images.
- The encoder is a part of a neural network that **compresses (encodes)** the input features into a compact *latent* representation

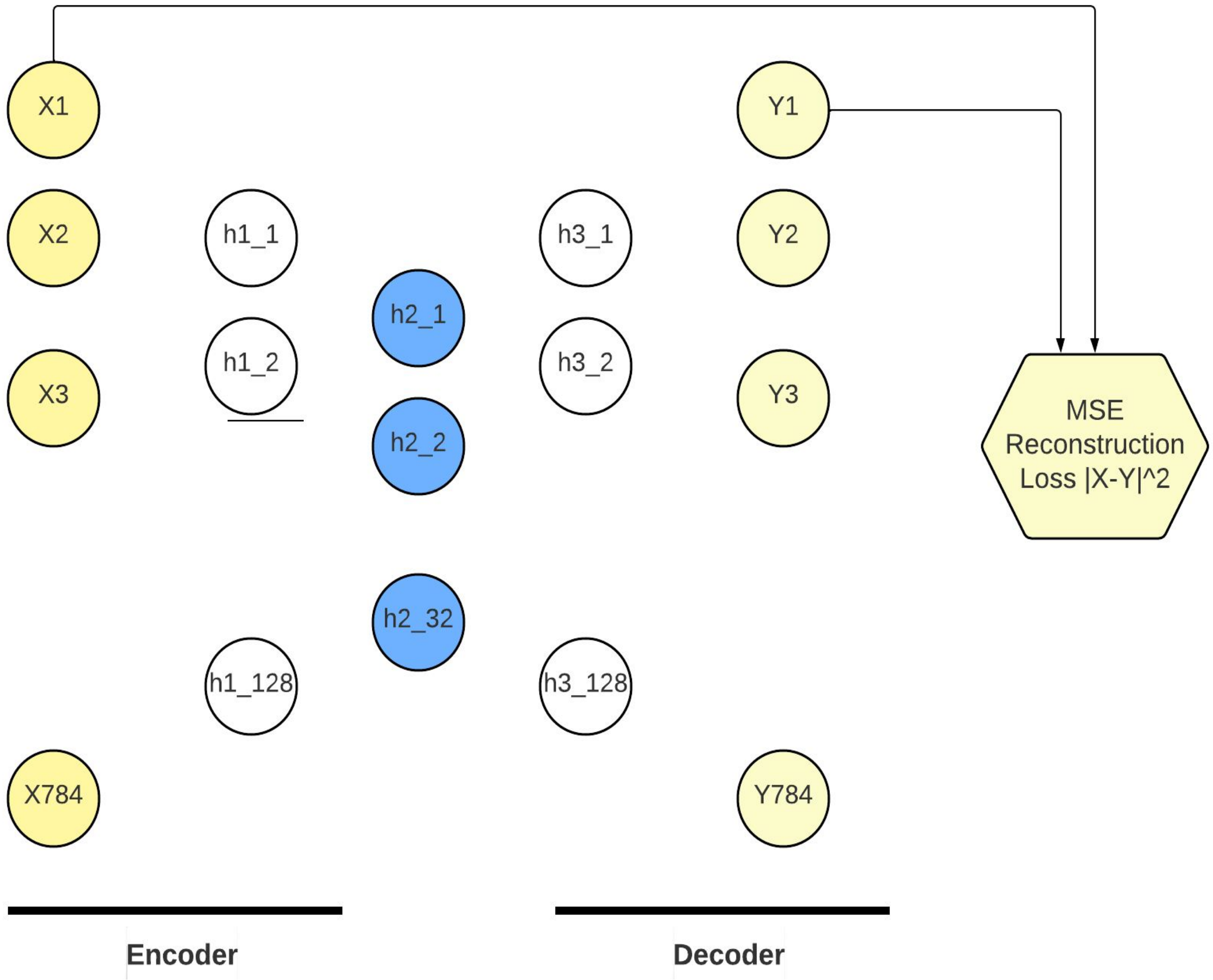
Decoder

- Decoder is the **opposite** of the encoder: it reconstructs the original data from its compressed latent representation
- Starting from H2, we will convert back it to X1
- But for this to happen, H2 needs to already have all important features so that we can guide the network
  - Imagine encoder is like sum
  - Sum(1, 2, 3, 4, 5) ⇒ 15
  - Decoder (15) ⇒ We can't get (1, 2, 3, 4, 5)



AutoEncoder

- It consists of 3 parts: encoder, decoder and reconstruction loss
- An **encoder**:
  - Input is X
  - Generates 1 or more hidden layers of **decreasing size**
  - The last hidden layer we call it a latent representation
- A **decoder**
  - Input: latent representation from the encoder
  - Generates 1 or more hidden layers of **increasing size**
  - The last layer matches the input size.
- Goal: the network output is very close to input
  - We can train with a simple **loss** like MSE (input, output)
  - If we trained it, then the latent representation is a company representation for the input


X1
X2
X3
X784
h1_1
h1_2
h1_128
h2_1
h2_2
h2_32
h3_1
h3_2
h3_128
Y1
Y2
Y3
Y784
MSE Reconstruction Loss |X-Y|^2
Encoder
Decoder

MNIST

- Let’s create a simple neural network
  - Input is 748 value
  - Intermediate a **single** hidden layer: We will explore sizes like 16, 32, 64 or 128
  - The output again is 748 values
- MSE is the loss function (no new code!_

```
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Encoder compresses data into 32 dimensions from 784 dimensions
autoencoder = MLPRegressor(hidden_layer_sizes=(32,), activation='relu',
                           solver='adam', max_iter=5000, random_state=42)


autoencoder.fit(X_train, X_train)   # input = output
X_test_reconstructed = autoencoder.predict(X_test)
mse = mean_squared_error(X_test, X_test_reconstructed)
print(f"Mean Squared Error on test data: {mse:.2f}")
```

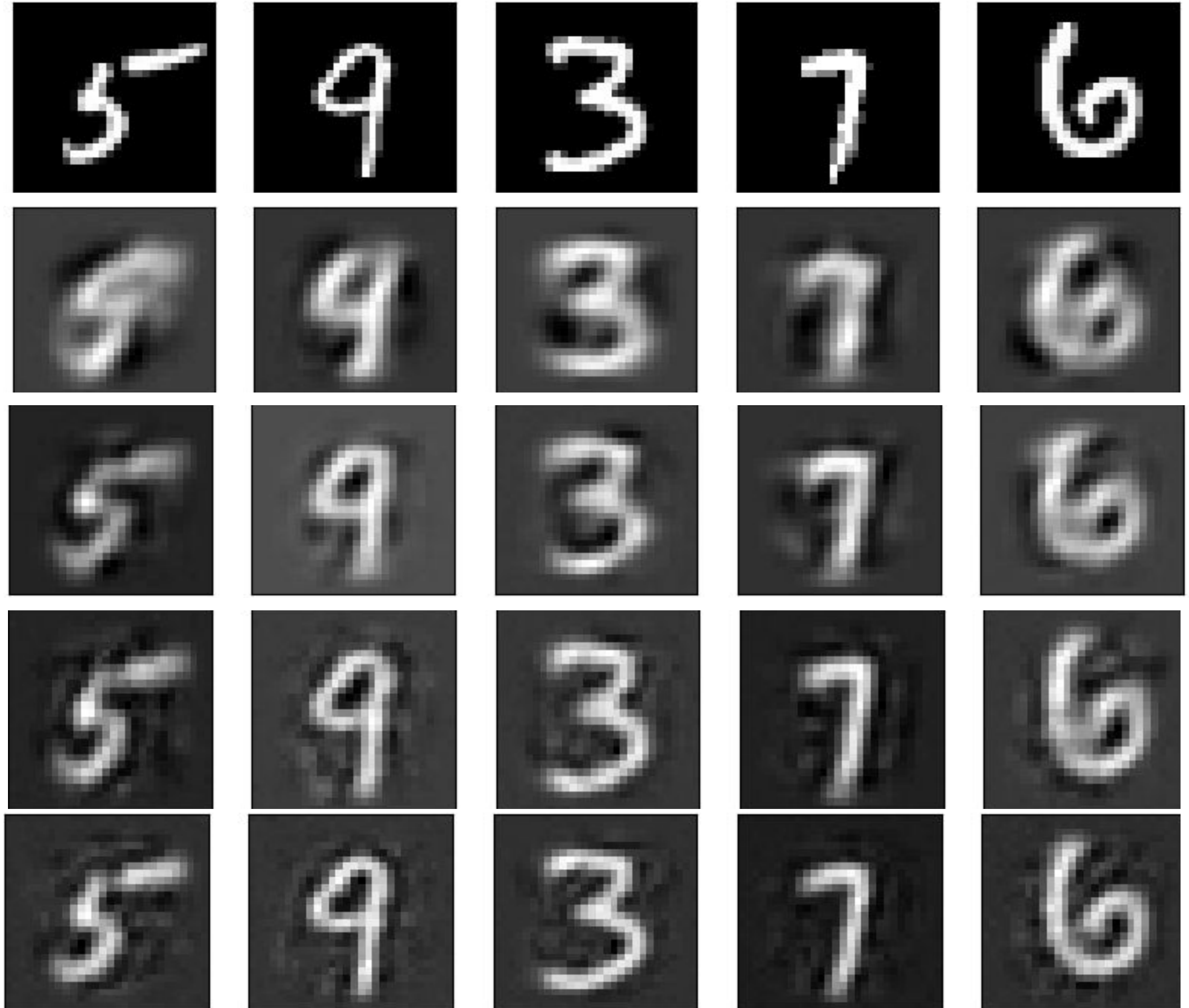
748 (original) = 28 x 28

748 ⇒ 16

748 ⇒ 32

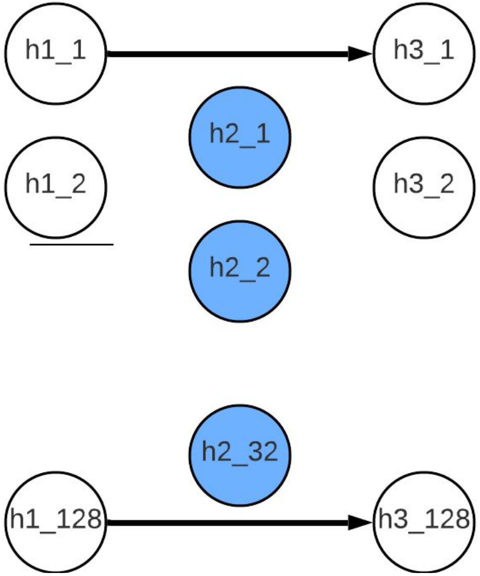
748 ⇒ 64

748 ⇒ 128



Skip Connections

- Skip (shortcut/residual) connections can connect some encoder node with the some decoder node (typically corresponding ones)
  - Address the **vanishing gradient** problem to train **very deep** deeper neural networks.
- **Vanishing Gradient Problem**: In very deep networks, gradients can become extremely small during backpropagation (no weight updates). Skip connections allows gradients to **flow** better



Misc

- The smaller the latent representation, the harder to summarize the key features from the input
  - So, we find a compact size that has a good MSE
- In DNN, we gradually shrink the encoder, then the decoder is a mirror

Denoising Autoencoder

- Imagine if you expect your input to have added noise
  - Or for example some input **features are missing**
- How can we change the autoencoder to deal with that?

- Simply train the network with noisy inputs (or drop elements)
- Now, train the network to reconstruct the right input
  - For example, cancel the noise or predict the missing elements
- This nice variant is called denoising autoencoder
  - For missing elements, you can drop with constant % or varying % based on the expectations

Principal Component Analysis (PCA)

- Similar to autoencoder, PCA transforms the input features into a new set of uncorrelated features known as **principal components**
  - **Ordered** with features with most of the variability in the original features
  - Principal components are **orthogonal** to each other
  - It steps involve Features Covariance Matrix, Eigenvectors and Eigenvalues
- Usage
  - Dimensionality reduction / Noise reduction / Data compression / visualization
- Cons
  - Assumes that the principal components are a **linear combination** of the original features
  - Selects features based on the **highest variance** (which may not the important ones)
  - Difficulty in **Interpretation**
  - Instability: **Small** changes in data can sometimes result in **different** principal components

SKlearn

- PCA is an old ML topics with interesting insights and Math
    - Study in the future
- PCA is very common step for many people to simplify data / visualize them, especially in unsupervised setup
    - Easy to run and get results
- **tSNE** is another common tool for **visualization** (we met before in code)

```
from sklearn.decomposition import PCA
pca = PCA(n_components=4)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

SKlearn (ChatGPT)

- Seems SKlearn has other compression algorithms!
  - Linear Discriminant Analysis (**LDA**): A supervised method that aims to maximize the separation between multiple classes.
    - from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
  - Factor Analysis (**FA**): Used to model observed variables and their underlying structure.
  - Independent Component Analysis (**ICA**): Find a linear representation of non-Gaussian data.
  - **Isomap**: Uses geodesic distance to preserve the manifold structure in the reduced dimensions.
  - Multidimensional Scaling (**MDS**): Reduces dimensions while trying to preserve the distances between instances.
  - Locally Linear Embedding (**LLE**): Focuses on preserving local neighborhood relationships.
  - **Sparse PCA**: A variation of PCA that introduces sparsity constraints on the components.
  - **Gaussian Random Projection**: Reduces dimensionality by projecting the original data into a randomly generated subspace.

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."