VIRGINIA TECH

Deep Learning and Autoencoders

Lifu Huang

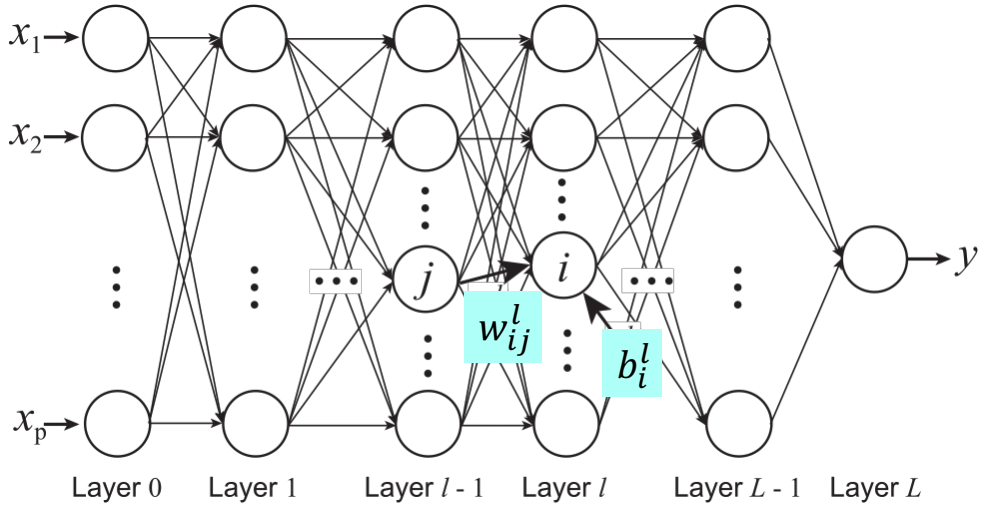
lifuh@vt.edu

Torgersen Hall, Suite 3160E

Three key ideas

- (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations
- End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to extract features
- Distributed Representations
  - No single neuron encodes everything
  - Groups of neurons work together

Multi Layer Perceptron (MLP)



$$a_i^l = f(z_i^l) = f\Big(\sum_j w_{ij}^l a_j^{l-1} + b_i^l\Big)$$

Activation value at node i at layer l

Activation Function

Linear Predictor

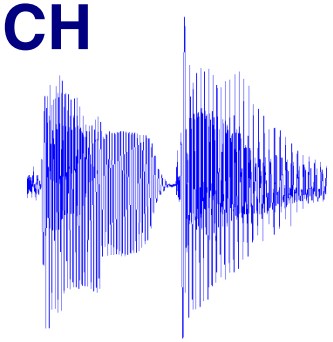
Traditional Machine Learning

VISION

hand-crafted features SIFT/HOG → your favorite classifier → “car”

fixed | learned

SPEECH



hand-crafted features MFCC → your favorite classifier → \'d ē p\

fixed | learned

NLP

This burrito place is yummy and fun!

hand-crafted features Bag-of-words → your favorite classifier → “+”

fixed | learned

Hierarchical Compositionality

VISION

pixels → edge → texton → motif → part → object

SPEECH

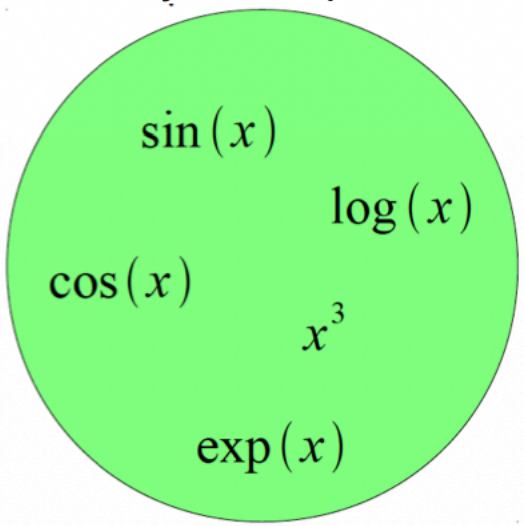
sample → spectral band → formant → motif → phone → word

NLP

character → word → NP/VP/.. → clause → sentence → story

Building a Complicated Function

Given a library of simple functions



Compose into a complicate function

- Deep Learning
- Grammar models
- Scattering transforms…

$$f(x) = \log(\cos(\exp(\sin^3(x))))$$

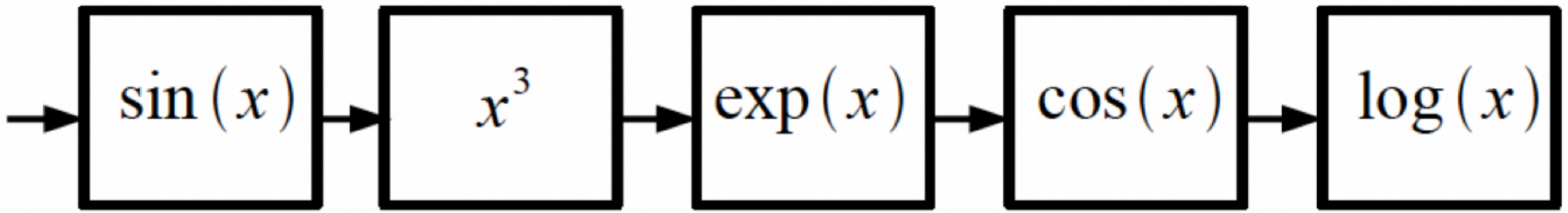
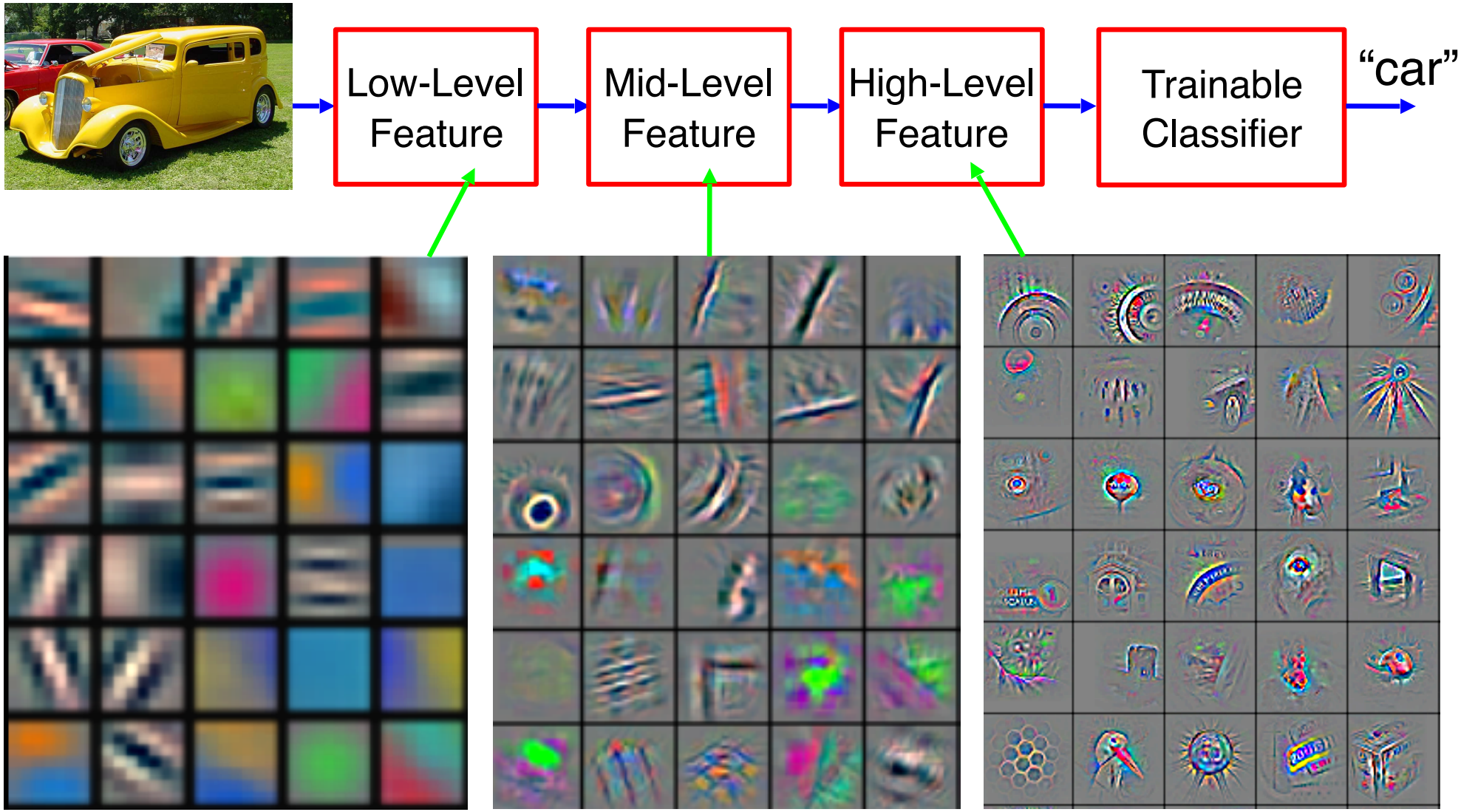


Figure from Aykut Erdem

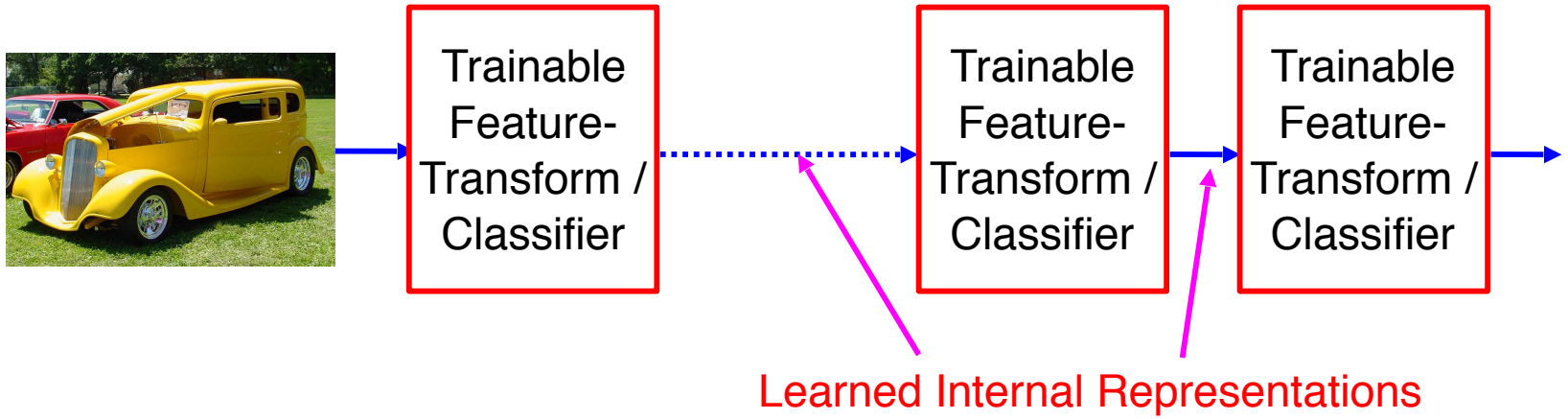
Deep Learning = Hierarchical Compositionality



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

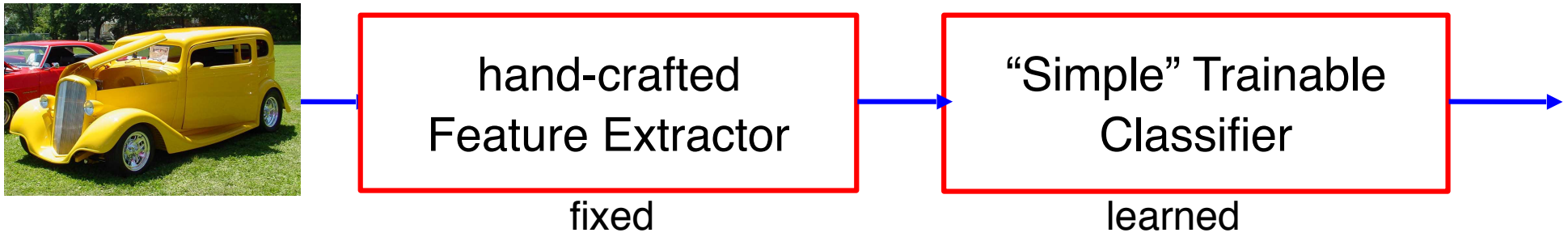
Deep Learning = End-to-End Learning

- A hierarchy of trainable feature transforms
  - Each module transforms its input representation into a higher-level one.
  - High-level features are more global and more invariant
  - Low-level features are shared among categories

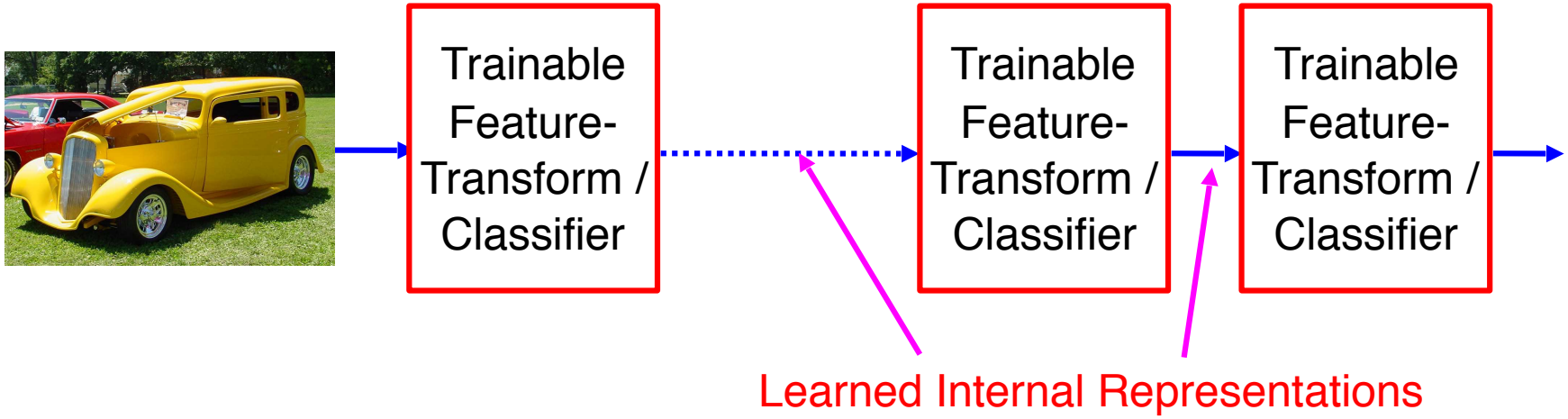


“Shallow” vs. Deep Learning

- “Shallow” models

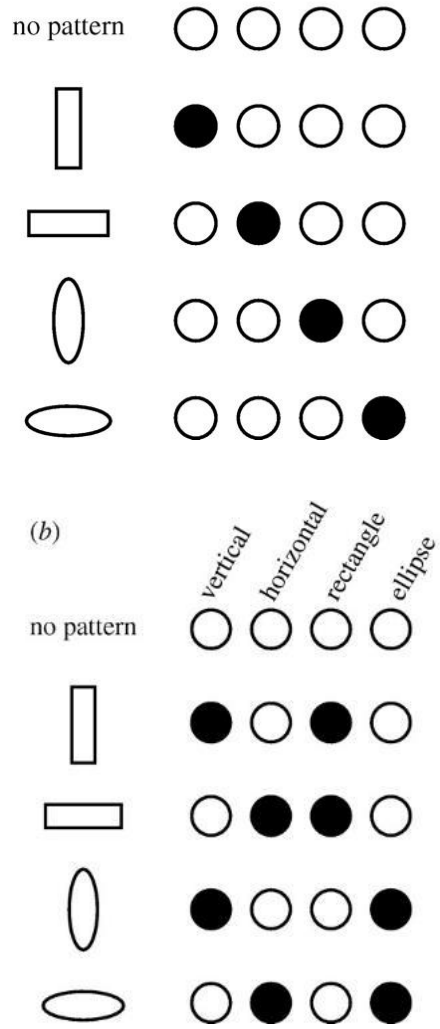


- Deep models



Localist Representations vs. Distributed Representations

- Each neuron must represent something
- The simplest way: dedicate one neuron to each thing
  - Easy to understand, easy to code by hand, easy to learn
  - But are very inefficient whenever the data has componential structure
- Distributed representation means a many-to-many relationship between two types of representation (such as concepts and neurons)
  - Each concept is represented by many neurons
  - Each neuron participates in the representation of many concepts

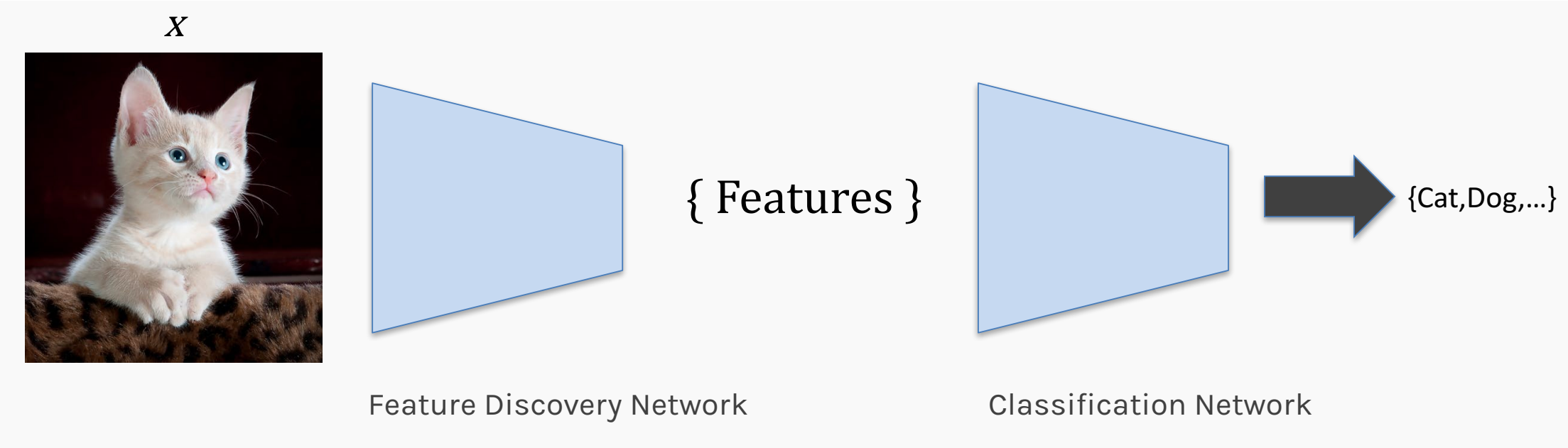


Autoencoders

- Unsupervised Representation Learning

Representation Learning

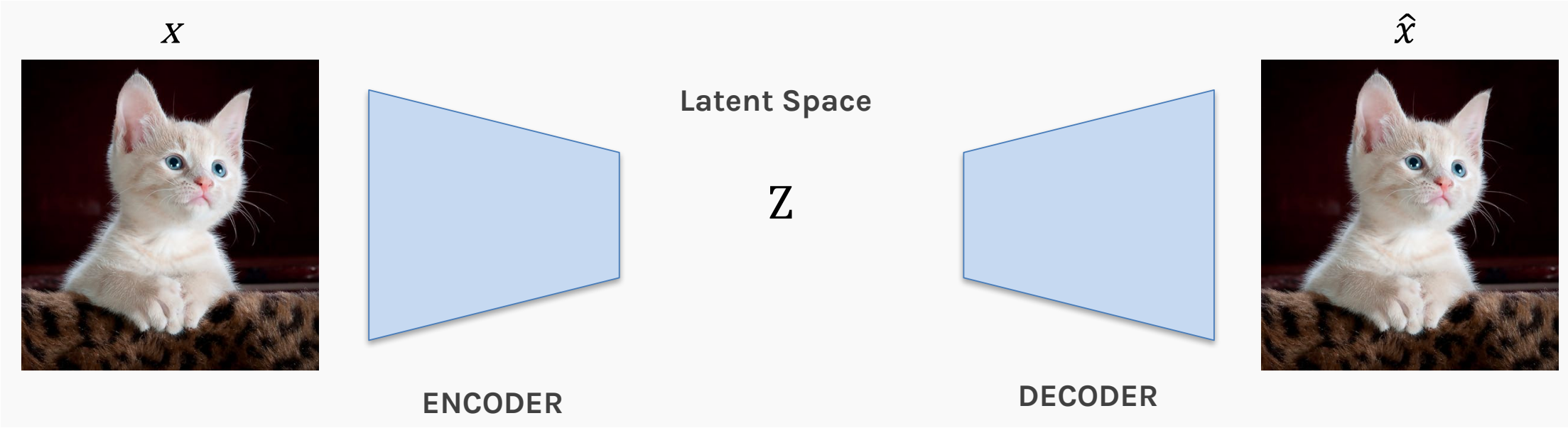
- Supervised Representation Learning
  - Train two networks by minimizing the prediction loss function



Slide from Pavlos Protopapas and Mark Glickman

Autoencoder

- Automatically finds the best way to encode the input so that the decoded version is as close as possible to the input



Slide from Pavlos Protopapas and Mark Glickman

Autoencoder

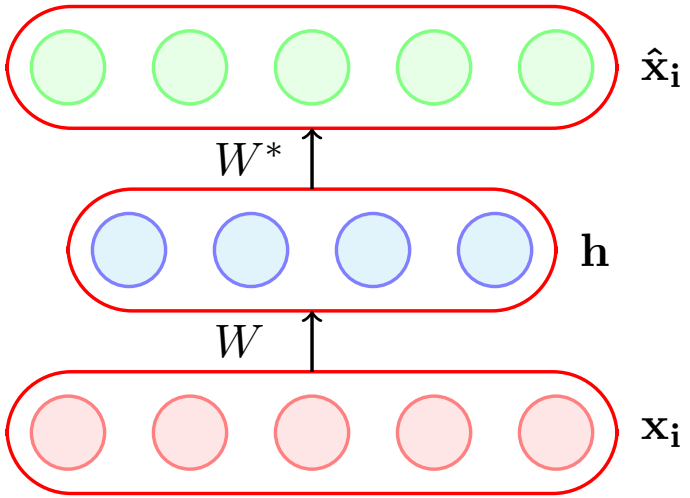
- Encoder: encodes the input x_i into a hidden representation h

$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$

- Decoder: decodes the input again from this hidden representation

$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Objective: minimize the distance between x_i and $\hat{x}_i$



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$

$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

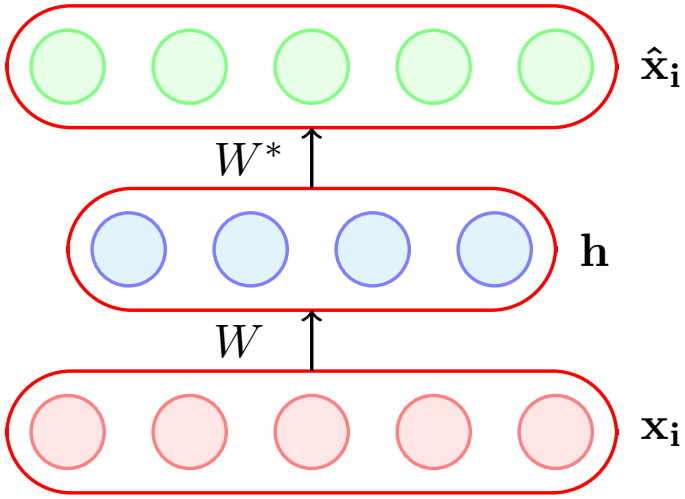
Autoencoder

- Chosen of f and g
  - Sigmoid function, Tanh function, …
- Chosen of Objective function
  - backpropagation

$$\min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij}-x_{ij})^2$$

$$i.e., \min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}(\hat{\mathbf{x}}_\mathbf{i}-\mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i}-\mathbf{x}_\mathbf{i})$$

$$\min_{\theta,w,w^*,\mathbf{b},\mathbf{c}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij}-x_{ij})^2+\lambda\|\theta\|^2$$

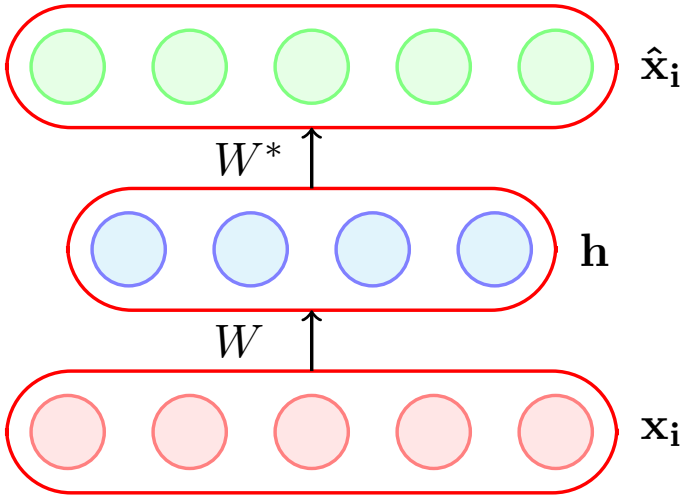


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i}+\mathbf{b})$$

$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h}+\mathbf{c})$$

Autoencoder

- General types of autoencoders
  - Undercomplete autoencoders if $\dim(h) < \dim(x_i)$
  - Overcomplete autoencoders if $\dim(h) > \dim(x_i)$
- If we are able to reconstruct $\hat{x}_i$ from h perfectly
  - h is a loss-free encoding of x_i
  - h captures all the important characteristics of x_i



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$

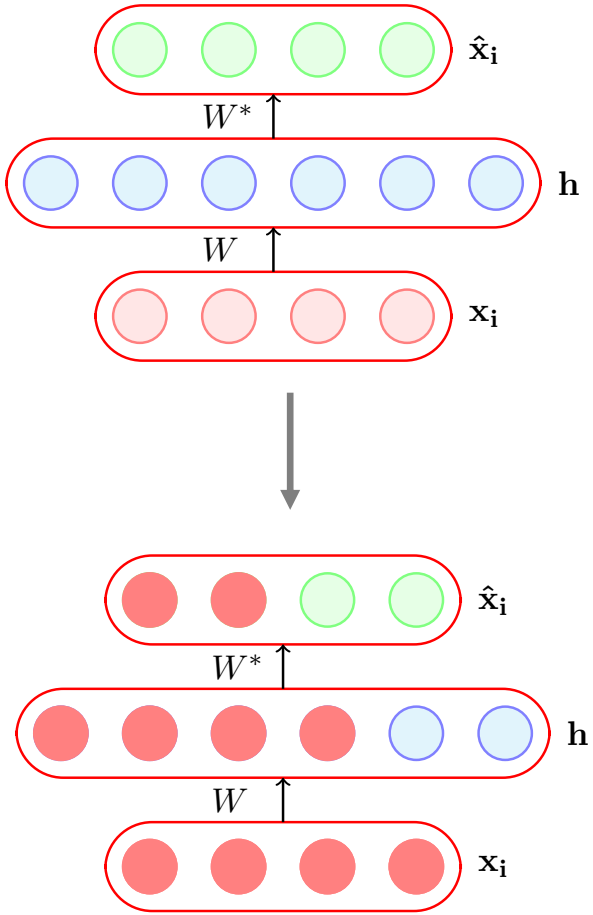
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

Problems of Overcomplete Autoencoders

- Overcomplete Autoencoders

$$\dim(\mathbf{h}) \geq \dim(\mathbf{x_i})$$

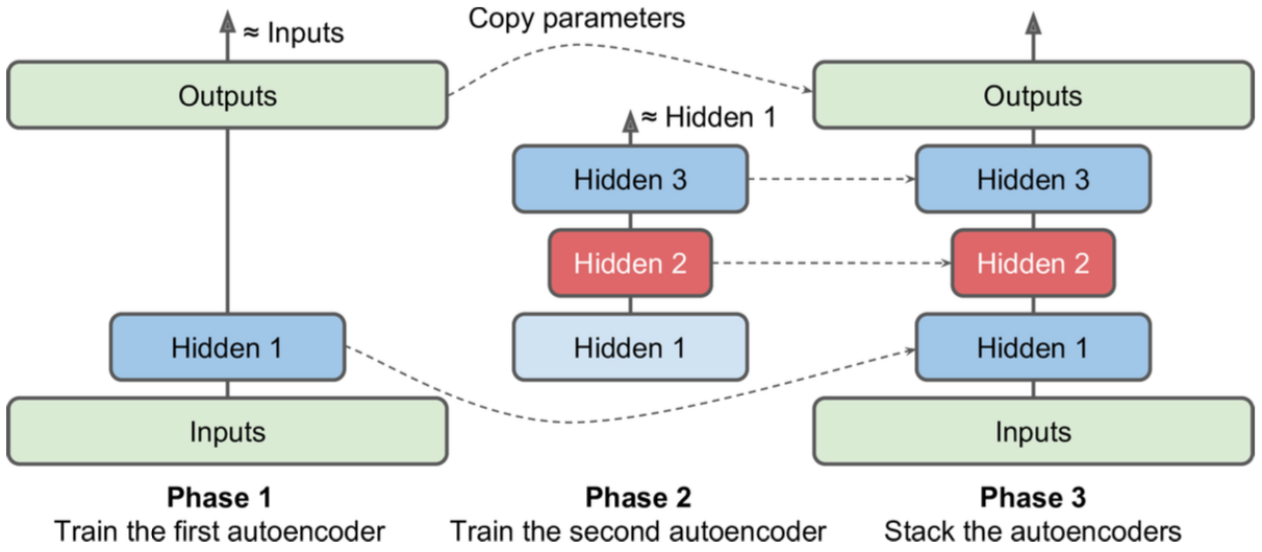
- Trivial Encoding:
  - Simply copying x_i into h and then copying h into $\hat{x}_i$
  - Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data



Variants of Autoencoders

- Stacked Autoencoders
- Denoising Autoencoders
- Sparse Autoencoders
- Contractive Autoencoders
- Variational Autoencoders
- … …

Stacked Autoencoders

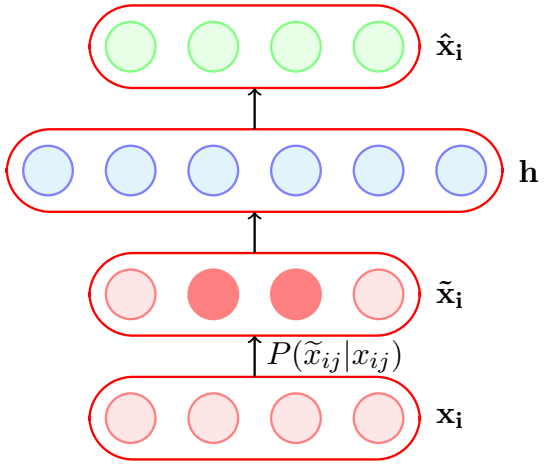


- A stacked autoencoder has multiple hidden layers
- Training
  - start with single hidden layer H_1
  - then use H_1's output as training set for Phase 2

Denoising Autoencoders

- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\tilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A $P(\tilde{x}_{ij}|x_{ij})$ used in practice: with probability q the input is flipped to 0, and with probability $(1-q)$ it is retained as it is

$$P(\tilde{x}_{ij} = 0|x_{ij}) = q$$
$$P(\tilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$

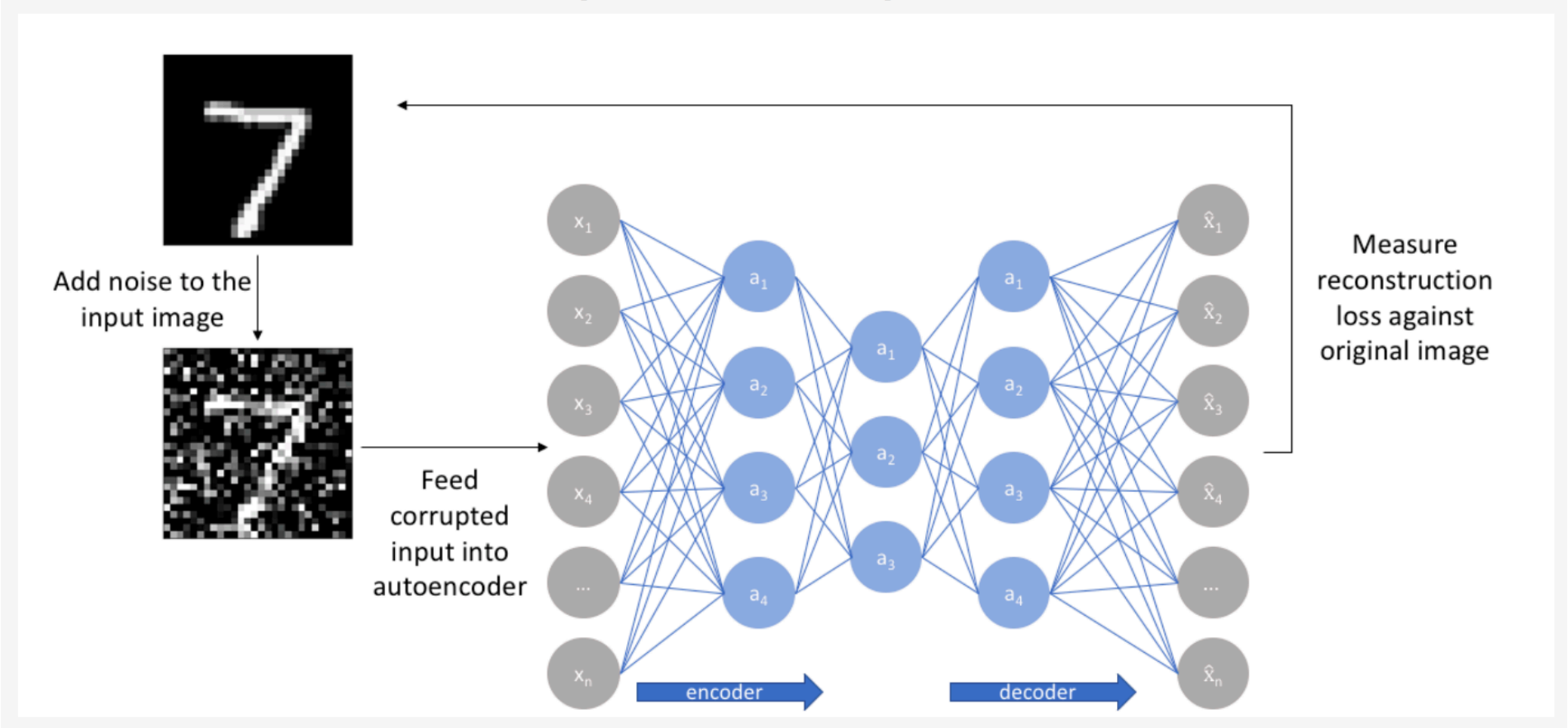


- Why?
  - It no longer makes sense for the model to copy the corrupted $\tilde{x}_i$ into $h(\tilde{x}_i)$ and then into $\hat{x}_i$ because the objective function will not be minimized by doing so
  - Instead, the model will have to capture the characteristics of the data correctly by relying on its interactions with other elements of x_i

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

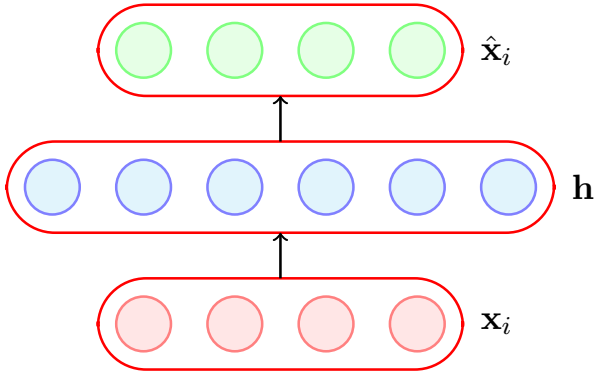
Denoising Autoencoders

- Trained with corrupted data points, but to reconstruct original ones



Sparse Autoencoders

- A hidden neuron with sigmoid activation will have values between 0 and 1
  - activated neurons when its output is close to 1
  - not activated when its output is close to 0
- A sparse autoencoder tries to ensure the neuron is inactive most of the times. Why?



0.4	0.5	0.5	0.4	0.6	0.5

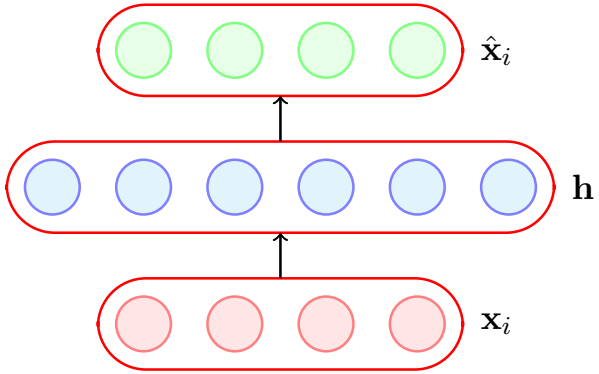
0.01	0.02	0.8	0.01	0.9	0.01

Which one is better?

Fewer nodes activating while still keeping its performance would guarantee that the autoencoder is actually learning **latent representations** instead of redundant information in our input data.

Sparse Autoencoders

- A hidden neuron with sigmoid activation will have values between 0 and 1
  - activated neurons when its output is close to 1
  - not activated when its output is close to 0
- A sparse autoencoder tries to ensure the neuron is inactive most of the times. Why?
- Objective:
  - encourage $\hat{\rho}_l \rightarrow \rho$ where ρ is typically very close to 0 (0.005)



0.4	0.5	0.5	0.4	0.6	0.5

0.01	0.02	0.8	0.01	0.9	0.01

Which one is better?

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_l} \qquad \hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

Reconstruction Loss

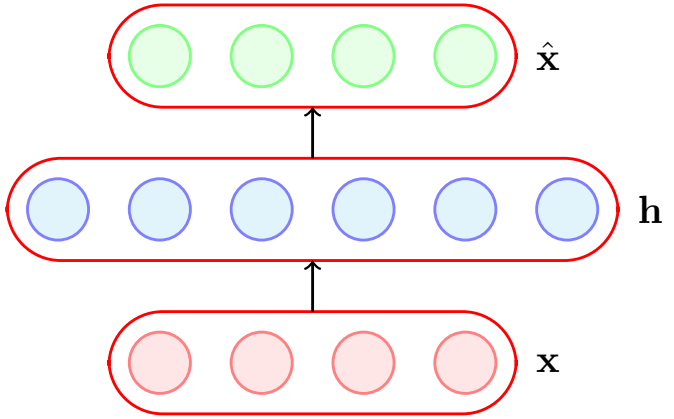
$$\hat{\mathscr{L}}(\theta) = \mathscr{L}(\theta) + \Omega(\theta)$$

Contractive Autoencoders

- A contractive autoencoder tries to prevent an overcomplete autoencoder from learning the identity function, and tries to learn more robust hidden representations
- Adding regularization in the objective:
  - $J_X(h)$ is the Jacobian of the encoder

$$\Omega(\theta) = \|J_{\mathbf{x}}(\mathbf{h})\|_F^2$$

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$



Jacobian of the encoder

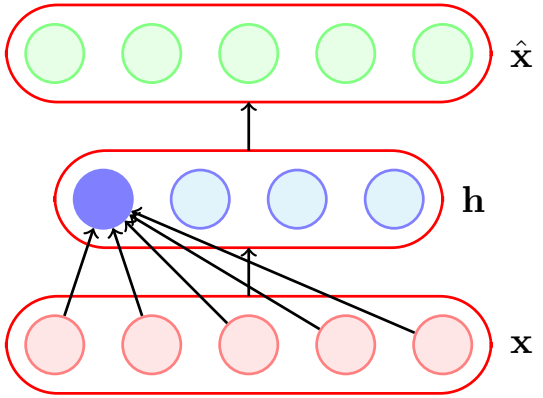
- Suppose the input has *n* dimensions, and the hidden layer has *k* dimensions

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$

- the (l, j) entry $\frac{\partial h_l}{\partial x_j}$ of the Jacobian captures the variation in the output of the l^{th} neuron with a small variation in the j^{th} input

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j}\right)^2$$

  - So, $\frac{\partial h_1}{\partial x_1} = 0$ means that this neuron is not very sensitive to variations in the input x_1
  - But it contradicts our goal of minimizing the reconstruction loss $\mathcal{L}(\theta)$ which requires h to capture variations in the input!!



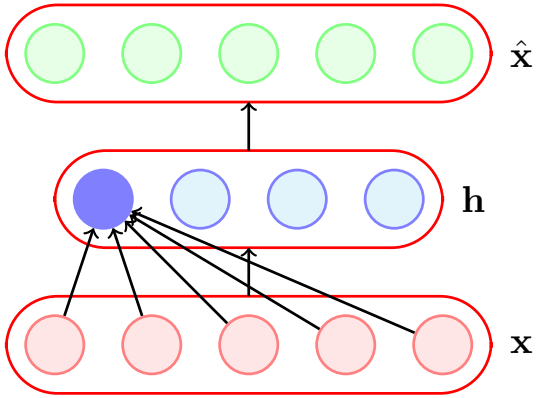
Jacobian of the encoder

- $\mathcal{L}(\theta)$ – Capture important variations in data
- $\Omega(\theta)$ – do not capture variations in data

- Tradeoff – capture only very important variations in the data

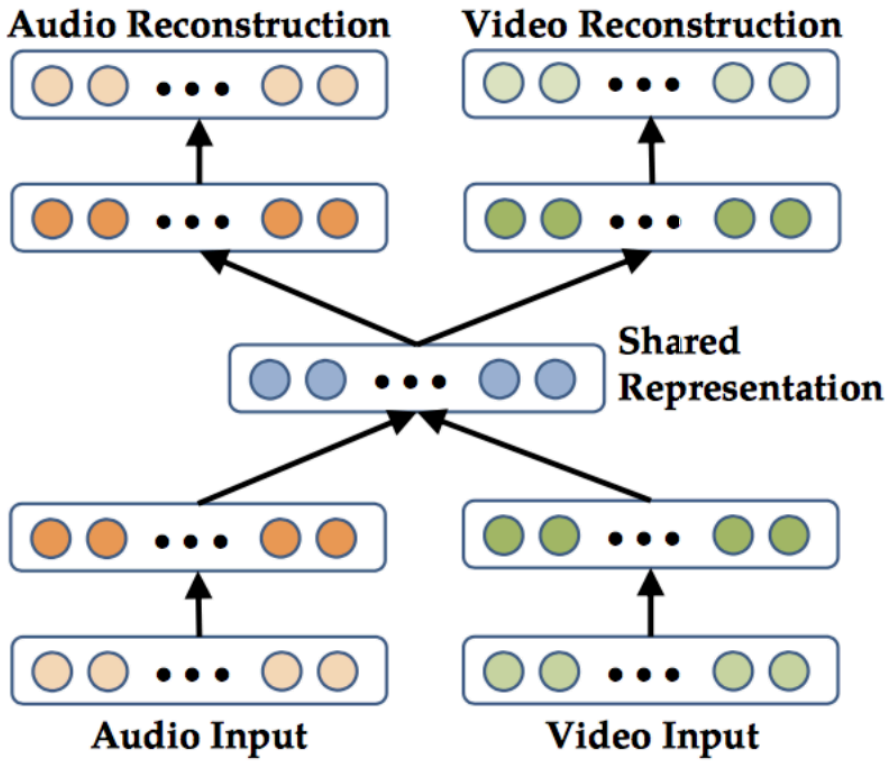
$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j} \right)^2$$



Application of Autoencoders

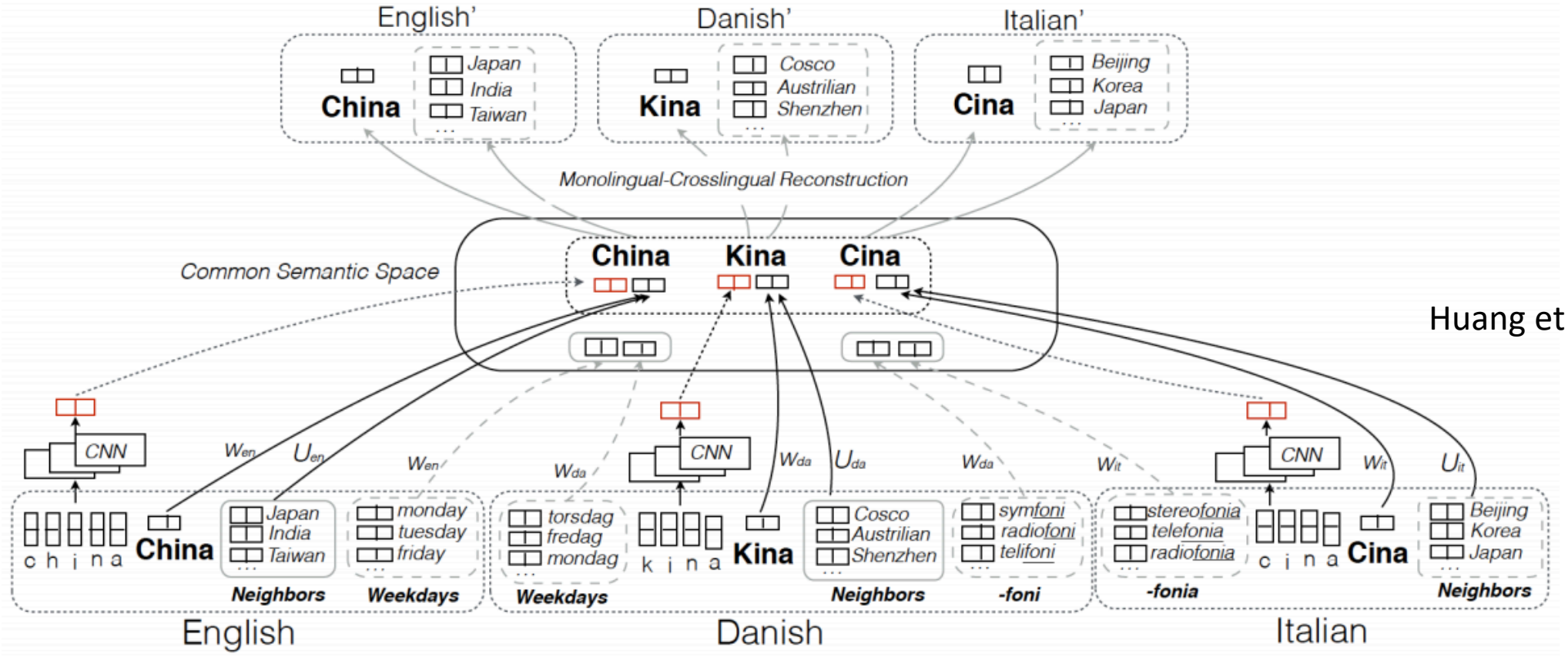
- Learning source-agnostic representations
- e.g., Multi-model representations
  - Each modality can be pre-trained using autoencoder
  - Then, train the full model by reconstructing both modalities using
    - both Audio & Video
    - just Audio
    - just Video



Ngiam et al., 2011

Application of Autoencoders

- Learning source-agnostic representations
- e.g., Multilingual Common Semantic Space



Huang et al., 2018