Auto Encoders

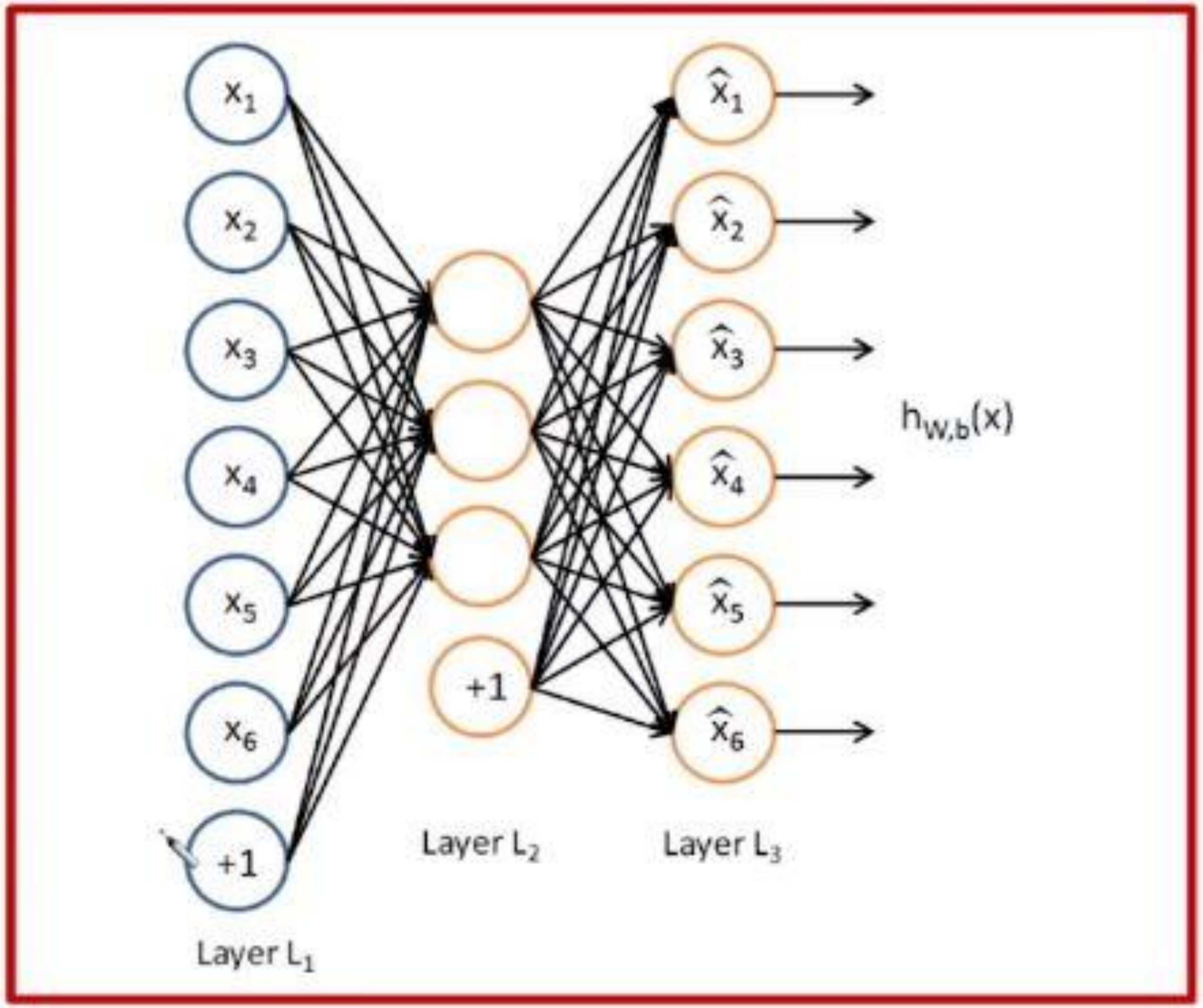
Introduction

- Auto-encoders are used to **capture structure in data, using unsupervised learning**
- Data is provided as input, and *the output of the network tries to* ***reconstruct the input***
- **Learning** is performed using **backpropagation** or related methods
- The network **captures** a **reduced** representation of inputs
- **Useful** for **pre-training a network**, improving learning and allowing greater depth

[1] Auto Encoders Architecture

- Autoencoders are a <u>multi-layer neural network</u> with a specific topology
- The <u>target output</u> of the network <u>is set to the input</u>
- The **<u>aim</u>** of training is to **<u>minimise the error of reconstruction</u>**
- **<u>Often</u>** a **<u>reduced</u>** <u>set of hidden units is used</u>, creating an information bottleneck
- **Weights** <u>between the input and hidden layer</u> are **often tied with** <u>weights between the hidden layer and output</u>

Auto Encoders Architecture



Auto Encoders Architecture

Application:

A process of sending data from cellphone to the cloud

process of sending data from cellphone to the cloud has three steps:

1. **Encoding**: in cellphone, map data $x(i)$ to compressed data $z(i)$.
2. **Sending**: send $z(i)$ to the cloud.
3. **Decoding**: in the cloud, map from compressed data $z(i)$ back to $\tilde{x}(i)$, which approximates the original data.

To map data back and forth more systematically, we propose that:

z and $\tilde{x}$ are functions of their inputs

Auto Encoders Architecture

Given an input vector $x \in [0, 1]^d$, hidden unit and output, **activations** are calculated as:

$$\mathbf{y} = \varphi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{z} = \varphi(\mathbf{W}'\mathbf{y} + \mathbf{b}')$$

Reconstruction <u>**error**</u> can be calculated using a number of methods, including squared error:

$$E = \frac{1}{2}\|\mathbf{z} - \mathbf{x}\|^2$$

Auto Encoders Architecture

$$\mathbf{y} = \varphi(\mathbf{W}\mathbf{x} + \mathbf{b}) \qquad E = \frac{1}{2}\|\mathbf{z} - \mathbf{x}\|^2$$

$$\mathbf{z} = \varphi(\mathbf{W}'\mathbf{y} + \mathbf{b}')$$

Let the goal is to have x̃(i) to approximate x(i) ,

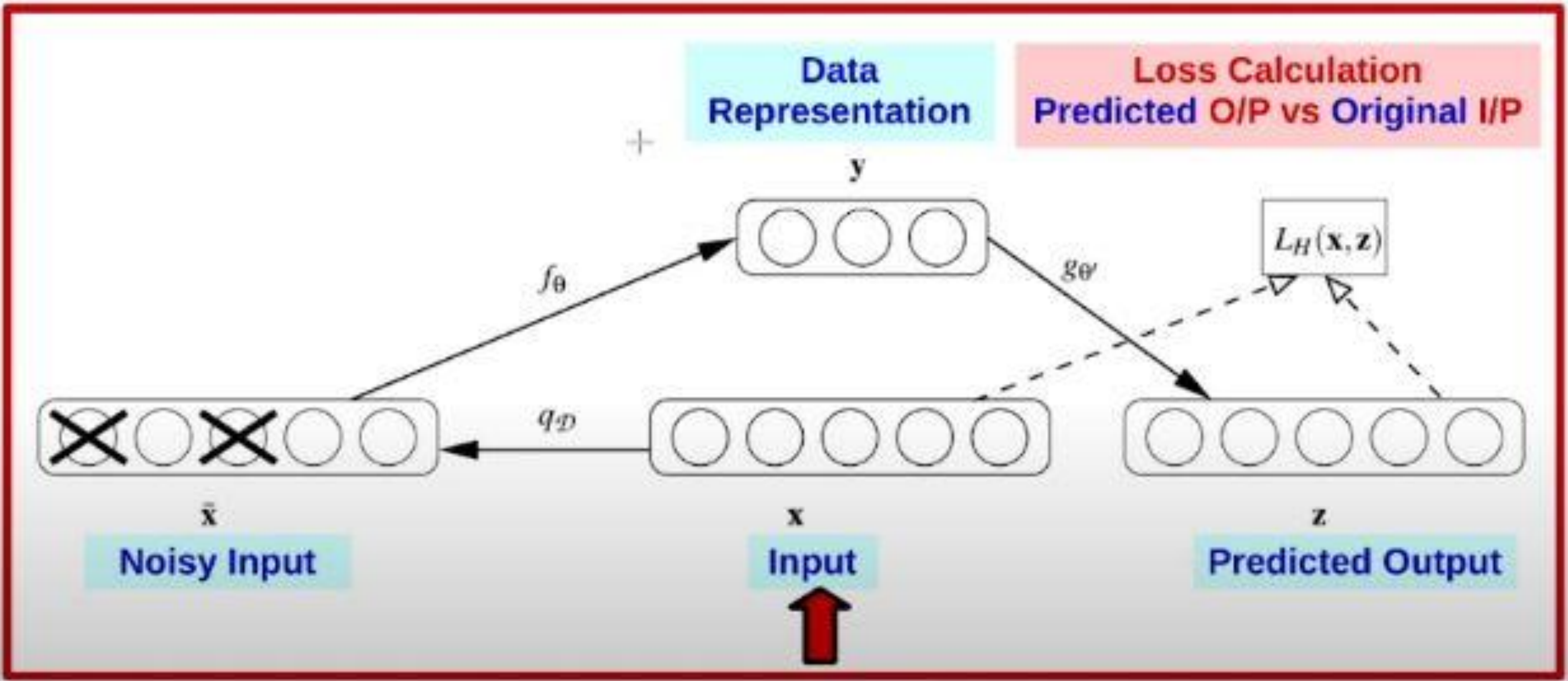
we can set up the following objective function

$$\boxed{\begin{aligned} J(W_1, b_1, W_2, b_2) &= \sum_{i=1}^{m} \left(\tilde{x}^{(i)} - x^{(i)} \right)^2 \\ &= \sum_{i=1}^{m} \left(W_2 z^{(i)} + b_2 - x^{(i)} \right)^2 \\ &= \sum_{i=1}^{m} \left(W_2 \left(W_1 x^{(i)} + b_1 \right) + b_2 - x^{(i)} \right)^2 \end{aligned}}$$

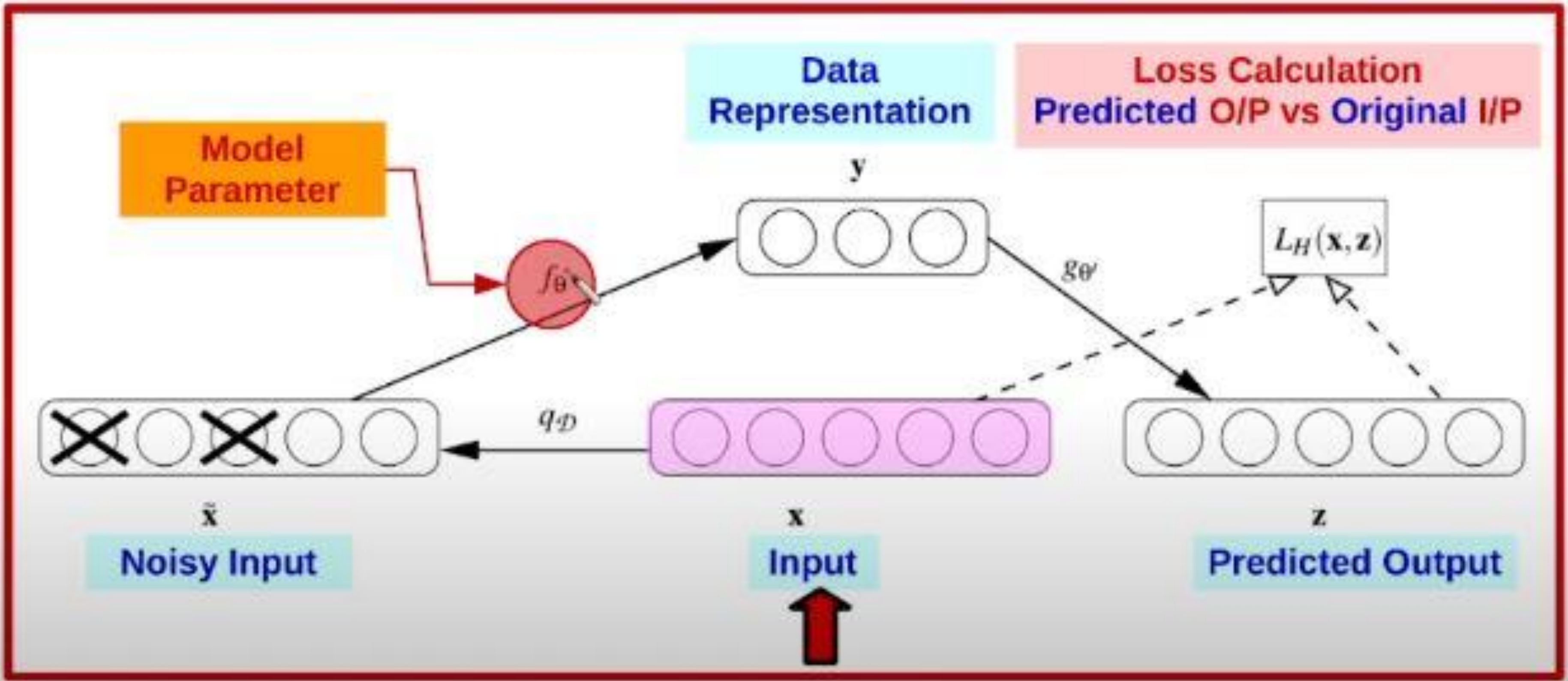
De-noising Auto Encoders Architecture

- The denoising auto-encoder architecture.
- "x" is stochastically corrupted (via q_D fn) to $\tilde{x}$.
- The auto-encoder then maps it to y (via encoder f_θ)
- The auto-encoder attempts to reconstruct x via decoder $g_{\theta'}$, producing reconstruction z.
- Reconstruction error is measured by loss $L_H(x, z)$.

De-noising Auto Encoders Architecture



De-noising Auto Encoders Architecture

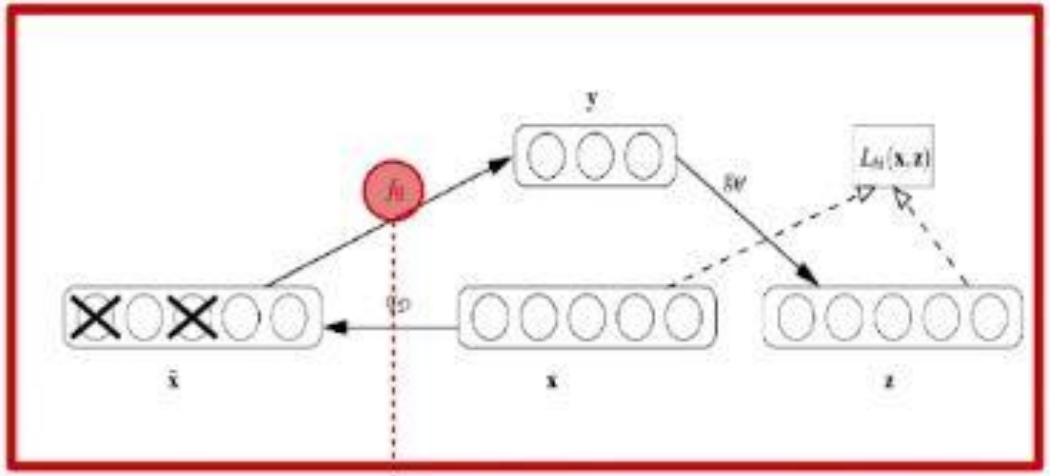


[2] Stacked Auto-Encoders

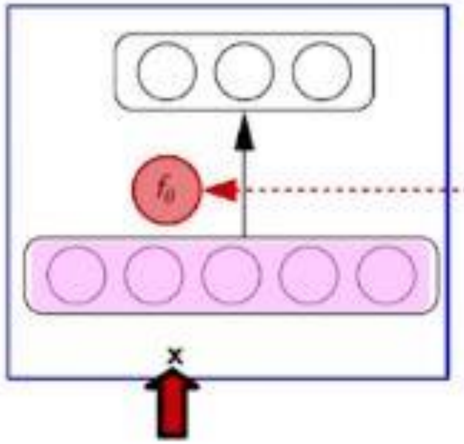
- Several AEs can be **stacked** to form a **deep hierarchy**
- Each layer **receives** its **input** from the latent representation of the **layer below**.
- **unsupervised** ***pre-training*** can be done in greedy, **layer-wise fashion**.
- Afterwards the **weights can be fine-tuned** using ***back-propagation,***

De-noising Stacked Auto Encoders Architecture

X	Original Input
$\widetilde{X}$	Noised Version of X
q_d	Function to add Noise
f_θ	Encoding Function
y	Encoded *Noisy* Input
g_θ	Decoding Function
Z	Decoded Noisy Input

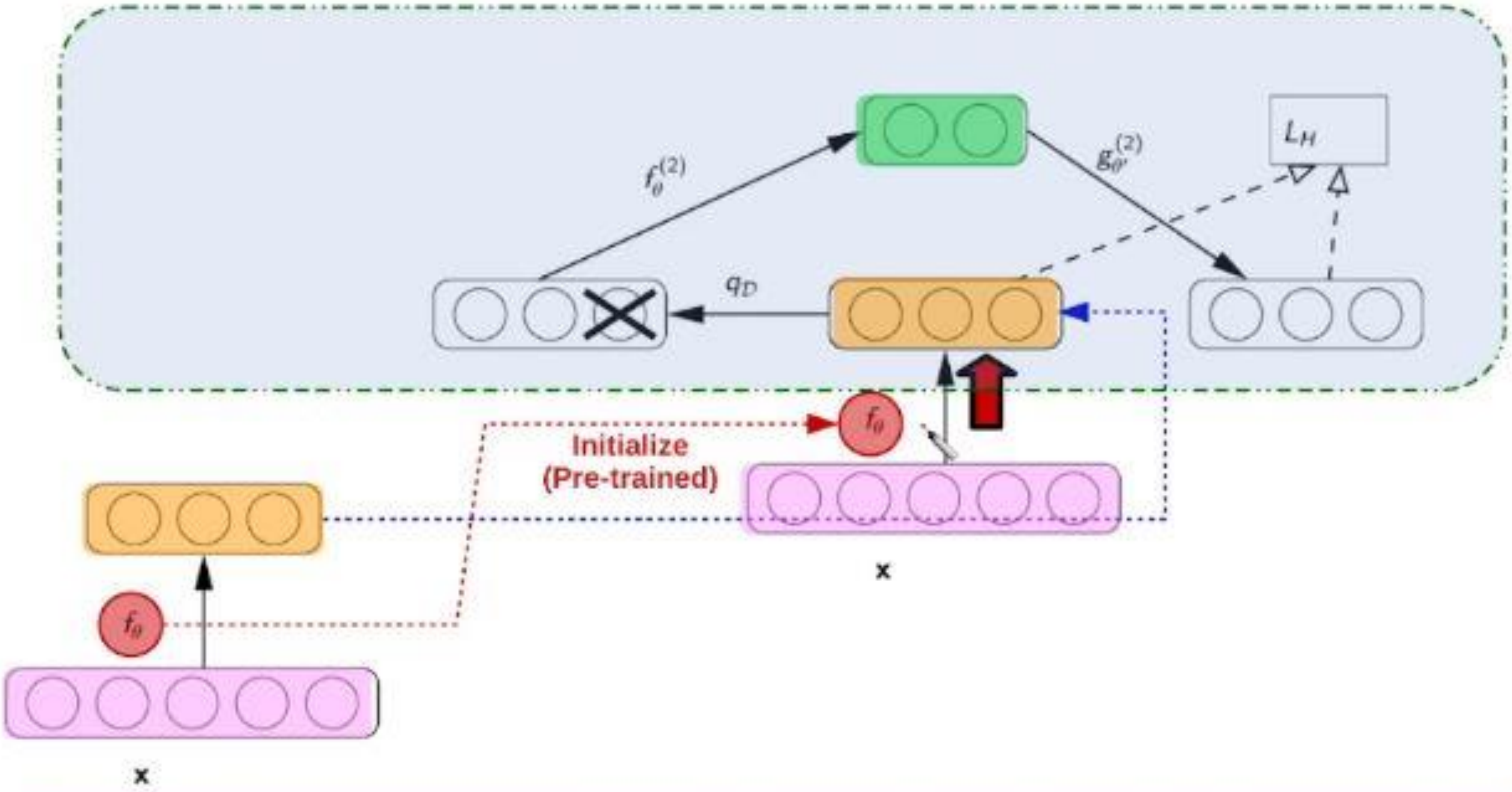


$L_H(X, Z)$ Loss between Decoded Output and Original Input



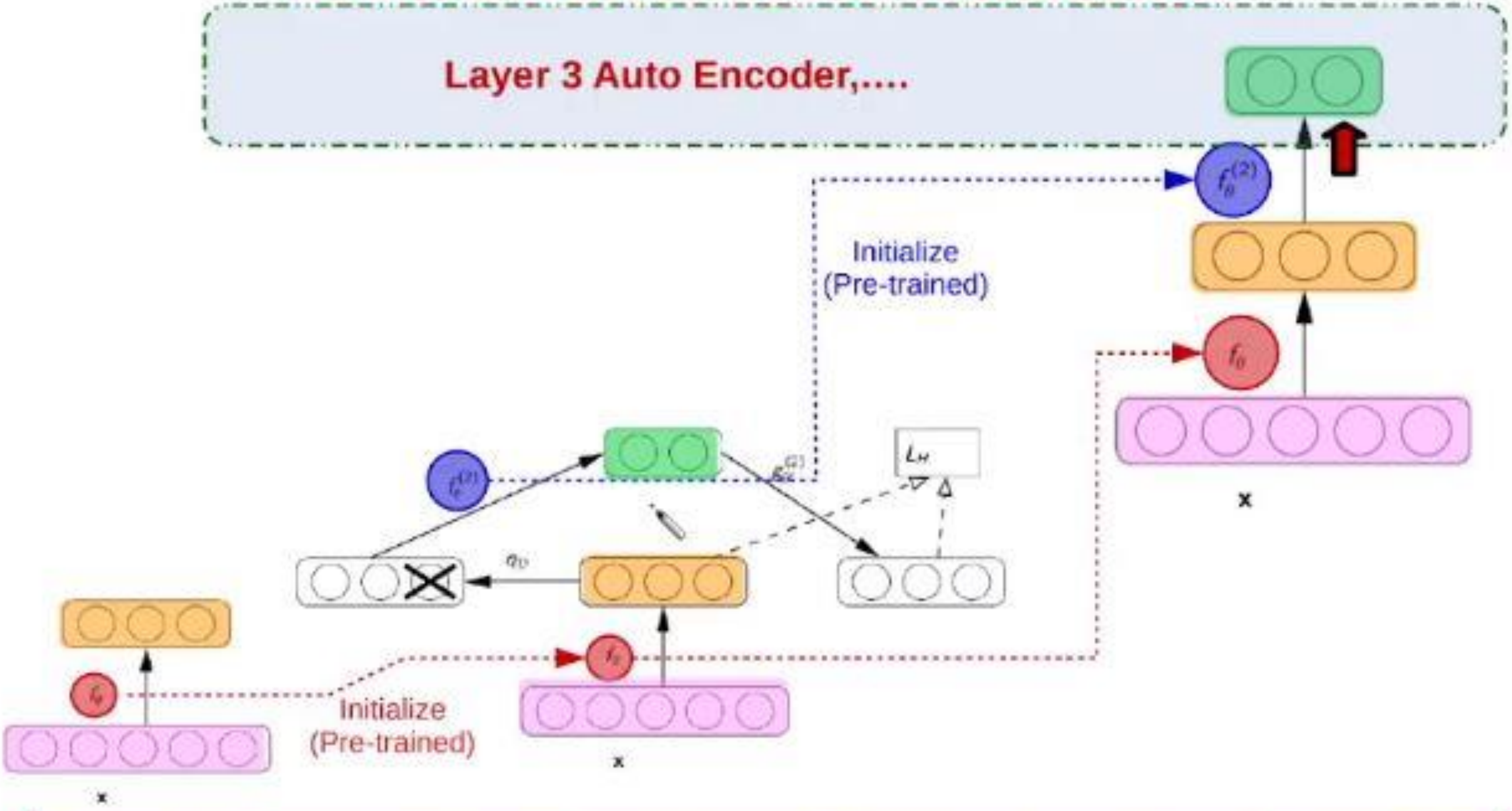
After training a first level denoising autoencoder,
its learned encoding function f_θ is used on clean input

De-noising Stacked Auto Encoders Architecture



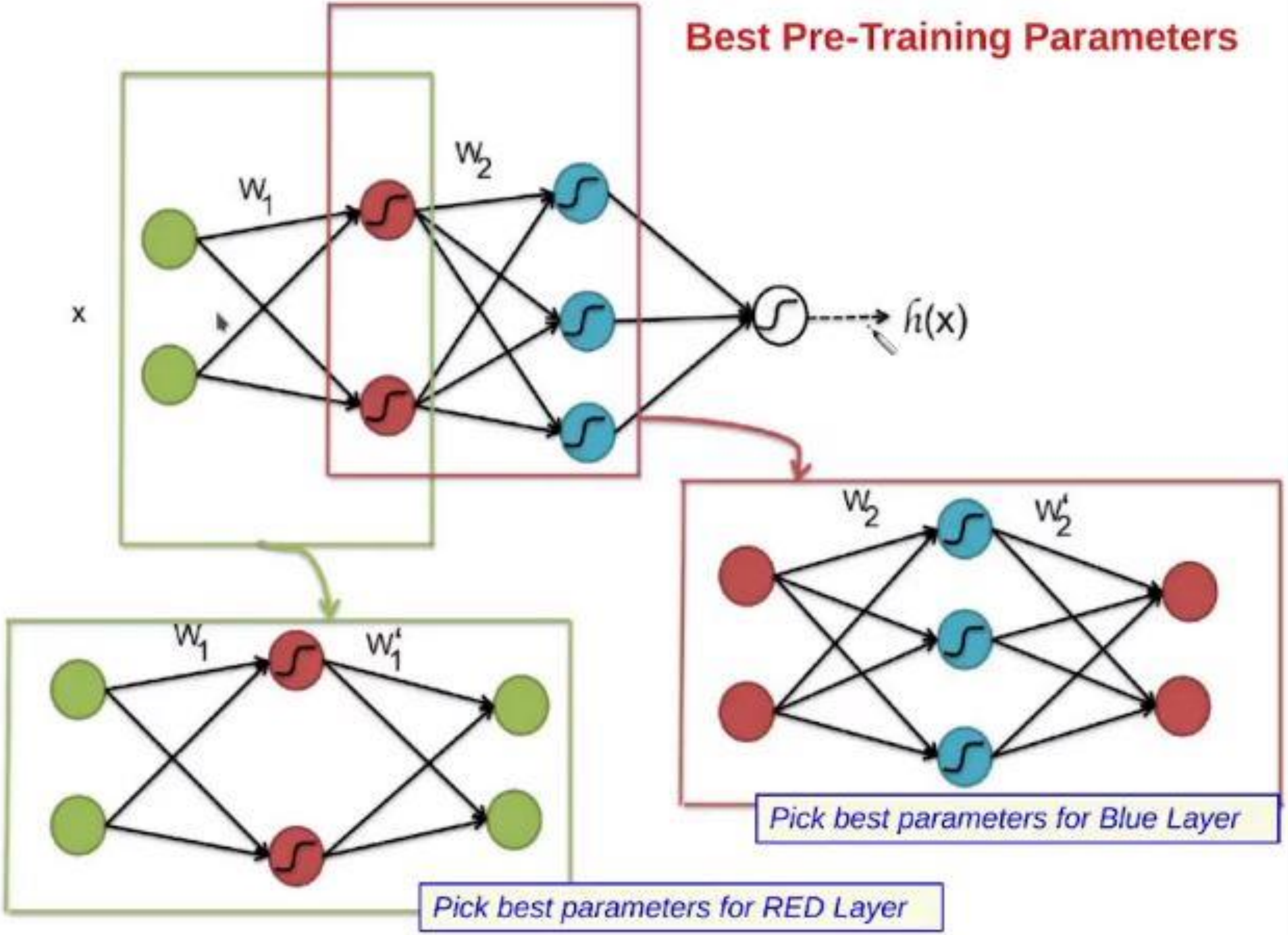
The resulting representation is used to train a second level denoising autoencoder to learn a second level encoding function $f_\theta^{(2)}$

De-noising Stacked Auto Encoders Architecture

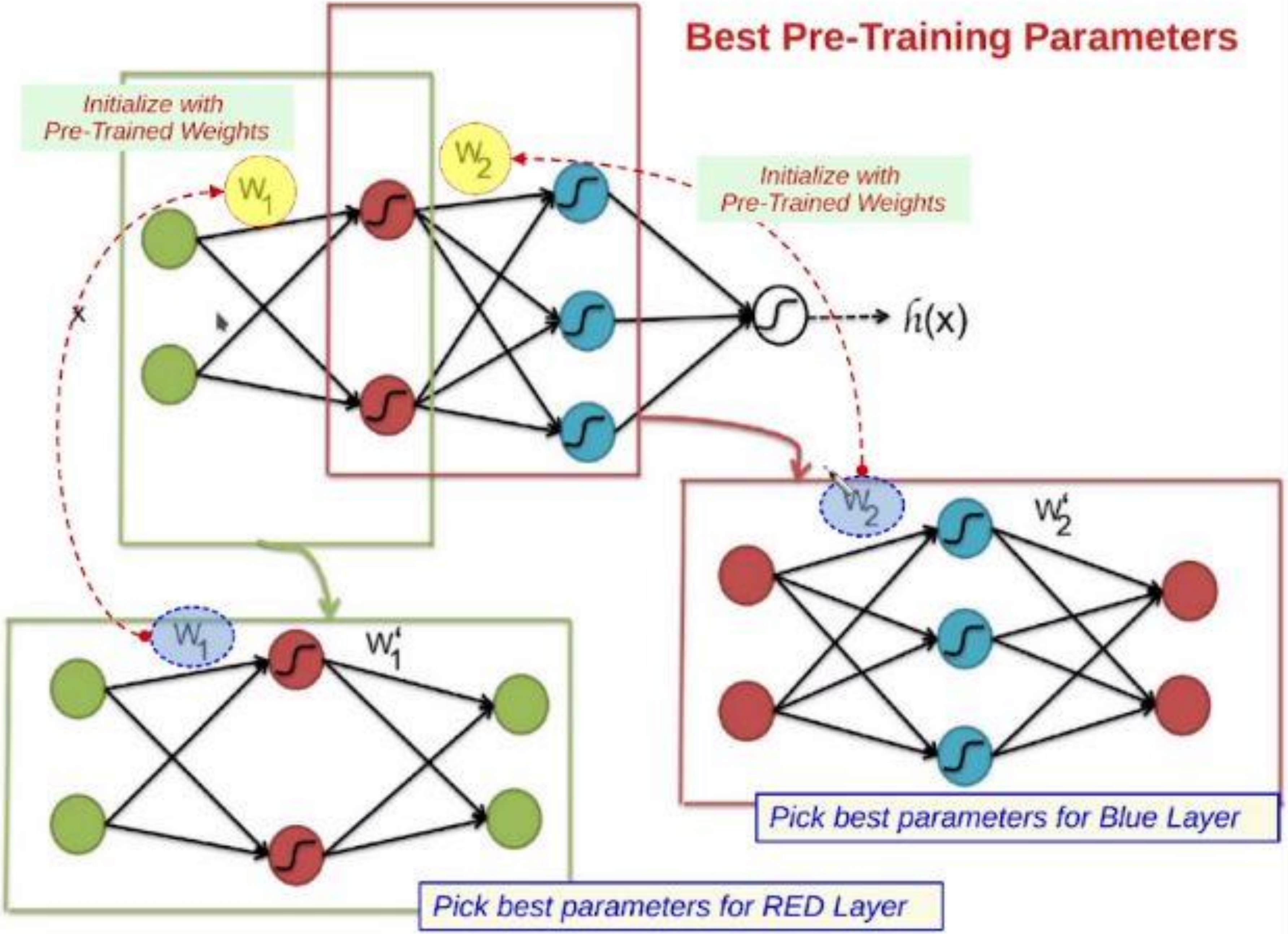


The resulting representation is used to train a Third level denoising autoencoder.
the procedure can be repeated

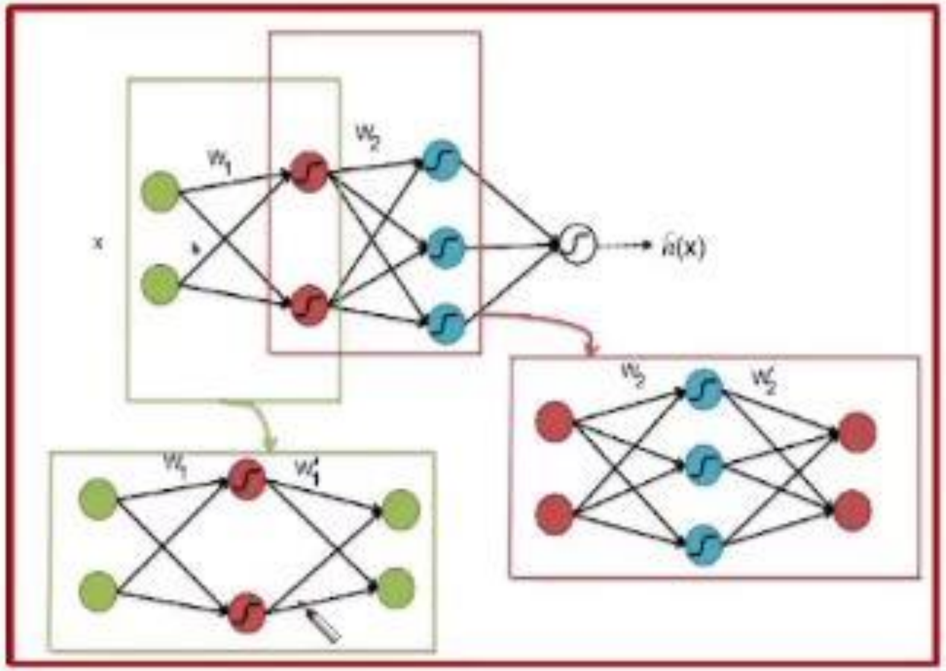
Best Pre-Training Parameters



Best Pre-Training Parameters



To train the **red neurons,** we will train an autoencoder that has parameters **W1** and **W1’** .

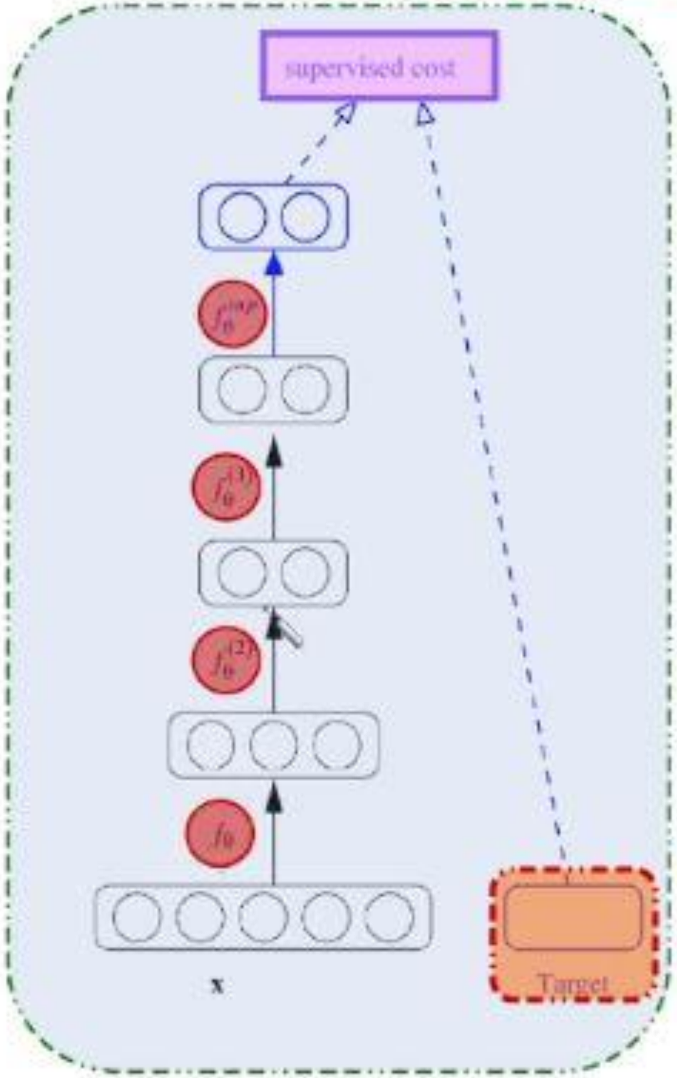


After this, we will use **W1** to compute the values for the **red neurons** for all of our data, which will then be used as input data to the subsequent autoencoder.

The parameters of the decoding process **W1’** will be discarded.

The subsequent autoencoder uses the values for the **red neurons** as inputs, and trains an autoencoder to predict those values by adding a decoding layer with parameters **W2’** .

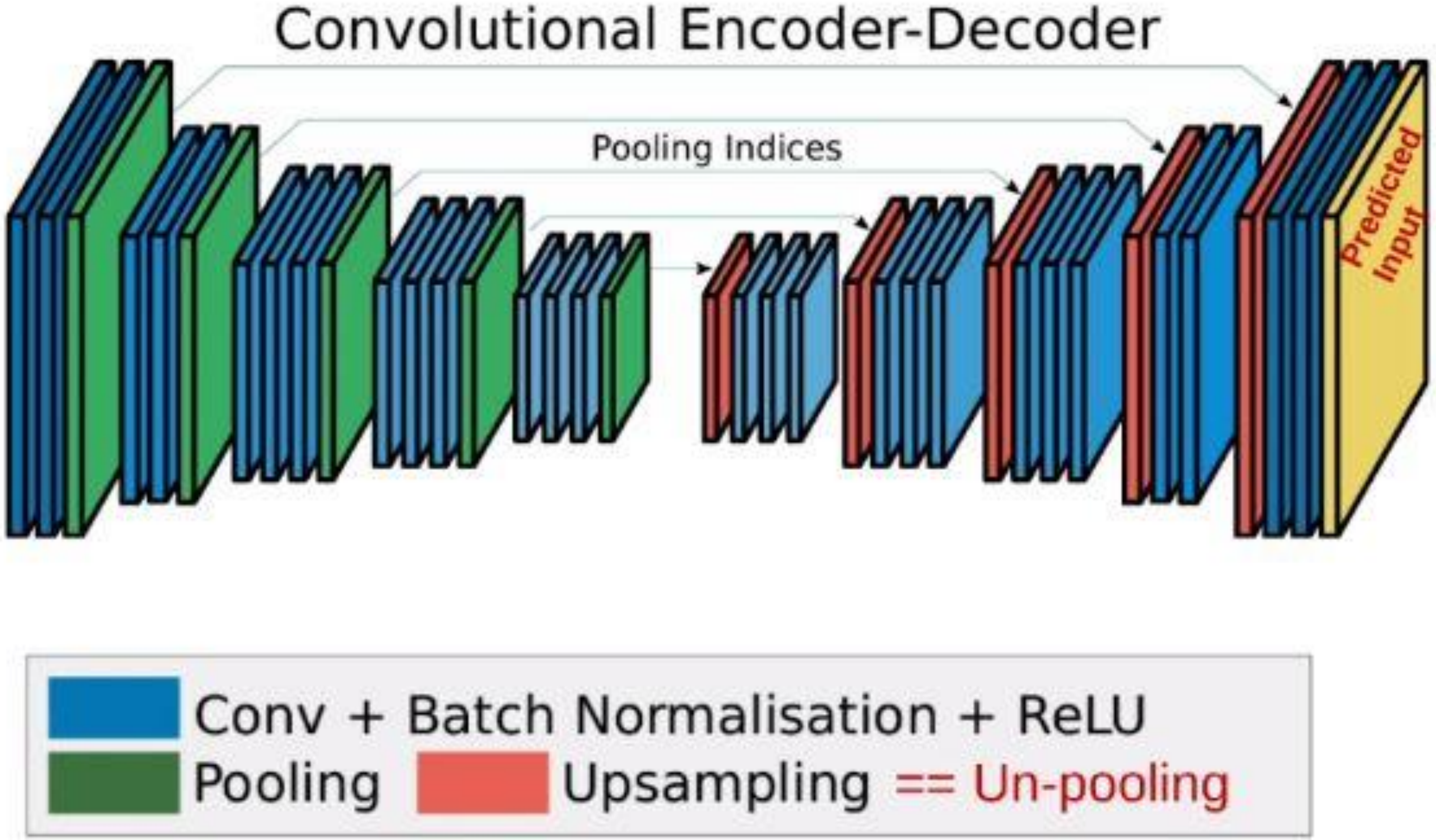
De-noising Stacked Auto Encoders Architecture



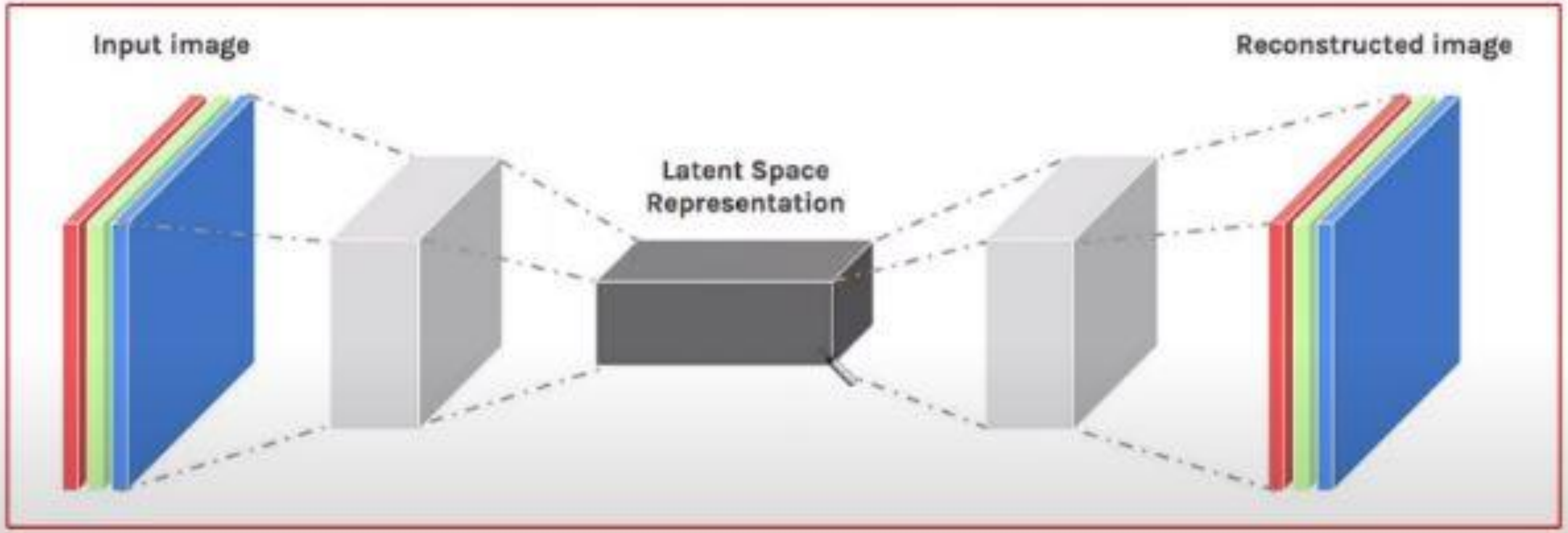
Fine-tuning of a deep network for classification.

- Train a stack of encoders as explained in the previous slides
- An **output layer is added** on top of the stack.
- The **parameters of the whole system** are **fine-tuned** to minimize the error in predicting the supervised target (e.g., class), by performing gradient descent on a **supervised cost**.

[3] Convolutional Auto Encoders Architecture



[3] Convolutional Auto Encoders Architecture



Convolutional Auto Encoders Training

For a single layer k, given the **convolution operator** $*$, max pooling operator Ψ, filter **weights** W_k and **biases** b_k , the projection operation is given by the following:

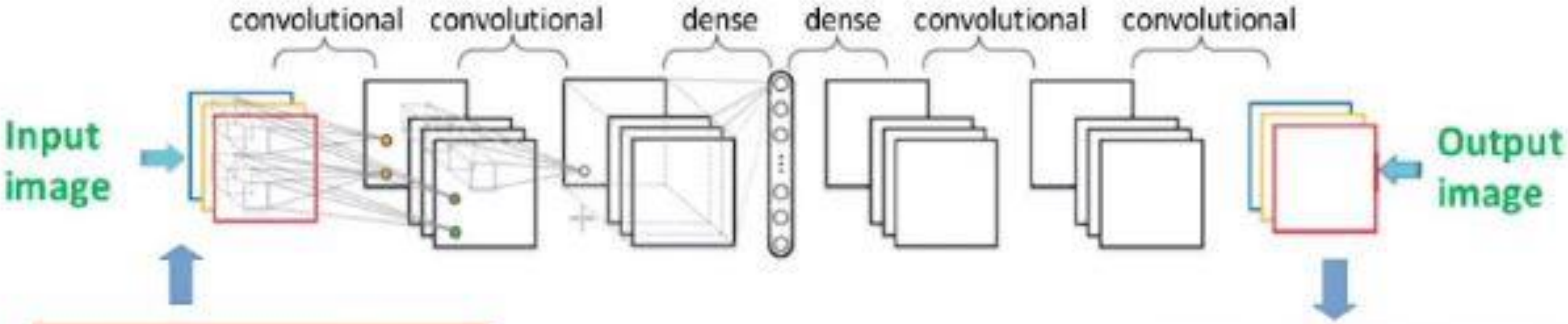
$$\Phi_k(\mathbf{X}) = \tanh(\Psi(\mathbf{X} * \mathbf{W}_k + \mathbf{b}_k))$$

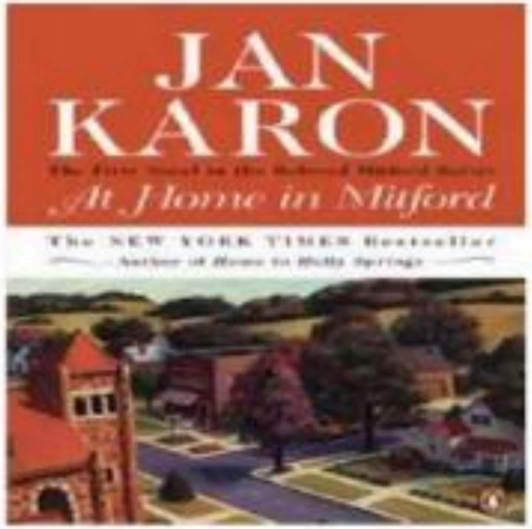
backward process is done by **simply inverting each component of the network**, taking *special care to invert the pooling operator Ψ*.

In general Ψ operator is non-invertible, so instead we formulate **two approximate inverses** that can be used in different cases:

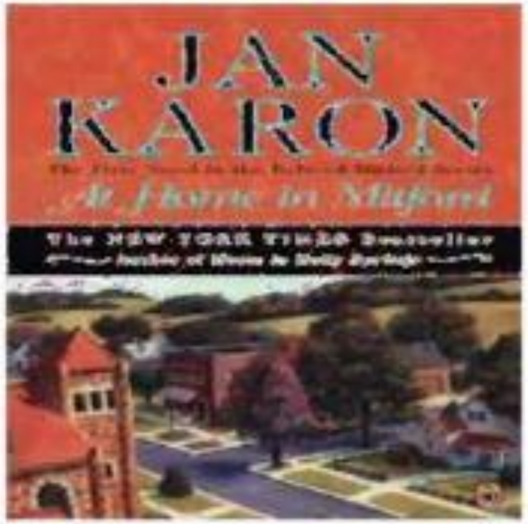
- A **single individual pooling location is picked randomly** and the pooled value is placed there, setting the other pooling location to zero
- The pooled value is instead **distributed evenly across both pooling locations**. This is used when training in the fine tuning stage,

Example of Reconstructed Image using CAE



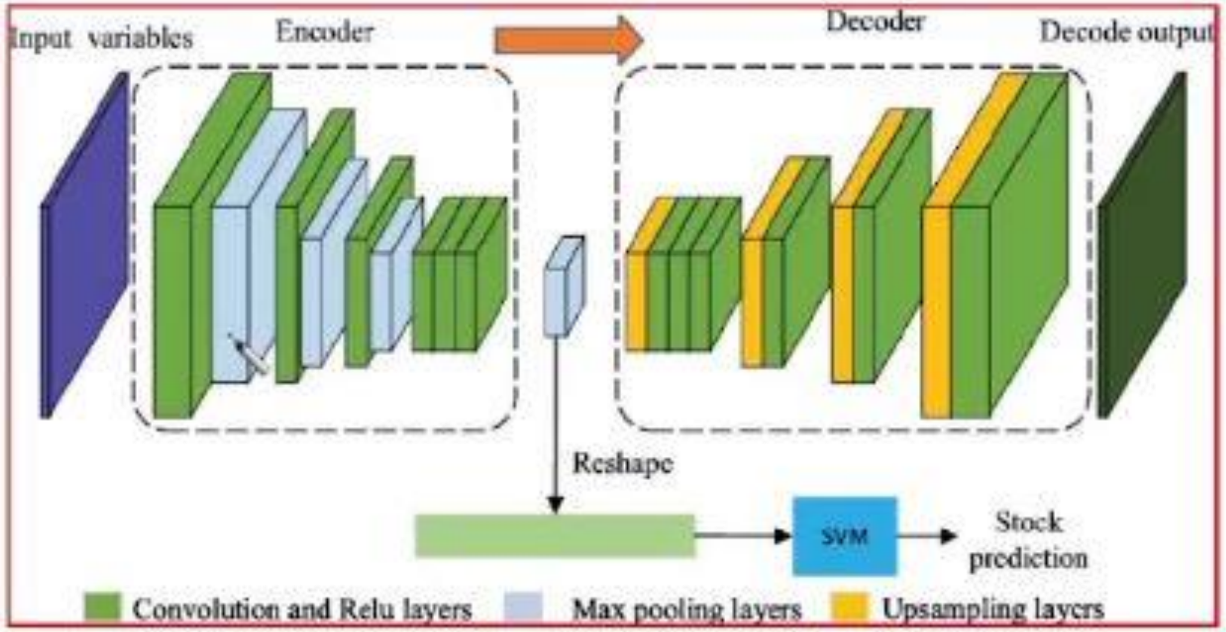


Needs more training
For better quality output



Other usage of Auto-Encoders

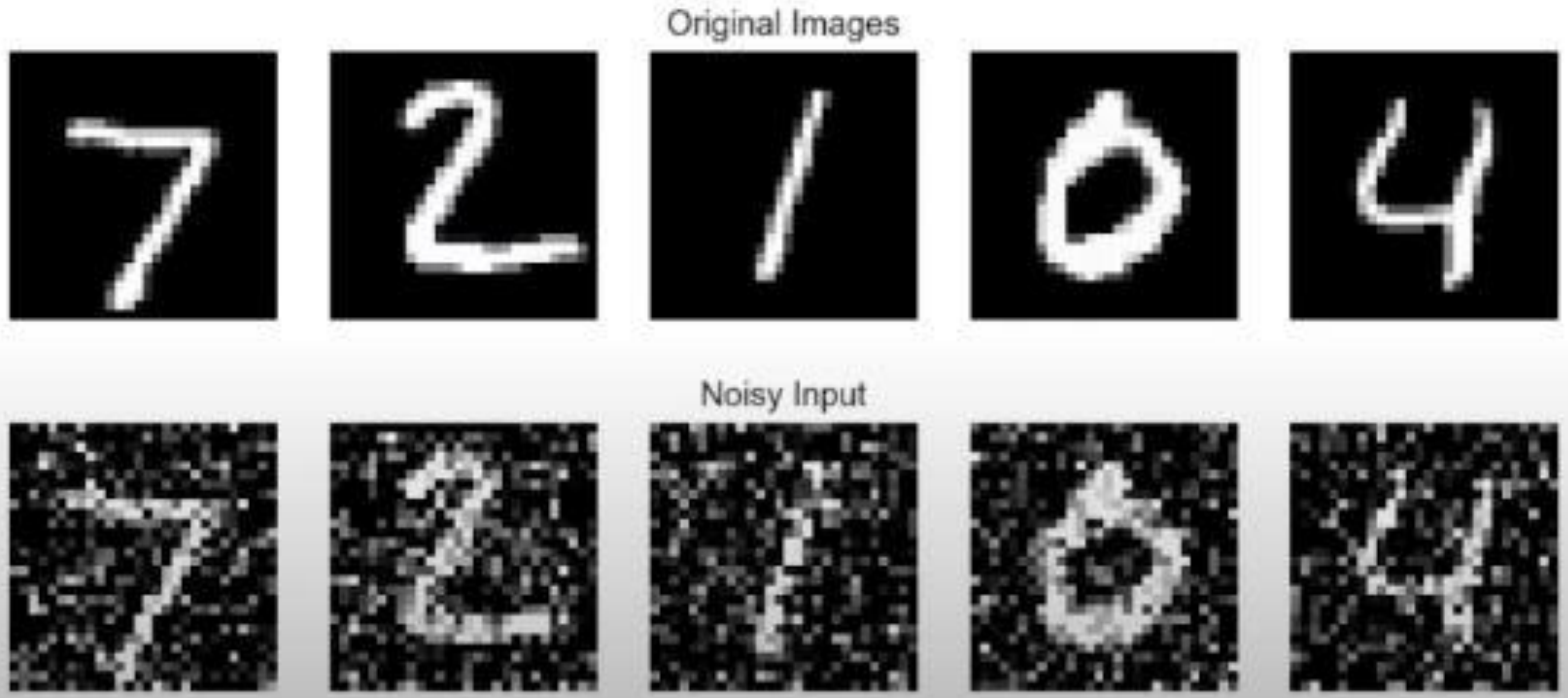
- the **top level activations** can be used as feature vectors for **SVMs or other classifiers**.



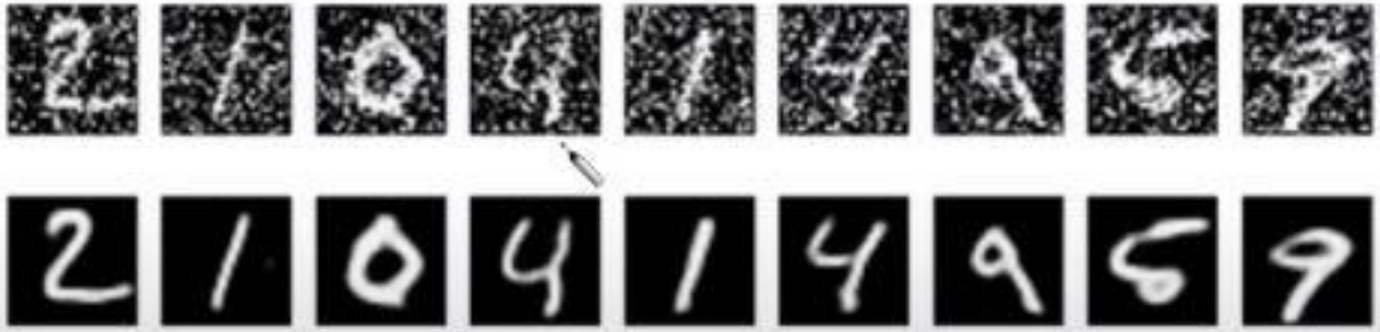
Other usage of Auto-Encoders

Analogously, a **CAE stack** (CAEs) can be used to **initialize a CNN** with identical topology prior to a supervised training stage.

Example
Apply 2-Layers Conv Auto-Encoder on NOISY Mnist dataset



Example
Apply 2-Layers Conv Auto-Encoder on NOISY Mnist dataset



Visualize Mnist Dataset using CAE

(represent each digit with 2 values)

Similar digits shapes have similar representations