Deep Belief Nets
and
Stacked Denoising Autoencoders

Smart Computing

智慧計算實驗室

SAIC

Outline

- Neural Networks Fall
- Deep Learning's Evolution
- Deep Learning with Pre-training
- Restricted Boltzmann Machines
- Deep Belief Nets (DBNs)
- Denoising Autoencoders
- Stacked Denoising Autoencoders (SDA)
- Summary

Neural Networks Fall

- Nonlinear problems can be learned and solved by inserting a hidden layer between the input and output layer
  - More layer, more pattern to express
- Theoretically, neural networks can approximate any function
  - Ignore time cost and over-fitting problem

But NNs didn't Work Well

- Some cases even have less accuracy!
- Backpropagation problem
  - An error is reversed in each layer
  - The weight of the network is adjusted at each layer in order (from output to input)
  - The error gradually disappears every time it backpropagates layers
  - Vanishing gradient problem!!

Outline

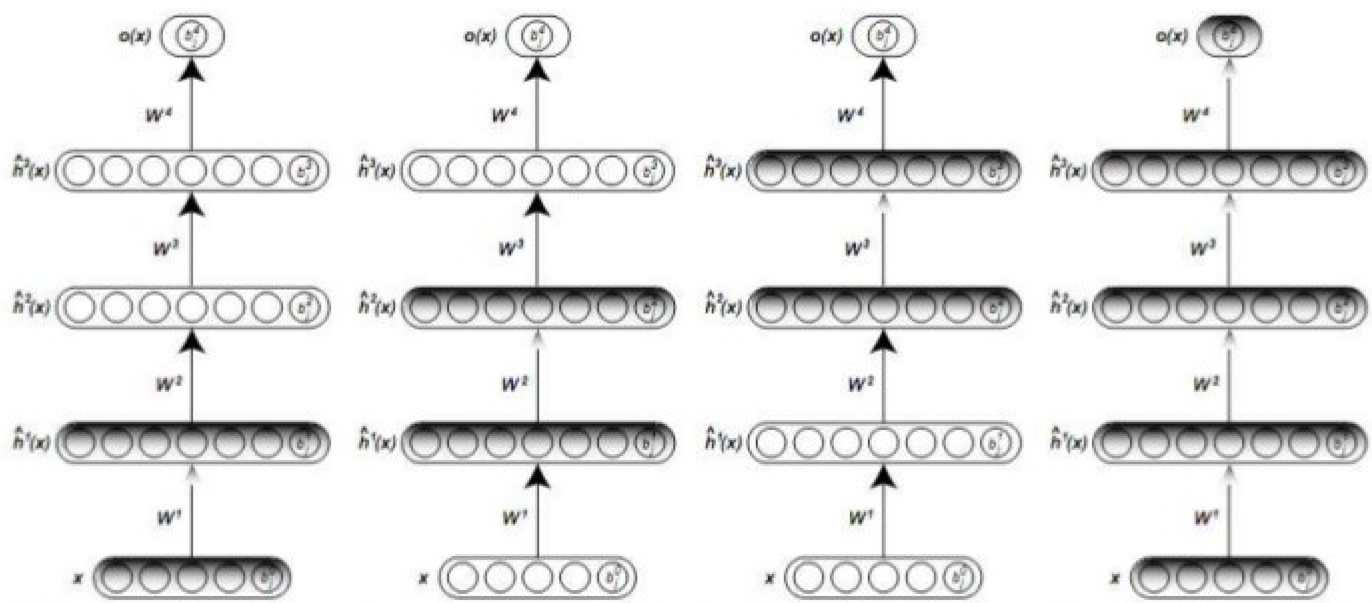
- Neural Networks Fall
- Deep Learning's Evolution
- Deep Learning with Pre-training
- Restricted Boltzmann Machines
- Deep Belief Nets (DBNs)
- Denoising Autoencoders
- Stacked Denoising Autoencoders (SDA)
- Summary

Deep Learning’s Evolution

- There are two algorithms that triggered deep learning's popularity
  - DBN by Hinton
    - https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf
  - SDA by Vincent et al.
    - http://www.iro.umontreal.ca/~vincentp/Publications/denoising_autoencoders_tr1316.pdf
- So, what is the common approach that solved the vanishing gradient problem?

Simple and Elegant Solution

- DBN and SDA use layer-wise training to solve vanishing gradient problem
  - Each layer adjusts the weights of the networks independently

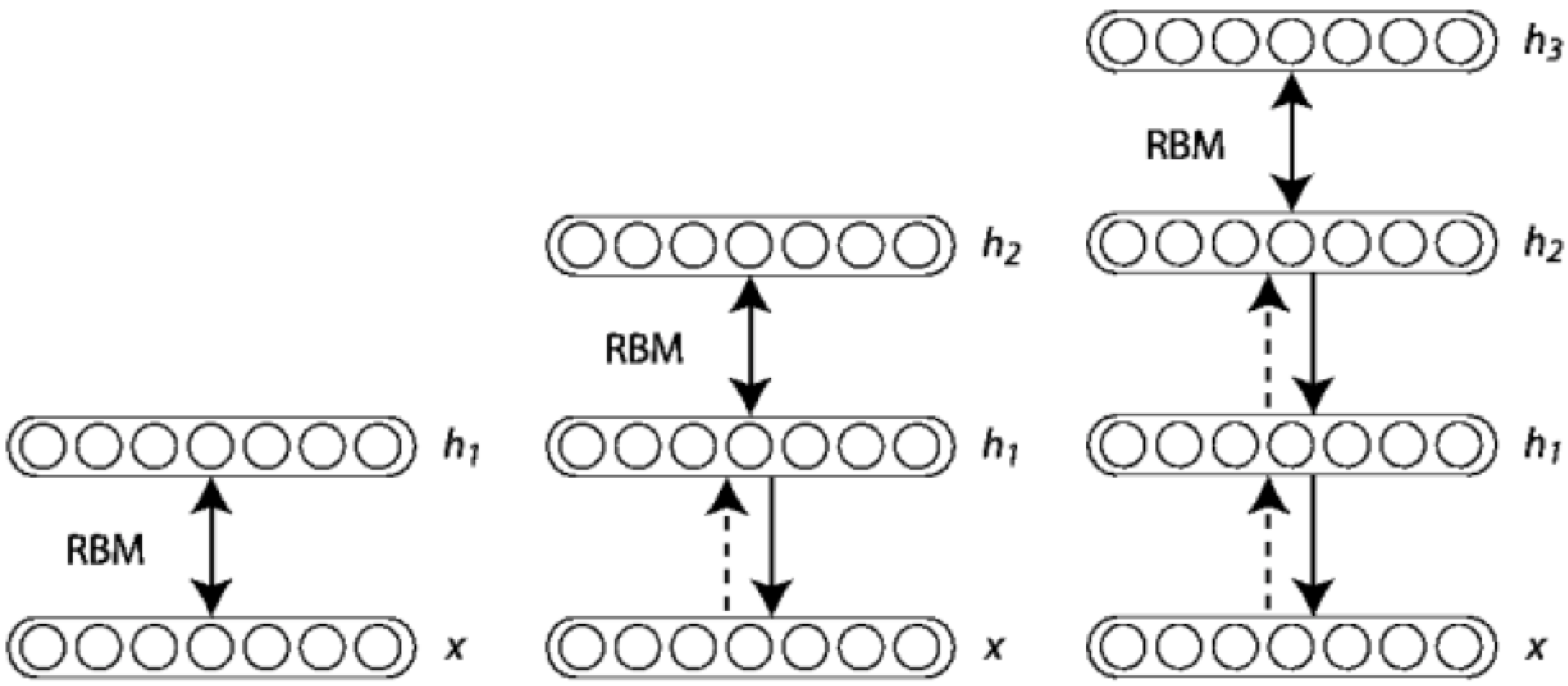


(a) First hidden layer pre-training (b) Second hidden layer pre-training (c) Third hidden layer pre-training (d) Fine-tuning of whole network

Pre-training and Fine-tuning

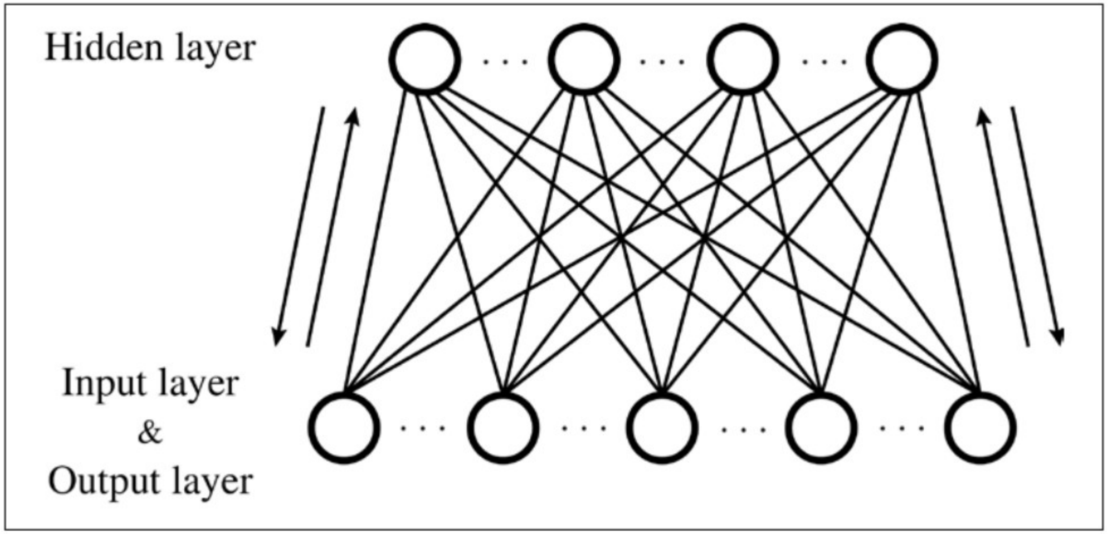
- This phase of layer-wise training is called pre-training
- The last adjustment phase is called fine-tuning
- The problem: if both layers are hidden (neither of the layers are input nor output layers), then how is the training done?

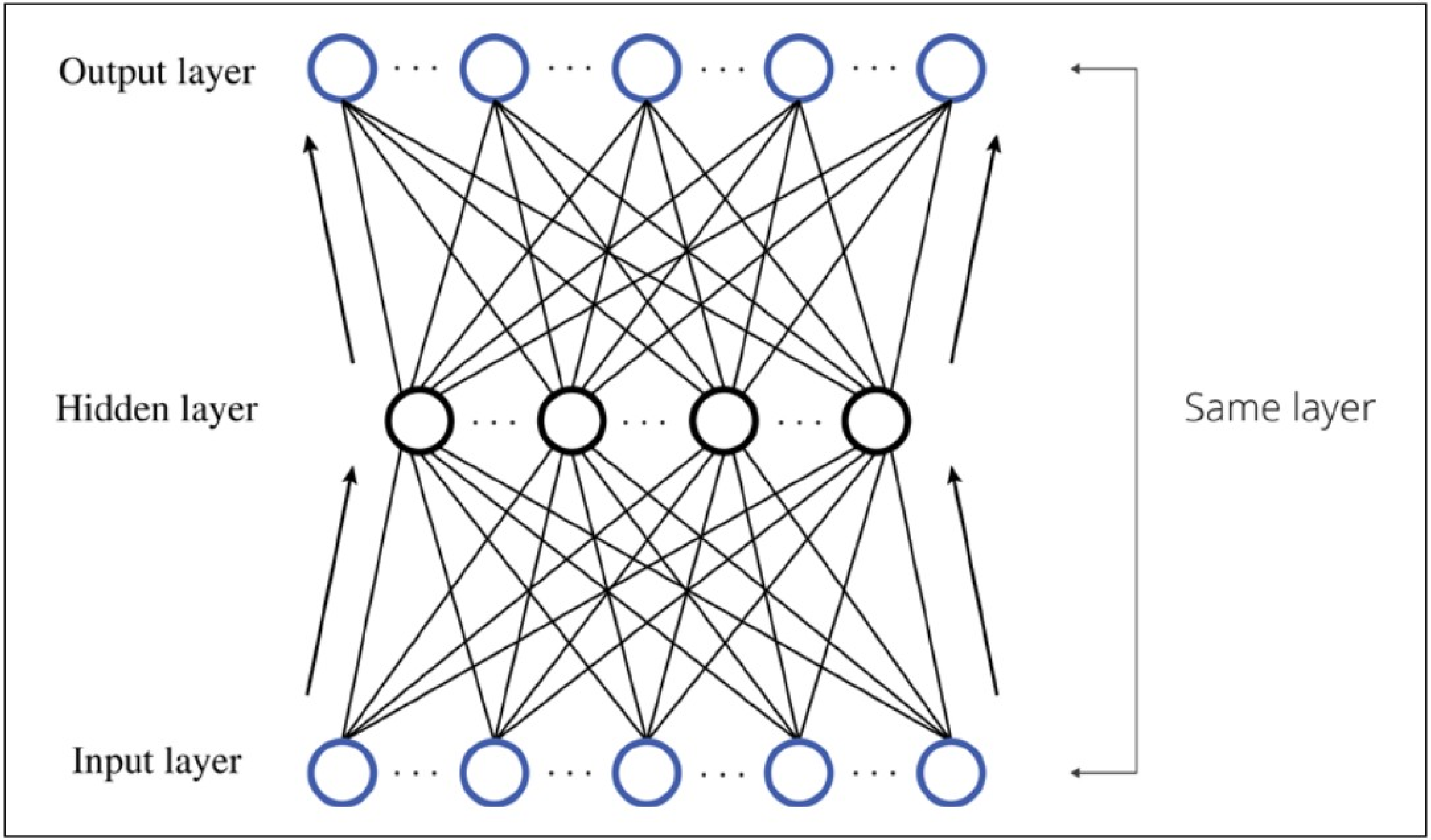
Piled Up Layers in Deep Structure



Learning Feature

- Features are learned from the input data in stages (and semi-automatically)
  - Where the deeper a layer becomes, the higher the feature it learns
  - “a machine can learn a concept”




Output layer
Hidden layer
Input layer
Same layer

In NN...

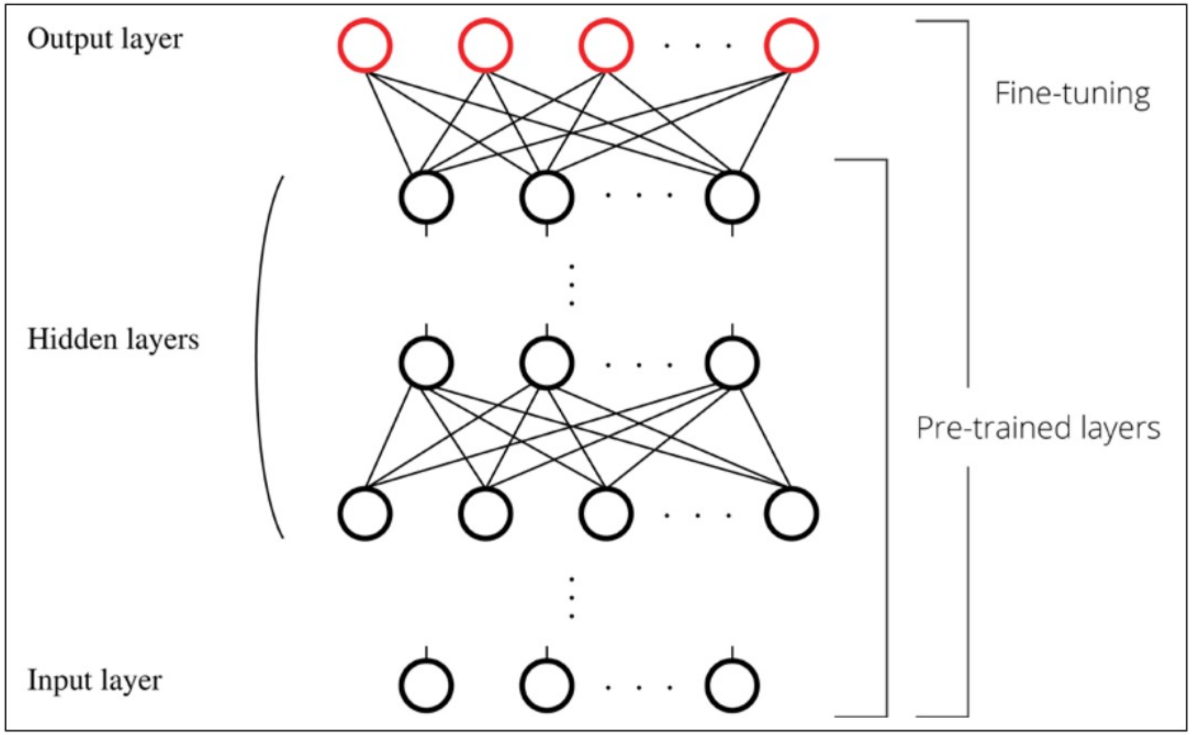
- Learning intends to minimize errors between the model's prediction output and the dataset output
  - Method: remove an error by finding a pattern from the input data and making data with a common pattern the same output value (for example, 0 or 1)
- What would then happen if we turned the output value into the input value?
  - The weight of networks should be adjusted to focus more on the part that reflects the common features!!

Pre-training

- The layer after the pre-training can be treated as normal feed-forward neural networks where the weight of the networks is adjusted
- After pre-training, features learned, and?
  - Pre-training is unsupervised training
  - Doesn't solve the classification problem
  - Fine-tuning to solve

Fine-tuning

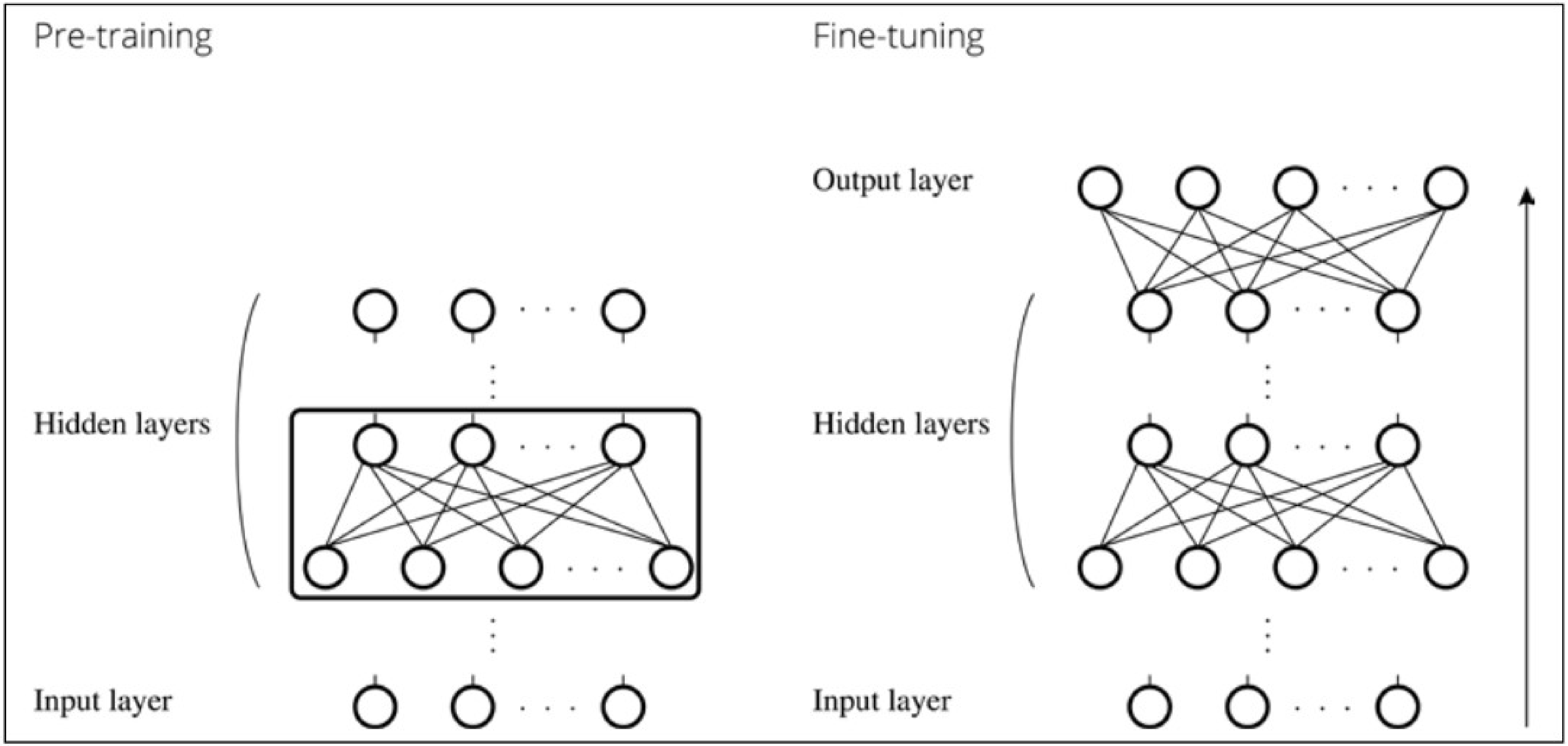
- The main roles of fine-tuning
  - Add an output layer that completed pre-training and to perform supervised training
  - Do final adjustments



Fine-tuning

- Normally, the weights of whole networks, including the weights adjusted in pre-training, will also be adjusted
  - Deep neural networks as one multi-layer neural network
  - Doesn't the vanishing gradient problem occur?
  - Once the pre-training is done, the learning starts from the network almost already adjusted
    - Proper error can be propagated to a layer close to an input layer

Pre-training with Fine-tuning

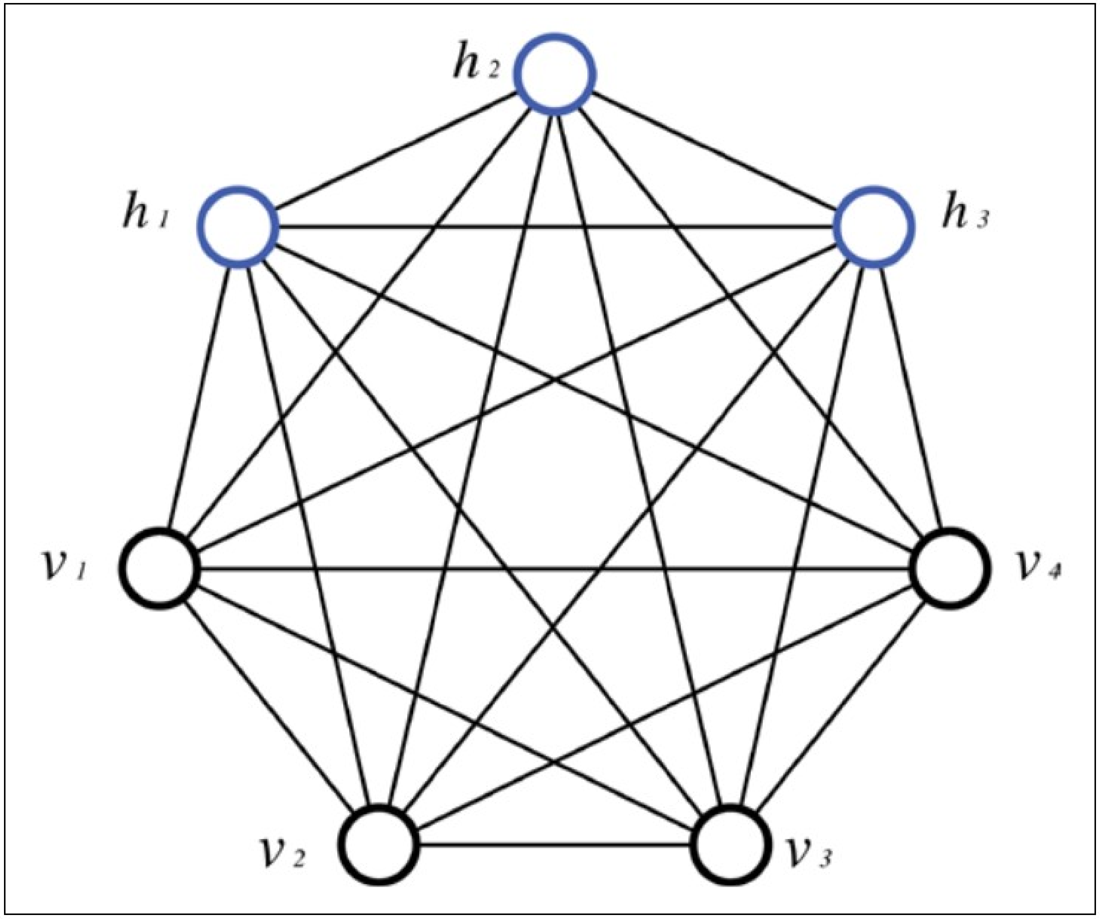


Outline

- Neural Networks Fall
- Deep Learning's Evolution
- Deep Learning with Pre-training
- Restricted Boltzmann Machines
- Deep Belief Nets (DBNs)
- Denoising Autoencoders
- Stacked Denoising Autoencoders (SDA)
- Summary

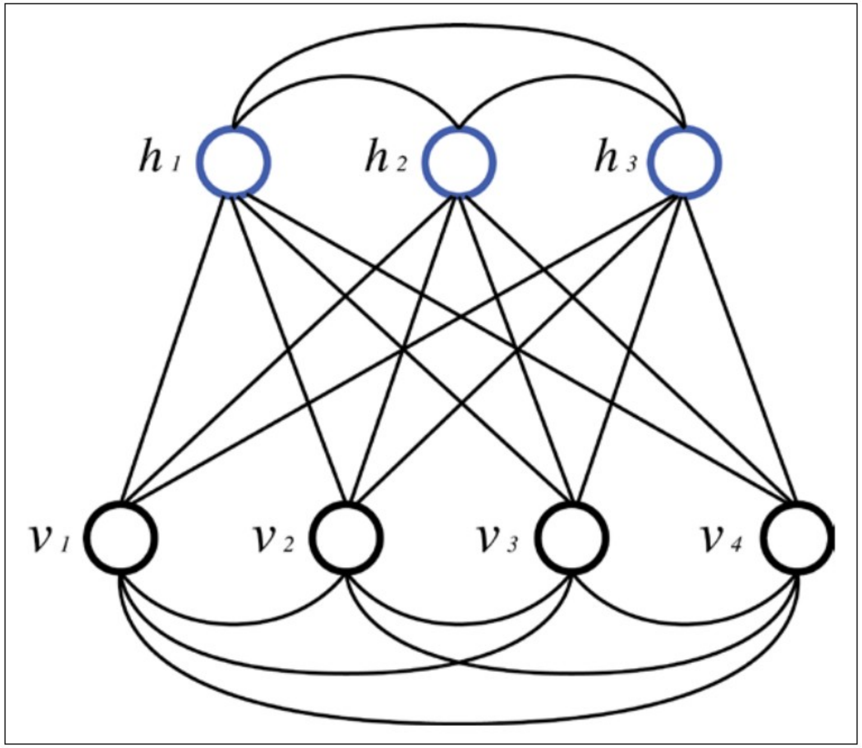
Restricted Boltzmann Machines

- Restricted Boltzmann Machines, RBM
- Boltzmann Machines, BM



Boltzmann Machines, BM

- The feature is to adopt the concept of energy in neural networks
- Fully connected, take an enormous amount of calculation time



RBM

- RBM with binary inputs is sometimes called Bernoulli RBM
- RBM is the energy-based model
  - Visible to hidden units

$$p\left(h_j = 1 \mid v\right) = \sigma\left(\sum_{i=1}^{D} w_{ij} v_i + c_j\right)$$

  - Hidden to visible units

$$p\left(v_i = 1 \mid h\right) = \sigma\left(\sum_{j=1}^{M} w_{ij} h_j + b_i\right)$$

Energy Function in RBM

$$E(v,h) = -b^T v - c^T h - h^T W v$$

$$= -\sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{j=1}^{M} \sum_{i=1}^{D} h_j w_{ij} v_i$$

- Joint probability density function

$$p(v,h) = \frac{1}{Z} \exp\left(-E(v,h)\right)$$

$$Z = \sum_{v,h} \exp\left(-E(v,h)\right)$$

$$p(v \mid \theta) = \sum_{h} P(v,h) = \frac{1}{Z} = \sum_{h} \exp\left(-E(v,h)\right)$$

Log Likelihood

$$
\begin{aligned}
In\, L(\theta \mid v) &= In\, p(v \mid \theta) \\
&= In \frac{1}{Z} \sum_h \exp\left(-E(v,h)\right) \\
&= In \frac{1}{Z} \sum_h \exp\left(-E(v,h)\right) - In \sum_{v,h} \exp\left(-E(v,h)\right)
\end{aligned}
$$

Smart Computing 智慧計算實驗室 SAIC

Gradient

$$\frac{\partial In\, L(\theta \mid v)}{\partial \theta} = \frac{\partial}{\partial \theta}\left(In \sum_{h} \exp\left(-E(v,h)\right)\right) - \frac{\partial}{\partial \theta}\left(In \sum_{v,h} \exp\left(-E(v,h)\right)\right)$$

$$= -\frac{1}{\sum_{h} \exp\left(-E(v,h)\right)} \sum_{h} \exp\left(-E(v,h)\right) \frac{\partial E(v,h)}{\partial \theta}$$

$$= +\frac{1}{\sum_{h} \exp\left(-E(v,h)\right)} \sum_{v,h} \exp\left(-E(v,h)\right) \frac{\partial E(v,h)}{\partial \theta}$$

$$= -\sum_{h} p(h \mid v) \frac{\partial E(v,h)}{\partial \theta} + \sum_{v,h} p(v \mid h) \frac{\partial E(v,h)}{\partial \theta}$$

Gradient of each Parameter

$$\frac{\partial In\, L(\theta \mid v)}{\partial w_{ij}} = \sum_h p(h \mid v) \frac{\partial E(v,h)}{\partial w_{ij}} + \sum_{v,h} p(h \mid v) \frac{\partial E(v,h)}{\partial w_{ij}}$$

$$= \sum_h p(h \mid v) h_j v_i - \sum_v p(v) \sum_h p(h \mid v) h_j v_i$$

$$= p(H_j = 1 \mid v) v_i - \sum_v p(v) p(H_j = 1 \mid v) v_i$$

$$\frac{\partial In\, L(\theta \mid v)}{\partial b_i} = v_i - \sum_v p(v) v_i$$

$$\frac{\partial In\, L(\theta \mid v)}{\partial c_j} = p(H_j = 1 \mid v) - \sum_v p(v) p(H_j = 1 \mid v)$$

Problem on Gradient

- Problem occurs: calculation of the probability distribution for all the {0, 1} patterns
  - Can't be solve within a realistic time
- Contrastive Divergence (CD)
  - The method for approximating data using Gibbs sampling

Contrastive Divergence

Here, $v^{(0)}$ is an input vector. Also, $v^{(k)}$ is an input (output) vector that can be obtained by sampling for k-times using this input vector.

Then, we get:

$$h_j^{(k)} \sim p\left(h_j \mid v^{(k)}\right)$$

$$h_i^{(k+1)} \sim p\left(v_i \mid h^{(k)}\right)$$

$$\frac{\partial In\, L(\theta \mid v)}{\partial \theta} = -\sum_{h} p(h \mid v) \frac{\partial E(v,h)}{\partial \theta} + \sum_{v,h} p(v,h) \frac{\partial E(v,h)}{\partial \theta}$$

$$\approx -\sum_{h} p\left(h \mid v^{(0)}\right) \frac{\partial E(v,h)}{\partial \theta} \sum_{v,h} p\left(h, v^{(k)}\right) \frac{\partial E\left(v^{(k)},h\right)}{\partial \theta}$$

$$w_{ij}^{(\tau+1)} = w_{ij}^{(\tau)} + \eta\left(p\left(H_j = 1 \mid v^{(0)}\right) v_i^{(0)} - p\left(H_j = 1 \mid v^{k}\right) v_i^{k} \right)$$

$$b_i^{(\tau+1)} = b_i^{(\tau)} + \eta\left(v_i^{(0)} - v_i^{k} \right)$$

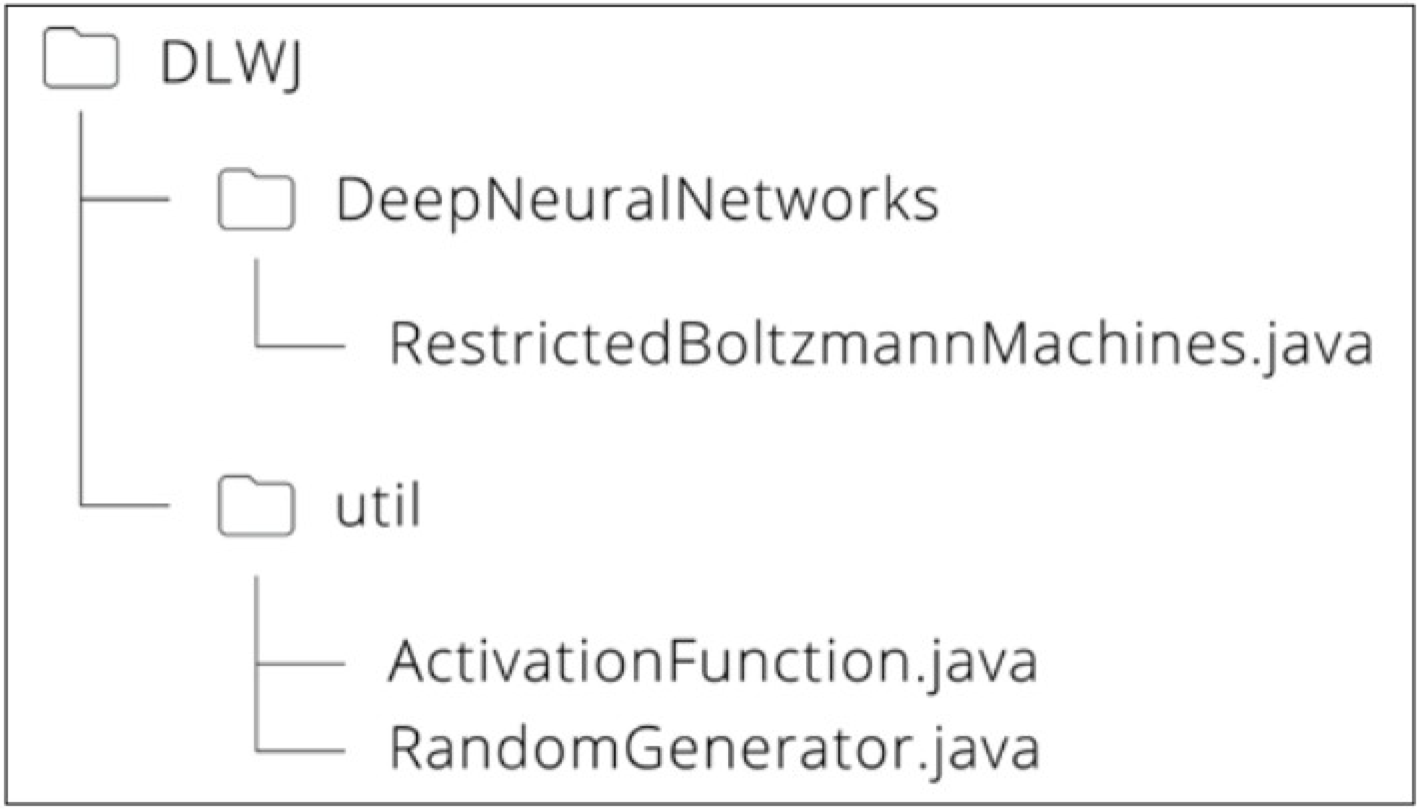
$$c_j^{(\tau+1)} = c_j^{(\tau)} + \eta\left(p\left(H_j = 1 \mid v^{(0)}\right) - p\left(H_j = 1 \mid v^{k}\right) \right)$$

τ is the number of iterations and η is the learning rate

Contrastive Divergence

- CD that performs sampling k-times is shown as CD-k
- It's known that CD-1 is sufficient when applying the algorithm to realistic problems

RBM Implementation



Outline

- Neural Networks Fall
- Deep Learning's Evolution
- Deep Learning with Pre-training
- Restricted Boltzmann Machines
- Deep Belief Nets (DBNs)
- Denoising Autoencoders
- Stacked Denoising Autoencoders (SDA)
- Summary

Deep Belief Nets (DBNs)

- Program flow
  - Setting up parameters for the model
  - Building the model
  - Pre-training the model
  - Fine-tuning the model
  - Testing and evaluating the model

```
DLWJ
├── DeepNeuralNetworks
│   ├── DeepBeliefNets.java
│   └── RestrictedBoltzmannMachines.java
├── MultiLayerNeuralNetworks
│   └── HiddenLayer.java
├── SingleLayerNeuralNetworks
│   └── LogisticRegression.java
└── util
    └── RandomGenerator.java
```

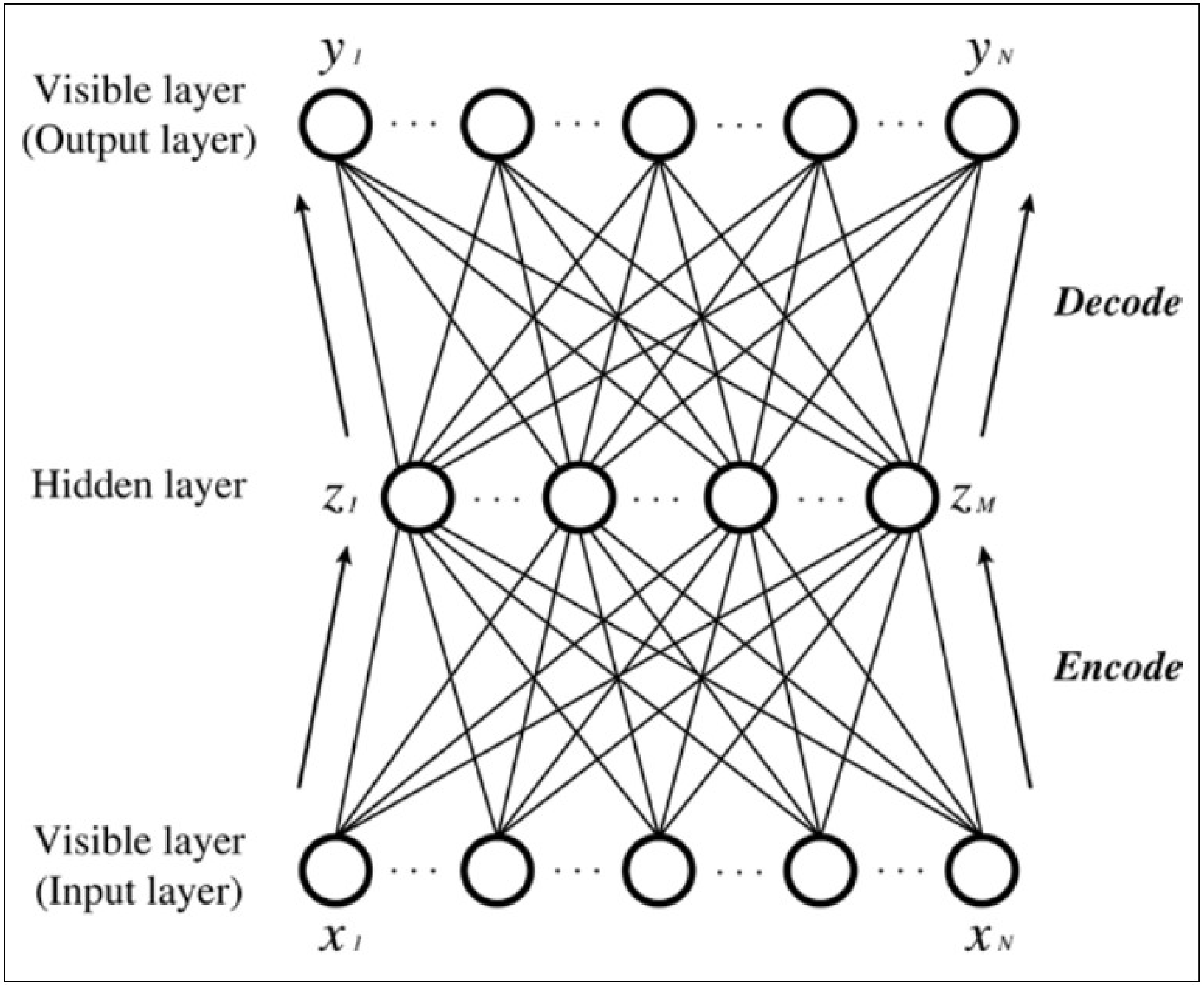
Outline

- Neural Networks Fall
- Deep Learning's Evolution
- Deep Learning with Pre-training
- Restricted Boltzmann Machines
- Deep Belief Nets (DBNs)
- Denoising Autoencoders
- Stacked Denoising Autoencoders (SDA)
- Summary

Denoising Autoencoders

- The method used in pre-training for SDA is called Denoising Autoencoders (DA)
  - DA is the method that emphasizes the role of equating inputs and outputs
- DA processing
  - adds some noise to input data intentionally
  - Data is partially damaged
  - DA performs learning as it restores corrupted data

Denoising Autoencoders



Denoising Autoencoders

$$z_j = \sigma\left(\sum_{i=1}^{N} w_{ij}\tilde{x}_i + c_j\right)$$

$$y_i = \sigma\left(\sum_{i=1}^{M} w_{ji} z_j + b_i\right)$$

$\tilde{x}$ is the corrupted data, the input data with noise

Evaluation Function of DA

$$E := -In\,L(\theta) = -\sum_{i=1}^{N}\left\{x_i In\, y_i + (1-x_i)\, In(1-y_i)\right\}$$

- Gradient

$$h_j := \sum_{i=1}^{N} w_{ji}\tilde{x}_i + c_j \qquad z_j = \sigma(h_j)$$

$$g_i := \sum_{j=1}^{M} w_{ji} z_j + b_i \qquad y_i = \sigma(g_i)$$

Smart Computing 智慧計算實驗室 SAIC

Therefore, only two terms are required. Let's derive them one by one:

$$\frac{\partial E}{\partial h_j} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial h_j} = \frac{\partial E}{\partial z_j} z_j \left(1 - z_j\right)$$

Here, we utilized the derivative of the `sigmoid` function:

$$\frac{d}{dx}\sigma(x) = \sigma(x)\left(1 - \sigma(x)\right)$$

Also, we get:

$$\frac{\partial E}{\partial z_j} = \sum_{i=1}^{N} \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_j}$$

$$= \sum_{i=1}^{N} w_{ji}\left(x_i - y_i\right)$$

$$\frac{\partial E}{\partial h_j}=\left(\sum_{i=1}^{N}w_{ji}\left(x_i-y_i\right)\right)z_j\left(1-z_j\right)$$

$$\frac{\partial E}{\partial g_i}=\frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial g_i}$$

$$=x_i-y_i$$

$$w_{ji}^{(k+1)}=w_{ji}^{(k)}+\eta\left[\left(\sum_{i=1}^{N}w_{ji}^{(k)}\left(x_i-y_i\right)\right)z_j\left(1-z_j\right)\tilde{x}_i+\left(x_i-y_i\right)z_j\right]$$

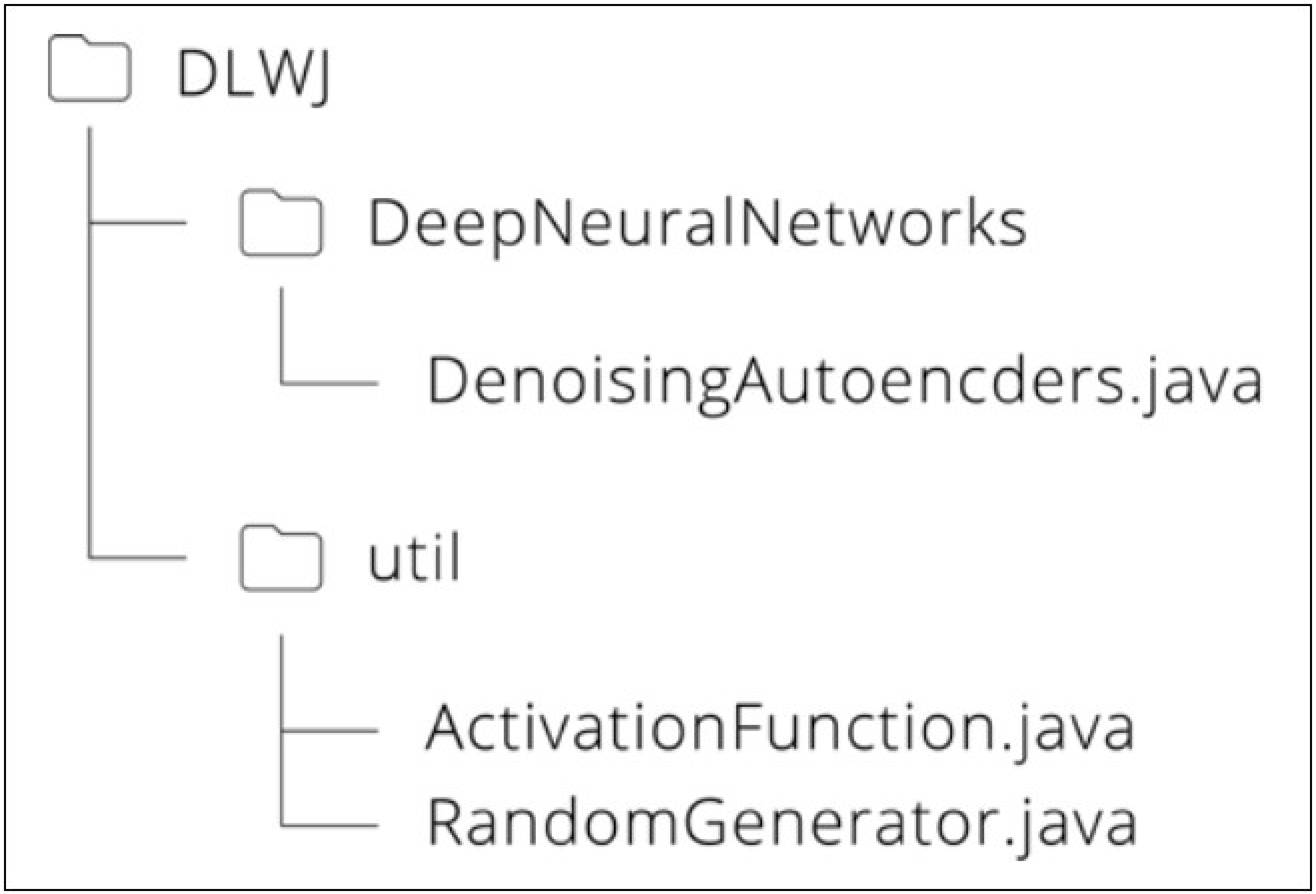
$$b_i^{(k+1)}=b_i^{k}+\eta\left(x_i-y_i\right)$$

$$c_j^{(k+1)}=c_j^{(k)}+\eta\left(\sum_{i=1}^{N}w_{ji}^{(k)}\left(x_i-y_i\right)\right)z_j\left(1-z_j\right)$$

智慧計算實驗室 SAIC Smart Computing

k is the number of iterations and η is the learning rate

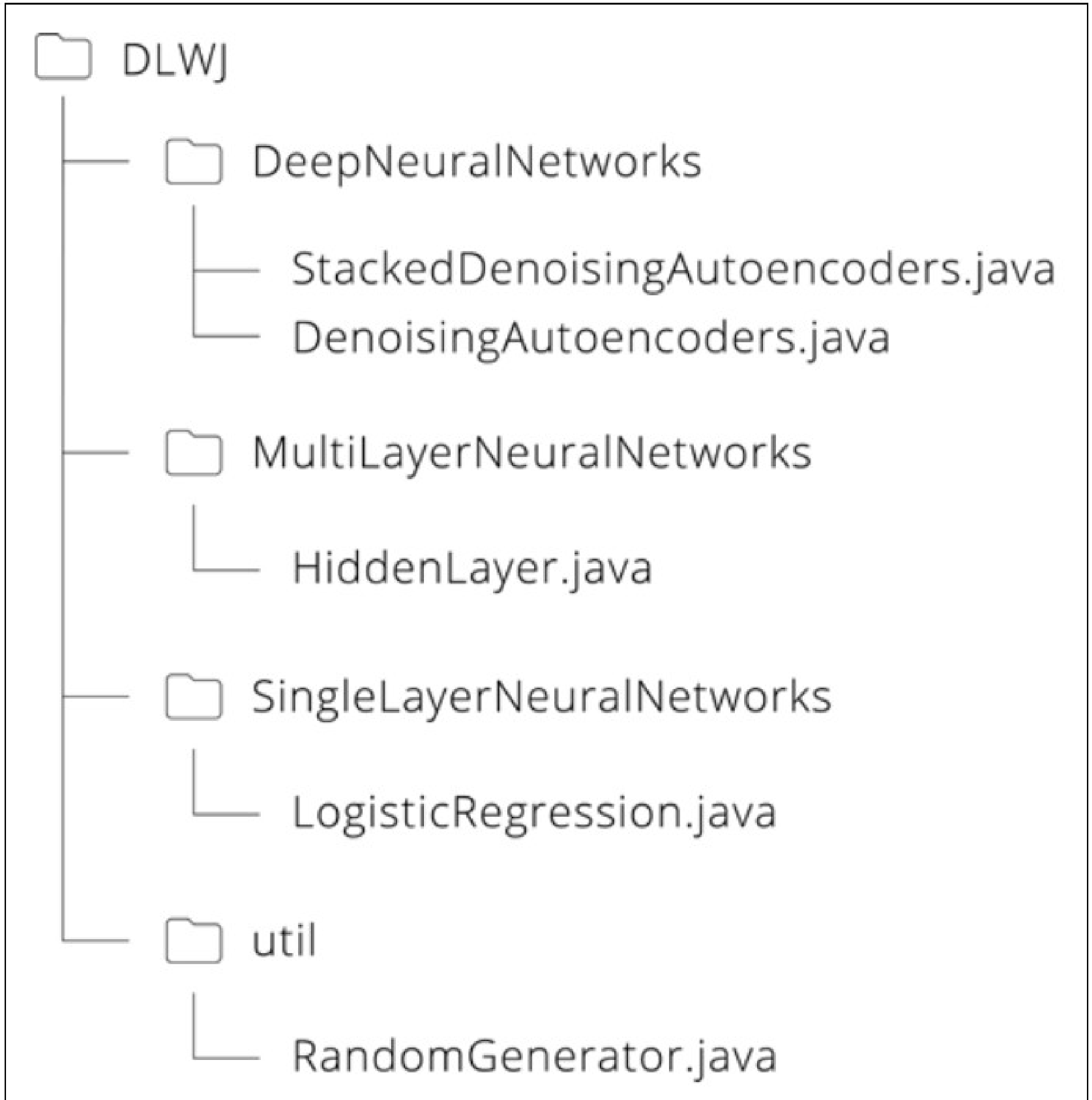
DA Implementation



Outline

- Neural Networks Fall
- Deep Learning's Evolution
- Deep Learning with Pre-training
- Restricted Boltzmann Machines
- Deep Belief Nets (DBNs)
- Denoising Autoencoders
- Stacked Denoising Autoencoders (SDA)
- Summary

Stacked Denoising Autoencoders



Outline

- Neural Networks Fall
- Deep Learning's Evolution
- Deep Learning with Pre-training
- Restricted Boltzmann Machines
- Deep Belief Nets (DBNs)
- Denoising Autoencoders
- Stacked Denoising Autoencoders (SDA)
- Summary

Summary

- The problem of the previous neural networks algorithm
- The breakthrough of deep learning
- RBM, Restricted Boltzmann Machines
- DBN with RBM
- DA, Denoising Autoencoders
- SDA with DA