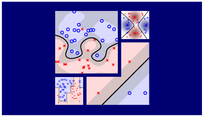
Machine Learning Techniques
(機器學習技法)



Lecture 13: Deep Learning

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering
National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

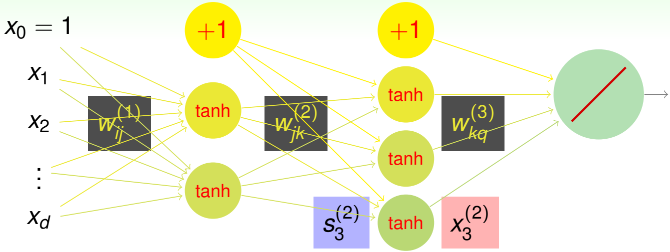
Lecture 12: Neural Network

automatic **pattern feature extraction** from **layers of neurons** with **backprop** for GD/SGD

Lecture 13: Deep Learning

- Deep Neural Network
- Autoencoder
- Denoising Autoencoder
- Principal Component Analysis

Physical Interpretation of NNet Revisited



- each layer: **pattern feature extracted** from data, **remember? :-)**
- how many neurons? how many layers?
  —more generally, **what structure?**
  - subjectively, **your design!**
  - objectively, **validation, maybe?**

structural decisions:
key issue for applying NNet

Shallow versus Deep Neural Networks

shallow: few (hidden) layers; deep: many layers

Shallow NNet

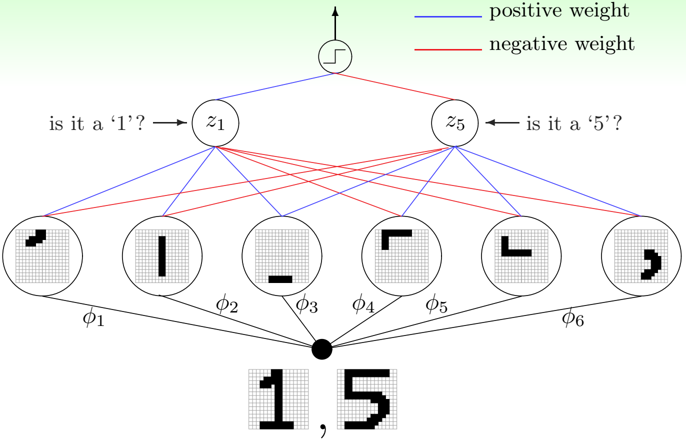
- more **efficient** to train (○)
- **simpler** structural decisions (○)
- theoretically **powerful enough** (○)

Deep NNet

- **challenging** to train ($\times$)
- **sophisticated** structural decisions ($\times$)
- **'arbitrarily' powerful** (○)
- more **'meaningful'?** (see next slide)

deep NNet (**deep learning**)
gaining attention in recent years

Meaningfulness of Deep Learning



- **'less burden'** for each layer: **simple** to **complex** features
- natural for **difficult** learning task with **raw features**, like **vision**

deep NNet: currently popular in **vision/speech/**...

Challenges and Key Techniques for Deep Learning

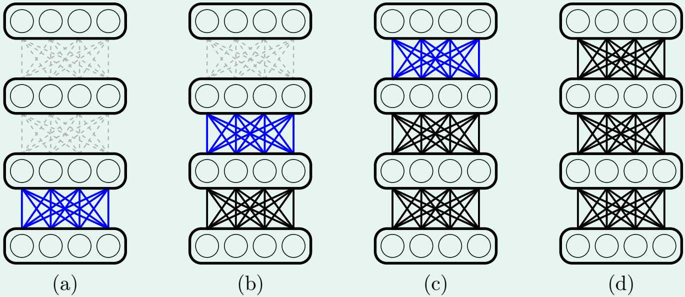
- difficult **structural decisions**:
  - subjective with **domain knowledge**: like **convolutional NNet** for images
- high **model complexity**:
  - no big worries if **big enough data**
  - **regularization** towards noise-tolerant: like
    - **dropout** (tolerant when network corrupted)
    - **denoising** (tolerant when input corrupted)
- hard **optimization problem**:
  - **careful initialization** to avoid bad local minimum: called **pre-training**
- huge **computational complexity** (worsen with **big data**):
  - novel hardware/architecture: like **mini-batch with GPU**

IMHO, careful **regularization** and **initialization** are key techniques

A Two-Step Deep Learning Framework

Simple Deep Learning

1. for $\ell = 1, \ldots, L$, **pre-train** $\left\{w_{ij}^{(\ell)}\right\}$ assuming $w_*^{(1)}, \ldots w_*^{(\ell-1)}$ fixed



2. **train with backprop** on **pre-trained** NNet to **fine-tune** all $\left\{w_{ij}^{(\ell)}\right\}$

will focus on **simplest** **pre-training** technique along with **regularization**

Fun Time

For a deep NNet for written character recognition from raw pixels, which type of features are more likely extracted after the first hidden layer?

1. pixels
2. stokes
3. parts
4. digits

Fun Time

For a deep NNet for written character recognition from raw pixels, which type of features are more likely extracted after the first hidden layer?

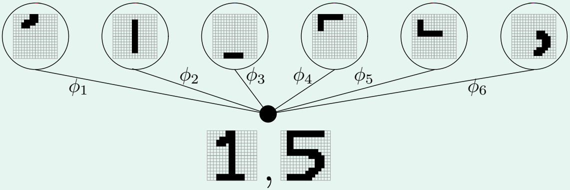
1. pixels
2. stokes
3. parts
4. digits

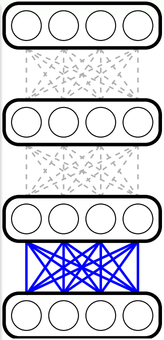
Reference Answer: 2

Simple strokes are likely the 'next-level' features that can be extracted from raw pixels.

Information-Preserving Encoding

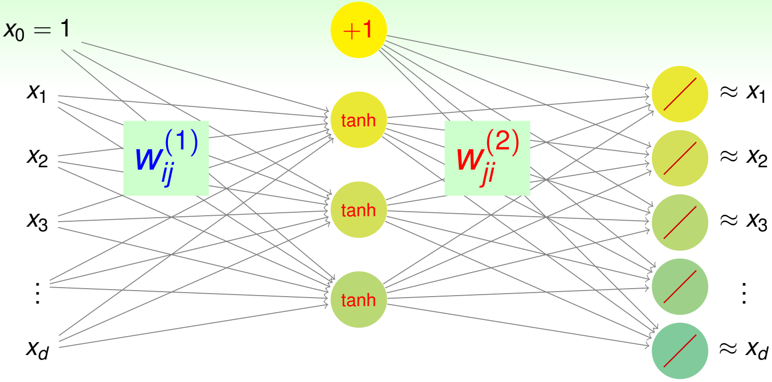
- **weights**: feature transform, i.e. **encoding**
- **good weights**: information-preserving encoding
  —next layer same info. with different representation
- **information-preserving**:
  decode accurately after encoding





idea: **pre-train weights** towards
information-preserving encoding

Information-Preserving Neural Network



- **autoencoder**: d—$\tilde{d}$—d NNet with goal $g_i(\mathbf{x}) \approx x_i$
  —learning to **approximate identity function**
- $w_{ij}^{(1)}$: encoding weights; $w_{ji}^{(2)}$: decoding weights

why **approximating identity function**?

Usefulness of Approximating Identity Function

if $\mathbf{g}(\mathbf{x}) \approx \mathbf{x}$ using some **hidden** structures on the **observed data $\mathbf{x}_n$**

- for supervised learning:
  - hidden structure (essence) of $\mathbf{x}$ can be used as reasonable transform $\boldsymbol{\Phi}(\mathbf{x})$

—learning **'informative' representation** of data

- for unsupervised learning:
  - density estimation: larger (structure match) when $\mathbf{g}(\mathbf{x}) \approx \mathbf{x}$
  - outlier detection: those $\mathbf{x}$ where $\mathbf{g}(\mathbf{x}) \not\approx \mathbf{x}$

—learning **'typical' representation** of data

autoencoder:

representation-learning through approximating identity function

Basic Autoencoder

basic **autoencoder**:

d—$\tilde{d}$—d NNet with error function $\sum_{i=1}^{d}(g_i(\mathbf{x}) - x_i)^2$

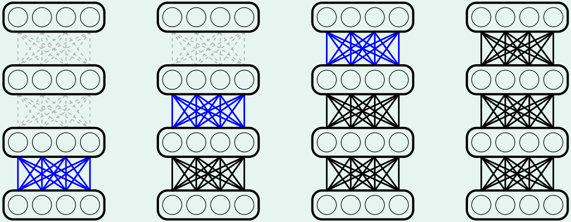
- backprop easily applies; **shallow** and easy to train
- usually $\tilde{d} < d$: **compressed** representation
- data: $\{(\mathbf{x}_1, \mathbf{y}_1 = \mathbf{x}_1), (\mathbf{x}_2, \mathbf{y}_2 = \mathbf{x}_2), \ldots, (\mathbf{x}_N, \mathbf{y}_N = \mathbf{x}_N)\}$
  —often categorized as **unsupervised learning technique**
- sometimes constrain $w_{ij}^{(1)} = w_{ji}^{(2)}$ as **regularization**
  —more **sophisticated** in calculating gradient

basic **autoencoder** in basic deep learning:
$\left\{w_{ij}^{(1)}\right\}$ taken as shallowly pre-trained weights

Pre-Training with Autoencoders

Deep Learning with Autoencoders

1. for $\ell = 1, \ldots, L$, **pre-train** $\left\{w_{ij}^{(\ell)}\right\}$ assuming $w_*^{(1)}, \ldots w_*^{(\ell-1)}$ fixed



(a) (b) (c) (d)

by **training basic autoencoder on** $\left\{\mathbf{x}_n^{(\ell-1)}\right\}$ **with** $\tilde{d} = d^{(\ell)}$

2. **train with backprop** on **pre-trained** NNet to **fine-tune** all $\left\{w_{ij}^{(\ell)}\right\}$

many successful **pre-training** techniques take **'fancier' autoencoders** with different **architectures** and **regularization schemes**

Fun Time

Suppose training a d-$\tilde{d}$-d autoencoder with backprop takes approximately $c \cdot d \cdot \tilde{d}$ seconds. Then, what is the total number of seconds needed for pre-training a d-$d^{(1)}$-$d^{(2)}$-$d^{(3)}$-1 deep NNet?

1. $c\left(d + d^{(1)} + d^{(2)} + d^{(3)} + 1\right)$
2. $c\left(d \cdot d^{(1)} \cdot d^{(2)} \cdot d^{(3)} \cdot 1\right)$
3. $c\left(dd^{(1)} + d^{(1)}d^{(2)} + d^{(2)}d^{(3)} + d^{(3)}\right)$
4. $c\left(dd^{(1)} \cdot d^{(1)}d^{(2)} \cdot d^{(2)}d^{(3)} \cdot d^{(3)}\right)$

Fun Time

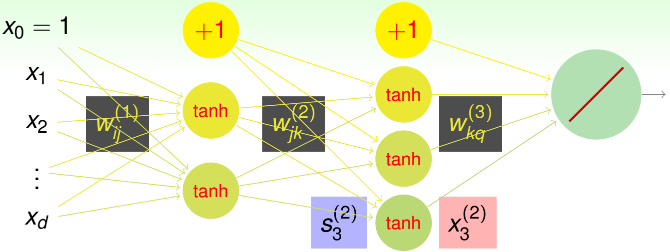
Suppose training a d-$\tilde{d}$-d autoencoder with backprop takes approximately $c \cdot d \cdot \tilde{d}$ seconds. Then, what is the total number of seconds needed for pre-training a d-$d^{(1)}$-$d^{(2)}$-$d^{(3)}$-1 deep NNet?

1. $c\left(d + d^{(1)} + d^{(2)} + d^{(3)} + 1\right)$
2. $c\left(d \cdot d^{(1)} \cdot d^{(2)} \cdot d^{(3)} \cdot 1\right)$
3. $c\left(dd^{(1)} + d^{(1)}d^{(2)} + d^{(2)}d^{(3)} + d^{(3)}\right)$
4. $c\left(dd^{(1)} \cdot d^{(1)}d^{(2)} \cdot d^{(2)}d^{(3)} \cdot d^{(3)}\right)$

Reference Answer: 3

Each $c \cdot d^{(\ell-1)} \cdot d^{(\ell)}$ represents the time for pre-training with one autoencoder to determine one layer of the weights.

Regularization in Deep Learning



watch out for overfitting, remember? :-)

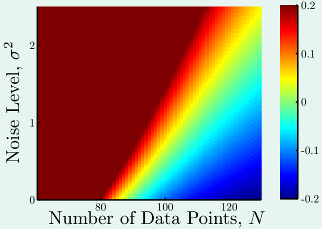
high **model complexity**: **regularization** needed

- structural decisions/**constraints**
- weight decay or weight elimination **regularizers**
- **early stopping**

next: another **regularization** technique

Reasons of Overfitting Revisited

stochastic noise



reasons of serious overfitting:

data size $N \downarrow$	overfit $\uparrow$
noise $\uparrow$	overfit $\uparrow$
excessive power $\uparrow$	overfit $\uparrow$

how to deal with **noise**?

Dealing with Noise

- direct possibility: **data cleaning/pruning, remember? :-)**
- a **wild** possibility: **adding noise** to data?

- idea: **robust** autoencoder should not only let $\mathbf{g}(\mathbf{x}) \approx \mathbf{x}$ but also allow $\mathbf{g}(\tilde{\mathbf{x}}) \approx \mathbf{x}$ even when $\tilde{\mathbf{x}}$ slightly different from $\mathbf{x}$
- **denoising** autoencoder:

  run basic autoencoder with data
  $$\{(\tilde{\mathbf{x}}_1, \mathbf{y}_1 = \mathbf{x}_1), (\tilde{\mathbf{x}}_2, \mathbf{y}_2 = \mathbf{x}_2), \ldots, (\tilde{\mathbf{x}}_N, \mathbf{y}_N = \mathbf{x}_N)\},$$
  where $\tilde{\mathbf{x}}_n = \mathbf{x}_n +$ **artificial noise**

  —often used **instead of basic autoencoder** in deep learning
- useful for data/image processing: $\mathbf{g}(\tilde{\mathbf{x}})$ a **denoised** version of $\tilde{\mathbf{x}}$
- effect: 'constrain/**regularize**' $\mathbf{g}$ towards **noise-tolerant** denoising

artificial noise/**hint** as **regularization**!
—practically also useful for other NNet/models

Fun Time

Which of the following cannot be viewed as a regularization technique?

1. hint the model with artificially-generated noisy data
2. stop gradient descent early
3. add a weight elimination regularizer
4. all the above are regularization techniques

Fun Time

Which of the following cannot be viewed as a regularization technique?

1. hint the model with artificially-generated noisy data
2. stop gradient descent early
3. add a weight elimination regularizer
4. all the above are regularization techniques

Reference Answer: 4

1 is our new friend for regularization, while 2 and 3 are old friends.

Linear Autoencoder Hypothesis

nonlinear autoencoder	linear autoencoder
sophisticated	simple

linear: more efficient? less overfitting? **linear first, remember? :-)**

linear hypothesis for k-th component $h_k(\mathbf{x}) = \sum_{j=0}^{\tilde{d}} w_{kj} \left(\sum_{i=1}^{d} w_{ij} x_i \right)$

consider three special conditions:

- **exclude** x_0: range of i **same** as range of k
- constrain $w_{ij}^{(1)} = w_{ji}^{(2)} = w_{ij}$: **regularization**
  —denote $\mathrm{W} = [w_{ij}]$ of size $d \times \tilde{d}$
- assume $\tilde{d} < d$: ensure **non-trivial** solution

linear autoencoder hypothesis:

$$\mathbf{h}(\mathbf{x}) = \mathrm{W}\mathrm{W}^T\mathbf{x}$$

Linear Autoencoder Error Function

$$E_{\text{in}}(\mathbf{h}) = E_{\text{in}}(\mathrm{W}) = \frac{1}{N}\sum_{n=1}^{N}\left\|\mathbf{x}_n - \mathrm{W}\mathrm{W}^T\mathbf{x}_n\right\|^2 \text{ with } d \times \tilde{d} \text{ matrix } \mathrm{W}$$

—analytic solution to minimize E_{in}? but **4-th order polynomial of w_{ij}**

let's familiarize the problem with linear algebra (**be brave! :-)**)

- eigen-decompose $\mathrm{W}\mathrm{W}^T = \mathrm{V}\Gamma\mathrm{V}^T$
  - $d \times d$ matrix V **orthogonal**: $\mathrm{V}\mathrm{V}^T = \mathrm{V}^T\mathrm{V} = \mathrm{I}_d$
  - $d \times d$ matrix Γ **diagonal** with $\leq \tilde{d}$ non-zero
- $\mathrm{W}\mathrm{W}^T\mathbf{x}_n = \mathrm{V}\Gamma\mathrm{V}^T\mathbf{x}_n$
  - $\mathrm{V}^T(\mathbf{x}_n)$: change of orthonormal basis (**rotate** or reflect)
  - $\Gamma(\cdots)$: set $\geq d - \tilde{d}$ components to 0, and **scale** others
  - $\mathrm{V}(\cdots)$: reconstruct by coefficients and basis (**back-rotate**)
- $\mathbf{x}_n = \mathrm{V}\mathrm{I}\mathrm{V}^T\mathbf{x}_n$: **rotate** and **back-rotate** cancel out

next: minimize E_{in} **by optimizing Γ and V**

The Optimal Γ

$$\min_{\mathrm{V}} \min_{\Gamma} \frac{1}{N} \sum_{n=1}^{N} \left\| \underbrace{\mathrm{V}\mathrm{I}\mathrm{V}^T \mathbf{x}_n}_{\mathbf{x}_n} - \underbrace{\mathrm{V}\Gamma\mathrm{V}^T \mathbf{x}_n}_{\mathrm{W}\mathrm{W}^T \mathbf{x}_n} \right\|^2$$

- **back-rotate** not affecting length: $\cancel{\mathrm{V}}$
- $\min_{\Gamma} \sum \|(\mathrm{I} - \Gamma)(\text{some vector})\|^2$: **want many** 0 within $(\mathrm{I} - \Gamma)$
- optimal diagonal Γ with rank $\leq \tilde{d}$:

$$\left\{ \begin{array}{c} \tilde{d} \textbf{ diagonal components } 1 \\ \text{other components } 0 \end{array} \right\} \Longrightarrow \text{without loss of gen. } \begin{bmatrix} \mathrm{I}_{\tilde{d}} & 0 \\ 0 & 0 \end{bmatrix}$$

$$\text{next: } \min_{\mathrm{V}} \sum_{n=1}^{N} \left\| \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & \mathrm{I}_{d-\tilde{d}} \end{bmatrix}}_{\mathrm{I}-\text{optimal } \Gamma} \mathrm{V}^T \mathbf{x}_n \right\|^2$$

The Optimal V

$$\min_{\mathrm{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{I}_{d-\tilde{d}} \end{bmatrix} \mathrm{V}^T \mathbf{x}_n \right\|^2 \equiv \max_{\mathrm{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} \mathrm{I}_{\tilde{d}} & 0 \\ 0 & 0 \end{bmatrix} \mathrm{V}^T \mathbf{x}_n \right\|^2$$

- $\tilde{d} = 1$: only first row $\mathbf{v}^T$ of V^T matters

  $$\max_{\mathbf{v}} \sum_{n=1}^{N} \mathbf{v}^T \mathbf{x}_n \mathbf{x}_n^T \mathbf{v} \text{ subject to } \mathbf{v}^T \mathbf{v} = 1$$

  - optimal $\mathbf{v}$ satisfies $\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \mathbf{v} = \lambda \mathbf{v}$
    —using Lagrange multiplier λ, remember? :-)
  - optimal $\mathbf{v}$: 'topmost' eigenvector of $\mathrm{X}^T\mathrm{X}$
- general $\tilde{d}$: $\{\mathbf{v}_j\}_{j=1}^{\tilde{d}}$ 'topmost' eigenvector**S** of $\mathrm{X}^T\mathrm{X}$
  —optimal $\{\mathbf{w}_j\} = \{\mathbf{v}_j \text{ with } [\![\gamma_j = 1]\!]\}$ = **top eigenvectors**

linear autoencoder: projecting to orthogonal patterns $\mathbf{w}_j$ **that 'matches' $\{\mathbf{x}_n\}$ most**

Principal Component Analysis

Linear Autoencoder or **PCA**

1. let $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$, and let $\mathbf{x}_n \leftarrow \mathbf{x}_n - \bar{\mathbf{x}}$
2. calculate $\tilde{d}$ top eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{\tilde{d}}$ of $\mathrm{X}^T\mathrm{X}$
3. return feature transform $\mathbf{\Phi}(\mathbf{x}) = \mathrm{W}(\mathbf{x} - \bar{\mathbf{x}})$

- linear autoencoder:
  maximize $\sum$(maginitude after projection)^2
- principal component analysis (PCA) from statistics:
  maximize $\sum$(variance after projection)
- both useful for **linear dimension reduction**
  though PCA more popular

linear dimension reduction:
useful for **data processing**

Fun Time

When solving the optimization problem

$$\max_{\mathbf{v}} \sum_{n=1}^{N} \mathbf{v}^T \mathbf{x}_n \mathbf{x}_n^T \mathbf{v} \text{ subject to } \mathbf{v}^T \mathbf{v} = 1,$$

we know that the optimal $\mathbf{v}$ is the 'topmost' eigenvector that corresponds to the 'topmost' eigenvalue λ of $\mathrm{X}^T\mathrm{X}$. Then, what is the optimal objective value of the optimization problem?

1. λ^1
2. λ^2
3. λ^3
4. λ^4

Fun Time

When solving the optimization problem

$$\max_{\mathbf{v}} \sum_{n=1}^{N} \mathbf{v}^T \mathbf{x}_n \mathbf{x}_n^T \mathbf{v} \text{ subject to } \mathbf{v}^T \mathbf{v} = 1,$$

we know that the optimal $\mathbf{v}$ is the 'topmost' eigenvector that corresponds to the 'topmost' eigenvalue λ of $\mathrm{X}^T\mathrm{X}$. Then, what is the optimal objective value of the optimization problem?

1. λ^1
2. λ^2
3. λ^3
4. λ^4

Reference Answer: 1

The objective value of the optimization problem is simply $\mathbf{v}^T\mathrm{X}^T\mathrm{X}\mathbf{v}$, which is $\lambda\mathbf{v}^T\mathbf{v}$ and you know what $\mathbf{v}^T\mathbf{v}$ must be.

Summary

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 13: Deep Learning

- Deep Neural Network

  difficult hierarchical feature extraction problem
- Autoencoder

  unsupervised NNet learning of representation
- Denoising Autoencoder

  using noise as hints for regularization
- Principal Component Analysis

  linear autoencoder variant for data processing

- **next: extracting 'prototype' instead of pattern**