Grasping Deep Learning from Fundamentals to Applications

June 15, 2023

Lecture 1 – Intro to Deep Learning

Instructors: Yufei Huang, PhD; Arun Das, PhD
What is deep learning?

- A model paradigm
  - Neural networks

- A learning paradigm
  - Supervised
  - Unsupervised
  - Reinforcement
  - Self-supervised
  - Transfer
  - Contrastive
  - …
Deep Learning brings deep revolution for AI

Simplify AI systems
Deep Learning brings deep revolution for AI

Maximize AI performance

Computer vision

ImageNet Challenge

ILSVRC’16 winner: Error rate 2.991%

Super-human precision

* Human-level performance: 5.1%

Speech recognition

According to Microsoft’s speech group:

Using DL

Microsoft achieves ‘human parity’ in speech recognition system
History of DL

1958 Perceptron
1969 Perceptron criticized
1974 Backpropagation
1998 Convolution Neural Networks for Handwritten Recognition
1995 SVM reigns
2006 Restricted Boltzmann Machine
2012 Google Brain Project on 16k Cores
2012 AlexNet wins ImageNet
History of DL – Artificial Neuron, 1943

McCulloch and Pitts
“A Logical Calculus of the Ideas Immanent in Nervous Activity”
History of DL – Perceptron, 1958

1957 Frank Rosenblatt

"[The Perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

THE NEW YORK TIMES
History of DL – Backpropagation, 1974

Measure how small changes in weights affect output
Can apply NN to regression

(1974) 1986
(Werbos) Rumelhart, Hinton, Williams
“Learning representations by back-propagating errors” (Nature)

Multilayer neural networks, etc.
History of DL –
NN winter; late 1990 – early 2000

- Not enough data
- Not enough computing power
- Imperfect activation function
History of DL –
Restricted Boltzmann machine 2005

Geoffrey Hinton
“The Godfather of deep learning”

98.2% on the MNIST test set
History of DL – Convolution 1998, 2010

Yann LeCun, NYU & Meta

99.50% on the MNIST test set
CURRENT BEST: 99.77% by committee of 35 conv. nets
History of DL – ImageNet 2012 -

Image recognition

• **AlexNet (2012)**
  - 7 layers;
  - 1000 labels; >1.2M images
  - 17% vs 25.7%

• **GoogLeNet; VGG (2014)**
  - 19 layers (VGG)
  - ~7%

• **Deep Residual Net (2015)**
  - 152 layers
  - 3.57% > human recognition
History of DL – ImageNet 2012 –

ImageNet Challenge

ILSVRC’16 winner: Error rate 2.991%

Super-human precision

* Human-level performance: 5.1%
DL in speech recognition

According to Microsoft’s speech group:

Using DL

Microsoft achieves 'human parity' in speech recognition system

WER 5.1%
Deep learning applications

- Image recognition
- Object detection
- Video game
- Image generation

AlphaGO

AlphaFold

Medical diagnosis

ChatGPT

- ImageNet Classification top-5 error (%)
- Revolution of Depth
- Groundtruth:
  - person
  - hat with a wide brim
  - hat with a wide brim (2)
  - hat with a wide brim (3)
  - oboe
  - oboe (2)
  - saxophone
  - trombone
  - person (2)
  - person (3)
  - person (4)

University of Pittsburgh
Categories of DL models

- **Supervised**
  - Deep (Dense) Neural Networks (DNN)
  - Convolutional Neural Networks (CNN)
  - Recurrent (RNN); LSTM, transformers

- **Unsupervised**
  - Auto-encoder (AE)
  - Variational AE (VAE)
  - Generative Adversarial Networks (GAN)
  - Diffusion models
Supervised learning (classification)

Goal

Labeled training data (lots of)

Training
Unsupervised DL models (Autoencoder, generative adversarial networks (GAN))

Goal: Learn the codes (latent representation) of data
Visualizing the code
Reconstruct
Generative Adversarial Networks (GAN)

**Goal:** Train a generator (decoder) $G$ that learns from data to generate samples from data distribution.

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} \left[ \log D(x) \right] + \mathbb{E}_{z \sim p_z(z)} \left[ \log (1 - D(G(z))) \right].
\]
GAN Arts

Deepfake

You can now watch the video.

https://d285xazlytdv8t.cloudfront.net/output.mp4

Drug design

Artificial intelligence takes on song-composing duties in Eurovision-inspired contest

BY RODRIGO PÉREZ ORTEGA | APR. 24, 2020
Processes of Building Deep Learning

- **Observation/Data:** $D = \{y, x\}$

- **Feature extraction and selection**

- **Modeling**
  - **Goal:** Model $D$
  - **Task:** Define $f$: $y = F(x; w)$

- **Training**
  - **Goal:** Infer $w$ (including hyper-parameters)
  - **Task 1:** Loss function - $L(D, w)$
  - **Task 2:** optimization
    - $\hat{w} = \arg \min_w L(D, w)$

- **Performance assessment**
  - accuracy, AUC, PR, mAP, …
Classification

image → label

Training data

5

Test data

6 → 5
Curve fitting (regression)
Processes of Building Deep Learning

- **Observation/Data:** $D = \{y, x\}$
- **Feature extraction and selection**
- **Modeling**
  - *Goal:* Model $D$
  - *Task:* Define $f: y = F(x; w)$
- **Training**
  - *Goal:* Infer $w$ (including hyper-parameters)
  - *Task 1:* Loss function - $L(D, w)$
  - *Task 2:* optimization
    - $\hat{w} = \text{argmin}_w L(D, w)$
- **Performance assessment**
  - accuracy, AUC, PR, mAP, …
### Single neuron

#### Pre-activation

- **Input:** 0, 8, 15, 22
- **Output:** 32, 46.4, 59, 71.6

\[ a = x \times 1.8 + 32 \]

#### Activation

Activation function (Sigmoid)

\[ F = g(a) = g(x \times 1.8 + 32) \]

Activation function:

\[ g(a) = \text{sigm}(a) = \frac{1}{1 + \exp(-a)} \]
(a) Step function or threshold function
(b) Sigmoid function \( y = \frac{1}{1+e^{-x}} \); takes a real-valued input and squashes it to range between 0 and 1
(c) ReLU function
(d) Softmax

**Classification Problems**

Softmax converts the input vector to a probabilistic domain. This is very important for us for the final output layer.
Single hidden layer NN or fully connected layers

– Hidden layer pre-activation:

\[ a_i = w_i^{(1)T} x + b_i^{(1)} \]

or

\[ a^{(1)} = W^{(1)} x + b^{(1)} \]

– Hidden layer activation:

\[ h^{(1)} = g(a^{(1)}) \]

– Output layer pre-activation:

\[ a^{(2)} = w^{(2)T} h^{(1)} + b^{(2)} \]

– Output layer activation:

\[ f(x; \theta) = o(a^{(2)}) \]

\[ O(\theta) \]: output activation function
Deep Neural Networks (DNN)

- Could have $L$ hidden layers:
  - layer pre-activation for $k > 0$ $(h^{(0)}(x) = x)$
    \[
    a^{(k)}(x) = b^{(k)} + W^{(k)}h^{(k-1)}(x)
    \]
  - hidden layer activation ($k$ from 1 to $L$):
    \[
    h^{(k)}(x) = g(a^{(k)}(x))
    \]
  - output layer activation ($k = L+1$):
    \[
    h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)
    \]
Processes of Building Deep Learning

- **Observation/Data**: $D = \{y, x\}$
- **Feature extraction and selection**
- **Modeling**
  - **Goal**: Model $D$
  - **Task**: Define $f$: $y = F(x; w)$
- **Training**
  - **Goal**: Estimate $w$ (including hyper-parameters)
  - **Task 1**: Loss function - $L(D, w)$
  - **Task 2**: optimization
    - $\hat{w} = \arg\min_w L(D, w)$
- **Performance assessment**
  - accuracy, AUC, PR, mAP, …

(DL) (hyper) parameters
Cross entropy; MSE
Stochastic gradient descent
Goal of DL training

– Determine model weights \((W, b)\), based on training data

Input: 0, 8, 15, 22, 38

Output: 32, 46.4, 59, 71.6, 100.4
Ingredients of DL Training

- Training data (labeled data)
- Loss function $L(D, w)$
- Optimizers

$$\hat{w} = \arg\min_w L(D, w)$$
Loss functions

- Assesses how good an estimate of \((W, b)\) is
- Popular loss functions

- Cross entropy loss (Classification)
- Mean square error (regression; real-valued output)
Optimization

\[ \hat{w} = \text{argmin}_w L(D, w) \]

- Solution (Optimizer): Stochastic gradient descent algorithm
- Efficient implementation: back-propagation

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbreviation</th>
<th>Mean momentum</th>
<th>Std. momentum</th>
<th>Strengths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stochastic Gradient Descent</td>
<td>SGD</td>
<td>FALSE</td>
<td>FALSE</td>
<td>Easy to understand</td>
</tr>
<tr>
<td>Nesterov Momentum</td>
<td>SGD w/ Nesterov</td>
<td>TRUE</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Root Mean Square Propagation</td>
<td>RMSProp</td>
<td>TRUE</td>
<td>FALSE</td>
<td>Works well w/ text input</td>
</tr>
<tr>
<td>Adaptive Moment Estimation</td>
<td>Adam</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Good default</td>
</tr>
</tbody>
</table>
What does it look like?

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Input Layer (input)</th>
<th>Hidden Layer</th>
<th>Output Layer (prediction)</th>
<th>Actual Output (label)</th>
<th>Compute Error (loss)</th>
<th>Loss function MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$h = f(x) = w_{1}x + b_{1}$</td>
<td>12.1</td>
<td>32</td>
<td>Loss(12, 32) -&gt; very large</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>$h = f(x) = w_{2}x + b_{2}$</td>
<td>19.9</td>
<td>32</td>
<td>Loss(19.9, 32) -&gt; moderately large</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>0</td>
<td>$h = f(x) = w_{n}x + b_{n}$</td>
<td>31.4</td>
<td>32</td>
<td>Loss(31.4, 32) -&gt; very small</td>
<td></td>
</tr>
</tbody>
</table>
Important concepts

- **Terminology**
  - **Batch size**: # of samples fed into an SGD step
  - **Epoch**: # of steps that takes to use all training samples
  - Learning rate
  - Initial value

![Diagram showing the relationship between loss and epoch with different learning rates](image)
Processes of Building Deep Learning

- **Observation/Data:** $D = \{y, x\}$
- **Feature extraction and selection**
- **Modeling**
  - **Goal:** Model $D$
  - **Task:** Define $f: y = F(x; w)$
- **Training**
  - **Goal:** Estimate $w$ (including hyper-parameters)
  - **Task 1:** Loss function - $L(D, w)$
  - **Task 2:** Optimization
    - $\hat{w} = \arg\min_w L(D, w)$
- **Performance assessment**
  - accuracy, AUC, PR, mAP, …
Evaluating classification performance

- **Errors**
  - **False Positive**: Incorrectly labeled as relevant
  - **False Negative**: Incorrectly labeled as not relevant

Prediction: + + - - + + +

Image:
- True Positive
- True Negative
- False Negative
- False Positive
Evaluating classification performance

- Types of detection outcomes

<table>
<thead>
<tr>
<th>Decision</th>
<th>( \mathcal{H}_0 ) (no signal)</th>
<th>( \mathcal{H}_1 ) (having signal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{H}_0 )</td>
<td>True negative</td>
<td>Miss, Type II, or False negative</td>
</tr>
<tr>
<td>( \mathcal{H}_1 )</td>
<td>False alarm, Type I, or False positive</td>
<td>Detection, True positive</td>
</tr>
</tbody>
</table>

- Types of probabilities

<table>
<thead>
<tr>
<th>Decision</th>
<th>( \mathcal{H}_0 ) (no signal)</th>
<th>( \mathcal{H}_1 ) (having signal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{H}_0 )</td>
<td>Specificity</td>
<td>( P_{Miss}, \beta ), False negative rate</td>
</tr>
<tr>
<td>( \mathcal{H}_1 )</td>
<td>( P_{FA}, \alpha ), False positive rate</td>
<td>( P_D, 1 - \beta ), power, sensitivity</td>
</tr>
</tbody>
</table>
Specificity and Sensitivity

- **Specificity**
  - True negative probability;
  - 1 - False positive probability;
  - Percentage of negative examples that are correctly labeled
  - \( \text{Specificity} = \frac{\# \text{ true negatives}}{\# \text{ negatives}} \)

- **Sensitivity (Recall)**
  - True positive probability;
  - Percentage of positive examples that are correctly labeled
  - \( \text{Recall} = \frac{\# \text{ true positives}}{\# \text{ positives}} \)
Accuracy and precision

- **Precision**
  - Percentage of positive labels that are correct
  - Precision = (# true positives) / (# true positives + # false positives)

- **Accuracy**
  - Percentage of correct labels
  - Accuracy = (# true positives + # true negatives) / (# of samples)
  - Accuracy = 1 – P(error)
Example

Prediction:

Image:

- True Positive
- True Negative
- False Negative
- False Positive
- False Positive
- True Positive

Specificity = 1/3
Recall = 2/3

Precision = 2/4
Accuracy = 3/6
ROC curve and Area Under the Curve (AUC)
When to use which measure?

- No preferred labels and proportion of labels is unknown
  - ROC
- No preferred labels and proportion of labels is known
  - Accuracy
- Have a preferred label and proportion of the preferred label is small.
  - Precision vs. recall
Measuring Success/Failure for Classification

- Can we evaluate classification performance using training data?
  - No, because these could be a classifier that can produce 0 error on training data. This is called overfitting.
  - Overfitting
    - Model performs well on training data but poorly on test data.
Use test data to measure success

- **Training Data**
  - data used to learn a model

- **Test Data**
  - data used to assess the accuracy of model

- What to do when you only have training data?
Cross Validation

- To avoid overfitting
  - train on part of available data, and test on rest
    - if dataset large (say, in 1000’s), can simply set aside \( \approx 1000 \) random examples as test
    - otherwise, use 10-fold cross validation
      - break dataset randomly into 10 parts
      - in turn, use each block as a test set, training on other 9 blocks