CS 480/680 Introduction to Machine Learning

Keven Qiu

Contents

1 Introduction **6**

I Classical Machine Learning **7**

2 Perceptron **8**

2.1 Linear Separator 8

2.2 Perceptron Algorithm 10

2.3 Convergence 10

3 Linear Regression **13**

3.1 General Regression 13

3.2 Linear Regression 15

4 Logistic Regression **18**

4.1 Multiclass Classification 20

5 Hard-Margin Support Vector Machines **21**

5.1 Margin 21

5.2 Transform to Standard Form 22

5.3 Comparison to Perceptron 22

5.4 Lagrangian Dual 23

6 Soft-Margin Support Vector Machine **24**

6.1 Hinge Loss 24

6.2 Lagrangian Dual . . . 26
6.3 Complementary Slackness . . . 26
6.4 Recovering Parameters . . . 27
6.5 Training by Gradient Descent . . . 27

7 Reproducing Kernels **28**
7.1 The Kernel Trick . . . 29
7.2 Kernel SVM . . . 30

8 Gradient Descent **32**
8.1 Gradient Descent in ML Models . . . 32
8.2 Convergence Analysis . . . 33
8.2.1 Taylor Series . . . 33
8.2.2 Convex Case . . . 33
8.2.3 Strong Convex Case . . . 35
8.2.4 Nonconvex Case . . . 35
8.3 Stochastic Gradient Descent . . . 36

II Neural Networks 37

9 Multilayer Perceptron/Fully Connected Neural Networks **38**
9.1 Introduction . . . 38
9.2 Multiclass Classification . . . 39
9.3 Activation Functions . . . 39
9.4 Training . . . 40
9.4.1 Computing the Gradient . . . 41
9.5 Universal Representation . . . 43
9.5.1 Dropout . . . 43
9.5.2 Batch Normalization . . . 43

10 Convolutional Neural Networks **45**

10.1 Convolution . . . 46
10.2 Size Calculation . . . 47
10.3 Convolutional = FC Layer with Weight Sharing . . . 47
10.4 Pooling . . . 47
10.5 CNN Architectures . . . 48
10.5.1 Descriptions . . . 49

11 Transformer **52**
11.1 Input/Output Embedding Layer . . . 53
11.2 Positional Encoding . . . 54
11.3 Attention Layer . . . 55
11.3.1 Matrix Form of Attention . . . 56
11.4 Learnable Attention Layer and Multihead Attention . . . 57
11.4.1 Masked Multihead Attention . . . 57
11.5 Feed-Forward Layer . . . 58
11.6 Layer Normalization . . . 58
11.7 Overview . . . 58
11.8 Transformer Loss . . . 58

III Modern Machine Learning Paradigms **59**

12 Large Language Models **60**
12.1 Transformer Review . . . 60
12.2 Labeling Smoothing . . . 60
12.3 BERT vs. GPT . . . 60
12.4 Pretraining–Fine-tuning–Inference Framework . . . 61
12.5 Generative Pretraining Transformer (GPT) . . . 61
12.5.1 Pretraining . . . 62
12.5.2 Fine-tuning . . . 62
12.5.3 Task-Dependent Architecture . . . 62

12.6 Bidirectional Encoder Representations from Transformers (BERT) 63
12.6.1 Pretraining Task A 64
12.6.2 Pretraining Task B 64
12.6.3 RoBERTa 64
12.7 GPT-2 64
12.8 GPT-3 65
12.9 GPT-3.5 65
12.10 Reinforcement Learning from Human Feedback 66
12.10.1 Step 1 66
12.10.2 Step 2 66
12.10.3 Step 3 66
12.11 GPT-4 67
12.11.1 Parameter-Efficient Fine-Tuning (PEFT) 67

13 Generative Adversarial Networks **68**
13.1 Generating Samples 68
13.1.1 Discriminator 69
13.1.2 Generator 70
13.2 Analysis of GAN 70

14 Self-Supervised Learning **73**
14.1 Geometric Transformation Recognition 74
14.1.1 Image Rotation 74
14.2 Patches 74
14.2.1 Relative Patch Position 74
14.2.2 Image Jigsaw Puzzle 74
14.3 Generative Modelling 74
14.3.1 Context Encoders 74
14.3.2 Image Colourization 75
14.3.3 Cross-Channel Prediction 75

14.3.4 Image Super-Resolution 75
14.4 Contrastive Learning 75
14.4.1 SimCLR 75

IV Trustworthy Machine Learning 77

15 Evasion Attacks 78
15.1 Principle of Generating Evasion Attacks 78
15.2 Evasion Attacks 79
15.2.1 Fast Gradient Sign Method (FGSM) Attack 79
15.2.2 Basic Iterative Method (BIM) Attack 80
15.2.3 Projected Gradient Descent (PGD) Attack 80
15.2.4 Multi-Targeted PGD Attack 81

16 Robustness 83
16.1 Adversarial Training 83
16.1.1 FGSM Attack 84
16.1.2 Ensemble Adversarial Training 84
16.1.3 PGD Attack 84
16.1.4 Trade-Off 84
16.1.5 Trade-Off Between Robustness and Accuracy 84
16.1.6 Classification-Calibrated Surrogate Loss 85
16.2 Limitations of Adversarial Training 86

17 Privacy 87

18 Ethics 88

19 Other Threats 89

Chapter 1

Introduction

Definition: Artificial Intelligence

A scientific field concerned with the development of algorithms that allow computers to learn without being explicitly programmed.

Definition: Machine Learning

A branch of Artificial Intelligence, which focuses on methods that learn from data and make predictions on unseen data.

There are three phases to machine learning: 1) training; 2) prediction; 3) evaluation

Definition: Paradigms of ML Algorithms

Supervised: learning with labeled data $(\mathbf{x}, y)$.
Unsupervised: discover patterns in unlabeled data $\mathbf{x}$.
Semi-supervised: using both labeled and unlabeled data.

Part I

Classical Machine Learning

Chapter 2

Perceptron

Dataset: Each column is a data point: n in total; each has d features. y is the label vector.

	Training samples						Test samples	
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\cdots$	$\mathbf{x}_n$	$\mathbf{x}'_1$	$\mathbf{x}'_2$
$\mathbb{R}^d \ni$ Feature	0	1	0	1	$\cdots$	1	1	0.9
	0	0	1	1	$\cdots$	0	1	1.1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	1	0	1	0	$\cdots$	1	1	-0.1
Label y	+	+	−	+	$\cdots$	−	?	?

$\mathbf{x}'_1$ and $\mathbf{x}'_2$ are test samples whose labels need to be predicted.

Definition: Bag of Words Representation of Text

If a word appears, the feature is 1.

Email Spam Detection Example: $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ and $y = [y_1, \ldots, y_n] \in \{\pm 1\}^n$.

Each column of X is an email $\mathbf{x}_i \in \mathbb{R}^d$, each with d binary features. Each entry in y is a label $y_i \in \{\pm 1\}$ indicating spam or not. Given new email $\mathbf{x}'$, predict spam or not.

For an OR dataset, the data is linearly separable meaning you can

2.1 Linear Separator

Definition: Inner Product

$$\langle a, b \rangle := \sum_j a_j b_j$$

where a_j and b_j are the jth elements of vectors $\mathbf{a}$ and $\mathbf{b}$.

Definition: Linear Function

$\forall \alpha, \beta \in \mathbb{R}, \forall \mathbf{x}, \mathbf{z} \in \mathbb{R}^d$,

$$f(\alpha \mathbf{x} + \beta \mathbf{z}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{z})$$

equivalently, $\exists \mathbf{w} \in \mathbb{R}^d$ such that $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle = \sum_j x_j w_j$.

Proof. ($\Longrightarrow$) Let $\mathbf{w} = [f(\mathbf{e}_1), \ldots, f(\mathbf{e}_d)]$, where $\mathbf{e}_i$ is the ith coordinate vector.

$$\begin{aligned} f(\mathbf{x}) &= f(x_1 \mathbf{e}_1 + x_2 \mathbf{e}_2 + \cdots + x_d \mathbf{e}_d) \\ &= x_1 f(\mathbf{e}_1) + \cdots x_d f(\mathbf{e}_d) = \langle \mathbf{x}, \mathbf{w} \rangle \end{aligned}$$

($\Longleftarrow$) We have

$$\begin{aligned} f(\alpha \mathbf{x} + \beta \mathbf{z}) &= \langle \alpha \mathbf{x} + \beta \mathbf{z}, \mathbf{w} \rangle \\ &= \alpha \langle \mathbf{x}, \mathbf{w} \rangle + \beta \langle \mathbf{z}, \mathbf{w} \rangle \\ &= \alpha f(\mathbf{x}) + \beta f(\mathbf{z}) \end{aligned}$$

Definition: Affine Function

$\exists \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$ such that $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$.

Definition: Thresholding

$$\text{sign}(t) = \begin{cases} +1 & t > 0 \\ -1 & t \leq 0 \end{cases}$$

Definition: Linear Separator

$\mathbf{w}$ and b uniquely determine the linear separator.

$$\hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b) = \begin{cases} +1 & \langle \mathbf{x}, \mathbf{w} \rangle + b > 0 \\ -1 & \langle \mathbf{x}, \mathbf{w} \rangle + b \leq 0 \end{cases}$$

$\mathbf{w}$ is orthogonal to the decision boundary $H = \{\mathbf{x} : \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$.

Proof. Any vector with both head and tail in H can be written as $\mathbf{x}\mathbf{x}' = \mathbf{x}' - \mathbf{x}$ for $\mathbf{x}, \mathbf{x}' \in H$.

$$\begin{aligned} \langle \mathbf{w}, \mathbf{x}\mathbf{x}' \rangle &= \langle \mathbf{w}, \mathbf{x}' - \mathbf{x} \rangle \\ &= \langle \mathbf{w}, \mathbf{x}' \rangle - \langle \mathbf{w}, \mathbf{x} \rangle \\ &= -b - (-b) \\ &= 0 \end{aligned}$$

b does not affect the orthogonality. Holds for any H. The length of $\mathbf{w}$ does not matter in determining the decision boundary.

2.2 Perceptron Algorithm

Algorithm 1 Training Perceptron

Input: Dataset $= (\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n$, initialization $\mathbf{w}_0 \in \mathbb{R}^d$ and $b_0 \in \mathbb{R}$
Output: $\mathbf{w}$ and b (a linear classifier $\text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b)$)
for $t = 1, 2, \ldots$ **do**
  receive index $I_t \in \{1, \ldots, n\}$ (can be random)
  if $y_{I_t}(\langle \mathbf{x}_{I_t}, \mathbf{w} \rangle + b) \leq 0$ **then** ▷ Update only when mistake happens
    $\mathbf{w} \leftarrow \mathbf{w} + y_{I_t}\mathbf{x}_{I_t}$
    $b \leftarrow b + y_{I_t}$

Typically set $\mathbf{w}_0 = \mathbf{0}$ and $b_0 = 0$. $\text{score}_{\mathbf{w},b}(\mathbf{x}) = \hat{y} = \langle \mathbf{w}, \mathbf{x} \rangle + b$.

Perceptron as an Optimization Problem

Find $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$ such that $\forall i, y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0$

When a mistake happens on $(\mathbf{x}, y)$ we want to update $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + y\mathbf{x}$ and $b_{k+1} = b_k + y$:

$$y[\langle \mathbf{x}, \mathbf{w}_{k+1} \rangle + b_{k+1}] = y[\langle \mathbf{x}, \mathbf{w}_k + y\mathbf{x} \rangle + b_k + y] = y[\langle \mathbf{x}, \mathbf{w}_k \rangle + b_k] + \underbrace{\|\mathbf{x}\|_2^2 + 1}_{\text{always positive}}$$

A trick to hide the bias term of the affine function is to pad a constant 1 to the end of each $\mathbf{x}$:

$$\langle \mathbf{x}, \mathbf{w} \rangle + b = \left\langle \underbrace{\begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}}_{\mathbf{x}_{pad}}, \underbrace{\begin{pmatrix} \mathbf{w} \\ b \end{pmatrix}}_{\mathbf{w}_{pad}} \right\rangle$$

The update when a mistake happens on $\langle \mathbf{x}, y \rangle$ is

$$\begin{cases} \mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x} \\ b \leftarrow b + y \end{cases} \iff \mathbf{w}_{pad} \leftarrow \mathbf{w}_{pad} + y\mathbf{x}_{pad}$$

2.3 Convergence

Theorem Convergence (Linearly Separable Case); Block (1962); Novikoff (1962)

Suppose $\exists \mathbf{w}^*$ such that $y_i \langle \mathbf{x}_i, \mathbf{w}^* \rangle > 0$ for all i. Assume that $\|\mathbf{x}_i\|_2 \leq C$ for all i and we normalize the $\mathbf{w}^*$ such that $\|\mathbf{w}^*\|_2 = 1$. Let us define the margin $\gamma := \min_i |\langle \mathbf{x}_i, \mathbf{w}^* \rangle|$. Then the Perceptron algorithm converges after C^2/γ^2 mistakes.
Remember normalizing $\mathbf{w}^*$ does not change the decision boundary of the linear classifier.

Proof. Recall that the update is $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$ when a mistake happens on $(\mathbf{x}, y)$. Consider the effect of an update on $\langle \mathbf{w}, \mathbf{w}^* \rangle$:

$$\langle \mathbf{w} + y\mathbf{x}, \mathbf{w}^* \rangle = \langle \mathbf{w}, \mathbf{w}^* \rangle + y \langle \mathbf{x}, \mathbf{w}^* \rangle = \langle \mathbf{w}, \mathbf{w}^* \rangle + |\langle \mathbf{x}, \mathbf{w}^* \rangle| \geq \langle \mathbf{w}, \mathbf{w}^* \rangle + \gamma$$

This means that for each update, $\langle \mathbf{w}, \mathbf{w}^* \rangle$ grows by at least $\gamma > 0$. Consider the effect of an update on $\langle \mathbf{w}, \mathbf{w} \rangle$:

$$\langle \mathbf{w} + y\mathbf{x}, \mathbf{w} + y\mathbf{x} \rangle = \langle \mathbf{w}, \mathbf{w} \rangle + \underbrace{2y \langle \mathbf{w}, \mathbf{x} \rangle}_{<0} + \underbrace{y^2 \langle \mathbf{x}, \mathbf{x} \rangle}_{\in[0, C^2]} \leq \langle \mathbf{w}, \mathbf{w} \rangle + C^2$$

This means for each update, $\langle \mathbf{w}, \mathbf{w} \rangle$ grows by at most C^2.

Let $\mathbf{w}_0 = \mathbf{0}$. Now we know that after M updates that $\langle \mathbf{w}, \mathbf{w}^* \rangle \geq M\gamma$ and $\langle \mathbf{w}, \mathbf{w} \rangle \leq MC^2$. Thus,

$$\begin{aligned} 1 \geq \cos(\mathbf{w}, \mathbf{w}^*) &= \frac{\langle \mathbf{w}, \mathbf{w}^* \rangle}{\|\mathbf{w}\| \, \|\mathbf{w}^*\|} \\ &\geq \frac{M\gamma}{\sqrt{MC^2} \times 1} \\ &= \sqrt{M} \frac{\gamma}{C} \end{aligned}$$

This implies $M \leq C^2/\gamma^2$. The larger the margin γ is, the more linearly separable the data is, and hence the fast the Perceptron algorithm will converge.

We want to minimize the Perceptron loss. Define

$$l(\mathbf{w}, \mathbf{x}_t, y) = -y_t \langle \mathbf{w}, \mathbf{x}_t \rangle \mathbb{I}[\text{mistake on } \mathbf{x}_t] = -\min\{y_t \langle \mathbf{w}, \mathbf{x}_t \rangle, 0\}$$

$$L(\mathbf{w}) = -\frac{1}{n} \sum_{t=1}^{n} y_t \langle \mathbf{w}, \mathbf{x}_t \rangle \mathbb{I}[\text{mistake on } \mathbf{x}_t]$$

where $\mathbb{I}$ is the indicator function.

Definition: (Stochastic) Gradient Descent Update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} l(\mathbf{w}_t, \mathbf{x}_t, y_t) = \mathbf{w}_t + \eta_t y_t \mathbf{x}_t \mathbb{I}[\text{mistake on } \mathbf{x}_t]$$

Set step size $\eta_t = 1$. If a mistake on $(\mathbf{x}_t, y_t)$, then

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t$$

Proposition

Perceptron is not unique since the algorithm stops as long as there is no mistake.

We want to maximize the smallest margin: support vector machines.

There is no separating hyperplane for the XOR dataset.

Proof. Suppose $\exists \mathbf{w}, b$ such that $y(\langle \mathbf{w}, \mathbf{x} \rangle + b) > 0$.

- $\mathbf{x}_1 = (0,0), y_1 = - \implies b < 0$
- $\mathbf{x}_2 = (1,0), y_2 = + \implies w_1 + b > 0$
- $\mathbf{x}_3 = (0,1), y_3 = + \implies w_2 + b > 0 \implies w_1 + w_2 + 2b > 0$
- $\mathbf{x}_4 = (1,1), y_4 = - \implies w_1 + w_2 + b < 0 \implies b > 0$

This is a contradiction.

Theorem Minsky and Papert (1969); Block and Levin (1970)

If there is no perfect separating hyperplane, then the Perceptron algorithm cycles.

Definition: Soft-Margin

Induced by a reasonable loss l and regularizer reg

$$\min_{\mathbf{w}} \hat{\mathbb{E}} l(y\hat{y}) + reg(\mathbf{w}), \text{ s.t. } \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$

Penalizing a mistake by the loss l, but not infinitely large.

Stopping Perceptron:

- Maximum number of iterations is reached
- Maximum allowed runtime is reached
- Training error stops changing
- Validation error stops decreasing

Multiclass Perceptron: Let c be the total number of classes. There are two methods:

1. One vs. All

   Let class k be positive and all other classes negative. Train Perceptron $\mathbf{w}_k$. There are a total of c imbalanced Perceptrons. Predict according to the highest score:

   $$\hat{y} := \arg\max_k \langle \mathbf{x}, \mathbf{w}_k \rangle$$

2. One vs. One Let class k be positive and class l be negative, discard other classes. Train Perceptron $\mathbf{w}_{k,l}$; $\binom{c}{2}$ balanced Perceptrons. Predict by majority vote:

   $$\hat{y} := \arg\max_k \sum_{i \neq k} \langle \mathbf{x}, \mathbf{w}_{k,l} \rangle > 0$$

Chapter 3

Linear Regression

3.1 General Regression

Definition: Regression

Given training data $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ is the feature vector for the i-th training sample and $y_i \in \mathcal{Y} \subseteq \mathbb{R}^t$ are the t responses where $t = 1$ or $t = \infty$, find $f : \mathcal{X} \to \mathcal{Y}$ such that

$$f(\mathbf{x}_i) \approx y_i$$

Theorem (Exact Interpolation Always Possible)

For any finite training data $(\mathbf{x}_i, y_i) : i = 1, \ldots, n$, there exist infinitely many functions f such that for all i,

$$f(\mathbf{x}_i) = y_i$$

- Cannot decide on a unique f.
- On new data $\mathbf{x}$, our prediction $\hat{y} = f(\mathbf{x})$ can vary significantly.
- Leveraging prior knowledge of f is important.
- Simplest explanation is usually correct.

Prior knowledge on the functional form of f such as linear or nonlinear.

There are three results of regression: underfitting, good fit/robust, overfitting.

Training and test data are both i.i.d. (independent and identically distributed) samples from the same unknown distribution $\mathcal{P}$. $((\mathbf{X}_i, Y_i) \sim \mathcal{P}$ and $(\mathbf{X}, Y) \sim \mathcal{P})$

Definition: Least Squares Regression

$$\min_{f:\mathcal{X}\to\mathcal{Y}} \mathbb{E}\,\|f(\mathbf{X}) - Y\|_2^2$$

where we use l_2 loss to measure error.
The square is to make the calculation of the gradient easier.

Definition: Regression Function

Optimal regression function to the least squares regression:

$$f^*(\mathbf{x}) = m(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$$

Bias-Variance Decomposition: To show the regression function is optimal. The expectation is with respect to both $\mathbf{X}$ and Y.

$$\begin{aligned}
\mathbb{E}\,\|f(\mathbf{X}) - Y\|_2^2 &= \mathbb{E}\,\|f(\mathbf{X}) - m(\mathbf{X}) + m(\mathbf{X}) - Y\|_2^2 \\
&= \mathbb{E}\,\|f(\mathbf{X}) - m(\mathbf{X})\|_2^2 + \mathbb{E}\,\|m(\mathbf{X}) - Y\|_2^2 \\
&+ \underbrace{2\mathbb{E}\,\langle f(\mathbf{X}) - m(\mathbf{X}), m(\mathbf{X}) - Y\rangle}_{=0} \\
&= \mathbb{E}\,\|f(\mathbf{X}) - m(\mathbf{X})\|_2^2 + \underbrace{\mathbb{E}\,\|m(\mathbf{X}) - Y\|_2^2}_{\text{noise (variance)}}
\end{aligned}$$

Showing the 0 term:

$$\begin{aligned}
\mathbb{E}\,\langle f(\mathbf{X}) - m(\mathbf{X}), m(\mathbf{X}) - Y\rangle &= \mathbb{E}_{\mathbf{X}}[\mathbb{E}_{Y|\mathbf{X}}\,\langle f(\mathbf{X}) - m(\mathbf{X}), m(\mathbf{X}) - Y\rangle] \\
&= \mathbb{E}_{\mathbf{X}}\,\langle f(\mathbf{X}) - m(\mathbf{X}), m(\mathbf{X}) - m(\mathbf{X})\rangle = 0
\end{aligned}$$

So the following holds true for any f. The noise variance is a constant term w.r.t. to f. We aim to choose $f \approx m$ to minimize the squared error.

$$\mathbb{E}\,\|f(\mathbf{X}) - Y\|_2^2 = \mathbb{E}\,\|f(\mathbf{X}) - m(\mathbf{X})\|_2^2 + \underbrace{\mathbb{E}\,\|m(\mathbf{X}) - Y\|_2^2}_{\text{noise (variance)}}$$

Remember $m(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ is our gold rule, however it is inaccessible since we do not know the conditional distribution, we are learning f from the training data D.

Let f_D be the regressor learned on D. We have proved that

$$\mathbb{E}_{\mathbf{X},Y}\,\|f_D(\mathbf{X}) - Y\|_2^2 = \mathbb{E}_{\mathbf{X}}\,\|f_D(\mathbf{X}) - m(\mathbf{X})\|_2^2 + \underbrace{\mathbb{E}_{\mathbf{X},Y}\,\|m(\mathbf{X}) - Y\|_2^2}_{\text{noise (variance)}}$$

Define $\overline{f}(\mathbf{X}) = \mathbb{E}_D[f_D(\mathbf{X})]$. We can break down the first term

$$\begin{aligned}
\mathbb{E}_D\mathbb{E}_{\mathbf{X}}\,\|f_D(\mathbf{X}) - m(\mathbf{X})\|_2^2 &= \mathbb{E}_{D,\mathbf{X}}\left\|f_D(\mathbf{X}) - \overline{f}(\mathbf{X}) + \overline{f}(\mathbf{X}) - m(\mathbf{X})\right\|_2^2 \\
&= \mathbb{E}_{\mathbf{X}}\left\|\overline{f}(\mathbf{X}) - m(\mathbf{X})\right\|_2^2 + \mathbb{E}_{D,\mathbf{X}}\left\|f_D(\mathbf{X}) - \overline{f}(\mathbf{X})\right\|_2^2 \\
&\underbrace{2\mathbb{E}_{D,\mathbf{X}}\left\langle \overline{f}(\mathbf{X}) - m(\mathbf{X}), f_D(\mathbf{X}) - \overline{f}(\mathbf{X})\right\rangle}_{=0}
\end{aligned}$$

Note that

$$\begin{aligned}\mathbb{E}_{D,\mathbf{X}}\left\langle f(\mathbf{X})-m(\mathbf{X}), f_D(\mathbf{X})-\overline{f}(\mathbf{X})\right\rangle &= \mathbb{E}_{\mathbf{X}}\mathbb{E}_D\left\langle \overline{f}(\mathbf{X})-m(\mathbf{X}), f_D(\mathbf{X})-\overline{f}(\mathbf{X})\right\rangle \\ &= \mathbb{E}_{\mathbf{X}}\left\langle \overline{f}(\mathbf{X})-m(\mathbf{X}), \mathbb{E}_D[f_D(\mathbf{X})]-\overline{f}(\mathbf{X})\right\rangle \\ &= \mathbb{E}_{\mathbf{X}}\left\langle \overline{f}(\mathbf{X})-m(\mathbf{X}), \overline{f}(\mathbf{X})-\overline{f}(\mathbf{X})\right\rangle = 0\end{aligned}$$

Definition: Bias-Variance Trade-Off

$$\underbrace{\mathbb{E}_{D,\mathbf{X},Y}\left\|f_D(\mathbf{X}-Y)\right\|_2^2}_{\text{test error}}$$
$$= \underbrace{\mathbb{E}_{\mathbf{X}}\left\|\mathbb{E}_D[f_D(\mathbf{X})]-m(\mathbf{X})\right\|_2^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{D,\mathbf{X}}\left\|f_D(\mathbf{X})-\mathbb{E}_D[f_D(\mathbf{X})]\right\|_2^2}_{\text{variance}} + \underbrace{\mathbb{E}_{\mathbf{X},Y}\left\|m(\mathbf{X})-Y\right\|_2^2}_{\text{noise (variance)}}$$

Bias^2 $\downarrow$, as the model capacity $\uparrow$ (model is more expressively powerful).
Variance $\uparrow$, as the model capacity $\uparrow$ (prediction of model is less stable).

Replace expectation with sample average $(\mathbf{X}_i, Y_i) \sim P$:

$$\min_{f:\mathcal{X}\to\mathcal{Y}} \hat{\mathbb{E}}\left\|f(\mathbf{X})-Y\right\|_2^2 := \frac{1}{n}\sum_{i=1}^{n}\left\|f(\mathbf{X}_i)-Y_i\right\|_2^2$$

Uniform law of large numbers: as training data size $n \to \infty$,

$$\hat{\mathbb{E}} \to \mathbb{E} \text{ and (hopefully) } \arg\min \hat{\mathbb{E}} \to \arg\min \mathbb{E}$$

3.2 Linear Regression

Recall the affine function $f(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$ with $W \in \mathbb{R}^{t\times d}$ and $\mathbf{b} \in \mathbb{R}^t$ and padding $\mathbf{x} \leftarrow \begin{pmatrix}\mathbf{x}\\1\end{pmatrix}, W \leftarrow [W, \mathbf{b}]$, hence $f(\mathbf{x}) = W\mathbf{x}$.

In matrix form, $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{(d+1)\times n}, Y = [y_1, \ldots, y_n] \in \mathbb{R}^{t\times n}$ and define $\|A\|_F = \sqrt{\sum_{ij} a_{ij}^2}$ where a_{ij} is the the ijth element of A.

$$\frac{1}{n}\sum_{i=1}^{n}\left\|f(\mathbf{x}_i)-y_i\right\|_2^2 = \frac{1}{n}\left\|WX-Y\right\|_F^2$$

Theorem (Fermat's Necessary Condition for Extremity)

If $\mathbf{w}$ is a minimizer (or maximizer) of a differentiable function f over an open set, then $f'(\mathbf{w}) = \mathbf{0}$.

Training: The loss function is defined as

$$\text{Loss}(W) = \frac{1}{n} \|WX - Y\|_F^2$$

Taking the derivative $\nabla_W \text{Loss}(W) = \frac{2}{n}(WX - Y)X^T$ and setting to zero results in the **normal equation**:

$$WXX^T = YX^T \implies W = YX^T(XX^T)^{-1}$$

Once you solve for W on the training set (X, Y), we can predict on unseen data X_{test}:

$$\hat{Y}_{test} = WX_{test}$$

Evaluating test error if labels were available:

$$\frac{1}{n_{test}} \left\| Y_{test} - \hat{Y}_{test} \right\|_F^2$$

We can compare the predicted value and errors to reduce training error as a means to reduce test error.

Ill-Conditioning: Let $X = \begin{pmatrix} 0 & \epsilon \\ 1 & 1 \end{pmatrix}$ and $y = \begin{pmatrix} 1 & -1 \end{pmatrix}$. If we solve the linear least squares regression:

$$\mathbf{w} = yX^T(XX^T)^{-1} = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} -1/\epsilon & 1 \\ 1/\epsilon & 0 \end{pmatrix} = \begin{pmatrix} -2/\epsilon & 1 \end{pmatrix}$$

Slight perturbation leads to chaotic behaviour. This happens whenever X is ill-conditioned, i.e. close to rank deficient.

Rank deficient $X \implies$ two columns in X are linearly dependent, but the corresponding y's might be different. This is a contradiction and lead to an unstable $\mathbf{w}$.

Definition: Ridge Regression

$$\min_W \frac{1}{n} \|WX - Y\|_F^2 + \lambda \|W\|_F^2$$

The normal equation is $W(XX^T + n\lambda I) = YX^T$. The regularization constant λ controls the trade-off. $\lambda = 0$ reduces to ordinary linear regression while $\lambda = \infty$ reduces to $W = \mathbf{0}$.

$$\begin{aligned} Loss(W) &= \frac{1}{n} \|WX - Y\|_F^2 + \lambda \|W\|_F^2 \\ \nabla_W Loss(W) = \frac{2}{n}(WX - Y)X^T + 2\lambda W &= 0 \\ WXX^T - YX^T + W(\lambda n I) &= 0 \\ W(XX^T + \lambda n I) &= YX^T \\ W &= YX^T(XX^T + \lambda n I)^{-1} \end{aligned}$$

Data Augmentation:

$$\frac{1}{n}\|WX - Y\|_F^2 + \lambda\|W\|_F^2 = \frac{1}{n}\left\|W\underbrace{\begin{pmatrix} X & \sqrt{n\lambda}I \end{pmatrix}}_{\tilde{X}} - \underbrace{\begin{pmatrix} Y & \mathbf{0} \end{pmatrix}}_{\tilde{Y}}\right\|_F^2$$

Augment X with $\sqrt{n\lambda}I$, i.e. p data points $\mathbf{x}_j = \sqrt{n\lambda}\mathbf{e}_j$ for $j = 1, \ldots, p$, augment Y with zero, which shrinks W towards the origin. Therefore, regularization = data augmentation.

Theorem (Lagrangian)

A hard constraint problem

$$\min_\theta L(\theta) = \frac{1}{n}\sum_{i=1}^n l(\theta; \mathbf{x}_i, y_i), \text{ s.t.} R(\theta) \le r$$

is equivalent to a soft constraint problem

$$\min_\theta L_R(\theta) = \frac{1}{n}\sum_{i=1}^n l(\theta; \mathbf{x}_i, y_i) + \lambda^* R(\theta)$$

for some regularization parameter $\lambda^* > 0$, in the sense that they have the same solutions.

Sparsity: Regularization $\iff$ constraint:

$$\min_{\|W\|_F \le \gamma} \frac{1}{n}\|WX - Y\|_F^2$$

Ridge regression $\to$ dense W which is more computation and harder to interpret.

Lasso (Tibshirani 1996): $\min_{\|W\|_1 \le \gamma} \frac{1}{n}\|WX - Y\|_F^2$.

$$\min_W \frac{1}{n}\|WX - Y\|_F^2 + \lambda\|W\|_1$$

Chapter 4

Logistic Regression

Though called logistic regression, it is for linear classification.

Definition: Logit

$$\langle \mathbf{x}, \mathbf{w} \rangle$$

Let $\mathcal{Y} = \{0, 1\}$ and directly learn confidence $p(\mathbf{x}; \mathbf{w}) := P(Y = 1 | X = \mathbf{x})$.

Given $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, n$, assume independence:

$$\begin{aligned} P(Y_1 = y_1, \ldots, Y_n = y_n | X_1 = \mathbf{x}_1, \ldots, X_n = \mathbf{x}_n) &= \prod_{i=1}^{n} P(Y_i = y_i | X_i = \mathbf{x}_i) \\ &= \prod_{i=1}^{n} [p(\mathbf{x}_i; \mathbf{w})]^{y_i} [1 - p(\mathbf{x}_i; \mathbf{w})]^{1 - y_i} \end{aligned}$$

Definition: Maximum Likelihood Estimation (MLE)

$$\max_{\mathbf{w}} \prod_{i=1}^{n} [p(\mathbf{x}_i; \mathbf{w})]^{y_i} [1 - p(\mathbf{x}_i; \mathbf{w})]^{1 - y_i}$$

or we can maximize the log-likelihood or equivalently, minimizing the negative log-likelihood

$$\min_{\mathbf{w}} \sum_{i=1}^{n} [-y_i \log p(\mathbf{x}_i; \mathbf{w}) - (1 - y_i) \log(1 - p(\mathbf{x}_i; \mathbf{w}))]$$

Definition: Odds Ratio

Defined as

$$\frac{\text{probability of event}}{\text{probability of non-event}}$$

Definition: Logit Transform

The log odds ratio. Assume it is a linear function:

$$\langle \mathbf{x}, \mathbf{w} \rangle = \log \frac{p(\mathbf{x}; \mathbf{w})}{1 - p(\mathbf{x}; \mathbf{w})}$$

where $\mathbf{x}$ is padded with 1 and $\mathbf{w}$ is padded with b.

Definition: Sigmoid Transform

$$p(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\langle \mathbf{x}, \mathbf{w} \rangle}}$$

The logit and sigmoid transformation are equivalent.

If we plug in the sigmoid parameterization $p(\mathbf{x}; \mathbf{w}) = \frac{1}{1+e^{-\langle \mathbf{x}, \mathbf{w} \rangle}}$:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \log[1 + e^{-\langle \mathbf{x}_i, \mathbf{w} \rangle}] + (1 - y_i) \langle \mathbf{x}_i, \mathbf{w} \rangle$$

Definition: Logistic Loss

If $y_i \in \{\pm 1\}$ in the above, then

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \log[1 + e^{-y_i \langle \mathbf{x}_i, \mathbf{w} \rangle}]$$

Solving Logistic Regression:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \log[1 + e^{-y_i \langle \mathbf{x}_i, \mathbf{w} \rangle}]$$

We can use gradient descent algorithm: $\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla_{\mathbf{w}} Loss(\mathbf{w})$.

Prediction:

$$p(\mathbf{x}; \mathbf{w}) = \text{sigmoid}(\langle \mathbf{x}, \mathbf{w} \rangle) = \frac{1}{1 + e^{-\langle \mathbf{x}, \mathbf{w} \rangle}}$$

$\hat{y} = 1$ if and only if $p(\mathbf{x}; \mathbf{w}) = P(Y = 1 | X = \mathbf{x}) > \frac{1}{2}$ if and only if $\langle \mathbf{x}; \mathbf{w} \rangle > 0$.

Therefore, $\hat{y} = \arg\max_k P(Y = k | X = \mathbf{x})$.

The decision boundary remains to be $H := \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle = 0\}$. This allows us to now predict $\hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle)$ as before, but with confidence $p(\mathbf{x}; \mathbf{w})$.

Logistic regression estimates the posterior probability $\eta(\mathbf{x}) := P(Y = 1 | X = \mathbf{x})$ under linear log-odds ratio assumption.
This confidence is meaningless if assumption does not hold true.

Classification requires comparing $\eta(\mathbf{x})$ with $\frac{1}{2}$.

4.1 Multiclass Classification

We have class $y \in \{1, \ldots, c\}$ and we want to learn $\{\mathbf{w}_1, \ldots, \mathbf{w}_c\}$ for each class.

Definition: Softmax Function

$$P(Y = k | X = \mathbf{x}; \mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_c]) = \frac{e^{\langle \mathbf{x}, \mathbf{w}_k \rangle}}{\sum_{l=1}^{c} e^{\langle \mathbf{x}, \mathbf{w}_l \rangle}}$$

Training: Use maximum likelihood estimation

$$\min_{\mathbf{w}} \hat{\mathbb{E}} \left[-\log \frac{e^{\langle X, \mathbf{w}_Y \rangle}}{\sum_{l=1}^{c} e^{\langle X, \mathbf{w}_l \rangle}} \right]$$

Prediction: $\hat{y} = \arg\max_k P(Y = k | X = \mathbf{x}; \mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_c])$

Chapter 5

Hard-Margin Support Vector Machines

5.1 Margin

Let $H := \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle + b = 0\}$. What is the distance from a point $\mathbf{x}_i$ to H?

$\mathbf{w}$ is orthogonal to H. Let $\mathbf{x}$ be any vector in H. The distance = length of the projection of $\mathbf{x}_i - \mathbf{x}$ onto $\mathbf{w}$.

$$dist(\mathbf{x}_i, H) = \frac{|\langle \mathbf{x}_i - \mathbf{x}, \mathbf{w} \rangle|}{\|\mathbf{w}\|_2} = \frac{|\langle \mathbf{x}_i, \mathbf{w} \rangle - \langle \mathbf{x}, \mathbf{w} \rangle|}{\|\mathbf{w}\|_2} = \frac{|\langle \mathbf{x}_i, \mathbf{w} \rangle + b|}{\|\mathbf{w}\|_2} = \frac{y_i \hat{y}_i}{\|\mathbf{w}\|_2}$$

where the second last equality is from $\mathbf{x} \in H$ and the last equality is from $y_i \hat{y}_i > 0$.

Definition: Margin

The smallest distance to a separating hyperplane H among all separable training data.

$$\min_i \frac{y_i \hat{y}_i}{\|\mathbf{w}\|_2} = \min_i \frac{|\langle \mathbf{x}_i, \mathbf{w} \rangle + b|}{\|\mathbf{w}\|_2}$$

The margin with respect to a separating hyperplane is the minimum distance to every point.

Our goal is to maximize the margin among all hyperplanes.

$$\max_{\mathbf{w}, b} \min_i \frac{y_i \hat{y}_i}{\|\mathbf{w}\|_2}, \text{ s.t.} y_i \hat{y}_i > 0, \forall i$$

5.2 Transform to Standard Form

Our current optimization problem is

$$\max_{\mathbf{w},b:\forall i, y_i\hat{y}_i>0} \min_i \frac{y_i\hat{y}_i}{\|\mathbf{w}\|_2}$$

Both the numerator and denominator are homogeneous in $(\mathbf{w}, b)$ meaning that $(\mathbf{w}, b)$ and $(c\mathbf{w}, cb)$ will have the same loss for $c > 0$. Varying c will not break the condition $y_i\hat{y}_i > 0$ and can change the numerator arbitrarily.

Let us fix the numerator arbitrarily to 1:

$$\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|_2} \text{ s.t. } \min_i y_i\hat{y}_i = 1$$

Change the max to min and squaring:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{s.t.} \quad & y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1, \forall i \end{aligned}$$

5.3 Comparison to Perceptron

Hard-Margin SVM: quadratic programming, unique solution, margin maximizing.

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{s.t.} \quad & y_i\hat{y}_i \geq 1, \forall i \end{aligned}$$

Perceptron: linear programming, infinitely many solutions, convergence rate depends on maximum margin.

$$\begin{aligned} \min_{\mathbf{w},b} \quad & 0 \\ \text{s.t.} \quad & y_i\hat{y}_i \geq 1, \forall i \end{aligned}$$

There are 3 parallel hyperplanes:

$$\begin{aligned} H &:= \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w}\rangle + b = 0\} \\ H_+ &:= \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w}\rangle + b = 1\} \\ H_- &:= \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w}\rangle + b = -1\} \end{aligned}$$

Support Vectors: Points lie on the supporting hyperplanes H_+ and H_-. If we delete data points not on the H_+ or H_-, H is still uniquely defined.

5.4 Lagrangian Dual

Introducing the Lagrangian multipliers (dual variables) $\alpha \in \mathbb{R}^n$:

$$\begin{aligned}
&\min_{\mathbf{w},b} \max_{\alpha \geq \mathbf{0}} \frac{1}{2} \|\mathbf{w}\|_2^2 - \sum_i \alpha_i [y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1] \\
&= \min_{\mathbf{w},b} \begin{cases} +\infty & \text{if } \exists i, y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) < 1 \text{ (set } \alpha_i \text{ as } +\infty) \\ \frac{1}{2} \|\mathbf{w}\|_2^2 & \text{if } \forall i, y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 \text{ (set } \alpha_i \text{ as 0 for all } i) \end{cases} \\
&= \min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|_2^2, \text{ s.t. } y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1, \forall i
\end{aligned}$$

Transfer a constrained optimization problem to an unconstrained one on $\mathbf{w}, b$.

In the above, we proved the hard-margin SVM is equivalent to

$$\min_{\mathbf{w},b} \max_{\alpha \geq \mathbf{0}} \frac{1}{2} \|\mathbf{w}\|_2^2 - \sum_i \alpha_i [y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1]$$

Swap min with max (when primal is convex (strong duality)):

$$\max_{\alpha \geq \mathbf{0}} \min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|_2^2 - \sum_i \alpha_i [y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1]$$

Solve the inner unconstrained problem by setting the derivative to 0:

$$\frac{\partial}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0, \ \frac{\partial}{\partial b} = -\sum_i \alpha_i y_i = 0$$

Plug $\mathbf{w}$ in the loss to eliminate inner problem:

$$\begin{aligned}
Loss(\alpha) &= \min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|_2^2 - \sum_i \alpha_i [y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1] \\
&= \frac{1}{2} \left\| \sum_i \alpha_i y_i \mathbf{x}_i \right\|_2^2 - \left\langle \sum_i \alpha_i y_i \mathbf{x}_i, \sum_i \alpha_i y_i \mathbf{x}_i \right\rangle - b \sum_i \alpha_i y_i + \sum_i \alpha_i
\end{aligned}$$

That is, $\max_{\alpha \geq \mathbf{0}} \sum_i \alpha_i - \frac{1}{2} \left\| \sum_i \alpha_i y_i \mathbf{x}_i \right\|_2^2$ such that $\sum_i \alpha_i y_i = 0$. Changing it to a minimization problem and expanding the norm:

$$\min_{\alpha \geq \mathbf{0}} -\sum_i \alpha_i + \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle, \text{ s.t. } \sum_i \alpha_i y_i = 0$$

Comparing the Primal to Dual: Primal variables: $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$, n primal inequalities, and 0 primal equalities.

Dual variables: $\alpha \in \mathbb{R}^n$ where each α_i corresponds to a data sample, n dual inequalities, and 1 dual equality.

Chapter 6

Soft-Margin Support Vector Machine

Suppose the data is not linearly separable, we want to penalize $y(\langle \mathbf{x}, \mathbf{w} \rangle + b) < 1$, i.e. small or negative $y\hat{y}$.

Recall that maximizing the margin is given by:

$$\max_{\mathbf{w}} \frac{1}{\|w\|_2}$$

or we can equivalently minimize

$$\min_{\mathbf{w}} \frac{1}{2} \|w\|_2^2$$

where the $\frac{1}{2}$ is for ease of gradient calculation.

6.1 Hinge Loss

Definition: Hinge Loss

Let $y \in \{\pm 1\}$ be the true label and $\hat{y} \in \mathbb{R}$ be the predicted confidence, then

$$l_{hinge}(y\hat{y}) = (1 - y\hat{y})^+ = \begin{cases} 1 - y\hat{y} & \text{if } y\hat{y} < 1 \\ 0 & \text{otherwise} \end{cases}$$

We want to penalize two types of points: one that is misclassified and one that is correctly classified, but too close to the decision boundary (within margin). Balance between margin maximization and the hinge loss.

Definition: Soft Margin SVM

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^{n} (1 - y_i \hat{y}_i)^+ \\ \text{s.t.} \quad & \hat{y}_i = \langle \mathbf{x}_i, \mathbf{w} \rangle + b, \forall i \end{aligned}$$

$\frac{1}{2}\|w\|_2^2$ is the margin maximization, $(1 - y_i\hat{y}_i)^+$ is the error penalty, and C is a hyper-parameter to control tradeoff.

Hard-margin SVM has hard constraints which we must respect while the soft-margin SVM has a soft penalty: the more you deviate, the heavier the penalty.

Our goal: minimize over $\mathbf{w}, b$. Recall for any event A, then $P[A] = \mathbb{E}\mathbb{I}[A]$ where $\mathbb{I}[A] = 1$ if A happens, and 0 otherwise.

$$P(Y \neq \text{sign}(\hat{Y})) = P(Y\hat{Y} \leq 0) = \mathbb{E}[\mathbb{I}[Y\hat{Y} \leq 0]] := \mathbb{E}[l_{0-1}(Y\hat{Y})]$$

So, our goal is to minimize $\mathbb{E}[l_{0-1}(Y\hat{Y})]$. Even with linear predictors, minimizing the above 0-1 error is NP-hard.

- The loss is not continuous at 0.
- The gradient of the loss is 0 almost surely.

Therefore, we need to consider a surrogate loss, e.g., the hinge loss.

Definition: Bayes' Rule

Given an instance $\mathbf{x}$, Bayes rule is given by

$$\eta(\mathbf{x}) := \arg\min_{\hat{y} \in \mathbb{R}} \mathbb{E}[l_{0-1}(Y\hat{y}) | X = \mathbf{x}]$$

$$\begin{aligned} \eta(\mathbf{x}) &= \arg\min_{\hat{y} \in \mathbb{R}} \mathbb{E}[l_{0-1}(Y\hat{y}) | X = \mathbf{x}] \\ &= \arg\min_{\hat{y} \in \mathbb{R}} P[Y \neq \text{sign}(\hat{y}) | X = \mathbf{x}] \end{aligned}$$

Definition: Classification Calibrated

We say a loss $l(y\hat{y})$ is classification calibrated if and only if for all $\mathbf{x}$,

$$\hat{y}(\mathbf{x}) := \arg\min_{\hat{y} \in \mathbb{R}} \mathbb{E}[l(Y\hat{y}) | X = \mathbf{x}]$$

has the same sign as the Bayes rule $\eta(\mathbf{x}) = \arg\min_{\hat{y} \in \mathbb{R}} \mathbb{E}[l_{0-1}(Y\hat{y}) | X = \mathbf{x}]$.

Theorem (Characterization Under Convexity)

Any convex loss l is classification calibrated if and only if l is differentiable at 0 and $l'(0) < 0$.
Therefore, the classifier that minimizes the expected hinge loss minimizes the expected zero-one loss.

6.2 Lagrangian Dual

Recall the soft-margin SVM: $\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^n (1 - y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b))^+$.

Apply $C \cdot (t)^+ := \max\{Ct, 0\} = \max_{0\le\alpha\le C} \alpha t$:

$$\min_{\mathbf{w},b} \max_{0\le\alpha\le C} \ \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^n \alpha_i [1 - y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b)]$$

We can swap min with max because strong duality holds due to convexity:

$$\max_{0\le\alpha\le C} \min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^n \alpha_i [1 - y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b)]$$

Solving the inner unconstrained problem by setting gradient to 0:

$$\frac{\partial}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = \mathbf{0}, \frac{\partial}{\partial b} = -\sum_{i=1}^n \alpha_i y_i = 0$$

We can plug back to eliminate the inner problem to get the dual:

$$\max_{0\le\alpha\le C} \sum_i \alpha_i - \frac{1}{2}\left\|\sum_i \alpha_i y_i \mathbf{x}_i\right\|_2^2 \text{ s.t. } \sum_i \alpha_i y_i = 0$$

We can change max to min and expand the norm:

$$\min_{0\le\alpha\le C} \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle - \sum_i \alpha_i \text{ s.t. } \sum_i \alpha_i y_i = 0$$

If $C \to \infty$, then it is a hard-margin SVM. If $C \to 0$, then it is a constant classifier.

6.3 Complementary Slackness

The following relation is to introduce dual variables:

$$C \cdot (t)^+ = \max\{Ct, 0\} = \max_{0\le\alpha\le C} \alpha t = \alpha^* t$$

The complementary slackness conditions are

- $t > 0 \implies \alpha^* = C$, $\alpha^* = C \implies t \ge 0$
- $t < 0 \implies \alpha^* = 0$, $\alpha^* = 0 \implies t \le 0$

6.4 Recovering Parameters

We can use α to determine the location of data points.

Recovering $\mathbf{w}^* := \sum_i \alpha_i^* y_i \mathbf{x}_i$.

We normally set C large enough so that there is at least one data point on $H_{\pm 1}$ or $y_i \hat{y}_i = 1$.

This point can be used to recover b^*: $y(\langle \mathbf{x}, \mathbf{w}^* \rangle + b^*) = 1 \implies b^* = y - \langle \mathbf{x}, \mathbf{w}^* \rangle$.

6.5 Training by Gradient Descent

$$\min_{\mathbf{w},b} \frac{1}{2\lambda} \|\mathbf{w}\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} l(y_i \hat{y}_i)$$

Gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left[\frac{\mathbf{w}}{\lambda} + \frac{1}{n} \sum_{i=1}^{n} l'(y_i \hat{y}_i) y_i \mathbf{x}_i \right]$$

The gradient of the hinge loss is

$$l'_{hinge}(t) = \begin{cases} -1 & t < 1 \\ 0 & t > 1 \\ [-1, 0] & t = 1 \end{cases} \text{ while we choose } l'_{perceptron}(t) = \begin{cases} -1 & t \leq 0 \\ 0 & t > 0 \end{cases}$$

Chapter 7

Reproducing Kernels

Linear classifiers may not be able to separate the data, such as the XOR dataset. We need a more complex classifier.

Definition: Quadratic Classifier

$$f(\mathbf{x}) = \langle \mathbf{x}, Q\mathbf{x} \rangle + \sqrt{2} \langle \mathbf{x}, \mathbf{p} \rangle + b$$

We predict as before: $\hat{y} = \text{sign}(f(\mathbf{x}))$ where $\mathbf{x} \in \mathbb{R}^d$. The weights can be learned: $Q \in \mathbb{R}^{d \times d}, \mathbf{p} \in \mathbb{R}^d, b \in R$.

Note that setting $Q = \mathbf{0}$ reduces to the linear case.

Definition: Vectorization

For any matrix $A \in \mathbb{R}^{m \times n}$, let $\vec{A}$ be the vectorization operation from $\mathbb{R}^{m \times n} \to \mathbb{R}^{mn}$ by copying each row and appending the bottom as a column vector.

$$\begin{aligned} f(\mathbf{x}) &= \langle \mathbf{x}, Q\mathbf{x} \rangle + \sqrt{2} \langle \mathbf{x}, \mathbf{p} \rangle + b \\ &= \left\langle \mathbf{x}\mathbf{x}^T, Q \right\rangle + \left\langle \sqrt{2}\mathbf{x}, \mathbf{p} \right\rangle + b \\ &= \langle \phi(\mathbf{x}), \mathbf{w} \rangle \end{aligned}$$

We define the feature map

$$\phi(\mathbf{x}) = \begin{bmatrix} \vec{\mathbf{x}\mathbf{x}^T} \\ \sqrt{2}\mathbf{x}, 1 \end{bmatrix}$$

where $\mathbf{x} \in \mathbb{R}^d \mapsto \phi(\mathbf{x}) \in \mathbb{R}^{d \times d + d + 1}$. The weights to be learned:

$$\mathbf{w} = \begin{bmatrix} \vec{Q} \\ \mathbf{p} \\ b \end{bmatrix} \in \mathbb{R}^{d \times d + d + 1}$$

The inner product of two matrices $\langle A, B \rangle$ is the inner product of the vectorization of A and B.

The function is nonlinear in $\mathbf{x}$, but linear in $\phi(\mathbf{x})$.

7.1 The Kernel Trick

The feature map $\phi : \mathbb{R}^d \to \mathbb{R}^{d \times d + d + 1}$ has huge dimensions. We do not have to operate in this high-dimensional feature space explicitly. In the dual form of the SVM, all we need is the inner product.

$$\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \left\langle \begin{bmatrix} \vec{\mathbf{x}\mathbf{x}^T} \\ \sqrt{2}\mathbf{x} \\ 1 \end{bmatrix}, \begin{bmatrix} \vec{\mathbf{z}\mathbf{z}^T} \\ \sqrt{2}\mathbf{z} \\ 1 \end{bmatrix} \right\rangle = (\langle \mathbf{x}, \mathbf{z} \rangle)^2 + 2 \langle \mathbf{x}, \mathbf{z} \rangle + 1 = (\langle \mathbf{x}, \mathbf{z} \rangle + 1)^2$$

We see that the inner product can be computed by the original vectors and can be done in $O(d)$ runtime.

We saw that given a feature map $\phi : \mathcal{X} \to \mathcal{H}$, $\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = k(\mathbf{x}, \mathbf{z})$. Can we do the converse: given $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, does there exist $\phi : \mathcal{X} \to \mathcal{H}$.

Definition: (Reproducing) Kernel

We call $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a (reproducing) kernel if and only if there exists some feature transform $\phi : \mathcal{X} \to \mathcal{H}$ so that $\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = k(\mathbf{x}, \mathbf{z})$.

Choosing a feature transform ϕ determines the unique corresponding kernel k and choosing a kernel k determines some feature transform ϕ.

Definition: Positive Semi-Definite (PSD)

For any $\alpha \in \mathbb{R}^n$,

$$\langle \alpha, K\alpha \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K_{ij} \geq 0$$

Theorem (Mercer)

$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if and only if for any $n \in \mathbb{N}$, for any $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, the kernel matrix K such that $K_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$ is symmetric and positive semi-definite.

Therefore, we do not need to find ϕ, the kernel trick allows us to just use a kernel.

Definition: Polynomial Kernel

Let p be a hyper-parameter, then

$$k(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + 1)^p$$

where a larger p means a higher-degree polynomial mapping ϕ.

Definition: Gaussian Kernel

$$k(\mathbf{x}, \mathbf{z}) = \exp(- \|\mathbf{x} - \mathbf{z}\|_2^2 / \sigma)$$

Definition: Laplace Kernel

Let σ be a hyper-parameter, then

$$k(\mathbf{x}, \mathbf{z}) = \exp(- \|\mathbf{x} - \mathbf{z}\|_2 / \sigma)$$

where the larger σ means a smoother ϕ, i.e. $\phi(x_1)$ and $\phi(x_2)$ will not differ too much for close x_1, x_2.

Proposition

If k is a kernel, then λk is a kernel for any $\lambda \geq 0$.

Proposition

If k_1 and k_2 are kernels, then $k_1 + k_2$ is a kernel.

Proposition

If k_1 and k_2 are kernels, then $k_1 k_2$ is a kernel.

Proposition

If k_t are kernels, then the limit $\lim_t k_t$, when it exists, is a kernel.

7.2 Kernel SVM

The dual for the soft-margin SVM is now

$$\begin{aligned} \min_{C \geq \alpha \geq 0} \quad & -\sum_i \alpha_i + \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s.t.} \quad & \sum_i \alpha_i y_i = 0 \end{aligned}$$

We can solve for $\alpha^* \in \mathbb{R}^n$ and recover $\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i)$. We do not know ϕ, so for testing, we only need to compute

$$f(\mathbf{x}) = \langle \phi(\mathbf{x}), \mathbf{w}^* \rangle = \left\langle \phi(\mathbf{x}), \sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i) \right\rangle = \sum_{i=1}^{n} \alpha_i^* y_i k(\mathbf{x}, \mathbf{x}_i)$$

We only need to know the dual vector α^*, the training set and kernel k.

Chapter 8

Gradient Descent

Many machine learning models can be formulated as an optimization problem.

Consider an unconstrained optimization

$$\min_x f(x)$$

Assume that f is differentiable with gradient $\nabla f(x)$. Denote the optimal criterion value by $f^* = \min_x f(x)$ and an optimal solution x^*.

Gradient Descent Template

Choose initial $x^{(0)} \in \mathbb{R}^n$ and repeat

$$x^{(k)} = x^{(k-1)} - \underbrace{t}_{\text{step size}} \cdot \nabla f(x^{(k-1)})$$

for $k = 1, 2, \ldots$.

The **negative gradient** is the steepest decreasing direction at that point. So if the step size is small and the function is convex, the algorithm will reach the minimizer.

8.1 Gradient Descent in ML Models

Perceptron:

$$\min_{\mathbf{w}} -\frac{1}{n} \sum_{i=1}^{n} y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \mathbb{I}[\text{mistake on } \mathbf{x}_i]$$

The gradient descent update is

$$\mathbf{w} \leftarrow \mathbf{w} + t \left[\frac{1}{n} \sum_{i=1}^{n} y_i \mathbf{x}_i \mathbb{I}[\text{mistake on } \mathbf{x}_i] \right]$$

Stochastic gradient descent update is

$$\mathbf{w} \leftarrow \mathbf{w} + t y_I \mathbf{x}_I \mathbb{I}[\text{mistake on } \mathbf{x}_i]$$

for some random index I.

Soft-Margin SVM:

$$\min_{\mathbf{w},b} \frac{1}{2\lambda} \|\mathbf{w}\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} l_{hinge}(y_i \hat{y}_i) \text{ s.t. } \hat{y}_i = \langle \mathbf{x}_i, \mathbf{w} \rangle + b$$

The gradient descent update is

$$\mathbf{w} \leftarrow \mathbf{w} - t \left[\frac{\mathbf{w}}{\lambda} + \frac{1}{n} \sum_{i=1}^{n} l'_{hinge}(y_i \hat{y}_i) y_i \mathbf{x}_i \right]$$

$$b \leftarrow b - t \left[\frac{1}{n} \sum_{i=1}^{n} l'_{hinge}(y_i \hat{y}_i) y_i \right]$$

8.2 Convergence Analysis

8.2.1 Taylor Series

Consider the min operation on the Taylor expansion of f locally at x, where x is the current iterate:

$$\min_y f(y) \approx \min_y \left[f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t} \|y - x\|_2^2 \right]$$

Taking the gradient with respect to y,

$$\frac{\partial f(y)}{\partial y} = 0 \implies \nabla f(x) + \frac{1}{t}(y - x) = 0 \implies y^* = x - t \cdot \nabla f(x)$$

Geometrically, Choose the next point $y = x^+$ to minimize the RHS:

$$x^{(i+1)} = \arg\min_y f(x^{(i)}) + \nabla f(x^{(i)})^T (y - x) + \frac{1}{2t} \left\| y - x^{(i)} \right\|_2^2$$

This will diverge if t is too large. If t is too small, it will be too slow.

8.2.2 Convex Case

Definition: Convex Function (Differentiable)

For any $x, y \in \mathbb{R}^n$,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$

For thin convex functions, L is bigger and choosing a step size that is too large will jump to the other side of the function, so it will diverge. For wider convex functions, L can be smaller and we are able to choose a larger step size.

Theorem (Convergence Rate For Convex Case)

Assume f is convex and differentiable, with $\text{dom}(f) = \mathbb{R}^n$, and ∇f is L-Lipschitz continuous: $L\mathbf{I} - \nabla^2 f(x)$ is positive semi-definite ($L\mathbf{I} \succeq \nabla^2 f(x)$). Gradient descent with fixed step size $t \leq \frac{1}{L}$ satisfies

$$f(x^{(k)}) - f^* \leq \frac{\left\|x^{(0)} - x^*\right\|_2^2}{2tk}$$

We say gradient descent has convergence rate $O(1/k)$. That is, it finds ϵ-suboptimal point in $O(1/\epsilon)$ iterations.

Proof. For any y, perform a quadratic expansion and obtain:

$$\begin{aligned}
f(y) &\leq f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}(y-x)^T\nabla^2 f(x)(y-x) \\
&\leq f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}L\|y-x\|_2^2 && (L\mathbf{I} \succeq \nabla^2 f(x)) \\
f(x^+) &\leq f(x) + \nabla f(x)^T(x^+-x) + \frac{1}{2}L\left\|x^+-x\right\|_2^2 && (y = x^+ = x - t\nabla f(x)) \\
&= f(x) + \nabla f(x)^T(x - t\nabla f(x) - x) + \frac{1}{2}L\|x - t\nabla f(x) - x\|_2^2 \\
&= f(x) - \left(1 - \frac{1}{2}Lt\right)t\|\nabla f(x)\|_2^2 && (1) \\
&\leq f(x) - \frac{1}{2}t\|\nabla f(x)\|_2^2 && (t \leq 1/L)
\end{aligned}$$

Each update decreases the function value by at least $\frac{1}{2}t\|\nabla f(x)\|_2^2$. Since f is convex

$$f(x^*) \geq f(x) + \nabla f(x)^T(x^* - x) \implies f(x) \leq f(x^*) + \nabla f(x)^T(x - x^*)$$

Plug in (1)

$$\begin{aligned}
f(x^+) &\leq f(x^*) + \nabla f(x)^T(x - x^*) - \frac{t}{2}\|\nabla f(x)\|_2^2 \\
f(x^+) - f(x^*) &\leq \frac{1}{2t}(2t\nabla f(x)^T(x - x^*) - t^2\|\nabla f(x)\|_2^2) \\
f(x^+) - f(x^*) &\leq \frac{1}{2t}(2t\nabla f(x)^T(x - x^*) - t^2\|\nabla f(x)\|_2^2 - \|x - x^*\|_2^2 + \|x - x^*\|_2^2) \\
f(x^+) - f(x^*) &\leq \frac{1}{2t}(\|x - x^*\|_2^2 - \|x - t\nabla f(x) - x^*\|_2^2) \\
f(x^+) - f(x^*) &\leq \frac{1}{2t}(\|x - x^*\|_2^2 - \left\|x^+ - x^*\right\|_2^2)
\end{aligned}$$

Sum over all iterations:

$$\begin{aligned}\sum_{i=1}^{k}(f(x^{(i)})-f(x^*)) &\le \sum_{i=1}^{k}\frac{1}{2t}(\left\|x^{(i-1)}-x^*\right\|_2^2-\left\|x^{(i)}-x^*\right\|_2^2)\\ &=\frac{1}{2t}(\left\|x^{(0)}-x^*\right\|_2^2-\left\|x^{(k)}-x^*\right\|_2^2)\\ &\le \frac{1}{2t}\left\|x^{(0)}-x^*\right\|_2^2\end{aligned}$$

which implies

$$f(x^{(k)})\le \frac{1}{k}\sum_{i=1}^{k}f(x^{(i)})\le f(x^*)+\frac{\left\|x^{(0)}-x^*\right\|_2^2}{2tk}$$

The first inequality holds because $f(x^{(i)})$ decreases as i increases.

8.2.3 Strong Convex Case

Definition: m-Strong Convexity

A function f is m-strong convex if $f(x)-m\left\|x\right\|_2^2$ is convex: $L\mathbf{I}\succeq\nabla^2 f(x)\succeq m\mathbf{I}$.

Theorem (Convergence Rate For Strong Convexity)

Let f be m-strongly convex and L-smooth. Gradient descent with fixed step size $t\le\frac{2}{m+L}$ satisfies

$$f(x^{(k)})-f^*\le\gamma^k\frac{L}{2}\left\|x^{(0)}-x^*\right\|_2^2$$

where $0\le\gamma\le 1$.

The rate under strong convexity is $O(\gamma^k)$, which is exponentially fast. That is, it finds ϵ-suboptimal point, i.e. $f(x^{(k)})-f^*<\epsilon$, in $O(\log(1/\epsilon))$ iterations.

Asking for optimality is too much. Focus on $\left\|\nabla f(x)\right\|_2\le\epsilon$.

8.2.4 Nonconvex Case

Assume f is differentiable with L-Lipschitz gradient and f is nonconvex.

Theorem (Convergence Rate For Nonconvex Case)

Gradient descent with fixed step size $t\le\frac{1}{L}$ satisfies

$$\min_{i=0,\dots,k}\left\|\nabla f(x^{(i)})\right\|_2^2\le\sqrt{\frac{2(f(x^{(0)})-f^*)}{t(k+1)}}$$

The gradient descent rate is $O(1/\sqrt{k})$ or $O(1/\epsilon^2)$, even in the nonconvex case for finding stationary points.

This rate cannot be improved by any deterministic algorithm.

8.3 Stochastic Gradient Descent

Consider the decomposable optimization (n is very large)

$$\min_w \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

Gradient descent has $w^+ = w - w - t\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x)$, but stochastic gradient descent has

$$w^+ = w - t \cdot \nabla f_I(x)$$

where I is a random index. The expectation of gradient descent is the same as stochastic gradient descent.

For gradient descent, the step size $t \leq 1/L$ and the convergence rate is $O\left(\frac{n}{\epsilon}\right)$. For stochastic gradient descent, the step size is $t = 1/k$ for $k = 1, 2, \ldots$ and the convergence rate is $O\left(\frac{1}{\sqrt{k}}\right)$. The randomness leads to a large variance of estimation of the gradient. Thus, SGD requires more iterations.

Part II

Neural Networks

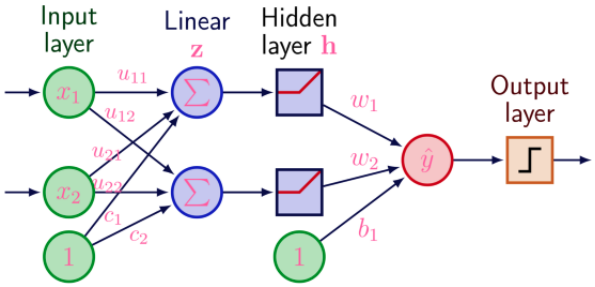
Chapter 9

Multilayer Perceptron/Fully Connected Neural Networks

9.1 Introduction

We have seen that no linear classifier can separate the XOR dataset, so it underfits the data. We can fix the input representation using a richer model (e.g. quadratic classifier) or keep it linear, but use a richer input representation. The two approaches are equivalent for certain classifiers, using a reproducing kernel.

A neural network can learn the feature map simultaneously with the linear classifier.



First linear transformation: $\mathbf{z} = U\mathbf{x} + \mathbf{c}$ where $U \in \mathbb{R}^{2\times 2}, c \in \mathbb{R}^2$.

Element-wise *nonlinear* activation function: $\mathbf{h} = \sigma(\mathbf{z})$.

Second linear transformation: $\hat{y} = \langle \mathbf{h}, \mathbf{w} \rangle + b$.

Output layer: $\text{sign}(\hat{y})$ or $\text{sigmoid}(\hat{y})$.

XOR Dataset: Consider a 2-layer NN:

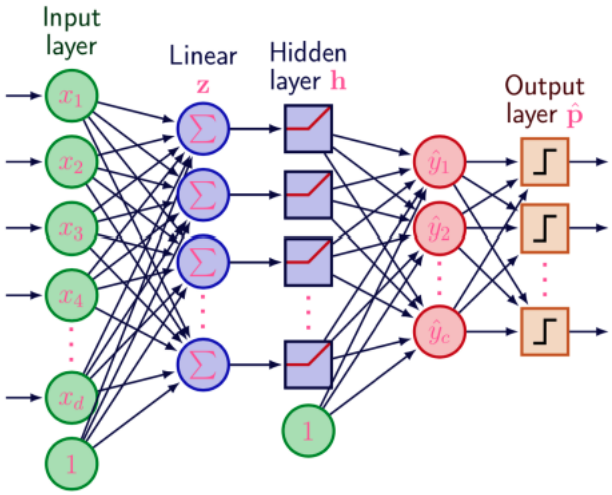
- 1st linear transformation: $\mathbf{z} = U\mathbf{x} + \mathbf{c}$ where $U \in \mathbb{R}^{2\times 2}, \mathbf{c} \in \mathbb{R}^2$
- Element-wise nonlinear activation: $\mathbf{h} = \sigma(\mathbf{z})$; use the Rectified Linear Unit (ReLU): $\sigma(t) = t_+ := \max\{t, 0\}$

- 2nd linear transformation: $\hat{y} = \langle \mathbf{h}, \mathbf{w} \rangle + b$

$$U = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 2 \\ -4 \end{bmatrix}, b = -1$$

- $\mathbf{x}_1 = (0,0), y = - \implies \mathbf{z}_1 = (0,-1), \mathbf{h}_1 = (0,0) \implies \hat{y} = -1$ ✓
- $\mathbf{x}_2 = (1,0), y = + \implies \mathbf{z}_2 = (1,0), \mathbf{h}_1 = (1,0) \implies \hat{y} = +1$ ✓
- $\mathbf{x}_3 = (0,1), y = + \implies \mathbf{z}_3 = (1,0), \mathbf{h}_1 = (1,0) \implies \hat{y} = +1$ ✓
- $\mathbf{x}_4 = (1,1), y = - \implies \mathbf{z}_4 = (2,1), \mathbf{h}_1 = (2,1) \implies \hat{y} = -1$ ✓

9.2 Multiclass Classification



$$\underbrace{\mathbf{z} = U\mathbf{x} + \mathbf{c}, \mathbf{h} = \sigma(\mathbf{z})}_{\text{learning feature } \mathbf{h}}, \quad \underbrace{\hat{\mathbf{y}} = W\mathbf{h} + \mathbf{b}, \hat{\mathbf{p}} = \text{softmax}(\hat{\mathbf{y}})}_{\text{learning linear classifier by logistic regression}}$$

9.3 Activation Functions

Definition: Sigmoid

$$\text{sgm}(t) = \frac{1}{1 + \exp(-t)}$$

Definition: Hyperbolic Tangent

$$\tanh(t) = 1 - 2\,\text{sgm}(2t)$$

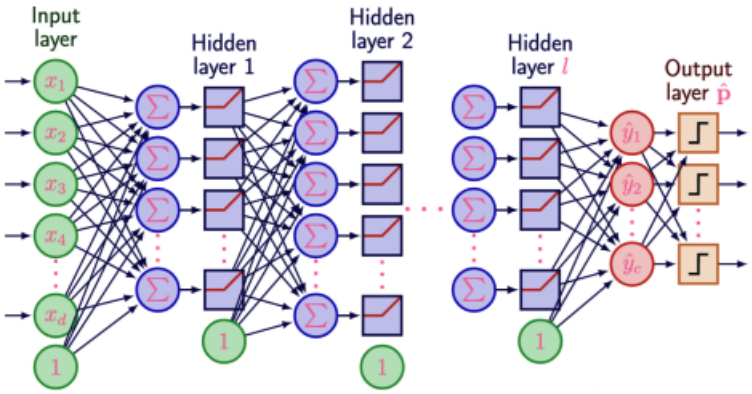
Definition: Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(t) = t_+ := \max\{t, 0\}$$

Definition: Exponential Linear Unit

$$\mathrm{ELU}(t) = (t)_+ + (t)_-(\exp(t) - 1)$$

9.4 Training



$$\hat{\mathbf{p}} = f(\mathbf{x}; \mathbf{w})$$

We need a loss function l to measure the difference between the prediction $\hat{\mathbf{p}}$ and the truth y. We can use squared loss $\|\hat{\mathbf{p}} - \mathbf{y}\|_2^2$ for regression or log-loss $-\log \hat{p}_y$ for classification.

We also need a training set $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$ to train the weights $\mathbf{w}$.

Stochastic Gradient Descent: Let $[l \circ f](\mathbf{x}_i, y_i; \mathbf{w}) := l[f(\mathbf{x}_i; \mathbf{w}), y_i]$

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} [l \circ f](\mathbf{x}_i, y_i; \mathbf{w})$$

The update is

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla [l \circ f](\mathbf{x}_i, y_i; \mathbf{w})$$

However, each iteration requires going through the entire training set, so we can just take a random minibatch $B \subseteq \{1, \ldots, n\}$.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \frac{1}{|B|} \sum_{i \in B} \nabla [l \circ f](\mathbf{x}_i, y_i; \mathbf{w})$$

There is trade-off between variance and computation time using randomness.

The learning rate decay decreases every few epochs: $\eta_t = \begin{cases} \eta_0 & t \leq t_0 \\ \eta_0/10 & t_0 < t \leq t_1 \\ \eta_0/100 & t_1 < t \end{cases}$.

Sublinear decay: $\eta_t = \eta_0/(1+ct)$ or $\eta_t = \eta_0/\sqrt{1+ct}$.

9.4.1 Computing the Gradient

The forward pass of the model is (these are calculated):

$$\begin{aligned} \mathbf{x} &= \text{input} \\ \mathbf{z} &= W\mathbf{x} + \mathbf{b}_1 \\ \mathbf{h} &= \text{ReLU}(\mathbf{z}) \\ \boldsymbol{\theta} &= U\mathbf{h} + \mathbf{b}_2 \\ J &= \frac{1}{2}\|\boldsymbol{\theta} - \mathbf{y}\|_2^2 \end{aligned}$$

The parameters to be learned are $W, \mathbf{b}_1, U, \mathbf{b}_2$. The network's gradient are $\frac{\partial J}{\partial W}, \frac{\partial J}{\partial \mathbf{b}_1}, \frac{\partial J}{\partial U}, \frac{\partial J}{\partial \mathbf{b}_2}$. The gradient of ReLU is

$$\text{ReLU}'(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Proposition

If $J : \mathbb{R}^{m\times n} \to \mathbb{R}$, then $\frac{\partial J(X)}{\partial X} \in \mathbb{R}^{m\times n}$. The output of the gradient is the same shape as the input.

Definition: Hadamard Product

If $a, b \in \mathbb{R}^n$, then

$$a \odot b = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{bmatrix}$$

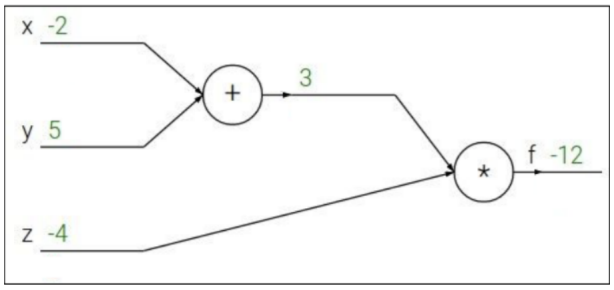
We can use the chain rule: the backward pass (back-propagation) of the model ($\odot$ is the

Hadamard product):

$$
\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{\theta}} &= \boldsymbol{\theta} - \mathbf{y} \\
\frac{\partial J}{\partial U} &= \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial U} = (\boldsymbol{\theta} - \mathbf{y})\mathbf{h}^T && (*) \\
\frac{\partial J}{\partial \mathbf{b}_2} &= \frac{dJ}{d\boldsymbol{\theta}} \frac{\partial \theta}{\partial \mathbf{b}_2} = \boldsymbol{\theta} - \mathbf{y} && (*) \\
\frac{\partial J}{\partial \mathbf{h}} &= \frac{\partial J}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{h}} = U^T(\boldsymbol{\theta} - \mathbf{y}) \\
\frac{dJ}{d\mathbf{z}} &= \frac{\partial J}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} = U^T(\boldsymbol{\theta} - \mathbf{y}) \odot \mathrm{ReLU}'(\mathbf{z}) \\
\frac{\partial J}{\partial W} &= \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial W} = (U^T(\boldsymbol{\theta} - \mathbf{y}) \odot \mathrm{ReLU}'(\mathbf{z}))\mathbf{x}^T && (*) \\
\frac{\partial J}{\partial \mathbf{b}_1} &= \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{b}_1} = U^T(\boldsymbol{\theta} - \mathbf{y}) \odot \mathrm{ReLU}'(\mathbf{z}) && (*)
\end{aligned}
$$

Existing frameworks will memorize the computational graph in the forward process and calculate the back-propagation automatically for you.

Example: $f(x, y, z) = (x + y)z$ with $x = -2, y = 5, z = -4$.



$q = x + y$, $\frac{\partial q}{\partial x} = 1$, $\frac{\partial q}{\partial y} = 1$ and $f = qz$, $\frac{\partial f}{\partial q} = z$, $\frac{\partial f}{\partial z} = q$. We want to calculate $\frac{\partial f}{\partial x}, \frac{df}{dy}, \frac{\partial f}{\partial z}$.

$$
\begin{aligned}
\frac{\partial f}{\partial z} &= q = 3 \\
\frac{\partial f}{\partial q} &= z = -4 \\
\frac{\partial f}{\partial y} &= \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = -4 \cdot 1 = -4 \\
\frac{\partial f}{\partial x} &= \frac{df}{dq} \frac{\partial q}{\partial x} = -4 \cdot 1 = -4
\end{aligned}
$$

9.5 Universal Representation

Theorem (Universal Approximation Theorem by 2-Layer Neural Networks)

For any continuous function $f : \mathbb{R}^d \to \mathbb{R}^c$ and any $\epsilon > 0$, there exists $k \in \mathbb{N}, W \in \mathbb{R}^{k \times d}, \mathbf{b} \in \mathbb{R}^k, U \in \mathbb{R}^{c \times k}$ such that

$$\sup_{\mathbf{x}} \|f(\mathbf{x}) - g(\mathbf{x})\|_2 < \epsilon$$

where $g(\mathbf{x}) = U(\sigma(W\mathbf{x} + \mathbf{b}))$ and σ is the element-wise ReLU operation.

As long as 2-layer MLP is wide enough (large k), it can approximate any continuous function arbitrarily closely.

Why Deep Learning: There exists a function such that a 2-layer MLP needs to be exponentially wide to approximate this function, whereas a 3-layer MLP only needs to be polynomial wide. Deep neural networks are more parameter efficient.

Theorem (Addition is the Only Continuous Multivariate Function)

For any $d \in \mathbb{N}$, there exist constants $\lambda_j > 0$, for $j = 1, \ldots, d$, $\mathbf{1}^T\lambda < 1$ and strictly increasing Lipschitz continuous functions $\varphi_k : [0, 1] \to [0, 1]$ for $k = 1, \ldots, 2d + 1$ such that for any continuous function $f : [0, 1]^d \to \mathbb{R}$, there exists some continuous univariate function $\sigma : [0, 1] \to \mathbb{R}$ so that

$$f(\mathbf{x}) = \sum_{k=1}^{2d+1} \sigma \left(\sum_{j=1}^{d} \lambda_j \varphi_k(x_j) \right)$$

9.5.1 Dropout

For each training minibatch, keep each hidden unit with probability q. We want a different and random network for each training minibatch.

Hidden units are less likely to collude to overfit training data.

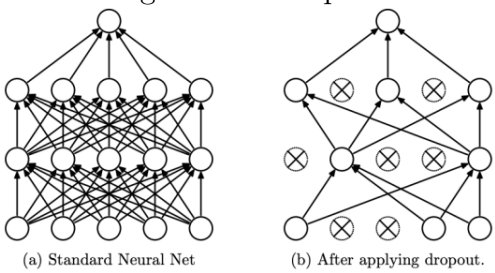
Inverted: multiply each $\mathbf{h}_k$ with a scaled binary mask $\mathbf{m}/q$. We use the full network for testing.

9.5.2 Batch Normalization

Definition: Batch Normalization

Normalize across a minibatch, independently for each feature.

Figure 9.1: Dropout



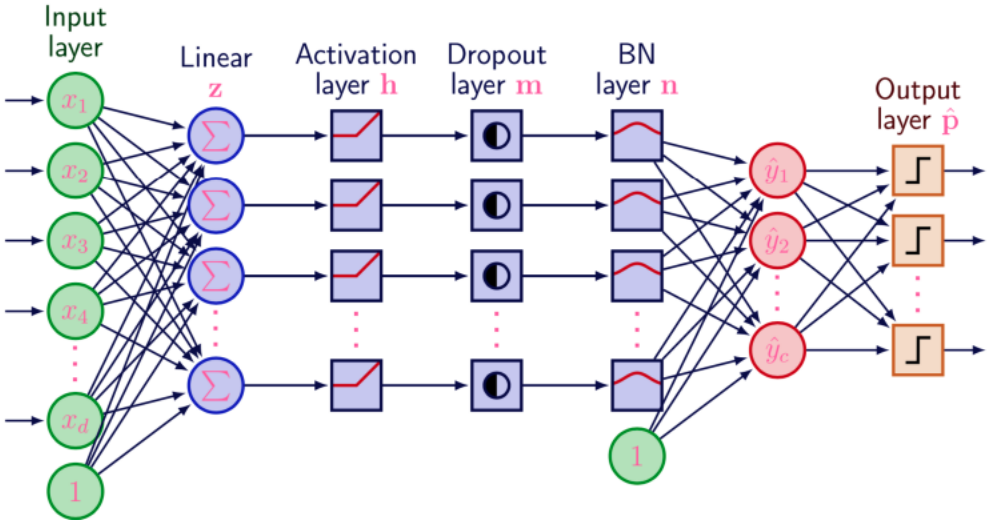
(a) Standard Neural Net (b) After applying dropout.

1: **Input**: Values of x over a minibatch $\mathcal{B} = \{x_{1\ldots}\}$
Parameters to be learned: γ, β
2: **Output**: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$
3: $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ ▷ minibatch mean
4: $\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$ ▷ minibatch variance
5: $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$ ▷ normalize
6: $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$

Definition: Layer Normalization

Normalize across features, independently for each sample.

Figure 9.2: Complete Multilayer Perceptron Architecture



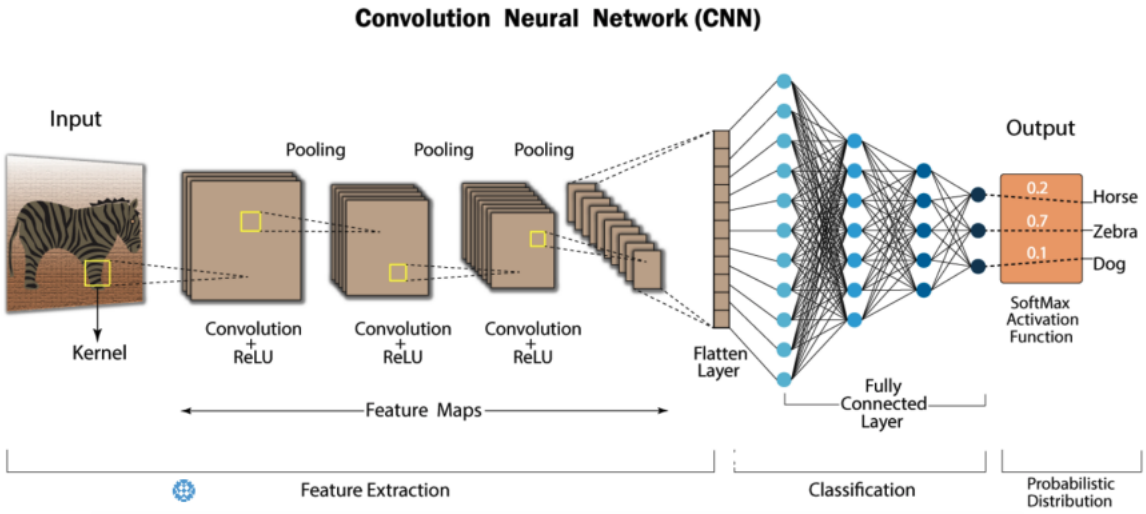
Chapter 10

Convolutional Neural Networks

The multilayer perceptron had function $f(\mathbf{x}) = W_2\sigma(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$. It had dense weights; each connection represents a weight to be learned, however, it is easy to overfit the training data.

We can consider weight sharing and a sparse weight matrix. We use the same weight for more than one connection and connect less weights.

Layers in a Convolutional Neural Network



Forms of Image Data

- Original image
- Red-green-blue channels: each channel are the RGB images
- Greyscale image

In Pytorch, we represent as a *tensor*. For coloured images, (b, c, m, n) to represent the tensor (batch, channels, rows, columns). For greyscale images, $(b, 1, m, n)$ or (b, m, n).

10.1 Convolution

Definition: Convolution

Let A be the convolutional kernel and X be the input matrix, then the convolution $A * X$ has output size A where each element of $A * X$ is the inner product of A with the $|A|$ submatrix of X.

Definition: One Channel Input

Take the inner product of the convolutional filter with the same size of the input matrix, which will be one element of the output matrix.

The human visual system uses the convolution operation. Traditional image processing algorithms also use the convolution operation.

Definition: Multichannel Input

Take the inner product of each convolutional kernel with the input channel and sum them all for each channel with a bias.

Definition: Filter (Kernel) Size

Width $\times$ height. By default, the number of channels of each filter is the same as that of that of the input.

Definition: Number of Kernels

Weights are not shared between different filters; determine the number of channels of output.

The number of weights to be learned is $c_{in} \times$ kernel size $\times\, c_{out}$.

Definition: Stride

Number of pixels to move the filter each time.

Typical stride $\leq$ filter size as to leave no gap. A larger stride makes neighbouring outputs less similar due to less overlap in the input window.

Definition: Padding

Add zeros (or other values) around boundary of input.

10.2 Size Calculation

Input size: $m \times n \times c$

Filter size: $a \times b \times c$

Stride: $s \times t$

Padding: $p \times q$

Pad p pixels on left/right and q pixels on top/bottom (typically $p = q$).

Move s pixels horizontally and t pixels vertically (typically $s = t$).

Output size: $\left\lfloor 1 + \frac{m+2p-a}{s} \right\rfloor \times \left\lfloor 1 + \frac{n+2q-b}{t} \right\rfloor$.

With $p = \left\lceil \frac{m(s-1)+a-s}{2} \right\rceil$ and $q = \left\lceil \frac{n(t-1)+b-t}{2} \right\rceil$, the output size = input size.

10.3 Convolutional = FC Layer with Weight Sharing

$$W = \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{bmatrix} \in \mathbb{R}^{2\times 2}, X = \begin{bmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

The circulant matrix is

$$W_{circ} = \begin{bmatrix} w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 & 0 \\ 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 \\ 0 & 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} \end{bmatrix} \in \mathbb{R}^{4\times 9}$$

$$\text{Vector}(X) = [x_{00}, x_{01}, x_{02}, x_{10}, x_{11}, x_{12}, x_{20}, x_{21}, x_{22}]^T \in \mathbb{R}^9$$

The circulant matrix W_{circ} is a modification of W where the convolution of W and X will be equal to the matrix multiplication W_{circ} and Vector(X).

$$\text{Vector}(W * X) = W_{circ}\,\text{Vector}(X) \in \mathbb{R}^4$$

Suppose we had a fully connected network with size 9 to size 4, then there are $9 \times 4 = 36$ parameters to be learned, but with weight sharing, we only have 4 parameters to learn (the output size).

10.4 Pooling

Definition: Pooling

Down sample input size to reduce computation and memory.

By default, it is performed on each slice separately. Hence output depth equals input depth.

There are max-pool, average-pool, (averaged) l_p-norm pool.

Size and stride as in convolution; no parameter; typically no padding.

Definition: Global Pooling

Pooling size = input size.

10.5 CNN Architectures

Figure 10.1: LeNet

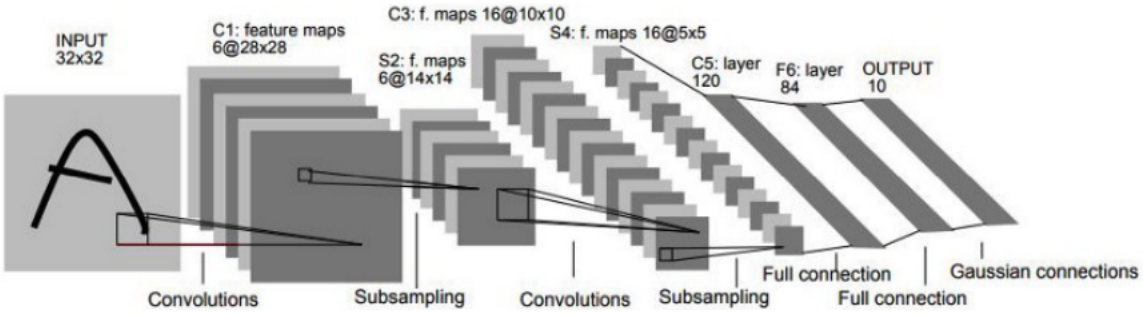


Figure 10.2: AlexNet

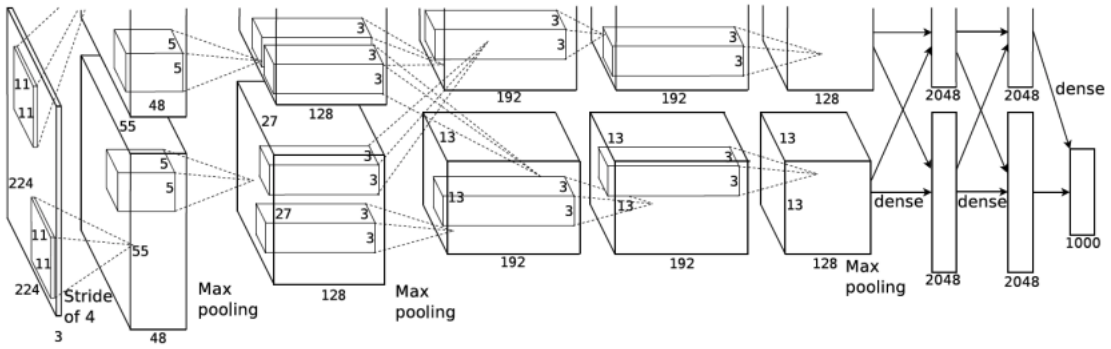


Figure 10.3: LeNet vs. AlexNet

10.5.1 Descriptions

Definition: 1×1 Convolution

Allows easy control of the depth (channels). It can save computation without affecting output size.

GoogLeNet: There are no fully connected (FC) layers. Deeper, but more efficient and better performance.

Deeper models can be harder to train due to vanishing/exploding gradient. This can be worse than shallower networks if not properly trained.

Residual Block: We can add a shortcut connection that allows skipping one or more layers. This effectively turns the block into learning residuals: output − input. It allows more direct backpropagation of the gradient through the "shortcut". We can also concatenate or add a linear layer if dimensions mismatch.

ResNet: The full ResNet architecture:

- Stack residual blocks.
- Every residual block has two 3×3 convolutional layers.
- Periodically, double number of filters and down sample spatially using stride 2.
- Additional convolutional layer at the beginning.
- No fully connected layers at the end (only fully connected 1000 to output classes).

Figure 10.4: VGGNet

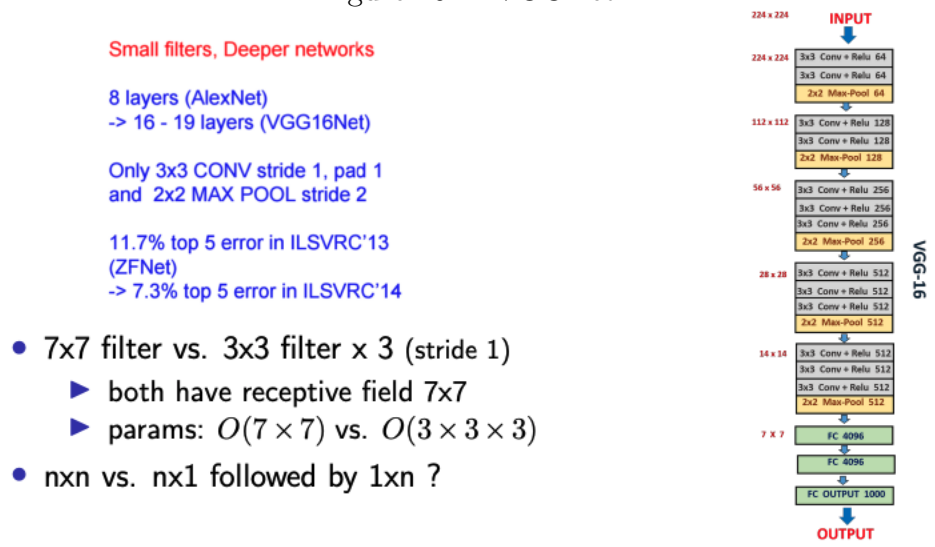
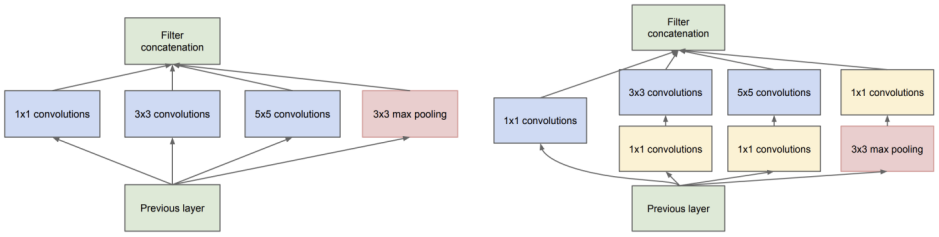


Figure 10.5: Inception



(a) Inception module, naïve version

(b) Inception module with dimension reductions

Figure 10.6: GoogLeNet

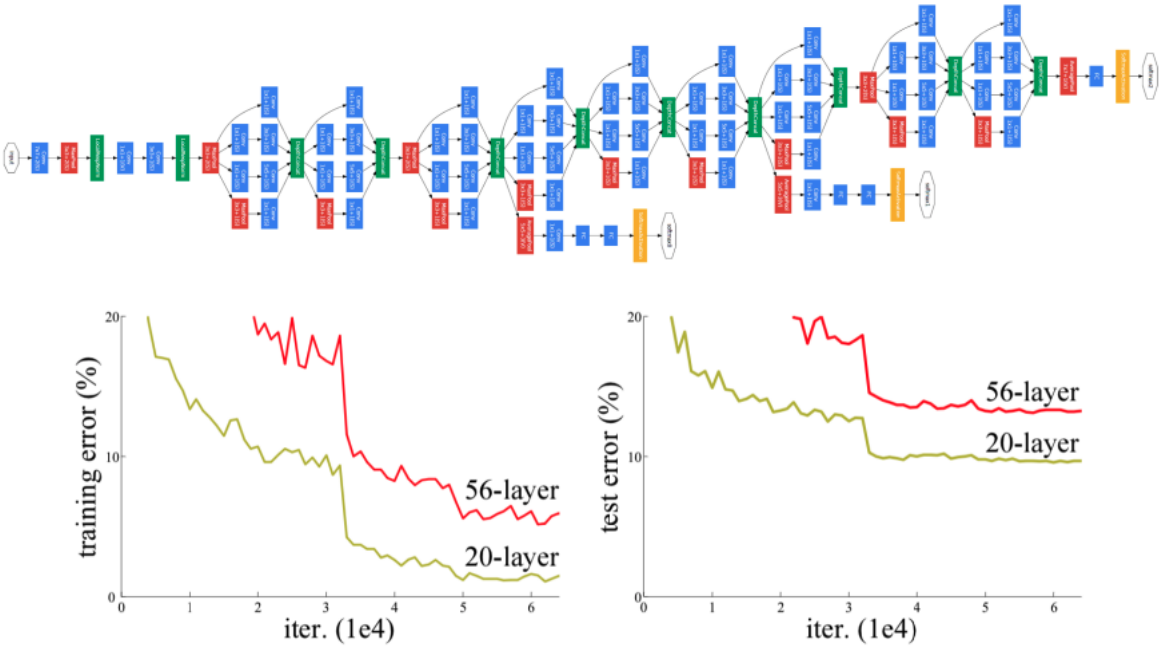


Figure 10.7: Residual Block

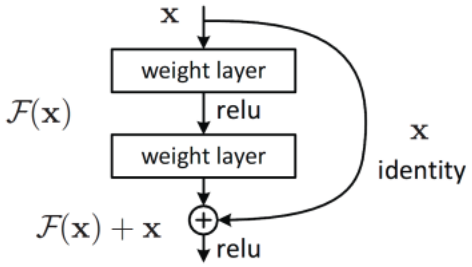
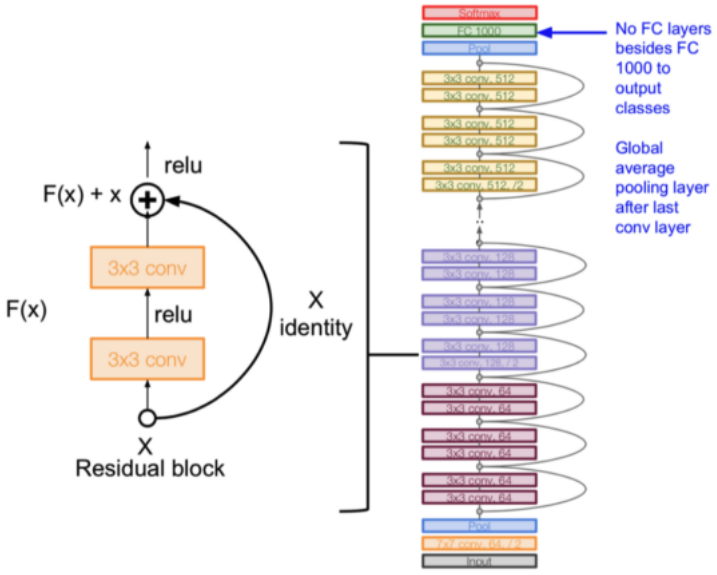


Figure 10.8: Residual Network (ResNet)



Chapter 11

Transformer

Capabilities of transformers: writing code and homework, analyzing texts, taking exams, reasoning, interacting with the real word, etc.

Papers to read:

- Transformer: Attention Is All You Need
- GPT: Improving Language Understanding by Generative Pre-training
- BERT: Pre-training of Deep Bidirectional Transformer for Language Understanding
- GPT-2: Language Models are Unsupervised Multitask Learners
- GPT-3: Language Models are Few-Shot Learners
- GPT-3.5: Training Language Models to Follow Instructions with Human Feedbacks
- GPT-4: Technical Report

Definition: Transformer

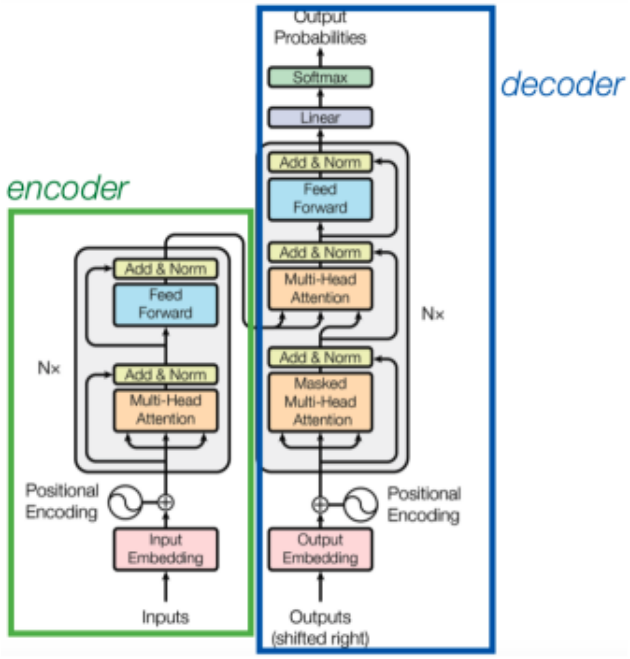
Designed for sequence-to-sequence text tasks (e.g. machine translation, question-answer, etc.). The mathematical goal is to output the probability

$$\arg\max_{Y} p(\mathbf{y}_1, \ldots, \mathbf{y}_m | \mathbf{x}_1, \ldots, \mathbf{x}_n)$$

Input and Output

Input Sequence: $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, the prompt.
Output Sequence: $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_m)$.
Goal: Compute $\arg\max_Y p(\mathbf{y}_1, \ldots, \mathbf{y}_m | \mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Figure 11.1: Transformer Architecture



Finding the total output probability might be hard, so we do one token after another (greedy): $\arg\max_{\mathbf{y}_k} p(\mathbf{y}_k|\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{y}_1, \ldots, \mathbf{y}_{k-1})$. This is also known as *auto-regressive*.

Example:

0. X: Where is University of Waterloo?
1. Y: [START]; $P(\text{It}|X\ [\text{START}])$ highest
2. Y: [START] It; $P(\text{is}|X\ [\text{START}]\ \text{It})$ highest
3. Y: [START] It is; $P(\text{in}|X\ [\text{START}]\ \text{It is})$ highest
4. Y: [START] It is in; $P(\text{Waterloo}|X\ [\text{START}]\ \text{It is in})$ highest
5. Y: [START] It is in Waterloo; $P([\text{END}]|X\ [\text{START}]\ \text{It is in Waterloo})$ highest
6. Y: [START] It is in Waterloo [END]

11.1 Input/Output Embedding Layer

Transformer Architecture: In the kth step, the input is $x_1, \ldots, x_n$ and the outputs (shifted right) is $y_1, \ldots, y_{k-1}$. The output probability is $p(y_k|x_1, \ldots, x_n, y_1, \ldots, y_{k-1})$.

Uses a tokenizer to break up text into individual words. *tiktoken* is a fast tokenizer used by OpenAI's models.

Definition: Token Embedding

A bijection from tokens to vectors.

Convert input tokens to vectors of dimension $d = 512$. Convert the decode outputted vectors to tokens.

Good properties: words of similar meaning are close in the embedding space.

11.2 Positional Encoding

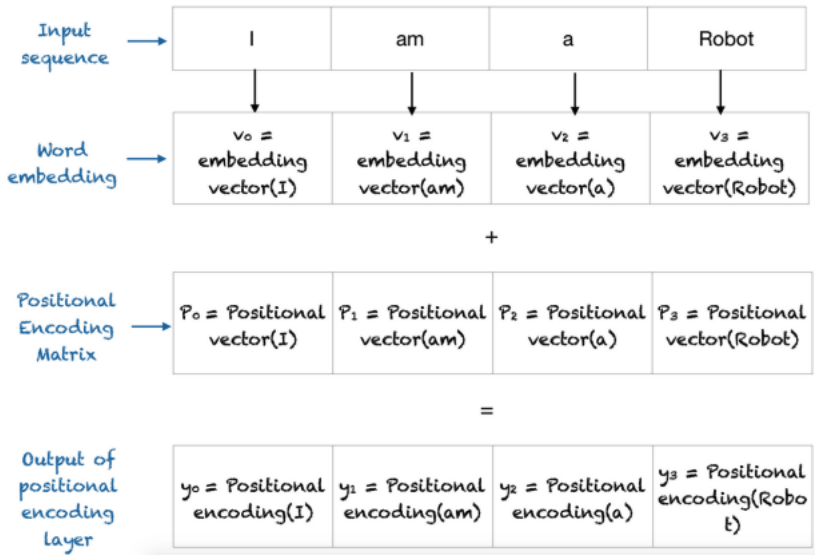
Word order matters, they produce different meanings.

Definition: Positional Encoding

$W^p \in \mathbb{R}^{n \times d}$.

$$W^p_{t,2i} = \sin(t/10000^{2i/d}), W^p_{t,2i+1} = \cos(t/10000^{2i/d}), i = 0, \ldots, \frac{d}{2} - 1$$

There are **NO** parameters to be learned. Simply add W^p to the $n \times d$ token embedding.

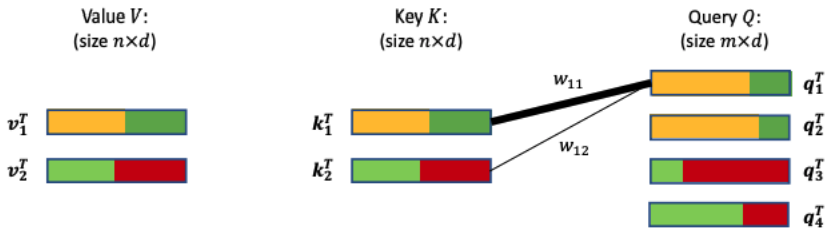


11.3 Attention Layer

Input and Output

Input: Value $V \in \mathbb{R}^{n \times d}$, Key $K \in \mathbb{R}^{n \times d}$, Query $Q \in \mathbb{R}^{m \times d}$.
Output: $m \times d$ matrix.

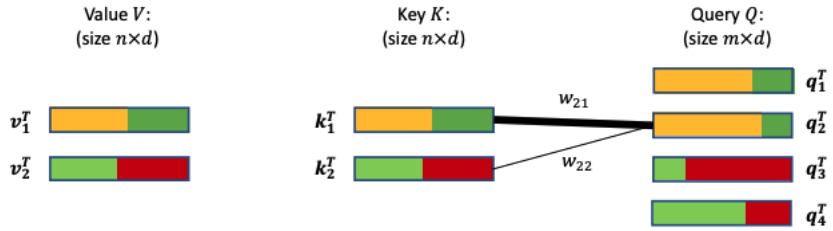
In the case where $V = K = Q$, they are the output of the positional encoder, and for $V = K \neq Q$, V, K are from the encoder and Q is from the decoder.



$w_{11} = \langle \mathbf{q}_1, \mathbf{k}_1 \rangle, w_{12} = \langle \mathbf{q}_1, \mathbf{k}_2 \rangle$, the first output row is

$$\text{Softmax}\left(\frac{w_{11}}{\sqrt{d}}\right)\mathbf{v}_1^T + \text{Softmax}\left(\frac{w_{12}}{\sqrt{d}}\right)\mathbf{v}_2^T$$

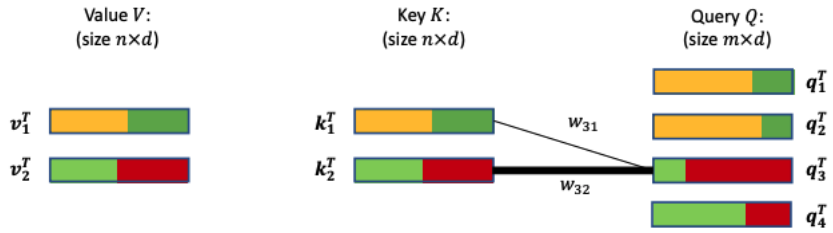
where $\mathbf{v}_1^T$ contributes more to the output row.



$w_{21} = \langle \mathbf{q}_2, \mathbf{k}_1 \rangle, w_{22} = \langle \mathbf{q}_2, \mathbf{k}_2 \rangle$, the second output row is

$$\text{Softmax}\left(\frac{w_{21}}{\sqrt{d}}\right)\mathbf{v}_1^T + \text{Softmax}\left(\frac{w_{22}}{\sqrt{d}}\right)\mathbf{v}_2^T$$

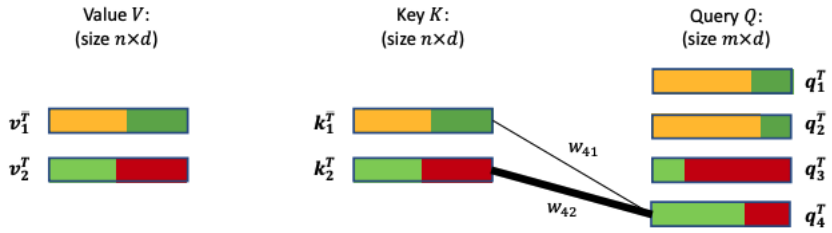
where $\mathbf{v}_1^T$ contributes more to the output row.



$w_{31} = \langle \mathbf{q}_3, \mathbf{k}_1 \rangle$, $w_{32} = \langle \mathbf{q}_3, \mathbf{k}_2 \rangle$, the third output row is

$$\text{Softmax}\left(\frac{w_{31}}{\sqrt{d}}\right)\mathbf{v}_1^T + \text{Softmax}\left(\frac{w_{32}}{\sqrt{d}}\right)\mathbf{v}_2^T$$

where $\mathbf{v}_2^T$ contributes more to the output row.



$w_{41} = \langle \mathbf{q}_4, \mathbf{k}_1 \rangle$, $w_{42} = \langle \mathbf{q}_4, \mathbf{k}_2 \rangle$, the fourth output row is

$$\text{Softmax}\left(\frac{w_{41}}{\sqrt{d}}\right)\mathbf{v}_1^T + \text{Softmax}\left(\frac{w_{42}}{\sqrt{d}}\right)\mathbf{v}_2^T$$

where $\mathbf{v}_2^T$ contributes more to the output row.

11.3.1 Matrix Form of Attention

Let $\mathbf{v}_i^T, \mathbf{k}_i^T, \mathbf{q}_i^T$ stand for the row vectors of value, key, and query. Let

$$V = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}, K = \begin{bmatrix} \mathbf{k}_1^T \\ \vdots \\ \mathbf{k}_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}, Q = \begin{bmatrix} \mathbf{q}_1^T \\ \vdots \\ \mathbf{q}_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}$$

Then using Softmax (row-wise),

$$\begin{aligned} \text{Attention}(V, K, Q) &= \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \\ &= \text{Softmax}\left(\frac{\begin{bmatrix} \mathbf{q}_1^T \\ \vdots \\ \mathbf{q}_m^T \end{bmatrix}\begin{bmatrix} \mathbf{k}_1 & \cdots & \mathbf{k}_n \end{bmatrix}}{\sqrt{d}}\right)\begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} \\ &= \begin{bmatrix} \text{Softmax}\left(\frac{\langle \mathbf{q}_1, \mathbf{k}_1 \rangle}{\sqrt{d}}\right)\mathbf{v}_1^T + \cdots + \text{Softmax}\left(\frac{\langle \mathbf{q}_1, \mathbf{k}_n \rangle}{\sqrt{d}}\right)\mathbf{v}_n^T \\ \vdots \\ \text{Softmax}\left(\frac{\langle \mathbf{q}_m, \mathbf{k}_1 \rangle}{\sqrt{d}}\right)\mathbf{v}_1^T + \cdots + \text{Softmax}\left(\frac{\langle \mathbf{q}_m, \mathbf{k}_n \rangle}{\sqrt{d}}\right)\mathbf{v}_n^T \end{bmatrix} \in \mathbb{R}^{m \times d} \end{aligned}$$

The inner product $\langle \mathbf{q}_i, \mathbf{k}_j \rangle$ measures *similarity*; the more similar, the more contributions.

Each output is a convex combination of value rows.

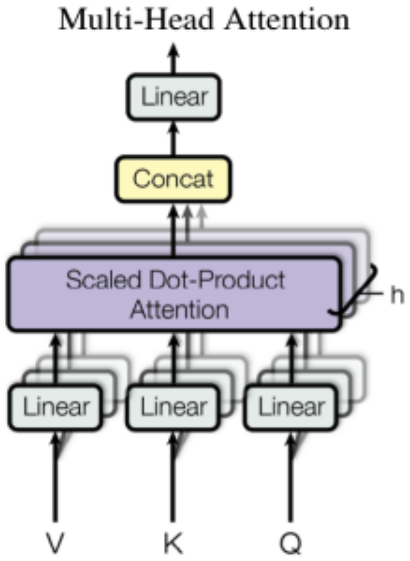
Definition: Self-Attention

$$Q = K = V$$

There is **NO** learnable parameters in this layer.

11.4 Learnable Attention Layer and Multihead Attention

$$\text{Attention}(VW^v, KW^k, QW^q) = \text{Softmax}\left(\frac{QW^q(KW^k)^T}{\sqrt{d}}\right)VW^v$$



We replace Q, K, V with $QW^q \in \mathbb{R}^{m \times d_k}, KW^k \in \mathbb{R}^{m \times d_k}, VW^v \in \mathbb{R}^{m \times d_v}$ respectively. The *learnable* linear layers are $\{W^q, W^k, W^v\} \in \mathbb{R}^{512 \times 64}$.

We can add $h = 8$ linear layers W_i^q's, W_i^k's, and W_i^v's in parallel and concatenate their output later. The output dimension is $64 \times 8 = 512$.

11.4.1 Masked Multihead Attention

Do not look at future words, instead mask them. Input the masked sequence into the attention layer.

11.5 Feed-Forward Layer

Definition: Feed-Forward Network

A two-layer MLP with ReLU activation:

$$\mathrm{MLP}(\mathbf{x}) = \max(0, \mathbf{x}^T W_1 + \mathbf{b}_1^T) \cdot W_2 + \mathbf{b}_2^T$$

11.6 Layer Normalization

Batch size (number of sequences) is often small, usually 1 in natural language processing (NLP) tasks. Therefore, batch normalization might not be a good choice. We often use layer normalization instead.

11.7 Overview

There are only 3 tunable hyperparameters. The number of layers $N = 6$, output dimension of all modules is $d = 512$, and the number of attention heads $h = 8$.

The module that connects encoder and decoder is a multihead attention, where the value and key are from encoder, and query from the decoder ($V = K \neq Q$). In the other two attention modules $V = K = Q$.

11.8 Transformer Loss

Pretraining task: predict next words. We train by minimizing the log-loss between true next word and predicted next word:

$$\min_W \hat{\mathbb{E}} \left[- \left\langle Y, \log \hat{Y} \right\rangle \right]$$

where $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_l]$ is the one-hot encoded output sequence and $\hat{Y} = [\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_l]$ is the predicted probabilities.

Part III

Modern Machine Learning Paradigms

Chapter 12

Large Language Models

12.1 Transformer Review

$$\text{Attention}(V, K, Q) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

Each output is a convex combination of value rows. Self-attention means $Q = K = V$.

The feed-forward network $MLP(\mathbf{x}) = \max(0, \mathbf{x}^T W_1 + \mathbf{b}_1^T)W_2 + \mathbf{b}_2^T$ is a two-layer MLP with ReLU activation with $W_1 \in \mathbb{R}^{d \times d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d}$ where d_{ff} is usually set as $4d$.

Attention complexity: $O(n^2 d + nd^2)$.
Feed-forward complexity: $O(d^3)$.
where n is the sequence length and d is the dimension of internal representation. Attention also costs $O(nd^2)$ due to linear transformation $V = VW$ where $W \in \mathbb{R}^{d \times d/h}$ where h is the number of heads.

12.2 Labeling Smoothing

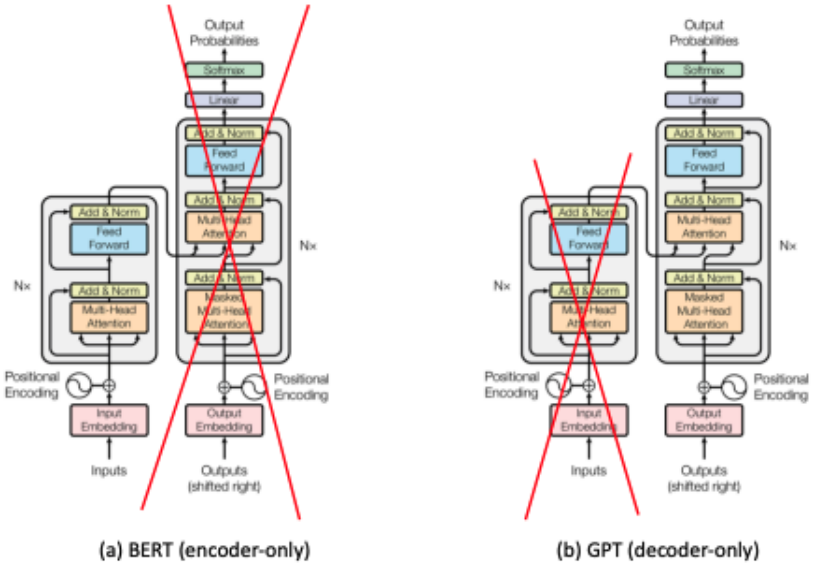
We train large language model (LLM) parameters Θ by minimizing the log-loss between true masked word and predicted masked word:

$$\min_{\Theta} \hat{\mathbb{E}}[-\left\langle Y, \log \hat{Y} \right\rangle]$$

For a training example with ground-truth label y, replace the label Y distribution $p(k|x) = \delta_{k,y}$ with $p'(k|x) = (1 - \epsilon_{ls})\delta_{k,y} + \epsilon_{ls}\frac{1}{C}$ where C is the number of classes.

12.3 BERT vs. GPT

BERT is an encoder, GPT is a decoder. Therefore, BERT is not a generative model.



BERT predicts a randomly-sampled middle word whereas GPT predicts the next word.

BERT is not auto-regressive and GPT is auto-regressive.

12.4 Pretraining–Fine-tuning–Inference Framework

The pretraining stage contains training on large unlabelled datasets such as Wikipedia, BookCorpus, etc. and is self-supervised training (hours to days). This will learn the pretrained weights.

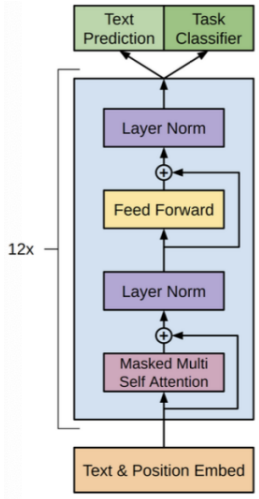
Fine-tuning trains on smaller labelled datasets such as SQuAD, MNLI/CMNLI, Similarity, etc. and is task-specific fine-tuning (minutes to hours). This will fine-tune the weights.

In the pretraining, the model predicts masked words: next words for GPT (harder) or middle words given context for BERT (easier).

It is harder to predict future words, therefore, GPT is better towards artificial general intelligence (AGI).

12.5 Generative Pretraining Transformer (GPT)

The GPT architecture:



12.5.1 Pretraining

Uses unsupervised pretraining through a language model:

$$\min_{\Theta} \hat{\mathbb{E}} \left[\underbrace{-\log \prod_{j=1}^{m} p(\mathbf{x}_j | \mathbf{x}_1, \ldots, \mathbf{x}_{j-1}; \Theta)}_{\text{log-likelihood of predicting next word}} \right]$$

Given the context consisting of previous tokens $\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}$ (prompt + previous outputted tokens), predict the current token $\mathbf{x}_j$.

The pretraining step only uses the Text Prediction output.

12.5.2 Fine-tuning

Uses supervised fine-tuning with task-aware transformations:

$$\min_{\Theta} \underbrace{-\hat{\mathbb{E}} \left[\log \prod_{j=1}^{m} p(\mathbf{y} | X_{1:m}; \Theta) \right]}_{\text{task-aware supervised loss}} \cdot \underbrace{-\lambda \hat{\mathbb{E}} \left[\log \prod_{j=1}^{m} p(\mathbf{x}_j | X_{1:j-1}; \Theta) \right]}_{\text{pretraining loss}}$$

Fine-tuning uses also uses Task Classifier, the classes are discussed are in the next section.

12.5.3 Task-Dependent Architecture

Definition: Classification

Classify a given text into a class.

Definition: Entailment

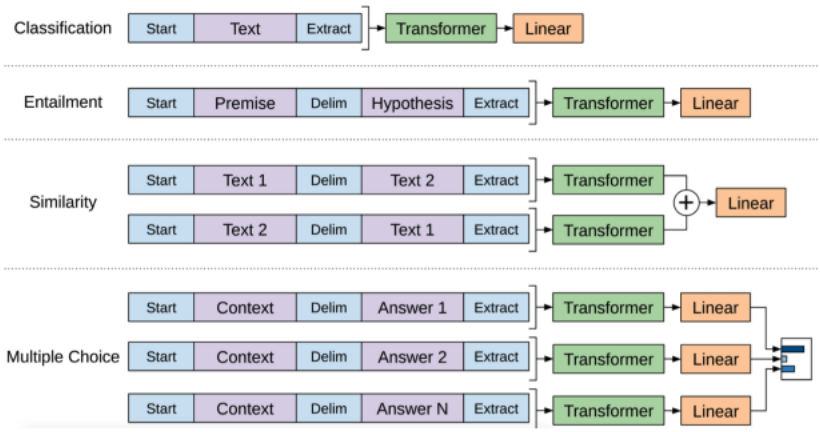
Determine whether a piece of text (hypothesis) contradicts or logically follows from another piece of text (premise).

Definition: Similarity

Predict whether two sentences are semantically equivalent.

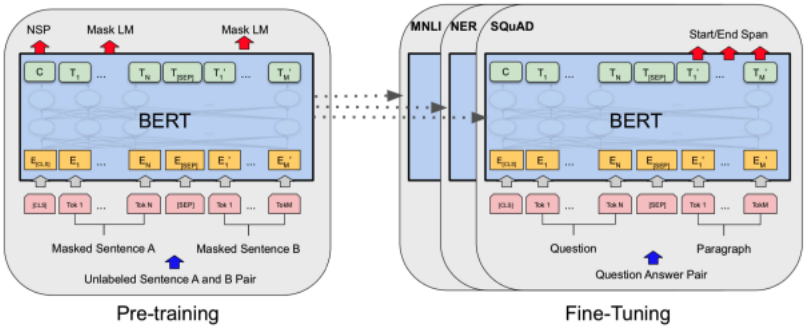
Definition: Multiple Choice

Given a context and N possible answers, choose the correct answer.



Early versions of GPT required task-dependent architectures. More recent version do not require it anymore since models became larger.

12.6 Bidirectional Encoder Representations from Transformers (BERT)



$\text{BERT}_{\text{BASE}}$ has $N = 12, d = 768, h = 12$, with total parameters 110M.

$\text{BERT}_{\text{LARGE}}$ has $N = 24, d = 1024, h = 16$, with total parameters 340M.

12.6.1 Pretraining Task A

Masked Language Model (Masked LM) randomly selects 15% input tokens to change to [Mask]. It adds softmax to predict the [Mask] tokens. It actually is 12% replaced with [Mask], 1.5% with random.

12.6.2 Pretraining Task B

Next Sentence Prediction (NSP): Given two sentences A and B, 50% of the time B is the actual next sentence that follows A (labeled IsNext), and 50% of the time it is a random sentence from the corpus (labeled NotNext).

The losses for Masked LM and NSP tasks are weighted, summed, and minimized.

$\text{BERT}_{\text{BASE}}$ and GPT have comparable number of parameters, but $\text{BERT}_{\text{BASE}}$ has better GLUE score.

12.6.3 RoBERTa

Trained model longer, with bigger batches, over more data. It removed the NSP objective and trained on longer sequences.

BERT uses a cross-encoder: two sentences are passed to the transformer. Given N sentences, there are as many as $\binom{N}{2}$ sentence pairs. 10,000 sentences require about 50M times of inference (65 hours) with BERT.

The sentence-transformer is a twin network that reduces 50M to 10K times of inference and 65 hours to 5 seconds.

12.7 GPT-2

A new dataset of millions of webpages called WebText. It is a 1.5B-parameter transformer with the same training method as GPT. GPT-2 performs on par with BERT on fine-tuning tasks, even though BERT is 10x larger. GPT-2 is good for zero-shot learning.

Definition: Zero-Shot Learning

Model predicts answer given only a natural language description with no gradient updates.

Definition: k-Shot Learning

Model predicts answer given a task description and k examples of the task with no gradient updates.

12.8 GPT-3

A transformer with 175B-parameters. Trained the same as GPT and GPT-2. When the network is large enough, there is in-context learning and chain-of-thought prompting.

In-context learning is few-shot learning.

Chain-of-thought prompting is giving an example of the reasoning process in the prompt. Human labelling chain-of-thought is costly; there is a new research direction prompt: prompt tuning/engineering.

Simply adding "Let's think step by step" in the prompt without a reasoning process can improve performance.

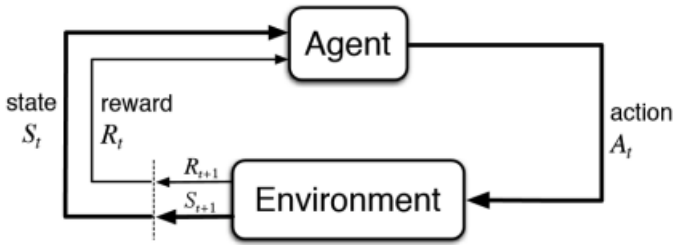
12.9 GPT-3.5

Definition: Policy Function

A function that maps state $\rightarrow$ action.

Uses human feedback to do reinforcement learning. An agent interacts with an environment by taking actions. The environment returns a reward for the action and a new state (representation of world at that moment). Agent uses a policy function to choose an action at a given state.

In the large language model, the state is the prompt, action is the output of the LLM, policy function is the LLM, and the reward is the levels of matching human feedback.



12.10 Reinforcement Learning from Human Feedback

This can be done in 3 steps:

1. Update LLM by back-propagation.
2. Freeze the LLM and learn the Reward Model (RM).
3. Freeze the RM and update the LLM by reinforcement learning.

12.10.1 Step 1

Collect demonstration data and train a supervised policy. Humans provide demonstrations of the desired behaviour on the input prompt distribution. Fine-tune a pretrained GPT-3 model on this data using supervised learning to get the SFT model.

12.10.2 Step 2

Collect comparison data and train a reward model. Collect a dataset of comparisons between model outputs where the labeller indicates which one among K outputs they prefer for a given input. Then train the reward model (RM) to predict human-preferred output.

Pairwise comparison loss to train reward model r_θ:

$$loss(\theta) = -\frac{1}{\binom{K}{2}}\mathbb{E}_{(x,y_w,y_l)}[\log(\sigma(r_\theta(x,y_w) - r_\theta(x,y_l)))]$$

where x is the prompt/input, y_w is the preferred output in (y_w, y_l), and σ is the sigmoid function. We encourage $r_\theta(x,y_w) >> r_\theta(x,y_l)$ by logistic loss.

We use the RM model because annotators are hard to give uniformly consistent scores for a given sentence. Since ranking is a discrete reward which suffers from discrete 0-1 reward. The RM model fits the discrete reward and outputs a real-valued reward. This is why we use the RM model instead of a human annotator.

OpenAI has created a human-computer interface for human annotators.

12.10.3 Step 3

Optimize a policy against the reward model using PPO. Use the output of the reward model as a scalar reward. Fine-tune the supervised policy to optimize this reward using a reinforcement learning algorithm.

$$\max_\phi \mathbb{E}_{(x,y)}[\underbrace{r_\theta(x,y)}_{\text{maximize reward}} - \beta\underbrace{\log(\pi_\phi^{RL}(y|x)/\pi^{SFT}(y|x))}_{\text{model is close to SFT model}}] + \gamma\underbrace{\mathbb{E}[\log(\pi_\phi^{RL}(x))]}_{\text{pretraining loss}}$$

ChatGPT and GitHub Copilot are based on GPT-3.5.

12.11 GPT-4

Released on March 14, 2023.

Multi-modality: allows images and text as input and text as output.

12.11.1 Parameter-Efficient Fine-Tuning (PEFT)

Large language models are becoming larger, so the industry has significant advantages over academia.

Definition: Low-Rank Adaptation (LoRA)

Pretrained weights are frozen, only train A and B.

Chapter 13

Generative Adversarial Networks

In generative-modelling, we would like to train a network that models a distribution. For LLM, we want to model the distribution of natural language. For images, we want a generative model to generate images.

Definition: Generative Model

Given *true data density* training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \sim p_{data}(\mathbf{x})$, we want to parameterize the data density $p_{\boldsymbol{\theta}}(\mathbf{x})$ estimated by the model by estimating $\boldsymbol{\theta}$ by minimizing the distance between p_{data} and $p_{\boldsymbol{\theta}}$:

$$\min_{\boldsymbol{\theta}} dist(p_{data} || p_{\boldsymbol{\theta}})$$

After training, we can generate new data $\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x})$, but we need a training set from p_{data} and an explicit form of $p_{\boldsymbol{\theta}}$.

Theorem (Representation Through Push-Forward)

Let r be any continuous distribution on $\mathbb{R}^h$. For any distribution p on $\mathbb{R}^d$, there exist push-forward maps $\mathbf{G} : \mathbb{R}^h \to \mathbb{R}^d$ such that

$$\mathbf{z} \sim r \implies \mathbf{G}(\mathbf{z}) \sim p$$

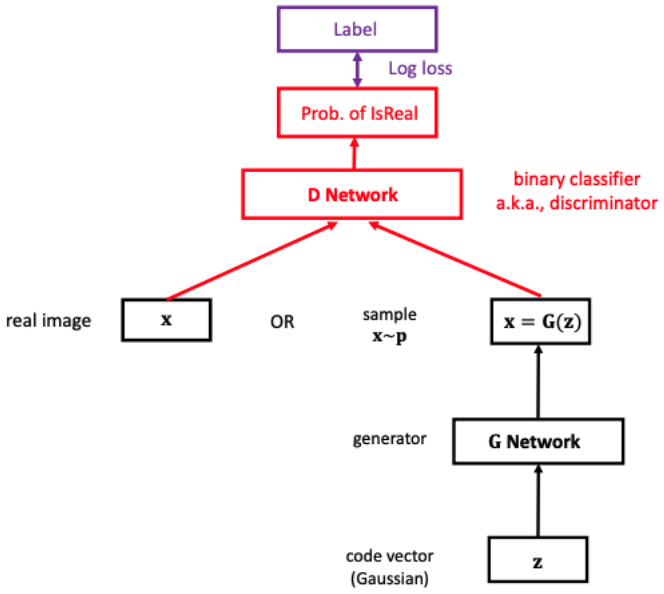
Without loss of generality, we may simply take r to be standard Gaussian noise.

13.1 Generating Samples

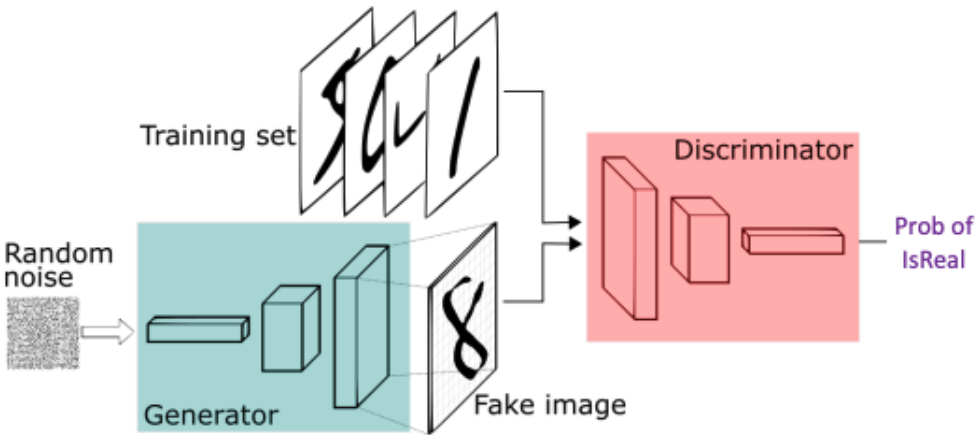
We start by sampling a code vector $\mathbf{z}$ from a simple distribution, such as Gaussian. The GAN computes a differentiable function $\mathbf{G}$, which maps $\mathbf{z}$ to an $\mathbf{x}$ in the data space.

$\mathbf{G}$ maps a sample from one distribution to a sample from another distribution. To define

the loss to distinguish two distributions, we can use a discriminator.



A GAN is a zero-sum game between the discriminator and generator. The discriminator's goal is to distinguish real images from fake images. The generator's goal is to generate images that look like real ones to confuse the discriminator.



13.1.1 Discriminator

For a fixed generator G, minimize the log loss over D (output probability of IsReal).

If $\mathbf{x}$ is real, minimize $-\log D(\mathbf{x})$ and if $\mathbf{x}$ is fake, minimize $-\log(1 - D(\mathbf{x}))$. Assume that $\mathbf{x}$ is from a real/fake distribution with equal chance:

$$\min_D -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}[\log(1 - D(G(\mathbf{z})))]$$

13.1.2 Generator

For a fixed discriminator D, maximize a log loss over G.

If $\mathbf{x}$ is real, maximize $-\log D(\mathbf{x})$ and if $\mathbf{x}$ is fake, maximize $-\log(1 - D(\mathbf{x}))$. Assume that $\mathbf{x}$ is from a real/fake distribution with equal chance:

$$\max_G -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}[\log(1 - D(G(\mathbf{z})))]$$

Therefore, we can learn G and D simultaneously by solving a minimax problem:

$$\max_G \min_D -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}[\log(1 - D(G(\mathbf{z})))]$$

We can replace expectation with empirical expectation since we do not have the true distribution:

$$\min_G \max_D \hat{\mathbb{E}}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \hat{\mathbb{E}}_{\mathbf{z}\sim\mathcal{N}(0,I)}[\log(1 - D(G(\mathbf{z})))]$$

To solve the problem

$$\min_G \max_D V(G, D) := \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}[\log(1 - D(G(\mathbf{z})))]$$

we alternate minimization-maximization. For G, fix D and update G by one-step gradient descent. For D, fix G and update D by one-step gradient ascent. Repeat until the algorithm reaches an approximate equilibrium.

The generator creates a larger image from a smaller image, however, the convolution operation can only make images smaller. So we need an "inverse" convolution.

Definition: Deconvolution (Transposed Convolution)

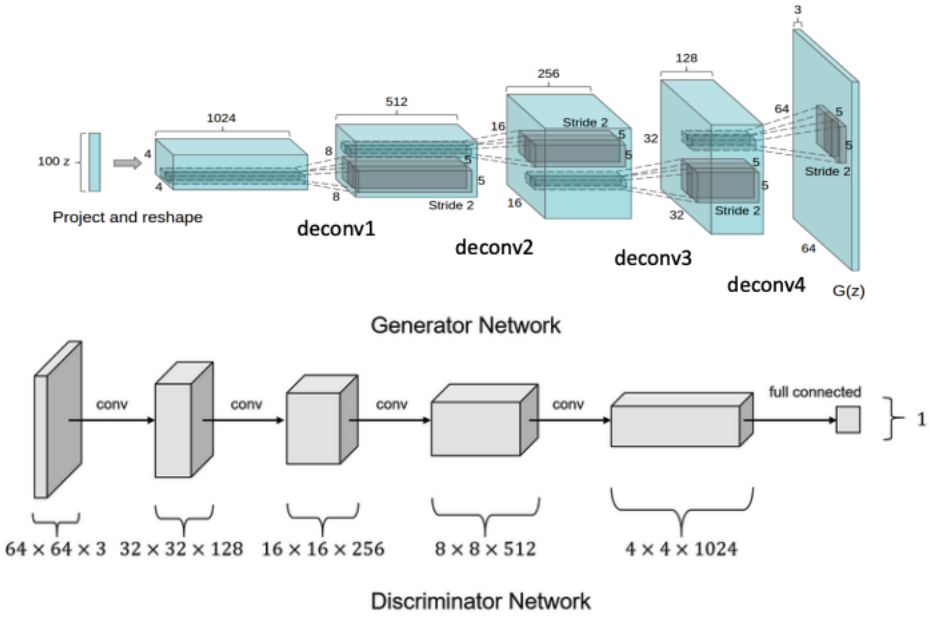
Hadamard product each entry in the image with every entry in the kernel. Then sum all entries where they overlap in the output image.

13.2 Analysis of GAN

Proposition (Solution of D^*)

Let $p_g(\mathbf{x})$ be the density of $\mathbf{x}$ estimated by the generator G. For G fixed, the optimal discriminator D is

$$D^*_G(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}$$



Proof.

$$\begin{aligned} V(G,D) &:= \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}[\log(1-D(G(\mathbf{z})))] \\ &= \int_{\mathbf{x}} p_{data}(\mathbf{x}) \log D(\mathbf{x}) d\mathbf{x} + \int_{\mathbf{z}} p_{\mathbf{z}}(\mathbf{z}) \log(1-D(G(\mathbf{z}))) d\mathbf{z} \\ &= \int_{\mathbf{x}} \underbrace{p_{data}(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1-D(\mathbf{x}))}_{f(D(\mathbf{x}))} d\mathbf{x} \end{aligned}$$

Note that if $f(s) = a\log s + b\log(1-s)$, then $\arg\max_s f(s) = \frac{a}{a+b}$, so $D^*_G(\mathbf{x}) := \arg\max_{D(\mathbf{x})} f(D(\mathbf{x})) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}$.

Theorem (Solution of G^*)

The global minimum of $\min_G \max_D V(G,D)$ is achieved if and only if $p_g = p_{data}$. The optimal objective value is $-\log 4$.

Proof.

$$\begin{aligned} V(G,D^*_G) &= \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D^*_G(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}[\log(1-D^*_G(G(\mathbf{z})))] \\ &= \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D^*_G(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim p_g}[\log(1-D^*_G(\mathbf{x}))] \\ &= \mathbb{E}_{\mathbf{x}\sim p_{data}}\left[\log \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}\right] + \mathbb{E}_{\mathbf{x}\sim p_g}\left[\log \frac{p_g(\mathbf{x})}{p_{data}(\mathbf{x})+p_g(\mathbf{x})}\right] \end{aligned}$$

Since $KL(P||Q) = \mathbb{E}_{\mathbf{x}\sim P}\left[\log \frac{p(\mathbf{x})}{q(\mathbf{x})}\right]$, so

$$\begin{aligned} V(G,D^*_G) &= -\log 4 + KL\left(p_{data}||\frac{p_{data}+p_g}{2}\right) + KL\left(p_g||\frac{p_{data}+p_g}{2}\right) \\ &= -\log 4 + 2JSD(p_{data}||p_g) \\ &\geq -\log 4 \end{aligned}$$

where JSD is the Jensen-Shannon divergence. The equality holds if and only if $p_{data} = p_g$.

Chapter 14

Self-Supervised Learning

Definition: Self-Supervised Learning

A subclass of unsupervised learning.
It learns useful feature representations from unlabelled data through pretraining tasks for downstream tasks.

Self-supervised learning steps:

1. Pretraining/Pretext step: build a task where the label is pseudo and constructed from unlabelled data.
2. Downstream step: Fine-tuning protocol and linear evaluation protocol.

Definition: Representation Learning

For the downstream task, fix the trained base model, and fine-tune the top layers on a small labelled dataset.

Benefits:

- Creating labelled datasets for each task.
- Vast amount of unlabelled data on the internet.
- Will not overfit.

Challenges:

- How to select suitable pretraining task for an application.
- No gold standard for comparison of learned feature representations.
- Selecting suitable loss functions, since there is no single objective as the test set accuracy in supervised learning.

14.1 Geometric Transformation Recognition

14.1.1 Image Rotation

Pretraining data: images rotated by a multiple of 90° at random.

Pretraining task: train a model to predict the rotation degree that was applied.

A single ConvNet model is used to predict one of four rotations. The model only needs to understand the location and type of the objects in images to determine rotation degree.

Evaluation on the PASCAL VOC dataset for classification, detection, and segmentation tasks.

14.2 Patches

14.2.1 Relative Patch Position

Pretraining data: multiple patches extracted from images.

Pretraining task: train a model to predict the relationship between the patches.

The patches are inputted into two ConvNets with shared weights. The model needs to understand the spatial context of images in order to predict the relative positions between the patches.

14.2.2 Image Jigsaw Puzzle

Pretraining data: 9 patches extracted in images.

Pretraining task: predict the positions of all 9 patches.

14.3 Generative Modelling

14.3.1 Context Encoders

Pretraining data: remove a random region in images.

Pretraining task: fill in a missing piece in the image.

Initially, the model uses an encoder-decoder architecture. Euclidean ℓ_2 distance is used as the reconstruction loss function L_{rec}.

In the downstream task, we use the encoder networks as the representation.

Improvement can be achieved by adding a GAN branch. Then use a weighted combination of the two losses $\lambda_{rec}L_{rec} + \lambda_{gan}L_{gan}$.

14.3.2 Image Colourization

Pretraining data: pairs of colours and grayscale images.

Pretraining task: predict the colours of the objects in grayscale images.

Uses an encoder-decoder architecture with convolutional layers. ℓ_2 loss between the coloured image and predicted colourized image.

In the downstream task, use the encoder as the representation.

14.3.3 Cross-Channel Prediction

Pretraining data: remove some of the image colour channels.

Pretraining task: predict the missing channel from the other image channels.

Downstream task: use the encoder in either $\mathcal{F}_1$ or $\mathcal{F}_2$ networks as representation.

14.3.4 Image Super-Resolution

Pretraining data: pairs of regular and downsampled low-resolution images.

Pretraining task: predict a high-resolution image that corresponds to a downsampled low-resolution image.

Uses GAN architecture. The paper did not consider downstream tasks other than super-resolution.

14.4 Contrastive Learning

14.4.1 SimCLR

We measure agreement by comparison. Distance of images of same content $\leq$ distance of images of different contents. Similarity of images of same content $\geq$ similarity of images of different contents.

This is the first method that is comparable with supervised learning on ImageNet by linear evaluation protocol.

Figure 14.1: Structure of SimCLR

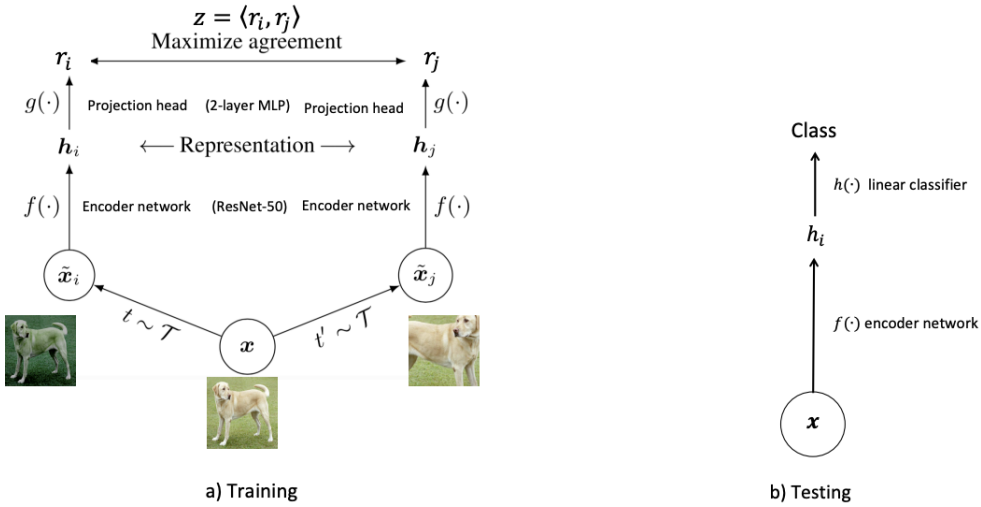
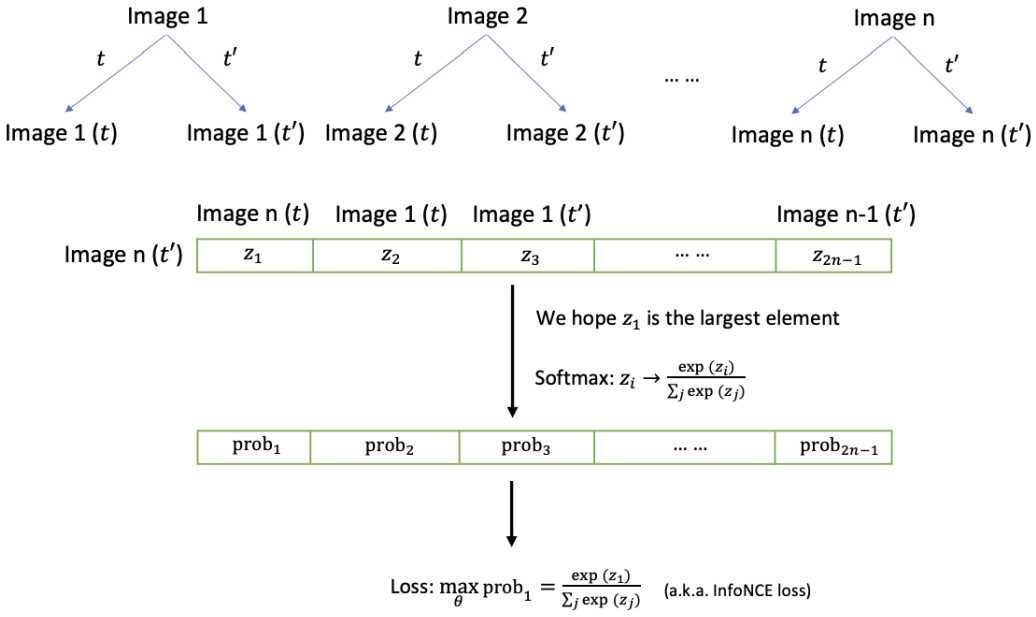


Figure 14.2: Loss function with Image n (t') as the anchor.



Part IV

Trustworthy Machine Learning

Chapter 15

Evasion Attacks

Definition: Evasion Attacks

Modify test images to fool the ML model when the ML model is fixed.

Definition: White-Box Attack

The attacker knows full information of the network.

Definition: Black-Box Attack

The attacker does not know anything about the network.

Definition: Untargeted Attack

The goal is to predict a wrong label that is not a specific label.

Definition: Targeted Attack

The goal is to predict some targeted label.

15.1 Principle of Generating Evasion Attacks

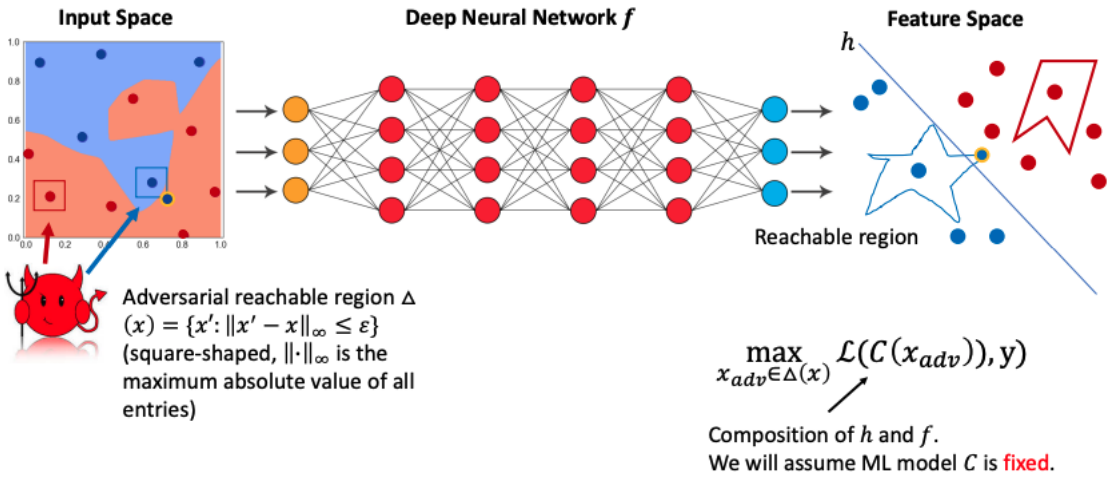
Definition: Adversarial Reachable Region

$$\Delta(x) := \{x' : \|x' - x\|_\infty \leq \varepsilon\}$$

Let the separator be h and neural network f, then we have

$$\max_{x_{adv} \in \Delta(x)} \mathcal{L}(h(f(x_{adv})), y)$$

Let C be the composition of h and f and we will assume the ML model C is fixed.



Generating adversarial examples reduces to the problem of solving

$$\max_{\|x_{adv} - x\|_\infty \leq \varepsilon} \mathcal{L}(C(x_{adv}), y)$$

We have different tools in optimization:

- Zero-order solvers (access to only the output of NN): black-box attack.
- First-order solvers (access to gradient information, e.g. FGSM, BIM, PGD, CW attacks, etc.): white-box attack; calculating gradient requires full information about NN.
- Second-order solvers (access to Hessian matrix, e.g. L-BFGS attack): white-box attack.

15.2 Evasion Attacks

15.2.1 Fast Gradient Sign Method (FGSM) Attack

The above is non-convex and hard to solve. The linear expansion at x is

$$\mathcal{L}(C(x_{adv}, w), y) \approx \underbrace{\mathcal{L}(C(x, w), y)}_{\text{constant}} + \langle x_{adv} - x, \nabla_x \mathcal{L}(C(x, w), y) \rangle$$

which can be reduced to

$$\max_{\|x_{adv} - x\|_\infty \leq \varepsilon} \langle x_{adv} - x, \nabla_x \mathcal{L}(C(x, w), y) \rangle$$

We have the closed-form solution $x^*_{adv} = x + \varepsilon \cdot sign(\nabla_x \mathcal{L}(C(x, w), y))$.

Definition: Fast Gradient Sign Method (FGSM)

$$x^*_{adv} = x + \varepsilon \cdot sign(\nabla_x \mathcal{L}(C(x, w), y))$$

Theorem (Hölder's Inequality)

For any vectors a, b, we have $\langle a, b\rangle \leq \|a\|_p \|b\|_q$, where $\frac{1}{p} + \frac{1}{q} = 1$ and $p, q \geq 1$.

$\|\cdot\|_p, \|\cdot\|_q$ are known as dual norms.

Proof.

$$\begin{aligned} Obj(x_{adv}) &= \langle x_{adv} - x, \nabla_x \mathcal{L}(C(x, w), y)\rangle \\ &\leq \|x_{adv} - x\|_\infty \|\nabla_x \mathcal{L}(C(x, w), y)\|_1 \\ &\leq \varepsilon \|\nabla_x \mathcal{L}(C(x, w), y)\|_1 \end{aligned}$$

The above solution achieves the upper bound and satisfies the constraint. □

FGSM is a **white-box, non-targeted** evasion attack. It is an evasion attack since it is functioning on the test image. It is a white-box attack since we need to calculate $\nabla_x \mathcal{L}(C(x, w), y)$ to create the adversarial image. FGSM calculates gradient *only once*. It is non-targeted since the attacker aims to maximize the loss with respect to the true label.

The reason for the sign operator in the solution is to remove the imbalance in the update when the gradient on one pixel is much larger. The method automatically reaches the boundary of adversarial reachable region for all pixels $\Delta(x) = \{x' : \|x' - x\|_\infty \leq \varepsilon\}$. It also gives better empirical attack success rate in experiments.

Sometimes, FGSM requires large ε in order to succeed.

15.2.2 Basic Iterative Method (BIM) Attack

BIM is a variant of FGSM; it repeatedly adds noise to the image x in multiple iterations in order to cause misclassification.

Definition: Basic Iterative Method (BIM)

Let t be the index of iterations and γ be the step size.

$$x^t = x^{t-1} + \gamma \cdot sign(\nabla_x \mathcal{L}(C(x^{t-1}, w), y))$$

The step size is different compared to FGSM and it uses an iterative procedure while FGSM uses a one-shot procedure.

15.2.3 Projected Gradient Descent (PGD) Attack

To resolve the issue of BIM, PGD involves a truncation operation.

Definition: Projected Gradient Descent (PGD)

$$x^t = clip_{(-\varepsilon,\varepsilon)}(x^{t-1} + \gamma \cdot sign(\nabla_x \mathcal{L}(C(x^{t-1}, w), y)))$$

For pixels with perturbation size larger than ε, clip truncates it to ε.

PGD also uses random initialization for x^0, by adding random noise to the original image from a uniform distribution in the range $(-\varepsilon, \varepsilon)$.

PGD is a **white-box, non-targeted** evasion attack. It is a white-box attack since we need to know the gradients $\nabla_x \mathcal{L}(C(x, w), y)$ of the model to create the adversarial image. PGD calculates the gradient *multiple times*. It is non-targeted since PGD aims to maximize the loss with respect to the true label.

Targeted PGD Attack

Gradient approaches (FGSM, BIM, PGD) can also be designed as targeted white-box attacks. In this case, the added perturbation noise aims to minimize the loss function of the image for a specific target class.

Untargeted objective (gradient ascent):

$$\max_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}), y_{true})$$

Targeted objective (gradient descent):

$$\min_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}), y_{target})$$

Untargeted iteration (based on maximizing loss function for true class):

$$x^t_{adv} = clip_{(-\varepsilon,\varepsilon)}(x^{t-1} + \gamma \cdot sign(\nabla_x \mathcal{L}(C(x^{t-1}, w), y_{true})$$

Targeted iteration (based on minimizing loss function for target class):

$$x^t_{adv} = clip_{(-\varepsilon,\varepsilon)}(x^{t-1} - \gamma \cdot sign(\nabla_x \mathcal{L}(F(x^{t-1}, w), y_{target})$$

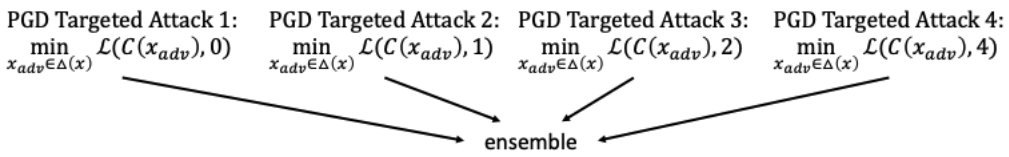
15.2.4 Multi-Targeted PGD Attack

This is attack is an untargeted attack. The idea is to do targeted attacks for all target classes and choose the one that can fool the classifier.

Ensemble: as long as one of the attacks is successful, we deem the multi-targeted attack successful.

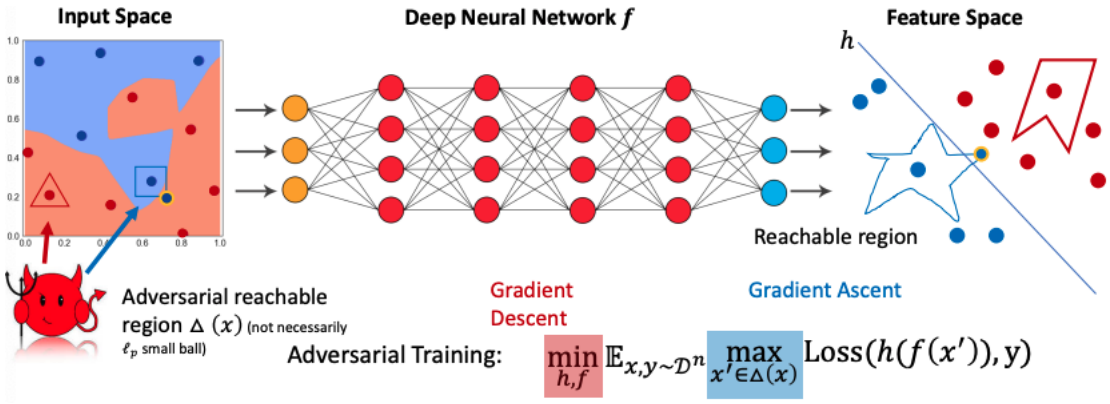
Both PGD and multi-targeted attacks are norm bounded: $\|x_{adv} - x\|_\infty \leq \varepsilon$. In practice, for MNIST choose $\varepsilon = 0.3$ and on CIFAR-10 choose $\varepsilon = 8/255 \approx 0.031$ to make adversarial image human-imperceptible.

Multi-targeted attack is slightly better than PGD.


PGD Targeted Attack 1:
$\min_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}), 0)$
PGD Targeted Attack 2:
$\min_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}), 1)$
PGD Targeted Attack 3:
$\min_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}), 2)$
PGD Targeted Attack 4:
$\min_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}), 4)$
ensemble

Chapter 16

Robustness



16.1 Adversarial Training

Definition: Adversarial Training

Trains the target classification model using adversarial examples

$$\min_{C} \mathbb{E}_{x,y\sim\mathcal{D}^n} \max_{x'\in\Delta(x)} Loss(C(x'), y)$$

Use the inner maximization to mimic behaviour of attacks. Use the outer minimization to update the weight of neural networks.

The adversarial examples are produced to attack the latest iterate of the classifer. By adding adversarial examples x' with true label y to the training set, the model will learn that x' belongs to the class y. Adversarial training is the most successful defense against evasion attacks.

16.1.1 FGSM Attack

Use FGSM attack to solve the inner maximization. Adversarial examples created by FGSM were added to train. The limitation is that the robust model is vulnerable to adversarial examples created by other attacks.

16.1.2 Ensemble Adversarial Training

Use a set of adversarial examples created by several fixed classifiers to train the model. The performance highly depends on the robustness of each model.

16.1.3 PGD Attack

Use PGD attack to solve the inner maximization. PGD can find stronger adversarial examples around an input sample x.

Demonstrated good robustness on MNIST and CIFAR-10, but due to high computational cost for creating PGD samples, it is difficult to scale to large datasets.

16.1.4 Trade-Off

Definition: Robustness-Accuracy Trade-Off

Adversarial training suffers from reduced accuracy on clean samples.

If the y-axis represents the difference between the classification error of an adversarially trained model and a naturally trained model, adversarial training reduces the natural accuracy for about $3-7\%$. Increasing the size of the dataset reduces the gap, while increasing perturbation size increases the gap.

16.1.5 Trade-Off Between Robustness and Accuracy

The robust error

$$R_{rob}(f) := \mathbb{E}_{x,y\sim\mathcal{D}} 1\{\exists x' \in \Delta(x) \text{ s.t. } f(x')y \leq 0\}, y \in \{\pm 1\}, \text{classifier } f : \mathcal{X} \to \mathbb{R}$$

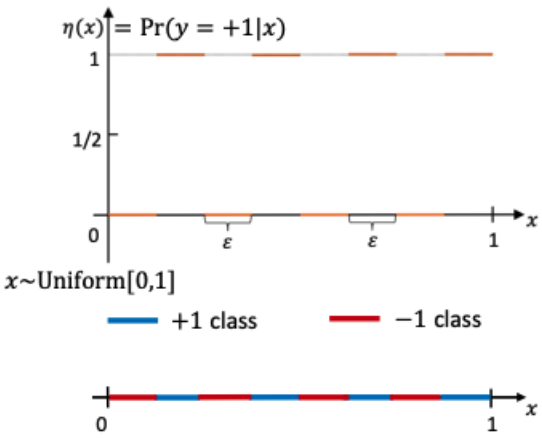
where 1 is the indicator function. The natural error

$$R_{nat}(f) := \mathbb{E}_{x,y\sim\mathcal{D}} 1\{f(x)y \leq 0\}$$

An example of trade-off (for norm-bounded threat model when $\Delta(x) = \mathbb{B}_p(x, \varepsilon)$):

Solution:

$$\min_f R_{nat}(f) + R_{rob}(f)/\lambda$$



	Bayes Optimal Classifier
$R_{nat}(f)$	0 (minimal R_{nat})
$R_{rob}(f)$	1

and instead of Bayes optimal classifier, use all +1 classifier where Computationally, weighted

All +1 Classifier
1/2
1/2 (minimal R_{rob})

average $R_{nat}(f) + R_{rob}(f)/\lambda$ is non-differentiable.

16.1.6 Classification-Calibrated Surrogate Loss

Can we design a differentiable surrogate loss for the trade-off between R_{rob} and R_{nat}.

Theorem (Bartlett et al.'06)

$$R_\phi(f) := \mathbb{E}_{x,y\sim\mathcal{D}}\phi(f(x)y)$$

Trades:

$$\min_f[\mathbb{E}_{x,y\sim\mathcal{D}}\phi(f(x)y) + \mathbb{E}_{x,y\sim\mathcal{D}}\max_{x'\in\Delta(x)}\phi(f(x)f(x')/\lambda)]$$

where the first expectation is to minimize difference between $f(x)$ and y for accuracy and the second expectation is to minimize difference between $f(x)$ and $f(x')$ for robustness. We use gradient ascent for the maximization.

Let TRADES Loss(f) be the objective function above.

Theorem (Informal Upper Bound Zhang el al.'19)

For any distribution $\mathcal{D}, f, \Delta(x), \lambda > 0$, we have

$$R_{rob}(f) - R^*_{nat} \leq \text{TRADES Loss}(f) - R^*_\phi$$

where

- R^*_{nat} is the minimal value of $R_{nat}(f)$ over all classifiers f.
- R^*_ϕ is the minimal value of $R_\phi(f) := \mathbb{E}_{x,y\sim\mathcal{D}}\phi(f(x)y)$ over all classifiers f.
- ϕ is the classification-calibrated surrogate loss.

Theorem (Informal Lower Bound Zhang et al.'19)

For any $\Delta(x)$, there exist a data distribution $\mathcal{D}$, a classifier f, and a $\lambda > 0$ such that

$$R_{rob}(f) - R^*_{nat} \geq \text{TRADES Loss}(f) - R^*_\phi$$

16.2 Limitations of Adversarial Training

Theorem (Training Dynamics of Adversarial Training)

Consider AT to learn a linear model $f(x) = w^T x$. The training dynamics of AT may lead to a cycle.

Proof. AT strategy:

1. Defender round
$$w^{(t-1)} = \arg\min_w Loss(w, x + \delta^{(t-1)})$$
2. Attacker round
$$\delta^{(t)} = \arg\max_\delta Loss(w^{(t-1)}, x + \delta)$$

During the defender round, we want to minimize the loss between the two classes. During the attacker round, we want to maximize this loss, which will move the data points orthogonal towards each other, which will switch sides.

The sign of w flips back and forth.

Chapter 17

Privacy

Chapter 18

Ethics

Chapter 19

Other Threats