Some thoughts on Mathematics and Computer Science

Chih-Jen Lin

Outline

Introduction

- When Prof. Chao asked me to give a lecture here, I wasn't quite sure what I should talk about
- I don't have new and emerging topic to share with you here.
- So instead I plan to talk about a topic "Mathematics and Computer Science"
- But why this topic?
- I will explain my motivation

Introduction (Cont'd)

- Sometime ago, in a town-hall meeting with some faculty members, one student asked why calculus is a required course
- I heard this from some faculty members as I wasn't there
- Anyway I think it really happened
- Here is the reaction from a professor:

  He said "When we were students, we didn't ask why xxx is a required course. We just took it."

Introduction (Cont'd)

- Then I asked myself if it's possible to give you some reasons
- That leads to this lecture

The Role of Mathematics in CS I

- One reason why some students' don't think calculus is important is that they think

$$\text{CS} = \text{programming}$$

- But many (or most) CS areas are beyond programming
- One issue is that in our required courses, things like calculus are seldom used
- Students can see that discrete mathematics are related to algorithms

The Role of Mathematics in CS II

- But they find calculus/linear algebra/statistics useful only after taking computer vision, signal processing, machine learning and others
- These are more advanced courses
- Note that CS is a rapidly changing area
- Before Internet, many CS companies just hired programmers
- For example, for Windows and Offices developments, Microsoft hired many programmers with an undergraduate degree

The Role of Mathematics in CS III

- Then Google started hiring many with Ph.D. or master degrees
- In compared with traditional software development, in the Internet era, analytics skills are more important
- This doesn't mean every engineer in big Internet companies has the job of developing analytics tools (e.g., deep learning software)
- Instead, most are users. They don't need to know all sophisticated details, but some basic understanding is essential

The Role of Mathematics in CS IV

- For example, as a user of deep learning, you probably need to roughly know how it works
- Otherwise you might now know what you are doing and what kinds of results you will get
- To have a basic understanding of these things, you need some mathematics background
- I am going to illustrate this point in the lecture

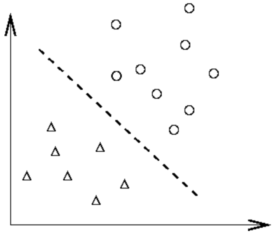
Outline

Neural Networks

- To discuss why mathematics is important in some CS areas, we can consider many examples
- We decide to talk about neural networks as deep learning is (incredibly) hot
- There are many types of neural networks, but we will consider the simplest one
- It's the fully connected network for multi-class classification
- So let's check what data classification is

Data Classification

We extract information to build a model for future prediction



Data Classification (Cont'd)

- The main task is on finding a model
- It's also called supervised learning

Data Classification (Cont'd)

- Given training data in different classes (labels known)

  Predict test data (labels unknown)
- Classic example: medical diagnosis

  Find a patient's blood pressure, weight, etc.

  After several years, know if he/she recovers

  Build a machine learning model

  New patient: find blood pressure, weight, etc

  Prediction
- Training and testing

Minimizing Training Errors

- Basically a classification method starts with minimizing the training errors

$$\min_{\text{model}} \quad \text{(training errors)}$$

- That is, all or most training data with labels should be correctly classified by our model
- A model can be a decision tree, a support vector machine, a neural network, or others
- There are various ways to introduce classification methods. Here we consider probably the most popular one

Minimizing Training Errors (Cont'd)

- For simplicity, let's consider the model to be a vector $\mathbf{w}$
- That is, the decision function is

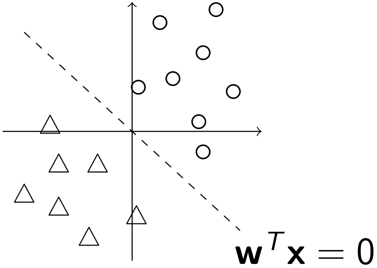
$$\operatorname{sgn}(\mathbf{w}^T\mathbf{x})$$

- For any data, $\mathbf{x}$, the predicted label is

$$\begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- The two-dimensional situation



- This seems to be quite restricted, but practically $\mathbf{x}$ is in a much higher dimensional space

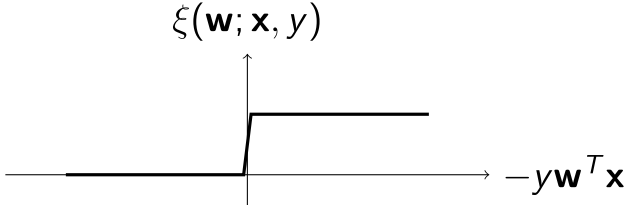
Minimizing Training Errors (Cont'd)

- To characterize the training error, we need a loss function $\xi(\mathbf{w};\mathbf{x},y)$ for each instance $(\mathbf{x},y)$
- Ideally we should use 0–1 training loss:

$$\xi(\mathbf{w};\mathbf{x},y) = \begin{cases} 1 & \text{if } y\mathbf{w}^T\mathbf{x} < 0, \\ 0 & \text{otherwise} \end{cases}$$

Minimizing Training Errors (Cont'd)

- However, this function is discontinuous. The optimization problem becomes difficult



- We need continuous approximations

Common Loss Functions

- Hinge loss (l1 loss)

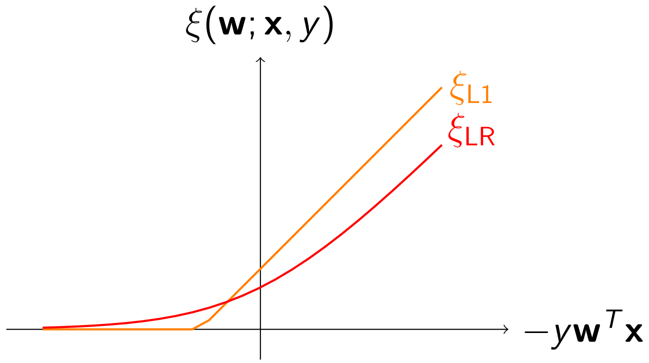
$$\xi_{\mathrm{L1}}(\mathbf{w}; \mathbf{x}, y) \equiv \max(0, 1 - y\mathbf{w}^T\mathbf{x}) \qquad (1)$$

- Logistic loss

$$\xi_{\mathrm{LR}}(\mathbf{w}; \mathbf{x}, y) \equiv \log(1 + e^{-y\mathbf{w}^T\mathbf{x}}) \qquad (2)$$

- Support vector machines (SVM): Eq. (1). Logistic regression (LR): (2)
- SVM and LR are two very fundamental classification methods

Common Loss Functions (Cont'd)



- Logistic regression is very related to SVM
- Their performance is usually similar

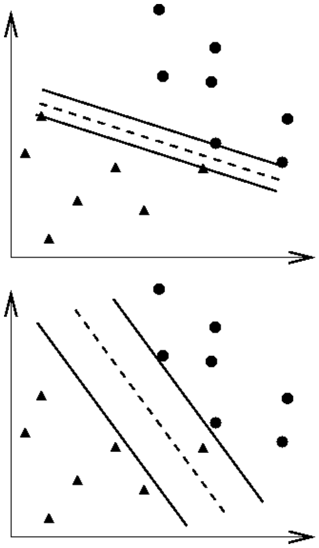
Common Loss Functions (Cont'd)

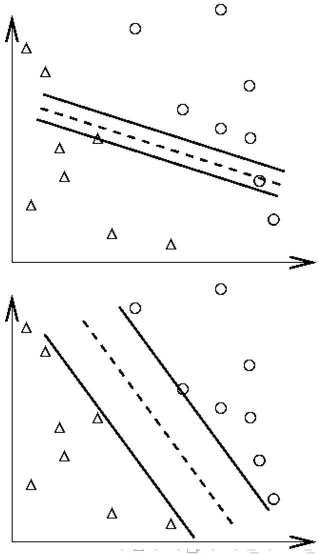
- However, minimizing training losses may not give a good model for future prediction
- Overfitting occurs

Overfitting

- See the illustration in the next slide
- For classification,
  You can easily achieve 100% training accuracy
- This is useless
- When training a data set, we should
  Avoid underfitting: small training error
  Avoid overfitting: small testing error

● and ▲: training; ○ and △: testing





Regularization

- To minimize the training error we manipulate the $\mathbf{w}$ vector so that it fits the data
- To avoid overfitting we need a way to make $\mathbf{w}$'s values less extreme.
- One idea is to make $\mathbf{w}$ values closer to zero
- We can add, for example,

$$\frac{\mathbf{w}^T\mathbf{w}}{2} \quad \text{or} \quad \|\mathbf{w}\|_1$$

to the function that is minimized

General Form of Linear Classification

- Training data $\{y_i, \mathbf{x}_i\}, \mathbf{x}_i \in R^n, i = 1, \ldots, l, \; y_i = \pm 1$
- l: # of data, n: # of features

$$\min_{\mathbf{w}} f(\mathbf{w}), \quad f(\mathbf{w}) \equiv \frac{\mathbf{w}^T\mathbf{w}}{2} + C \sum_{i=1}^{l} \xi(\mathbf{w}; \mathbf{x}_i, y_i)$$

- $\mathbf{w}^T\mathbf{w}/2$: regularization term
- $\xi(\mathbf{w}; \mathbf{x}, y)$: loss function
- C: regularization parameter (chosen by users)

From Linear to Nonlinear

- We now have linear classification because the decision function

$$\mathrm{sgn}(\mathbf{w}^T\mathbf{x})$$

  is linear
- We will see that neural networks (NN) is a nonlinear classifier

Neural Networks

- We will explain neural networks using the the same framework for linear classification
- Among various types of networks, we consider fully-connected feed-forward networks for multi-class classification

Neural Networks (Cont'd)

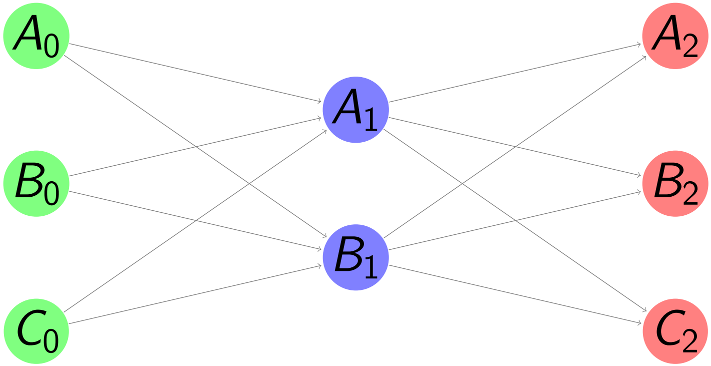
- Our training set includes $(\mathbf{y}_i, \mathbf{x}_i)$, $i = 1, \ldots, l$.
- $\mathbf{x}_i \in R^{n_0}$ is the feature vector.
- $\mathbf{y}_i \in R^K$ is the label vector.
- K: $\#$ of classes
- If $\mathbf{x}_i$ is in class k, then

$$\mathbf{y}_i = [\underbrace{0, \ldots, 0}_{k-1}, 1, 0, \ldots, 0]^T \in R^K$$

- A neural network maps each feature vector to one of the class labels by the connection of nodes.

Neural Networks (Cont'd)

- Between two layers a weight matrix maps inputs (the previous layer) to outputs (the next layer).



Neural Networks (Cont'd)

- The weight matrix W^m at the mth layer is

$$W^m = \begin{bmatrix} w^m_{11} & \cdots & w^m_{1n_m} \\ & \ddots & \\ w^m_{n_{m-1}1} & \cdots & w^m_{n_{m-1}n_m} \end{bmatrix}_{n_{m-1}\times n_m},$$

- n_m : # neurons at layer m
- n_{m-1} : # neurons at layer $m-1$
- L: number of layers
- $n_0 =$ # of features, $n_L =$ # of classes
- Let z^m be the input of mth layer. $z^0 = \mathbf{x}$ and z^L is the output

Neural Networks (Cont'd)

From $(m-1)$th layer to mth layer

$$\begin{aligned} \mathbf{s}^m &= (W^m)^T \boldsymbol{z}^{m-1}, \\ z_j^m &= \sigma(s_j^m),\ j = 1, \ldots, n_m, \end{aligned}$$

$\sigma(\cdot)$ is the activation function. We collect all variables:

$$\boldsymbol{\theta} = \begin{bmatrix} \text{vec}(W^1) \\ \vdots \\ \text{vec}(W^L) \end{bmatrix} \in R^n$$

$$\begin{aligned} &n : \text{total \# variables} \\ =\ &n_0 n_1 + \cdots + n_{L-1} n_L \end{aligned}$$

Neural Networks (Cont'd)

- We solve the following optimization problem,

$$\min_{\boldsymbol{\theta}} \quad f(\boldsymbol{\theta}),$$

  where

$$f(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} + C\sum\nolimits_{i=1}^{l} \xi(z^{L,i}(\boldsymbol{\theta}); \mathbf{x}_i, \mathbf{y}_i).$$

  C: regularization parameter
- $z^L(\boldsymbol{\theta}) \in R^{n_L}$: last-layer output vector of $\mathbf{x}$.
  $\xi(z^L; \mathbf{x}, \mathbf{y})$: loss function. Example:

$$\xi(z^L; \mathbf{x}, \mathbf{y}) = ||z^L - \mathbf{y}||^2$$

Neural Networks (Cont'd)

- That is, we hope

$$\mathbf{y} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad z^L = \begin{bmatrix} \pm 0.00\cdots \\ \vdots \\ \pm 0.00\cdots \\ 1.00\cdots \\ \pm 0.00\cdots \\ \vdots \\ \pm 0.00\cdots \end{bmatrix}$$

Neural Networks (Cont'd)

- The formulation is as before, but loss function is more complicated
- This NN method has been developed for decades. So what's new about deep learning?
- Though there are some technical advances, one major thing is that more layers often lead to better results

Solving Optimization Problems I

- How do you minimize

$$f(\boldsymbol{\theta})?$$

- Usually by a descent method
- That is, we find a sequence

$$\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \ldots,$$

such that

$$f(\boldsymbol{\theta}_1) > f(\boldsymbol{\theta}_2) > f(\boldsymbol{\theta}_3) > \cdots$$

Solving Optimization Problems II

- Hopefully
$$\lim_{k \to \infty} f(\boldsymbol{\theta}_k)$$
exists and is the smallest function value
- Now you see that calculus is used. You need to know what limit is
- But how to obtain
$$f(\boldsymbol{\theta}_{k+1}) < f(\boldsymbol{\theta}_k)$$
- Usually by gradient descent

Gradient Descent I

- Taylor expansion. If

$$f(\theta) : R^1 \rightarrow R^1$$

$$f(\theta_k + d) = f(\theta_k) + f'(\theta_k)d + \frac{1}{2}f''(\theta_k)d^2 + \cdots$$

- This is the one-dimensional case
- Now we have multiple variables

$$f(\boldsymbol{\theta}) : R^n \rightarrow R^1$$

Gradient Descent II

- So we need multi-dimensional Taylor expansion

$$f(\boldsymbol{\theta}_k + \mathbf{d}) = f(\boldsymbol{\theta}_k) + \nabla f(\boldsymbol{\theta}_k)^T \mathbf{d} + \cdots$$

- We don't get into details, but $\nabla f(\boldsymbol{\theta})$ is called the gradient
- Gradient is the multi-dimensional first derivative

$$\nabla f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1} \\ \vdots \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix}$$

Gradient Descent III

- Let
$$f(\boldsymbol{\theta}_k + \mathbf{d}) \approx f(\boldsymbol{\theta}_k) + \nabla f(\boldsymbol{\theta}_k)^T \mathbf{d}$$
and we can find $\mathbf{d}$ by
$$\min_{\mathbf{d}} \nabla f(\boldsymbol{\theta}_k)^T \mathbf{d}$$
- But easily this value goes to $-\infty$
- If
$$\nabla f(\boldsymbol{\theta}_k)^T \mathbf{d} = -100,$$
then
$$100 \nabla f(\boldsymbol{\theta}_k)^T \mathbf{d} = -10,000$$

Gradient Descent IV

- Thus we need to confine the search space of **d**

$$\begin{aligned} \min_{\mathbf{d}} \quad & \nabla f(\boldsymbol{\theta}_k)^T \mathbf{d} \\ \text{subject to} \quad & \|\mathbf{d}\| = 1 \end{aligned} \tag{3}$$

- Here $\|\mathbf{d}\|$ means the length of **d**:

$$\sqrt{d_1^2 + \cdots + d_n^2}$$

- How to solve (3)?

Gradient Descent V

- We will use Cauchy inequality

$$\begin{aligned}&(a_1 b_1 + \cdots + a_n b_n)^2\\ \le&(a_1^2 + \cdots + a_n^2)(b_1^2 + \cdots + b_n^2)\end{aligned}$$

- When

$$\mathbf{d} = \frac{-\nabla f(\boldsymbol{\theta}_k)}{\|\nabla f(\boldsymbol{\theta}_k)\|},$$

  we have

$$\begin{aligned}&\|\nabla f(\boldsymbol{\theta}_k)^T \mathbf{d}\|^2 = \|\nabla f(\boldsymbol{\theta}_k)\|^2\\ =&\|\nabla f(\boldsymbol{\theta}_k)\|^2 \|\mathbf{d}\|^2\end{aligned}$$

Gradient Descent VI

- Equality holds for Cauchy inequality
- Thus the minimum of (3) is obtained
- However, we may not have

$$f(\boldsymbol{\theta}_k + \mathbf{d}) < f(\boldsymbol{\theta}_k)$$

- Instead, we need to search for a step size

Gradient Descent VII

- Specifically we try

$$\alpha = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$$

until

$$f(\boldsymbol{\theta}_k + \alpha \mathbf{d}) < f(\boldsymbol{\theta}_k) + \sigma \nabla f(\boldsymbol{\theta}_k)^T \mathbf{d}, \tag{4}$$

where $\sigma \in (0, 1/2)$.

- The condition (4) is usually called sufficient decrease condition in optimization
- The algorithm becomes

Gradient Descent VIII

```
While $\boldsymbol{\theta}$ isn't optimal
    • $\mathbf{d} = -\nabla f(\boldsymbol{\theta})$ and $\alpha \leftarrow 1$
    • while true
            If (4) holds
                break
            else
                $\alpha \leftarrow \alpha/2$
    • $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\mathbf{d}$
```

- The procedure to search for α is called line search

Gradient Descent IX

- Instead of

$$\alpha = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$$

we can use

$$\alpha = 1, \beta, \beta^2, \beta^3, \ldots,$$

where

$$0 < \beta < 1$$

Step-size Search I

- Why
$$\sigma \in (0, \frac{1}{2})?$$
The use of $1/2$ is for convergence though we won't discuss details
- Q: how do we know that the line search procedure is guaranteed to stop?

Step-size Search II

- In fact we can prove that if

$$\nabla f(\boldsymbol{\theta})^T \mathbf{d} < 0 \tag{5}$$

  then there exists $\alpha^* > 0$ such that

$$f(\boldsymbol{\theta} + \alpha \mathbf{d}) < f(\boldsymbol{\theta}) + \sigma \nabla f(\boldsymbol{\theta})^T (\alpha \mathbf{d}), \forall \alpha \in (0, \alpha^*)$$

- Any **d** satisfying (5) is called a descent direction

Step-size Search III

- Proof: assume the result is wrong. There exists a sequence
$$\{\alpha_t\}$$
with
$$\lim_{t\to\infty} \alpha_t = 0 \text{ and } \alpha_t > 0, \forall t$$
such that
$$f(\boldsymbol{\theta} + \alpha_t \mathbf{d}) \geq f(\boldsymbol{\theta}) + \sigma \alpha_t \nabla f(\boldsymbol{\theta})^T \mathbf{d}, \forall t$$

Step-size Search IV

- Then

$$\begin{aligned}&\lim_{\alpha_t \to 0} \frac{f(\boldsymbol{\theta} + \alpha_t \mathbf{d}) - f(\boldsymbol{\theta})}{\alpha_t} \\ =&\nabla f(\boldsymbol{\theta})^T \mathbf{d} \geq \sigma \nabla f(\boldsymbol{\theta})^T \mathbf{d}\end{aligned}$$

However,

$$\nabla f(\boldsymbol{\theta})^T \mathbf{d} < 0 \text{ and } \sigma > 0$$

cause a contradiction

Step-size Search V

- Q: how do you formally say

$$\lim_{\alpha \to 0} \frac{f(\boldsymbol{\theta} + \alpha \mathbf{d}) - f(\boldsymbol{\theta})}{\alpha} = \nabla f(\boldsymbol{\theta})^T \mathbf{d}?$$

- Let

$$g(\alpha) \equiv f(\boldsymbol{\theta} + \alpha \mathbf{d})$$

- We essentially calculate

$$\lim_{\alpha \to 0} \frac{g(\alpha) - g(0)}{\alpha} \tag{6}$$

- By the definition of the first derivative

Step-size Search VI

$$(6) \text{ is } g'(0)$$

- But what are

$$g'(\alpha) \text{ and then } g'(0)?$$

- We have

$$\begin{aligned}
& g'(\alpha) \\
=& \frac{\partial f(\boldsymbol{\theta} + \alpha \mathbf{d})}{\partial \theta_1} \frac{\partial(\theta_1 + \alpha d_1)}{\partial \alpha} + \cdots + \\
& \frac{\partial f(\boldsymbol{\theta} + \alpha \mathbf{d})}{\partial \theta_n} \frac{\partial(\theta_n + \alpha d_n)}{\partial \alpha} \\
=& \frac{\partial f(\boldsymbol{\theta} + \alpha \mathbf{d})}{\partial \theta_1} d_1 + \cdots + \frac{\partial f(\boldsymbol{\theta} + \alpha \mathbf{d})}{\partial \theta_n} d_n \\
=& \nabla f(\boldsymbol{\theta} + \alpha \mathbf{d})^T \mathbf{d}
\end{aligned}$$

Step-size Search VII

and

$$g'(0) = \nabla f(\boldsymbol{\theta})^T \mathbf{d}$$

This is multi-variable chain rule

- Statement of multi-variable chain rule: let

$$x = x(t) \text{ and } y = y(t)$$

be differentiable at t and suppose

$$z = f(x, y)$$

Step-size Search VIII

is differentiable at (x, y). Then

$$z(t) = f(x(t), y(t))$$

is differentiable at t and

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}$$

Gradient of NN I

- Recall that NN optimization problem is

$$\min_{\boldsymbol{\theta}} \quad f(\boldsymbol{\theta}), \quad \text{where}$$

$$f(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} + C \sum\nolimits_{i=1}^{l} \xi(\boldsymbol{z}^{L,i}(\boldsymbol{\theta}); \mathbf{x}_i, \mathbf{y}_i).$$

- How to calculate the gradient?
- Now $\boldsymbol{z}^L$ is actually a function of all variables

$$\boldsymbol{z}^L(\boldsymbol{\theta})$$

Gradient of NN II

- What we will calculate is

$$\nabla f(\boldsymbol{\theta}) = \theta + C \sum_{i=1}^{l} \nabla_{\boldsymbol{\theta}} \xi(z^{L,i}(\boldsymbol{\theta}); \mathbf{x}_i, \mathbf{y}_i)$$

- So what is

$$\nabla_{\boldsymbol{\theta}} \xi(z^{L}(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y})?$$

Gradient of NN III

- We have

$$\frac{\partial \xi(\boldsymbol{z}^L(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y})}{\partial \theta_1} =$$
$$\frac{\partial \xi(\boldsymbol{z}^L(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y})}{\partial z^L_1} \frac{\partial z^L_1(\boldsymbol{\theta})}{\partial \theta_1} + \cdots + \frac{\partial \xi(\boldsymbol{z}^L(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y})}{\partial z^L_{n_L}} \frac{\partial z^L_{n_L}(\boldsymbol{\theta})}{\partial \theta_1}$$

$$\frac{\partial \xi(\boldsymbol{z}^L(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y})}{\partial \theta_2} =$$
$$\frac{\partial \xi(\boldsymbol{z}^L(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y})}{\partial z^L_1} \frac{\partial z^L_1(\boldsymbol{\theta})}{\partial \theta_2} + \cdots + \frac{\partial \xi(\boldsymbol{z}^L(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y})}{\partial z^L_{n_L}} \frac{\partial z^L_{n_L}(\boldsymbol{\theta})}{\partial \theta_n}$$

Gradient of NN IV

- Thus

$$\nabla_{\boldsymbol{\theta}} \xi(z^L(\boldsymbol{\theta}); \mathbf{x}, \mathbf{y}) = \begin{bmatrix} \frac{\partial z_1^L(\boldsymbol{\theta})}{\partial \theta_1} & \cdots & \frac{\partial z_{n_L}^L(\boldsymbol{\theta})}{\partial \theta_1} \\ & \ddots & \\ \frac{\partial z_1^L(\boldsymbol{\theta})}{\partial \theta_n} & \cdots & \frac{\partial z_{n_L}^L(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix} \begin{bmatrix} \frac{\partial \xi}{\partial z_1^L} \\ \vdots \\ \frac{\partial \xi}{\partial z_{n_L}^L} \end{bmatrix}$$

where

$$\begin{bmatrix} \frac{\partial z_1^L(\boldsymbol{\theta})}{\partial \theta_1} & \cdots & \frac{\partial z_{n_L}^L(\boldsymbol{\theta})}{\partial \theta_1} \\ & \ddots & \\ \frac{\partial z_1^L(\boldsymbol{\theta})}{\partial \theta_n} & \cdots & \frac{\partial z_{n_L}^L(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix}$$

Gradient of NN V

is called the Jacobian of $z^L(\boldsymbol{\theta})$

- We see that chain rule is used again
- There are a lot of more details about the gradient evaluation but let's stop here
- The point is that techniques behind deep learning is quite complicated and needs lots of mathematics
- Next let's switch to the issue of computation

Outline

Matrix Multiplication I

- We will show that to calculate

$$f(\boldsymbol{\theta})$$

  the main operation from one layer to next is a matrix-matrix product

- Recall from $(m-1)$th layer to mth layer

$$\begin{aligned} \mathbf{s}^m &= (W^m)^T \boldsymbol{z}^{m-1}, \\ z_j^m &= \sigma(s_j^m),\ j = 1, \ldots, n_m, \end{aligned}$$

  where $\sigma(\cdot)$ is the activation function.

Matrix Multiplication II

- Now each instance $\mathbf{x}_i$ has $z^{m-1,i}$
- So we have
  $$z^{m-1,1}, \ldots, z^{m-1,l}$$
  if there are l training instances
- Thus
  $$\begin{bmatrix} \mathbf{s}^{m,1} & \cdots & \mathbf{s}^{m,l} \end{bmatrix} = W_m^T \begin{bmatrix} z^{m-1,1} & \cdots & z^{m-1,l} \end{bmatrix} \in R^{n_m \times l},$$
  where
  $$W_m \in R^{n_{m-1} \times n_m}$$

Matrix Multiplication III

- The main cost in calculating function value of NN is the

  matrix-matrix product

  between every two layers
- You know how to do matrix multiplication.

$$C = AB$$

  is a mathematics operation with

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Matrix Multiplication IV

- At the first glance, it has nothing to do with computer science
- But have you ever thought about a question: why do people use GPU for deep learning?
- An Internet search shows the following answer from `https://www.quora.com/Why-are-GPUs-well-suited-to-deep-learning`
- "Deep learning involves huge amount of matrix multiplications and other operations which can be massively parallelized and thus sped up on GPU-s."

Matrix Multiplication V

- As a computer science student, we need to know a bit more details
- I am going to use CPU rather than GPU to give an illustration – how computer architectures may affect a mathematics operation

Optimized BLAS: an Example by Using Block Algorithms I

- Let's test the matrix multiplication
- A C program:

```
#define n 2000
double a[n][n], b[n][n], c[n][n];

int main()
{
   int i, j, k;
   for (i=0;i<n;i++)
```

Optimized BLAS: an Example by Using Block Algorithms II

```
    for (j=0;j<n;j++) {
        a[i][j]=1; b[i][j]=1;
    }

for (i=0;i<n;i++)
    for (j=0;j<n;j++) {
        c[i][j]=0;
        for (k=0;k<n;k++)
            c[i][j] += a[i][k]*b[k][j];
    }
```

Optimized BLAS: an Example by Using Block Algorithms III

```
}
```

- A Matlab program

```
n = 2000;
A = randn(n,n); B = randn(n,n);
t = cputime; C = A*B; t = cputime -t
```

- To remove the effect of multi-threading, use

```
matlab -singleCompThread
```

- Timing is an issue

  Elapsed time versus CPU time

Optimized BLAS: an Example by Using Block Algorithms IV

```
cjlin@linux1:~$ matlab -singleCompThread
>> a = randn(3000,3000);tic; c = a*a; toc
Elapsed time is 4.520684 seconds.
>> a = randn(3000,3000);t=cputime; c = a*a;
t=cputime-t

t =
    4.3500
```

Optimized BLAS: an Example by Using Block Algorithms V

```
cjlin@linux1:~$ matlab
>> a = randn(3000,3000);tic; c = a*a; toc
Elapsed time is 1.180799 seconds.
>> a = randn(3000,3000);t=cputime; c = a*a;
t=cputime-t

t =
     8.4400
```

- Matlab is much faster than a code written by ourselves. Why ?

Optimized BLAS: an Example by Using Block Algorithms VI

- Optimized BLAS: data locality is exploited
- Use the highest level of memory as possible
- Block algorithms: transferring sub-matrices between different levels of storage

  localize operations to achieve good performance

Memory Hierarchy I

$$\begin{tikzcd}
\text{CPU} \arrow[d] \\
\text{Registers} \arrow[d] \\
\text{Cache} \arrow[d] \\
\text{Main Memory} \arrow[d] \\
\text{Secondary storage (Disk)}
\end{tikzcd}$$

Memory Hierarchy II

- $\uparrow$: increasing in speed
- $\downarrow$: increasing in capacity
- When I studied computer architecture, I didn't quite understand that this setting is so useful
- But from optimized BLAS I realize that it is extremely powerful

Memory Management I

- Page fault: operand not available in main memory
  transported from secondary memory
  (usually) overwrites page least recently used
- I/O increases the total time
- An example: $C = AB + C$, $n = 1,024$
- Assumption: a page 65,536 doubles = 64 columns
- 16 pages for each matrix
  48 pages for three matrices

Memory Management II

- Assumption: available memory 16 pages, matrices access: column oriented

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

  column oriented: 1 3 2 4

  row oriented: 1 2 3 4
- access each row of A: 16 page faults, $1024/64 = 16$
- Assumption: each time a continuous segment of data into one page
- Approach 1: inner product

Memory Management III

```
for i =1:n
  for j=1:n
    for k=1:n
      c(i,j) = a(i,k)*b(k,j)+c(i,j);
    end
  end
end
```

We use a matlab-like syntax here

- At each (i,j): each row a(i, 1:n) causes 16 page faults

Memory Management IV

Total: $1024^2 \times 16$ page faults

- at least 16 million page faults
- Approach 2:

```
for j =1:n
  for k=1:n
    for i=1:n
      c(i,j) = a(i,k)*b(k,j)+c(i,j);
    end
  end
end
```

Memory Management V

- For each j, access all columns of A
  A needs 16 pages, but B and C take spaces as well
  So A must be read for every j
- For each j, 16 page faults for A
  1024×16 page faults
  C, B : 16 page faults
- Approach 3: block algorithms (nb $= 256$)

Memory Management VI

```
for j =1:nb:n
  for k=1:nb:n
    for jj=j:j+nb-1
      for kk=k:k+nb-1
        c(:,jj) = a(:,kk)*b(kk,jj)+c(:,jj);
      end
    end
  end
end
```

In MATLAB, `1:256:1025` means 1, 257, 513, 769

Memory Management VII

- Note that we calculate

$$\begin{bmatrix} A_{11} & \cdots & A_{14} \\ & \vdots & \\ A_{41} & \cdots & A_{44} \end{bmatrix} \begin{bmatrix} B_{11} & \cdots & B_{14} \\ & \vdots & \\ B_{41} & \cdots & B_{44} \end{bmatrix}$$

$$= \begin{bmatrix} A_{11}B_{11} + \cdots + A_{14}B_{41} & \cdots \\ \vdots & \ddots \end{bmatrix}$$

Memory Management VIII

- Each block: 256×256

$$\begin{aligned} C_{11} &= A_{11}B_{11} + \cdots + A_{14}B_{41} \\ C_{21} &= A_{21}B_{11} + \cdots + A_{24}B_{41} \\ C_{31} &= A_{31}B_{11} + \cdots + A_{34}B_{41} \\ C_{41} &= A_{41}B_{11} + \cdots + A_{44}B_{41} \end{aligned}$$

- For each (j, k), $B_{k,j}$ is used to add $A_{:,k}B_{k,j}$ to $C_{:,j}$

Memory Management IX

- Example: when $j = 1, k = 1$

$$\begin{aligned} C_{11} &\leftarrow C_{11} + A_{11}B_{11} \\ &\vdots \\ C_{41} &\leftarrow C_{41} + A_{41}B_{11} \end{aligned}$$

- Use Approach 2 for $A_{:,1}B_{11}$
- $A_{:,1}$: 256 columns, $1024 \times 256/65536 = 4$ pages. $A_{:,1}, \ldots, A_{:,4} : 4 \times 4 = 16$ page faults in calculating $C_{:,1}$
- For A: 16×4 page faults

Memory Management X

- B: 16 page faults, C: 16 page faults
- Now let's try to compare approaches 1 and 2
- We see that approach is faster. Why?
- C is row-oriented rather than column-oriented

Optimized BLAS Implementations

- OpenBLAS

  `http://www.openblas.net/`

  It is an optimized BLAS library based on GotoBLAS2 (see the story in the next slide)

  It's a successful open-source project developed in China
- Intel MKL (Math Kernel Library)

  `https://software.intel.com/en-us/mkl`

Some Past Stories about Optimized BLAS

- BLAS by Kazushige Goto

  `https://www.tacc.utexas.edu/research-development/tacc-software/gotoblas2`

- See the NY Times article: "Writing the fastest code, by hand, for fun: a human computer keeps speeding up chips"

  `http://www.nytimes.com/2005/11/28/technology/28super.html?pagewanted=all`

Homework I

- We would like to compare the time for multiplying two $8,000$ by $8,000$ matrices
- Directly using sources of blas
  `http://www.netlib.org/blas/`
- Intel MKL
- OpenBLAS
- You can use BLAS or CBLAS
- Try to comment on the use of multi-core processors.

Conclusions I

- In general I don't think we should have too many required courses
- However, some of them are very basic and are very useful in advanced topics
- Some students do not think basic mathematics courses (e.g., calculus) are CS courses. But that may not be the case
- When I evaluate applications for graduate schools by checking their transcripts, very often I first look at the grade of calculus

Conclusions II

- I hope that through this lecture you have seen that some mathematics techniques are very related to CS topics