Newton Method for Convolutional Neural Networks

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Outline

Outline

Introduction

- Training a neural network involves a difficult optimization problem
- SG (stochastic gradient) is the major optimization technique for deep learning.
- SG is simple and effective, but sometimes not robust (e.g., selecting the learning rate may be difficult)
- Is it possible to consider other methods?
- In this work, we investigate Newton methods

Outline

Optimization and Neural Networks

- In a typical setting, a neural network is no more than an empirical risk minimization problem
- We will show an example using convolutional neural networks (CNN)
- CNN is a type of networks useful for image classification

Convolutional Neural Networks (CNN)

- Consider a K-class classification problem with training data

$$(\boldsymbol{y}^i, Z^{1,i}), \quad i = 1, \ldots, \ell.$$

$\boldsymbol{y}^i$: label vector $\qquad Z^{1,i}$: input image

- If $Z^{1,i}$ is in class k, then

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

- CNN maps each image $Z^{1,i}$ to $\boldsymbol{y}^i$

Convolutional Neural Networks (CNN)

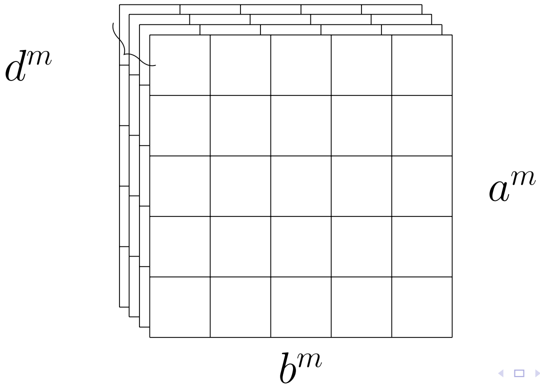
- Typically, CNN consists of multiple convolutional layers followed by fully-connected layers.
- We discuss only convolutional layers.
- Input and output of a convolutional layer are assumed to be images.

Convolutional Layers

For mth layer, let the input be an image

$$a^m \times b^m \times d^m.$$

a^m: height, b^m: width, and d^m: #channels.



Convolutional Layers (Cont'd)

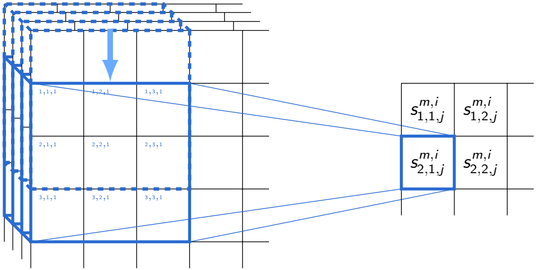
- Consider d^{m+1} filters.
- Each filter includes weights to extract local information
- Filter $j \in \{1, \ldots, d^{m+1}\}$ has dimensions

$$h \times h \times d^m.$$

$$\begin{bmatrix} w^{m,j}_{1,1,1} & & w^{m,j}_{1,h,1} \\ & \ddots & \\ w^{m,j}_{h,1,1} & & w^{m,j}_{h,h,1} \end{bmatrix} \ldots \begin{bmatrix} w^{m,j}_{1,1,d^m} & & w^{m,j}_{1,h,d^m} \\ & \ddots & \\ w^{m,j}_{h,1,d^m} & & w^{m,j}_{h,h,d^m} \end{bmatrix}.$$

h: filter height/width (m of h^m omitted)

Convolutional Layers (Cont'd)



- To compute the jth channel of output, we scan the input from top-left to bottom-right to obtain the sub-images of size $h \times h \times d^m$
- Then calculate the inner product between each sub-image and the jth filter

Convolutional Layers (Cont'd)

- It's known that convolutional operations can be done by matrix-matrix and matrix-vector operations
- Let's collect images of all channels as the input

$$
\begin{aligned}
& Z^{m,i} \\
= & \begin{bmatrix} z^{m,i}_{1,1,1} & z^{m,i}_{2,1,1} & \cdots & z^{m,i}_{a^m,b^m,1} \\ \vdots & \vdots & \ddots & \vdots \\ z^{m,i}_{1,1,d^m} & z^{m,i}_{2,1,d^m} & \cdots & z^{m,i}_{a^m,b^m,d^m} \end{bmatrix} \\
\in & \mathbb{R}^{d^m \times a^m b^m}.
\end{aligned}
$$

Convolutional Layers (Cont'd)

- Let all filters

$$W^m = \begin{bmatrix} w^{m,1}_{1,1,1} & w^{m,1}_{2,1,1} & \cdots & w^{m,1}_{h,h,d^m} \\ \vdots & \vdots & \ddots & \vdots \\ w^{m,d^{m+1}}_{1,1,1} & w^{m,d^{m+1}}_{2,1,1} & \cdots & w^{m,d^{m+1}}_{h,h,d^m} \end{bmatrix}$$
$$\in \mathbb{R}^{d^{m+1} \times hhd^m}$$

  be variables (parameters) of the current layer
- Usually a bias term is considered but we omit it here

Convolutional Layers (Cont'd)

- Operations at a layer

$$S^{m,i} = W^m \phi(Z^{m,i}) \qquad Z^{m+1,i} = \sigma(S^{m,i}),$$

- $\phi(Z^{m,i})$ collects all sub-images in $Z^{m,i}$ into a matrix

$$\phi(Z^{m,i}) = \begin{bmatrix} z^{m,i}_{1,1,1} & z^{m,i}_{1+s^m,1,1} & & z^{m,i}_{1+(a^{m+1}-1)s^m,1+(b^{m+1}-1)s^m,1} \\ z^{m,i}_{2,1,1} & z^{m,i}_{2+s^m,1,1} & & z^{m,i}_{2+(a^{m+1}-1)s^m,1+(b^{m+1}-1)s^m,1} \\ \vdots & \vdots & \cdots & \vdots \\ z^{m,i}_{h,h,1} & z^{m,i}_{h+s^m,h,1} & & z^{m,i}_{h+(a^{m+1}-1)s^m,h+(b^{m+1}-1)s^m,1} \\ \vdots & \vdots & & \vdots \\ z^{m,i}_{h,h,d^m} & z^{m,i}_{h+s^m,h,d^m} & & z^{m,i}_{h+(a^{m+1}-1)s^m,h+(b^{m+1}-1)s^m,d^m} \end{bmatrix}$$

Convolutional Layers (Cont'd)

- σ is an element-wise activation function
- In the matrix-matrix product

$$S^{m,i} = W^m \phi(Z^{m,i}), \tag{1}$$

each element is the inner product between a filter and a sub-image

Optimization Problem

- We collect all weights to a vector variable $\boldsymbol{\theta}$.

$$\boldsymbol{\theta} = \begin{bmatrix} \text{vec}(W^1) \\ \vdots \\ \text{vec}(W^L) \end{bmatrix} \in R^n, \quad n : \text{total \# variables}$$

- The output of the last fully-connected layer L is a vector $\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta})$.
- Consider any loss function such as the squared loss

$$\xi_i(\boldsymbol{\theta}) = ||\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}) - \boldsymbol{y}^i||^2.$$

Optimization Problem (Cont'd)

- The optimization problem is

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

where

$$\begin{aligned} f(\boldsymbol{\theta}) &= \text{regularization} + \text{losses} \\ &= \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{\ell}\sum_{i=1}^{\ell}\xi_i(\boldsymbol{\theta}) \end{aligned}$$

- C: regularization parameter.

Outline

Mini-batch Stochastic Gradient

- We begin with explaining why stochastic gradient (SG) is popular for deep learning
- Recall the function is

$$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{\ell}\sum_{i=1}^{\ell}\xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

- The gradient is

$$\frac{\boldsymbol{\theta}}{C} + \frac{1}{\ell}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{\ell}\xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

Mini-batch Stochastic Gradient (Cont'd)

- Going over all data is time consuming
- From

$$E(\nabla_{\boldsymbol{\theta}} \xi(\boldsymbol{\theta}; \boldsymbol{y}, Z^1)) = \frac{1}{\ell} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{\ell} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

we may just use a subset S (called a batch)

$$\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

Mini-batch SG: Algorithm

1: Given an initial learning rate η.
2: **while** **do**
3: Choose $S \subset \{1, \ldots, \ell\}$.
4: Calculate

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$

5: May adjust the learning rate η
6: **end while**

- But deciding a suitable learning rate may be tricky

Why SG Popular for Deep Learning?

- The special property of data classification is essential

$$E(\nabla_{\boldsymbol{\theta}} \xi(\boldsymbol{\theta}; \boldsymbol{y}, Z^1)) = \frac{1}{\ell} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{\ell} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})$$

  Indeed stochastic gradient is less used outside machine learning

- High-order methods with fast final convergence may not be needed in machine learning

  An approximate solution may give similar accuracy to the final solution

Why SG Popular for Deep Learning? (Cont'd)

- Easy implementation. It's simpler than methods using, for example, second derivative
- Non-convexity plays a role
  - For convex, a global minimum usually gives a good model (loss is minimized)
    Thus we want to efficiently find the global minimum
  - But for non-convex, efficiency to reach a stationary point is less useful

Drawback of SG

- Tuning the learning rate is not easy
- Thus if we would like to consider other methods, robustness rather than efficiency may be the main reason

Newton Method

- Newton method finds a direction $\boldsymbol{d}$ that minimizes the second-order approximation of $f(\boldsymbol{\theta})$

$$\min_{\boldsymbol{d}} \quad \nabla f(\boldsymbol{\theta})^\top \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^\top \nabla^2 f(\boldsymbol{\theta})\boldsymbol{d}. \tag{2}$$

- If $\nabla^2 f(\boldsymbol{\theta})$ is positive definite, (2) is equivalent to solving

$$\nabla^2 f(\boldsymbol{\theta})\boldsymbol{d} = -\nabla f(\boldsymbol{\theta}).$$

Newton Method (Cont'd)

while stopping condition not satisfied **do**

Let G be $\nabla^2 f(\boldsymbol{\theta})$ or its approximation

Exactly or approximately solve

$$G\boldsymbol{d} = -\nabla f(\boldsymbol{\theta})$$

Find a suitable step size α (e.g., line search)

Update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{d}.$$

end while

Hessian may not be Positive Definite

Hessian of $f(\boldsymbol{\theta})$ is (derivation omitted)

$$\nabla^2 f(\boldsymbol{\theta}) = \frac{1}{C}\mathcal{I} + \frac{1}{\ell}\sum_{i=1}^{\ell}(J^i)^\top B^i J^i$$
$$+ \text{ a non-PSD (positive semi-definite) term}$$

$\mathcal{I}$: identity, B^i: simple PSD matrix, J^i: Jacobian of $z^{L+1,i}(\boldsymbol{\theta})$

$$J^i = \begin{bmatrix} \frac{\partial z_1^{L+1,i}}{\partial \theta_1} & \cdots & \frac{\partial z_1^{L+1,i}}{\partial \theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{n_{L+1}}^{L+1,i}}{\partial \theta_1} & \cdots & \frac{\partial z_{n_{L+1}}^{L+1,i}}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^{n_{L+1} \times n}$$

n_{L+1}: # classes
n: # total variables

Positive Definite Modification of Hessian

- Several strategies have been proposed.
- For example, Schraudolph (2002) considered the Gauss-Newton matrix (which is PD)

$$G = \frac{1}{C}\mathcal{I} + \frac{1}{\ell}\sum_{i=1}^{\ell}(J^i)^\top B^i J^i \approx \nabla^2 f(\boldsymbol{\theta}).$$

- Then Newton linear system becomes

$$G\boldsymbol{d} = -\nabla f(\boldsymbol{\theta}). \tag{3}$$

Memory Difficulty

- The Gauss-Newton matrix G may be too large to be stored

$$G : \# \text{ variables } \times \# \text{ variables}$$

- Many approaches have been proposed (through approximation)
- For example, we may store and use only diagonal blocks of G

Memory Difficulty (Cont'd)

- Here we try to use the original Gauss-Newton matrix G without aggressive approximation
- Reason: we should show first that for median-sized data, standard Newton is more robust than SG
- Otherwise, there is no need to develop techniques for large-scale problems

Hessian-free Newton Method

- If G has certain structures, it's possible to use iterative methods (e.g., conjugate gradient) to solve the Newton linear system by a sequence of matrix-vector products

  $$G\boldsymbol{v}^1, G\boldsymbol{v}^2, \ldots$$

  without storing G
- This is called Hessian-free in optimization

Hessian-free Newton Method (Cont'd)

- The Gauss-Newton matrix is

$$G = \frac{1}{C}\mathcal{I} + \frac{1}{\ell}\sum_{i=1}^{\ell}(J^i)^\top B^i J^i$$

- Matrix-vector product without explicitly storing G

$$G\boldsymbol{v} = \frac{1}{C}\boldsymbol{v} + \frac{1}{\ell}\sum_{i=1}^{\ell}((J^i)^\top(B^i(J^i\boldsymbol{v}))).$$

- Examples of using this setting for deep learning include Martens (2010), Le et al. (2011), and Wang et al. (2018).

Hessian-free Newton Method (Cont'd)

- However, for the conjugate gradient process,

$$J^i \in \mathbb{R}^{n_{L+1} \times n}, i = 1, \ldots, \ell,$$

can be too large to be stored (ℓ is # data)

- Total memory usage is

$$\begin{aligned} & n_{L+1} \times n \times \ell \\ = & \ \# \text{ classes} \times \# \text{ variables} \times \# \text{ data} \end{aligned}$$

Hessian-free Newton Method (Cont'd)

- The product involves

$$\sum_{i=1}^{\ell}((J^i)^\top(B^i(J^i\boldsymbol{v}))).$$

- We can trade time for space: J^i is calculated when needed (i.e., at every matrix-vector product)
- On the other hand, we may not need to use all data points to have $J^i, \forall i$
- We will discuss the subsampled Hessian technique

Subsampled Hessian Newton Method

- Similar to gradient, for Hessian we have

$$E(\nabla^2_{\boldsymbol{\theta},\boldsymbol{\theta}}\xi(\boldsymbol{\theta};\boldsymbol{y},Z^1)) = \frac{1}{\ell}\nabla^2_{\boldsymbol{\theta},\boldsymbol{\theta}}\sum_{i=1}^{\ell}\xi(\boldsymbol{\theta};\boldsymbol{y}^i,Z^{1,i})$$

- Thus we can approximate the Gauss-Newton matrix by a subset of data
- This is the subsampled Hessian Newton method (Byrd et al., 2011; Martens, 2010; Wang et al., 2015)

Subsampled Hessian Newton Method

- We select a subset $S \subset \{1, \ldots, \ell\}$ and have

$$G^S = \frac{1}{C}\mathcal{I} + \frac{1}{|S|}\sum_{i \in S}(J^i)^T B^i J^i \approx G.$$

- The cost of storing J^i is reduced from

$$\propto \ell \quad \text{to} \quad \propto |S|$$

Subsampled Hessian Newton Method

- With enough data, direction obtained by

$$G^S \boldsymbol{d} = -\nabla f(\boldsymbol{\theta})$$

  may be close to that by

$$G\boldsymbol{d} = -\nabla f(\boldsymbol{\theta})$$

- Computational cost per matrix-vector product is saved
- On CPU we may afford to store $J^i, \forall i \in S$
- On GPU, which has less memory, we calculate $J^i, \forall i \in S$ when needed

Calculation of Jacobian Matrix

- Now we know the subsampled Gauss-Newton matrix-vector product is

$$G^S \boldsymbol{v} = \frac{1}{C}\boldsymbol{v} + \frac{1}{|S|}\sum_{i \in S}\left((J^i)^T\left(B^i(J^i\boldsymbol{v})\right)\right) \qquad (4)$$

- We briefly discuss how to calculate J^i

Calculation of Jacobian Matrix (Cont'd)

The Jacobian can be partitioned with respect to layers.

$$J^i = \begin{bmatrix} \frac{\partial z_1^{L+1,i}}{\partial \theta_1} & \cdots & \frac{\partial z_1^{L+1,i}}{\partial \theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{n_{L+1}}^{L+1,i}}{\partial \theta_1} & \cdots & \frac{\partial z_{n_{L+1}}^{L+1,i}}{\partial \theta_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(W^1)^\top} & \cdots & \frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(W^L)^\top} \end{bmatrix}$$

We check details of one layer. It's difficult to calculate the derivative if using a matrix form

$$S^{m,i} = W^m \phi(Z^{m,i})$$

Calculation of Jacobian Matrix (Cont'd)

We can rewrite it to

$$\text{vec}(S^{m,i}) = (\phi(Z^{m,i})^\top \otimes \mathcal{I}_{d^{m+1}})\text{vec}(W^m),$$

where

$\otimes$: Kronecker product $\qquad \mathcal{I}_{d^{m+1}}$: Identity

If

$$\boldsymbol{y} = A\boldsymbol{x}, \text{ with } \boldsymbol{y} \in \mathbb{R}^p \text{ and } \boldsymbol{x} \in \mathbb{R}^q$$

then

$$\frac{\partial \boldsymbol{y}}{\partial (\boldsymbol{x})^\top} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_q} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_p}{\partial x_1} & \cdots & \frac{\partial y_p}{\partial x_q} \end{bmatrix} = A$$

Calculation of Jacobian Matrix (Cont'd)

Therefore,

$$\frac{\partial z^{L+1,i}}{\partial \text{vec}(W^m)^\top} = \frac{\partial z^{L+1,i}}{\partial \text{vec}(S^{m,i})^\top} \frac{\partial \text{vec}(S^{m,i})}{\partial \text{vec}(W^m)^\top}$$

$$= \frac{\partial z^{L+1,i}}{\partial \text{vec}(S^{m,i})^\top} (\phi(Z^{m,i})^\top \otimes \mathcal{I}_{d^{m+1}}).$$

Further, (detailed derivation omitted)

$$\frac{\partial z^{L+1,i}}{\partial \text{vec}(S^{m,i})^\top} = \frac{\partial z^{L+1,i}}{\partial \text{vec}(Z^{m+1,i})^\top} \odot \left(\mathbb{1}_{n_{L+1}} \text{vec}(\sigma'(S^{m,i}))^\top \right),$$

where $\odot$ is element-wise product, and

Calculation of Jacobian Matrix (Cont'd)

$$\frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \mathrm{vec}(Z^{m,i})^{\top}} = \frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \mathrm{vec}(S^{m,i})^{\top}} (\mathcal{I}_{a^{m+1}b^{m+1}} \otimes W^{m}) P_{\phi}^{m}.$$

- Thus a backward process can calculate all the needed values
- We see that with suitable representation, the derivation is manageable
- Major operations can be performed by matrix-based settings (details not shown)
- This is why GPU is useful

Outline

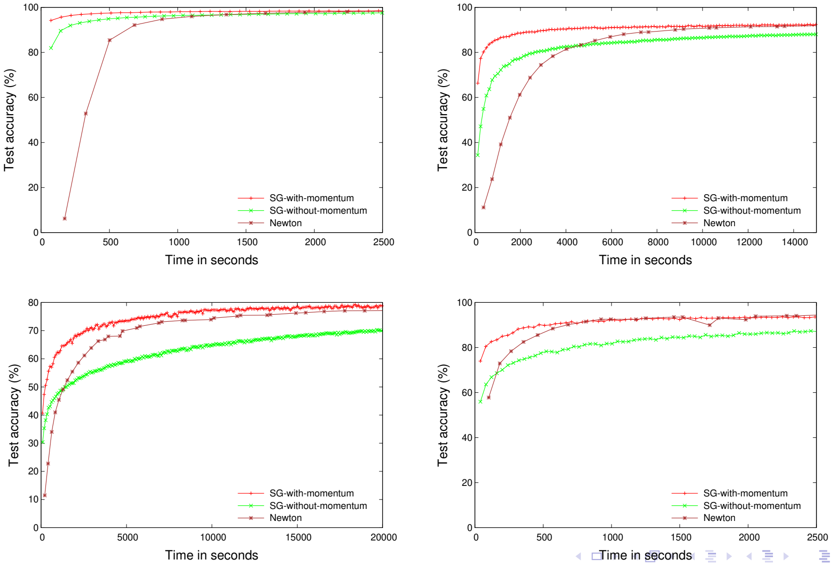
Running Time and Test Accuracy

- Four sets are considered

  MNIST, SVHN, CIFAR10, smallNORB
- For each method, best parameters from a validation process are used

  We will check parameter sensitivity later
- Two SG implementations are used
  - Simple SG shown earlier
  - SG with momentum (details not explained here)
- SG with momentum is a reasonably strong baseline

Running Time and Test Accuracy (Cont'd)



Running Time and Test Accuracy (Cont'd)

- Clearly, SG has faster initial convergence
- This is reasonable as a second-order method is slower in the beginning
- But if cost for parameter selection is considered, Newton may be useful

Experiments on Parameter Sensitivity

- Consider a fixed regularization parameter

$$C = 0.01\ell$$

- For SG with momentum, we consider the following initial learning rates

$$0.1, 0.05, 0.01, 0.005, 0.001, 0.0003, 0.0001$$

- For Newton, there is no particular parameter to tune. We check the size of subsampled Hessian:

$$|S| = 10\%, 5\%, 1\% \text{ of data}$$

Results by Using Different Parameters

Each line shows the result of one problem

Newton			SG				
Sampling rate			Initial learning rate				
10%	5%	1%	0.03	0.01	0.003	0.001	0.0003
99.2%	99.2%	99.1%	9.9%	10.3%	99.1%	99.2%	99.0%
92.7%	92.7%	92.2%	19.5%	92.4%	93.0%	92.7%	92.3%
78.2%	78.3%	75.4%	10.0%	63.1%	79.5%	79.2%	76.9%
94.9%	95.0%	94.6%	64.7%	95.0%	95.0%	95.0%	94.8%

We find that

- a too large learning rate causes SG to diverge, and
- a too small rate causes slow convergence

Outline

Conclusions

- Stochastic gradient method has been popular for CNN
- It is simple and useful, but sometimes not robust
- Newton is more complicated and has slower initial convergence
- However, it may be overall more robust
- By careful designs, the implementation of Newton isn't too complicated

Conclusions (Cont'd)

- Results presented here are based on the paper by Wang et al. (2019)
- An ongoing software development is at `https://github.com/cjlin1/simpleNN`
- Both MATLAB and Python are supported
- MATLAB: joint work with Chien-Chih Wang and Tan Kent Loong (NTU)
- Python: joint work with Pengrui Quan (UCLA)