Deep Learning (CS324)

4. Optimisation and regularisation* (Continued)

Prof. Jianguo Zhang

SUSTech

*Based on http://www.deeplearningbook.org chapters 7 and 8

Plagiarism policy

- Regulations on Academic Misconduct in Assignments for Undergraduate Students in the SUSTech Department of Computer Science and Engineering
- （南方科技大学计算机科学与工程系本科生作业抄袭学术不端行为的认定标准及处理办法）
- From Spring 2018, the plagiarism policy applied by the Computer Science and Engineering department is the following:
- * **If an undergraduate assignment is found to be plagiarized, the first time the score of the assignment will be 0.**
- * **The second time the score of the course will be 0.**

Plagiarism policy

- As it may be difficult when two assignments are identical or nearly identical who actually wrote it, the policy will apply to BOTH students, unless one confesses having copied without the knowledge of the other.

Plagiarism policy
What is OK, and what isn't OK?

- It's OK to work on an assignment with a friend, and think together about the program structure, share ideas and even the global logic. At the time of actually writing the code, you should write it alone.
- It's OK to use in an assignment a piece of code found on the web, as long as you indicate in a comment where it was found and don't claim it as your own work.
- It's OK to help friends debug their programs (you'll probably learn a lot yourself by doing so).
- It's OK to show your code to friends to explain the logic, as long as the friends write their code on their own later.
- It's **NOT OK** to take the code of a friend, make a few cosmetic changes (comments, some variable names) and pass it as your own work.

Plagiarism policy

- **It is important to sign the following forms:**

- Undergraduate Students Assignment Declaration Form (English Version)
- Or
- 本科生作业承诺书 (the Chinese version)

- Those will be made available on BB and please sign it and return this together with the assignment 1. (Failing to sign this, you might not be able to continue your UG program.)

Summary

- Gradient descent and variants
- Momentum and learning rate
- Regularisation
- **Parameters initialisation**
- **Input normalisation**
- **Batch normalisation**

Weight initialisation

- Deep learning training is iterative and strongly depends on the initialisation point (i.e., how we initialise the parameters of the network)
- Important principle: **weight asymmetricity**
  - *Why?* Because if 2 units share the same activation, same weights, and same inputs, they will be updated in the same way (no learning)
  - So, **don't** give same value to all weights (e.g., 0)
  - Sample weights from Gaussian or uniform dists

Weight initialisation

- But, be careful:
  - **Large weights** will have strong symmetry-breaking effect and help to propagate strong signal during forward and backward propagation
  - However they may also cause exploding values, saturation of units
  - But we also don't want **small weights**…
  - **Also**, we want to maintain the same variance for input and output (because outputs are inputs of next layer)

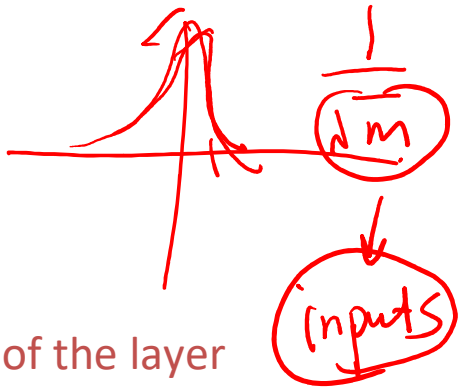
Weight initialisation

- Uniform distribution initialisation (tanh)

$$W_{ij} \sim U\left(-\sqrt{\left(\frac{6}{m+n}\right)}, \sqrt{\left(\frac{6}{m+n}\right)}\right)$$

- Xavier initialisation (tanh)

$$W_{ij} \sim N\left(0, \sqrt{\frac{1}{m}}\right)$$



Where *m* is the number of inputs and *n* is the number of outputs of the layer

Good tutorial: https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/

Weight initialisation

- Uniform distribution initialisation (sigmoid)

$$W_{ij} \sim U\left(-4\sqrt{\left(\frac{6}{m+n}\right)}, 4\sqrt{\left(\frac{6}{m+n}\right)}\right)$$

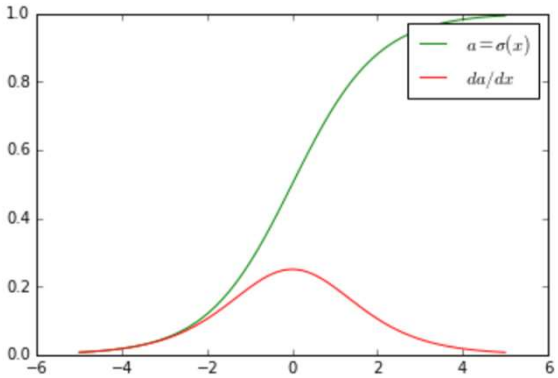
- ReLU initialisation

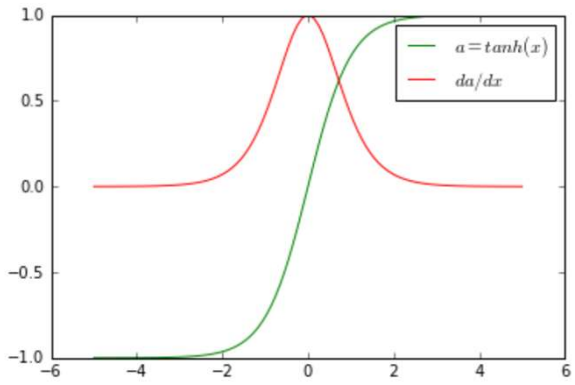
$$W_{ij} \sim N\left(0, \sqrt{\frac{2}{m}}\right)$$

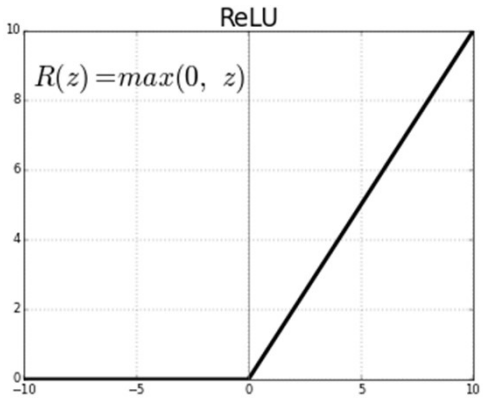
Where *m* is the number of inputs and *n* is the number of outputs of the layer

Data pre-processing

- Activation functions are usually centred around zero

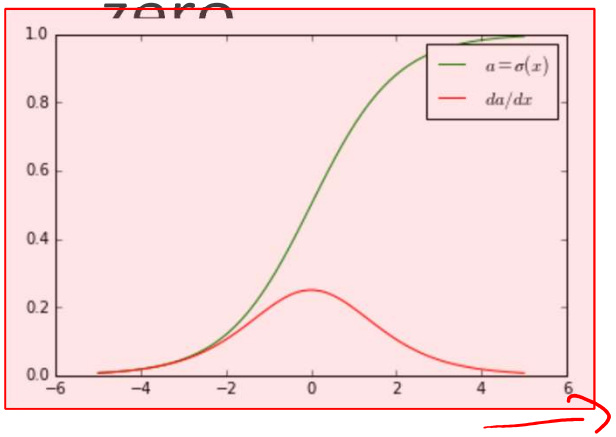


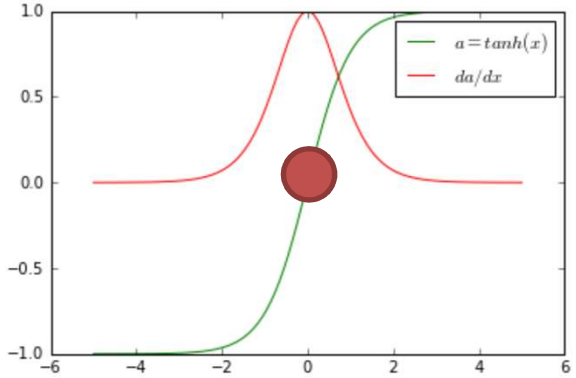


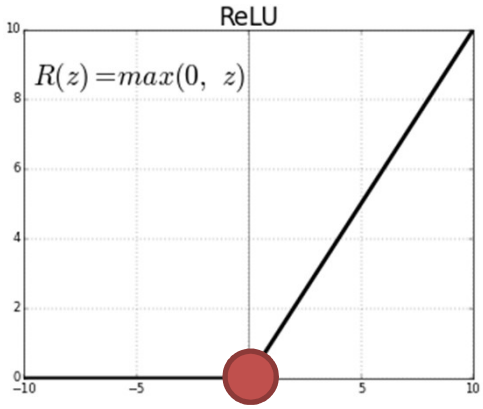


Data pre-processing

- Activation functions are usually centred around zero





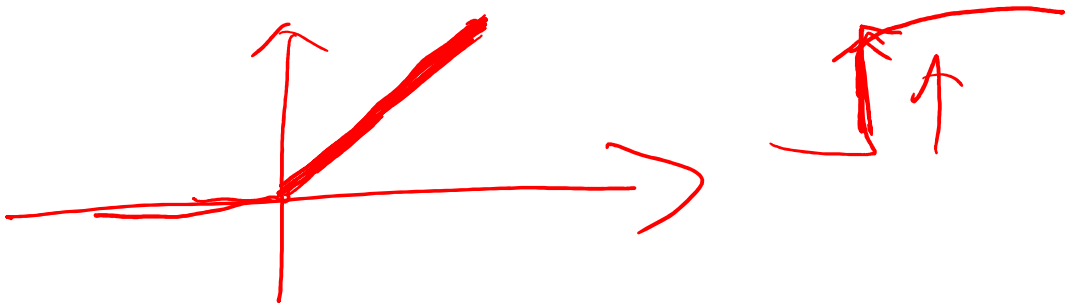


Not the sigmoid :(…

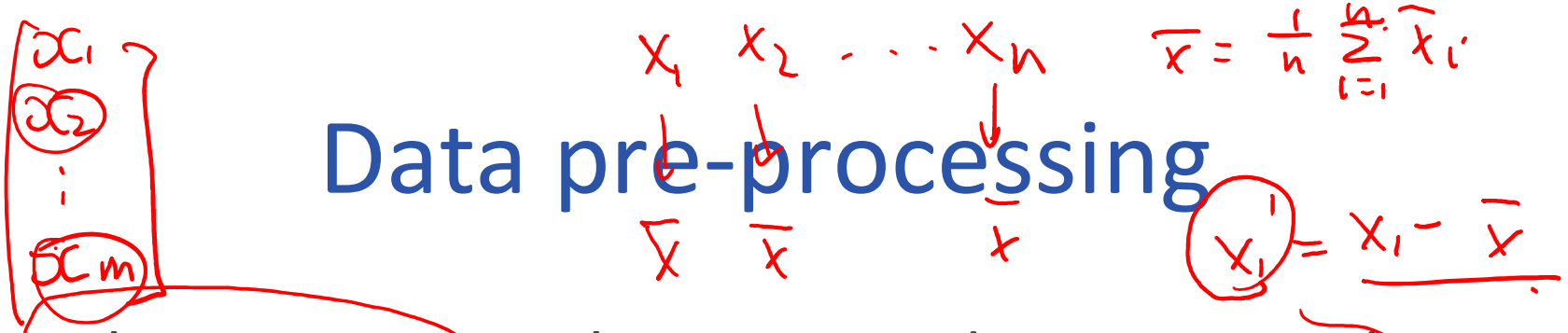
Data pre-processing

- Activation functions are usually centred around zero
- This is a good thing because it helps us to avoid saturation, which leads to **vanishing gradients**
- At the same time, we like **one-sided saturations** because they help to avoid variance due to noise

More about this here: https://blog.paperspace.com/vanishing-gradients-activation-function/



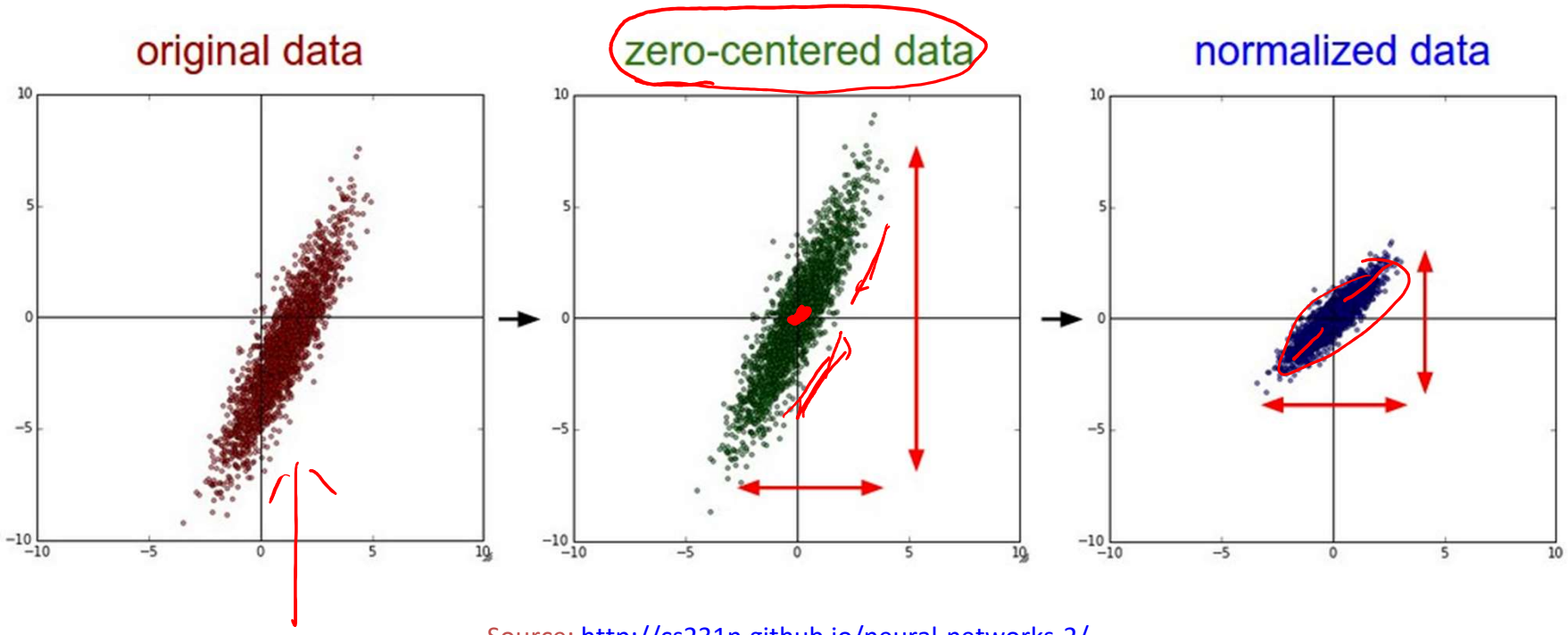
Data pre-processing



- Subtract mean so that training data is centred around zero as well
  - Otherwise may cause vanishing gradients
- Scale inputs to have similar diagonal covariances
  - Otherwise input samples with very different covariances can generate very different gradients and make the gradient update harder

Unit normalisation

- When input variables are normally distributed, subtract mean and divide by standard deviation



Source: http://cs231n.github.io/neural-networks-2/

Batch normalisation

- Two important principles. The distribution of the data fed to the layers of a network should be:
  - zero-centred
  - constant through time and data (mini-batches)

Batch normalisation

- Two important principles. The distribution of the data fed to the layers of a network should be:
  - ~~zero-centred~~ (done)
  - constant through time and data (mini-batches)
- We already took care of the first point, but what about the second one?

Batch normalisation

- Two important principles. The distribution of the data fed to the layers of a network should be:
  - ~~zero-centred~~ (done)
  - constant through time and data (mini-batches)
- We already took care of the first point, but what about the second one?
  - Normalise the activations of each layer!

Batch normalisation

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$$

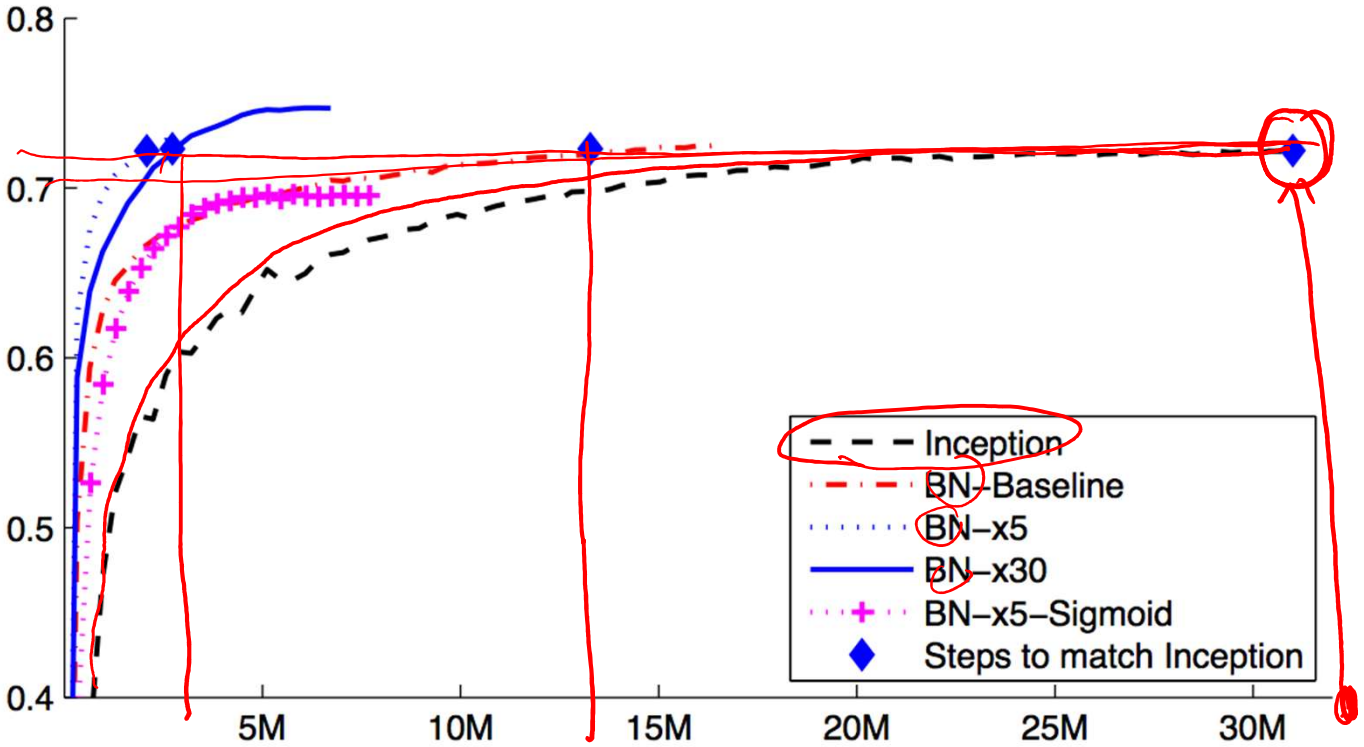
$$\tilde{\boldsymbol{Z}} = \boldsymbol{Z} - \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{Z}_{i,:}$$

$$\hat{\boldsymbol{Z}} = \frac{\tilde{\boldsymbol{Z}}}{\sqrt{\epsilon + \frac{1}{m}\sum_{i=1}^{m} \tilde{\boldsymbol{Z}}_{i,:}^2}}$$

$$\boldsymbol{H} = \max\{0, \boldsymbol{\gamma}\hat{\boldsymbol{Z}} + \boldsymbol{\beta}\}$$

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"
Ioffe and Szegedy 2015

Batch normalisation



"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"
Ioffe and Szegedy 2015

Summary

- Gradient descent and variants
- Momentum and learning rate
- Regularisation
- Parameters initialisation
- Input normalisation
- Batch normalisation