Connectionist Computing
COMP 30230/41390

Gianluca Pollastri

office: E0.95, Science East.

email: gianluca.pollastri@ucd.ie

Credits

- **Geoffrey Hinton, University of Toronto.**
  - borrowed some of his slides for "Neural Networks" and "Computation in Neural Networks" courses.

- **Ronan Reilly, NUI Maynooth.**
  - slides from his CS4018.
- **Paolo Frasconi, University of Florence.**
  - slides from tutorial on Machine Learning for structured domains.

Lecture notes on Brightspace

- Strictly confidential...

- Slim PDF version will be uploaded later, typically the same day as the lecture.

- If there is demand, I can upload onto Brightspace last year’s narrated slides.. (should be very similar to this year’s material)

Books

- No book covers large fractions of this course.

- Parts of chapters 4, 6, (7), 13 of Tom Mitchell's "Machine Learning"
- Parts of chapter V of Mackay's "Information Theory, Inference, and Learning Algorithms", available online at:

http://www.inference.phy.cam.ac.uk/mackay/itprnn/book.html

- Chapter 20 of Russell and Norvig's "Artificial Intelligence: A Modern Approach", also available at:

http://aima.cs.berkeley.edu/newchap20.pdf

- More materials later..

Marking

- 3 landmark papers to read, and submit a 10-line summary on Brightspace about: each worth 6-7%
- a connectionist model to build and play with on some sets, write a report: 30%
- Final Exam in the RDS (50%)

Programming assignment

- Implement a Multi-Layer Perceptron, test it.
- The description on Brightspace.
- Submit through Brightspace code and test results by Dec the 5th at 23:59, any time zone of your choice (Baker Island?).
- 30% of the overall mark
- One third of a grade down every day late, that is: if you deserve an A and you're 1 day late you get an A-, 2 days late a B+, etc.

Assignment 2

- Read the paper "Finding structure in time", by Elman (1990), up to and excluding the *Discovering the notion "word"* section.

- The paper is on Brightspace.

- Submit a 250 word MAX summary by October the 30th at 23:59 on Brightspace, any time zone of your choice (Baker Island?).
- <u>7%</u> of the overall mark
- 1/3 of a grade down every 2 days late, that is: if you deserve an A and you're 1-2 day late you get an A-, 3-4 days late a B+, etc.
- You are responsible for making sure I get it..

Fighting overfitting: combining networks

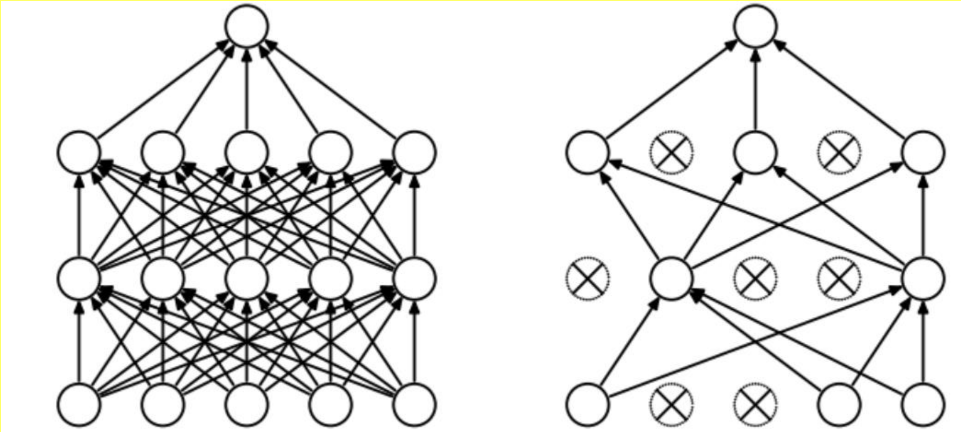
- **Averaging the predictions of many different networks is a good way to avoid overfitting. *Ensemble, ensembling..***
- **It works best if the networks are as different as possible.**

Ensembling with just 1 training: dropout

- **During training, at each step knock out some (different) randomly chosen connections.**

- **When predicting, use all connections (you need to introduce a normalising constant for this to work correctly).**

- **This is equivalent to having a very large ensemble of networks, but you train only once!**



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

Overfitting: summary

- Problem that occurs when the network memorises the examples but not the underlying trends.
- It can be prevented by:
  - limiting number of weights
  - weight decay
  - early stopping
  - combining networks (especially if they disagree)

Learning and gradient descent problems

- Overfitting (general learning problem): the model memorises the examples very well but generalises poorly.
- GD is slow... how can we speed it up?
  - GD does not guarantee that the direction of maximum descent points to the minimum.
  - Sometimes we would like to run where it's flat and slow down when it gets too steep. GD does precisely the contrary.
- Local minima?

Batch learning

- The ideal rule computes one step on the error surface for a whole pass of the training set (epoch). This is called ***batch learning***:

$$\Delta w_{ji} = -\eta \frac{\partial E(\text{all examples})}{\partial w_{ji}}$$

Online learning

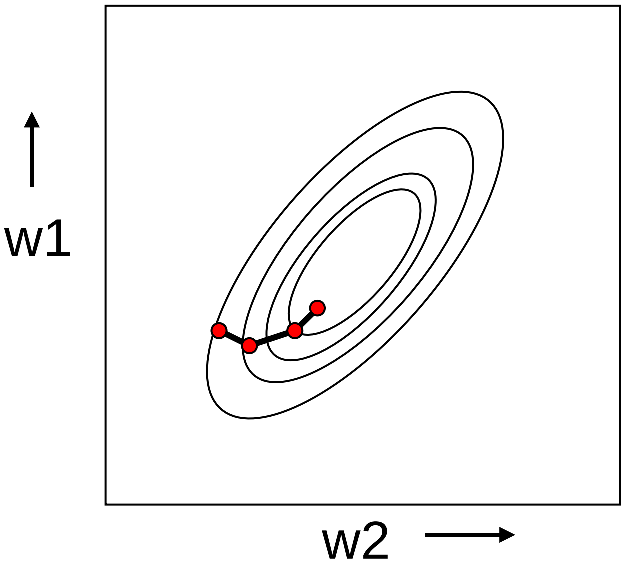
- **An alternative to batch learning is computing the weight update for each example. This is called *online learning*:**

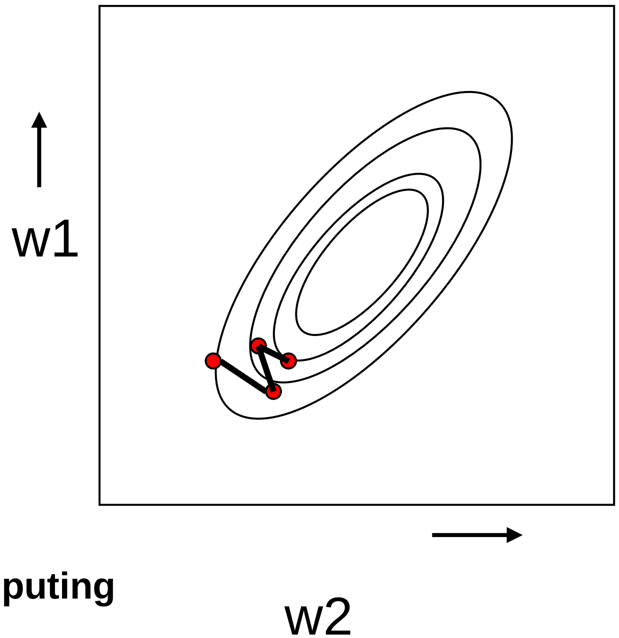
$$\Delta w_{ji} = -\eta \frac{\partial E(\text{one example})}{\partial w_{ji}}$$

Online versus batch learning

- **Batch learning does steepest descent on the error surface**

Online learning zig-zags around the direction of steepest descent.



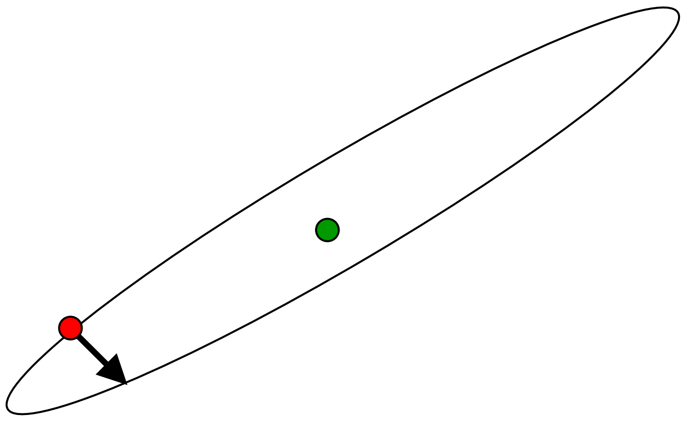


Batch and online

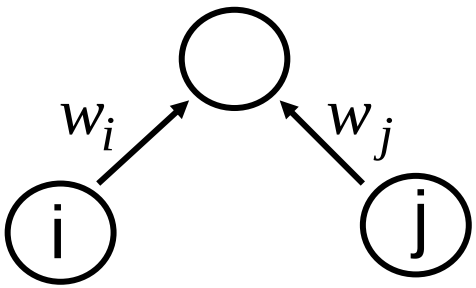
- Batch learning is theoretically more sound.
- Online learning can be *a lot* faster: one step for each example vs one step for the whole training set.
- Intermediate versions are also possible, for instance update after every group of X examples, where X is smaller than the size of the training set.
- In practice it depends on the problem, the format of the input, etc. if online learning is applicable. Trial and error..

More gradient descent problems

- The direction of steepest descent does not necessarily point at the minimum.
- Can we preprocess the data or do something to the gradient so that we move directly towards the minimum?



Fixing up the error surface



Suppose that inputs i and j are highly correlated. Changing wi will change the error in the same way as changing wj.

So changing both will have larger effects on E than changing a weight that isn't correlated to the others

- **Preprocess each input so that it isn't correlated to the others**
- **The gradient will now point at the minimum**
- **This means computing the <u>covariance matrix</u> among all the outputs (OK), inverting it (*not OK*, if there are many inputs), and multiplying each input by the inverse.**

$$Cov(x_i, x_j) = \frac{1}{N}\sum_{n=1}^{N}(x_{i,n} - \mu_i)(x_{j,n} - \mu_j)$$

Yet another GD problem

- The gradient is large where the error is steep, small where the error is flat.
- In general, this is a silly way of going. We would like to run where it's flat and boring and go cautiously where it gets steep.

..fixing it

- Use an adaptive learning rate
  - Increase the rate slowly if it’s not diverging
  - Decrease the rate quickly if it starts diverging
- Use momentum
  - Instead of using the gradient to change the position of the weight, use it to change the velocity of change.
- Use fixed step
  - Use gradient to decide where to go, but always go at the same pace.

.. fixing it (2)

- Normalise the gradient (this is equivalent to adapting the learning rate) based on sum of past gradients or squared gradients or a combination thereof, either all the way back to the beginning of training or (better) over a past window of steps.
- Depending on the precise combination of terms adopted, various techniques: Adagrad, Adadelta, RMSprop, Adam, etc.
- These can speed up training *a lot* and often make the selection of a learning rate much less critical.

Summary on GD problems/improvements

- Compute GD a step on a fraction of the examples. It’s way faster, and helps with local minima (stochastic GD).
- Don’t discount the possibility of working on encoding the inputs in a more efficient way.
- Tamper at will with the magnitude of the gradient and/or the learning rate – this can make learning considerably faster.