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.

Summary: invariances

- **Often as tough a problem as learning after invariances are tackled.**
- **Possible solutions:**
  - **network design**
  - **features**
  - **normalisation**
  - **brute force**

Alternatives: softmax and relative entropy

$$f(z_i) = softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$E = -\sum t_j log y_j$$

- **Non-local non linearity.**
- **Outputs add up to 1 (can be interpreted as the probability of the output given the input).**

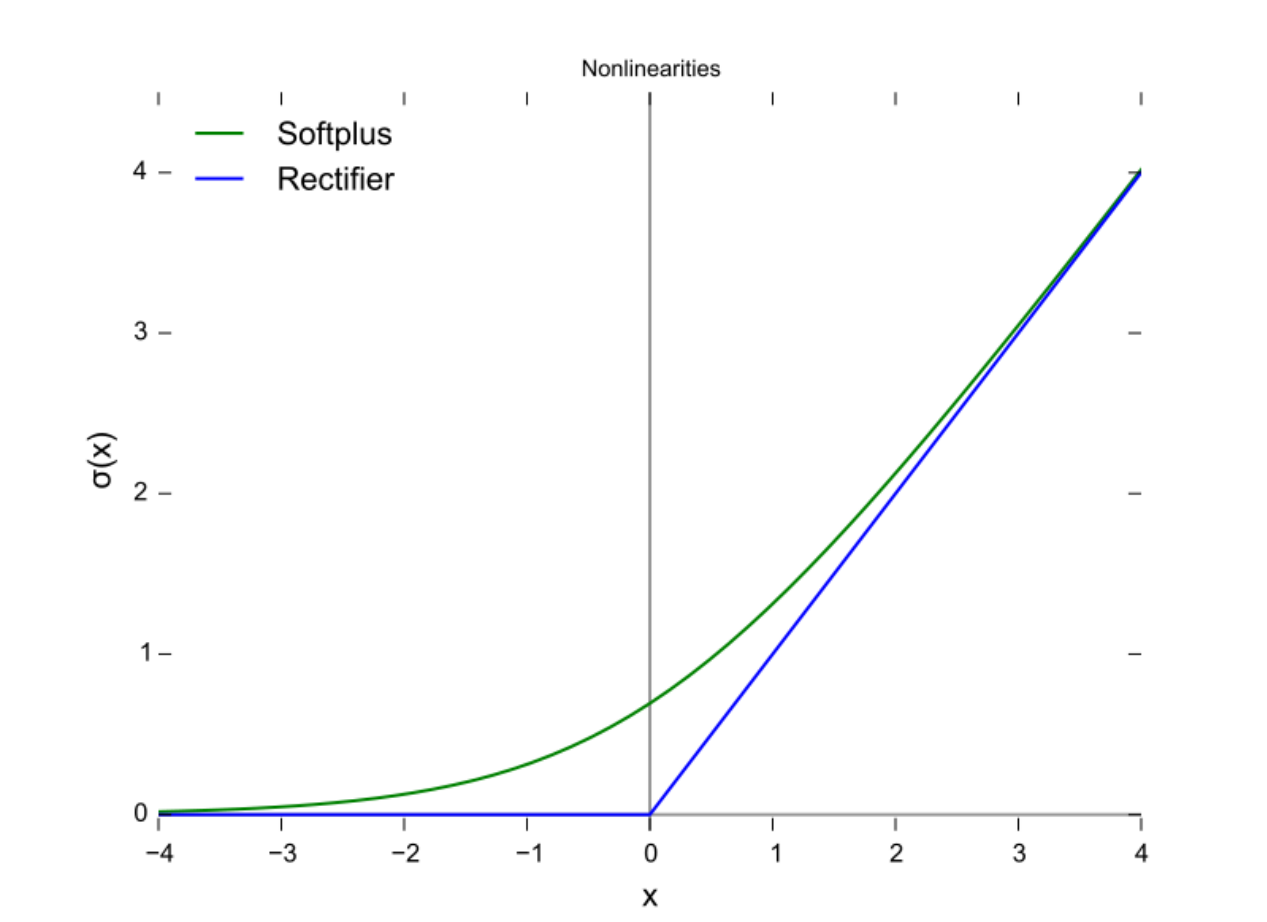
Gradient descent with softmax

- **No f'() : the steepness of the cost function balances the flatness of the output non-linearity.**

$$\frac{\partial E}{\partial w_{ji}} = (y_j - t_j)x_i$$

More squashing functions

- **ReLU**
- **Smooth ReLU**
- **Also: Leaky ReLU, parametric ReLU, ..**



Learning and gradient descent problems

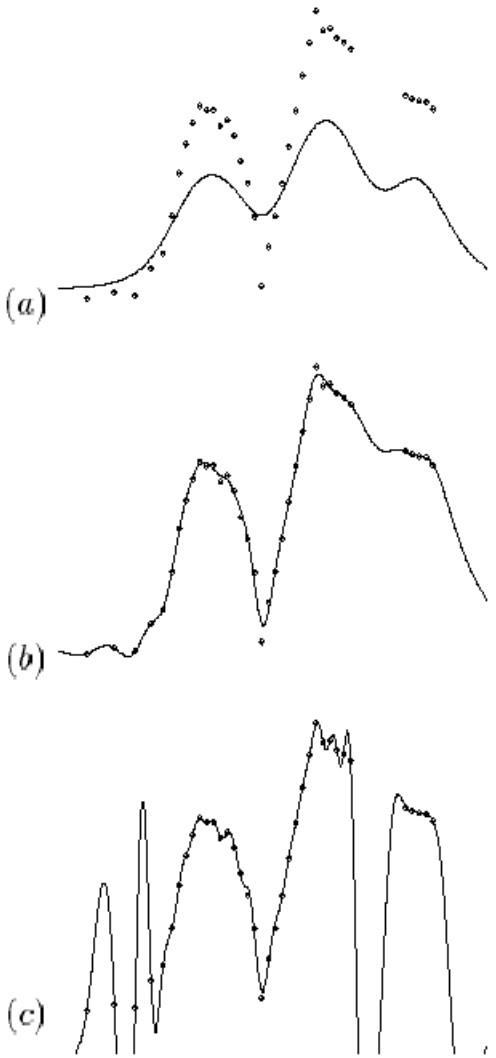
- **Overfitting (general learning problem): the model memorises the examples very well but generalises poorly.**
- **GD is slow... how can we speed it up?**
  - **GD does not guarantee that the direction of maximum descent points to the minimum.**
  - **Sometimes we would like to run where it's flat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

Overfitting

- **The training data contains information about the regularities in the mapping from input to output. But it also contains noise**
  - **The target values may be unreliable.**
  - **There will be accidental regularities just because of the particular training cases that were chosen.**
- **When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.**
  - **So it fits both kinds of regularity.**
  - **If the model is very flexible it can model the sampling error really well. This is a disaster.**

Overfitting

- a) can't fit the examples well enough.
- b) looks great.
- c) fits all points even better than b) but isn't the easiest hypothesis.

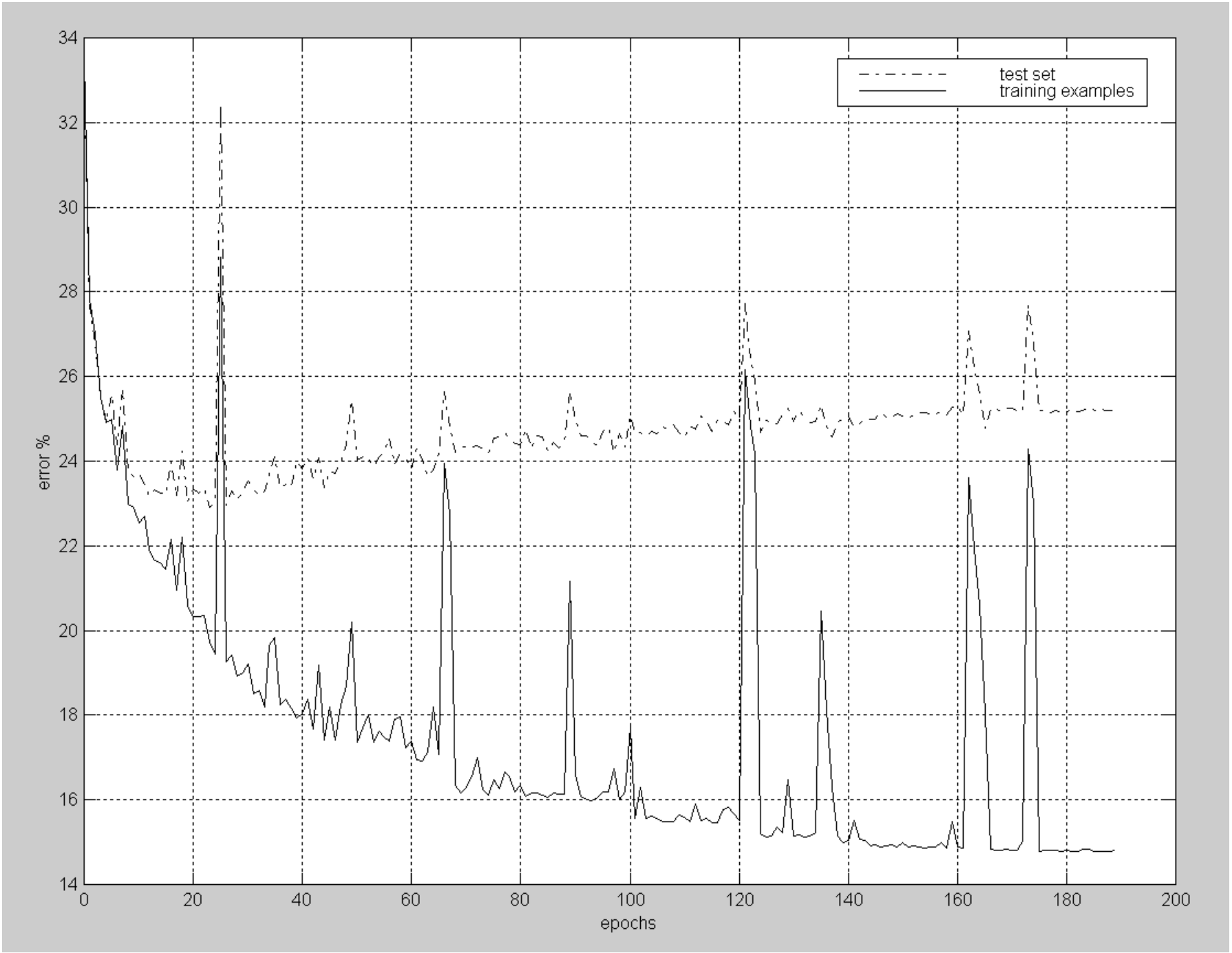


Overfitting: formally

- **A learner overfits the data if**
  - **It outputs a hypothesis $h(\cdot) \in H$ having true error ε and empirical error E, but**
  - **There is another $h'(\cdot) \in H$ having**

$$E' > E \text{ and } \varepsilon' < \varepsilon$$

- **In practice, during learning the error on the training examples decreases all along, but the error in generalisation reaches a minimum and then starts growing again.**


test set
training examples
error %
epochs
34
32
30
28
26
24
22
20
18
16
14
0
20
40
60
80
100
120
140
160
180
200

Ways to prevent overfitting

- **Use a model that has the right capacity:**
  - **enough to model the true regularities**
  - **not enough to also model the spurious regularities (assuming they are weaker).**
  - **problem: how do we know which regularities are real?**
- **Standard ways to limit the capacity of a neural net:**
  - **Limit the number of hidden units.**
  - **Limit the size of the weights.**
  - **Stop the learning before it has time to overfit.**

Limiting the size of the model: using fewer Hidden Units

- If n=number of inputs and M=number of hidden units the VC of a MLP is proportional to (n M logM).
- Reducing M reduces the capacity of the model -> prevents overfitting.
- Of course we need to know the exact capacity we need. Too many weights: overfitting. Too few weights: underfitting.
- In practice: trial and error.

Limiting the size of the model: weight decay

- **<u>Weight-decay</u> involves adding an extra term to the cost function that penalises the squared weights.**
  - **Keeps weights small unless they have large error derivatives.**

$$C = E + \frac{\lambda}{2} \sum_i w_i^2$$

$$\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$$

Weight decay

- **It prevents the network from using weights that it does not need.**
- **It tends to keep the network in the linear region, where its capacity is lower.**
- **This helps to stop it from fitting the sampling error. It makes a smoother model in which the output changes more slowly as the input changes.**
- **It can often improve generalisation a lot.**

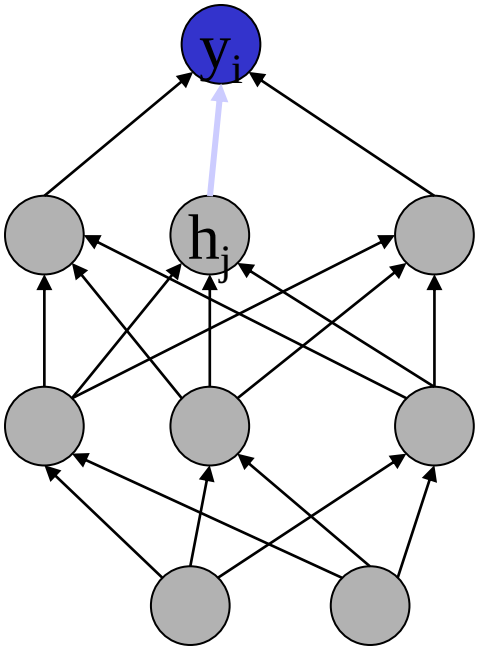
Weight decay as *regularisation*

- **Weight decay is a form of regularisation.**
- **Many other forms of it may be devised, all of which have the same basic aim (prevent the model from overfitting).**
- **The difference between them is what function we use to penalise large weights.**

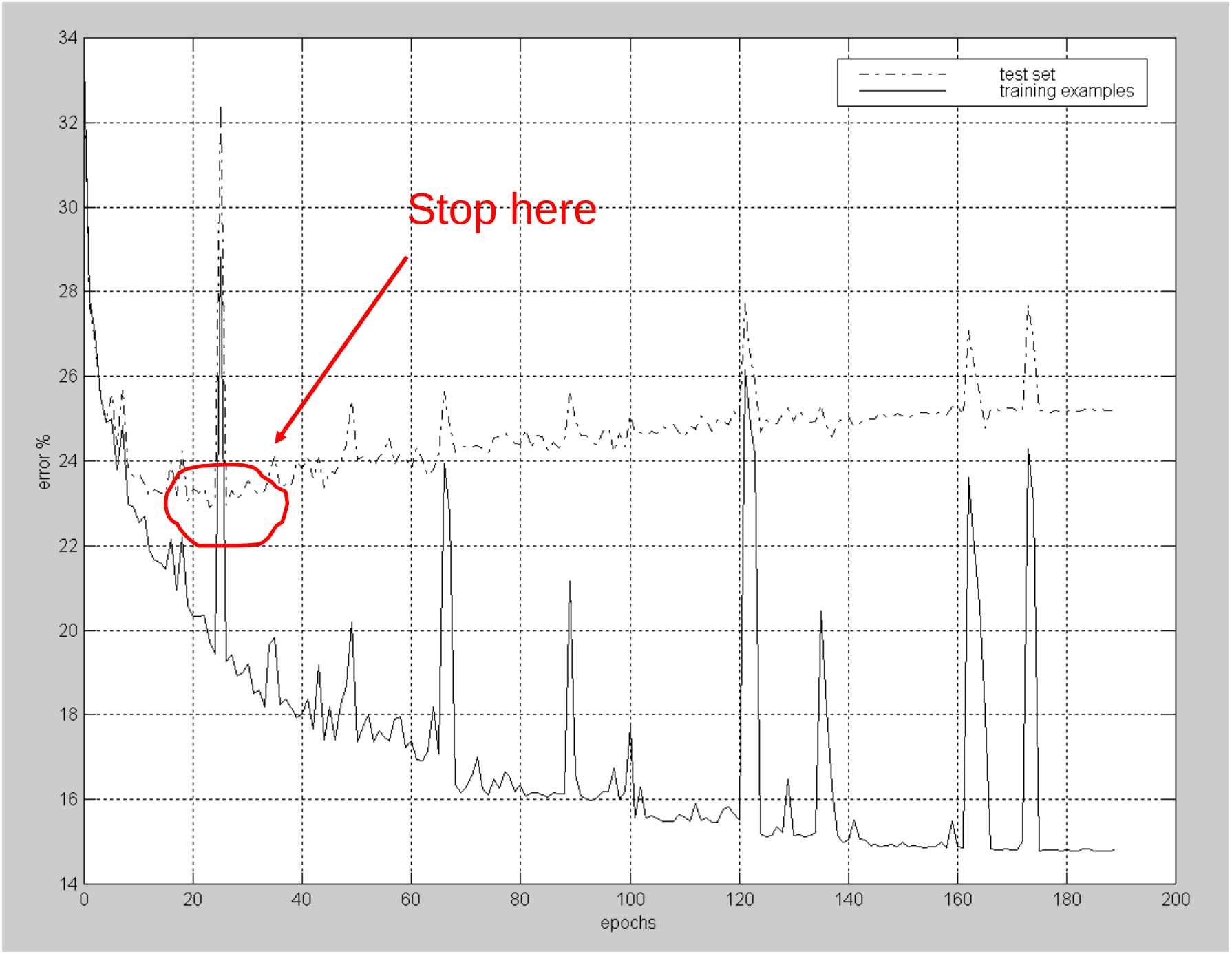
Preventing overfitting: early stopping

- If we have lots of data and a big model, it's very expensive to keep re-training it with different amounts of weight decay.
- It is much cheaper to start with very small weights and let them grow until the performance in generalisation starts getting worse.
- The capacity of the model is limited because the weights have not had time to grow big.

Overfitting in MLP



- **Start learning with small weights (symmetry breaking)**
- **The mapping X→Y is nearly linear: number of effective free parameters (and VC-dim) nearly as in SL perceptron**
- **As optimisation proceeds hidden units tend to get out of the linear region, increasing the effective number of free parameters**
- **A variable-size hypothesis space**


Stop here
test set
training examples
error %
epochs
34
32
30
28
26
24
22
20
18
16
14
0
20
40
60
80
100
120
140
160
180
200

Model selection

- **How do we decide which model is the best? How do we decide when we have the right capacity?**
- **We can use test data. A problem though is that we get an unfair prediction of the error rate we would get on new test data.**
- **So use a separate *validation set* to do model selection.**

Sets

- **Divide the total dataset into three subsets:**
  - **Training data is used for learning the parameters of the model.**
  - **Validation data is not used for learning but is used for deciding what type of model and what amount of regularisation works best.**
  - **Test data is used to get a final, unbiased estimate of how well the network works. We expect this estimate to be worse than on the validation data.**
- **We could then re-divide the total dataset to get another (or more than one) unbiased estimate of the true error rate. N-fold Cross-Validation.**