COMP307/AIML420 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

More on on objective functions and practical aspects of neural networks

Outline

- Objective function
  - Cross entropy
- Weight update frequency
- Learning rate
- Overfitting
- Stopping criteria
- Local minima
- NN architecture
- Momentum

Objective/Loss Function

- Minimising the loss function = finding a good NN
  - Loss is function of weights/parameters, typically denoted as θ or W
- Loss function is often written as J_θ
- The function changes if you change the data set, but the data set is fixed during the optimisation
- Examples:
  - Squared (L2) error
    - For regression
    - In its simplest form: $J_\theta = \sum_{i \in \mathrm{A}} (d_i - y_i)^2$
      - With d_i and y_i scalars
    - More general formulation for vectors: $J_\theta = \sum_{m \in A} \|d_m - y_m\|^2$
      - With d_m and y_m vectors;
      - Norm squared $\|d_m - y_m\|^2$ is sum squared error for vector elements
  - Cross entropy
    - For classification
    - It is complicated

Entropy

- Consider K symbols with probabilities $\{p_1, p_2, \cdots, p_k, \cdots, p_K\}$
  - For example, letters in a document

- Entropy measures (average) information = disorder
  - $-\log p_k$ is ground-truth information in observing symbol k
    - Is minimum number of bits you need to spend for transmitting "it was symbol k"
    - Letter "x" is rare, (p_i small), so observing it provides a lot of information when reading; $-\log_2(0.0029) = 8.4$ bits of information
      - *Her name has an "x" in it* is highly informative (rare)
  - Entropy $\mathrm{H} = -\sum_k p_k \log p_k$ is *average* information per symbol:
    - How many bits we need to spend on average per symbol for a sequence of symbols each drawn from the distribution $\{p_1, p_2, \cdots, p_N\}$
    - Example: the average information per letter in text (without accounting for dependencies)

Cross Entropy

- Consider K symbols/classes with probabilities $\{p_1, p_2, \cdots, p_k, \cdots, p_K\}$

- Entropy is $\mathrm{H} = -\sum_k p_k \log p_k$

- Cross entropy measures bits to transmit if we use our model instead:
  - Model distribution is $\{q_1, q_2, \ldots, q_N\}$; what we think the distribution is
  - $-\log\ q_i$ is what our model says the information is when we observe symbol k
    - How many bits we think we need to spend for transmitting "it was symbol k"
  - Cross entropy $\mathrm{D} = -\sum_k p_k \log q_k$ is *average model* information per symbol
    - How many bits we need to spend on average per symbol for a sequence of symbols each drawn from the distribution $\{p_1, p_2, \cdots, p_K\}$ if we use our model distribution as a basis for assigning bits
  - Cross entropy is always larger than, or equal to, entropy: $D \geq H$
    - Hence, if we minimise cross entropy we try to approximate $\{p_1, p_2, \cdots, p_K\}$ with our model $\{q_1, q_2, \cdots, q_K\}$

Measuring (Cross) Entropy for Data

- Entropy is $\mathrm{H} = -\sum_k p_k \log p_k$
- The $\sum_k p_k$ part indicates we are averaging over the distribution
  - We weight $-\log p_k$ proportional to its rate of occurrence
- For a database A with data $\{x\}$ the symbols k will appear proportional to their probability. Let k_x be the symbol of observation x, then

$$\mathrm{H} \approx -\frac{1}{|A|}\sum_{x \in A} \log p_{k_x}$$

- Similarly, an estimate of cross entropy is:

$$\mathrm{D} \approx -\frac{1}{|A|}\sum_{x \in A} \log q_{k_x}$$

- $|A|$ is number of data in A (cardinality of A)
- We will be sloppy and use “=“ in the following

Cross Entropy Loss for Classification I

symbol k can be entire image

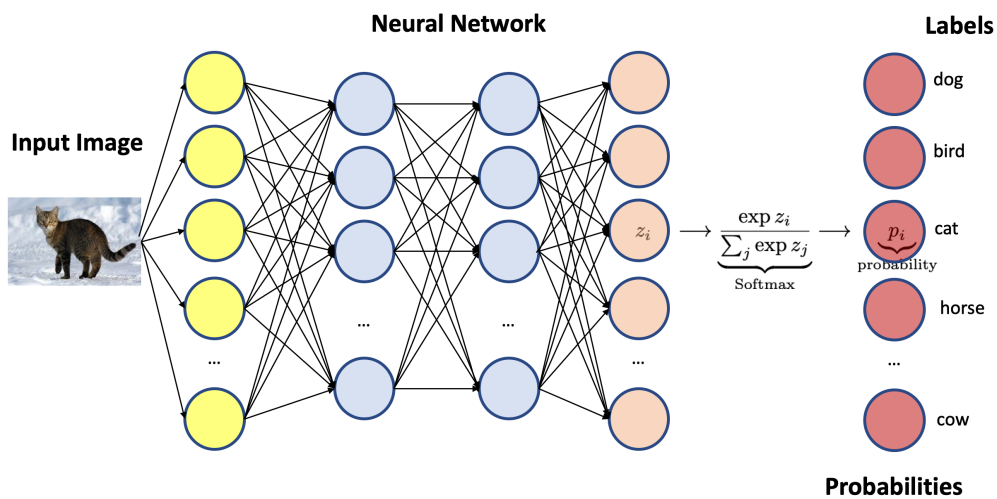
- We have data pairs (class, input symbol) = (c, k);

  hence $\mathrm{D} = -\sum_{(c,k)} p(c,k) \log q(c,k)$

- We consider classification problems where each input k_x belongs to one class
- Rewrite (use product rule): $\mathrm{D} = -\sum_{(c,k)} p(k)p(c|k) \log (q(c|k)p(k))$
  - $q(c|k)$ is model probability of class c for input symbol k with $(q(k) = p(k))$
- For data (see previous slide): $\mathrm{D} = -\frac{1}{N}\sum_x p(c_x|k_x) \log (q(c_x|k_x)p(k_x))$
  - We assumed that averaging data x is like averaging over symbols k
- Remove constant $p(k_x)$, loss is: $\mathrm{J}_\theta = -\sum_x p(c_x|k_x) \log q(c_x|k_x)$
- Observed data *one-hot*: $p(c_x|k_x) = 0$ or $p(c_x|k_x) = 1$
- Can also write $\mathrm{J}_\theta = -\sum_c \sum_{x \in A_c} \log q(c|k_x)$
  - A_c is the subset of training data of class c

Cross Entropy Loss for Classification II

- Loss function $\mathrm{J}_\theta = -\sum_x p(c_x|k_x) \log q(c_x|k_x)$
- Have a neural network for $q(c|k)$
  - Weights / parameters θ; one output neuron for each class
- Ensure that $\sum_c q(c|k) = 1$ by using *softmax* on output layer
  - Converts any output vector to probabilities: $q(c|k) = \frac{\exp z_c(k)}{\sum_c \exp z_c(k)}$
  - Often already integrated into the cross entropy function
- Apply SGD to J_θ to find parameters of classifier NN $q(c|k)$



Weight Update Frequency

- Traditional View:
  - With frequency of weight update = frequency of passes
  - Online learning: a pass for each new input-output example
    - Very slow
  - Offline learning: a pass for all the training instances
    - Weight change is the sum of the changes for all the training instances
    - Slow and often impossible

- Batch learning: a pass for a batch (subset of training instances)
  - Weight change is the sum of the changes for all instances in the batch
  - Now the standard approach

- Batch learning leads to stochastic gradient descent (SGD)
- Offline learning is true gradient descent

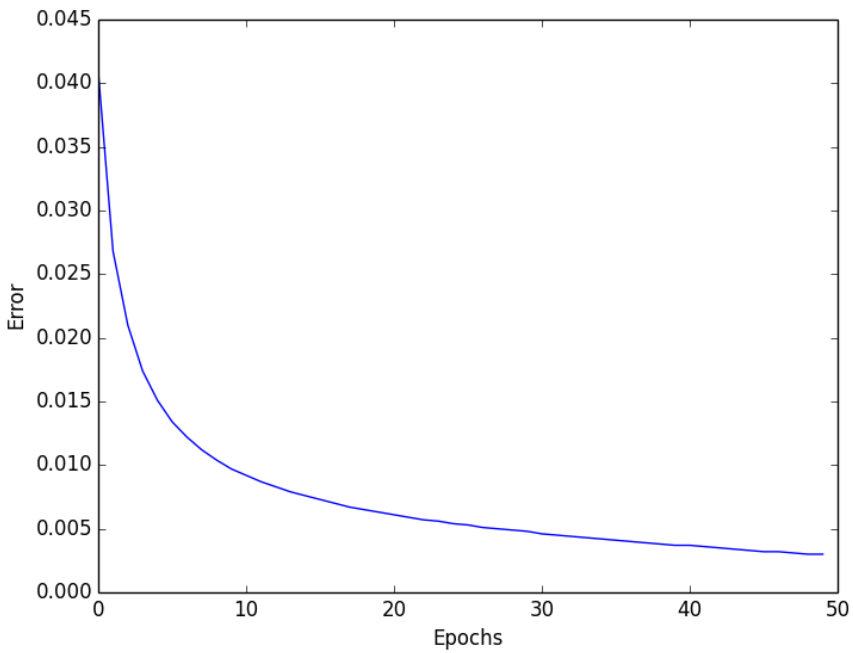
Batch Size

- Assuming a weight $w = 0.2$
- 4 new training instances
- Learning with batch size 1
  - Instance 1, $\Delta w = 0.1$, $w \rightarrow 0.3$
  - Instance 2, $\Delta w = 0.05$, $w \rightarrow 0.35$
  - Instance 3, $\Delta w = 0.03$, $w \rightarrow 0.38$
  - Instance 4, $\Delta w = 0.01$, $w \rightarrow 0.39$
- Learning with batch size 4
  - Instance 1, $\Delta w = 0.1$, $w = 0.2$ unchanged
  - Instance 2, $\Delta w = 0.08$, $w = 0.2$ unchanged
  - Instance 3, $\Delta w = -0.03$, $w = 0.2$ unchanged
  - Instance 4, $\Delta w = 0.05$, $w = 0.2$ unchanged
  - $w \rightarrow 0.2 + 0.1 + 0.08 - 0.03 + 0.05 = 0.4$
- Note that learning rate can scale changes Δw up and down

Epoch and Batch Size

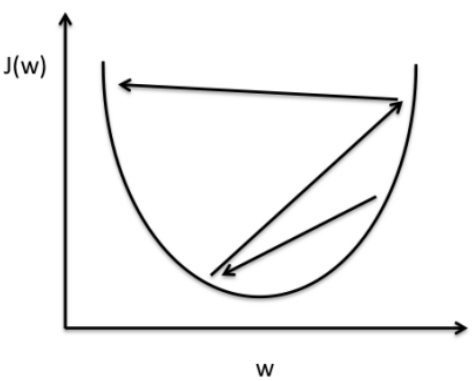
- Epoch: period when all the training instances are used once

- 10000 training instances, batch size = 500, then need 20 iterations to complete one epoch

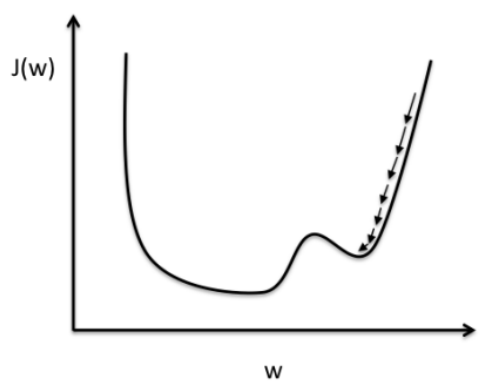


More on Learning Rate

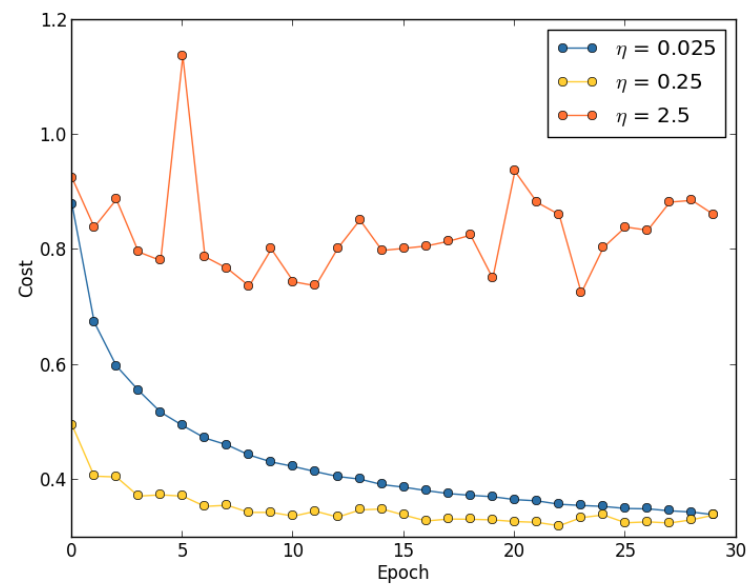
- Learning rate: $\Delta w_{i\to j} = \eta o_i o_j (1 - o_j)\beta_j,$ (sigmoidal case)
- Large learning rate may cause oscillating behaviour
- Small learning rate may cause slow convergence
- Use trial and error to find good value (0.1 – 0.0001 to start)
- Use schedule (standard for optimisers in PyTorch, JAX, etc.)



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

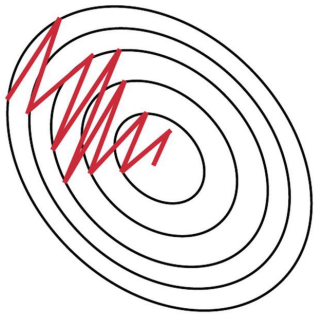


Optimisers Use Momentum

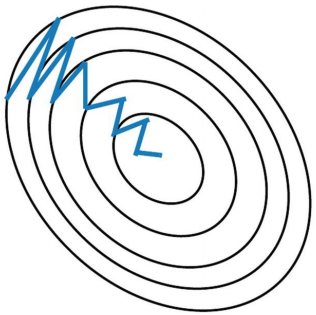
- Large NNs may take days or weeks to train
- Optimiser speed is important
- Momentum is standard optimization approach
  - Simple example: use gradient from last training step

$$\Delta w_{i\to j}(t) \leftarrow \eta o_i o_j \left(1 - o_j\right)\beta_j + \alpha \Delta w_{i\to j}(t-1)$$

  - Choose parameters for problem



Stochastic Gradient Descent **without** Momentum

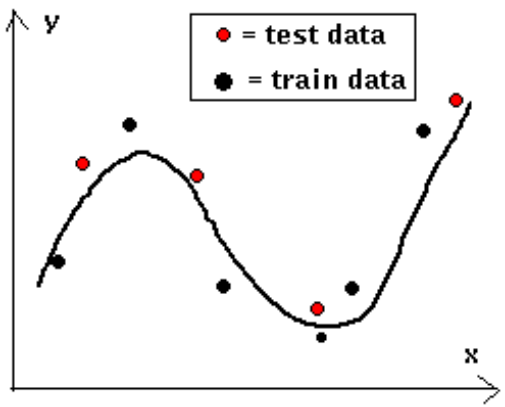


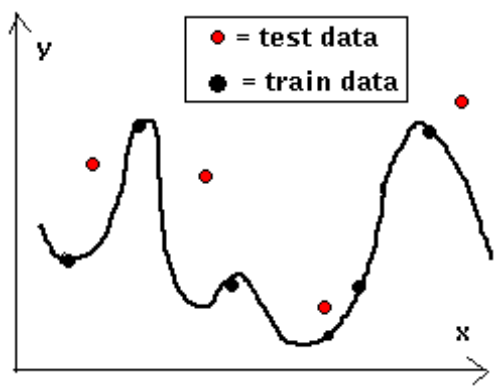
Stochastic Gradient Descent **with** Momentum

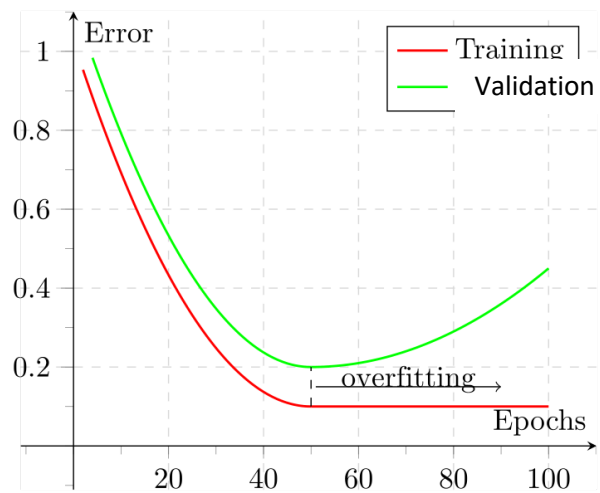
- In practice: use an *optimiser library,* usually *Adam*; does it for you
  - Adam is cited 170.000 times!

Overfitting

- High accuracy on training set, but poor accuracy on test set
- Very common problem; caused by
  - Training for too long
  - Standard argument (so-so): too many weights (parameters) to train
  - Too few training instances
- Conventional thinking: the more parameters to train, the more data (training instances) needed for accuracy





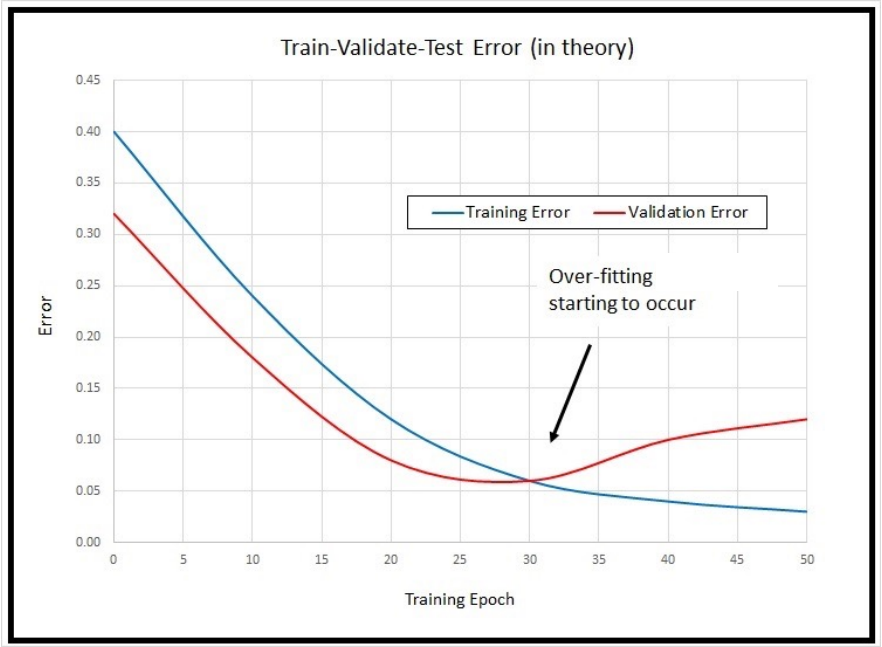


Stopping Criteria

- Traditional:
  - When a certain number of epochs is reached
  - When the error (e.g., mean/total squared error) on the training set is smaller than a threshold
  - Proportion of correctly classified training instances (i.e. accuracy) is larger than a threshold
- Early stopping:
  - Validation control to avoid overfitting
  - Stop when validation error goes back up

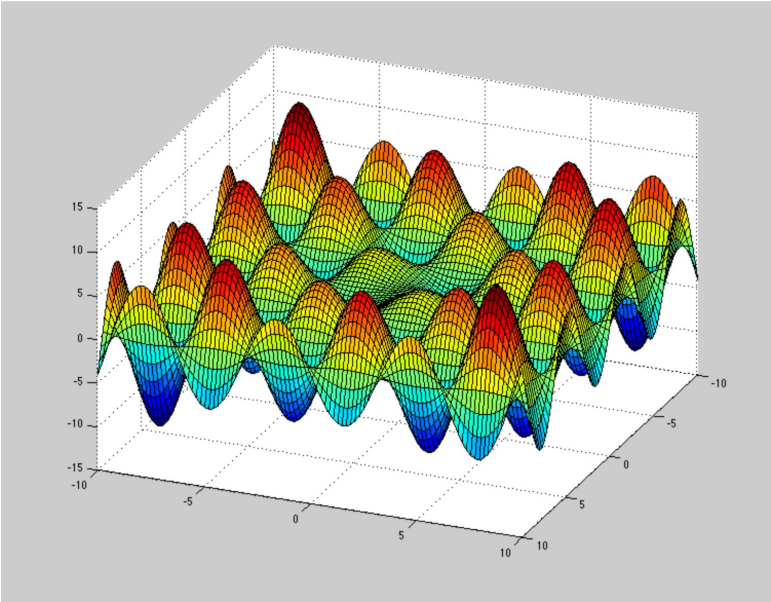
Validation Control

- Split the training set into *training* and *validation* sets:
  - We now have training, validation, and test data sets
- Use training set to compute the weight changes
- Every *m* (e.g., 10, 50, 100) epochs apply the current NN to the validation set to calculate the validation error
- Stop training when the error on the validation set increases
- Only use test set when all is done

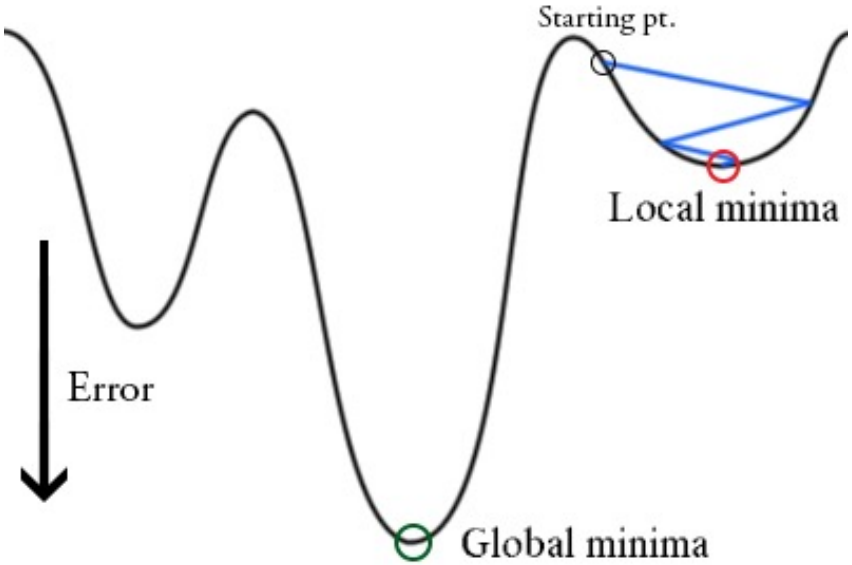


Remember: Local Minima

- For each weight vector, we can calculate the loss of the NN
- The loss surface/landscape can be very irregular: many local minima
- SGD goal: finding a *good* minimum
- Don't even dream about finding the global minimum



Surface with 2 weights



Surface with 1 weight

NN Architecture: Inputs/Outputs

- How many input and output nodes?
  - Usually determined by the problem
  - Number of input nodes equals the number of *input features*
  - Number of outputs
  - One output node for binary classification (true/false)
  - N output nodes for N-class classification; output is $(q_1, q_2, \ldots, q_N)$
    - Example: output (0.1, 0.7, 0.2) means class 2 most probable
  - Appropriate number of outputs for regression
  - Equals dimensionality of data (e.g., no of pixels) for generation
- Traditional thinking (not quite right): best to have as few hidden layers/nodes as possible: better generalisation, less work

NN Architecture: Hidden Layers

- Almost all NNs are organized in layers
- *Features* or *activations*: outputs of individual neurons in a layer (after the nonlinearity)
- How many hidden layers/nodes?
  - Universal approximation Theorem: one hidden layer is always enough
    - But has infinite width
    - We don't know how to train such a system
  - Traditional thinking (not quite right): best to have as few hidden layers/nodes as possible: better generalisation, less work:
    - Make the best guess you can
      - If training is unsuccesful try more hidden nodes
      - If training is successful *perhaps* try fewer hidden nodes
  - Pruning methods eliminate connections or nodes that contribute little; is tedious to implement in practice

Check List for Design

- Data arrangement for network training
  - Batch-wise data arrangement (batch size)
  - Training / validation / test sets

- Network setup
  - System level decisions (LLM, diffusion, basic classifier, ...)
  - Architecture components (MLP, ResNet, Unet, transformer, etc.)
  - Number of input/output nodes
  - How many hidden layers how many nodes in each layer
    - More layers and neurons will train slower
  - Values for the parameters controlling the training process / the optimisers, e.g., learning rate, initial weights, momentum
  - Stopping criteria (validation control) / number of epochs

Summary

- Loss functions
  - Mean squared error
  - Not discussed but common: Mean absolute error
  - Cross entropy
  - Not discussed: loss functions for unsupervised case
    - No known desired output
- We find a local minimum
- Overfitting

- Decisions:
  - NN architecture
  - Batch size
  - Optimiser setting
  - Stopping criteria