COMP6237 Data Mining

Discovering Groups

Part 2: Visualising and embedding data

Jonathon Hare
jsh2@ecs.soton.ac.uk

Introduction

- Visualisation in 2D
    - Principal Component Analysis
    - Self Organising Maps
    - Multidimensional Scaling
    - t-SNE
- Embedding as feature encoding

Problem statement (again!)

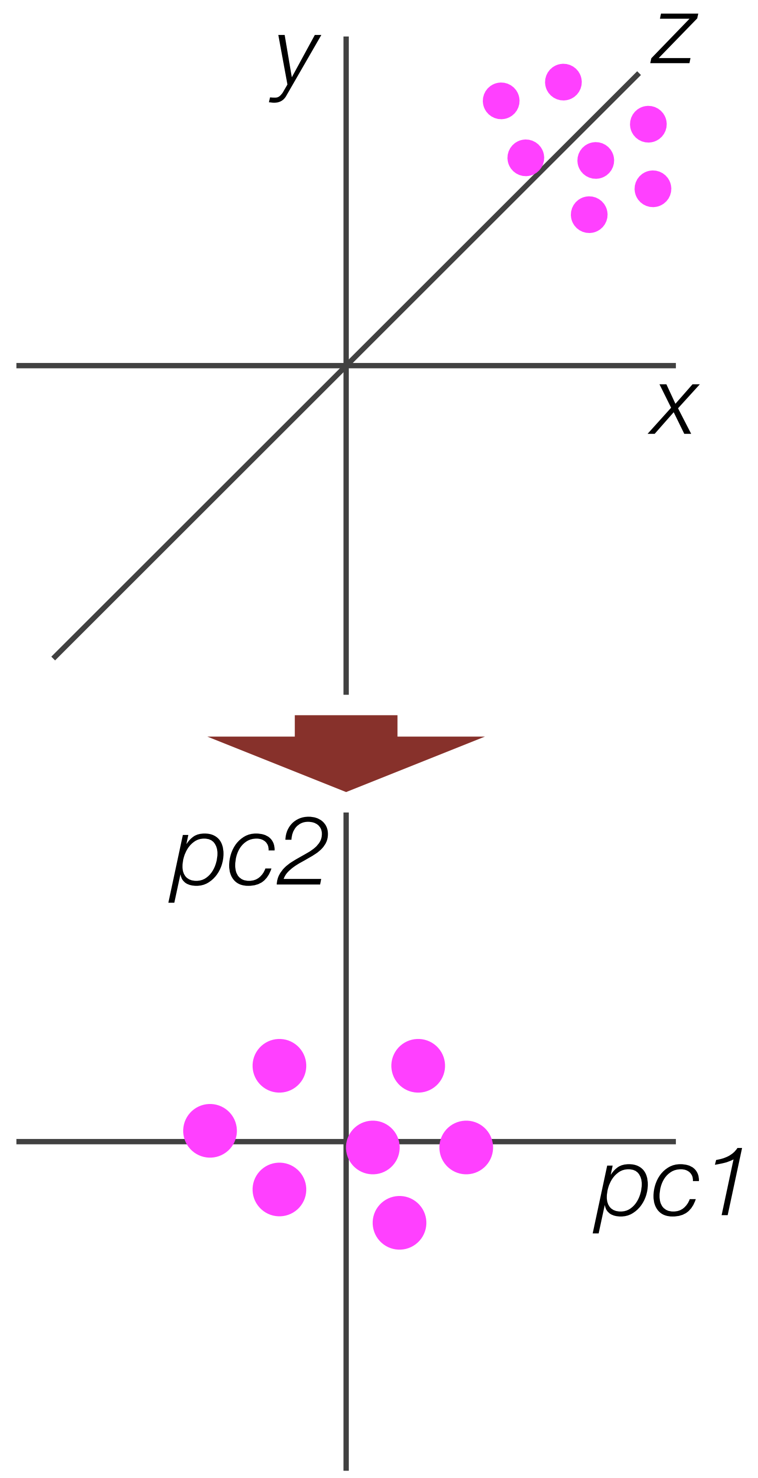
- Understanding large datasets is **hard**
  - especially if the data is represented by high dimensional features
- In order to explore a dataset we might:
  - want to be able to understand which *data items* are *similar* to each other
  - want to be able to understand which *features* are *similar* to each other

Visualising data in two dimensions

- Sometimes we just want to visualise how items of data relate to each other
  - i.e. want to plot them on a 2D surface in such a way that things that are similar are close together (specifically have a small Euclidean distance)

PCA

- We've already looked at how PCA works
    - Can use PCA to project features to a 2D space based on the first and second principal axes



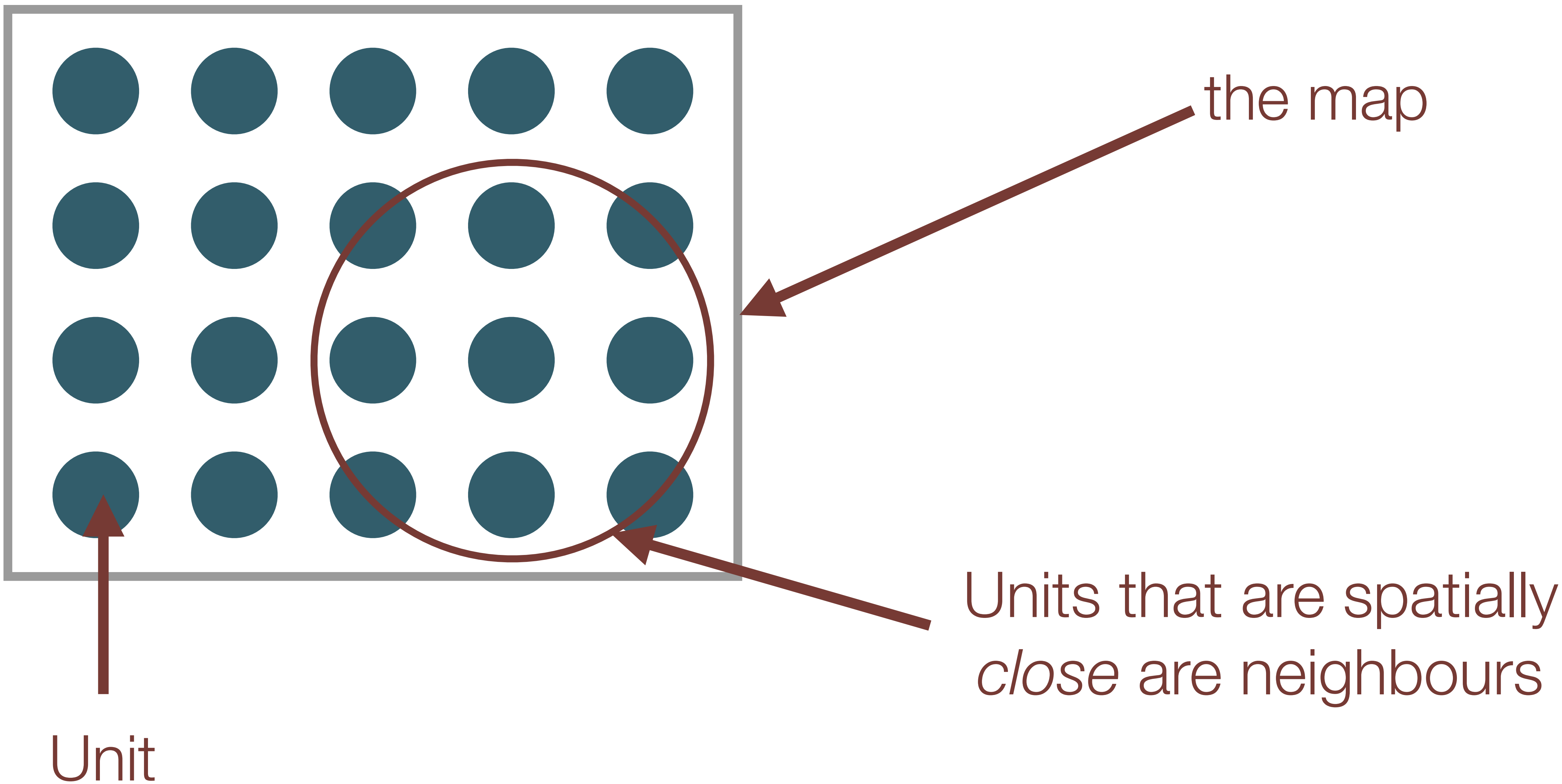
Demo showing PCA on real data

PCA Problems

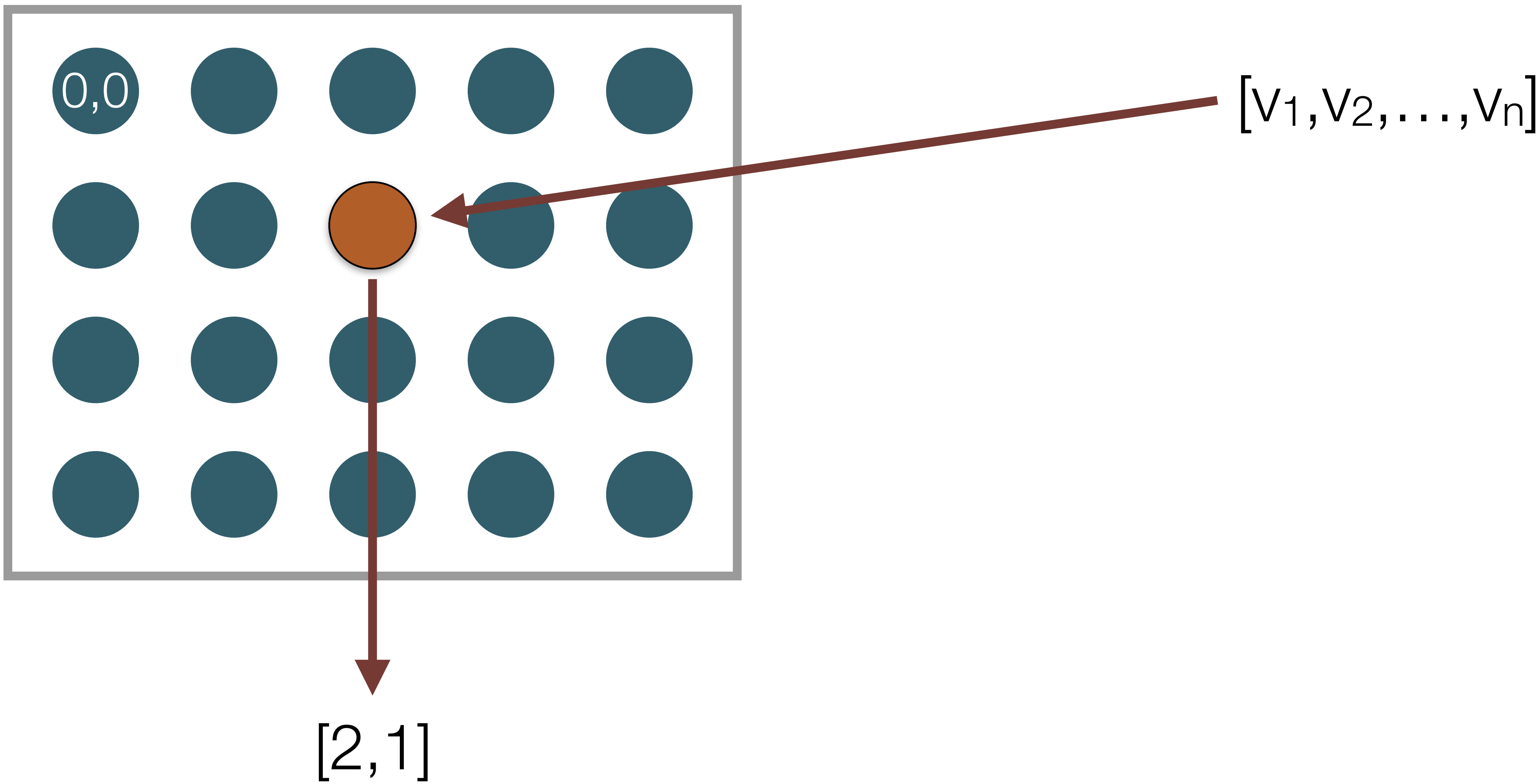
- No control over the distance measure
- Just because axes are oriented along greatest variance, doesn’t mean similar things will appear close together
    - bear in mind PCA is only a rotation of the original space followed by truncation of less significant dimensions

Self Organising Maps

- SOMs invented by Kohonen in the ‘80s
    - A type of neural net
        - Use competitive learning rather than back-prop against an known objective
        - Use a neighbourhood function to preserve the topology of the input features
    - Best not to think to hard about the neural network analogy though!

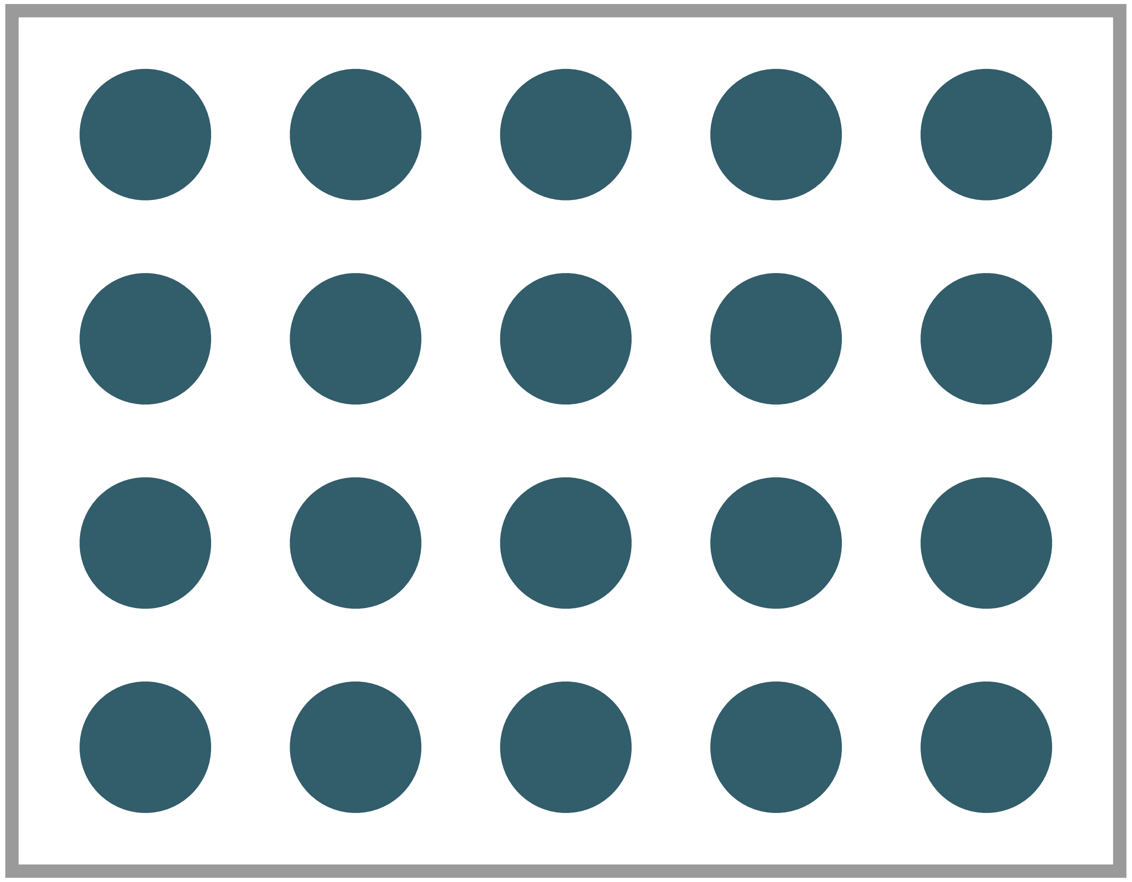


- Each unit has an associated *weight vector*
  - The dimensionality of this vector is equal to the dimensionality of the input feature space
- The location of a unit in the map can be considered to be its (2D) coordinate



In the **projection** phase, the SOM maps high dimensional vectors to a 2D coordinate through the unit that has the closest weight vector (in terms of Euclidean distance)

This is the **Best Matching Unit** (**BMU**)



Before a SOM can be used to project data, it must first be trained - the weights of each unit need to be learned

1. Randomly assign weights to each unit

2. Traverse each input vector in the input data set

    1. Find the BMU by computing the Euclidean distance of the input vector to each unit and picking the unit with the smallest distance

    2. Update the units in the **neighbourhood** of the BMU (including the BMU itself) by pulling them closer to the input vector:
    $W_v(s+1) = W_v(s)+\Theta(u,v,s)\alpha(s)(\mathbf{D}(t)-W_v(s))$

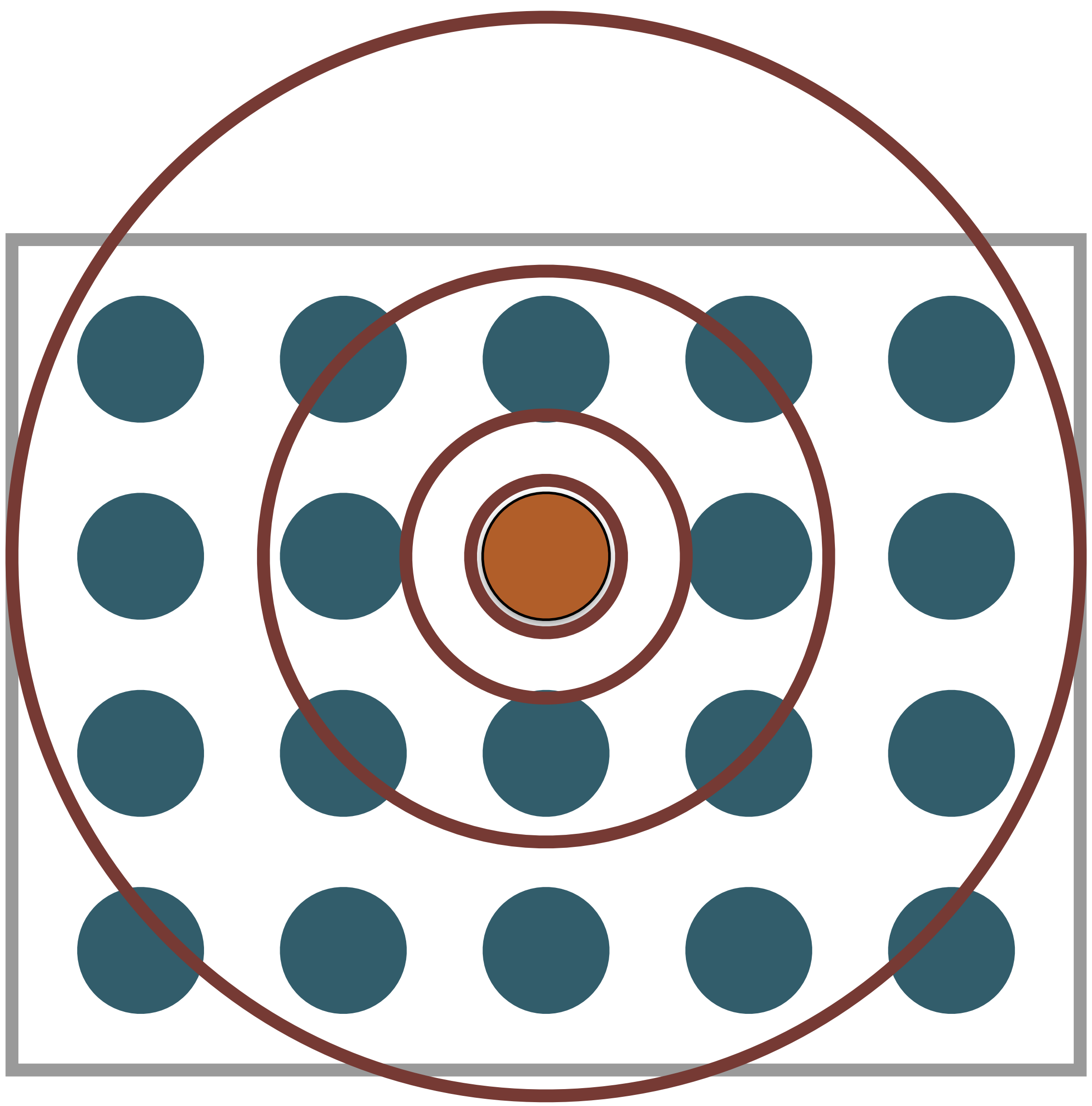
3. Increase s and repeat from step 2 while $s < \lambda$

current iteration

max iterations

$$W_v(s+1) = W_v(s)+\Theta(u,v,s)\alpha(s)(D(t)-W_v(s))$$

Learning rate decay
[gets smaller over time (*s*)]



Neighbourhood weighting function
(often Gaussian)
[gets smaller over time (*s*)]

Demo showing SOM on RGB Colours

Multidimensional Scaling

- Start with data in a high dimensional space and a set of corresponding points in a lower dimensional space
    - attempt to optimise the position of points in lower dimensional space so their Euclidean distances *are like* the distances between the high dimensional points
        - Can use any arbitrary distance measure in the high dimensional space

- Two main categories:
  - **Metric MDS**: tries to match distances
  - **Non-metric MDS**: tries to preserve rankings
- Only requires distances between items as input
  - Unlike PCA and SOM there is no explicit mapping
- Both categories work in the same way - try to minimise a **stress** function
  - Measures goodness of fit between two spaces
  - Can be linear or nonlinear

Stress Functions

- Lots of stress functions:
  - **Least-squares scaling/Kruskal-Shepard scaling**
  - **Shepard-Kruskal non-metric scaling**
  - **Sammon Mapping:**

Low-dimensional Euclidean distance

$$S(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n) = \sum_{i \neq j} \frac{(\delta_{ij} - \|\mathbf{z}_i - \mathbf{z}_j\|)^2}{\delta_{ij}}$$

All combinations of points with the exception of points to themselves

High-dimensional distance

Minimising the Sammon Mapping

- Non-linear… need to use *gradient descent*
  - Starting at an arbitrary point take steps in the direction of the gradient, with size equal to the *gradient* multiplied by a *learning rate,* until convergence:

$$\mathbf{z}_j(k+1) = \mathbf{z}_j(k) - \gamma_k \nabla_{\mathbf{z}_j} S(\mathbf{z}_1(k), \mathbf{z}_2(k), ..., \mathbf{z}_n(k))$$

  where the derivative of the Sammon stress is :

$$\nabla_{\mathbf{z}_j} S(\cdot) = 2 \sum_{i \neq j} \left(\frac{\|\mathbf{z}_i(k) - \mathbf{z}_j(k)\| - \delta_{ij}}{\delta_{ij}} \right) \left(\frac{\mathbf{z}_j(k) - \mathbf{z}_i(k)}{\|\mathbf{z}_i(k) - \mathbf{z}_j(k)\|} \right)$$

Demo showing MDS

SNE
Stochastic Neighbour Embedding

- Similar basis to MDS
- Instead of directly optimising distances we optimise the distribution of the data:
  - We want the distribution of points in the low dimensional space to mirror the distribution in the high dimensional space

SNE: *Source Distribution*

- Define conditional probability that **high-dimensional** x_i would pick x_j as a neighbour if the neighbours were picked in proportion to their probability density under a Gaussian centred at x_i:

$$p_{j|i} = \frac{\exp\left(-\parallel x_i - x_j \parallel^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\parallel x_i - x_k \parallel^2 / 2\sigma_i^2\right)}$$

- The SNE algorithm chooses σ for each datapoint such that smaller σ is chosen for points in dense parts of the space, and larger σ is chosen for points in sparse parts

SNE: *Target Distribution*

- Define conditional probability that **low-dimensional** y_i would pick y_j as a neighbour if the neighbours were picked in proportion to their probability density under a Gaussian centred at y_i:

$$q_{j|i} = \frac{\exp\left(- \| y_i - y_j \|^2\right)}{\sum_{k \neq i} \exp\left(- \| y_i - y_k \|^2\right)}$$

- (We assume variance of all Gaussians is 1/sqrt(2) in this space)

SNE: *Cost Function to minimise*

- Kullback-Leibler Divergence:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log \frac{P(i)}{Q(i)}$$

  - *Measure of how one distribution diverges from another*

- Cost for SNE is the KL Divergence summed over all data points:

$$C = \sum_i \sum_j p_{i|j}\,\log \frac{p_{j|i}}{q_{j|i}}$$

SNE
Stochastic Neighbour Embedding

- The cost C can be minimised using *gradient descent*
- …but…
  - It’s **difficult** to optimise
  - It leads to “crowded” visualisations in which things clump together in the centre (this is also true of Sammon mapping)

t-SNE
t-distributed Stochastic Neighbour Embedding

- Modifies the cost function to overcome SNE weaknesses:
  - Use a symmetric version of the cost
    - Simplified gradients (faster to compute)
  - Use Student’s t-distribution in the lower dimension space instead of a Gaussian
    - Alleviate crowding

t-SNE

t-distributed Stochastic Neighbour Embedding

- Modifies the cost function to overcome SNE weaknesses:
  - Use a symmetric version of the cost
    $$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$
    - Simplified gradients (faster to compute)
  - Use Student's t-distribution in the lower dimension space instead of a Gaussian
    - Alleviate crowding
    $$q_{ij} = \frac{f(\| x_i - x_j \|)}{\sum_{k \neq i} f(\| x_i - x_k \|)} \quad \text{with} \quad f(z) = \frac{1}{1 + z^2}$$

Demo showing t-SNE

Other dimensionality reduction techniques

- We've only scratched the surface
  - Lots of other techniques and variants
    - ISOMAP
    - Locally Linear Embedding (LLE)
    - Principal curves
    - Autoencoders
    - …

Encoding features with embeddings

- Rather than using a technique to project high dimensional data in a 2D or 3D space, could we target a *medium* dimensionality which captures the key features?

- Yes! We call this an “embedding”.
  - Embeddings have many uses
    - We’ll start with “word embeddings” today & we’ll see other uses in later lectures

One Hot Encoding

- Let's assume we want to create a vector representation for all the words in the dictionary.
  - Obvious place to start: a Bag of Words representation with a single word:
    - ```
      a ->            [1,0,0,0,0,0,…,0]
      abbot ->        [0,1,0,0,0,0,…,0]
      abbreviate ->   [0,0,1,0,0,0,…,0]
      ```
    - *Known as a "one-hot encoding"*
      ```

One Hot Encoding Problems

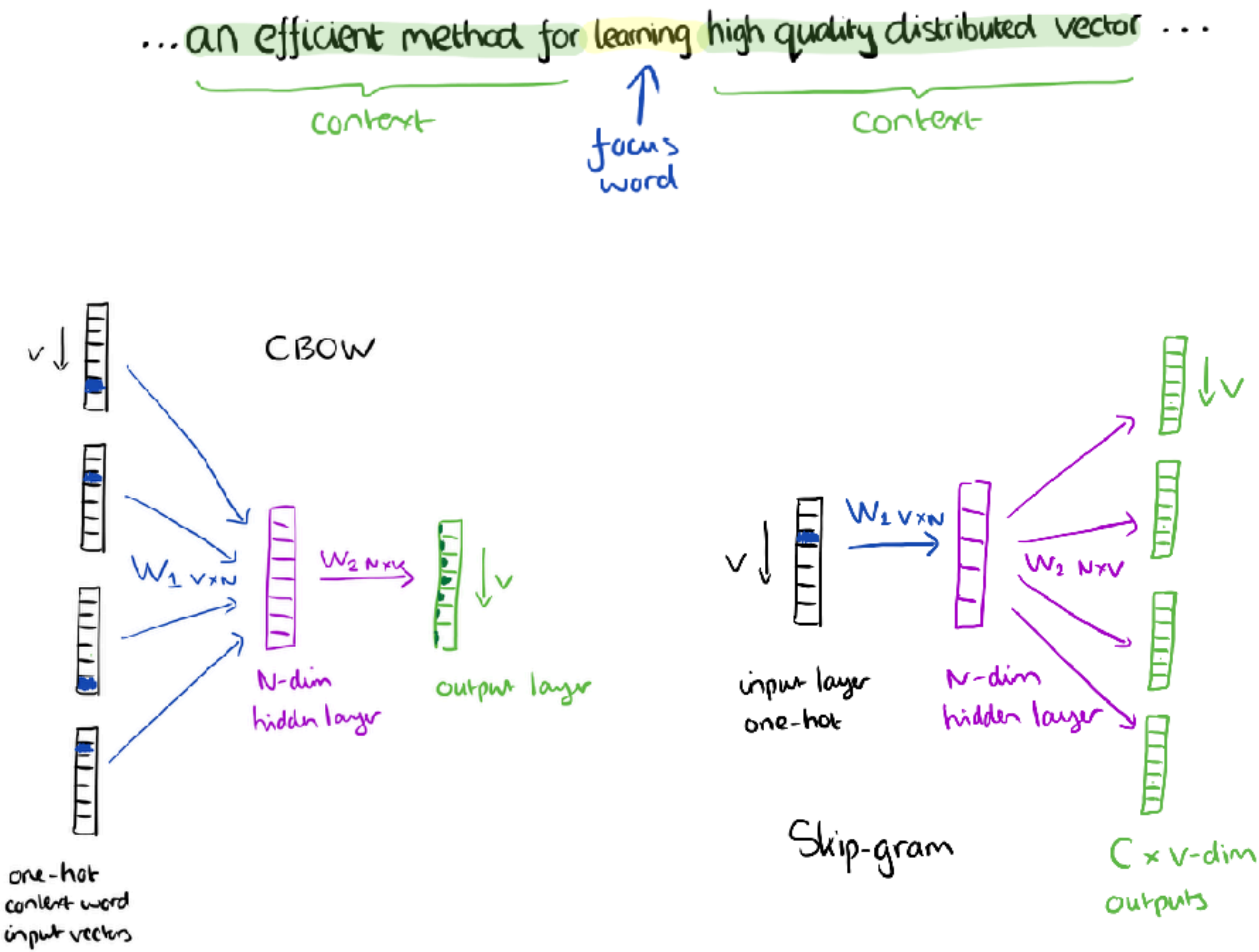
- All vectors are orthogonal
  - Is this a problem?
    - Implies independence of words…
    - But human languages have synonymous terms:
      e.g. *cat, kitten, tabby, tomcat, tom, queen, mouser*
- The vectors are very long (albeit sparse)

Word Embeddings 1

- Can we build a better vector representation of words?
  - Lower dimensionality (but dense)
  - Capturing synonymous words
  - What about capturing algebraic semantics?
    - word2vec(“Brother”) - word2vec(“Man”) + word2vec(“Woman”) = word2vec(“Sister”)

Word Embeddings 2

- Many models of mapping words to vectors have been proposed.
  - A pair of commonly used models is known as “word2vec” and was introduced by Mikolov *et al.* at Google
    - They’re both shallow two-layer neural nets, but trained on *lots* of data
    - Ironically, although the paper introducing the models has 5500 citations, it was never officially published after being rejected (and heavily slated by the reviewers) of ICLR 2013!
  - Another popular model is GloVe “Global Vectors for Word Representation” by Pennington *et al.*
  - All these models have all the features from the previous slides!
  - *Note that practically speaking, you don’t have to train the models - you can just download a pretrained variant*



Images from https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Word2Vec Demo

Summary

- Dimensionality reduction and visualisation is of key importance to understanding data
- Embedding gives us potentially more powerful ways of computing vector representations
- Lots of different approaches - we've barely scratched the surface
  - The visualisation approaches we've seen will be useful for the individual coursework