INFORMATION RETRIEVAL

Luca Manzoni
lmanzoni@units.it

Lecture 7-bis

NEURAL NETWORKS IN IR

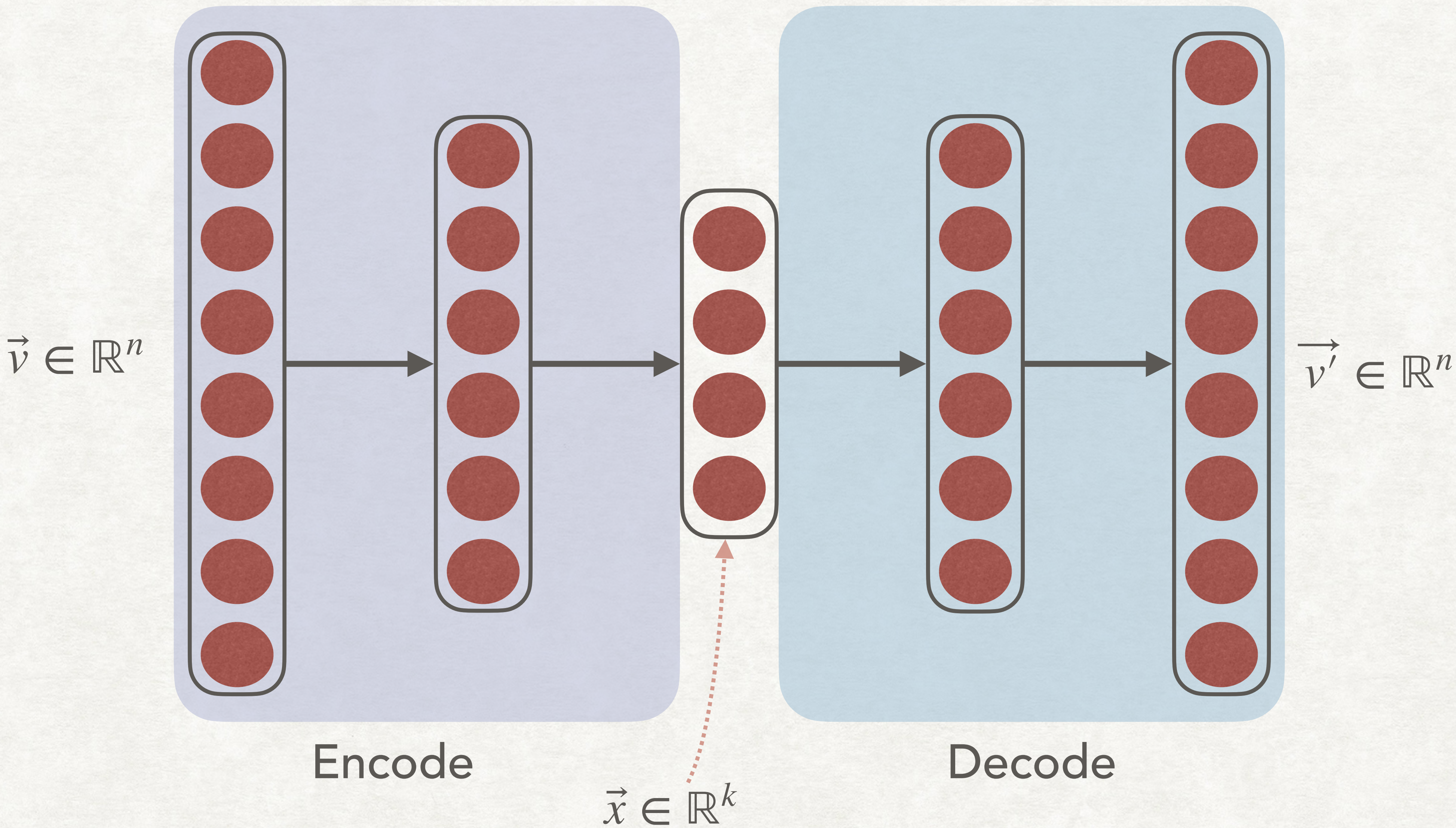
NEURAL NETWORKS AND IR

BECAUSE NEURAL NETWORKS ARE EVERYWHERE NOW

- Suggested reading:
  Bhaskar Mitra, Nick Craswell
  An Introduction to Neural Information Retrieval
  Foundations and Trends in Information Retrieval, 2018
- We assume some knowledge of neural networks
- We will see (briefly) some of the possible applications of neural networks in IR:
  - Learning of term representation
  - Recommender systems using NN

AUTOENCODER

AND DIMENSIONALITY REDUCTION



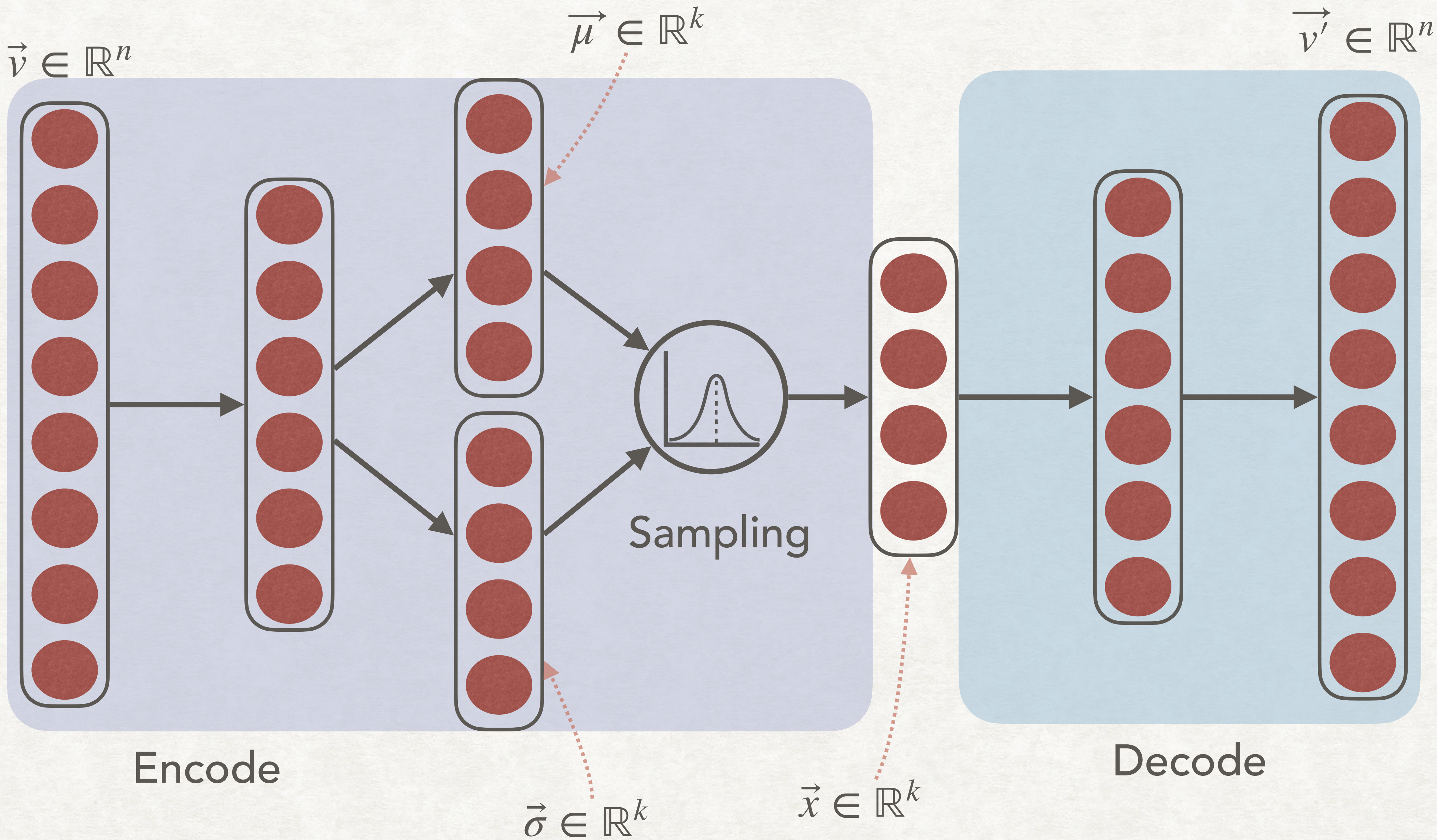
AUTOENCODER

AND DIMENSIONALITY REDUCTION

- Autoencoders are based on the *information bottleneck method* and typically have a “hourglass” shape.
- The input is a vector $\vec{v} \in \mathbb{R}^n$, the output is also a vector $\overrightarrow{v'} \in \mathbb{R}^n$.
- We want the output vector to be the same as the input vector (i.e., the network learns the identity function).
- The loss function is usually $\mathscr{L}_{\text{autoencoder}} = \|\vec{v} - \overrightarrow{v'}\|^2$.
- The “bottleneck” is a vector $\vec{x} \in \mathbb{R}^k$, with $k \ll n$ which represents an encoding of $\vec{v}$ in a lower dimensional space.

VARIATIONAL AUTOENCODER

A "SMOOTHER" AUTOENCODER



VARIATIONAL AUTOENCODER

A "SMOOTHER" AUTOENCODER

- Similar to an autoencoder, but the encoding part of the network generates two vectors:
  - $\overrightarrow{\mu} = (\mu_1, \mu_2, \ldots, \mu_k)$ of the means
  - $\vec{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_k)$ of the standard deviations
- The vector $\vec{x}$ is obtained by sampling k normal distributions with mean a variance obtained by $\overrightarrow{\mu}$ and $\vec{\sigma}$: $x_i \sim N(\mu_i, \sigma_i^2)$.
- This should allow to learn a "smoother" latent space.

VARIATIONAL AUTOENCODER

A "SMOOTHER" AUTOENCODER

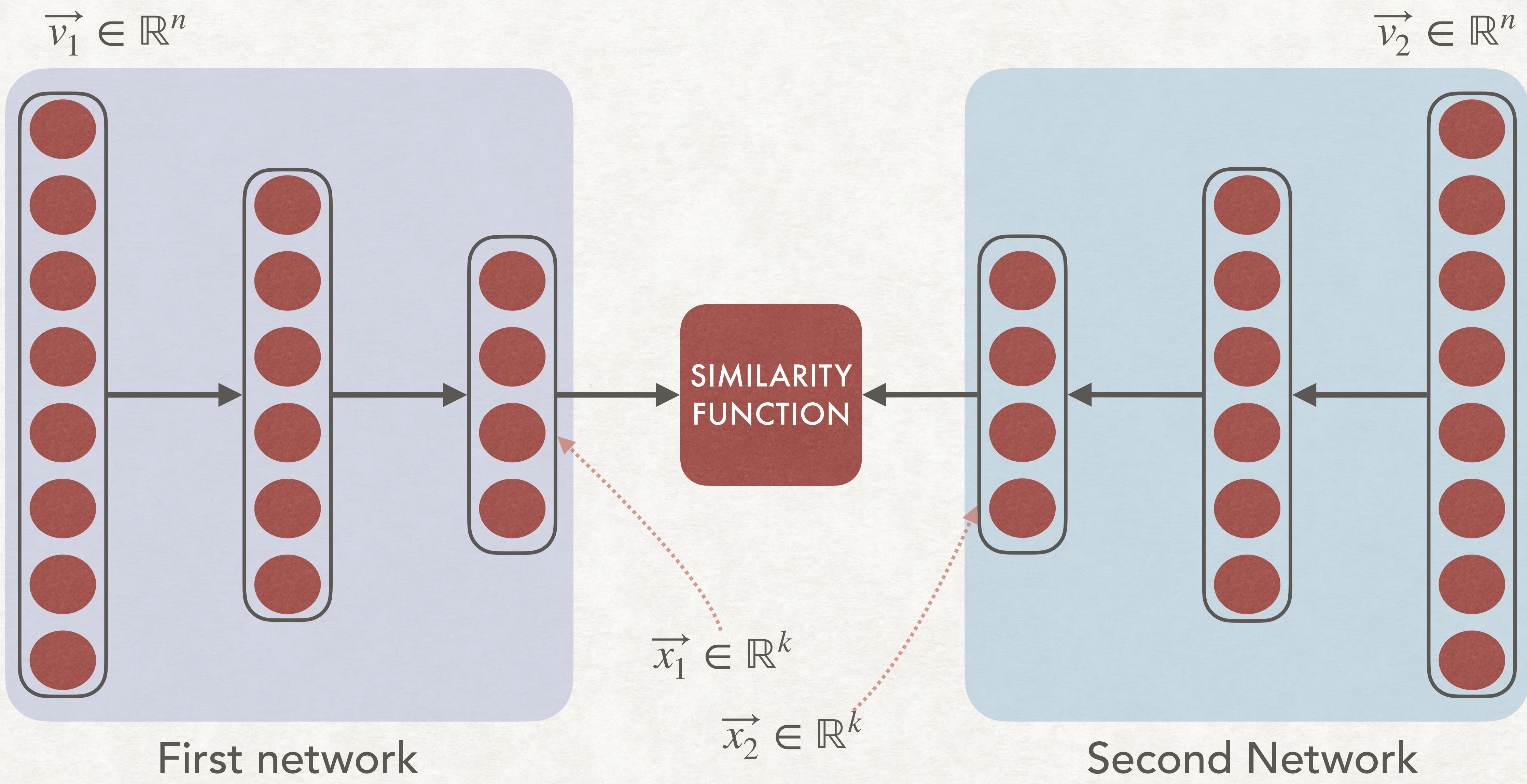
- The loss function should try to penalise setting the standard deviations too close to zero.
- This means that there are two components in the loss function:
  - Reconstruction error: $\mathscr{L}_{\text{reconstruction}} = \|\vec{v} - \overrightarrow{v'}\|^2$
  - Kullback–Leibler divergence with respect to a unit gaussian:

$$\mathscr{L}_{\text{KL-divergence}} = \sum_{i=1}^{k} \sigma_i^2 + \mu_i^2 - log(\sigma_i) + 1$$

- The loss function is then: $\mathscr{L}_{\text{VAE}} = \mathscr{L}_{\text{reconstruction}} + \mathscr{L}_{\text{KL-divergence}}$

SIAMESE NETWORKS

A REPRESENTATION FOR COMPUTING SIMILARITY



SIAMESE NETWORKS

A REPRESENTATION FOR COMPUTING SIMILARITY

- Autoencoders and VAE use a latent space representation that is useful for reconstructing the original input…
- …but sometimes we are interested in a latent space representation that is useful for computing similarities.
- Two networks (models) maps two inputs $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$ into the same latent space, obtaining $\overrightarrow{x_1}$ and $\overrightarrow{x_2}$.
- We compute the similarity between $\overrightarrow{x_1}$ and $\overrightarrow{x_2}$ in the latent space using a classical similarity measure, like cosine similarity.

SIAMESE NETWORKS

A REPRESENTATION FOR COMPUTING SIMILARITY

- A possible way of learning for siamese networks is to consider each input sample as a triple.
- We obtain three outputs: $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, and $\vec{y}$.
- We know that $\overrightarrow{x_1}$ should be more similar to $\vec{y}$ that $\overrightarrow{x_2}$.
- We define the loss function to represent this relation:
- $\mathscr{L}_{\text{siamese}} = \log\left(1 + e^{-\gamma(sim(\vec{y},\overrightarrow{x_1}) - sim(\vec{y},\overrightarrow{x_2}))}\right)$
  where γ is a parameter, usually set to 10.

DOCUMENT AUTOENCODER

LEARNING A LATENT REPRESENTATION

- Relevant paper:
  Salakhutdinov, R. and G. Hinton. 2009. "*Semantic hashing*".
  International Journal of Approximate Reasoning. 50(7): 969–978.

- Idea: use auto-encoders to learn a latent-space representation of a document.

- A network is trained using a one-hot encoding of the 2000 most common terms (without stopwords) to produce a binary vector encoding of the documents.

DOCUMENT AUTOENCODER

LEARNING A LATENT REPRESENTATION

- Similar documents with the same hash vector can be efficiently retrieved.
- The auto encoder acts as an hash function where similar documents ends up in the same “bin”.
- In other works also variational auto encoders were used.
- Main problem: a vocabulary of a few thousand words might be too small in practical problems.
- Possible solution: use of trigraphs instead of words as input.

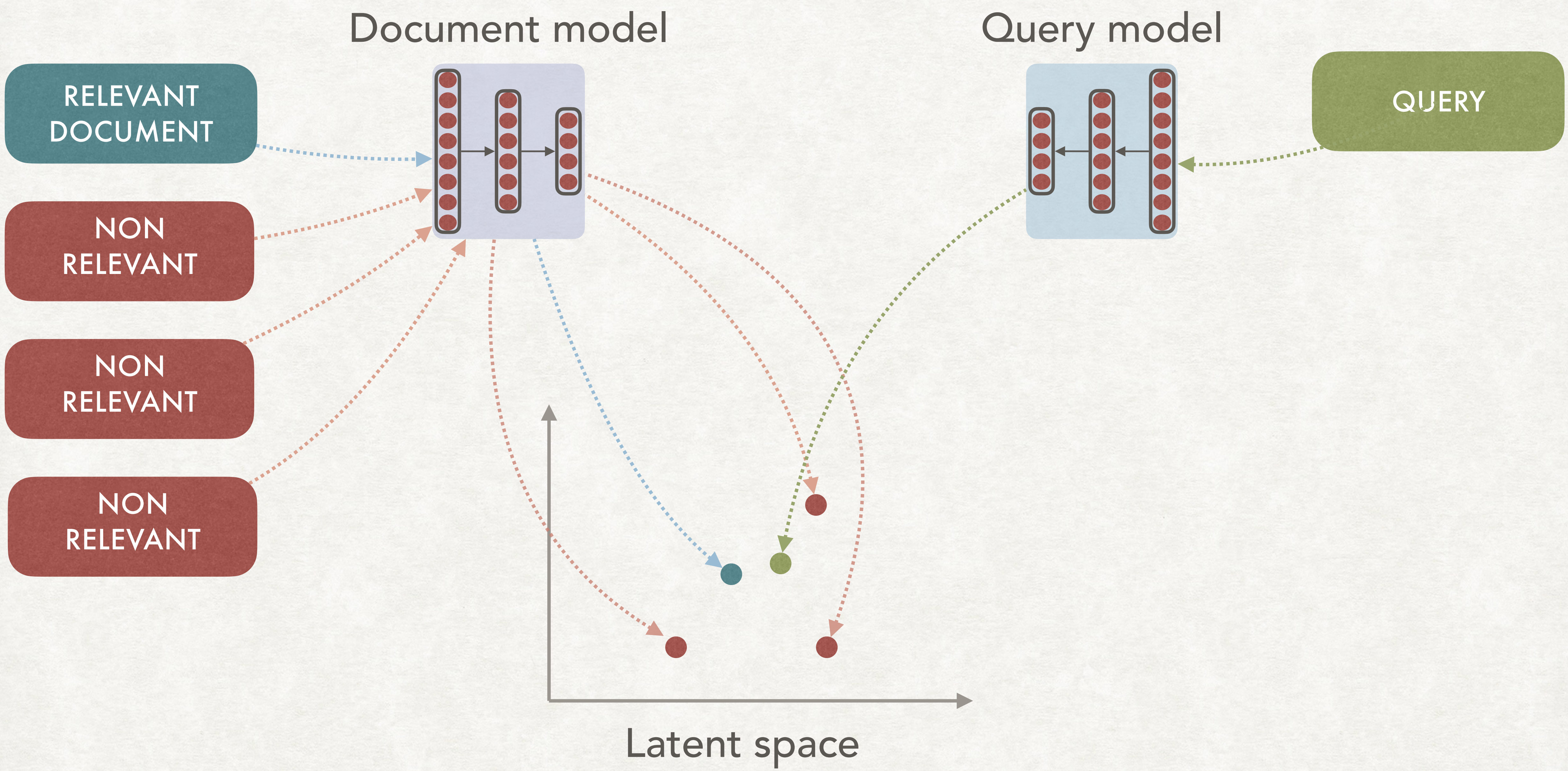
SIAMESE NETWORKS

LEARNING BY DOCUMENTS AND QUERIES

- One approach is to learn a representation using both documents and queries at the same time.
- An approach using siamese networks is the *Deep Semantic Similarity Model* (DSSM).
- Relevant paper:
  Huang, P.-S., X. He, J. Gao, L. Deng, A. Acero, and L. Heck. 2013. "Learning deep structured semantic models for web search using clickthrough data". In: Proc. CIKM. ACM. 2333–2338.
- Two models, one for the query and one for the documents.

SIAMESE NETWORKS

LEARNING BY DOCUMENTS AND QUERIES



SIAMESE NETWORKS

LEARNING BY DOCUMENTS AND QUERIES

- The document titles and the queries are represented as a collection of trigraphs.
- Each sample consists of a query $\vec{q}$, a relevant document $\overrightarrow{d^+}$ and a set of non-relevant document D^- randomly sampled from the full collection.
- The cosine similarity was used as the similarity measure.
- The loss function used was:

$$\mathscr{L}_{\text{dssm}}(\vec{q}, \overrightarrow{d^+}, D^-) = -\log\left(\frac{e^{\gamma\cos(\overrightarrow{d^+}, \vec{q})}}{\sum_{\vec{d} \in D^- \cup \{\overrightarrow{d^+}\}} e^{\gamma \cos(\vec{d}, \vec{q})}}\right)$$

LEXICAL AND SEMANTIC MATCHING

AND THE PROBLEM WITH RARE TERMS

- Embeddings into a latent space as the ones produced by NN have one problem: they tend to produce poor embeddings for rare terms.
- For rare terms a "classical" lexical matching is more effective.
- But for other queries, looking at the semantics via the embedding is more effective (the documents do not contain the same terms as the query).
- In general, lexical and semantic matching tends to perform well on different kinds of queries.

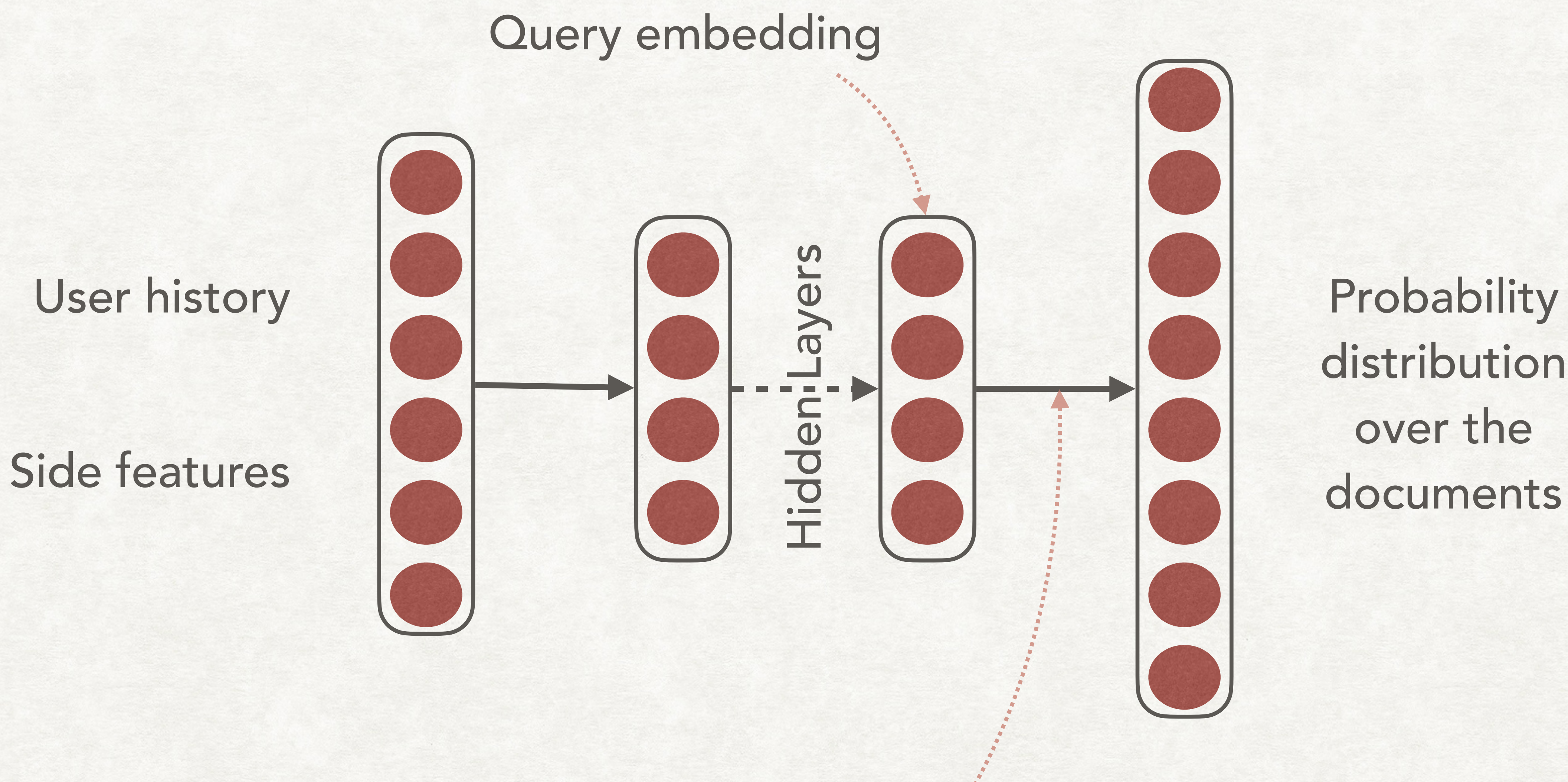
DNN FOR RECOMMENDER SYSTEMS

AN EXAMPLE

- It is possible to use a DNN to build a recommender system to improve with respect to matrix factorisation:
  - Input: a vector $\vec{x}$ representing the user query. It can contain sparse features (e.g., watch history, liked items) and dense features (e.g., time of the last interaction with the system).
  - Output $\hat{p}$ is a probability distribution across all documents in the corpus representing the probability that the user will like/be interested/watch them.
    This can be obtained using a softmax activation in the last layer.

DNN FOR RECOMMENDER SYSTEMS

AN EXAMPLE



The weights of this layer (the softmax layer) forms the item embeddings

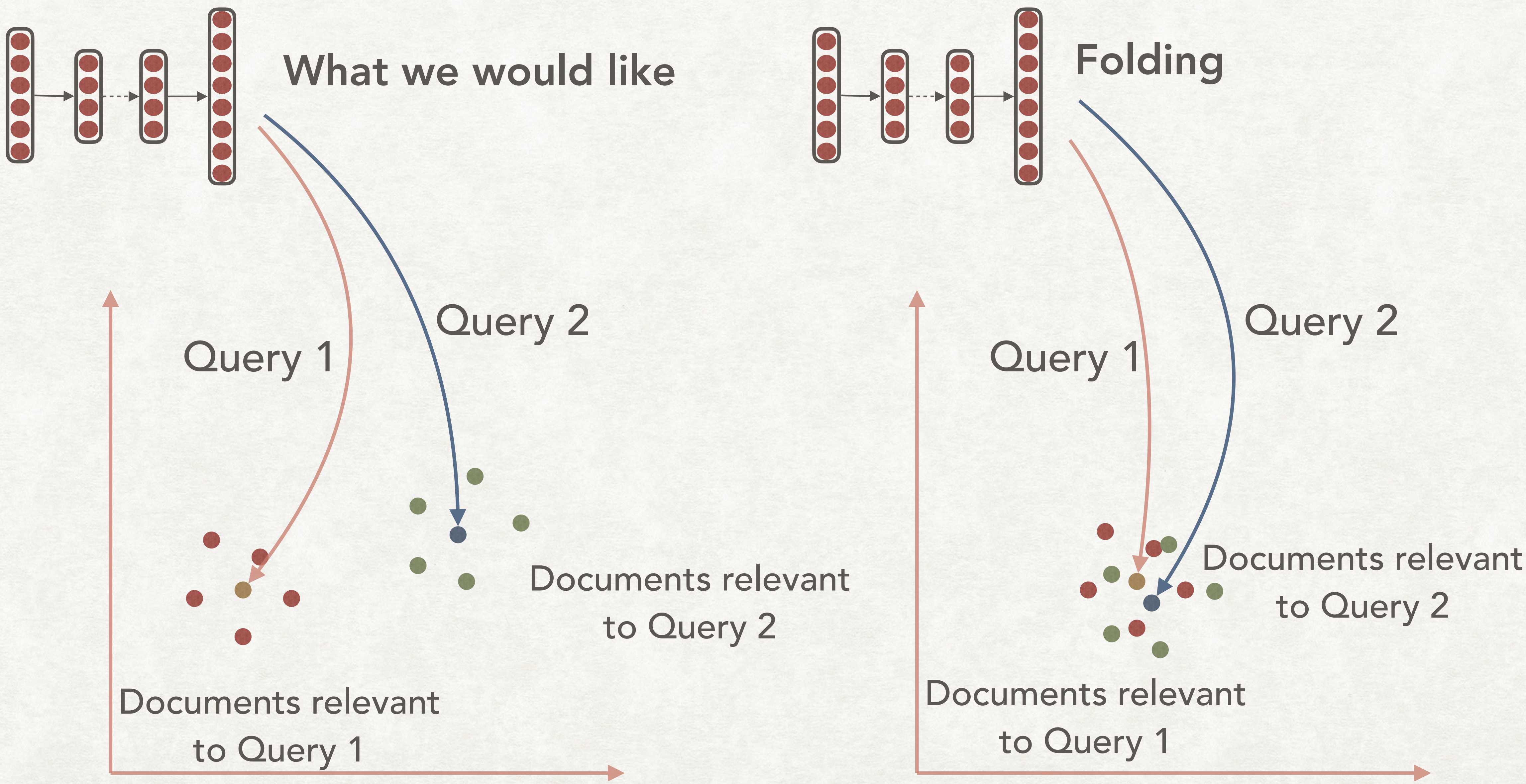
DNN FOR RECOMMENDER SYSTEMS

AN EXAMPLE

- How to compute the loss function?
- We might want to consider a function of the difference between $\hat{p}$ (the predicted distribution) and p (the real one)…
- Except that we do not know the entirety of p.
- We can try to compute the gradient only for the positive item of p (the one that the user liked)…
- …but we can have the problem of *folding*.

DNN FOR RECOMMENDER SYSTEMS

AN EXAMPLE



DNN FOR RECOMMENDER SYSTEMS

AN EXAMPLE

- We use *negative sampling.*
- Instead of learning only from positive example we sample a set of irrelevant documents as negative examples.
  We can do it in two ways:
  - Uniform sampling
  - Higher probability of being sampled to items with a large output value. They contribute more to the gradient.

DNN FOR RECOMMENDER SYSTEMS

ADVANTAGES AND DISADVANTAGES

- DNN can easily incorporate additional features for personalisation.
- DNN can adapt to new queries.
- DNN are more difficult to scale to handle a very large corpus.
- WALS is less prone to folding than DNN.
- The item embeddings (weights of the last layer) can be stored, but the query embedding (output of all layers but the last) must be re-computed every time.