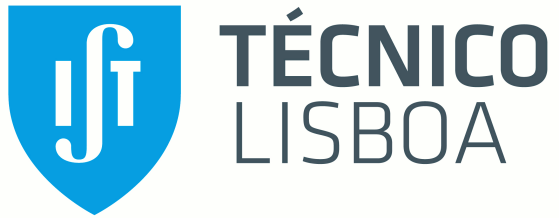
Lecture 7: Representation Learning

André Martins, Francisco Melo, Mário Figueiredo



Deep Learning Course, Fall 2021

Announcements

Deadline for Homework #1 is this **Wednesday end of day**!

- Please submit your solutions and code in Fenix.
- No late days allowed!!
- Solutions will be posted the day after.

Homework #2 will be posted this Wednesday!

- Deadline Jan 31.
- Start early!!

Today's Roadmap

Today's lecture is about:

- Representation learning.
- Principal component analysis (PCA) and auto-encoders.
- Denoising auto-encoders.
- Distributed representations.
- Word embeddings and negative sampling.
- Multilingual and contextual word embeddings.

Outline

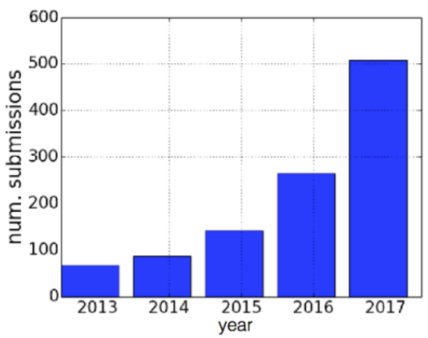
Representations

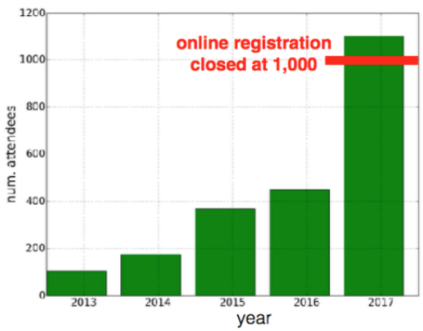
- A key feature of NNs is their ability to learn representations $\phi(\boldsymbol{x})$
- Standard linear models require manually engineered features $\phi(\boldsymbol{x})$
- Representations are useful for several reasons:
  - (i) They can make our models more expressive and more accurate
  - (ii) They may allow transferring representations from one task to another
- We talked about (i) when discussing the multi-layer perceptron
- In this lecture, we'll focus on (ii)

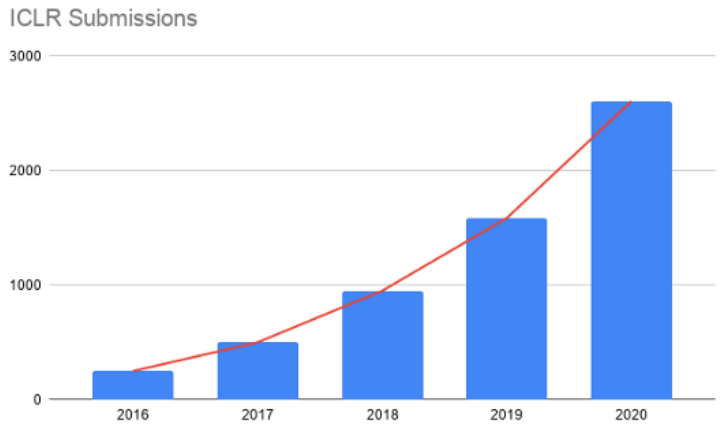
Representation Learning

This is becoming a extremely popular topic!

Number of submissions to the "International Conference on Learning Representations" (ICLR):







Representation learning almost became of synonym of deep learning

Outline

Hierarchical Compositionality

Key Idea: deep(er) NNs learn **coarse-to-fine** representation layers.

Vision:

- pixels $\rightarrow$ edges $\rightarrow$ textons $\rightarrow$ motifs $\rightarrow$ parts $\rightarrow$ objects $\rightarrow$ scenes

Speech:

- audio samples $\rightarrow$ spectral bands $\rightarrow$ formants $\rightarrow$ motifs $\rightarrow$ phonemes $\rightarrow$ words

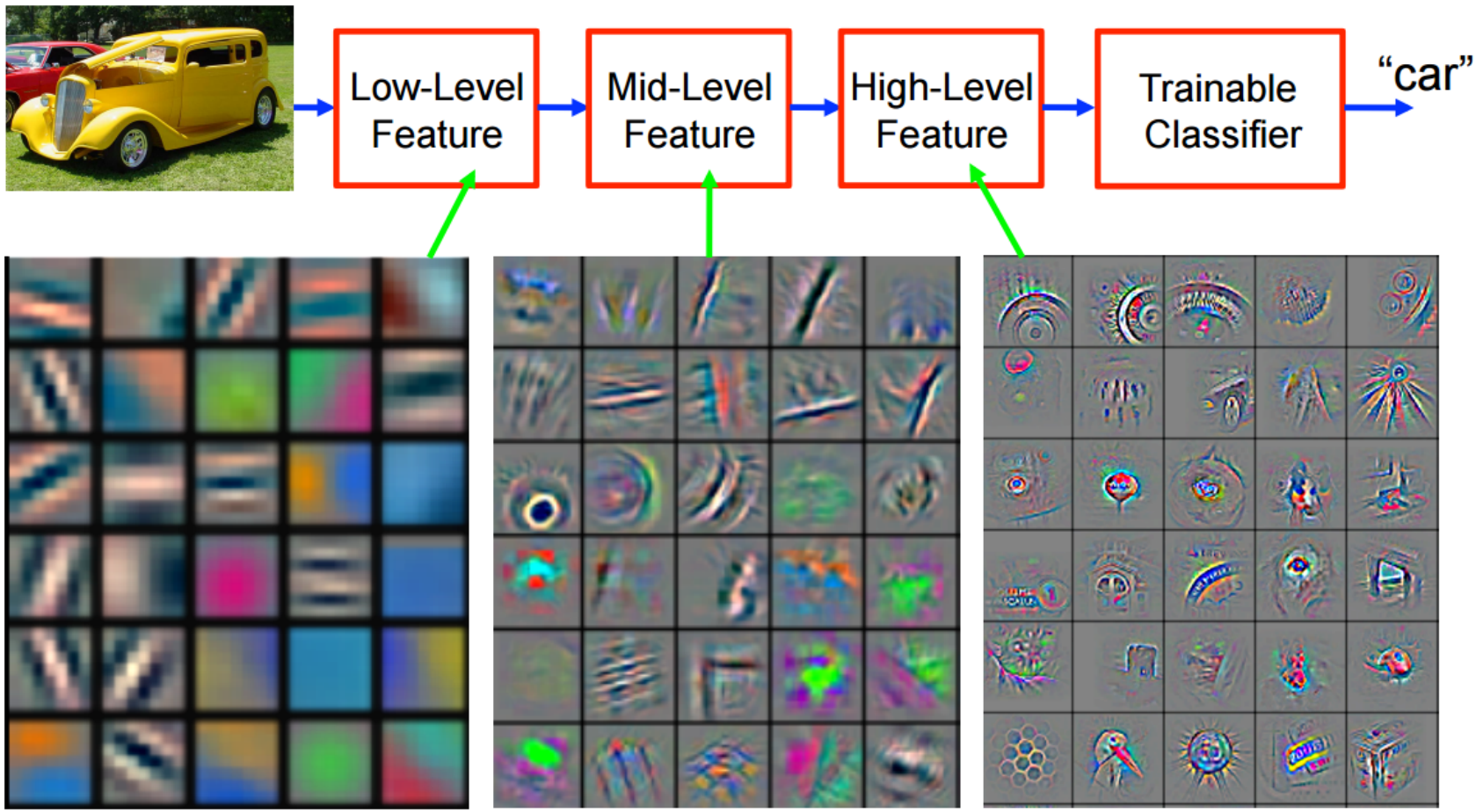
Text:

- characters $\rightarrow$ words $\rightarrow$ phrases $\rightarrow$ sentences $\rightarrow$ stories

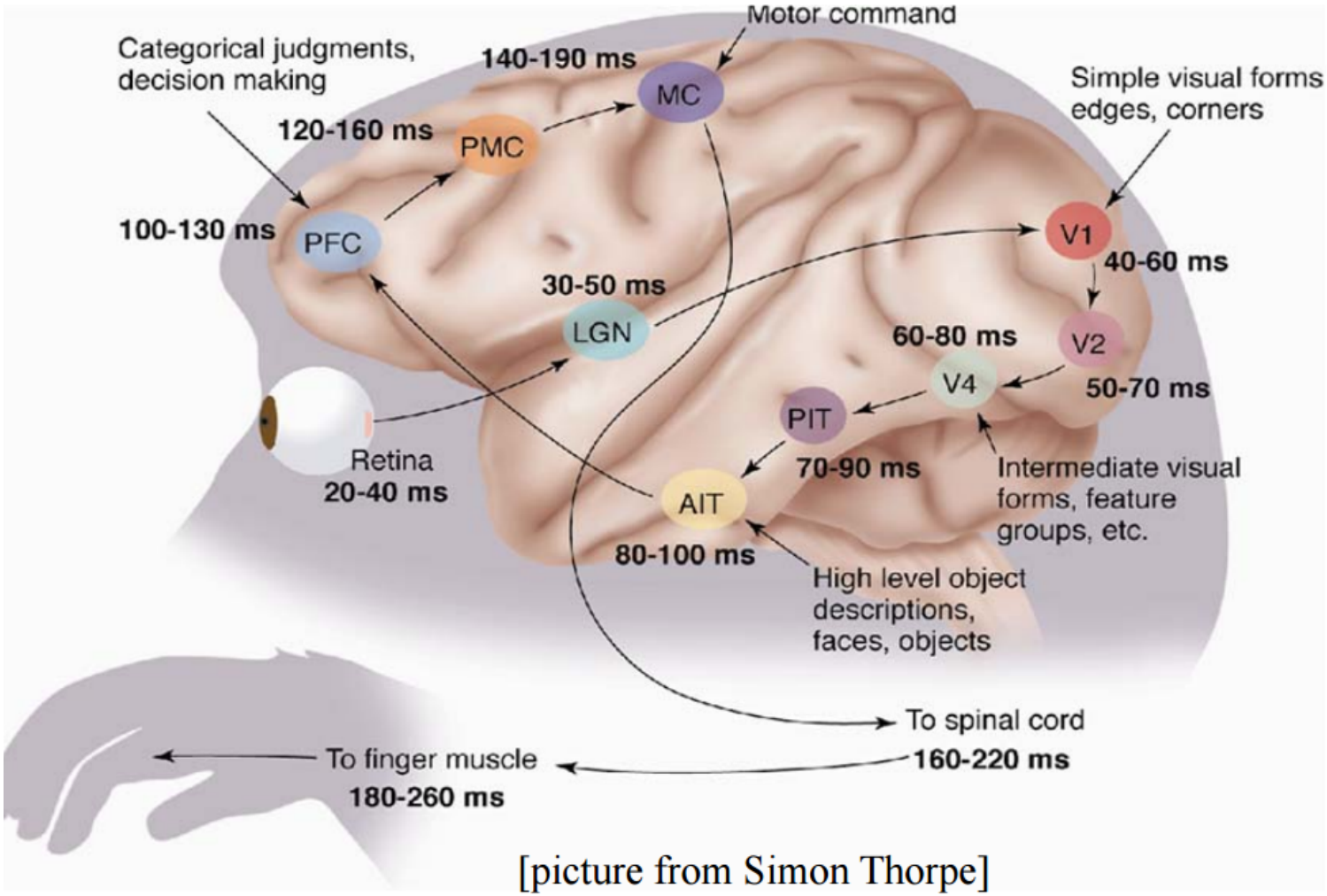
(Slide inspired by Marc'Aurelio Ranzato and Yann LeCun)

Hierarchical Compositionality

Feature visualization of convolutional NNs trained on ImageNet (Zeiler and Fergus, 2013):



The Mammalian Visual Cortex is Hierarchical



(Slide inspired by Marc'Aurelio Ranzato and Yann LeCun)

What's in Each Layer

- Bottom level layers (closer to inputs) tend to learn **low-level representations** (corners, edges)

- Upper level layers (farther away from inputs) learn **more abstract representations** (shapes, forms, objects)

This holds for images, text, etc.

Outline

Distributed Representations (Hinton, 1984)

This is a central concept in neural networks.

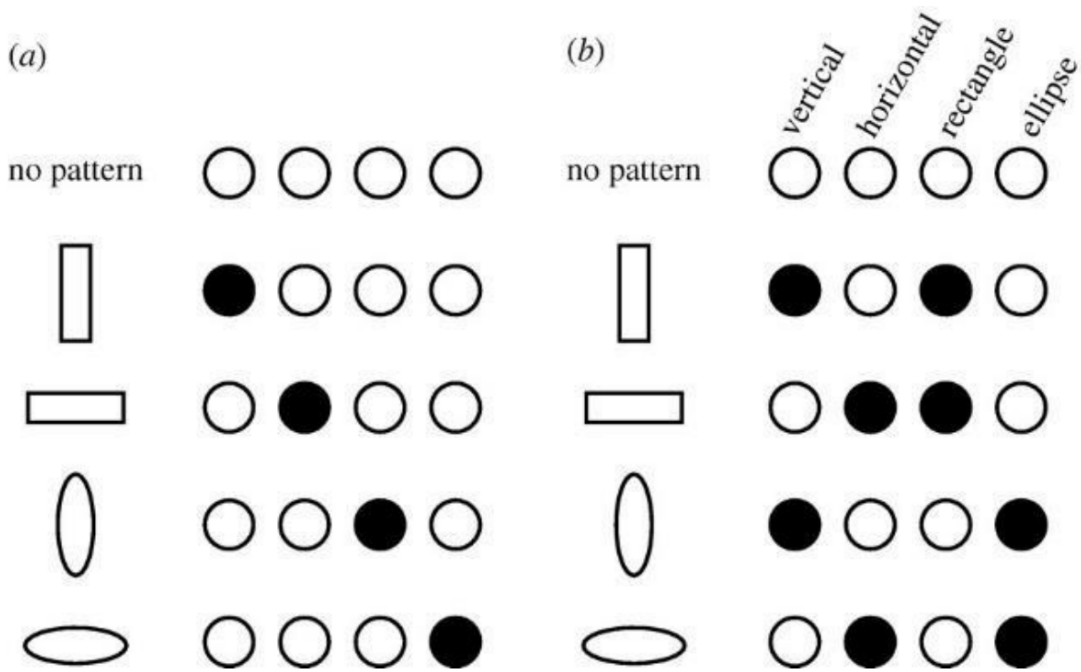
Key questions:

- How can a NN so effectively represent objects, if it has only a few hidden units (i.e. much fewer than possible objects)?
- What is each hidden unit actually representing?
- How can a NN generalize to objects that is has not seen before?

Local vs Distributed Representations

Consider two alternative representations:

- Local (one-hot) representations (one dimension per object)
- Distributed representations (one dimension per property)



(Slide inspired by Moontae Lee and Dhruv Batra)

Distributed Representations

Key idea: no single neuron "encodes" everything; groups of neurons (e.g. in the same hidden layer) work together!

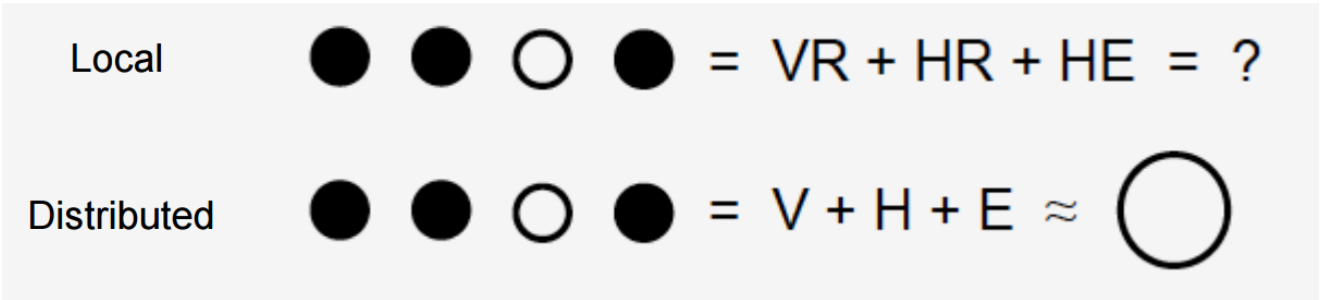
Distributed Representations

Key idea: no single neuron "encodes" everything; groups of neurons (e.g. in the same hidden layer) work together!

cf. the grandmother cell

The Power of Distributed Representations

- Distributed representations are more compact (there can be $O(\exp N)$ objects combining N properties)
- They are also more powerful, as they can generalize to unseen objects in a meaningful way:



(Slide inspired by Moontae Lee and Dhruv Batra)

The Power of Distributed Representations

- For this to work, hidden units should capture diverse properties of objects (not all capturing the same property)
- Usually ensured by random initialization of the weights
- Initializing all the units to the same weights, we would never break the symmetry!
- Side note: a NN computes the same function if we permute the hidden units within the same layer (order doesn't matter, only diversity)

Next: how to learn useful object representations from raw inputs (no labels)?

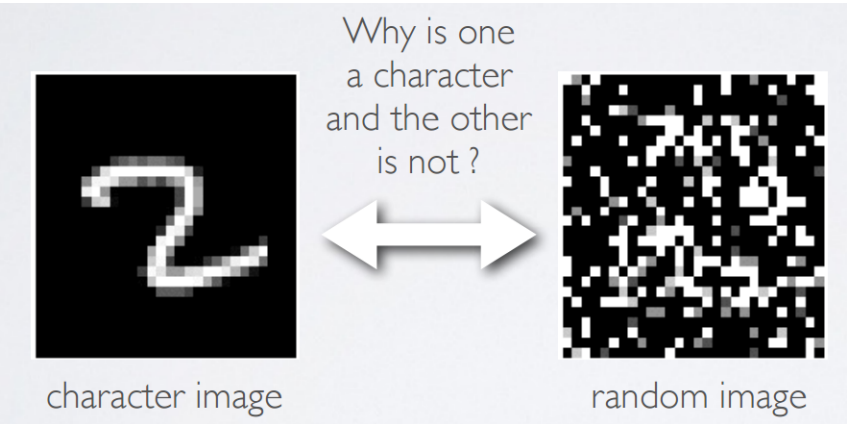
Example: Unsupervised Pre-Training

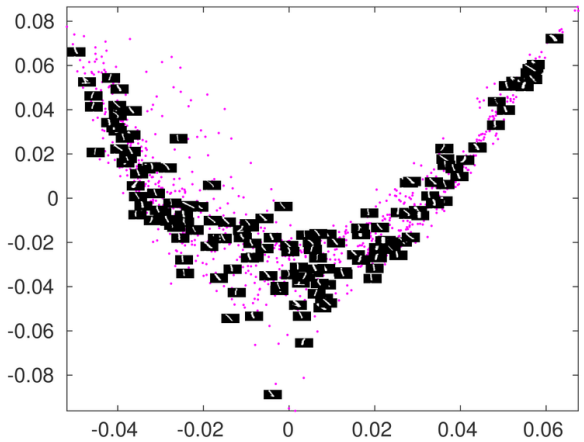
- Training deep NNs (with many hidden layers) can be challenging
- This has been a major difficulty with NNs for a long time
- Initialize hidden layers using unsupervised learning (Erhan et al., 2010):
  - Force network to represent latent structure of input distribution
  - Encourage hidden layers to encode that structure
  - This can be done with an auto-encoder!

Outline

Data Manifold

Key idea: learn the manifold where the input objects live





(Image credit: Hugo Larochelle)

Learn representations that encode well points in that manifold

Auto-Encoders

Auto-encoder: feed-forward NN trained to reproduce its input at the output

Encoder:

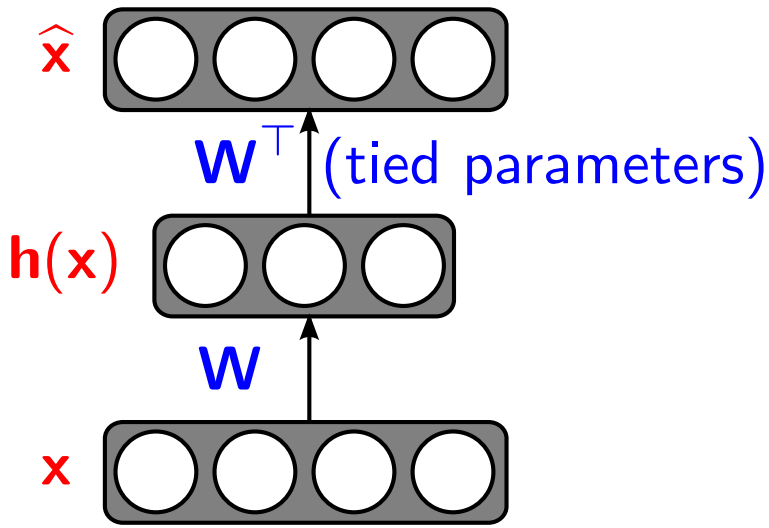
$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

Decoder:

$$\widehat{\boldsymbol{x}} = \boldsymbol{W}^\top \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{c}$$

Loss function (for real-valued inputs):

$$L(\widehat{\boldsymbol{x}}; \boldsymbol{x}) = \frac{1}{2}\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|^2$$



The Simplest Auto-Encoder

What happens if the activation function $\boldsymbol{g}$ is linear?

The Simplest Auto-Encoder

What happens if the activation function $\boldsymbol{g}$ is linear?

Principal Component Analysis (PCA)!

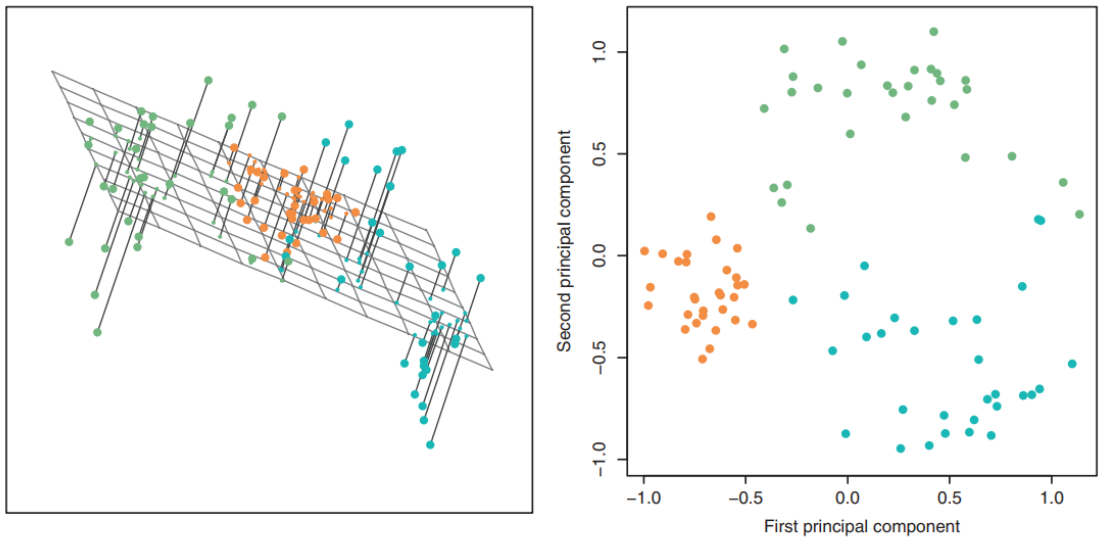


FIGURE 10.2. *Ninety observations simulated in three dimensions.* Left: *the first two principal component directions span the plane that best fits the data. It minimizes the sum of squared distances from each point to the plane.* Right: *the first two principal component score vectors give the coordinates of the projection of the 90 observations onto the plane. The variance in the plane is maximized.*

(From "An Introduction to Statistical Learning" by James, Witten, Hastie, Tibshirani)

Interlude: PCA

Consider N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$

Interlude: PCA

Consider N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$

Goal: find good K-dimensional representation of the points, with $K < D$,

$$\boldsymbol{x}^{(i)} \simeq \hat{\boldsymbol{x}}^{(i)} = \sum_{j=1}^{K} \alpha_j^{(i)} \ \boldsymbol{u}_j,$$

where $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_K\} \subset \mathbb{R}^D$ is an orthonormal basis of a subspace of $\mathbb{R}^D$.

Interlude: PCA

Consider N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$

Goal: find good K-dimensional representation of the points, with $K < D$,

$$\boldsymbol{x}^{(i)} \simeq \hat{\boldsymbol{x}}^{(i)} = \sum_{j=1}^{K} \alpha_j^{(i)} \ \boldsymbol{u}_j,$$

where $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_K\} \subset \mathbb{R}^D$ is an orthonormal basis of a subspace of $\mathbb{R}^D$.

If $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_K\}$ is fixed, and the approximation is in Euclidean norm,

$$\hat{\boldsymbol{x}}^{(i)} = \arg\min_{\boldsymbol{x}'} \|\boldsymbol{x}^{(i)} - \boldsymbol{x}'\|_2^2, \quad \text{subject to } \boldsymbol{x}' \in \text{span}(\{\boldsymbol{u}_1, ..., \boldsymbol{u}_K\})$$

Interlude: PCA

Consider N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$

Goal: find good K-dimensional representation of the points, with $K < D$,

$$\boldsymbol{x}^{(i)} \simeq \hat{\boldsymbol{x}}^{(i)} = \sum_{j=1}^{K} \alpha_j^{(i)} \, \boldsymbol{u}_j,$$

where $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_K\} \subset \mathbb{R}^D$ is an orthonormal basis of a subspace of $\mathbb{R}^D$.

If $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_K\}$ is fixed, and the approximation is in Euclidean norm,

$$\hat{\boldsymbol{x}}^{(i)} = \arg\min_{\boldsymbol{x}'} \|\boldsymbol{x}^{(i)} - \boldsymbol{x}'\|_2^2, \quad \text{subject to } \boldsymbol{x}' \in \text{span}(\{\boldsymbol{u}_1, ..., \boldsymbol{u}_K\})$$

has a well-known solution: orthogonal projection,

$$\hat{\boldsymbol{x}}^{(i)} = \sum_{j=1}^{K} \boldsymbol{u}_j \boldsymbol{u}_j^T \boldsymbol{x}^{(i)} = \boldsymbol{U}\boldsymbol{U}^T \boldsymbol{x}^{(i)}$$

where $\boldsymbol{U} = \left[\boldsymbol{u}_1, ..., \boldsymbol{u}_K\right] \in \mathbb{R}^{D \times K}$; notice that $\boldsymbol{U}^T \boldsymbol{U} = \mathsf{I}$.

Interlude: PCA

Minimizing the average error for the N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$:

$$\min_{\boldsymbol{U}^T\boldsymbol{U}=\mathrm{I}} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}\|_2^2$$

Interlude: PCA

Minimizing the average error for the N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$:

$$\min_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}\|_2^2$$

Notice that

$$\begin{aligned}
\|\boldsymbol{x}^{(i)} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}\|_2^2 &= \big((\boldsymbol{x}^{(i)})^T - (\boldsymbol{x}^{(i)})^T\boldsymbol{U}\boldsymbol{U}^T\big)\big(\boldsymbol{x}^{(i)} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}\big) \\
&\overset{(a)}{=} \|\boldsymbol{x}^{(i)}\|_2^2 - (\boldsymbol{x}^{(i)})^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)} \\
&= \|\boldsymbol{x}^{(i)}\|_2^2 - \sum_{j=1}^{K} \underbrace{(\boldsymbol{x}^{(i)})^T\boldsymbol{u}_j}_{\text{scalar}}\underbrace{\boldsymbol{u}_j^T\boldsymbol{x}^{(i)}}_{\text{scalar}} \\
&= \overbrace{\|\boldsymbol{x}^{(i)}\|_2^2}^{\text{indep. of } \boldsymbol{U}} - \sum_{j=1}^{K} \boldsymbol{u}_j^T\boldsymbol{x}^{(i)}(\boldsymbol{x}^{(i)})^T\boldsymbol{u}_j
\end{aligned}$$

(a) since $-2(\boldsymbol{x}^{(i)})^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)} + (\boldsymbol{x}^{(i)})^T\boldsymbol{U}\overbrace{\boldsymbol{U}^T\boldsymbol{U}}^{\mathsf{I}}\boldsymbol{U}^T\boldsymbol{x}^{(i)} = -(\boldsymbol{x}^{(i)})^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}$

Interlude: PCA

Minimizing the average error for the N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$:

$$\arg \min_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}\|_2^2 = \arg \max_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} \boldsymbol{u}_j^T \boldsymbol{x}^{(i)} (\boldsymbol{x}^{(i)})^T \boldsymbol{u}_j$$

Interlude: PCA

Minimizing the average error for the N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$:

$$\arg \min_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}\|_2^2 = \arg \max_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} \boldsymbol{u}_j^T\boldsymbol{x}^{(i)}(\boldsymbol{x}^{(i)})^T\boldsymbol{u}_j$$

$$= \arg \max_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \sum_{j=1}^{K} \boldsymbol{u}_j^T \underbrace{\left(\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)}(\boldsymbol{x}^{(i)})^T\right)}_{\hat{\Sigma}} \boldsymbol{u}_j$$

Interlude: PCA

Minimizing the average error for the N points in $\mathbb{R}^D$: $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$:

$$\arg \min_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}^{(i)}\|_2^2 = \arg \max_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} \boldsymbol{u}_j^T\boldsymbol{x}^{(i)}(\boldsymbol{x}^{(i)})^T\boldsymbol{u}_j$$

$$= \arg \max_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \sum_{j=1}^{K} \boldsymbol{u}_j^T \underbrace{\left(\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)}(\boldsymbol{x}^{(i)})^T\right)}_{\hat{\Sigma}} \boldsymbol{u}_j$$

$$= \arg \max_{\boldsymbol{U}^T\boldsymbol{U}=\mathsf{I}} \sum_{j=1}^{K} \boldsymbol{u}_j^T\hat{\Sigma}\boldsymbol{u}_j$$

$\hat{\Sigma}$ is the sample covariance, assuming centred data: $\sum_{i=1}^{N} \boldsymbol{x}^{(i)} = 0$.

PCA

Let's start with $K = 1$,

$$\boldsymbol{u}_1 = \arg \max_{\|\boldsymbol{u}\|_2^2 = 1} \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u}$$

PCA

Let's start with $K = 1$,

$$\boldsymbol{u}_1 = \arg \max_{\|\boldsymbol{u}\|_2^2=1} \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u}$$

Lagrangian:

$$L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} + \lambda(1 - \|\boldsymbol{u}\|_2^2)$$

PCA

Let's start with $K = 1$,

$$\boldsymbol{u}_1 = \arg \max_{\|\boldsymbol{u}\|_2^2=1} \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u}$$

Lagrangian:

$$L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} + \lambda(1 - \|\boldsymbol{u}\|_2^2)$$

Setting the gradient to zero: $2\hat{\Sigma}\boldsymbol{u} - 2\lambda\boldsymbol{u} = 0 \;\; \Rightarrow \;\; \hat{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$,

PCA

Let's start with $K = 1$,

$$\boldsymbol{u}_1 = \arg \max_{\|\boldsymbol{u}\|_2^2=1} \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u}$$

Lagrangian:

$$L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} + \lambda(1 - \|\boldsymbol{u}\|_2^2)$$

Setting the gradient to zero: $2\hat{\Sigma}\boldsymbol{u} - 2\lambda\boldsymbol{u} = 0 \;\; \Rightarrow \;\; \hat{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$,

i.e., $\boldsymbol{u}$ is an eigenvector of $\hat{\Sigma}$. Plugging above:

$$\boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} = \boldsymbol{u}^T \boldsymbol{u} \lambda = \|\boldsymbol{u}\|_2^2 \lambda = \lambda$$

PCA

Let's start with $K = 1$,

$$\boldsymbol{u}_1 = \arg \max_{\|\boldsymbol{u}\|_2^2=1} \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u}$$

Lagrangian:

$$L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} + \lambda(1 - \|\boldsymbol{u}\|_2^2)$$

Setting the gradient to zero: $2\hat{\Sigma}\boldsymbol{u} - 2\lambda\boldsymbol{u} = 0 \;\; \Rightarrow \;\; \hat{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$,

i.e., $\boldsymbol{u}$ is an eigenvector of $\hat{\Sigma}$. Plugging above:

$$\boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} = \boldsymbol{u}^T \boldsymbol{u} \lambda = \|\boldsymbol{u}\|_2^2 \lambda = \lambda$$

Conclusion: $\boldsymbol{u}_1$ is the eigenvector of the largest eigenvalue of $\hat{\Sigma}$.

PCA

Let's start with $K = 1$,

$$\boldsymbol{u}_1 = \arg \max_{\|\boldsymbol{u}\|_2^2=1} \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u}$$

Lagrangian:

$$L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} + \lambda(1 - \|\boldsymbol{u}\|_2^2)$$

Setting the gradient to zero: $2\hat{\Sigma}\boldsymbol{u} - 2\lambda\boldsymbol{u} = 0 \;\; \Rightarrow \;\; \hat{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$,

i.e., $\boldsymbol{u}$ is an eigenvector of $\hat{\Sigma}$. Plugging above:

$$\boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} = \boldsymbol{u}^T \boldsymbol{u} \lambda = \|\boldsymbol{u}\|_2^2 \lambda = \lambda$$

Conclusion: $\boldsymbol{u}_1$ is the eigenvector of the largest eigenvalue of $\hat{\Sigma}$.

Notice: $\hat{\Sigma}$ is positive semi-definite, thus all eigenvalues are non-negative.

PCA

Let's start with $K = 1$,

$$\boldsymbol{u}_1 = \arg \max_{\|\boldsymbol{u}\|_2^2=1} \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u}$$

Lagrangian:

$$L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} + \lambda(1 - \|\boldsymbol{u}\|_2^2)$$

Setting the gradient to zero: $2\hat{\Sigma}\boldsymbol{u} - 2\lambda\boldsymbol{u} = 0 \;\; \Rightarrow \;\; \hat{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$,

i.e., $\boldsymbol{u}$ is an eigenvector of $\hat{\Sigma}$. Plugging above:

$$\boldsymbol{u}^T \hat{\Sigma} \boldsymbol{u} = \boldsymbol{u}^T \boldsymbol{u} \lambda = \|\boldsymbol{u}\|_2^2 \lambda = \lambda$$

Conclusion: $\boldsymbol{u}_1$ is the eigenvector of the largest eigenvalue of $\hat{\Sigma}$.

Notice: $\hat{\Sigma}$ is positive semi-definite, thus all eigenvalues are non-negative.

Easy to extend to $K > 1$.

Interlude: PCA

$$\arg \max_{\boldsymbol{U}^T \boldsymbol{U} = \mathrm{I}} \sum_{j=1}^{K} \boldsymbol{u}_j^T \hat{\Sigma} \boldsymbol{u}_j = \text{eigenvectors of the top } K \text{ eigenvalues of } \hat{\Sigma}$$

Recall that $\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}^{(i)} (\boldsymbol{x}(i))^T = \frac{1}{N} \boldsymbol{X}^T \boldsymbol{X}$

Interlude: PCA

$$\arg \max_{\boldsymbol{U}^T \boldsymbol{U} = \mathsf{I}} \sum_{j=1}^{K} \boldsymbol{u}_j^T \hat{\Sigma} \boldsymbol{u}_j = \text{eigenvectors of the top } K \text{ eigenvalues of } \hat{\Sigma}$$

Recall that $\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}^{(i)} (\boldsymbol{x}(i))^T = \frac{1}{N} \boldsymbol{X}^T \boldsymbol{X}$

Notice that the eigenvalues of $\hat{\Sigma}$ are the singular values (SV) of $\boldsymbol{X}/\sqrt{N}$.

PCA corresponds to SVD (SV decomposition) of the scaled data matrix $\boldsymbol{X}$

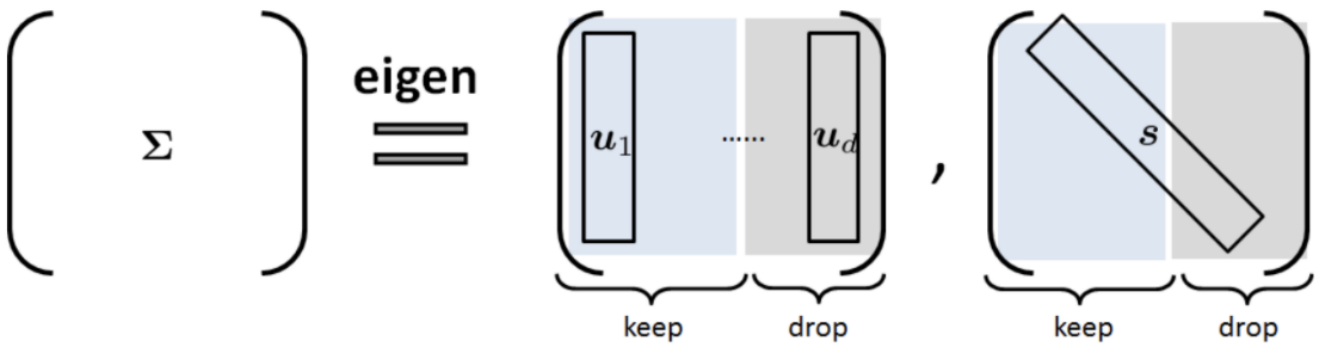
Interlude: PCA

$$\arg \max_{\boldsymbol{U}^T \boldsymbol{U} = \mathsf{I}} \sum_{j=1}^{K} \boldsymbol{u}_j^T \hat{\Sigma} \boldsymbol{u}_j = \text{eigenvectors of the top } K \text{ eigenvalues of } \hat{\Sigma}$$

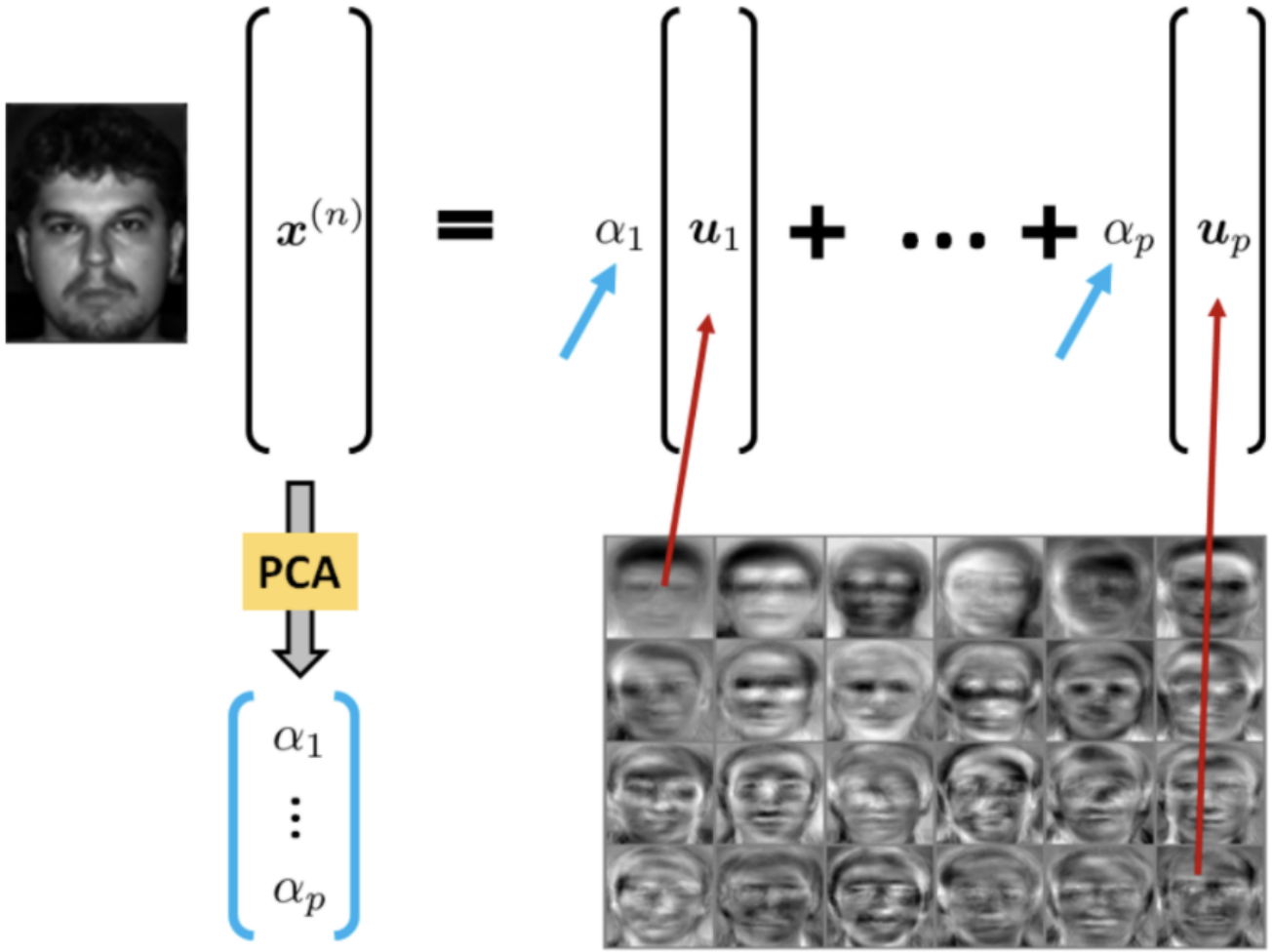
Recall that $\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}^{(i)} (\boldsymbol{x}(i))^T = \frac{1}{N} \boldsymbol{X}^T \boldsymbol{X}$

Notice that the eigenvalues of $\hat{\Sigma}$ are the singular values (SV) of $\boldsymbol{X}/\sqrt{N}$.

PCA corresponds to SVD (SV decomposition) of the scaled data matrix $\boldsymbol{X}$



PCA: EigenFaces



Back to the Linear Auto-Encoder

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the data matrix ($N > D$),

Back to the Linear Auto-Encoder

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the data matrix ($N > D$),

Assume $\boldsymbol{W} \in \mathbb{R}^{K \times D}$ with $K < D$ (no biases, assuming $\boldsymbol{X}$ is centred)

Back to the Linear Auto-Encoder

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the data matrix ($N > D$),

Assume $\boldsymbol{W} \in \mathbb{R}^{K \times D}$ with $K < D$ (no biases, assuming $\boldsymbol{X}$ is centred)

We want to minimize, with g identity, thus $\hat{\boldsymbol{x}}^{(i)} = \boldsymbol{W}^\top \boldsymbol{W}\, \boldsymbol{x}^{(i)}$,

$$\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}\|_2^2 = \|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_F^2 = \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}^\top \boldsymbol{W}\|_F^2$$

where $\|\cdot\|_F^2$ is the Frobenius matrix norm and $\boldsymbol{W}^\top \boldsymbol{W}$ has rank K.

Back to the Linear Auto-Encoder

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the data matrix ($N > D$),

Assume $\boldsymbol{W} \in \mathbb{R}^{K \times D}$ with $K < D$ (no biases, assuming $\boldsymbol{X}$ is centred)

We want to minimize, with g identity, thus $\hat{\boldsymbol{x}}^{(i)} = \boldsymbol{W}^\top \boldsymbol{W}\, \boldsymbol{x}^{(i)}$,

$$\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}\|_2^2 = \|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_F^2 = \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}^\top \boldsymbol{W}\|_F^2$$

where $\|\cdot\|_F^2$ is the Frobenius matrix norm and $\boldsymbol{W}^\top \boldsymbol{W}$ has rank K.

From the Eckart-Young theorem, the minimizer is truncated SVD of $\boldsymbol{X}^\top$:

$$\widehat{\boldsymbol{X}}^\top = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K^\top,$$

where $\boldsymbol{\Sigma}_K$ is a diagonal matrix containing the top K singular values of $\boldsymbol{X}^\top$, and the columns of $\boldsymbol{U}_K$ are the corresponding left singular vectors.

Back to the Linear Auto-Encoder

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the data matrix ($N > D$),

Assume $\boldsymbol{W} \in \mathbb{R}^{K \times D}$ with $K < D$ (no biases, assuming $\boldsymbol{X}$ is centred)

We want to minimize, with g identity, thus $\hat{\boldsymbol{x}}^{(i)} = \boldsymbol{W}^\top \boldsymbol{W}\, \boldsymbol{x}^{(i)}$,

$$\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}\|_2^2 = \|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_F^2 = \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}^\top \boldsymbol{W}\|_F^2$$

where $\|\cdot\|_F^2$ is the Frobenius matrix norm and $\boldsymbol{W}^\top \boldsymbol{W}$ has rank K.

From the Eckart-Young theorem, the minimizer is truncated SVD of $\boldsymbol{X}^\top$:

$$\widehat{\boldsymbol{X}}^\top = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K^\top,$$

where $\boldsymbol{\Sigma}_K$ is a diagonal matrix containing the top K singular values of $\boldsymbol{X}^\top$, and the columns of $\boldsymbol{U}_K$ are the corresponding left singular vectors.

The solution is $\boldsymbol{W} = \boldsymbol{U}_K^\top$, which gives as desired:

$$\widehat{\boldsymbol{X}}^\top = \boldsymbol{W}^\top \boldsymbol{W}\boldsymbol{X}^\top = \boldsymbol{U}_K \boldsymbol{U}_K^\top \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K^\top.$$

Back to the Linear Auto-Encoder

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the data matrix ($N > D$),

Assume $\boldsymbol{W} \in \mathbb{R}^{K \times D}$ with $K < D$ (no biases, assuming $\boldsymbol{X}$ is centred)

We want to minimize, with g identity, thus $\hat{\boldsymbol{x}}^{(i)} = \boldsymbol{W}^\top \boldsymbol{W}\, \boldsymbol{x}^{(i)}$,

$$\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}\|_2^2 = \|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_F^2 = \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}^\top \boldsymbol{W}\|_F^2$$

where $\|\cdot\|_F^2$ is the Frobenius matrix norm and $\boldsymbol{W}^\top \boldsymbol{W}$ has rank K.

From the Eckart-Young theorem, the minimizer is truncated SVD of $\boldsymbol{X}^\top$:

$$\widehat{\boldsymbol{X}}^\top = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K^\top,$$

where $\boldsymbol{\Sigma}_K$ is a diagonal matrix containing the top K singular values of $\boldsymbol{X}^\top$, and the columns of $\boldsymbol{U}_K$ are the corresponding left singular vectors.

The solution is $\boldsymbol{W} = \boldsymbol{U}_K^\top$, which gives as desired:

$$\widehat{\boldsymbol{X}}^\top = \boldsymbol{W}^\top \boldsymbol{W}\boldsymbol{X}^\top = \boldsymbol{U}_K \boldsymbol{U}_K^\top \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{U}_K \boldsymbol{\Sigma}_K \boldsymbol{V}_K^\top.$$

Conclusion: the optimal linear auto-encoder coincides with PCA.

Auto-Encoders

PCA fits a linear manifold (affine space) to the data

By using non-linear activations, we obtain more sophisticated codes (i.e. representations).

We need some sort of regularization to:

- encourage a smooth representation (small perturbations of the input will lead to similar codes)
- avoid overfitting to the training data

Some Variants of Auto-Encoders

- **Sparse auto-encoders:** use many hidden units, but add a ℓ_1 regularization term to encourage sparse representations of the input
- **Denoising auto-encoders:** regularize by adding noise to the input; the goal is to learn a smooth representation function that allows to output the denoised input (inspired by image denoising)
- **Stacked auto-encoders:** several auto-encoders on top of each other
- **Variational auto-encoders:** a generative probabilistic model that minimizes a variational bound (this will be covered in another lecture!)

Regularized Auto-Encoders

- To regularize auto-encoders, regularization may be added to the loss
- The goal is then to minimize $L(\widehat{\boldsymbol{x}}; \boldsymbol{x}) + \Omega(\boldsymbol{h}, \boldsymbol{x})$
- For example:
  - regularizing the code $\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \|\boldsymbol{h}\|^2$
  - regularizing the derivatives $\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i \|\nabla_{\boldsymbol{x}} h_i\|^2$
- The encoder and decoder parameters may be shared or not.

Sparse Auto-Encoders

- Most auto-encoders learn low-dimensional codes, e.g., they reduce input dimensionality (bottleneck shape $K < D$).
- But one exception are sparse auto-encoders:
  - Sparse auto-encoders incorporate a sparsity penalty $\Omega(\boldsymbol{h})$ on the code layer, e.g., $\Omega(\boldsymbol{h}) = \lambda \|\boldsymbol{h}\|_1$
  - Typically the number of hidden units is large, e.g., larger than the input dimension
  - The sparsity penalty encourages sparse codes, where most hidden units are inactive.

Stochastic Auto-Encoders

- In this case, the encoder and decoder are not deterministic functions, but involve some noise injection

- We have a distribution $p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$ for the encoder and a distribution $p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$ for the decoder

- The auto-encoder can be trained to minimize

$$-\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h}).$$

Denoising Auto-Encoders

- Use a perturbed version of the input, $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{n}$, where $\boldsymbol{n}$ is random noise (e.g. Gaussian noise $\boldsymbol{n} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$)
- Instead of minimizing $\frac{1}{2}\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|^2$, minimize $\frac{1}{2}\|\widehat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}\|^2$
- This is a form of implicit regularization that ensures smoothness: it forces the system to represent well not only the data points, but also their perturbations

Denoising Auto-Encoders

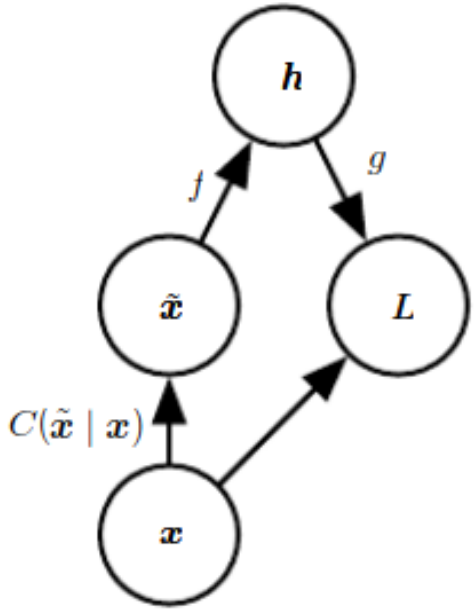


Figure 14.3: The computational graph of the cost function for a denoising autoencoder, which is trained to reconstruct the clean data point $\boldsymbol{x}$ from its corrupted version $\tilde{\boldsymbol{x}}$. This is accomplished by minimizing the loss $L = -\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$, where $\tilde{\boldsymbol{x}}$ is a corrupted version of the data example $\boldsymbol{x}$, obtained through a given corruption process $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$. Typically the distribution p_{decoder} is a factorial distribution whose mean parameters are emitted by a feedforward network g.

(From Goodfellow et al.'s book.)

Denoising Auto-Encoders

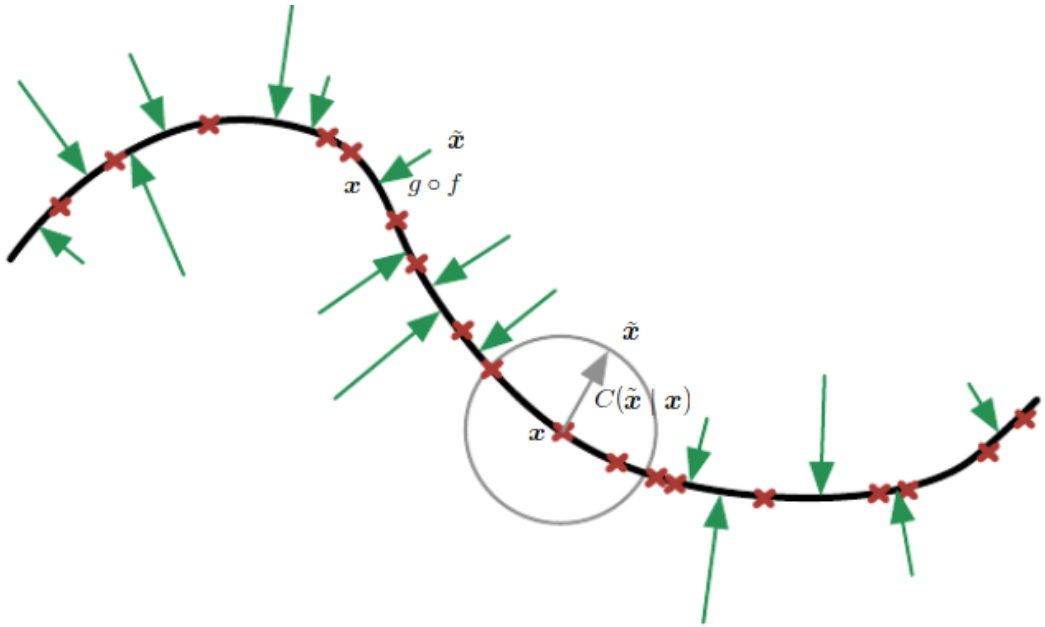


Figure 14.4: A denoising autoencoder is trained to map a corrupted data point $\tilde{\boldsymbol{x}}$ back to the original data point $\boldsymbol{x}$. We illustrate training examples $\boldsymbol{x}$ as red crosses lying near a low-dimensional manifold, illustrated with the bold black line. We illustrate the corruption process $C(\tilde{\boldsymbol{x}} \mid \boldsymbol{x})$ with a gray circle of equiprobable corruptions. A gray arrow demonstrates how one training example is transformed into one sample from this corruption process. When the denoising autoencoder is trained to minimize the average of squared errors $||g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}||^2$, the reconstruction $g(f(\tilde{\boldsymbol{x}}))$ estimates $\mathbb{E}_{\mathbf{x}, \tilde{\mathbf{x}} \sim p_{\text{data}}(\mathbf{x}) C(\tilde{\mathbf{x}} \mid \mathbf{x})}[\mathbf{x} \mid \tilde{\mathbf{x}}]$. The vector $g(f(\tilde{\boldsymbol{x}})) - \tilde{\boldsymbol{x}}$ points approximately toward the nearest point on the manifold, since $g(f(\tilde{\boldsymbol{x}}))$ estimates the center of mass of the clean points $\boldsymbol{x}$ that could have given rise to $\tilde{\boldsymbol{x}}$. The autoencoder thus learns a vector field $g(f(\boldsymbol{x})) - \boldsymbol{x}$ indicated by the green arrows. This vector field estimates the score $\nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})$ up to a multiplicative factor that is the average root mean square reconstruction error.

Why Do We Use Auto-Encoders?

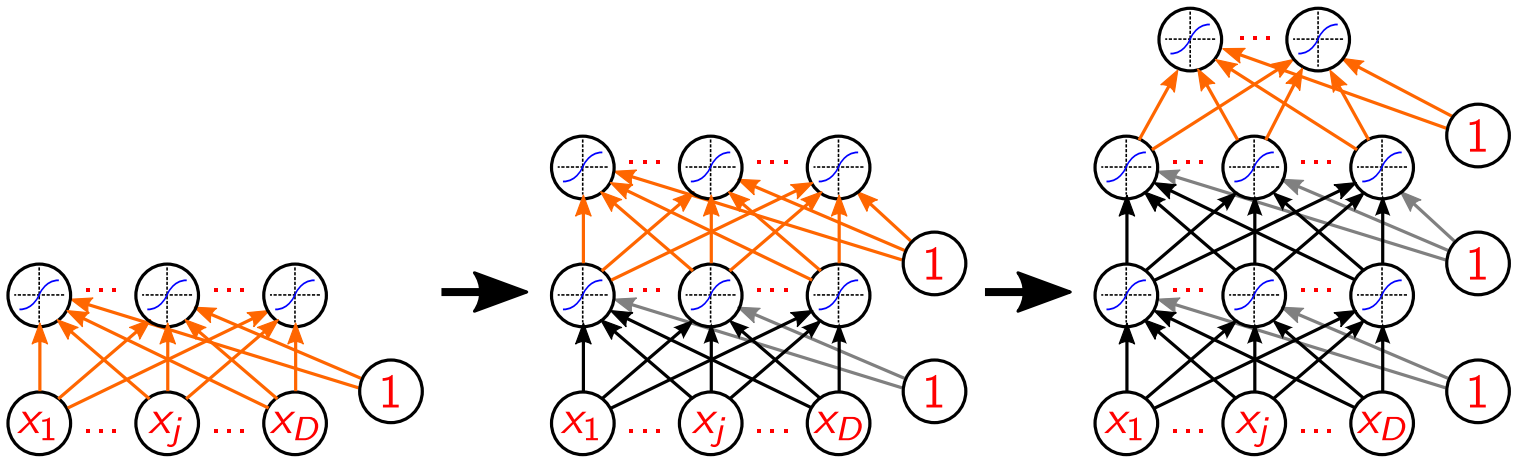
Historically, training deep neural networks was hard

One of the initial successful uses of auto-encoders was for unsupervised pre-training (Erhan et al., 2010).

Unsupervised Pre-Training

A greedy, layer-wise procedure:

- train one layer at a time, from first to last, with unsupervised criterion (e.g. an auto-encoder)
- fix the parameters of previous hidden layers
- previous layers viewed as feature extraction



Pre-training initializes the parameters in a region such that the near local optima overfit less the data.

Fine-Tuning

Once all layers are pre-trained:

- add output layer
- train the whole network using supervised learning

Supervised learning is performed as in a regular feed-forward network:

- forward propagation, backpropagation and update
- all parameters are "tuned" for the supervised task at hand
- representation is adjusted to be more discriminative

Other Applications of Auto-Encoders

- Dimensionality reduction
- Information retrieval and semantic hashing (via binarizing the codes)
- Conversion of discrete inputs to low-dimensional continuous space

Outline

Word Representations

We'll focus now on recent methods for learning representations of words in natural language

Also called word embeddings

This has been an extremely successful application of representation learning

It's still a very active area of research!

Distributional Similarity

Key idea: represent a word by means of its neighbors

- *"You shall know a word by the company it keeps"* (J. R. Firth, 1957)
- One of the most successful ideas of modern statistical NLP!

For example:

- Adjectives are normally surrounded by nouns
- Words like *book*, *newspaper*, *article*, are commonly surrounded by *reading*, *read*, *writes*, but not by *flying*, *eating*, *sleeping*

Word Embeddings

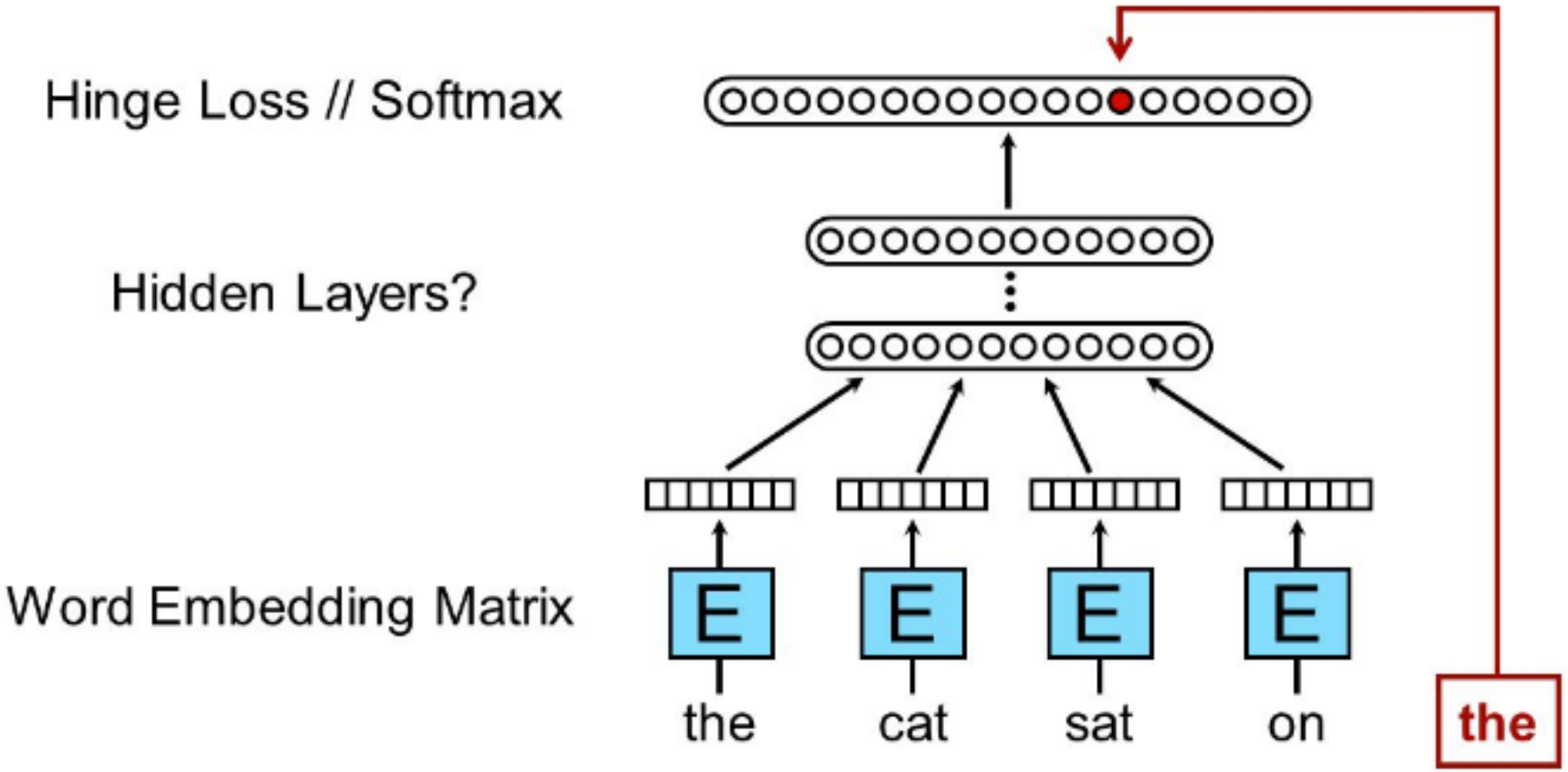
How do we obtain lower dimensional vector representations of words?

Two possible methods:

- Factorization of a co-occurrence word/context matrix (latent semantic analysis, etc.)
- Directly learn low-dimensional vectors by training a network to *predict* the context of a given word

We'll focus on the latter, incarnated in the **word2vec** toolkit (Mikolov et al., 2013), which follows previous ideas of Bengio et al. (2003) and Collobert et al. (2011).

Neural Language Model (Bengio et al., 2003)



(Image credits: Quoc Le)

Neural Language Model (Bengio et al., 2003)

- Each word is associated with a continuous vector (a word embedding)
- Given the context (previous K words), predict the next word
- This is done by concatenating the word embeddings in the context window, then propagating them through a feedforward neural network
- The output layer is a gigantic softmax that assigns a probability value to each word in the vocabulary

Variants of this model achieved better accuracy than smoothed K-th order Markov models

As a by-product: word embeddings!

The embedding matrix is a lookup table that assigns a continuous vector to every word in the vocabulary.

Neural Language Model

In this class, we are not concerned with language modeling (the actual task), but rather about the quality of the embeddings (the representations we learn for that task).

Some Insights

If we don't care about language modeling as a task:

1. We don't need to have a "left-to-right model" where we try to predict the next word given the context
2. We don't need to predict the probability of every word, we might just make sure that the true word is more likely than a random word

These insights underlie the word2vec model of Mikolov et al. (2013).

Word2Vec (Mikolov et al., 2013)

Considers a context window around each word in the sentence.

Word2vec comes with two variants:

- **Skip-gram**: predict surrounding context words in a window of length m of every word
- **Continuous bag-of-words** (CBOW): predict the central word from the context

We'll focus on the skip-gram model (more widely used).

Skip-Gram

Goal: maximize the log probability of any context word given the current center word:

$$J(\Theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m,\ j \neq 0} \log p_\Theta(x_{t+j} \mid x_t)$$

There are two sets of parameters $\Theta = (\boldsymbol{u}, \boldsymbol{v})$:

- Embeddings $\boldsymbol{u}_o$ for each word o appearing as the center word
- Embeddings $\boldsymbol{v}_c$ for each word c appearing in the context of another word

Define a log-bilinear model: $p_\Theta(x_{t+j} = c \mid x_t = o) \propto \exp(\boldsymbol{u}_o \cdot \boldsymbol{v}_c)$

Every word gets two vectors!

In the end, we use the $\boldsymbol{u}$ vectors as the word embeddings and discard the $\boldsymbol{v}$ vectors

The Large Vocabulary Problem

Recall that we have

$$p_\Theta(x_{t+j} = c \mid x_t = o) = \frac{\exp(\boldsymbol{u}_o \cdot \boldsymbol{v}_c)}{\sum_c' \exp(\boldsymbol{u}_o \cdot \boldsymbol{v}_c')}$$

This objective requires a softmax over the entire vocabulary

Unfortunately, with large vocabularies this leads to very slow training :(

Workarounds:

- Stochastic sampling
- Noise contrastive estimation
- Negative sampling

More details in these notes: `https://arxiv.org/pdf/1410.8251.pdf`

We'll focus on negative sampling.

Negative Sampling

Key idea:

- replace the gigantic softmax by binary logistic regressions for a true pair (center word and word in its context window) and a couple of random pairs (the center word with a random word):

$$J_t(\Theta) = \log \sigma(\boldsymbol{u}_o \cdot \boldsymbol{v}_c) + \sum_{i=1}^{k} \log \sigma(-\boldsymbol{u}_o \cdot \boldsymbol{v}_{j_i}), \qquad j_i \sim P(x)$$

- Several strategies for the sampling distribution $P(x)$ (uniform, unigram frequency, etc.)

Negative sampling is a simple form of unsupervised pre-training.

Linear Relationships

- These representations are very good at encoding dimensions of similarity!
- Word analogies can be solved quite well just by doing vector subtraction in the embedding space
- Syntactically:

$$\boldsymbol{x}_{\text{apple}} - \boldsymbol{x}_{\text{apples}} \approx \boldsymbol{x}_{\text{car}} - \boldsymbol{x}_{\text{cars}} \approx \boldsymbol{x}_{\text{family}} - \boldsymbol{x}_{\text{families}}$$

- Semantically:

$$\begin{aligned} \boldsymbol{x}_{\text{shirt}} - \boldsymbol{x}_{\text{clothing}} &\approx \boldsymbol{x}_{\text{chair}} - \boldsymbol{x}_{\text{furniture}} \\ \boldsymbol{x}_{\text{king}} - \boldsymbol{x}_{\text{man}} &\approx \boldsymbol{x}_{\text{queen}} - \boldsymbol{x}_{\text{woman}} \end{aligned}$$

Visualization

Typical word embedding dimensions are on the hundreds (e.g. 300)

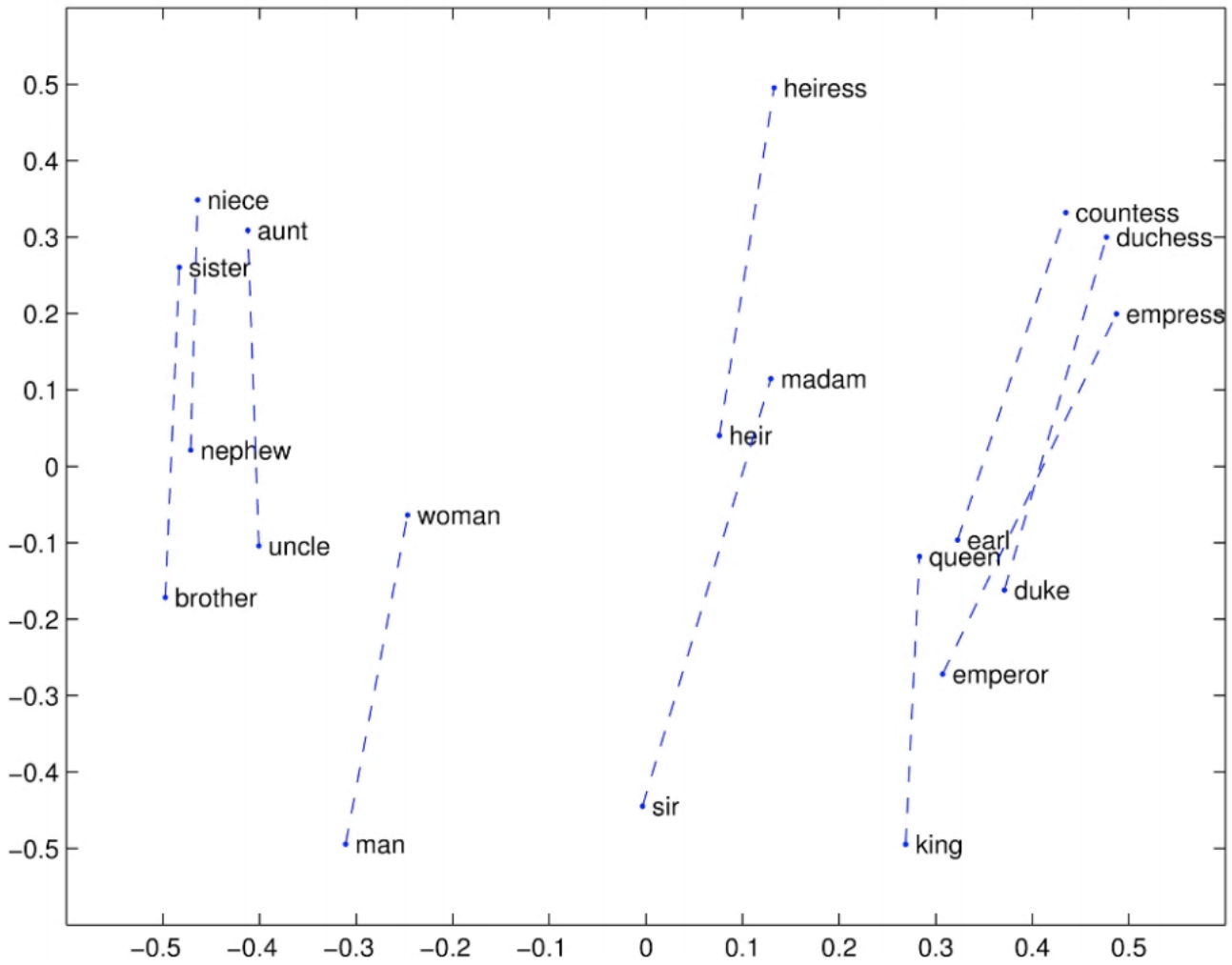
How can we visualize these embeddings?

Simple way: project them in 2D with something like PCA!

Most used: t-distributed stochastic neighbor embedding (t-SNE, Maaten and Hinton 2008)

`https://lvdmaaten.github.io/tsne`

Word Analogies (Mikolov et al., 2013)



(Slide credit to Richard Socher)

Other Methods for Obtaining Word Embeddings

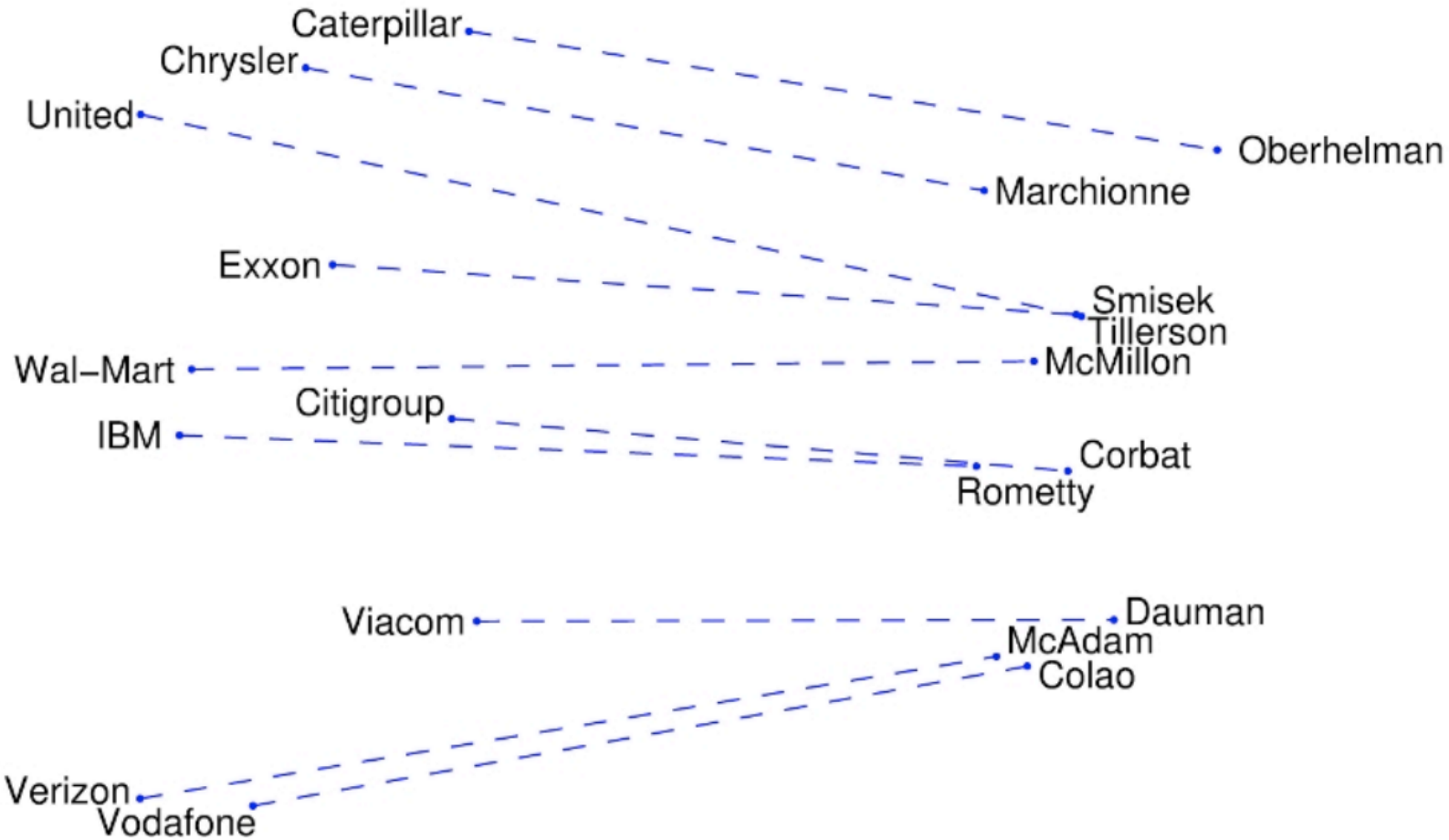
GloVe: Global Vectors for Word Representation (Pennington et al., 2014)

- `https://nlp.stanford.edu/projects/glove`
- Training is performed on aggregated global word-word co-occurrence statistics from a corpus

fastText (Bojanowski et al., 2016): embeds also character *n*-grams for generating embeddings for out-of-vocabulary words

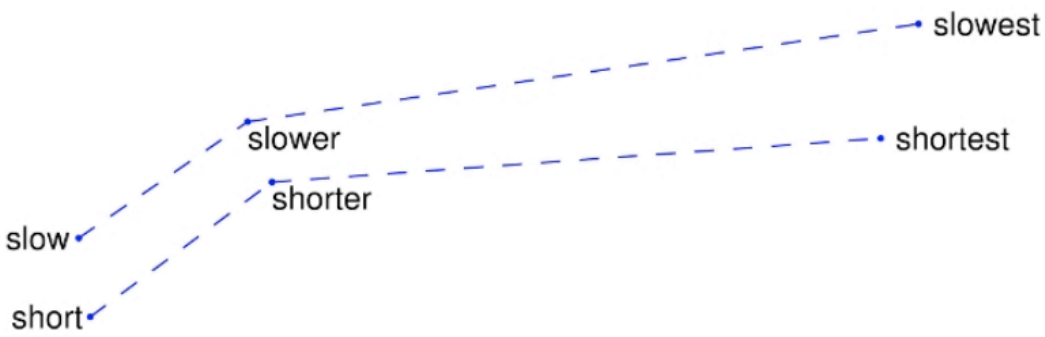
- `https://fasttext.cc` (from FAIR)
- open-source, free, lightweight library that allows users to learn text representations and text classifiers
- contains multi-lingual word vectors for 157 different languages

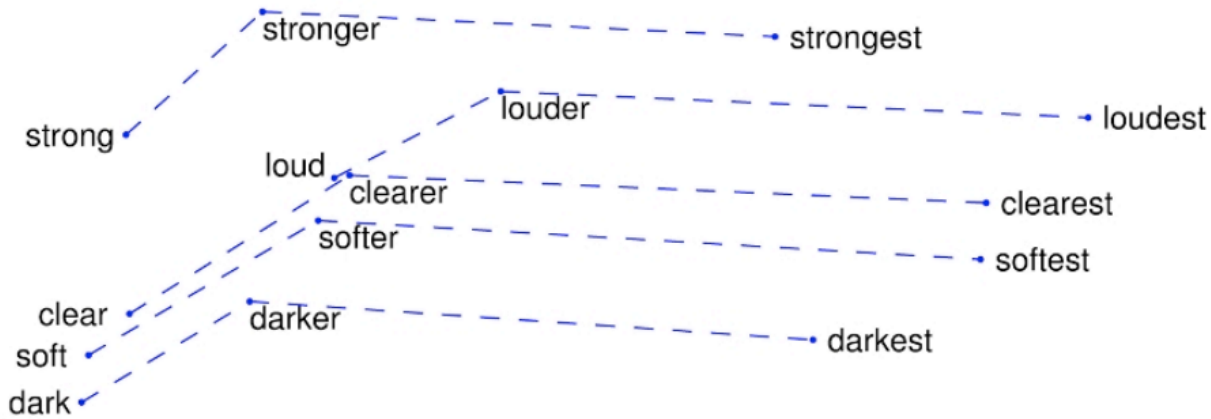
GloVe Visualizations: Company → CEO



(Slide credit to Richard Socher)

GloVe Visualizations: Superlatives





(Slide credit to Richard Socher)

Word Embeddings: Some Open Problems

- Can we have word embeddings for multiple languages in the same space?
- How to capture polysemy?
- These word embeddings are static, can we compute embeddings on-the-fly depending on the context?

Cross-Lingual Word Embeddings

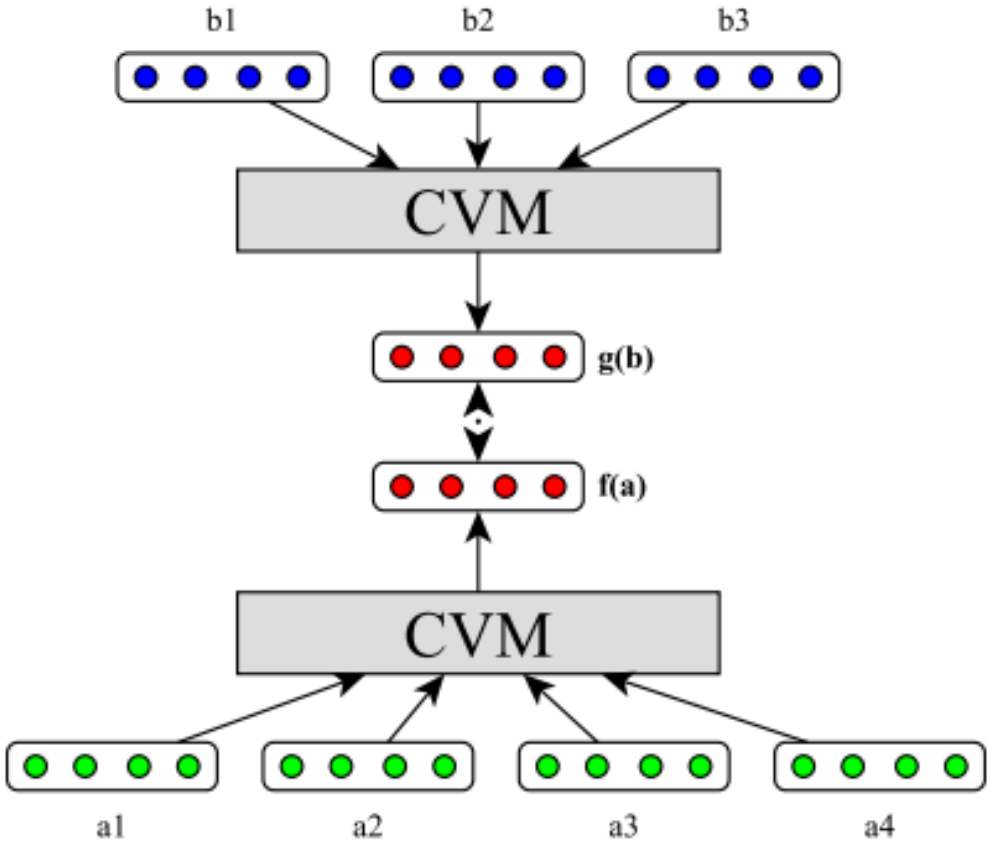


Figure 1: Model with parallel input sentences a and b. The model minimises the distance between the sentence level encoding of the bitext. Any composition functions (CVM) can be used to generate the compositional sentence level representations.

(From Hermann and Blunsom (2014).)

Cross-Lingual Word Embeddings

Key idea:

- use a corpus of parallel sentences in two languages
- define a composition function to obtain a sentence representation given word embeddings
- apply a loss function that encourages the sentence represention in the two languages to be similar
- negative sampling works here too: true pair vs fake pair.

Cross-Lingual Word Embeddings

Other approaches:

- Define a bilingual dictionary and apply canonical correlation analysis (Faruqui and Dyer, 2014)
- Task-specific embeddings with convex optimization (Ferreira et al., 2016)
- Learn the two embeddings separately, and then apply a linear transformation to put them in a shared space (Artetxe et al., 2017)
- Adversarial training (Lample et al., 2018)

This is a very active area of research!

Contextual Embeddings

Words can have different meanings, depending on which context they appear in.

In 2018, a model called ELMo learned context-dependent embeddings and achieved impressive results on 6 NLP downstream tasks (Peters et al., 2018)

Key idea:

- Pre-train a BILSTM language model on a large dataset (we'll see in a later class what this is)
- Save all the encoder parameters at all layers, not only the embeddings
- Then, for your downstream task, tune a scalar parameter for each layer, and pass the entire sentence through this encoder.

BERT, GPT, etc.

Some time later, a Transformer-based model (BERT) achieved even better performance:

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Huge improvements in multiple NLP tasks!

(Trained on 64 TPU chips!!)

Other related models include GPT-2, GPT-3, etc.

This will be covered in a later lecture!

Outline

Conclusions

- Neural nets learn internal representations that can be transferred across tasks
- Distributed representations are exponentially more compact and allow generalizing to unseen objects
- Deeper neural nets exhibit hierarchical compositionality: upper level layers learn more abstract/semantic representations than bottom level layers
- Auto-encoders are an effective means for learning representations
- Word embeddings are continuous representations of words that are extremely useful in NLP

Thank you!

Questions?

References I

Artetxe, M., Labaka, G., and Agirre, E. (2017). Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 451–462.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.

Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471.

Ferreira, D., Almeida, M. S. C., and Martins, A. F. T. (2016). Jointly Learning to Embed and Predict with Multiple Languages. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Hermann, K. M. and Blunsom, P. (2014). Multilingual Models for Compositional Distributional Semantics. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Hinton, G. E. (1984). Distributed representations.

Lample, G., Ott, M., Conneau, A., Denoyer, L., and Ranzato, M. (2018). Phrase-based & neural unsupervised machine translation. *arXiv preprint arXiv:1804.07755*.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*.