Lecture 6: Representation Learning

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Announcements

Deadline for Homework 1 is **Friday, December 23, 23:59**

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

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 $\phi(x)$
- Standard linear models require manually engineered features $\phi(x)$

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

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)

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

Hierarchical Compositionality

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

Vision:

- pixels → edges → textons → motifs → parts → objects → scenes

Speech:

- samples → spectral bands → formants → motifs → phonemes → words

Hierarchical Compositionality

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

Vision:

- pixels → edges → textons → motifs → parts → objects → scenes

Speech:

- samples → spectral bands → formants → motifs → phonemes → words

Text:

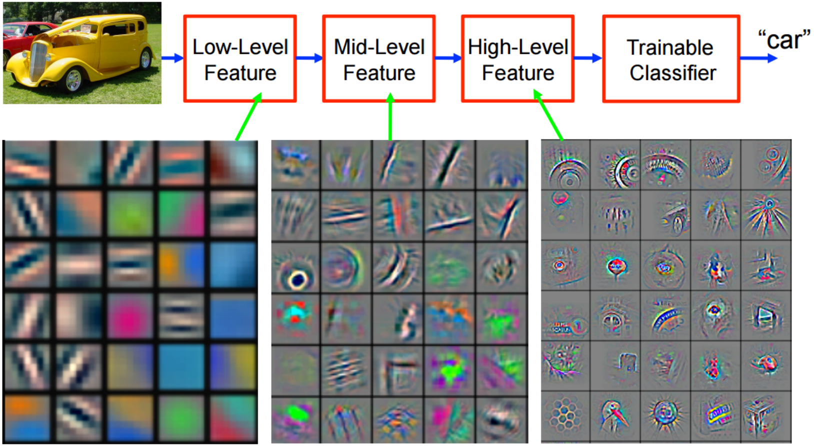
- characters → words → phrases → sentences → stories

(Inspired by Marc'Aurelio Ranzato and Yann LeCun)

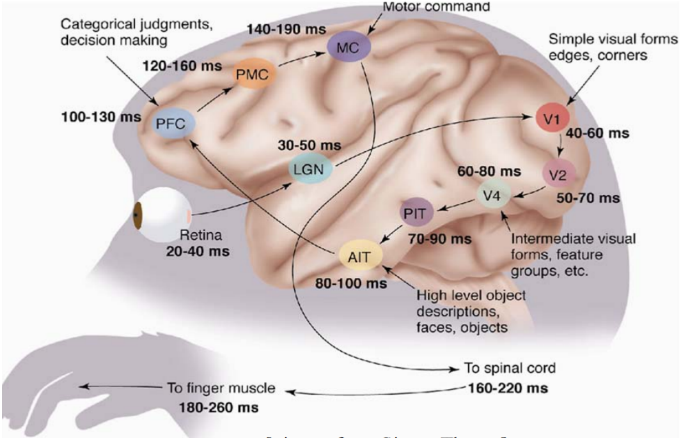
Hierarchical Compositionality

Feature visualization of convolutional NNs trained on ImageNet

(Zeiler and Fergus, 2013)



The Mammalian Visual Cortex is Hierarchical



[picture from Simon Thorpe]

(LGN = lateral geniculate nucleus; PIT = posterior inferotemporal area; AIT = anterior inferotemporal area; PFC = prefrontal cortex; PMC = premotor cortex; MC = motor cortex)

What Is Learned in Each Layer

- Layers closer to inputs learn low-level representations (corners, edges)

What Is Learned in Each Layer

- Layers closer to inputs learn low-level representations (corners, edges)
- Layers farther away from inputs learn more abstract representations (shapes, forms, objects)

What Is Learned in Each Layer

- Layers closer to inputs learn low-level representations (corners, edges)
- Layers farther away from inputs learn more abstract representations (shapes, forms, objects)
- This holds, not only for images, but also text, sounds, ...

Outline

1. Representation Learning
   - Hierarchical Compositionality
   - Distributed Representations
   - Auto-Encoders
   - Word Embeddings

2. Conclusions

Distributed Representations (Hinton, 1984)

This is a central concept in neural networks.

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)?

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?

Distributed Representations (Hinton, 1984)

This is a central concept in neural networks.

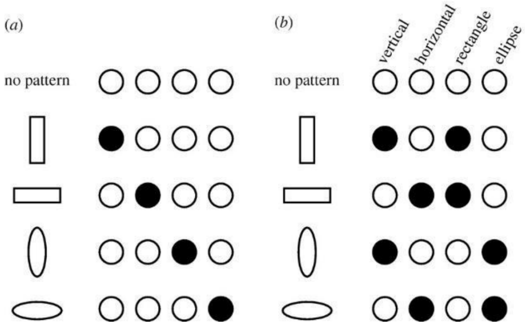
Key questions:

- 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)



(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!

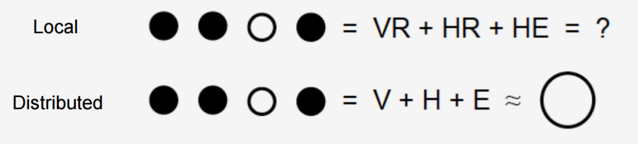
Distributed Representations

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:



(Inspired by Moontae Lee and Dhruv Batra)

The Power of Distributed Representations

- Hidden units should capture diverse properties of objects (not all capturing the same property)

The Power of Distributed Representations

- Hidden units should capture diverse properties of objects (not all capturing the same property)

- Usually ensured by random initialization of the weights

The Power of Distributed Representations

- 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 would never break the symmetry!

The Power of Distributed Representations

- 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 would never break the symmetry!
- Side note: a NN computes the same function if we permute the hidden units within a layer (order doesn't matter, only diversity)

The Power of Distributed Representations

- 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 would never break the symmetry!
- Side note: a NN computes the same function if we permute the hidden units within a layer (order doesn't matter, only diversity)

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

Example: Unsupervised Pre-Training

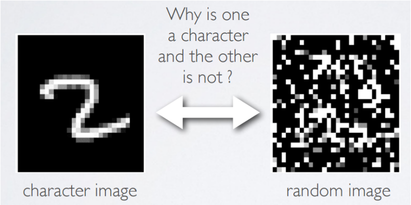
- Training deep NNs (with many hidden layers) can be challenging
- This has been a major difficulty with NNs for a long time
- Initialize hidden layers using unsupervised learning (Erhan et al., 2010):
  - Force network to represent latent structure of input distribution
  - Encourage hidden layers to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

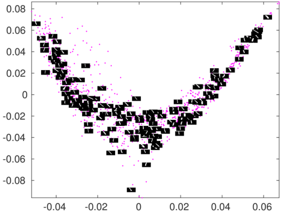
Outline

1. Representation Learning
   - Hierarchical Compositionality
   - Distributed Representations
   - Auto-Encoders
   - Word Embeddings
2. Conclusions

Data Manifold

Key idea: learn the manifold where the input objects live





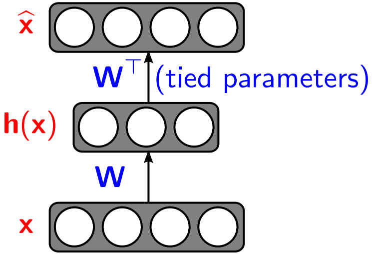
(Image credit: Hugo Larochelle)

Learn representations that encode well points in that manifold

Auto-Encoders

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

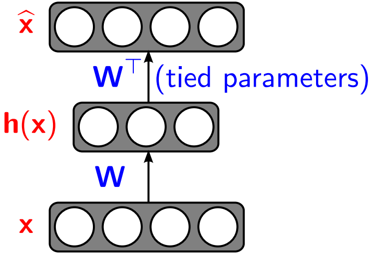
- Encoder: $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$



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

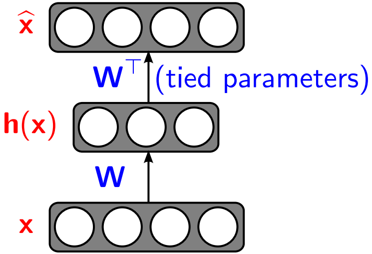


Auto-Encoders

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 $\boldsymbol{x} \in \mathbb{R}^D$):

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



Auto-Encoders

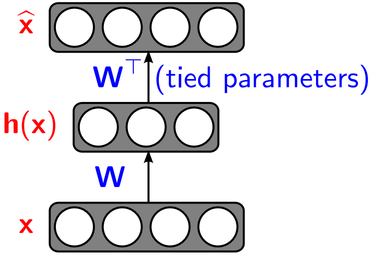
Auto-encoder: feed-forward NN trained to reproduce its input at the output

- 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 $\boldsymbol{x} \in \mathbb{R}^D$):

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

- Objective (dropping the biases):

$$\widehat{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} \sum_i \|\boldsymbol{W}^\top \boldsymbol{g}(\boldsymbol{W}\boldsymbol{x}_i) - \boldsymbol{x}_i\|^2$$



The Simplest Auto-Encoder: Linear

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

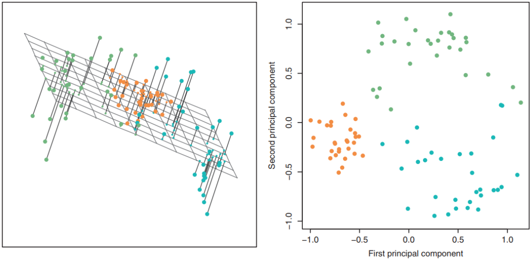


FIGURE 10.2. *Ninety observations simulated in three dimensions.* Left: *the first two principal component directions span the plane that best fits the data. It minimizes the sum of squared distances from each point to the plane.* Right: *the first two principal component score vectors give the coordinates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

An Important Tool: Singular Value Decomposition

- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$

An Important Tool: Singular Value Decomposition

- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - ✓ columns of $\boldsymbol{U} \in \mathbb{R}^{m \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A})$;

An Important Tool: Singular Value Decomposition

- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - ✓ columns of $\boldsymbol{U} \in \mathbb{R}^{m \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A})$;
  - ✓ columns of $\boldsymbol{V} \in \mathbb{R}^{n \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A}^T)$;

An Important Tool: Singular Value Decomposition

- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - ✓ columns of $\boldsymbol{U} \in \mathbb{R}^{m \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A})$;
  - ✓ columns of $\boldsymbol{V} \in \mathbb{R}^{n \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A}^T)$;
  - ✓ $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, ..., \sigma_r)$ is a $r \times r$ diagonal matrix;

An Important Tool: Singular Value Decomposition

- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^T$
  - ✓ columns of $\boldsymbol{U} \in \mathbb{R}^{m \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A})$;
  - ✓ columns of $\boldsymbol{V} \in \mathbb{R}^{n \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A}^T)$;
  - ✓ $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, ..., \sigma_r)$ is a $r \times r$ diagonal matrix;
  - ✓ $\sigma_1, ..., \sigma_r$ are square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ or $\boldsymbol{A}\boldsymbol{A}^T$;

An Important Tool: Singular Value Decomposition

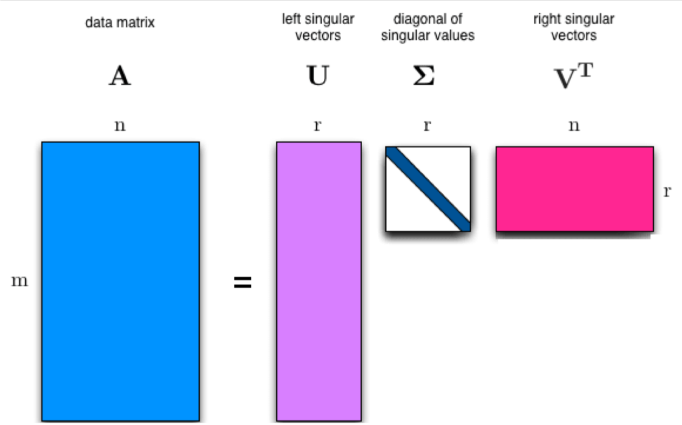
- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - ✓ columns of $\boldsymbol{U} \in \mathbb{R}^{m \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A})$;
  - ✓ columns of $\boldsymbol{V} \in \mathbb{R}^{n \times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A}^T)$;
  - ✓ $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, ..., \sigma_r)$ is a $r \times r$ diagonal matrix;
  - ✓ $\sigma_1, ..., \sigma_r$ are square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ or $\boldsymbol{A}\boldsymbol{A}^T$;
  - ✓ $\sigma_1, ..., \sigma_r$ are called singular values.

An Important Tool: Singular Value Decomposition

- Any rank-r matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ can be written as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - ✓ columns of $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A})$;
  - ✓ columns of $\boldsymbol{V} \in \mathbb{R}^{n\times r}$ are an orthonormal basis of $\mathcal{R}(\boldsymbol{A}^T)$;
  - ✓ $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, ..., \sigma_r)$ is a $r \times r$ diagonal matrix;
  - ✓ $\sigma_1, ..., \sigma_r$ are square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ or $\boldsymbol{A}\boldsymbol{A}^T$;
  - ✓ $\sigma_1, ..., \sigma_r$ are called singular values.
- Orthonormality of $\boldsymbol{U}$ and $\boldsymbol{V}$: $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$.

Singular Value Decomposition (SVD)

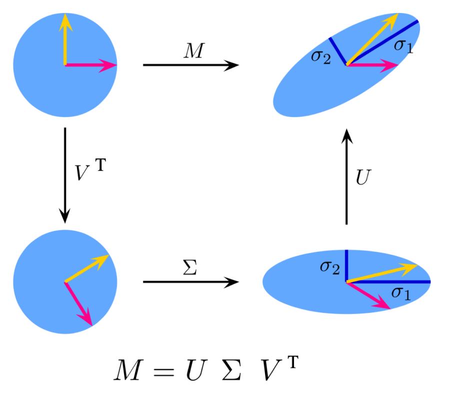
- $\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times r}$.



Picture credits: Mukesh Mithrakumar

Singular Value Decomposition (SVD)

- $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{m \times r}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times r}$.



Picture credits: Wikipedia

Linear Auto-Encoder

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

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)

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

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

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

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

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

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.

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

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

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:

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

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

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

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 $\boldsymbol{X}^\top$:

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

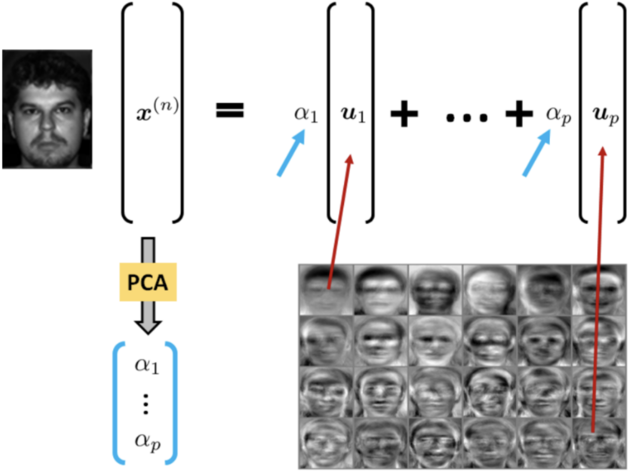
where $\boldsymbol{\Sigma}_K$ is a diagonal matrix containing the top K singular values of $\boldsymbol{X}^\top$, and the columns of $\boldsymbol{U}_K$ are the corresponding left singular vectors.

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

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

This is called principle component analysis (PCA)

PCA: EigenFaces



Auto-Encoders

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

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).

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 representations)
- 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

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)

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

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

Regularized Auto-Encoders

- To regularize auto-encoders, regularization may be added to the loss
- The goal is then to minimize $L(\widehat{\boldsymbol{x}}; \boldsymbol{x}) + \Omega(\boldsymbol{h}, \boldsymbol{x})$

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 $\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i \|\nabla_{\boldsymbol{x}} h_i\|^2$

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 $K < D$).

Sparse Auto-Encoders

- Most auto-encoders learn low-dimensional codes, e.g., they reduce input dimensionality (bottleneck shape $K < D$).
- 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/randomness

Stochastic Auto-Encoders

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

Stochastic Auto-Encoders

- In this case, the encoder and decoder are not deterministic functions, but involve some noise/randomness
- Uses 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 $\boldsymbol{n} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$)

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 $\frac{1}{2}\|\widehat{\boldsymbol{x}} - \tilde{\boldsymbol{x}}\|^2$

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

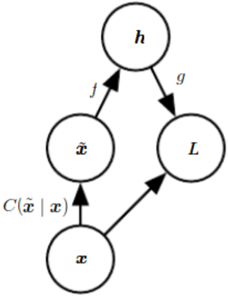


Figure 14.3: The computational graph of the cost function for a denoising autoencoder, which is trained to reconstruct the clean data point $\boldsymbol{x}$ from its corrupted version $\tilde{\boldsymbol{x}}$. This is accomplished by minimizing the loss $L = -\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$, 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

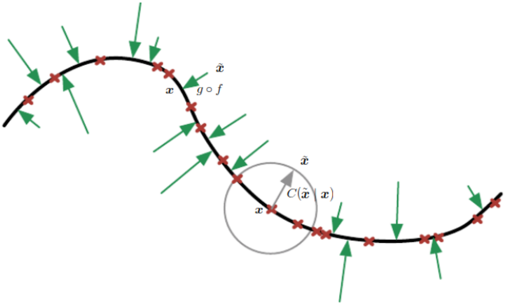


Figure 14.4: A denoising autoencoder is trained to map a corrupted data point $\tilde{\boldsymbol{x}}$ back to the original data point $\boldsymbol{x}$. We illustrate training examples $\boldsymbol{x}$ as red crosses lying near a low-dimensional manifold, illustrated with the bold black line. We illustrate 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)

Unsupervised Pre-Training

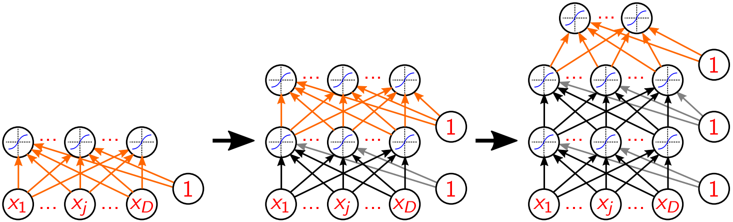
A greedy, layer-wise procedure:

- 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

Unsupervised Pre-Training

A greedy, layer-wise procedure:

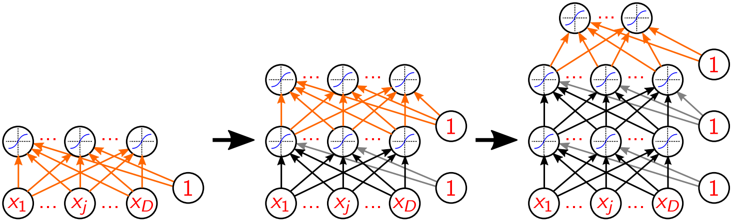
- train one layer at a time, from first to last, with unsupervised criterion (e.g. an auto-encoder)
- fix the parameters of previous hidden layers
- previous layers viewed as feature extraction



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

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

1. Representation Learning

   Hierarchical Compositionality

   Distributed Representations

   Auto-Encoders

   Word Embeddings

2. Conclusions

Word Representations

- Learning representations of words in natural language;
- Also called word embeddings;
- An extremely successful application of representation learning;
- Still an 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!

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

- Obtain (lower dimensional) vector representations of words.

Word Embeddings

- 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

Word Embeddings

- Obtain (lower dimensional) vector representations of words.

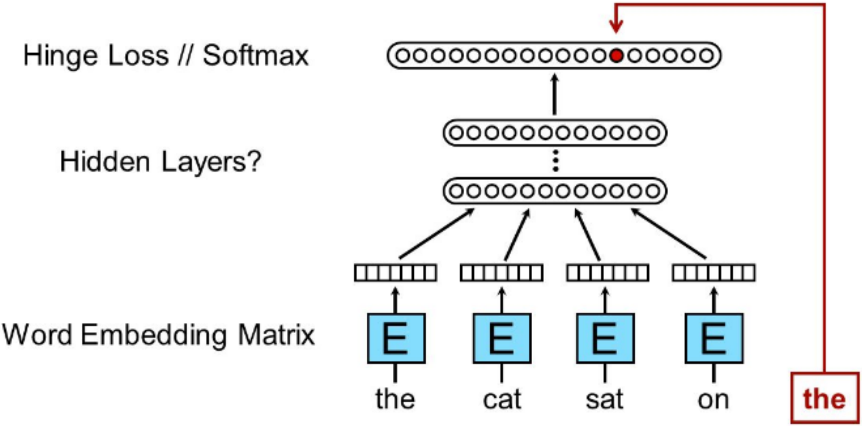
- Two possible methods:

  ✓ 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 focus on the latter, namely word2vec (Mikolov et al., 2013), which follows previous ideas (Bengio et al., 2003; Collobert et al., 2011).

Neural Language Model (Bengio et al., 2003)



(Image credits: Quoc Le)

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Learning the embeddings:

- Each word is associated with a vector (word embedding)

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Learning the embeddings:

- Each word is associated with a vector (word embedding)
- Given the context (previous K words), predict the next word

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Learning the embeddings:

- Each word is associated with a vector (word embedding)
- Given the context (previous K words), predict the next word
- The word embeddings in the context window are concatenated into a vector that is fed to neural network

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Learning the embeddings:

- Each word is associated with a vector (word embedding)
- Given the context (previous K words), predict the next word
- The word embeddings in the context window are concatenated into a vector that is fed to neural network
- The output layer is a huge softmax assigning probabilities to each word in the vocabulary

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Learning the embeddings:

- Each word is associated with a vector (word embedding)
- Given the context (previous K words), predict the next word
- The word embeddings in the context window are concatenated into a vector that is fed to neural network
- The output layer is a huge softmax assigning probabilities to each word in the vocabulary
- The network is trained by SGD with backpropagation including the embedding matrix.

Neural Language Model (Bengio et al., 2003)

Embedding matrix: assigns a vector to every word in the vocabulary.

Learning the embeddings:

- Each word is associated with a vector (word embedding)
- Given the context (previous K words), predict the next word
- The word embeddings in the context window are concatenated into a vector that is fed to neural network
- The output layer is a huge softmax assigning probabilities to each word in the vocabulary
- The network is trained by SGD with backpropagation including the embedding matrix.

Variants of this model outperform smoothed K-th order Markov models

Some Insights

- Often, we are not concerned with language modelling (the addressed task), but with the quality of the embeddings learned

Some Insights

- Often, we are not concerned with language modelling (the addressed task), but with the quality of the embeddings learned

- If we don't care about language modelling,

  ✓ We don't need to have a "left-to-right model" where we try to predict the next word given the context

  ✓ We don't need to predict the probability of every word, just make sure that the true word is more likely than a random one

Some Insights

- Often, we are not concerned with language modelling (the addressed task), but with the quality of the embeddings learned
- If we don't care about language modelling,
  - ✓ We don't need to have a "left-to-right model" where we try to predict the next word given the context
  - ✓ We don't need to predict the probability of every word, just make sure that the true word is more likely than a random one
- 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 (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

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

Skip-Gram

- **Objective:** maximize the log probability of any context word given the central word:

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

Skip-Gram

- **Objective:** maximize the log probability of any context word given the central word:

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

- There are 2 sets of parameters (2 embedding matrices) $\Theta = (\boldsymbol{u}, \boldsymbol{v})$:
  - ✓ Embeddings $\boldsymbol{u}_o$ for each word o appearing as the center word
  - ✓ Embeddings $\boldsymbol{v}_c$ for each word c appearing as context of another word

Skip-Gram

- **Objective:** maximize the log probability of any context word given the central 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 2 sets of parameters (2 embedding matrices) $\Theta = (\boldsymbol{u}, \boldsymbol{v})$:
  - ✓ Embeddings $\boldsymbol{u}_o$ for each word o appearing as the center word
  - ✓ Embeddings $\boldsymbol{v}_c$ for each word c appearing as context of another word
- Uses a log-bilinear model: $p_\Theta(x_{t+j} = c \mid x_t = o) \propto \exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)$

Skip-Gram

- **Objective:** maximize the log probability of any context word given the central 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 2 sets of parameters (2 embedding matrices) $\Theta = (\boldsymbol{u}, \boldsymbol{v})$:
  - ✓ Embeddings $\boldsymbol{u}_o$ for each word o appearing as the center word
  - ✓ Embeddings $\boldsymbol{v}_c$ for each word c appearing as context of another word
- Uses a log-bilinear model: $p_\Theta(x_{t+j} = c \mid x_t = o) \propto \exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)$
- Every word gets two vectors

Skip-Gram

- **Objective:** maximize the log probability of any context word given the central 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 2 sets of parameters (2 embedding matrices) $\Theta = (\boldsymbol{u}, \boldsymbol{v})$:
  - ✓ Embeddings $\boldsymbol{u}_o$ for each word o appearing as the center word
  - ✓ Embeddings $\boldsymbol{v}_c$ for each word c appearing as context of another word
- Uses a log-bilinear model: $p_\Theta(x_{t+j} = c \mid x_t = o) \propto \exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)$
- Every word gets two vectors
- In the end, we use the $\boldsymbol{u}$ as the word embeddings, discarding $\boldsymbol{v}$

The Large Vocabulary Problem

- Recall that

$$p_\Theta(x_{t+j} = c \mid x_t = o) = \frac{\exp(\boldsymbol{u_o}^T \boldsymbol{v}_c)}{\sum_{c'} \exp(\boldsymbol{u_o}^T \boldsymbol{v}_{c'})}$$

- This is a softmax over the entire vocabulary (often > 50000)

The Large Vocabulary Problem

- Recall that

$$p_\Theta(x_{t+j} = c \mid x_t = o) = \frac{\exp(\boldsymbol{u_o}^T \boldsymbol{v_c})}{\sum_{c'} \exp(\boldsymbol{u_o}^T \boldsymbol{v_{c'}})}$$

- This is a softmax over the entire vocabulary (often > 50000)
- With large vocabularies, this leads to very slow training. Possible workarounds:
  - ✓ Stochastic sampling
  - ✓ Noise contrastive estimation
  - ✓ Negative sampling

The Large Vocabulary Problem

- Recall that

$$p_\Theta(x_{t+j} = c \mid x_t = o) = \frac{\exp(\boldsymbol{u_o}^T \boldsymbol{v_c})}{\sum_{c'} \exp(\boldsymbol{u_o}^T \boldsymbol{v_{c'}})}$$

- This is a softmax over the entire vocabulary (often > 50000)
- With large vocabularies, this leads to very slow training. Possible workarounds:
  - ✓ Stochastic sampling
  - ✓ Noise contrastive estimation
  - ✓ Negative sampling
- More details in `https://arxiv.org/pdf/1410.8251.pdf`

The Large Vocabulary Problem

- Recall that

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

- This is a softmax over the entire vocabulary (often > 50000)
- With large vocabularies, this leads to very slow training. Possible workarounds:
  - ✓ Stochastic sampling
  - ✓ Noise contrastive estimation
  - ✓ Negative sampling
- More details in `https://arxiv.org/pdf/1410.8251.pdf`
- We focus on negative sampling.

Negative Sampling

Key idea:

- Replace the huge softmax by binary logistic regressions for a true pair (center word and word in its context window) and k of random pairs (center word, random word):

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

Negative Sampling

Key idea:

- Replace the huge softmax by binary logistic regressions for a true pair (center word and word in its context window) and k of random pairs (center word, random word):

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

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

Negative Sampling

Key idea:

- Replace the huge softmax by binary logistic regressions for a true pair (center word and word in its context window) and k of random pairs (center word, random word):

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

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

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

Linear Relationships

- Work embeddings are good at encoding dimensions of similarity

Linear Relationships

- Work embeddings are good at encoding dimensions of similarity
- Word analogies can be solved well simply via subtraction in the embedding space

Linear Relationships

- Work embeddings are good at encoding dimensions of similarity
- Word analogies can be solved well simply via subtraction in the embedding space
- Syntactically:

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

Linear Relationships

- Work embeddings are good at encoding dimensions of similarity
- Word analogies can be solved well simply via 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 embedding dimensions are in the hundreds (e.g. 300)
- How can we visualize these embeddings?

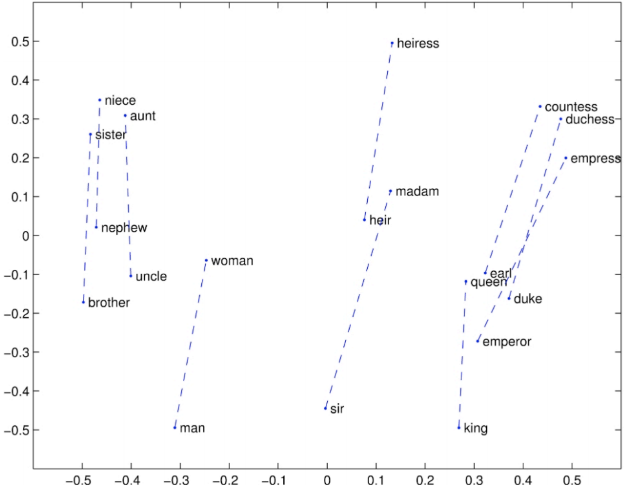
Visualization

- Typical embedding dimensions are in the hundreds (e.g. 300)
- How can we visualize these embeddings?
- Simple way: project them in 2D with something like PCA

Visualization

- Typical embedding dimensions are in the hundreds (e.g. 300)
- How can we visualize these embeddings?
- Simple way: project them in 2D with something like PCA
- Most used: t-SNE (t-distributed stochastic neighbor embedding (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

Other Methods for Obtaining Word Embeddings

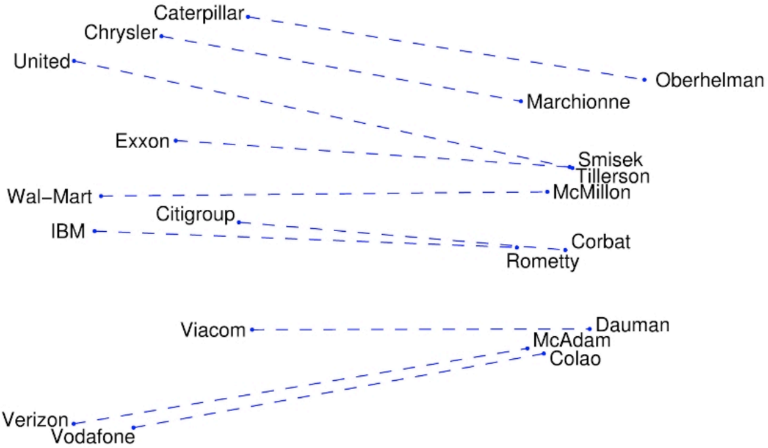
GloVe: Global Vectors for Word Representation (Pennington et al., 2014)

- `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

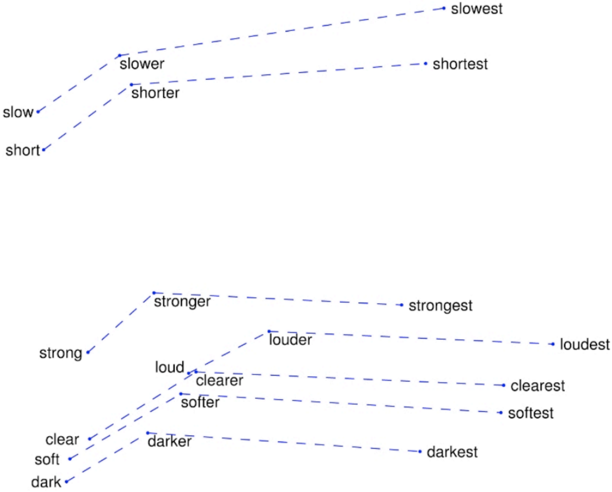
- `https://fasttext.cc` (from FAIR)
- open-source, free, lightweight library that allows users to learn text representations and text classifiers
- contains multi-lingual word vectors for 157 different languages

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

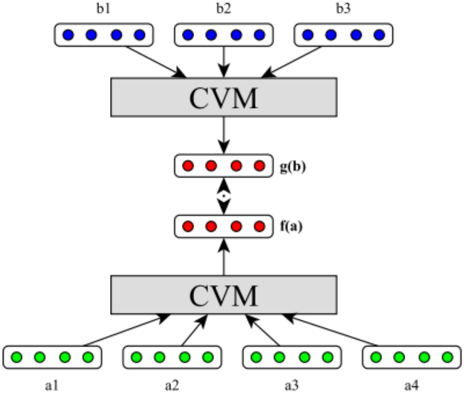


Figure 1: Model with parallel input sentences a and b. The model minimises the distance between the sentence level encoding of the bitext. Any composition functions (CVM) can be used to generate the compositional sentence level representations.

(From Hermann and Blunsom (2014).)

Cross-Lingual Word Embeddings

- Use a corpus of parallel sentences in two languages

Cross-Lingual Word Embeddings

- Use a corpus of parallel sentences in two languages
- Define a composition function to obtain a sentence representation given word embeddings

Cross-Lingual Word Embeddings

- 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 representions in the two languages to be similar

Cross-Lingual Word Embeddings

- 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)

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)

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)

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)

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 the context

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*.