IE 678 Deep Learning

02 – Feedforward Neural Networks

Part 1: Embeddings

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2024-1

From linear models to FNNs (1)

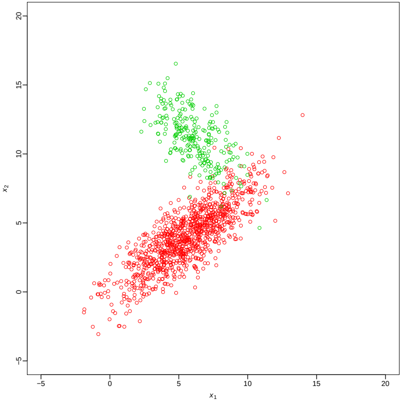
- Consider: prediction task with inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$
  - Goal: learn a function from $\mathcal{X}$ to $\mathcal{Y}$
- Simple approach: use a (generalized) **linear model**
  - Inputs must be real-valued feature vectors $\boldsymbol{x} \in \mathbb{R}^D$
  - Outputs are a real value (e.g., linear or logistic regression)
- Visually:

$$\boldsymbol{x} \in \mathbb{R}^D \longrightarrow \boxed{\text{Linear model}} \longrightarrow \hat{y} \in \mathbb{R}$$

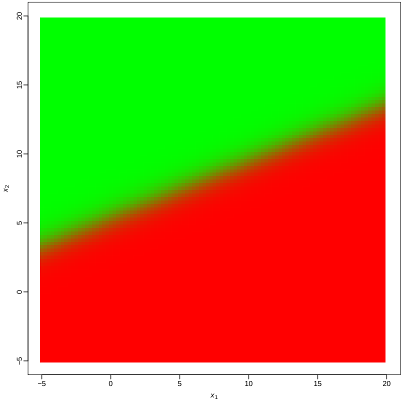
- Recall: $\hat{y} = \phi(\boldsymbol{w}^\top \boldsymbol{x} + b)$, where
  - $\boldsymbol{w} \in \mathbb{R}^D$ is a **weight vector** (one weight per feature, learned)
  - $b \in \mathbb{R}$ is a **bias term** (learned)
  - ϕ is a **mean function** (e.g., identity or logistic function)
- Problem: low representational capacity due to linearity assumption

Example: Logistic regression (from ML course)

Data ($\boldsymbol{x} \in \mathbb{R}^2$)

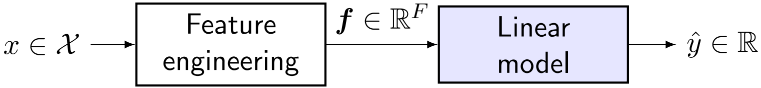


Prediction ($\hat{y} \in [0, 1]$)



From linear models to FNNs (2)

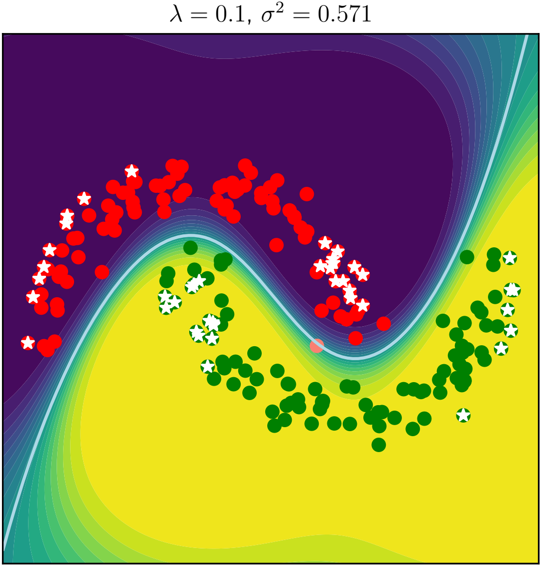
- Representational capacity can be addressed by feature engineering or using kernel methods
  - Allows to use arbitrary inputs spaces $x \in \mathcal{X}$ by mapping them to real-valued vectors
  - To do so, uses *pre-specified* feature extractor $f : \mathcal{X} \to \mathbb{R}^F$
- Visually:



- Problem: which feature extractor?
  - Key to good performance
  - Hard to get right (domain experts, extensive experimentation, ...)
  - To see this: can you write a suitable feature extractor for classifying images? (if not, see here)

Example: L1VM (from ML course)

L1VM, RBF kernel, logistic regression



From linear models to FNNs (3)

- DL methods can be interpreted as an approach to *learn* features
  - Input objects $x \in \mathcal{X}$ are transformed into dense, continuous, low-dimensional representations called **embeddings** $\boldsymbol{z} \in \mathbb{R}^Z$
  - Useful to represent complex objects (categorical data, textual data, graph data, tabular data, images, . . .)
  - Think: complex to work with objects, simple to work with embeddings
  - Useful **embedding space** = goal of **representation learning**
  - E = embedding dimensionality
- Visually:

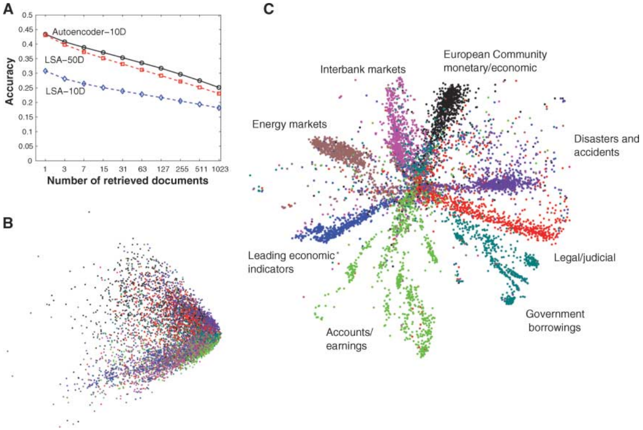
$$x \in \mathcal{X} \longrightarrow \boxed{\text{Deep learning}} \xrightarrow{\boldsymbol{z} \in \mathbb{R}^Z} \boxed{\text{Linear model}} \longrightarrow \hat{y} \in \mathbb{R}$$

- Key point: instead of engineering features manually, embeddings are learned from data $\rightarrow$ Main topic of this course
  - Embeddings also called: **latent code**, **distributed representations**
  - Embedding space also called: **latent space**

Example: Document embeddings

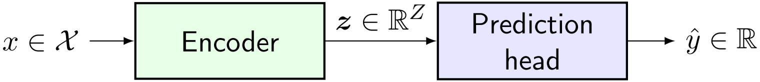
804414 newswire stories, inputs = per-document rel. frequencies of 2000 most common word stems ($x \in \mathbb{R}^{2000}$), shown here is 2D embedding ($z \in \mathbb{R}^2$) of two different methods (left: linear, right: autoencoder)

Fig. 4. (**A**) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (**B**) The codes produced by two-dimensional LSA. (**C**) The codes produced by a 2000-500-250-125-2 autoencoder.



Encoders and prediction heads

- Functions that transform objects $x \in \mathcal{X}$ to embeddings $\boldsymbol{z} \in \mathbb{R}^Z$ are known as **encoders**
- Functions that transform embeddings $\boldsymbol{z} \in \mathbb{R}^Z$ to predictions $y \in \mathbb{R}$ are known as **prediction heads**
  - Can be linear or more complex
  - Typically much simpler than encoder
  - E.g., when prediction head is logistic regression, then positive and negative instances are ideally linearly separable in embedding space
- Visually:



- Both encoder and prediction head are learned neural (sub)networks

Contrastive learning

- Embeddings can be used in other ways as well
- E.g., to compare objects, potentially across multiple modalities
  - Useful, for example, for zero- and few-shot prediction
  - Learned via a "contrastive learning" approach (more later)
- Example: CLIP embeddings for images and text

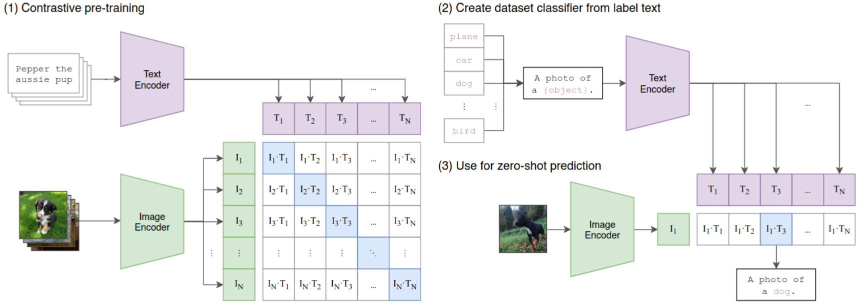
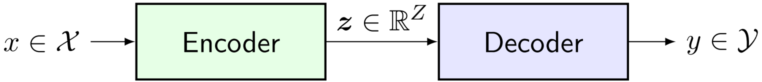


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

Structured prediction / deep generative models

- To handle more complex output spaces $\mathcal{Y}$, we may replace the prediction head by a component that "generates" output
- Functions that transform embeddings $\boldsymbol{z} \in \mathbb{R}^Z$ to (complex) outputs $y \in \mathcal{Y}$ are known as **decoders**
  - Note: in such models, embedding dimensionality Z may or may not depend on input x
- Visually:



Example: unCLIP (DALL-E 2)

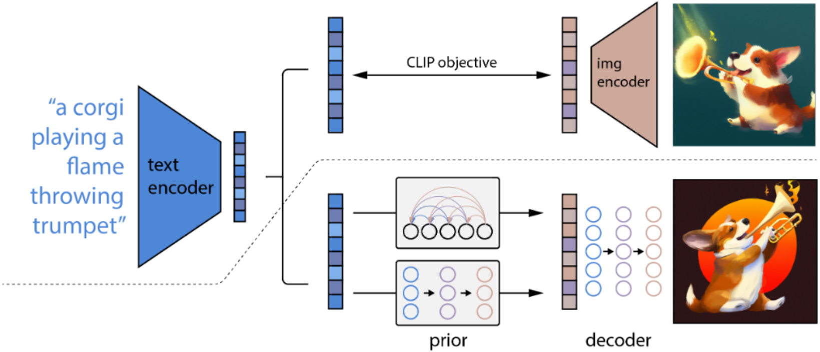


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.