Topic 5: autoencoders

Damon Wischik
What is an autoencoder?

A classifier
Input: labelled data \((X_n, Y_n)_{n=1..N}\)
Task: predict the output \(Y\) given input \(X\)

An autoencoder
Input: unlabelled data \((X_n)_{n=1..N}\)
Task: given an input, reconstruct it
Challenge: squeeze the data through a “bottleneck”
What’s the point in learning to recreate the input?

It can help with multitask / transfer / semi-supervised learning.

Train a neural network with two objectives:
(a) output the target label $Y$
(b) reproduce the input

- This is useful if labels are low entropy
e.g. sentiment classification of text.
  The “reproduce the input” objective (b) gives extra
  feedback, which helps backpropagation learn useful
  features.
- It’s also useful if you have lots of unlabelled data and
  only a little labelled data.
The heart of autoencoding

We hope it will learn a useful / meaningful latent representation.

Surely, if it didn’t learn a good representation, it’d have no chance of reconstructing the input from just a few variables!

A 4-dimensional representation

\{'digit': 6, 'slant': UPRIGHT, 'weight': MEDIUM, 'style': LOOSE\}
What sort of representations does it actually learn?

MNIST image | 4-dimensional representation | reconstruction
---|---|---
[6] | [1.4400] | [6]
1.5164 | 0.3757 | [4]
3.2569 | |

Source images

Reconstructions after 0.1 epochs

Reconstructions after 2 epochs

Reconstructions after 3 epochs

PCA plot showing the latent representations

colour = true digit
If we had a good representation, we could ...

- Pick a random $Z$, and decode. 
  *This should let us synthesize entirely new images.*

- Take two source images $X_1$ and $X_2$, encode to get $Z_1$ and $Z_2$, let $Z = (1 - \lambda)Z_1 + \lambda Z_2$, and decode $Z$. 
  *This should generate a smooth interpolation between the two inputs, where each intermediate looks “nice”.*

- Take a source image $X$, encode it to get $Z$, then vary the “digit” field of $Z$ and decode. 
  *This should give a family of digits with the same handwriting.*
Autoencoders are a tool for dimension reduction

- It’s **easier to train** a supervised learner from dimension-reduced features than from the raw dataset.
- The reduced dimensions are **meaningful axes** for our dataset; this is useful for interpolation etc.
- We can **synthesize new data**, by sampling randomly in the reduced-dimension space.

None of this works well off-the-shelf (hence the papers we will study).

And in fact the entire premise is dodgy.

*We haven’t specified a proper evaluation criterion. Without this we can’t compare models, or tune hyperparameters; we’re just blindly hacking.*
How should we validate an autoencoder? A thought experiment...

- In training, the aim is to minimize the reconstruction loss \( \mathbb{E}_{X \sim \text{train}} L(X, \hat{X}) \).
- The obvious way to validate is to run the network on unseen data (the holdout/validation dataset), and measure the reconstruction loss \( \mathbb{E}_{X \sim \text{test}} L(X, \hat{X}) \).
- But consider a super-intelligent autoencoder, which has learnt to encode input pixel \( i \) into bit \( i \) of the latent variable \( Z \in \mathbb{R} \). This autoencoder is surely not what we want — but it will score perfectly.

Input: unlabelled data \( (X_n)_{n=1..N} \)
Reconstruction loss metric: \( L(X, \hat{X}) \)
Autoencoders are a tool for dimension reduction

- It’s **easier to train** a supervised learner from dimension-reduced features than from the raw dataset.
- The reduced dimensions are **meaningful axes** for our dataset; this is useful for interpolation etc.
- We can **synthesize new data**, by sampling randomly in the reduced-dimension space.

None of this works well off-the-shelf (hence the papers we will study).

**Just like PCA!**

*Does PCA give us any insight into the problem of validation?*
Given a collection of points $X_1, ..., X_N \in \mathbb{R}^d$
PCA looks for a linear subspace of dimension $e < d$ to represent the data.

PCA is an autoencoder.
- It encodes $X \in \mathbb{R}^d$ into $Z \in \mathbb{R}^e$
- The decoder positions the linear subspace $\mathbb{R}^e$ within $\mathbb{R}^d$
- PCA seeks to minimize mean square error

This picture depicts dimension reduction from $\mathbb{R}^2$ to $\mathbb{R}^1$.
- With $e = d$ we’d get perfect reconstruction (but no dimension reduction)
- There are hacks to pick a useful $e < d$ ...
PCA only looks for linear subspaces. It is incapable of overfitting (as long as $e < d$).

If we allow nonlinear enc and dec, surely we can describe the data better.

Too much capacity $\rightarrow$ overfitting.
In the story of autoencoders, there are three overlapping challenges.

1. Denoising
2. Fair rep.
3. VAE
4. CVAE
5. $\beta$VAE

Formulate AE so that we can validate / compare models.
Coerce AE into producing meaningful representations.
Be good at synthesizing new data.
<table>
<thead>
<tr>
<th>Schedule</th>
<th>Assessment</th>
<th>Arrangements</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 January (1 hour)</td>
<td>▪ participation ≈ 5%</td>
<td>There is a Cambridge Gitlab repository for this course, with a toy MNIST example in Pytorch.</td>
</tr>
<tr>
<td></td>
<td>▪ presentation ≈ 15%</td>
<td>Presenters will contribute working code.</td>
</tr>
<tr>
<td></td>
<td>▪ project report 70%</td>
<td>Participants should also contribute issues / pull requests / code. (This gives you participation marks.)</td>
</tr>
<tr>
<td>27 January (1 hour)</td>
<td>Presenters, please chat with me the Friday before your presentation.</td>
<td></td>
</tr>
<tr>
<td>3 February (1 hour)</td>
<td>You should all read the papers, try the code, and participate in the discussion.</td>
<td></td>
</tr>
<tr>
<td>10 February (2 hours)</td>
<td>(Please introduce yourself. I’ll record for marking purposes.)</td>
<td></td>
</tr>
</tbody>
</table>