Getting creative with autoencoders

Tijmen Tieleman, CEO & CTO at minds.ai

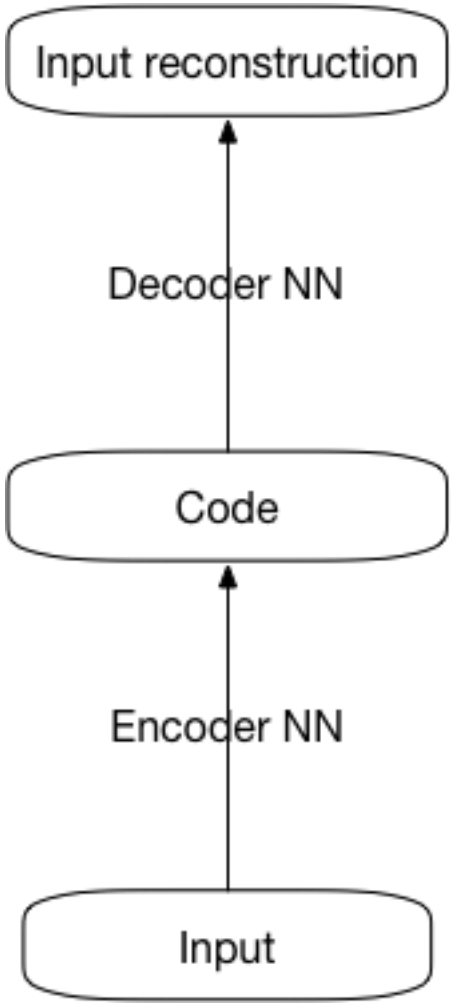
WiSSAP 2019, Trivandrum

Overview

- This is a fairly general talk
  - Application-agnostic
  - General ideas, rather than recent research
- This is a fairly specialized talk
  - It’s about Deep Learning
  - More specifically: representation learning
  - More specifically: unsupervised representation learning
  - More specifically: autoencoders

Autoencoders

- Training objective: reconstruct the input
- Components: encoder and decoder
- Usual purpose: representation learning
- Generative view: variational autoencoders
  - Components: encoder and ~~decoder~~ generator
- Flexible
  - Get creative with encoder and/or generator
  - Interpretable codes can be arranged
    - Code units are only partially hidden units

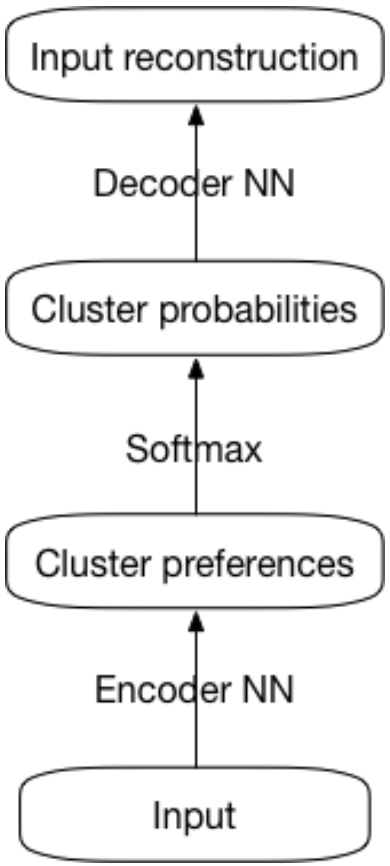


Clustering

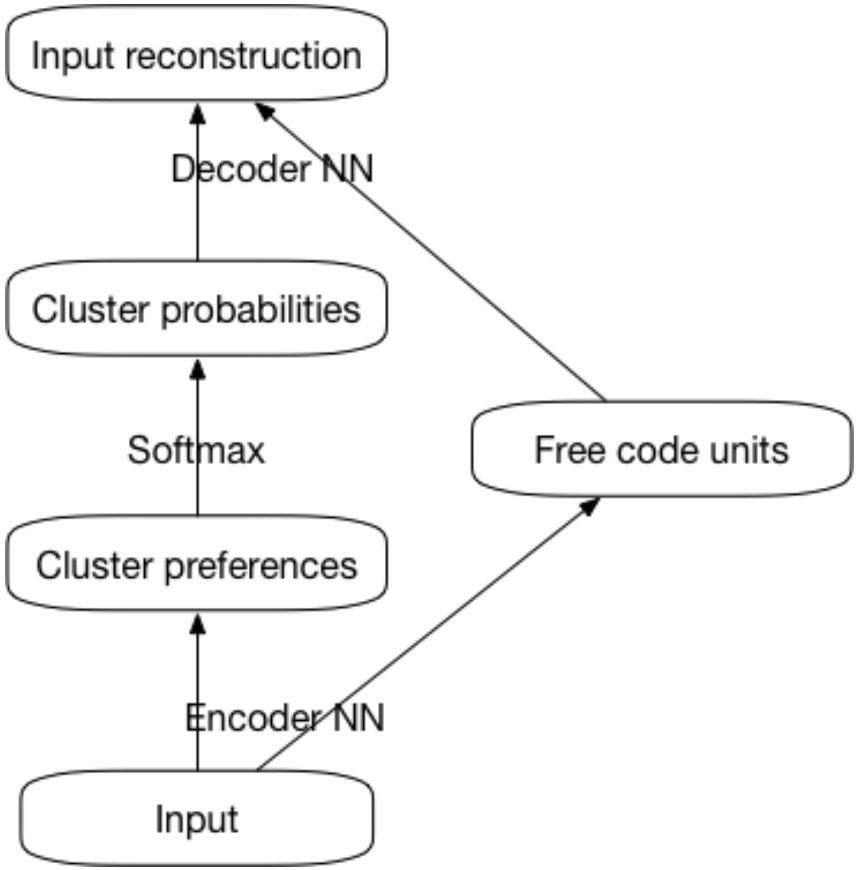
- “k means”
- K means as an (odd) autoencoder
  - Encoder (not NN)
  - Generator (constant for each cluster)
- Implicit assumption
- More powerful generator: non-trivial covariance
  - Different implicit assumption

Clustering autoencoder

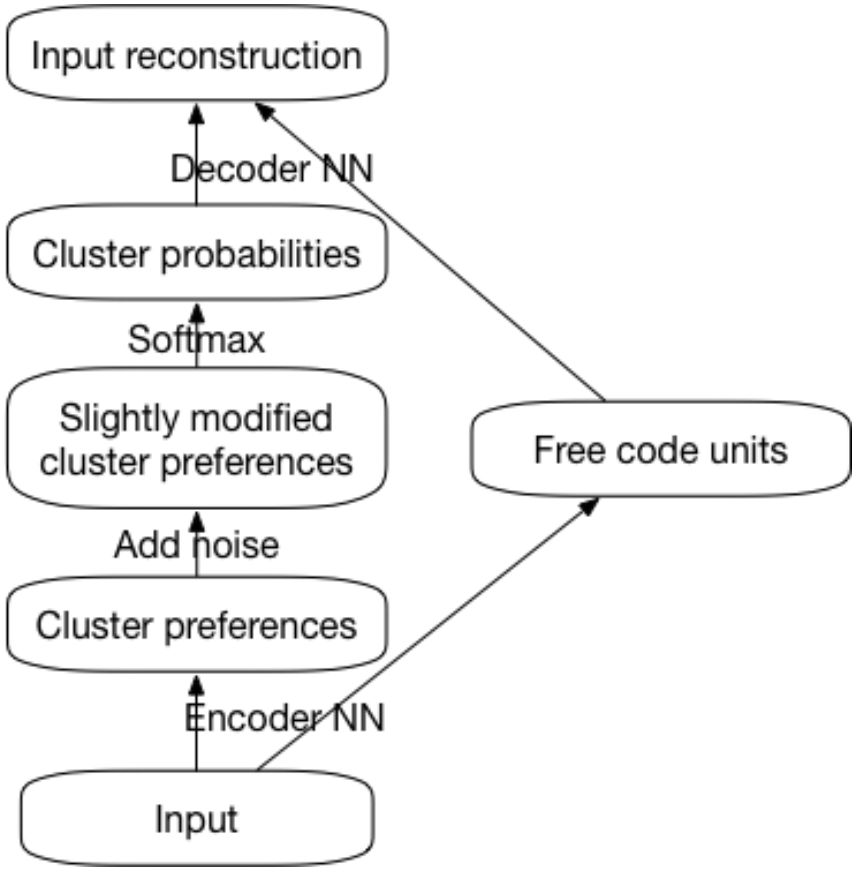
Naïve



With free code units



With free code units and discretization



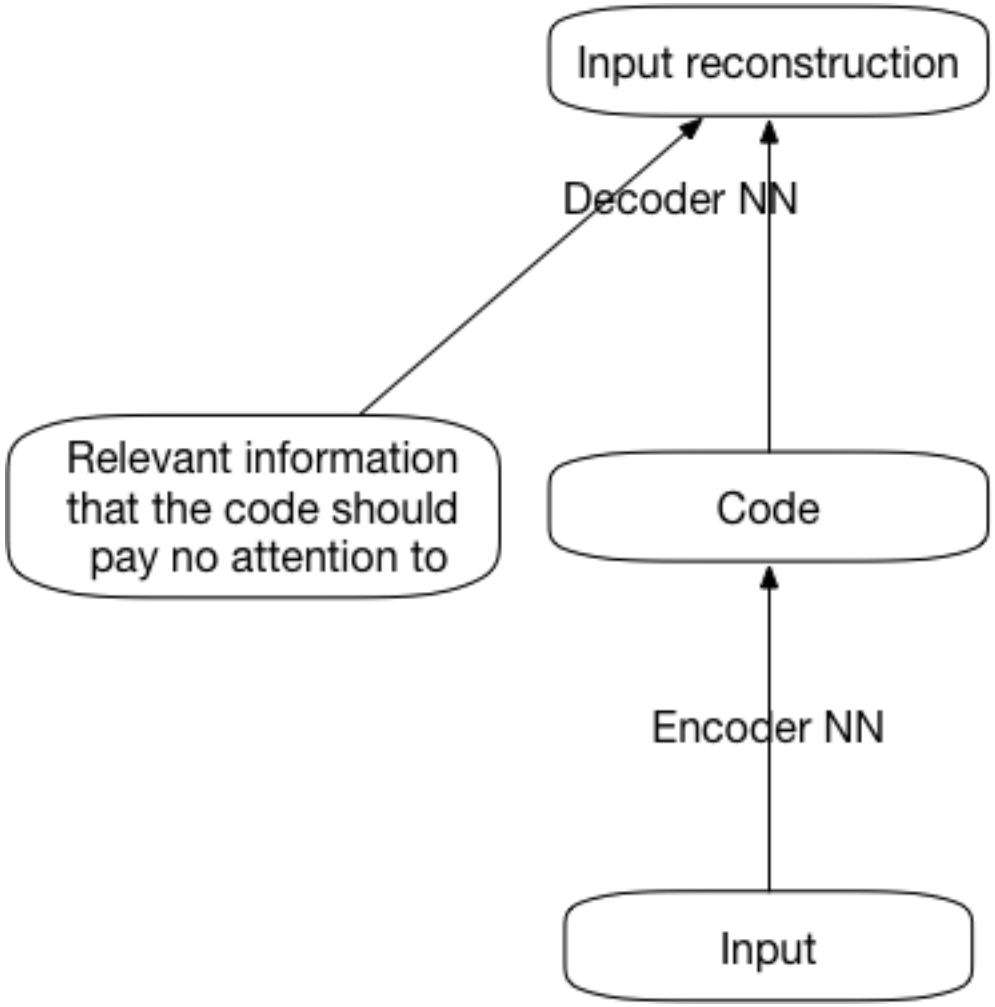
Intermission: a few minutes for questions about the past few slides.

Getting creative with autoencoders

- Clustering autoencoder
- Ignoring autoencoder
- Binarizing autoencoder
- Biased autoencoder
- Computer graphics autoencoder
- Independent features autoencoder
- Etc
- Feel free to combine

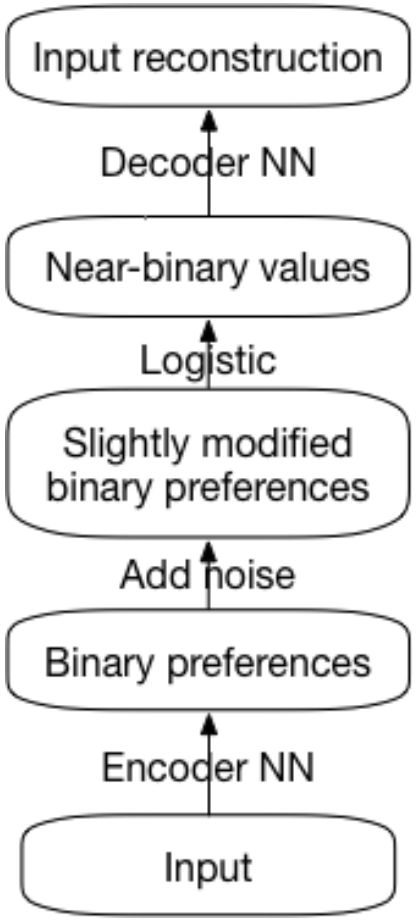
Ignoring autoencoder

- Goal: code that ignores some information
  - E.g. ignoring speaker information
  - E.g. ignoring the spoken text
- Application: focusing on essentials
- Application: domain adaptation
- Application: anti-discrimination
- Application: reconstructing with different information
- Method 1: provide that information to the decoder
- Method 2: insist that the to-be-hidden information cannot be reconstructed



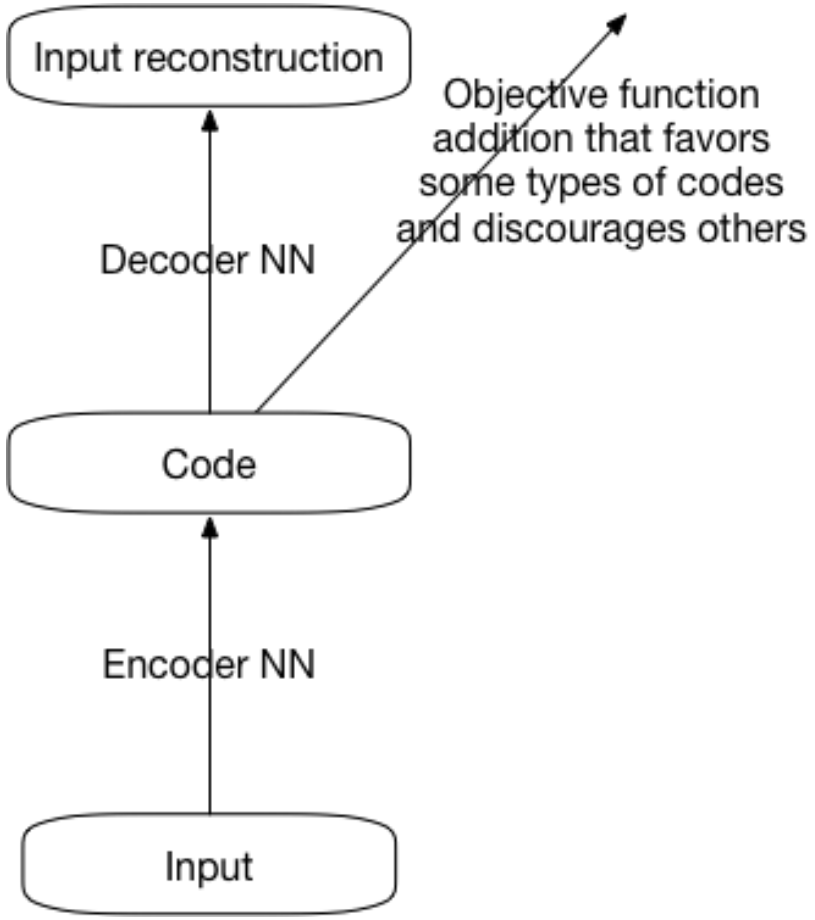
Binarizing autoencoder

- Goal: binary code
- Application: ”semantic hashing”
- Application: extreme compression (lossy)
- Method: use discretization noise on a bank of logistic code units



Biased autoencoder

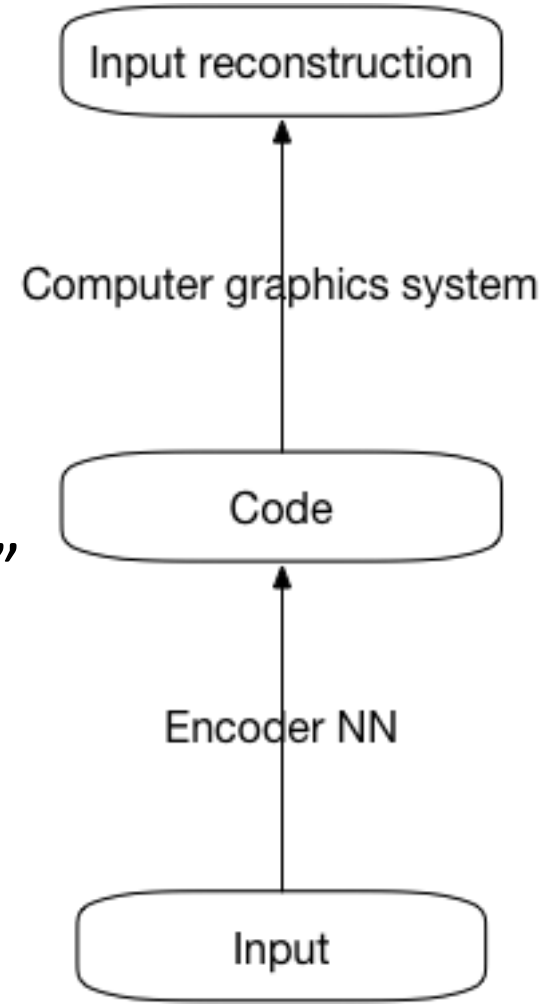
- Goal: codes that tend to be of certain kinds
  - E.g. a softmax with 2 commonly used units and 10 rarely used ones.
  - E.g. binarizing and preferring sparsity
- Method: simply push in the objective



Intermission: a few minutes for questions about the past few slides.

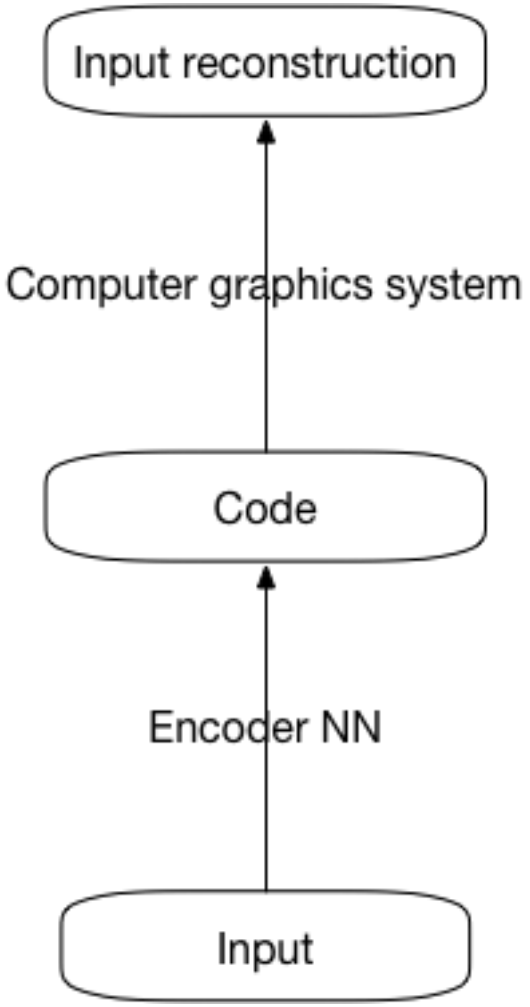
Computer graphics autoencoder

- More generally: domain-specific generator
  - E.g. audio synthesis autoencoder
- Goal: highly interpretable codes
- Goal: help the encoder understand the nature of images
- Application: reverse rendering, for “image parsing” or as preparation for generating image variants
- Application: using the better encoder as a good featurizer
- Method: replace the generator



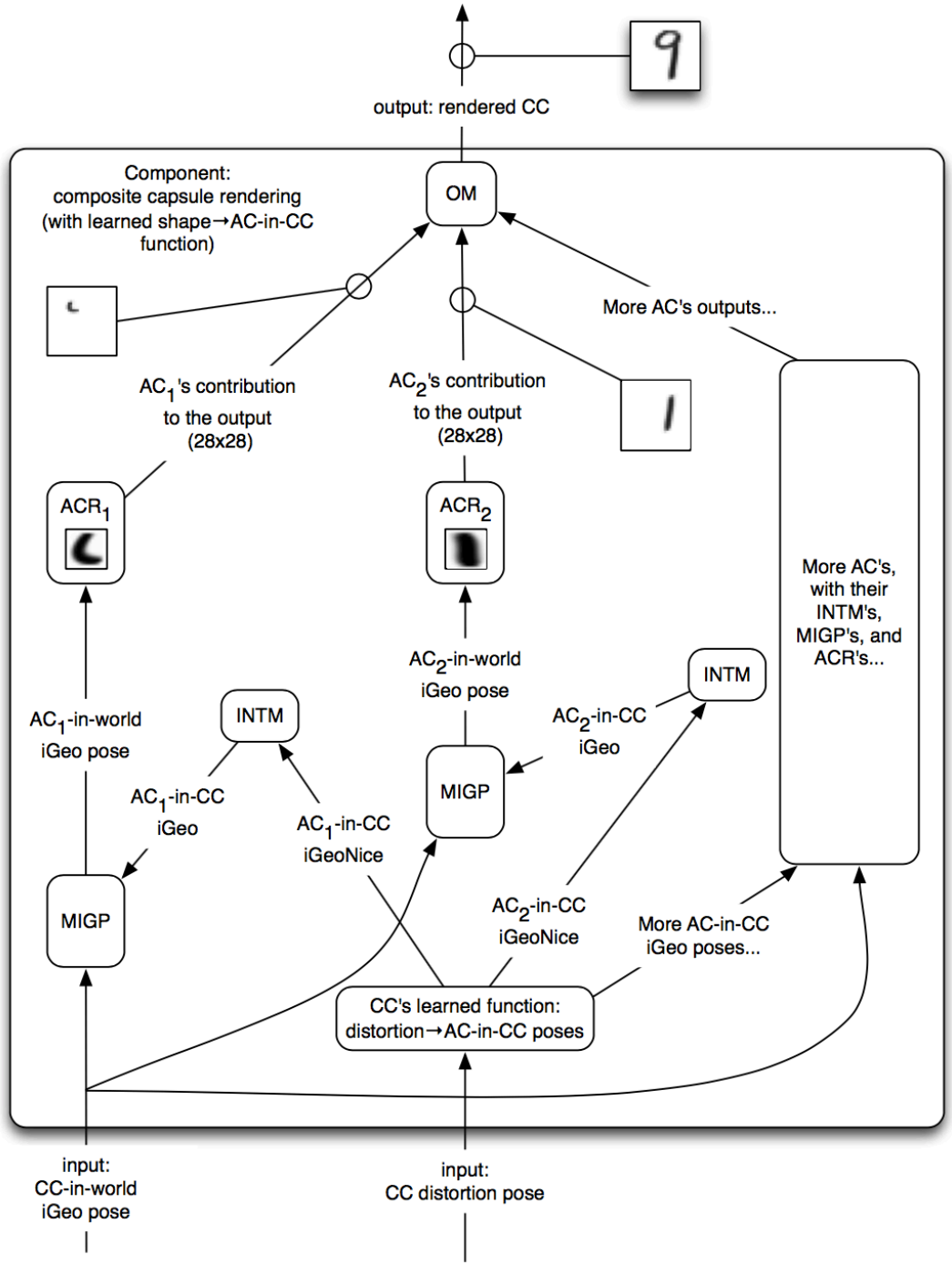
Computer graphics as generator

- Problem: it has to be differentiable
- Solution 1: implement a reduced CG system in TensorFlow
  - Weakness: “reduced” is unfortunate
  - Nice: it can contain learnables
- Solution 2: learn a second NN to approximate a real CG system; backpropagate through the approximation
  - Weakness: approximations are a pain
  - Nice: it has the potential for full CG
  - The codes still have to be continuous



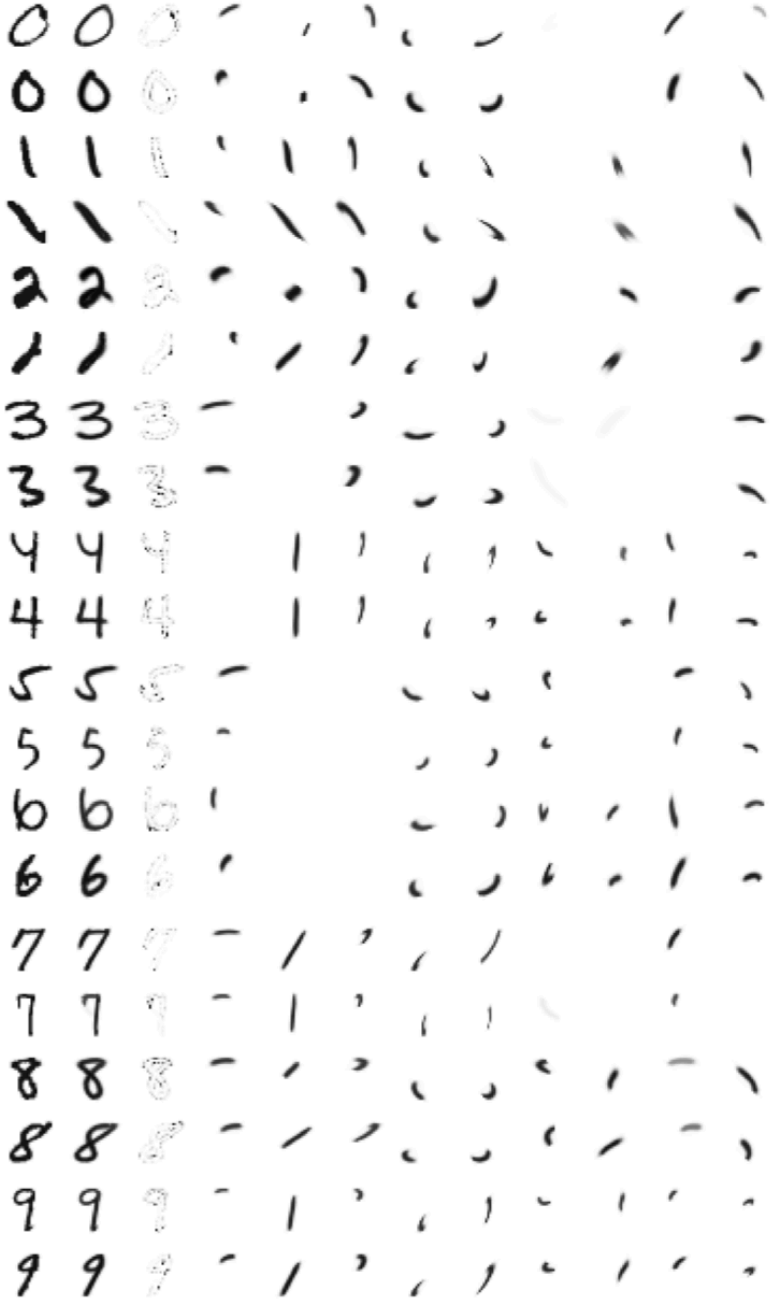
PoC: graphics for MNIST

- Uses “capsules”
- Learned constant: small image templates
- Learned function: free code unit values → affine transformation for the templates
- Hard-coded function: apply affine transformations to the templates
- Affine transformed templates are added together to create final output



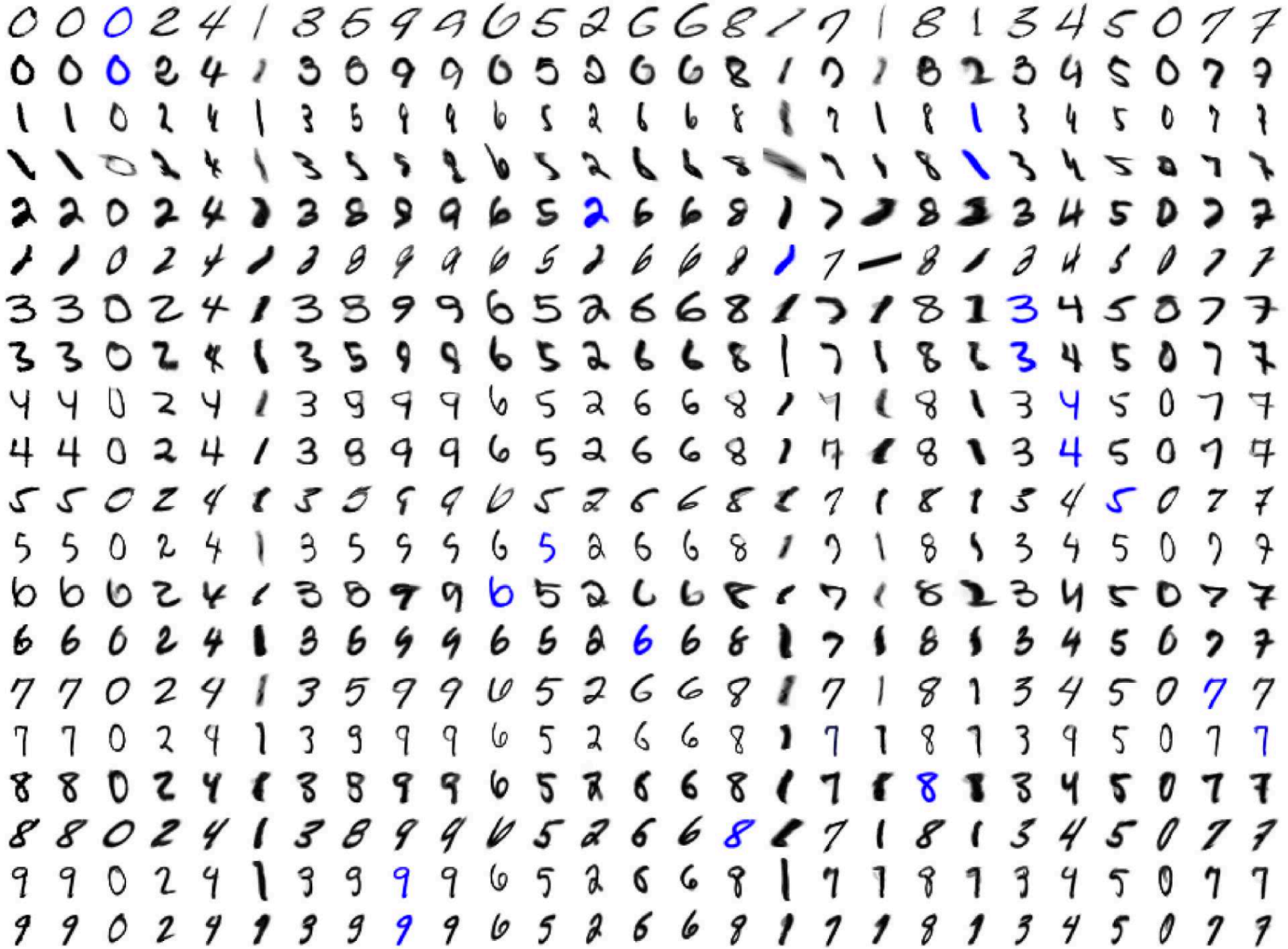
PoC: graphics for MNIST

- Uses “capsules”
- Learned constant: small image templates
- Learned function: free code unit values → affine transformation for the templates
- Hard-coded function: apply affine transformations to the templates
- Affine transformed templates are added together to create final output



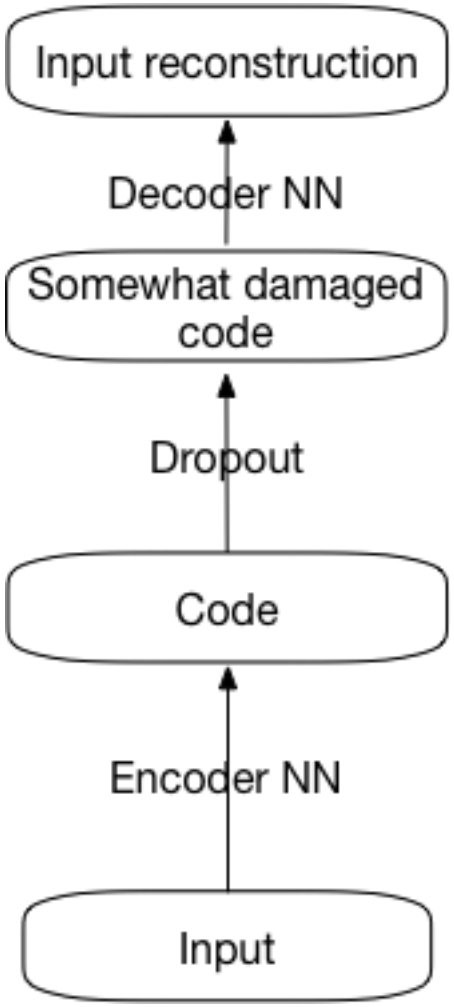
Clustering MNIST

- K means clustering wouldn’t work well: the generator is too primitive
  - We need a more powerful generator
  - It has to understand that a shifted image is still about the same image
  - Let’s use this computer graphics generator
- Unsupervised clustering into 25 clusters
- Good classification accuracy with just 25 labeled cases



Independent features autoencoder

- Goal: code components that are somewhat independently interpretable
- Application: identifying the patterns in the data
- Application: generating data variants with a higher chance of meaningful results
- Method: apply dropout to the code components



Independent features

- This focuses on one image (see the central column)
- This shows data variants by varying one feature at a time
- The 8 features are clearly somewhat independent
- The variants are meaningful

← Feature value →

↓ Feature index →

Tools for crafting specialized autoencoders

- Impose some structure on the code
  - E.g. clustering autoencoder, computer graphics autoencoder
- Apply deterministic editing to the code
  - E.g. ignoring autoencoder
- Apply random editing to the code
  - E.g. binarizing autoencoder, clustering autoencoder, independent features autoencoder
- Apply an objective function addition to the code
  - E.g. sparse binarizing autoencoder
- Hard-code (part of) the generator
  - E.g. computer graphics autoencoder

Conclusion

- Creating custom specialized autoencoders is doable
  - There is a collection of basic tools for this
  - This can improve interpretability and learning power
- Next stage for DL: combining learnable and hard-coded components
  - Learnable components provide “intuition”
  - Hard-coded components provide the power of computation
  - Either alone is quite limited
  - AlphaGo has hard-coded as main() and learnable as subroutine
  - The CG autoencoder has learnable as main() and hard-coded as subroutine

Thank you