Image retrieval using multilayer neural networks

Geoffrey Hinton
Overview

• An efficient way to train a multilayer neural network to extract a low-dimensional representation.

• Document retrieval *(published work with Russ Salakhutdinov)*
  - How to model a bag of words with an RBM
  - How to learn binary codes
  - Semantic hashing: retrieval in no time

• Image retrieval *(published work with Alex Krizhevsky)*
  - How good are 256-bit codes for retrieval of small color images?
  - Ways to use the speed of semantic hashing for much higher-quality image retrieval *(work in progress).*
Deep Autoencoders
(with Ruslan Salakhutdinov)

• They always looked like a really nice way to do non-linear dimensionality reduction:
  – But it is very difficult to optimize deep autoencoders using backpropagation.

• We now have a much better way to optimize them:
  – First train a stack of 4 RBM’s
  – Then “unroll” them.
  – Then fine-tune with backprop.
A comparison of methods for compressing digit images to 30 real numbers.
Compressing a document count vector to 2 numbers

We train the autoencoder to reproduce its input vector as its output. This forces it to compress as much information as possible into the 2 real numbers in the central bottleneck.

These 2 numbers are then a good way to visualize documents.

We need a special type of RBM to model counts.
First compress all documents to 2 numbers using a type of PCA.
Then use different colors for different document categories.
First compress all documents to 2 numbers. Then use different colors for different document categories.
The replicated softmax model: How to modify an RBM to model word count vectors

- **Modification 1:** Keep the binary hidden units but use “softmax” visible units that represent 1-of-
- **Modification 2:** Make each hidden unit use the same weights for all the visible softmax units.
- **Modification 3:** Use as many softmax visible units as there are non-stop words in the document.
  - So its actually a family of different-sized RBMs that share weights. Its not a single generative model.
- **Modification 4:** Multiply each hidden bias by the number of words in the document (not done in our earlier work)
- The replicated softmax model is much better at modeling bags of words than LDA topic models (in NIPS 2009)
The replicated softmax model

All the models in this family have 5 hidden units. This model is for 8-word documents.
Finding real-valued codes for retrieval

• Train an auto-encoder using 10 real-valued units in the code layer.

• Compare with Latent Semantic Analysis that uses PCA on the transformed count vector.

• Non-linear codes are much better.
Retrieval performance on 400,000 Reuters business news stories.

The graph shows the accuracy of three methods: Autoencoder-10D, LSA-50D, and LSA-10D, as the number of retrieved documents increases from 1 to 1023.
Finding binary codes for documents

- Train an auto-encoder using 30 logistic units for the code layer.
- During the fine-tuning stage, add noise to the inputs to the code units.
  - The “noise” vector for each training case is fixed. So we still get a deterministic gradient.
  - The noise forces their activities to become bimodal in order to resist the effects of the noise.
  - Then we simply threshold the activities of the 30 code units to get a binary code.
Using a deep autoencoder as a hash-function for finding approximate matches

“supermarket search”
Another view of semantic hashing

• Fast retrieval methods typically work by intersecting stored lists that are associated with cues extracted from the query.
• Computers have special hardware that can intersect 32 very long lists in one instruction.
  – Each bit in a 32-bit binary code specifies a list of half the addresses in the memory.
• Semantic hashing uses machine learning to map the retrieval problem onto the type of list intersection the computer is good at.
How good is a shortlist found this way?

• Russ has only implemented it for a million documents with 20-bit codes --- but what could possibly go wrong?
  – A 20-D hypercube allows us to capture enough of the similarity structure of our document set.

• The shortlist found using binary codes actually improves the precision-recall curves of TF-IDF.
  – Locality sensitive hashing (the fastest other method) is much slower and has worse precision-recall curves.
Semantic hashing for image retrieval

• Currently, image retrieval is typically done by using the captions. Why not use the images too?
  – Pixels are not like words: individual pixels do not tell us much about the content.
  – Extracting object classes from images is hard.

• Maybe we should extract a real-valued vector that has information about the content?
  – Matching real-valued vectors in a big database is slow and requires a lot of storage

• Short binary codes are easy to store and match
A two-stage method

• First, use semantic hashing with 30-bit binary codes to get a long “shortlist” of promising images.
• Then use 256-bit binary codes to do a serial search for good matches.
  – This only requires a few words of storage per image and the serial search can be done using fast bit-operations.
• But how good are the 256-bit binary codes?
  – Do they find images that we think are similar?
Some depressing competition

• Inspired by the speed of semantic hashing, Weiss, Fergus and Torralba (NIPS 2008) used a very fast spectral method to assign binary codes to images.
  – This eliminates the long learning times required by deep autoencoders.
• They claimed that their spectral method gave better retrieval results than training a deep auto-encoder using RBM’s.
  – But they could not get RBM’s to work well for extracting features from RGB pixels so they started from 384 GIST features.
  – This is too much dimensionality reduction too soon.
A comparison of deep auto-encoders and the spectral method using 256-bit codes
(Alex Krizhevsky)

• Train auto-encoders “properly”
  – Use Gaussian visible units with fixed variance. Do not add noise to the reconstructions.
  – Use a cluster machine or a big GPU board.
  – Use a lot of hidden units in the early layers.
• Then compare with the spectral method
  – The spectral method has no free parameters.
• Also compare with Euclidean match in pixel space
Krizhevsky’s deep autoencoder

The encoder has about 67,000,000 parameters.

It takes a few GTX 285 GPU days to train on two million images.

There is no theory to justify this architecture.

256-bit binary code
The next step

• Implement the semantic hashing stage for images.

• Check that a long shortlist still contains many good matches.
  – It works OK for documents, but they are very different from images.
  – Losing some recall may be OK. People don’t miss what they don’t know about.
An obvious extension

• Use a multimedia auto-encoder that represents captions and images in a single code.
  – The captions should help it extract more meaningful image features such as “contains an animal” or “indoor image”

• RBM’s already work much better than standard LDA topic models for modeling bags of words.
  – So the multimedia auto-encoder should be
    + a win (for images)
    + a win (for captions)
    + a win (for the interaction during training)
A less obvious extension

• Semantic hashing gives incredibly fast retrieval but it's hard to go much beyond 32 bits.
• We can afford to use semantic hashing several times with variations of the query and merge the shortlists
  – It's easy to enumerate the Hamming ball around a query image address in ascending address order, so merging is linear time.
• Apply many transformations to the query image to get transformation independent retrieval.
  – Image translations are an obvious candidate.
Summary

• Restricted Boltzmann Machines provide an efficient way to learn a layer of features without any supervision.
  – Many layers of representation can be learned by treating the hidden states of one RBM as the data for the next.

• This allows us to learn very deep nets that extract short binary codes for unlabeled images or documents.
  – Using 32-bit codes as addresses allows us to get approximate matches at the speed of hashing.

• Semantic hashing is fast enough to allow many retrieval cycles for a single query image.
  – So we can try multiple transformations of the query.
A more interesting extension

• Computer vision uses images of uniform resolution.
  – Multi-resolution images still keep all the high-resolution pixels.

• Even on 32x32 images, people use a lot of eye movements to attend to different parts of the image.
  – Human vision copes with big translations by moving the fixation point.
  – It only samples a tiny fraction of the image at high resolution. The “post-retinal” image has resolution that falls off rapidly outside the fovea.
  – With less “neurons” intelligent sampling becomes even more important.
How to perceive a big picture with a small brain

• Even a human brain cannot afford high-resolution everywhere.
  – By limiting the input we make it possible to use many layers of dense features intelligently.

• For fine discrimination that requires high-resolution in several different places we must integrate over several fixations.

A much better “retina”.
A more human metric for image similarity

• Two images are similar if fixating at point X in one image and point Y in the other image gives similar post-retinal images.

• So use semantic hashing on post-retinal images.
  – The address space is used for post-retinal images and each address points to the whole image that the post-retinal image came from.
  – So we can accumulate similarity over multiple fixations.

• The whole image addresses found after each fixation have to be sorted to allow merging 😞
Starting from a better input representation

• First learn a good model for object recognition that can deal with multiple objects in the same image.

• Then use the outputs of the last hidden layer as the inputs to a deep autoencoder.

• This should work really well.
  – Euclidean distance on the activities in the last hidden layer already works extremely well.
Euclidean nearest neighbors using the 4096 activities in the last hidden layer