13. Autoencoders

Recall: Encoder Networks

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>01000</td>
<td>01000</td>
</tr>
<tr>
<td>00100</td>
<td>00100</td>
</tr>
<tr>
<td>00010</td>
<td>00010</td>
</tr>
<tr>
<td>00001</td>
<td>00001</td>
</tr>
</tbody>
</table>

- identity mapping through a bottleneck
- also called N–M–N task
- used to investigate hidden unit representations

Autoencoder Networks

- output is trained to reproduce the input as closely as possible
- activations normally pass through a bottleneck, so the network is forced to compress the data in some way
- like the RBM, Autoencoders can be used to automatically extract abstract features from the input

Outline

- Autoencoder Networks (14.1)
- Regularized Autoencoders (14.2)
- Stochastic Encoders and Decoders (14.4)
- Generative Models
- Variational Autoencoders (20.10.3)
Autoencoder Networks

Autoencoder as Pretraining

- after an autoencoder is trained, the decoder part can be removed and replaced with, for example, a classification layer
- this new network can then be trained by backpropagation
- the features learned by the autoencoder then serve as initial weights for the supervised learning task

Greedy Layerwise Pretraining

- Autoencoders can be used as an alternative to Restricted Boltzmann Machines, for greedy layerwise pretraining.
- An autoencoder with one hidden layer is trained to reconstruct the inputs. The first layer (encoder) of this network becomes the first layer of the deep network.
- Each subsequent layer is then trained to reconstruct the previous layer.
- A final classification layer is then added to the resulting deep network, and the whole thing is trained by backpropagation.

Avoiding Trivial Identity

- if there are more hidden nodes than inputs (which often happens in image processing) there is a risk the network may learn a trivial identity mapping from input to output
- we generally to avoid this by introducing some form of regularization

If the encoder computes $z = f(x)$ and the decoder computes $g(f(x))$ then we aim to minimize some distance function between $x$ and $g(f(x))$

$$E = L(x, g(f(x)))$$
Regularized Autoencoders (14.2)

- sparse autoencoders
- autoencoders with dropout at hidden layer(s)
- contractive autoencoders
- denoising autoencoders

Sparse Autoencoder (14.2.1)

- one way to regularize an autoencoder is to add a penalty term to the cost function, based on the hidden unit activations
- this is analogous to the weight decay term we previously used for supervised learning
- one popular choice is to penalize the sum of the absolute values of the activations in the hidden layer
  \[ E = L(x, g(f(x))) + \lambda \sum_i |h_i| \]
- this is sometimes known as L1-regularization (because it involves the absolute value rather than the square); it can encourage some of the hidden units to go to zero, thus producing a sparse representation

Contractive Autoencoder (14.2.3)

- another popular penalty term is the L2-norm of the derivatives of the hidden units with respect to the inputs
  \[ E = L(x, g(f(x))) + \lambda \sum_i \|\nabla_x h_i\|^2 \]
- this forces the model to learn hidden features that do not change much when the training inputs x are slightly altered

Denoising Autoencoder (14.2.2)

Another regularization method, similar to contractive autoencoder, is to add noise to the inputs, but train the network to recover the original input

repeat:
  sample a training item \(x^{(i)}\)
  generate a corrupted version \(\tilde{x}\) of \(x^{(i)}\)
  train to reduce \(E = L(x^{(i)}, g(f(\tilde{x})))\)
end
Generative Models

- Sometimes, as well as reproducing the training items \( \{x^{(i)}\} \), we also want to be able to use the decoder to generate new items which are of a similar “style” to the training items.
  - squared error assumes an underlying Gaussian distribution, whose mean is the output of the network
  - cross entropy assumes a Bernoulli distribution, with probability equal to the output of the network
  - softmax assumes a Boltzmann distribution

Stochastic Encoders and Decoders (14.4)

- For autoencoders, the decoder can be seen as defining a conditional probability distribution \( p_\theta(x|z) \) of output \( x \) for a certain value \( z \) of the hidden or “latent” variables.
- In some cases, the encoder can also be seen as defining a conditional probability distribution \( q_\phi(z|x) \) of latent variables \( z \) based on an input \( x \).
- We have seen an example of this with the Restricted Boltzmann Machine, where \( q_\phi(z|x) \) and \( p_\theta(x|z) \) were Bernoulli distributions.

Cost Functions and Probability

- We saw previously how the loss (cost) function at the output of a feedforward neural network (with parameters \( \theta \)) can be seen as defining a probability distribution \( p_\theta(x) \) over the outputs. We then train to maximize the log of the probability of the target values.
  - squared error assumes an underlying Gaussian distribution, whose mean is the output of the network
  - cross entropy assumes a Bernoulli distribution, with probability equal to the output of the network
  - softmax assumes a Boltzmann distribution

Variational Autoencoder (20.10.3)

Instead of producing a single \( z \) for each \( x^{(i)} \), the encoder (with parameters \( \phi \)) can be made to produce a mean \( \mu_{z|x^{(i)}} \) and standard deviation \( \Sigma_{z|x^{(i)}} \). This defines a conditional (Normal) probability distribution \( q_\phi(z|x^{(i)}) \)

We then train the system to maximize

\[
E_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)}|z)] - D_{\text{KL}}(q_\phi(z|x^{(i)}) \parallel p(z))
\]

- the first term enforces that any sample \( z \) drawn from the conditional distribution \( q_\phi(z|x^{(i)}) \) should, when fed to the decoder, produce something approximating \( x^{(i)} \)
- the second term encourages \( q_\phi(z|x^{(i)}) \) to approximate \( p(z) \)
- in practice, the distributions \( q_\phi(z|x^{(i)}) \) for various \( x^{(i)} \) will occupy complementary regions within the overall distribution \( p(z) \)
Variational Autoencoder produces reasonable results
- tends to produce blurry images
- often end up using only a small number of the dimensions available to $z$

References:
- http://kvfrans.com/variational-autoencoders-explained/