CPSC 440: Machine Learning

Autoencoders
Winter 2022
Autoencoders are neural networks with same input and output.
- Includes a bottleneck layer: with dimension ‘k’ smaller than input ‘d’.
- First layers “encode” the input into bottleneck.
- Last layers “decode” the bottleneck into a (hopefully valid) input.

This is unsupervised, and is a non-linear generalization of PCA.
(I am not showing the ‘h’ functions to keep the diagram simple.)
Encoder as Learning a Representation

• Consider the encoder part of the network:
  – Takes features ‘\(x^i\)’ and makes low-dimensional ‘\(z^i\)’.

• Ways you could use the encoder:
  – Use \(z^i\) as compressed input (reduce memory needed).
  – Set bottleneck size to 2, and plot the \(z^i\) to visualize the data.
  – Try to interpret what the bottleneck features \(z^i\) mean.
  – Use the \(z^i\) as features for supervised learning.
    • For the special case of PCA and regression with L2 loss, this is called “partial least squares”.
    – You could add a supervised ‘\(y^i\)’ to final layer of trained autoencoder, fit with SGD.
      • This is called “unsupervised pre-training”.
      • If you use unlabeled data to do this initialization, an example of “self-supervised” learning.
        – Usually it is easier to get a lot of unlabeled data than it is to get labeled data.
PCA vs. Deep Autoencoder (Document Data)

(these days I would recommend t-SNE for making visualizations like this)
Decoder as Generative Model

• Consider the **decoder** part of the network:
  – Takes low-dimensional ‘z’ and makes features ‘x’.

• Can be used for **outlier detection**:
  – Check distance to original features to detect outliers.

• Can be used to generate new data:
  – The ‘z’ close to training examples should generate new valid samples.
  – But this is **not density estimation**, since we are not modeling p(z) yet.
Font Manifold

• Going from **encoding to decoding** for different fonts:

![Image of font manifold and letter t]

  
  – The above was generated by a Gaussian process and not an autoencoder.
  – But the decoder part of autoencoders is trying to do something like this.
Neural Networks with Multiple Outputs

- Previous neural networks we have seen only have 1 output ‘y’.
- In autoencoders, we have ‘d’ outputs (one for each feature).

\[
\hat{x}_i = v_1^T h(w_3^T h(w_2^T h(w_1^T x)))
\]

\[
\hat{x}_j = v_2^T h(w_3^T h(w_2^T h(w_1^T x)))
\]

\[
\hat{x}_j = v_2^T h(w_3^T h(w_2^T h(w_1^T x)))
\]

- For training, we add up the loss across all ‘j’:

\[
f(w_1^T, w_2^T, v) = \frac{1}{2} \sum_{i=1}^{d} (\hat{x}_i - x_i)^2
\]

\[
f(w_1^T, w_2^T, v) = \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \log(1 + \exp(-\hat{x}_j x_j))
\]

- Fit with SGD (sampling random ‘i’), and usual deep learning tricks can be used.
  - Even though network has multiple outputs, ‘f’ is a scalar so AD works as before.
  - For images, may want to use convolution layers.
Denoising Autoencoders

• A common variation on autoencoders is denoising autoencoders:
  – Use “corrupted” inputs, and learn to reconstruct uncorrupted originals.
  – “Learn a model that removes the noise”. Easy to get lots of training data.
    • You can apply the model to denoise new images.
    • Do not necessarily need a “bottleneck” layer.
Image Colourization

• Video: https://www.youtube.com/watch?v=ys5nMO4Q0iY
Image Colourization

• Instead of noisy inputs, you use de-coloured inputs:

• Another application is **super-resolution**:
  – Learn to output a high-resolution image based on low-resolution images.

Next Topic: Multi-Label Classification
Motivation: Multi-Label Classification

• Consider multi-label classification:

\[
X = \left[ \begin{array}{c} \vdots \end{array} \right]_d \quad Y = \left[ \begin{array}{c} \vdots \end{array} \right]_n
\]

• Which of the ‘k’ objects are in this image?
  – There may be more than one “correct” class label.

http://image-net.org/challenges/LSVRC/2013/
Independent Classifier Approach

• One way to build a multi-label classifier:
  – Train a classifier for each label.
    • Train a neural network that predicts +1 if the image contains a dog, and -1 otherwise.
    • Train a neural network that predicts +1 if the image contains a cat, and -1 otherwise.
    • ...
  – To make predictions for the ‘k’ classes, concatenate predictions of the ‘k’ models.

• Can think of this as a “product of independent classifiers”.

• Drawbacks:
  – Lots of parameters: k*(number of parameters for base classifier).
  – Each classifier needs to “relearn from scratch”.
    • Each classifier needs to learn its own Gabor filters, how corners and light works, and so on.
    • A lot of visual features for “dog” might also help us predict “cat”.
Encoding-Decoding for Multi-Label Classification

- Multi-label classification with an encoding-decoding approach:
  - Input is connected to a hidden layer.
  - Hidden layer is connected to multiple output units.

- Prediction: compute hidden layer, compute activations, compute output:

\[ \hat{y} = V h(W_x) \]

- Number of parameters and cost is \( O(dm + mk) \) for ‘k’ classes and ‘m’ hidden units.
  - If we trained a separate network for each class, number of parameters and cost would be \( O(kdm) \) (for ‘W’ for each class)

- Might have multiple layers, convolution layers, and so on. And no need to have a “bottleneck” layer.
We usually assume that the classes are independent given last layer:

\[ p(y_1, y_2, \ldots, y_k | x, W, V) = p(y_1 | x, W, V)p(y_2 | x, W, V) \cdots p(y_k | x, W, V) \]

with:

\[ p(y_c = 1 | x, W, V) = \frac{1}{1 + \exp(-\theta_c h(Wx))} \quad \text{for} \quad c = 1, \ldots, k \]

Conditioned on features/parameters, this is ultimately a fancy product of Bernoullis model:

- \( p(y_1, y_2, \ldots, y_k | x, W, V) = p(y_1 | x, W, V)p(y_2 | x, W, V) \cdots p(y_k | x, W, V) \)
- This makes decoding and other inference problems easy: you do inference on each \( y_c \) independently.
Encoding-Decoding for Multi-Label Classification

• The **negative log-likelihood** we optimize for MLE:
  \[
  f(W, V) = \sum_{i=1}^{n} \sum_{c=1}^{k} \log \left( 1 + \exp \left( -y_i^c v_c^\top h(Wx_i') \right) \right)
  \]

• Use backpropagation or AD to compute gradient, train by SGD.
  – You randomly sample a training example ‘i’ and compute gradient for all labels.
  – The updates of ‘W’ lead to **features that are useful across classes**.
  – The updates of ‘V’ focus on getting the class labels right given the features.

• Important:
  – We assumed **independence of labels** given the last layer.
  – But the **last layer can reflect dependencies**.
    • If “dog” and “human” are frequently together, this should be reflected in the hidden layer.
      – For example, \( \theta_{human} \) might be higher when the features give a high value for \( \theta_{dog} \).
Pre-Training for Multi-Label Classification

• Consider a scenario where we get a new class label.
  – For example, we get new images that contain horses (not seen in training).

  ![Diagram of neural network](image)

• Instead of training from scratch, we could:
  – Add an extra set of weights $v_{k+1}$ to the final layer for the new class.
  – Train these weights with the encoding weights ‘$W$’ fixed.
    • This is a simple/convex logistic regression problem.
    • If we already have “features” that are good for many classes, we may be able to learn a new class with very-few training examples!
Pre-Training for Multi-Label Classification

• Using an existing network for new problems is called “pre-training”
  – Typically, we start with a network trained on a large dataset.
  – We use this network to give us features to fit a smaller dataset.
    • “Few-shot learning”.

• Depending the setup, you may also update ‘W’ and the other ‘v_c’.
  – Useful if you have a lot of data on the new class.
  – In this case, would typically mix in new examples with old ones.

• Increasing trend in vision and language to using pre-training a lot.
  – No need to learn everything about language for every language task!
Summary

• **Autoencoders:**
  – Neural network where the output is the input.
  – Encode data into a bottleneck layer, then decode predict original input.
  – Can be used for visualization, compression, outlier detection, pre-training.

• **Denoising autoencoders** train to uncorrupt/enhance images.
  – Can be used for removing noise, adding colour, super-resolution, and so on.

• **Multi-label classification:**
  – Classification with more than one label per example.

• **Encoding-Decoding approach to multi-label classification:**
  – Have all classes shared the same hidden layer(s).
  – Reduces number of parameters.
  – Models dependencies between classes, while keeping inference easy.

• **Pre-training:**
  – Use parameters from model trained on large diverse dataset, to initialize SGD for new dataset.

• Next time: helping teach fish to drive?