
COMP90051 Statistical Machine Learning

Semester 2, 2017
Lecturer: Andrey Kan

Copyright: University of Melbourne
This lecture

- Deep learning
  - Representation capacity
  - Deep models and representation learning

- Convolutional Neural Networks
  - Convolution operator
  - Elements of a convolution-based network

- Autoencoders
  - Learning efficient coding
Deep Learning and Representation Learning

Hidden layers viewed as feature space transformation
Representational capacity

- ANNs with a single hidden layer are universal approximators.
- For example, such ANNs can represent any Boolean function:
  \[
  \begin{align*}
  OR(x_1, x_2) & \quad u = g(x_1 + x_2 - 0.5) \\
  AND(x_1, x_2) & \quad u = g(x_1 + x_2 - 1.5) \\
  NOT(x_1) & \quad u = g(-x_1)
  \end{align*}
  \]
  \[
  g(r) = 1 \text{ if } r \geq 0 \text{ and } g(r) = 0 \text{ otherwise}
  \]
- Any Boolean function over \( m \) variables can be implemented using a hidden layer with up to \( 2^m \) elements.
- More efficient to stack several hidden layers.
Deep networks

\[ s = \tanh(A'x) \quad t = \tanh(B's) \quad u = \tanh(C't) \quad z = \tanh(D'u) \]
Deep ANNs as representation learning

• Consecutive layers form representations of the input of increasing complexity

• An ANN can have a simple linear output layer, but using complex non-linear representation

\[ z = \tanh \left( D' \left( \tanh \left( C' \left( \tanh \left( B' \left( \tanh \left( A' x \right) \right) \right) \right) \right) \right) \] 

• Equivalently, a hidden layer can be thought of as the transformed feature space, e.g., \( u = \phi(x) \)

• Parameters of such a transformation are learned from data

Bias terms are omitted for simplicity
ANN layers as data transformation

input data → transformation layers (S1, S2, ..., Sp1, ..., tp2) → output layers (t1, t2, ..., up3) → output data

The model transforms input data through layers to produce output data.
ANN layers as data transformation

pre-processed data

the model

\[ x_1 \rightarrow s_1 \rightarrow t_1 \rightarrow u_1 \rightarrow z_1 \]
\[ x_2 \rightarrow s_2 \rightarrow t_2 \rightarrow u_2 \rightarrow z_2 \]
\[ \ldots \]
\[ x_p \rightarrow s_{p1} \rightarrow t_{p2} \rightarrow u_{p3} \rightarrow z_q \]
ANN layers as data transformation

pre-processed data → \( x_1, x_2, \ldots, x_p \) → ANN layers → \( s_1, s_{p1}, t_1, t_2, t_{p2}, u_1, u_{p3} \) → the model → \( z_1, z_2, \ldots, z_q \)
ANN layers as data transformation

pre-processed data

the model
Depth vs width

• A single infinitely wide layer used in theory gives a universal approximator

• However depth tends to give more accurate models
  * Biological inspiration from the eye:
  * first detect small edges and color patches;
  * compose these into smaller shapes;
  * building to more complex detectors, such as textures, faces etc.

• Seek to mimic layered complexity in a network

• However vanishing gradient problem affects learning with very deep models
Animals in the zoo

- Recurrent neural networks are not covered in this subject
- If time permits, we will cover **autoencoders**. An autoencoder is an ANN trained in a specific way.
  * E.g., a multilayer perceptron can be trained as an autoencoder, or a recurrent neural network can be trained as an autoencoder.
Convolutional Neural Networks (CNN)

Based on repeated application of small filters to patches of a 2D image or range of a 1D input
Convolution

\[ b_2 \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \]

\( \Sigma \)

\( a_1 \quad a_2 \quad a_3 \quad a_4 \quad \ldots \quad \ldots \quad \ldots \quad \ldots \)

sliding window

\( b \) is output vector

\( a \) is input vector

\( \times w_1 \quad \times w_2 \quad \times w_3 \)
Convolution

\[ b_i = \sum_{\delta=-C}^{C} a_{(i+\delta)} w_{(\delta+C+1)} \]

\[ b = a \ast w \]

\[ w = [w_1, w_2, w_3]^T \]

*Later in the subject, we will also use an unrelated definition of kernel as a function representing a dot product.
Convolution on 2D images

\[ B_{ij} = \sum_{\delta_i = -C}^{C} \sum_{\delta_j = -D}^{D} A_{i+\delta_i,j+\delta_j} W_{\delta_i+C+1,\delta_j+D+1} \]

\[ B = A \ast W \]
Filters as feature detectors

- **Convolve with a vertical edge filter**

  \[
  \begin{bmatrix}
  -1 & 0 & 1 \\
  -1 & 0 & 1 \\
  -1 & 0 & 1 
  \end{bmatrix}
  \]

- **Activation function**

- **Input image** \( A \)

- **Filtered image**
Filters as feature detectors

\[
A \text{ is input image}
\]

convolve with a vertical edge filter

activation function

filtered image
Stacking convolutions

- Develop complex representations at different scales and complexity
- Filters are learned from training data!
CNN for computer vision

patches of $48 \times 48$

2D convolution

fully connected

linear regression

1 $\times$ 720

48 $\times$ 48 $\times$ 5
downsampling

12 $\times$ 12 $\times$ 10

flattening

1 $\times$ 1440

1 $\times$ 720

1 $\times$ 1440

3D convolution

downsampling

24 $\times$ 24 $\times$ 10

24 $\times$ 24 $\times$ 5

Implemented by Jizhizi Li
based on LeNet5: http://deeplearning.net/tutorial/lenet.html
Components of a CNN

• Convolutional layers
  * Complex input representations based on convolution operation
  * Filter weights are learned from training data

• Downsampling, usually via Max Pooling
  * Re-scaling to smaller resolution, limits parameter explosion

• Fully connected parts and output layer
  * Merges representations together
Downsampling via max pooling

• Special type of processing layer. For an $m \times m$ patch
  \[ v = \max(u_{11}, u_{12}, \ldots, u_{mm}) \]

• Strictly speaking, not everywhere differentiable. Instead, “gradient” is defined heuristically
  * Tiny changes in values of $u_{ij}$ that is not the maximum do not change $v$
  * If $u_{ij}$ is the maximum value, tiny changes in that value change $v$ linearly

• As such use $\frac{\partial v}{\partial u_{ij}} = 1$ if $u_{ij} = v$, and $\frac{\partial v}{\partial u_{ij}} = 0$ otherwise

• Forward pass records maximising element, which is then used in the backward pass during back-propagation
Convolution as a regulariser

Restriction: same color – same weight

Fully connected, unrestricted
Document classification (Kalchbrenner et al, 2014)

Structure of text important for classifying documents

Capture patterns of nearby words using 1d convolutions
Autoencoder

An ANN training setup that can be used for unsupervised learning or efficient coding
Autoencoding idea

- **Supervised learning:**
  - Univariate regression: predict $y$ from $x$
  - Multivariate regression: predict $y$ from $x$

- **Unsupervised learning:** explore data $x_1, \ldots, x_n$
  - No response variable

- For each $x_i$ set $y_i \equiv x_i$

- Train an ANN to predict $y_i$ from $x_i$

- Pointless?
Autoencoder topology

• Given data without labels $x_1, \ldots, x_n$, set $y_i \equiv x_i$ and train an ANN to predict $z(x_i) \approx x_i$

• Set the hidden layer $u$ in the middle “thinner” than the input

adapted from: Chervinskii at Wikimedia Commons (CC4)
Introducing the bottleneck

• Suppose you managed to train a network that gives a good restoration of the original signal $z(x_i) \approx x_i$

• This means that the data structure can be effectively described (encoded) by a lower dimensional representation $u$
Dimensionality reduction

• Autoencoders can be used for compression and dimensionality reduction via a non-linear transformation

• If you use linear activation functions and only one hidden layer, then the setup becomes almost that of Principal Component Analysis (coming up in a few weeks)
  ∗ The difference is that ANN might find a different solution, it doesn’t use eigenvalues
Tools

• **Tensorflow, Theano, Torch**
  * python / lua toolkits for deep learning
  * symbolic or automatic differentiation
  * GPU support for fast compilation
  * Theano tutorials at [http://deeplearning.net/tutorial/](http://deeplearning.net/tutorial/)

• **Various others**
  * Caffe
  * CNTK
  * deeplearning4j ...

• **Keras: high-level Python API. Can run on top of TensorFlow, CNTK, or Theano**
This lecture

• Deep learning
  * Representation capacity
  * Deep models and representation learning

• Convolutional Neural Networks
  * Convolution operator
  * Elements of a convolution-based network

• Autoencoders
  * Learning efficient coding