Machine Learning: Algorithms and Applications

Advanced Multimedia Research Lab
University of Wollongong

Artificial Neural Networks and Deep Learning: An Introduction (II)

Outline

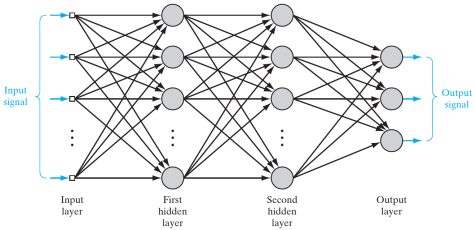

Figure 1: Fully connected MLP (Haykin 2009)

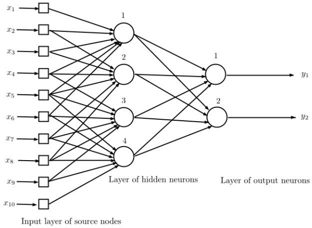

Figure 2: Simple CNN architecture (Haykin 2009)

- Input-output relationship for a layer can be described by matrix multiplication

$$\boldsymbol{v_i} = \boldsymbol{W}\boldsymbol{v_{i-1}} \qquad (1)$$

Note the use of vector (bold) notation in the multiplication

- Input-output relationship for hidden layer is described by convolution (weight matrix will have to be Toeplitz)

$$v_j = \sum_{i=1}^{6} \omega_i x_{i+j-1}; \quad j = 1, 2, 3, 4 \qquad (2)$$

$\{\omega_i\}_{i=1}^{6}$ constitute the same set of weights shared by all four hidden neurons

CNN - Convolutional Neural Network

- CNNs are a special class of multilayer perceptron that are well suited to processing data with grid-like topology.
- Examples include
  - Time-series data : 1-D grid taking samples at regular intervals
  - Image data : 2-D grid of pixels
- CNNs are networks that use convolution in place of general matrix multiplication in at least one of the layers
- CNN is a way of building prior information into neural network design (see Figure 2)
  1. Network architecture restriction - use local connection a.k.a receptive fields
  2. Constrain the choice of synaptic weight - use weight sharing
- CNNs are sometimes designed to recognize 2-D shapes with high degree of invariance to translation, scaling, skewing and other distortions

CNN - Convolutional Neural Network

- More generally structural constrain in CNN aim to achieve:
  - Feature extraction: Each neuron takes its synaptic inputs from a local receptive filed of neurons forcing it to extract local features
  - Feature mapping: Each computational layer has multiple feature maps; they form planes in which neurons are forced to share same set of synaptic weights; beneficial effects
    - shift invariance - achieved through convolution followed by activation function
    - reduction in the number of free parameters - achieved through weight sharing
  - Subsampling (a.k.a pooling) - Convolutional layer followed by a computational layer performing local averaging and subsampling
    - feature map resolution reduction
    - reduction of sensitivity of feature map output to shifts and other distortions

Typical contemporary CNN architecture

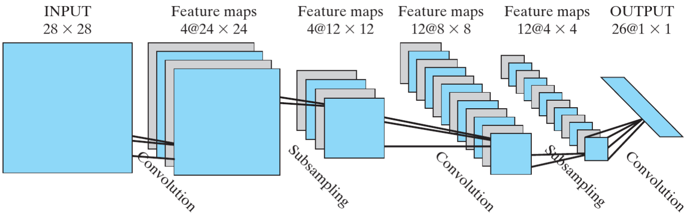

Figure 3: CNN architecture designed for hand-written character recognition (Haykin 2009)

1. Four (4) hidden layers and an output layer
2. First hidden layer - convolution; four feature maps of 24×24 neurons; each neuron assigned a receptive field of size 5×5
3. Second hidden layer - subsampling and local averaging; 4 feature maps of 12×12; each neuron has receptive field size 2×2
4. Third hidden layer - convolution; 12 feature maps of 8×8 neurons;
5. Fourth layer - subsampling and local averaging; 12 feature maps of 4×4 neurons
6. Final layer convolution; 26 neurons, each assigned to one of possible 26 characters; each neuron assigned to receptive field 4×4

What is convolution? A deeper insight

- Given two functions $x(n)$ and $\omega(n)$ the convolution of the two functions is written as

$$s(n) = \sum_k x(k)\omega(n-k) = \sum_k x(n-k)\omega(k) \tag{3}$$

  Equation 3 says, flip the function $\omega(n)$ relative to $x(n)$ and slide it across the function $x(n)$, each time compute the product of overlapping samples; note commutative property of convolution

- For the purpose of neural networks, $x(n)$ is the input; $\omega(n)$ is the kernel and $s(n)$ is the feature map
- In the two-dimensional case we have

$$s(n,m) = \sum_{k,l} x(k,l)\omega(n-k, m-l) = \sum_{k,l} \omega(k,l)x(n-k, m-l) \tag{4}$$

  Summation is over the non-zero values of the kernel; usually defined to be finite in spatial extent

- Convolution as defined in Equation 4 is rarely used in machine learning; rather use cross-correlation (but referred to as convolution); See Figure 4

$$s(n,m) = \sum_{k,l} x(k,l)\omega(n+k, m+l) \tag{5}$$

Convolution (without flipping) operation in 2-D case

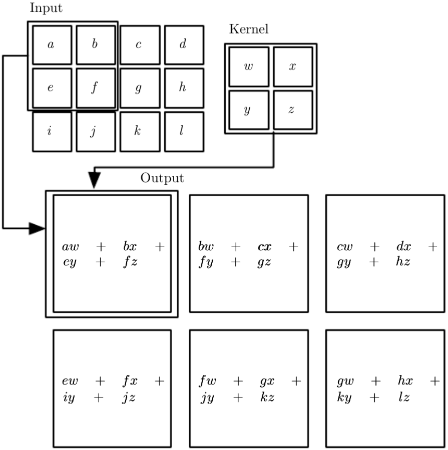

Figure 4: 2-D convolution operation (Goodfellow et al. 2016)

Convolution benefits revisited

- Convolution introduces three ideas beneficial for machine learning systems:
  1. Sparse interactions
  2. parameter sharing
  3. Equivariant representation

Sparse interactions

- Sparse weights leads to needing fewer parameters to store and improving statistical efficiency
- In deep CNN deeper layers indirectly interact with a larger portion of the input allowing network to efficiently describe complicated interactions

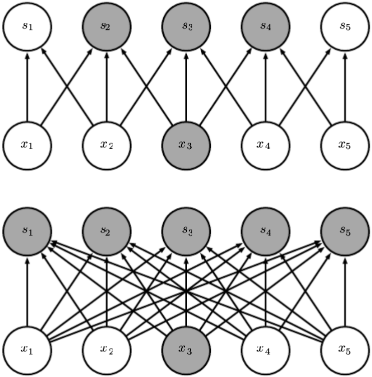

Figure 5: Sparse connectivity viewed from input (note how many outputs affected by one input) (Goodfellow et al. 2016)

Sparse interactions

- Sparse weights leads to needing fewer parameters to store and improving statistical efficiency
- In deep CNN deeper layers indirectly interact with a larger portion of the input allowing network to efficiently describe complicated interactions

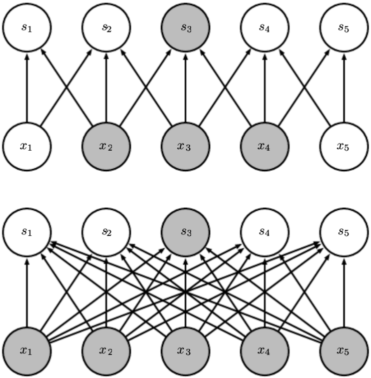

Figure 5: Sparse connectivity viewed from output (note how many inputs affect one output) (Goodfellow et al. 2016)

Parameter sharing

- Rather than learn separate set of parameters for every location we learn only one set
- Since kernel is usually much less than input data size, convolution is more efficient than dense matrix multiplication

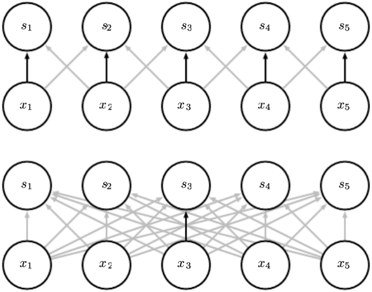

Figure 6: Parameter sharing; (top) central element in 3-element kernel is used multiple times; (bottom) Shown central element of weight matrix is used once in the fully connected architecture (Goodfellow et al. 2016)

Equivariance

- Convolution as considered here leads to a form of parameter sharing that generates a property called equivariance to translation. In signal processing this is the same as linear time invariance.
- Equivariant means that if the input changes, the output changes in the same way.
- For example, let g map an image I to another image I'; $I' = g(I)$ $I'(x, y) = I(x - 1, y)$; applying convolution after the mapping is the same as applying the convolution and then transforming the result.
- In a data stream if we delay a feature in time (or spatially) the same representation appears in the output later (by amout of the delay)

Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- Examples:
  1. Max pooling operation reports the maximum output within a rectangular neighbourhood (E.g. see Figure 7)
  2. L^2-norm of a rectangular neighbourhood
  3. Weighted average based on distance from central pixel
- Pooling helps to make the representation approximately invariant to small translations of input (e.g. see Figure 8)
- Use of pooling can be interpreted as using a strong prior that the function being learned (in the layer) must be invariant to small translation
- Pooling over over spatial regions leads to translation invariance
- Pooling over separately parametrized convolutions leads to learning transformations that are invariant (e.g. see Figure 9)

Pooling

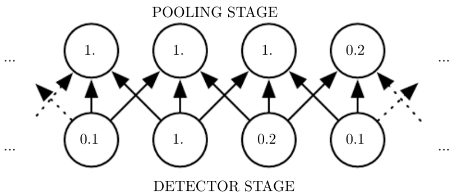

Figure 7: Max pooling of width 3 and stride 1 (Goodfellow et al. 2016)

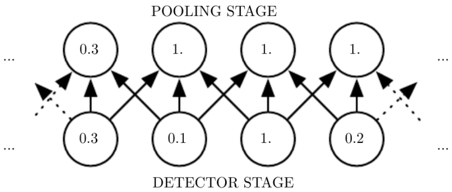

Figure 8: Max pooling of width 3 and stride 1 and input shifted to the right (to show invariance to translation) (Goodfellow et al. 2016)

Pooling

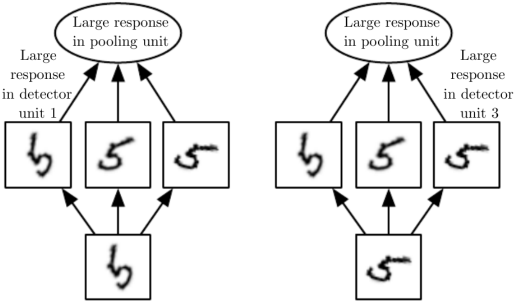

Figure 9: Pooling over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. (Goodfellow et al. 2016)

Pooling

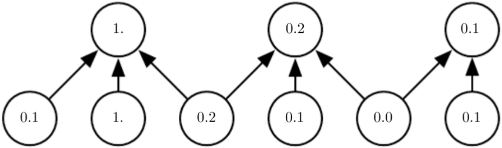

Figure 10: Pooling with downsampling: max-pooling with a pool width of three and a stride between pools of two. (Goodfellow et al. 2016)

Application - human action recognition

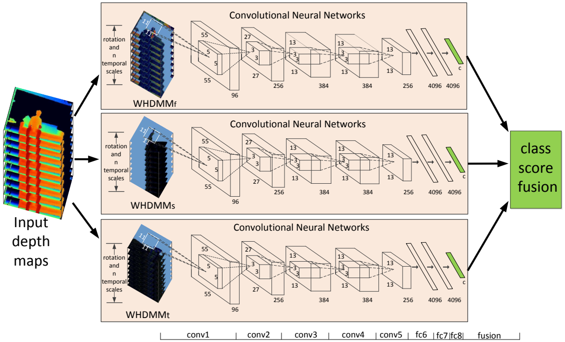

Figure 11: CNN architecture for human action recognition from depth maps; original depth data rotated in 3D pointclouds mimics camera rotation; weighted depth motion maps (WDMM) at several temporal scales, called Weighted Hierarchical Depth Motion Maps (WHDMM); WHDMM are encoded into Pseudo-RGB images in which the spatial-temporal motion patterns in videos can be effectively converted into spatial structures (edges and textures) for input to ConvNets (Wang et al. 2016)

Autoencoders

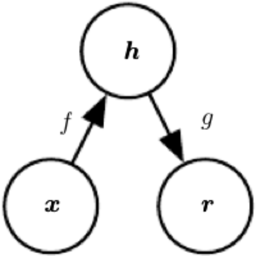

Figure 1: General structure of an autoencoder; input x maps to an output r (reconstruction) through internal representation or code h (Goodfellow et al. 2016)

- Conceptually an autoencoder is a feedforward network trained to copy its input to its output (albeit imperfectly)
- Structure (see Figure 1) has a hidden layer h describing the code representing the input
- Autoencoder has two parts: encoder function $h = f(x)$ and decoder function $r = g(h)$ producing a reconstruction
- Generalization of autoencoder to stochastic mappings: $p_{\text{encoder}}(h|x)$ and $p_{\text{decoder}}(x|h)$
- Typical training strategy is similar to that used for feedforward networks - minibatch gradient descent

Undercomplete/Overcomplete autoencoders

- Constraining h to have smaller dimension than $x$results in an undercomplete autoencoder
- h captures the most salient features of input
- Learning entails minimizing a loss function

$$L(x, g(f(x))) \tag{1}$$

  L penalizes $g(f(x))$ being dissimilar to x

- If dimension of code is greater than that of input we have overcomplete autoencoder
- Any architecture of autoencoder can be trained without the risk of over-capacity or learning a trivial identity, by using regularization
- Regularization can inpart properties to loss function:
  - sparsity of representtaion
  - smallness of derivative of representation
  - robustness to noise
  - robustness to missing data

Sparse autoencoders

- Sparse autoencoder has cost function used for training in the form of reconstruction error and sparsity penalty on the code layer h:

$$L(x, g(f(x))) + \Omega(h) \tag{2}$$

where h is the decoder output; $h = f(x)$ typically (see Figure 1)

- Sparse autoencoders are useful in learning features that can be input for other tasks, e.g. classification
- Sparse autoencoders can be interpreted as approximating maximum likelihood training of generative model that has latent variables (in this case h)
- In this respect, it is maximizing

$$\log p_{\text{model}}(h, x) = \log p_{\text{model}}(h) + \log p_{\text{model}}(x|h) \tag{3}$$

$\log p_{\text{model}}(h)$ can be sparsity-inducing

Denoising autoencoders

- Denoising aims to reduce the noise in signals
- Denoising autoencoders minimize

$$L(x, g(f(\tilde{x}))) \tag{4}$$

  where $\tilde{x}$ is a copy of x corrupted by some form of noise
- Training process forces f and g to implictly learn the structure of $p_{\text{data}}(x)$
- Another form of regularization $\lambda \sum_i ||\nabla_x h_i||^2$ forces the learning of a function that does not change much when x changes slightly:

$$L(x, f(g(x))) + \lambda \sum_i ||\nabla_x h_i||^2 \tag{5}$$

Denoising autoencoders

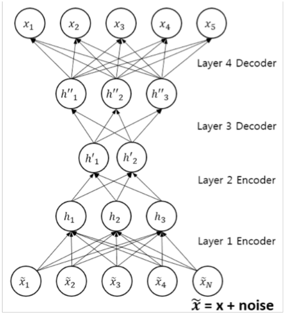

Figure 2: Stacked convolutional denoising autoencoder

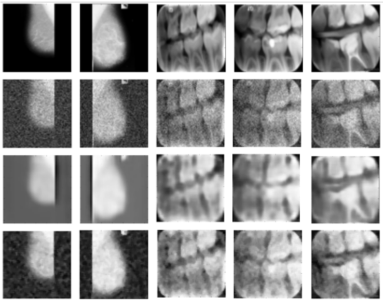

Figure 3: Comparison between output of stacked convolutional denoising autoencoder and median filter; Gaussian noise: $\mu = 0$, $\sigma = 1$

More autoencoders - cost functions

Contractive autoencoder

- Regularization is introduced on the code $h = f(x)$ to encourage derivatives of f to be as small as possible

$$\Omega(h) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$$

- Contracctive autoencoder and denoising autoencoder are connected when input noise is small and Gaussian (Goodfellow et al. (2016))

More autoencoders - cost functions

Predictive sparse decomposition

- Model is a hybrid of sparse coding and parametric autoencoders
- Predicts output of iterative inference
- Both $f(x)$ and $g(h)$ are parametric and h is controlled during optimization
- Cost function:

$$\|x - g(h)\|^2 + \lambda |h|_1 + \gamma \|h - f(x)\|^2$$

Image reconstruction from projections

- Algebraic reconstruction technique (ART) iteratively solves the constrained optimization problem:

$$\min_x \|x\|_2^2 \quad \text{such that} \quad Hx = y \tag{6}$$

  where y are the measured projections and x is the image to be reconstructed

- In terms of autoencoder we can write:
  - Encoder: $h = f(y)$
  - Decoder: $y = g(h)$

- We learn the code h and apply it for the reconstruction?

- Possible cost function:

$$\begin{aligned} J &= L(y, g(f(y))) + \Omega(h, y) \\ &= L(y, g(f(y))) + \lambda \left\| \nabla_y h_i \right\|^2 \end{aligned} \tag{7}$$

Bibliography

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press.

Haykin, S. (2009), *Neural Networks and Learning Machines*, Third edn, Pearson Education.

Wang, P., Li, W., Gao, Z., Zhang, J., Tang, C. & Ogunbona, P. (2016), 'Action recognition from depth maps using deep convolutional neural network', *IEEE Transactions on Human Machine Systems* pp. 498–509.