Machine Learning

Neural Networks 3

Hiroshi Shimodaira and Hao Tang

12 February 2024

Ver. 1.0

Topics - you should be able to explain after this week

- Problems with multi-layer neural networks trained with EBP
- Overfitting and generalisation
- Techniques to mitigate the overfitting
- Convolutional neural networks
- Recurrent neural networks
- Other neural network models
- Sequence to sequence models

History of artificial neural networks till 1990

1940s	Warren McCulloch and Walter Pitts : 'threshold logic'
	Donald Hebb : 'Hebbian learning'
1957	Frank Rosenblatt : 'Perceptron'
1969	Marvin Minsky and Seymour Papert : limitations of neural networks
1980	Kunihiro Fukushima: 'Neocognitoron'
1986	D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors" (1974, Paul Werbos)

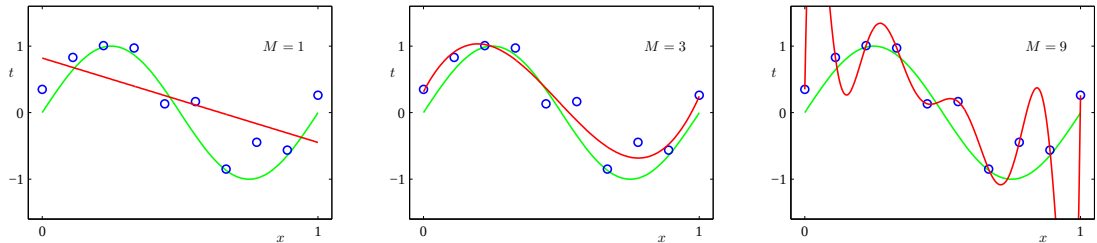
Problems with multi-layer neural networks trained with EBP

- Still difficult to train
  - Computationally very expensive (e.g. weeks of training)
  - Slow convergence ('vanishing gradients')
  - Difficult to find the optimal network topology
- Poor generalisation
  - Very good performance on the training set
  - Poor performance on the test set

Overfitting and generalisation

Example of curve fitting by a polynomial function:

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{k=0}^{M} w_k x^k$$



(after Fig 1.4 in PRML C. M. Bishop (2006))

- cf. memorisation of the training data

Generalisation in neural networks

- How many hidden units (or, how many weights) do we need?
- Optimising training set performance does not necessarily optimise test set performance
  - Network too "flexible": Too many weights compared with the number of training examples
  - Network not flexible enough: Not enough weights (hidden units) to represent the desired mapping
- **Generalisation Error**: The predicted error on unseen data. How can the generalisation error be estimated?
  - Training error?

$$E_{\text{train}} = \frac{1}{2} \sum_{\text{trainingset}} \sum_{k=1}^{K} (y_k - t_k)^2$$

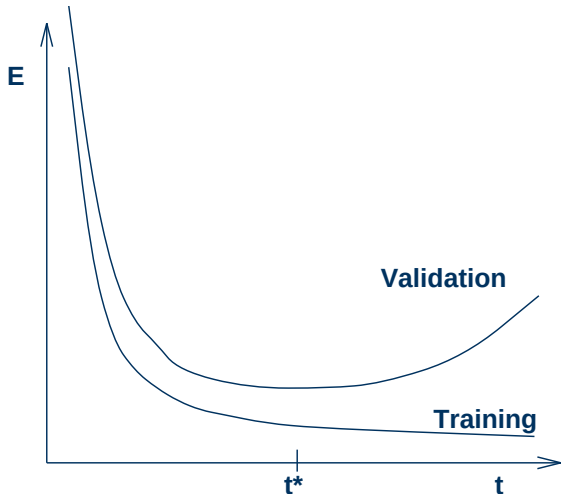
  - Cross-validation error?

$$E_{\text{xval}} = \frac{1}{2} \sum_{\text{validationset}} \sum_{k=1}^{K} (y_k - t_k)^2$$

Overtraining in neural networks

- Overtraining (overfitting) corresponds to a network function too closely fit to the training set (too much flexibility)
- Undertraining corresponds to a network function not well fit to the training set (too little flexibility)
- Solutions
  - If possible increasing both network complexity in line with the training set size
  - Use prior information/knowledge to constrain the network functions: Structural Stabilisation
  - Control the effective flexibility: **early stopping** and **regularisation**

Early stopping



Early stopping (*cont.*)

- Use validation set to decide when to stop training
- Training-set error monotonically decreases as training progresses
- Validation-set error will reach a minimum then start to increase
- "Effective Flexibility" increases as training progresses
- Network has an increasing number of "effective degrees of freedom" as training progresses
- Network weights become more tuned to training data
- Very effective — used in many practical applications such as speech recognition and optical character recognition

Regularisation – penalising complexity

- Original error function

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}||\hat{\mathbf{y}}_n - \mathbf{y}_n||^2$$

- Regularised error function (aka weight decay)

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}||\hat{\mathbf{y}}_n - \mathbf{y}_n||^2 \; + \; \frac{\beta}{2}\sum||\boldsymbol{w}||^2$$

Breakthrough

1957 Frank Rosenblatt : 'Perceptron'

1986 D. Rumelhart, G. Hinton, and R. Williams: 'Backpropagation'

2006 G. Hinton *et al.*(U. Toronto)

"Reducing the dimensionality of data with neural networks", Science, July 2006

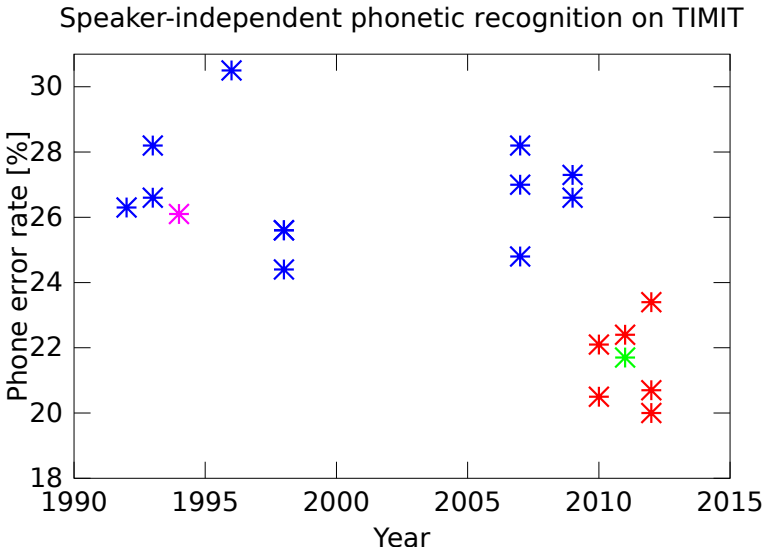
2009 J. Schmidhuber (Swiss AI Lab IDSIA)

Winner at ICDAR2009 handwriting recognition competition

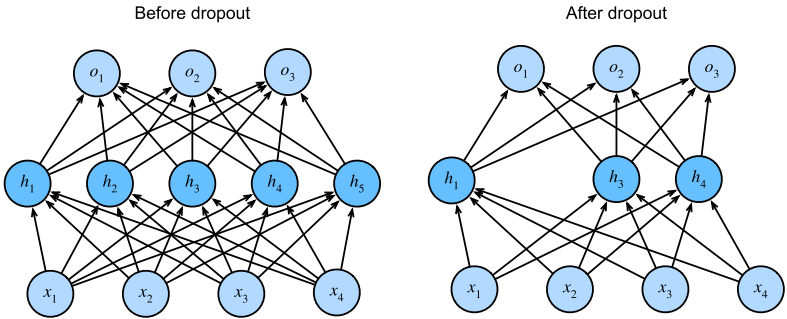
2011- many papers from U.Toronto, Microsoft, IBM, Google, ...

- What's the idea?
  - Pretraining
    - A single layer of feature detectors $\rightarrow$ Stack it to form several hidden layers
  - Fine-tuning, dropout
  - GPU
  - Convolutional network (CNN), Long short-term memory (LSTM)
  - Rectified linear unit (ReLU)

Speech recognition



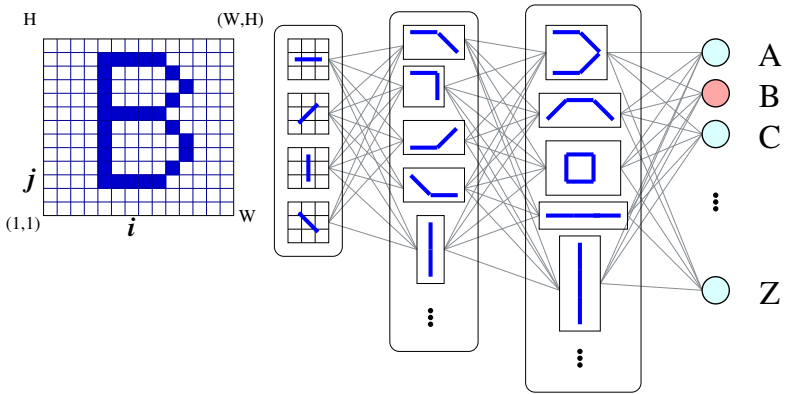
Another regularisation - dropout



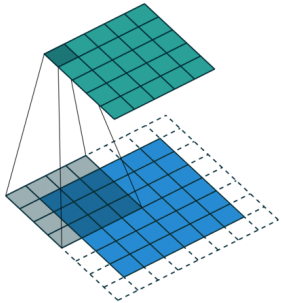
After Figure 5.6.1 MLP before and after dropout of Dive into Deep Learning

Background of convolutional neural networks

- Primary visual cortex (D. Hubel and T. Wiesel in 1959)
- Neocognitron (K. Fukushima in 1979)
- Nice to have shift/scale/rotation invariant classifiers
- Hard to train fully-connected neural networks on image data



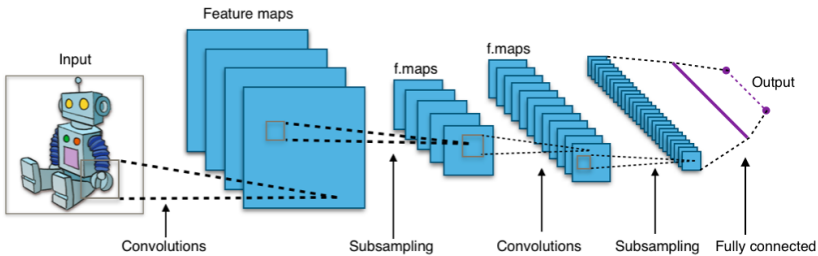
Convolutional neural networks (CNNs)



After Convolution arithmetic - Arbitrary padding no strides of Wikimedia common

Components of CNNs

- Convolutional layers
- Pooling layers
- Normalisation layers
- (Fully connected layers)



After Typical CNN architecture of Wikimedia common

Convolution *(NE)*

- The output of applying a filter $w(n)$ to an input $x(n)$ is given as convolution, denoted as $[\mathbf{w} \circledast \mathbf{x}](n)$ or $w(n) \circledast x(n)$:

$$y(n) = [\mathbf{w} \circledast \mathbf{x}](n) = \sum_{\ell=-\infty}^{\infty} w(\ell)\, x(n-\ell)$$

- If $w(n)$ is defined in $[-L, L]$,

$$\begin{aligned} y(n) &= \sum_{\ell=-L}^{L} w(\ell)\, x(n-\ell) \\ &= \sum_{k=-L}^{L} w(k)\, x(n+k) \quad \text{where } k = -\ell \end{aligned}$$

- Further simplification by assuming w(n) is defined in $[0, L-1]$ yields:

$$y(n) = \sum_{k=0}^{L-1} w(k)\, x(n+k)$$

CNNs – Convolutional layers

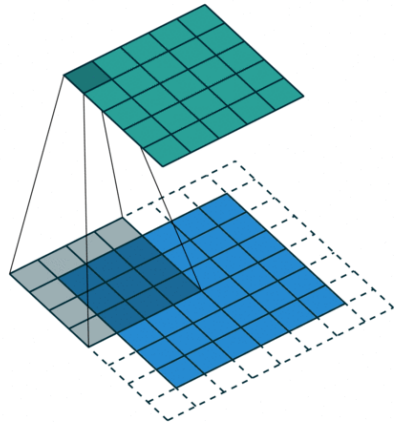
- 1D convolution

$$[\mathbf{w} \circledast \mathbf{x}](i) = \sum_{u=0}^{L-1} w_u \, x_{i+u}$$

- 2D convolution

$$[\mathbf{W} \circledast \mathbf{X}](i,j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v} \, x_{i+u,j+v}$$

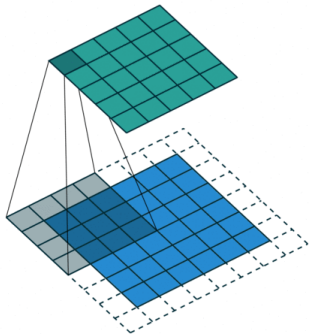
- filters (aka kernels)
  - Assuming local correlation / connectivity
  - Parameter sharing (shared weights)
- Parameters for convolution
  - Stride
  - Padding

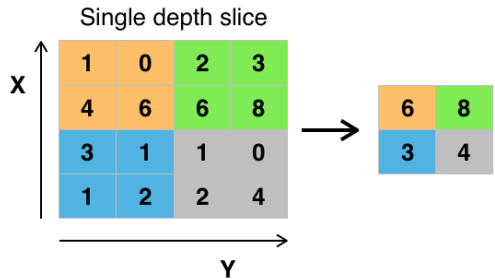


After Convolution arithmetic - Arbitrary padding no strides of Wikimedia common

CNNs – Pooling layers

- Downsampling → increase robustness against small shift
- Types of pooling
  - Max pooling
  - Average pooling





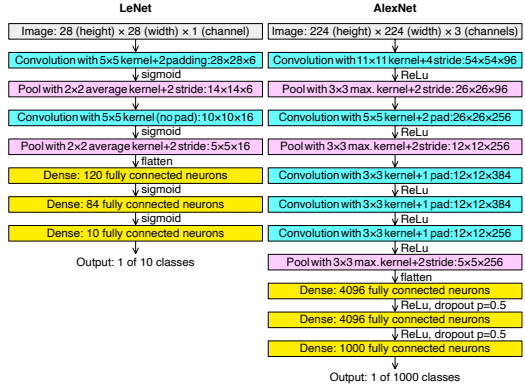
After Convolution arithmetic - Arbitrary padding no strides of Wikimedia common *After Max pooling of Wikimedia common*

CNNs – Normalisation layers

- Batch normalisation *(S. Loffe and C. Szebedy, Google, 2015)*
  - Normalise the output distribution of a layer for each mini-batch – zero mean and a unit variance.

CNN based systems

- LeNet (LeCun *et al.*, 1998)
- AlexNet (Alex Sutskever *et al.*, 2017)
- VGG16 (K.Simonyan and A.Zisserman, 2014)
- ResNet (He *et al.*, 2016)



After Comparison image neural networks of Wikimedia common

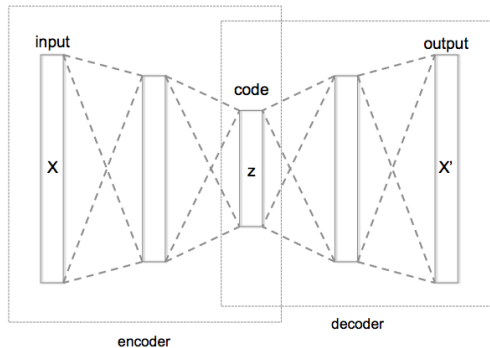
Other neural networks

- Autoencoder

$$\min_{\theta,\phi} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - \text{Decoder}_\theta(\text{Encoder}_\phi(\mathbf{x}_i))\|^2$$

z:

  - Embedded features, bottleneck features
  - Latent representation
  - cf. PCA



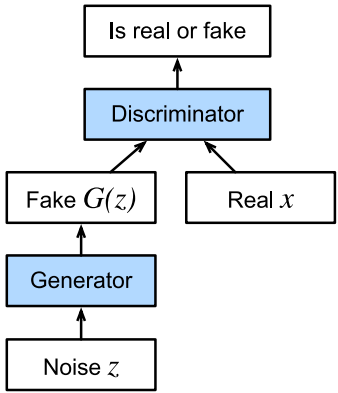
After Autoencoder structure of Wikimedia common

- Variational Auto Encoder (VAE)

  [D.Kingma and M.Welling, 2013]

Other neural networks (*cont.*)

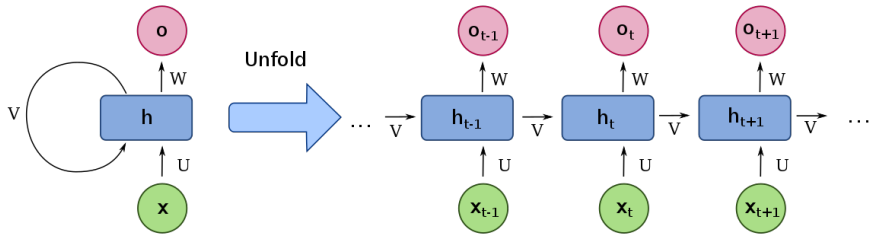
- Generative adversarial network (GAN) [I. Goodfellow *et al.*, 2014]



After Figure 18.1.1 Generative Adversarial Networks of Dive into Deep Learning

Neural networks for sequences

- Recurrent Neural Network (RNN)

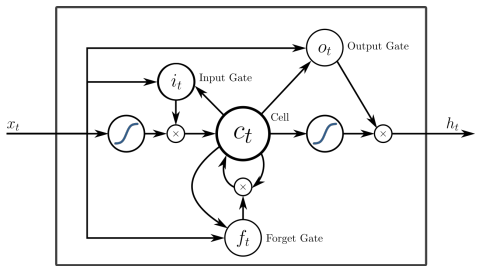


After Recurrent neural network unfold of Wikimedia common

- Vanishing/exploding gradients problem

Neural networks for sequences (*cont.*)

- Long short-term memory (LSTM) S.Hochreiter and J.Schmidhuber, 1995

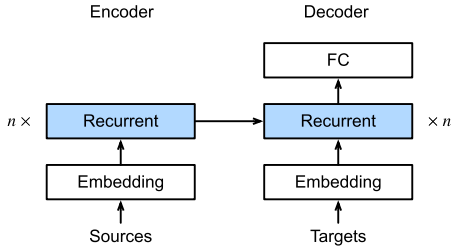


After A peephole LSTM unit with input (i.e. i), output (i.e. o), and forget (i.e. f) gates of Wikimedia common

- Gated recurrent unit (GRU) Kyunghyun Cho *et al.*, 2014

Sequence to sequence models

- seq2seq ··· encoder-decoder model with attention



From Figures 10.7.1 and 10.7.2 of Dive into Deep Learning

Sequence to sequence models

- Transformer
  - BERT (Bidirectional Encoder Representations from Transformers) J.Devlin *et al.*, Google, 2018
    Question answering, natural language interface
  - GPT-2 (Generative Pre-trained Transformer 2) , A.Radford *et al.*, OpenAI, 2019) $\cdots$ 1.5 billion parameters.
    Text translation/generation/summarisation, question answering
  - GPT-3 $\cdots$ 175 billion parameters
  - DALL-E $\cdots$ 12-billion parameter version of GPT-3 trained to generate images from text descriptions
- Overparameterisation: Belkin *et al.*, Neyshabur *et al.*, Allen-Zhu *et al.*
- Scaling laws

  Chinchilla scaling (Hoffmann *et al.*, 2022):

  $$L = \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + \text{const.}$$

  L: loss, N: number of parameters, D: dataset size

  - Hestness, *et al.* Deep Learning Scaling is Predictable, Empirically, 2017 [Baidu]
  - Kaplan, *et al.* Scaling Laws for Neural Language Models, 2020 [OpenAI]
  - Hoffmann, *et al.* Training Compute-Optimal Large Language Models, 2022 (DeepMind)

Other topics *(NE)*

- Transfer learning
- Multi-task learning
- Data augmentation
- Domain adaptation
- Zero-shot learning
- Contrastive learning