Neural Networks

Advanced Topics in High-Performance Computing

Faisal Qureshi

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Feed forward neural networks

- Approximate some function $y = f^*(\mathbf{x})$ by learning parameters θ s.t. $\tilde{y} = f(\mathbf{x}; \theta)$
- Feed forward neural networks can be seen as *directed acyclic graphs*

$$y = f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$$

- Training examples specify the output of the *last* layer
  - Network needs to figure out the inputs/outputs for the *hidden* layers

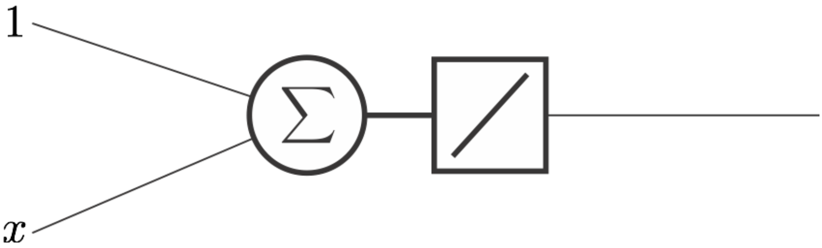
Extending linear models

How can we extend linear models?

- Specify a very general ϕ s.t. the model becomes $y = \theta^T \phi(\mathbf{x})$
  - Problem with generalization
  - Difficult to encode *prior* information needed to solve AI-level tasks
- Engineer ϕ for the task at hand
  - Tedious
  - Difficult to transfer to new tasks
- Neural networks approaches
  - $y = f(\mathbf{x}; \theta, w) = \phi(\mathbf{x}; \theta)^T w$ i.e. use parameters θ to learn ϕ and use w to map $\phi(\mathbf{x})$ to the desired output y
  - The training problem is non-convex
  - Key advantage: a designer just need to specify the right family of functions and not the exact function ϕ

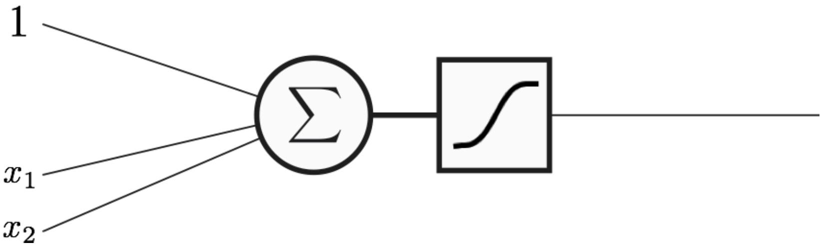
Linear regression

Perceptron with linear activation for linear regression



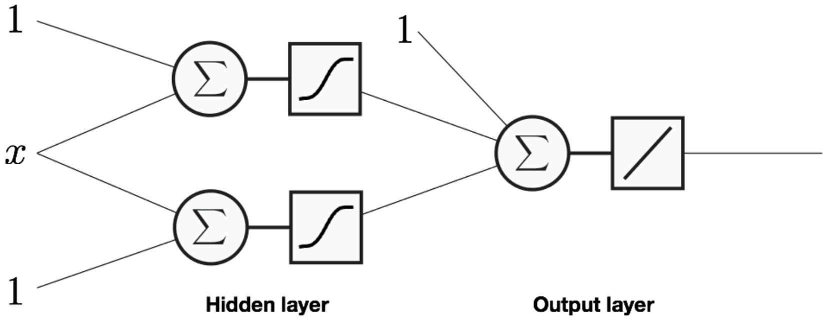
Classification - Linear separating plane

Perceptron with sigmoid activation for classification



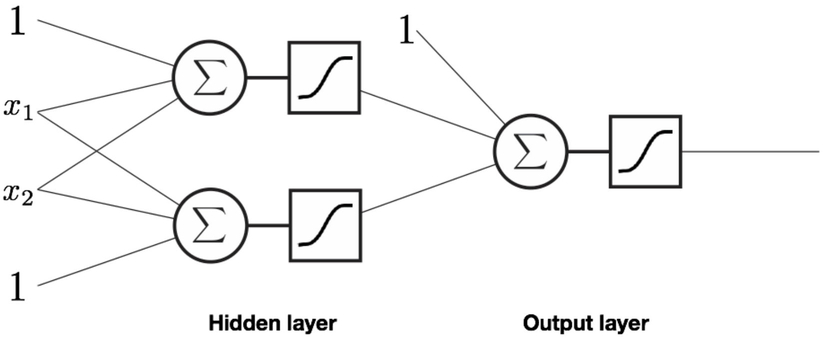
Regression - 2 layer, 3 perceptron neural network

Last layer has linear activation



Classification - 2 layer, 3 perceptron neural network

Last layer has sigmoid activation



The time before deep networks

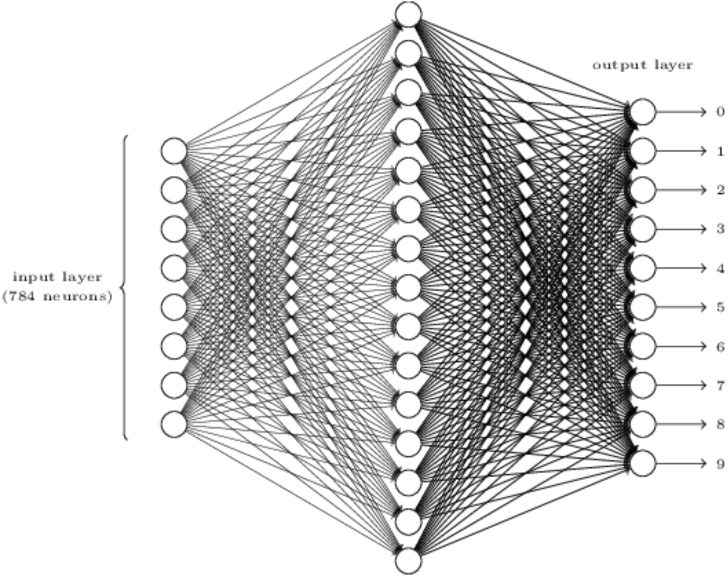


Figure 1: Neural networks for digit recognition

Neural networks

Old view

- Shallow and wide
- One hidden layer can represent any function
- Focus was on efficient ways to optimize (train)

Current view

- Deep networks - multi-layer networks
- Access to data
- Advances in computer science, physics and engineering
- Deep networks outperform humans on many tasks

Gradient-based learning in neural networks

- Non-linearities of neural networks render most cost functions non-convex
- Use iterative gradient based optimizers to drive cost function to lower values
- Gradient descent applied to non-convex cost functions has no guarantees is sensitive to initial conditions
  - Initialize weights to small random values
  - Initialize biases to zero or small positive values

Cost functions

- Most modern neural networks are trainined using *maximum likelihood* principle
- When parametric values defines a distribution $p(y|\mathbf{x};\theta)$ the negative log-likelihood is the cross-entropy between the training data and model predictions
- Advantage of using maximum likelihood: we get cost for free, which is $-\log p(y|\mathbf{x})$
- Gradient of the cost function must be large (and predictable)

Another advantage of using negative log likelihood as a cost function

When hidden or output units saturate, their gradients become really small, creating difficulties for gradient based learning methods. Many output units contain and $\exp()$, for example softmax, an advantage of using negative log likelihood is also that it undoes the effects of $\exp()$ preventing saturation

Output units

The role of the output units is to provide some additional transformations from the features computed by the hidden layers to complete the task at hand:

$$y = f(\mathbf{h}),$$

where $\mathbf{h} = f(\mathbf{x}; \theta)$ are the features computed by the hidden layer.

- Linear units
- Sigmoid units
- Softmax units

Hidden units

- ReLU
- Leaky ReLU
- Parametric ReLU
- Maxout
- Dropout
- Logistic, sigmoid, hyperbolic tangent
    - Rarely used as hidden units these days, except for recurrent networks

Regularization for deep networks 1

Regularization: any modification to reduce generalization error but not the training errors:

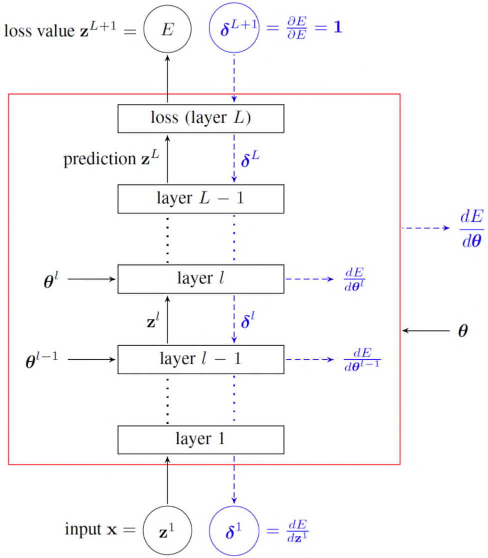
- extra constraints and penalties
- prior knowledge

Deep learning is applied to extremely complex tasks. Consequently, regularization is not as simple as controlling the number of parameters

Regularization for deep networks 2

- Parameter norm penalties
- Data augmentation
  - Fake data
  - Successful in classification/object recognition tasks
- Noise injection
  - Applying random noise to the inputs
  - Applying random noise to hidden layers' inputs
    - Data augmentation at multiple levels of abstraction
  - Data augmentation almost always improves the performance of a neural network
  - Noise added to the weights
    - Recurrent neural networks
    - A practical stochastic implementation of Bayesian inference over weights
  - Noise can also bve added to target outputs

Deep learning: backpropagation



$$
\begin{aligned}
\mathbf{z}^{l+1} &= \mathbf{f}^l(\mathbf{z}^l;\theta^l) \\
\delta^l &= \delta^{l+1}\frac{\partial \mathbf{f}^l(\mathbf{z}^l;\theta^l)}{\partial \mathbf{z}^l} \\
\delta_i^l &= \sum_j \delta_j^{l+1}\frac{\partial f_j^l(\mathbf{z}^l;\theta^l)}{\partial z_j^l} \\
\frac{\partial E}{\partial \theta^l} &= \delta^{l+1}\frac{\partial \mathbf{f}^l(\mathbf{z}^l;\theta^l)}{\partial \theta^l} \\
\frac{\partial E}{\partial \theta_i^l} &= \sum_j \delta_j^{l+1}\frac{\partial f_j^l(\mathbf{z}^l;\theta^l)}{\partial \theta_i^l}
\end{aligned}
$$

From Nando de Freitas

Deep learning: linear layer

$$z_j = f_j(\mathbf{x}; \theta_j) = \sum_i x_i \theta_{ji}$$

Deep learning: ReLU layer

$$z_j = f_j(x_j) = \max(0, x_j)$$

Convolutional Neural Network (ConvNet, CNN)

Suggested by Kunihiko Fukushima, 1980

LeNet, by Yann LeCun, 1998, to classify hand-written digits

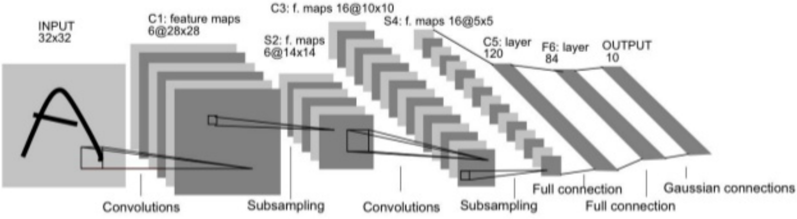


Figure 2: Convnet

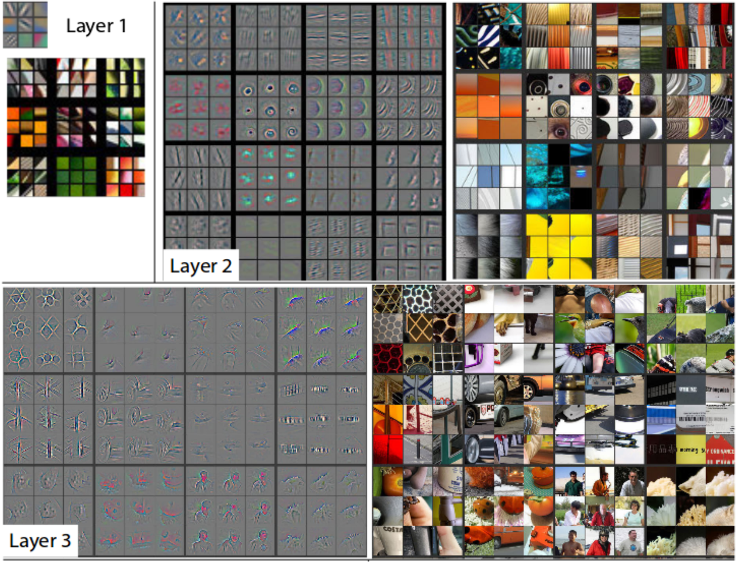
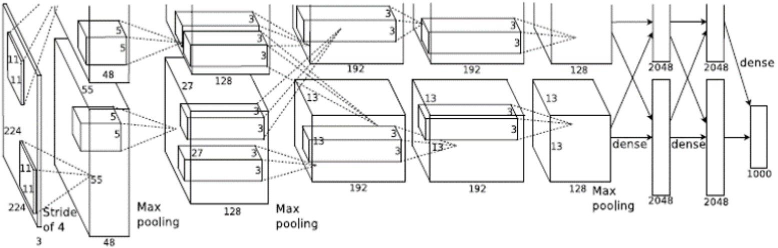


Figure 3: Feature maps (Matthew Zeiler & Rob Fergus)

Convolution

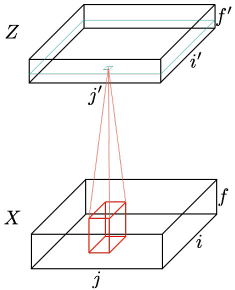
Alexnet



Alexnet

Image convolution layer

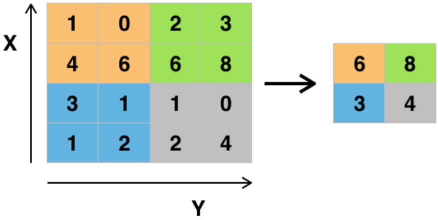
$$\mathbf{y}_{i',j',f'} = b_{f'} + \sum_{i=1}^{H_f} \sum_{j=1}^{W_f} \sum_{f=1}^{F} \mathbf{x}_{i'+i-1,j'+j-1,f} \theta_{ijff'}$$



Convolution layer

Image max-pooling layer

$$\mathbf{y}_{i',j'} = \max_{i,j \in \Omega(i',j')} \mathbf{x}_{i,j}$$



Max pool with 2x2 filter and stride 2

Readings

- Ch. 6-9 of Deep Learning by I. Goodfellow et al.