Introduction to Artificial Neural Networks part 2

Pattern recognition

2. 4. 2019

Outline

- Recap from previous part
- On the usefulness of gradient descent
- Simple ANN example
  - PyTorch ultra-fast intro
- CNN part II
- Simple CNN example
- Other useful ANNs

Back-propagation /recap/

- Gradient descent
  - Adjust weights in the direction of steepest gradient of the error/cost function
  - Ideally, error computed from every sample
  - In reality: computationally infeasible (e.g. memory problems)
  - $\rightarrow$ stochastic gradient descent (mini-batches)
- Compute gradient for the error/cost function:

$$J = \sum_{i=1}^{n} \frac{1}{2} (y_i - \hat{y}_i)^2 \tag{1}$$

- Activation function (sigmoid function):

$$y_i = \sigma \left(\sum_{j=1}^{m} x_j w_{ij} - \theta_i \right) \tag{2}$$

- Separate the sum from the function:

$$y_i = \sigma (z_i) \tag{3}$$

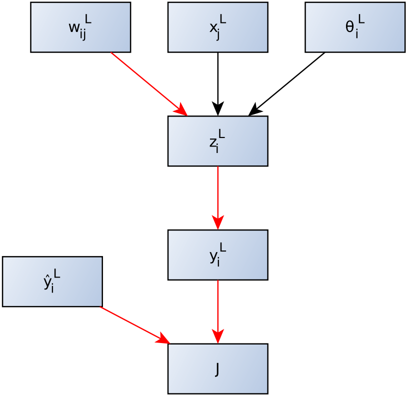
where

$$z_i = \sum_{j=1}^{n} x_j w_{ij} - \theta_i \tag{4}$$

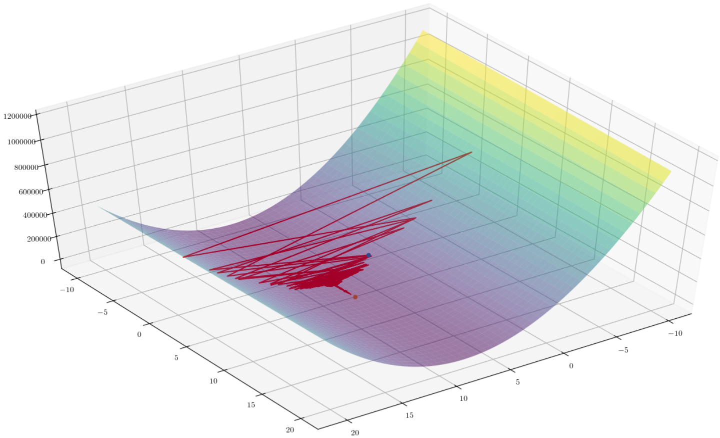
Back-propagation /recap/

$$\frac{\partial J}{\partial w_{ij}^L} = \frac{\partial z_i^L}{\partial w_{ij}^L} \frac{\partial y_i^L}{\partial z_i^L} \frac{\partial J}{\partial y_i^L}$$

$$\frac{\partial J}{\partial w_{ij}^L} = \sum_{j=1}^{m} y_j^{L-1} \sigma' \left(z_i^L \right) \left(y_i^L - \hat{y}_i \right)$$



Gradient descent



Usefulness of Gradient descent

$$y = Xw, \tag{5}$$

- where X is a n by $\#dim$ matrix containing input data, y are the "labels" or output data, and w are the weights used to generate the data (unknown in real world)
- Standard representation of a system of linear equations:

$$Ax = b$$

- In our case, $b \approx y$, $A \approx X$, and $x \approx w$, therefore we can rewrite equation (5) as:

$$Xw = y$$

- and to calculate w, we want to find values satisfying:

$$w^* = \arg\min_w \|Xw - y\|^2 \tag{6}$$

Usefulness of Gradient descent

- Equation equation (6) can be solved* by:

$$X^{-1}Xw = X^{-1}y$$

$$\mathcal{I}w = X^{-1}y$$

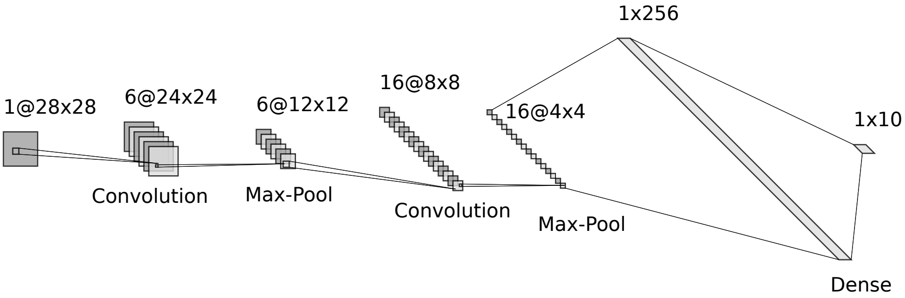
- For simple data, this method is faster and more efficient!
- So, why GD?
  - higher dimensionality
  - larger datasets
  - $\sim$ anything that cannot actually be computed via analytical solution with the available resources

Convolutional neural networks part 2

- Motivation
  - larger input size (RGB image $32 \times 32 \times 3 \sim 3000$ weights per neuron)
  - patterns in the input can shifted $\rightarrow$ in FCN small change in input = large in output – bad for images
- Hyperparameters
  - # of inputs (input dimension)
  - kernel/filter size
  - # of filters
  - padding & stride
- Often used in combination with FCN

Simple CNN on MNIST

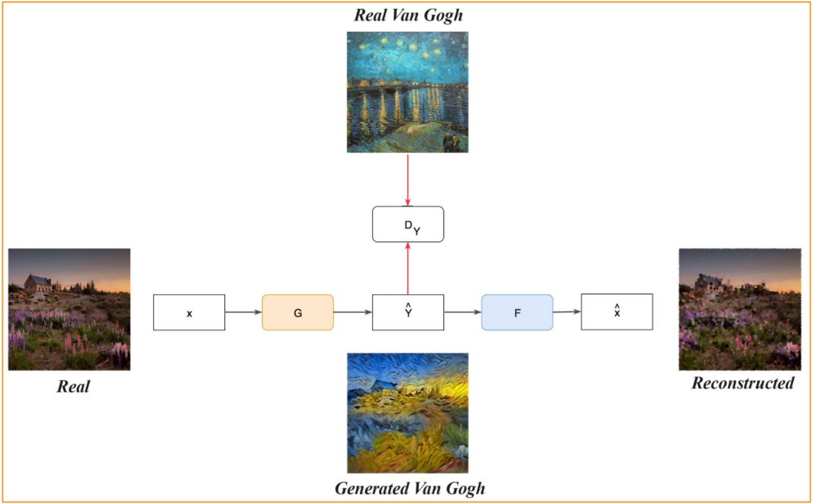
Simple CNN on MNIST



Generative adversarial networks

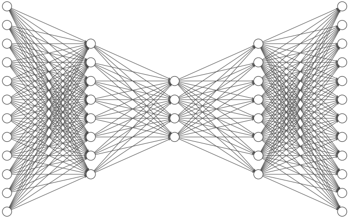
- Data sampled from an unknown distribution
  - only samples available
- Generator G network attempts to generate similar data
- Discriminator D attempts to recognize “fake” data
- Both G and D are trained together
  - G attempts to approximate the original distribution
  - D improves its classification error
- Benefits
  - less training data required
  - artificial data generation
  - “image arithmetics
- Problems:
  - convergence, discrepancy between G and D
- One of the most interesting concepts in ML in the past years
  - many variations exist

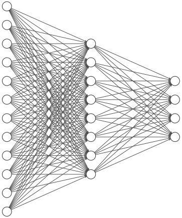
Generative adversarial networks

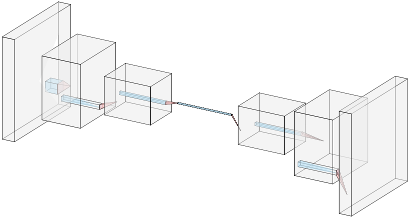


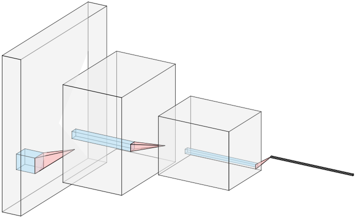
Autoencoders

- FCN & CNN autoencoders









Autoencoders

- Network is trained to generate the same values on output as were provided on input
- Afterwards, the "top half" of the network is discarded
- Uses:
  - dimensionality reduction
  - visualization
  - encoding (compression) with decoder
  - combination with GAN

Thank you for your attention