Data Mining and Machine Learning

Madhavan Mukund

Lecture 18, Jan–Apr 2020

https://www.cmi.ac.in/~madhavan/courses/dmml2020jan/

Loss functions (costs) for neural networks

- So far, we have assumed mean sum-squared error as the loss function.
- Consider single neuron, two inputs $x = (x_1, x_2)$

$$C = \frac{1}{n} \sum_{i=1}^{n} (y_i - a_i)^2, \text{ where } a_i = \sigma(z_i) = \sigma(w_1 x_1^i + w_2 x_2^i + b)$$

- For gradient descent, we compute $\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b}$
  - For $j = 1, 2$,

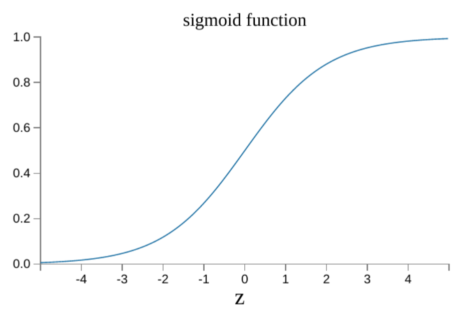
    $$\frac{\partial C}{\partial w_j} = \frac{2}{n} \sum_{i=1}^{n} (y_i - a_i) \cdot -\frac{\partial a_i}{\partial w_j} = \frac{2}{n} \sum_{i=1}^{n} (a_i - y_i) \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial w_j}$$

    $$= \frac{2}{n} \sum_{i=1}^{n} (a_i - y_i) \sigma'(z_i) x_j^i$$

  - $$\frac{\partial C}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (a_i - y_i) \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (a_i - y_i) \sigma'(z_i)$$

Loss functions ...

- $\frac{\partial C}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)x_j^i, \ \frac{\partial C}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(a_i - y_i)\sigma'(z_i)$
- Each term in $\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b}$ is proportional to $\sigma'(z_i)$
- Ideally, gradient descent should take large steps when $a - y$ is large
  - $\sigma(z)$ is flat at both extremes
  - If a is completely wrong, $a \approx (1 - y)$, we still have $\sigma'(z) \approx 0$
  - Learning is slow even when current model is far from optimal



Cross entropy

- A better loss function

$$C(a,y) = \begin{cases} -\ln(a), & \text{if } y = 1 \\ -\ln(1-a), & \text{if } y = 0 \end{cases}$$

  - If $a \approx y$, $C(a,y) \approx 0$ in both cases
  - If $a \approx 1-y$, $C(a,y) \to \infty$ in both cases

- Combine into a single equation

$$C(a,y) = -[y\ln(a) + (1-y)\ln(1-a)]$$

  - $y = 1 \Rightarrow$ second term vanishes, $C = -\ln(a)$
  - $y = 0 \Rightarrow$ first term vanishes, $C = -\ln(1-a)$

- This is called cross entropy

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y)\ln(1-\sigma(z))]$

- $$\begin{aligned}\frac{\partial C}{\partial w_j} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial w_j} &= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial w_j} \\ &= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial z}\frac{\partial z}{\partial w_j} \\ &= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\sigma'(z)x_j \\ &= -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j\end{aligned}$$

Cross entropy and gradient descent ...

- $\dfrac{\partial C}{\partial w_j} = -\left[\dfrac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$

- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$

- Therefore, $$\begin{aligned}\frac{\partial C}{\partial w_j} &= -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j \\ &= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j \\ &= (\sigma(z) - y)x_j \\ &= (a-y)x_j\end{aligned}$$

- Similarly, $\dfrac{\partial C}{\partial b} = (a-y)$

- Thus, as we wanted, the gradient is proportional to $a-y$

- The greater the error, the faster the learning rate

Cross entropy . . .

- Overall,
  - $$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}(a_i - y_i)x_j^i$$
  - $$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(a_i - y_i)$$
- Cross entropy allows the network to learn faster when the model is far from the true one
- Other theoretical justifications to justify using cross entropy
  - Derive from goal of maximizing log-likelihood of model
  - Will be addressed in advanced ML course

Case study: Handwritten digits

- MNIST database has 1000 samples of handwritten digits $\{0,1,\ldots,9\}$

- Assume input *segmented* as individual digits

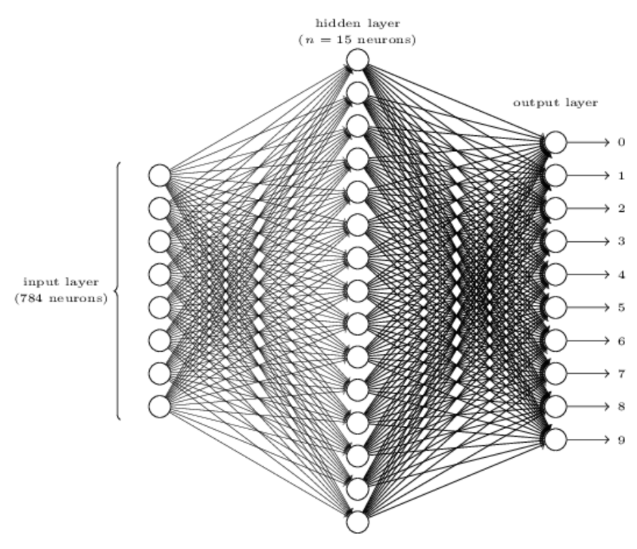
Handwritten digits ...

- Each image is $28 \times 28 = 784$ pixels
- Each pixel is a grayscale value from 0.0 (white) to 1.0 (black)
- Building a neural network
  - Linearize the image row-wise, inputs are $x_1, x_2, \ldots, x_{784}$
  - Single hidden layer, with 15 nodes
  - Output layer has 10 nodes, a decision a_j for each digit $j \in \{0, 1, \ldots, 9\}$
  - Final output is the maximum among these
    - ⋆ Naïvely, $\arg\max_j a_j$
    - ⋆ Softmax : $\arg\max_j \frac{e^{a_j}}{\sum_j e^{a_j}}$

      Smooth approximation to $\arg\max$

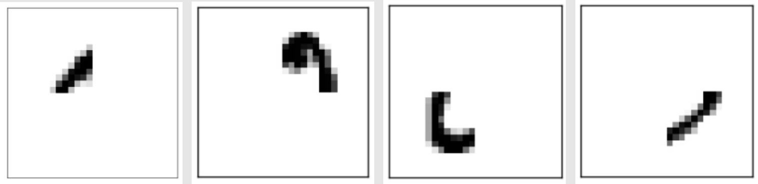
Handwritten digits . . .

Neural network to recognize handwritten digits

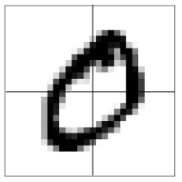


Handwritten digits . . .

- Intuitively, the internal node recognize features
- Combinations of features identify digits
- Hypothetically, suppose first four hidden neurons focus on four quadrants of image.



- This combination favours the verdict 0

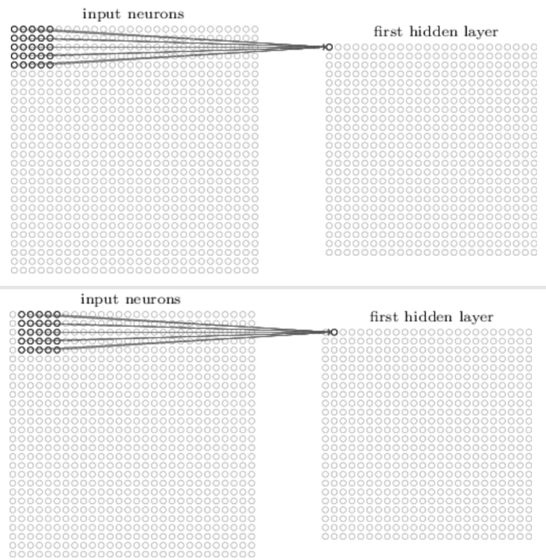


Identifying images

- Suppose we have a network that can recognize hand-written $\{0, 1, \ldots, 9\}$ of size 28×28
- We want to find these images in a larger image
- Slide a window of size 28×28 over the larger image
- Apply the original network to each window
- Pool the results to see if the digit occurs anywhere in the image

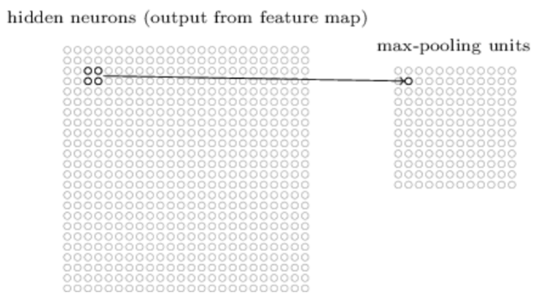
Convolutional neural network

- Each "window" connects to a different node in the first hidden layer.



Convolutional neural network . . .

- Sliding window performs a convolution of the window network function across the entire input
  - Convolutional Neural Network (CNN)
- Combine these appropriately — e.g., max-pool partitions features into small regions, say 2×2, and takes the maximum across each region



Convolutional neural network ...

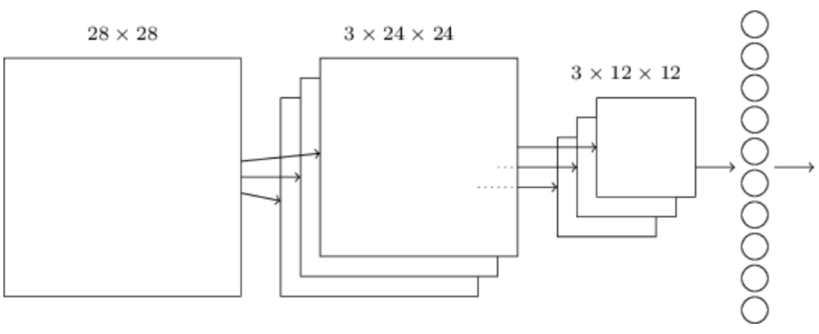
Example

- Input is 28×28 MNIST image
- Three features, each examines a 5×5 window
  - Construct a separate hidden layer for each feature
  - Each feature produces a hidden layer with 24×24 nodes
- Max-pool of size 2×2 partitions each feature layer into 12×12
  - Second hidden layer
- Finally, three max-pool layers from three hidden features are combined into 10 indicator output nodes for $\{0, 1, \ldots, 9\}$

Convolutional neural network ...

Network structure for this example

- Input layer is 28×28
- Three hidden feature layers, each 24×24
- Three max-pool layers, each 12×12
- 10 output nodes



Deep learning

- Hidden layers extract "features" from the input
- Individual features can be combined into more complex ones
- Networks with multiple hidden layers are called deep neural networks
  - How deep is deep?
  - Originally even 4 layers was deep.
  - Recently, 100+ layers have been used
- The main challenge is learning the weights
  - Vanishing gradients
  - Enormously large training data