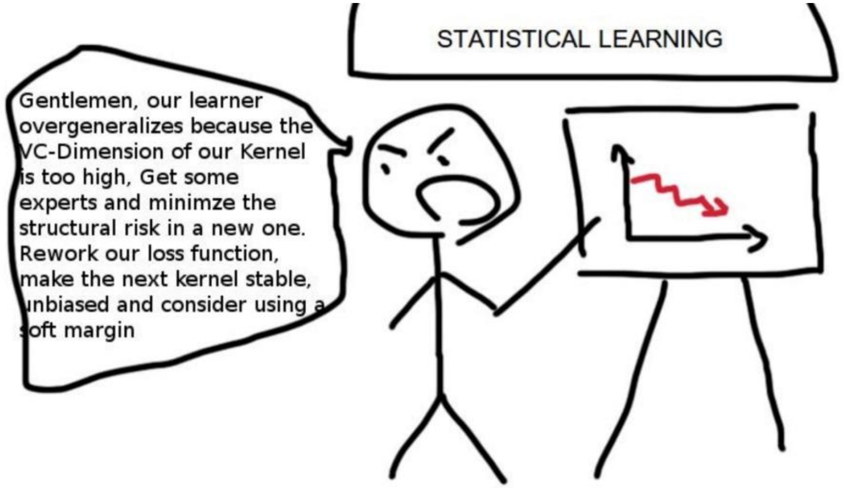
Multilayer Perceptrons

Gautam Kamath

Onto Neural Networks



Recall: The XOR Problem

- (Draw it)
- There is no linear separator which separates the +’s from the –’s
- Can prove it, but I won’t
- How can we solve this problem?
- Kernels
    - Apply mapping to data, use linear model on top
    - Can be some generic kernel, “hand-crafted” features (domain expertise), etc.
- Neural Network
    - *Learn* a mapping of data (from data), use linear model on top
    - Learn a *representation*

Drawing some old models

- (Draw perceptron in graphical form)
- $x \in \mathbf{R}^2, \tilde{y} = x_1 w_1 + x_2 w_2 + b = \langle x, w \rangle + b$
- (Add on sigmoid to output to make into logistic regression)
- $\text{sigmoid}(t) = \sigma(t) = \frac{1}{1+e^{-t}}$
- $\hat{y} = \frac{1}{1+\exp(-\langle w,x \rangle - b)}$ (logistic regression)

Drawing a simple multilayer perceptron

- (Draw 2LNN with width-2 hidden layer, x input, u and c first layer weights and biases, z hidden layer pre-activation, f non-linearity, h hidden layer post-activation, w and b second layer weights and biases, $\tilde{y}$ output)
- (Label input layer, hidden layer, representation layer, output layer, non-linear activation $\mathbf{R} \rightarrow \mathbf{R}$)

Some calculations for XOR

- $z = Ux + c, h = f(z), \tilde{y} = \langle h, w \rangle + b$
- Consider: $U = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, w = \begin{bmatrix} 2 \\ -4 \end{bmatrix}, b = -1$
  - Parameters to be learned, but suppose they're just given for now
- Choose $f(t) = \max(0, t) = \mathrm{ReLU}(t)$ (draw)
  - *Activation function* – this is a hyperparameter choice
- $x_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, y_1 = -1.\ z = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, h = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tilde{y} = -1$
- $x_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, y_1 = 1.\ z = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, h = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tilde{y} = 2 - 1 = 1$, etc.
- Adding the non-linear f really gave it a lot more powerful!

Drawing a bigger multilayer perceptron

- (Draw wider three-layer MLP, input $x \in \mathbf{R}^d$, output $\tilde{y} \in \mathbf{R}^m$)
  - (Illustrate depth and width, representation layer)
- $z^{(1)} = W^{(1)}x, h^{(1)} = f\left(z^{(1)}\right), z^{(2)} = W^{(2)}h^{(1)}, \ldots$
- What to do with output $\tilde{y} \in \mathbf{R}^m$?
  - Put through *softmax* to get distribution over m classes (confidences of each)
  - $\hat{y}_i = \frac{\exp(\tilde{y}_i)}{\sum_{j=1}^m \exp(\tilde{y}_j)}$
- What loss function? Use the *cross-entropy* loss
  - $\ell_\theta(x, y) = -\sum_{i=1}^m y_i \log \hat{y}_i$
    - Use "one-hot encoding" of y: if $y = c$, then $y_c = 1$, and $y_i = 0$ for other entries
    - $\hat{y} = g_\theta(x)$, where g_θ is a (somewhat complicated) function

Activation Functions (Draw)

- Non-linear
- Sigmoid: $\sigma(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t}$
- Tanh: $\tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$
- ReLU: $\mathrm{relu}(t) = \max(0, t)$
  - Still has a strong gradient signal even if t is large

Training

- Loss function: $\arg\min_\theta L = \frac{1}{n}\sum_i \ell_\theta\left(x^{(i)}, y^{(i)}\right)$
  - Recall $\ell_\theta(x, y) = -\sum_{j=1}^{m} y_j \log \hat{y}_j$
- Just use gradient descent!
  - $\theta^t = \theta^{t-1} - \eta_t \nabla L_{\theta^{t-1}}$
- But… how to compute ∇L?
  - Recall $\hat{y} = g_\theta(x)$ is some complicated function… how to take derivative?
  - Luckily computers are very good at this
  - Automatic differentiation, backpropagation algorithm
    - Chain rule + dynamic programming

Backpropagation (a simple example)

- Simple case: $x \in \mathbf{R}$, $f, g: \mathbf{R} \rightarrow \mathbf{R}$
- Say $y = g(x), z = f(y) = f(g(x))$
- Chain rule: $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
- More complex: $z = ux, h = f(z), y = wh, L = g(y)$ (draw)
- Can compute several derivatives easily: $\frac{dL}{dy}, \frac{dy}{dw}, \frac{dy}{dh}, \frac{dh}{dz}, \frac{dz}{du}$
- But we care about derivatives of L wrt parameters u, w
- $\frac{dL}{dw} = \frac{dL}{dy} \cdot \frac{dy}{dw}$ and $\frac{dL}{du} = \frac{dL}{dy} \cdot \frac{dy}{dh} \cdot \frac{dh}{dz} \cdot \frac{dz}{du}$

Backpropagation (multivariate)

- Say $x \in \mathbf{R}^m, y \in \mathbf{R}^n, z \in \mathbf{R}$
- $z = f(y) = f(g(x))$
- Then $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$ (draw, must sum over all “paths” from x_i to z)
- Alternate notation: $\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z$
  - Product of Jacobian matrix and gradient
  - But the big picture: gradients can be decomposed as product of appropriate derivatives of subsequent layers using chain rule
  - Need appropriate dynamic programming to do efficiently (backpropagation)

Universality of MLPs

- Any continuous function $g: [0,1]^d \rightarrow \mathbf{R}$ can be approximated arbitrarily well by some 2 layer NN with arbitrary non-polynomial activation
- (maybe draw)
- Caveat: may require the hidden layer to be incredibly wide