IFT 6135: Representation Learning

Michael Noukhovitch

Winter 2019

Notes written from Aaron Courville's lectures.

Contents

1 Neural Networks

1.1 Artificial Neuron

$g(b + w^T x)$

pre-activation $b + w^T x$

connection weights w

neuron bias b

activation function g

1.2 Activation Functions

linear a

sigmoid $\frac{1}{1+\exp(-a)}$

tanh $\frac{\exp a - \exp -a}{\exp(a)+\exp(-a)}$

ReLU $\max(0, a)$

maxout $\max_{j\in[1,k]} a_j$

softmax $\frac{\exp(a_i)}{\sum_c \exp(a_c)} \forall i$

1.3 Neural Networks

1.3.1 Single Layer

neuron capacity a single neuron can do binary classification iff linearly separable

universal approximation theorem (Hornik, 1991) a single-layer NN can approximate any continuous function given enough hidden units

1.3.2 Multi Layer

input $h^{(0)} = x$

hidden layer pre $a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)$

hidden layer activation $h^{(k)} = g(a^{(k)}(x))$

output $h^{(L+1)}(x) = o(a^{(L+1)}(x))$

1.4 Biological Inspiration

2 Training Neural Networks

2.1 Empirical Risk Minimization

learning as optimization

$$\operatorname*{argmin}_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)};\theta), y^{(t)}) + \lambda\Omega(\theta)$$

loss function l is a surrogate for what we truly want (upper bound)

regularizer Ω penalizes certain values of θ

2.2 Stochastic Gradient Descent

initialize θ
for n epochs **do**
  foreach training example $x^{(t)}, y^{(t)}$ **do**
    $\Delta \leftarrow -\nabla_{\theta} l(f(x^{(t)};\theta), y^{(t)}) - \lambda\nabla_{\theta}\Omega(\theta)$
    $\theta \leftarrow \theta + \alpha\Delta$
  end
end

gradient ∇

learning rate α

this requires:

- a loss function
- gradient computation 2.3
- a regularizer 2.4
- initialization 2.5

2.3 Gradient Computation

2.3.1 Manual

given a categorical output with classes c, let softmax output be $f(x)_c = p(y = c|x)$

gradients for output

$$\frac{\partial}{\partial f(x)_c} - \log f(x)_y = \frac{-1_{(y=c)}}{f(x)_y}$$

$$\nabla_{f(x)} - \log f(x)_y = \frac{-1}{f(x)_y} \begin{bmatrix} 1_{(y=0)} \\ \dots \\ 1_{(y=C-1)} \end{bmatrix}$$

$$= \frac{-e(y)}{f(x)_y}$$

gradients for pre-activation

$$\frac{\partial}{\partial a^{(L+1)}(x)} - \log \sigma(a^{(L+1)}(x))_y = -(1_{(y=c)} - f(x)_y)$$

$$\nabla_{f(x)} - \log f(x)_y = -(e(y) - f(x))$$

where $e(y)$ gives the one-hot vector of length C with 1 at index y, and σ is the sigmoid function

2.3.2 Backpropogation

To simplify things, use the chain rule to rewrite gradients in terms of in terms of the layers above them

compute output gradient $\nabla_{a^{(L+1)}(x)} - \log f(x)_y = -(e(y) - f(x))$
for $k = L + 1 \to 1$ **do**
  hidden layer weights
    $\nabla_{W^{(k)}(x)} - \log f(x)_y = (\nabla_{a^{(k)}(x)} - \log f(x)_y) h^{(k-1)}(x)^T$
  hidden layer biases
    $\nabla_{b^{(k)}(x)} - \log f(x)_y = \nabla_{a^{(k)}(x)} - \log f(x)_y$
  output below
    $\nabla_{h^{(k-1)}(x)} - \log f(x)_y = W^{(k)^T} (\nabla_{a^{(k)}(x)} - \log f(x)_y)$
  pre-activation below
    $\nabla_{a^{(k-1)}(x)} - \log f(x)_y = (\nabla_{h^{(k-1)}(x)} - \log f(x)_y) \bigodot [\dots, g'(a^{(k-1)}(x)_j, \dots]$
end

2.3.3 Flow Graph

represent execution as a modular, acyclic flow graph of boxes with

- method `fprop` children $\to$ parents
- method `bprop` parents $\to$ children

debug with **finite difference approximation**

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x - \epsilon) - f(x + \epsilon)}{2\epsilon}$$

2.4 Regularization

2.4.1 Methods

L2 $\sum_{k,i,j}(W_{i,j}^{(k)})^2$

- gradient $2W^{(k)}$
- like a gaussian prior

L1 $\sum_{k,i,j}|W_{i,j}^{(k)}|$

- gradient $\mathrm{sign}(W^{(k)})$
- laplacian prior, pushes weights to be 0

early stopping stop training when validation error increases (with lookahead)

2.4.2 Bias-Variance Tradeoff

for a learning algorithm:

variance variance between using different training sets

bias difference between average model and true solution

2.5 Initialization

- bias $\to 0$
- weights $\sim$ `uniform`$(-b, b)$, $\quad b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$
  - 0 doesn't work for tanh
  - same value makes everything behave same

2.6 Model Selection

training set train the model

validation set select hyperparameters

test set estimate generalization error

grid search try all hyperparameters

random search sample distribution of hyperparameters

2.7 Optimization

2.7.1 SGD

Training NN with SGD

- **non-convex** because there isn't a global optimum
- convergence if $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

Tricks

decaying learning rate e.g. $\frac{\alpha}{1+\delta t}$

mini-batching using > 1 example for gradient computation

exponentially decaying average of previous gradients

2.7.2 Newton's Method

locally approximate loss using Taylor Expansion, minimize by solving

$$\begin{aligned}0 &= \nabla_\theta l(f(x;\theta^{(t)}), y) + (\nabla_\theta^2 l(f(x;\theta^{(t)}), y))(\theta - \theta^{(t)}) \\ \theta^{(t+1)} &= \theta^{(t)} - (\nabla_\theta^2 l(f(f;\theta^{(t)}), y))^{-1}(\nabla_\theta l(f(x;\theta^{(t)}), y))\end{aligned}$$

but only practical if there are few parameters (to invert Hessian), and locally convex (invertible Hessian)