IFT 6135: Representation Learning

Michael Noukhovitch

Winter 2019

Notes written from Aaron Courville's lectures.

Contents

1 Neural Networks **3**
1.1 Artificial Neuron . . . 3
1.2 Activation Functions . . . 3
1.3 Neural Networks . . . 3
1.3.1 Single Layer . . . 3
1.3.2 Multi Layer . . . 3
1.4 Biological Inspiration . . . 4

2 Training Neural Networks **4**
2.1 Empirical Risk Minimization . . . 4
2.2 Stochastic Gradient Descent . . . 4
2.3 Gradient Computation . . . 4
2.4 Backpropogation . . . 5
2.5 Flow Graph . . . 5

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

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

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

where ∇ is the gradient, α is the learning rate

2.3 Gradient Computation

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

gradients for output

$$\begin{aligned} \frac{\partial}{\partial f(x)_c} - \log f(x)_y &= \frac{-1_{(y=c)}}{f(x)_y} \\ \nabla_{f(x)} - \log f(x)_y &= \frac{-1}{f(x)_y} \begin{bmatrix} 1_{(y=0)} \\ \cdots \\ 1_{(y=C-1)} \end{bmatrix} \\ &= \frac{-e(y)}{f(x)_y} \end{aligned}$$

gradients for pre-activation

$$\begin{aligned} \frac{\partial}{\partial a^{(L+1)}(x)} - \log \sigma(a^{(L+1)}(x))_y &= -(1_{(y=c)} - f(x)_y) \\ \nabla_{f(x)} - \log f(x)_y &= -(e(y) - f(x)) \end{aligned}$$

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

2.4 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

$\nabla_{a^{(k-1)}(x)} - \log f(x)_y = (\nabla_{h^{(k-1)}(x)} - \log f(x)_y) \bigodot [\ldots, g'(a^{(k-1)}(x)_j, \ldots]$

end

2.5 Flow Graph