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 $\theta$
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

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

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

2.3.2 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 $\text{sign}(W^{(k)})$
- 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 $\sim \texttt{uniform}(-b, b), \quad b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$
  - 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 First-Order Optimization

2.7.1 SGD

stochastic gradient descent

1. sample a minibatch of m examples
2. gradient estimate $\hat{h} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(x_i, y_i; \theta)$
3. apply update $\theta \leftarrow \theta - \epsilon \hat{h}$

properties

- **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 Momentum

keep momentum of previous updates,

0. momentum parameter α
1. gradient estimate h
2. velocity update $v \leftarrow \alpha v - \epsilon h$
3. update $\theta \leftarrow \theta + v$

properties

- accelerate learning with small, consistent gradients
- gradients that are consistently seen will accumulate
- max step size ("terminal velocity") $\frac{1}{1-\alpha} \epsilon ||g||$

2.7.3 Nesterov Momentum

Sutskever et al (2013) evaluate the gradient after an interim velocity update

0. momentum parameter α
1. apply interim velocity update $\tilde{\theta} \leftarrow \theta + \alpha v$
2. gradient estimate h at interim using $\tilde{\theta}$
3. velocity update $v \leftarrow \alpha v - \epsilon h$

4. update $\theta \leftarrow \theta + v$

intuition

- first make a big jump in previously accumulated gradient
- measure gradient where you end up
- make a correction jump

2.7.4 Adagrad

Duchi et al (2010) adapt the learning rate per parameter proportional to sum of squares of that gradient

0. numerical stability const δ
0. initialize gradient accumulation $r = 0$
1. gradient estimate h
2. accumulate squared gradient $r \leftarrow r + h \odot h$
3. per-parameter update $\Delta\theta \leftarrow -\frac{\epsilon}{\delta+\sqrt{r}} \odot h$
4. update $\theta \leftarrow \theta + \Delta\theta$

adapted learning rates

1. makes later learning very slow
2. great for convex, not so much for NN

2.7.5 RMSProp

Hinton (2012?) modifies AdaGrad accumulation into exponentially moving average

0. decay rate ρ, numerical stability const δ
0. initialize gradient accumulation $r = 0$
1. gradient estimate h
2. accumulate squared gradient $r \leftarrow \rho r + (1-\rho) h \odot h$
3. per-parameter update $\Delta\theta \leftarrow -\frac{\epsilon}{\delta+\sqrt{r}} \odot h$
4. update $\theta \leftarrow \theta + \Delta\theta$

you can add nesterov momentum $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot h$

generally better with NNs

- performs better in non-convex setting
- decay ρ allows control of length scale of EMA
- effective with NNs but can be difficult to tune hyperparams

2.7.6 Adam

"Adaptive Moments" (Kingma et al, 2014)

0. step size ϵ, default 0.001
0. decay rates for moment estimates ρ_1, ρ_2 defaults $0.9, 0.999$
0. numerical stability const δ
0. initialize 1st and 2nd moment variables $s = 0, r = 0$
1. gradient estimate h
2. update biased 1st moment estimate $s \leftarrow \rho_1 s + (1 - \rho_1)h$
3. update biased 2nd moment estimate $r \leftarrow \rho_2 r + (1 - \rho_r)h \odot h$
4. correct bias in 1st moment $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
5. correct bias in 2nd moment $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
6. per-parameter update $\Delta\theta \leftarrow -\epsilon \frac{\hat{s}}{\delta+\sqrt{\hat{r}}} \odot h$
7. update $\theta \leftarrow \theta + \Delta\theta$

variant of RMSProp + momentum with corrections

- momentum incorporated directly as estimate of first order moment (with exponential weighting) of the gradient
- bias corrections to estimates of first-order momentum (momentum) and second-order moment to account for init at origin

2.7.7 AdamW

Loschilov and Hutter (2019) decouple weight decay from optimization wrt loss function

1.

in SGD, reparametrizing L_2 hyperparam λ based on learning rate can make it equivalent to weight decay

$$\begin{aligned} L_2 &= \frac{\lambda'}{2}||\theta||_2^2 \\ \theta &\leftarrow \theta - \epsilon\nabla_\theta(L + L_2) \\ &\leftarrow \theta - \epsilon\nabla_\theta L - \epsilon\lambda'\theta \end{aligned}$$

set $\lambda' = \frac{\lambda}{\epsilon}$

$$\leftarrow (1 - \lambda)\theta - \epsilon\nabla_\theta L \text{ which is weight decay}$$

but this equivalence may not work

- it doesn't hold for adaptive methods like Adam
- it requires you to set $\lambda' = \frac{\lambda}{\epsilon}$ which might not be the best

2.8 Second-Order Optimization

2.8.1 Newton's Method

approximate loss $J(\theta)$ near some point θ_0 using Taylor Expansion

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)$$

taking the gradient

$$\nabla_\theta J(\theta) \approx \nabla_\theta J(\theta_0) + H(\theta - \theta_0)$$

and solving for the critical point $\nabla_\theta J(\theta) = 0$

$$\theta^* = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$$

this jumps directly to the minimum in a quadratic convex problem but as long as H is positive definite, can be applied iteratively in non-quadratic.

1. comute gradient g
2. compute Hessian H
3. compute inverse Hessian H^{-1}
4. apply update $\theta \leftarrow \theta - H^{-1} g$

but inverting is $O(n^3)$, memory is $O(n^2)$ and models are too large

2.8.2 Conjugate Gradient

just use 2nd order info to find directions conjugate to previous using β_t, not undo progress

1. comute gradient g_t
2. compute $\beta_t = \frac{(g_t - g_{t-1})^T g_t}{g_{t-1}^T g_{t-1}}$
3. compute search direction $\rho_t = -g_t + \beta_t \rho_{t-1}$
4. perform line search and find $\epsilon^* = \text{argmin}_\epsilon \, J(\theta_t + \epsilon \rho_t)$
5. apply update $\theta \leftarrow \theta - \epsilon^* \rho_t$