Deep Learning Summer School

Michael Noukhovitch

Summer 2018

Contents

1 Neural Networks I (Hugo Larochelle)

1.1 Basics

1.1.1 Artificial Neuron

pre-activation $a(x) = w^T x + b$

activation function $h(x) = g(a(x))$

1.1.2 Capacity of Neural Networks

combining two simple linear neurons, can create a more complex non-linear shape

universal approximation a single hidden layer NN can approximate any continuous function arbitrarily well (given enough neurons)

1.1.3 Multilayer Neural Network

we can have L hidden layers, e.g. at layer k

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

output layer $(k = L + 1)$

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

1.1.4 Activation Function

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

tanh $\frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$ squashes between -1 and 1

relu $\max(0, a)$ does not saturate

softmax multi-class conditional probabilities

1.2 Training Neural Networks

1.2.1 Optimization

learning as an optimization problem

loss function $l(f(x; \theta), y)$ between output and true label

regularizer $\Omega(\theta)$ penalize certain parameters θ

use **Stochastic Gradient Descent**

1. initialize θ

   for N epochs

for each example x, y

2. calculate gradient $\Delta = -\nabla l(\dots)$
3. take a step $\theta \leftarrow \theta + \alpha\Delta$

1.2.2 Loss Function

maximize the probability of correct class (**maximum likelihood**)
equivalently **minimize** the negative log-probability aka **cross-entropy**

$$l(f(x), y) = -\sum_c 1_(y = c) \dots$$

1.2.3 Backpropogation

use chain rule to compute gradients

1. compute gradient wrt pre-activation

$$\nabla_{a^{(L+1)}(x)} - \log f(x)_y \leftarrow -(e(y) - f(x))$$

for layer k from $L + 1$ to 1

2. compute gradient of hidden layer parameter
3. compute gradient of hidden layer below
4. compute gradient of pre-activation below

reversing the flow graph representation gives us backprop for free Section 2

1.2.4 Initialization

- biases $\leftarrow 0$
- weights $\leftarrow$ random sample
  - 0 doesn't work with tanh
  - same value doesn't work
  - break symmetry, close to 0

1.2.5 Model Selection

search for the best hyperparameters with

- **grid search** search all possible options
- **random search** sample from distribution over hyperparams
- bayesian optimization, pbt, ...

use a **validation set** of examples to choose the best model
stop training with **early stopping** when validation error is lowest

1.2.6 Tricks

- **normalize** your (real-valued) data
- **decay** your learning rate
- update your gradient on a **batch** of examples
- use exponential average of previous gradients, ”gaining **momentum**”
- use adaptive learning rates: **Adagrad**, **RMSProp**, **Adam**

1.2.7 Gradient Checking

debug your implementation of fprop/bprop with a finite-difference approximation

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

1.2.8 Debug on Small Dataset

overfit on a subsubset of your dataset, issues:

- units saturated before first update? initialization, regularization
- training error unstable? learning rate scheduling

1.3 Regularization

1.3.1 Unsupervised Pretraining

unsupervised pretraining initialize hidden layers by using unsupervised learning to represent the latent structure of data

fine-tuning training after initialization to adapt to data

auto-encoder feed-forward NN traine to reproduce it's input

1.3.2 Dropout

dropout remove hidden units stochastically (usually $p = 0.5$)

1.3.3 Batch Normalization

normalize inputs to speed up training

- normalize each unit's pre-activation
- subsubtract mean and std deviation, calculated per minibatch
- learn a linear transformation of the normalized pre-activation
- account for it during backprop
- use global mean and std deviation at test time

2 Automatic Differentiation **(David Duvenaud)**

2.1 Basic Autodiff

auto diff programmatically finding gradients for operations

forward mode building the jacobian on, in the order of initial operations (expensive)

reverse mode keeping track of jacobian at every step and adding then in reverse

2.1.1 Implementation

static read and generate source code (Tensorflow)

dynamic monitor function exection at runtime (Pytorch, autograd)

1. trace execution as composition of primitives
2. define vector-Jacobian product (VJP) operator for each primitive
3. compose VJPs backwards

fun stuff:

- we get higher order autodiff for free
  since tape-based dynamic autodiff traces execution, we automatically trace the autodifferentiation itself, and just need to follow the execution trace of the previous autodiff
- forward mode is actually just a special case of reverse mode
- VJPs are as cheap as gradients

2.2 Advanced Autodiff

2.2.1 Higher Order Ops

higher order gradients are possible by changing vector-Jacobian for vector-Hessian etc ...

2.2.2 Meta-Optimization

can optimize through the whole network to learn the learning rate for the training

2.2.3 Implicit Function Theorem

3 Neural Networks II **Hugo Larochelle**

3.1 Types

supervised examples have labels

unsupervised examples don't have labels

semi-supervised some examples have labels

multi-task multiple labels per example

transfer multiple labels but test on a specific label

structured labels have arbitrary structure

domain adaptation training and testing distributions are different

zero-shot examples are completely novel

3.2 Intriguing Properties

- there are powerful small changes that create visual adversarial examples
- bad local optima are unlikely (for high dimensional loss surface like NNs)
- NNs are strangely non-convex
- flat minima are better than sharp minima (probably?)
- NNs can easily memorize
- knowledge can be distilled
- catastrophic forgetting is a real issue with SGD

4 Intro to Convolutional Neural Networks **Jon Shlens**

4.1 Convolutional Neural Networks

images are different from other data

- fully connected layers would use too many parameters to input an image
- convolutions provide translational invariance

convolutional neural networks

- learn filters that pass over the image
- use fewer parameters
- use more computations

4.2 Modern Developments

- normalization stabilizes activations and is important for training
  Group Normalization (Wu and He 2018)
- vanishing gradients motivate deeper, better architectures
- architectures transfer across tasks
- learned architecture searches find even better models
  NAS (Zoph 2016), DARTS (Liu et al 2018)

4.3 Understanding CNNs

how to approach what CNNs understand at each layer?

- find images/pixels that elicit largest activation
- reconstruct image from network activations
- distort pixels to amplify activations (deep dream)
- change lower level activations while maintaining high-level (neural style transfer)