Training Neural Networks I

overview

- ▼ One time setup
  - ▼ activation functions, preprocessing, weight initialization, regularization, gradient checking
- ▼ Training dynamics
  - ▼ babysitting the learning process, parameter updates, hyperparameter optimization
- ▼ Evaluation
  - ▼ model ensembles

Activation Functions

1. Sigmoid
   a. 3 problems
      - saturated neurons kill the gradients
        - sigmoid function is flat for too negative, too positive
      - sigmoid outputs are not zero-centered
        - consider what happends when the input to a neuron x is always positive
        - the gradients on w always all positive or all negative
        - this is also why you want zero-mean data
        - inefficient gradient update
      - exp is a bit compute expensive
2. Tanh
   a. zero centered
   b. problems
      - still kills gradients when saturated

3. ReLU (rectified linear unit)
    a. does not saturate (in +region)
    b. very computationally efficient
    c. converges much faster than sigmoid/tanh in practice (e.g., 6x)
    d. actually more biologically plausible than sigmoid
    e. problems
        - not zero-centered output
        - an annoyance
            - what is the gradient when x<0?
            - do have saturation when x is negative
            - kill the gradient in half of the region (active relu, dead relu)
            - dead relu will never actiate → never update
            - happends through training (e.g., too big learning rate)
4. Leaky ReLU
    a. does not saturate
    b. computationally efficient
    c. convergew much faster than sigmoid/tanh in practice
    d. will not die
5. PReLU (parametric rectifier)
6. ELU (exponential linear units)
    a. all benefits of relu
    b. closer to zero mean outputs
    c. negative saturation regime compared with leaky relu adds some robustness to noise
    d. problem
        - computation requires exp
7. Maxout neuron
    a. does not have the basic form of dot product → nonlinaerity

b. generlizes relu and leaky relu
c. linear regime
d. does not saturate
e. does not die
f. problem
    - doubles the number of parameters/neuron

Data Preprocessing

1. Preprocess the data
    a. zero center
    b. normalize
        i. 이미지는 별로 안함
    c. decorrelate
    d. whitening
2. Weight initilization
    a. what happens when w=0 init is used?
        - all the neurons are doing the same operation
        - output the same thing with same gradient and update in the same way
        - all neurons are exactly the same
    b. Small random numbers
        i. gaussian with zero mean and 1e-2 standard deviation
        ii. works okay for small networks but problems with deeper networks
            - standard deviation shrinks near the 0
        iii. all activations become zero
        iv. think about the backward pass. what do the gradients look like
            - x is samll, weights are very small, not update
        v. weights big, sample from 1 instead of 0.01

    vi. almost all neurons completely saturated, either -1 and 1
        - gradients will be all zero

3. Xavier initialization
    a. reasonale initialization
        i. mathematical derivation assumes linear activations
    b. variance input = variance output
    c. small umber of inputs divide small value → large weight
    d. many inputs smaller weights
4. batch normalization
    a. a batch of activations at some layer
    b. to make each dimension unit gaussian, apply a vanilla differentiabler function
    c. usually inserted arfter fully connected or convolutional layers and before nonlinearity
    d. per activation map one mean one ddeviation for conv
    e. problem
        - do we necessariliy want a unit gaussian input to a tanh layer
    f. normalize and then allow the network to squash therange if it wants to
    g. note the network an learn to recover the identity mapping
    h. property
        - improves gradient flow through the network
        - allows higher learning rates
        - reduces the strong dependence on initialization
        - acts as a form of regularization in a funny way, and slightly reduces the need for tdropout, maybe
    i. note: at test time batchnorm layer functions differently
        - the mean std are not computed based on the batch
        - instead a single fixed empirical mean of activations during training is used

- e.g., can be estimated during training with running averages

Babysitting the Learning Process

1. Choose the architecture
    a. say we start with one hidden lyaer of 50 neurons
    b. double check that the loss is reasonable
    c. lets try to train now
        - make sure that you can overfit very small portion of the training data
    d. start with small regularization and find learning rate that makes the loss go down
        - loss not going down: learning rate too low
    e. notice train/val accuracy goest to 20% though, what’s up with that?
        - remember this is softmax
        - loss exploding : learning rate too high
2. Hyperparameter optimization
    a. cross-validation strategy
        i. coarse → find cross-validation in stages
    b. first: only a few epochs to get rough idea of what params work
    c. second: longer running time, finer search
    d. tip for detecting explosions in the solver: if the cost is ever > 3 * original cost, break out early
    e. note it’s best to optimize in log space
    f. now run finer search
    g. adjust range
    h. random search vs grid search
    i. hyperparameters to play with
        - network architecture
        - learning rate, its decay schedule, update type

- regularization (l2/dropout strength)

j. big gap → overfitting → increase regularization strength?
k. no gap → increase model capacity?
l. track the ratio of weight updates / weight magnitudes