Presentation Notes

Ankur Mishra

5/11/2018

Contents

1 Neural Networks

1.1 Backpropagation

Way of computing the influence of every value on a computational graph by recursively using multivariable chain rule through the graph.

Chain Rule: $dL/dx = dL/dz * dz/dx$

You can break your backprop into other functions and find their derivatives. An example of this is breaking $\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$ into the sigmoid function: $\frac{1}{1+e^{-x}}$

- plus (+) gate distrubutes gradients equally
- max gate routes gradient to max
- multiply (*) switches inputs and multiplies each by global gradient, equal inputs are picked arbitrarily

If two gradients combine when backpropagating –> add their gradients Linear Score Function: $f = W * x$
Two Layer Neural Network: $f = W_2 * max(0, W_1 * x)$
or Three Layers N-Network $f = W_3 * max(W_2 * max(0, W_1 * x))$
Bigger Networks are more powerful.

1.2 Image Pre-processing

Commonly pre-processing of images is done by mean centering. This either means to to subtract the mean value of each pixel by a [32,32,3] array, or to find the per-channel mean, which is subtract the mean from each pixel's RGB channels.

1.3 Weight Initialization

Setting weights to 0 will return 0 throughout network. Even .01 returns near zero values over the last few layers of a network in both forward and backward pass, which is known as vanishing gradient. Setting weight to 1, will supersaturate network, as all neurons come out as -1 or 1. The solution is Xavier initialization.

1.3.1 Xavier Initialization

W = np.random.randn(fan_{in}, fan_{out}) / np.sqrt(fan_{in}) for tanh(x). This breaks when using ReLU, so use W = np.random.randn(fan_{in}, fan_{out}) / np.sqrt(fan_{in} / 2) for ReLU.

1.4 Batch Normalization

This is to normalize data where you apply this equation to each layer:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{(Var(x^{(k)}))}}$$

Which is a vanilla differentiable function. What it does is it computes the mean of every feature and then divides by it.

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

After this the function scaled by γ and then is shifted by β, which changes the range if the network wants to. Through learning the network can either learn to take it out or take advantage of it.
The general process of this is first the mini-batch mean is computed, then its variance. Using these two things, the values are normalized and finally are scaled and shifted.

1.4.1 Perks of Using It

1. Improves Gradient Flow
2. Allows for High Learning Rates
3. Reduces dependence on strong weight initialization
4. Acts like regularization and slightly reduces need for dropout

2 Convolutional Neural Networks

2.1 Basics

Convolutions have height, width, and depth. Convolutional layer is the building block to a CNN. Take a filter and slide it accross the image, while computing dot products (convolve). This creates an activation map, whose dimensions are calculated by the number of distinct position it crosses. This is repeated for each filter and the the number of repeats will result in your new depth. If there is another the convolutional layer, then each filters depth will be the same as the new depth of the activation map.
Over each level of convolution a group of interesting pieces will be developed, and deeper levels will create templates for features found in the image.
In the activiation map, white corresponds to high activiations and blacker shades mean lower activations.

2.1.1 General Process

An image is processed by a convolutional layer, then a RELU layer and then it is repeated. After that, you pool it. Then after a certain number of convolutional layers, there is a fully connected layer at the end that will score the image accordingly.

1. Takes Volume of Size $W_1 x H_1 x D_1$
2. Takes Four Hyper Parameters
   (a) Number of Filters K
   (b) Their Spatial Extent F
   (c) The Stride S
   (d) The amount of 0 Padding P
3. Produces Volume of Size $W_2 x H_2 x D_2$

(a) $W_2 = \frac{W_1+F+2P}{S} + 1$

(b) $H_2 = \frac{H_1+F+2P}{S} + 1$

(c) $D_2 = K$

4. The number of parameters = (filter dimensions + $1_{\text{(for bias)}}$) * (number of filters)

2.2 Spatial Dimensions

2.2.1 Strides

$OutputSize = \frac{N-F}{stride} + 1$

2.2.2 Padding

Adding zero padded border for convenience as each layer stays the same dimensions and also the dimensions dont get smaller.

2.3 Pooling Layers

Make input volume smaller and more managable, by down sampling.

3 Markov Processes

- Where the environmnent fully observable
- Almost all RL problems can be characterized as MDPs

3.1 Markov Property

- P[$S_{(t+1)}$ | $S_{t]}$ = P[$S_{(t+1)}$| S_1, ..., $S_{t]}$
- Future is irrelavent of past, only related to present
- Given $S_{(t)}$, you don't need anything else to find to find next state s'
- Transition Matrix P defines probabilites for all successive states S'

3.2 Markov Chains

M = {S, T}

- Episodes are random sequences that are sampled.
- S = State Space
- T = Transition Probability or the probabililty of entering the next state

3.3 Markov Reward Process

M = {S, T, r}

- MRP is a tuple of (S is a finite set of states, P is a state of the transition probability matrix, Reward Function R, dicount factor γ)
- $R = E[R_{(t+1)} \mid S_t = s]$

$R_{(t+1)}$ is the amount of reward we approximate for the next state given state s

- We care about the cumulative reward

3.3.1 Return (goal)

Definition: total discounted reward from time-step t

- $G_t = R_{(t+1)} + \gamma * (R_{(t+1)}) + \ldots$
- Made finite by the γ
- γ is going to have to be [0,1]; 0 discounted factor means you only care about present Reward, 1 factor means you care about all of them
- Discount factor is used because we don't have a perfect model, avoids infinite returns, and animals show a preference for immediate reward

3.3.2 Bellman Equation

The Bellman Equation determines value of a state. It is comprised of immediate reward ($R_{(t+1)}$) and value of next state (γ*v($S_{(t+1)}$))

- Equation: $v(s) = E[G_t \mid S_t = s] = E[R_{(t+1)} + \gamma * v(S_{(t+1)}) \mid S_t = s]$

It is a linear quation and can be solved.

3.4 Markov Decison Process

M = {S, A, T, r}

- MDP is the same as MRP except with the addition of A (the action space)

3.4.1 partially observed MDP

M = {S, A , O , T, E, r}

- O observation space
- E emission probability

3.4.2 Policy

π(a|s) = P[A_t = a | S_t = s]

- A policy defines the behavior of an agent. It picks the actions that get the most reward.
-