Lecture 1 Introduction

Generalized Cylinder 广义圆柱体

Pictorial Structure 图结构模型

CS231 focuses on one of the most important problems of visual recognition — image classification

Year 2010 NEC-UIUC

Year 2011 Super Vision

Year 2012 Google Net VGG

Year 2015 MSRA

Data → Labeled : PASCAL

Fei-Fei Li

Lecture 2 Image Classification

All pixel change when the camera moves

Challenges: Deformation
Occlusion → robust
Background Clutter
Intra-class variation
...

data-driven approach

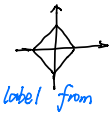
① Collect a dataset of images and labels

② use machine learning to train a classifier

③ Evaluate the classifier on new images

Distance Metric to compare images

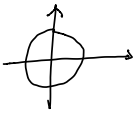
★ L1 distance $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$



★ k-Nearest neighbors : Instead of copying label from nearest neighbor, take majority vote from k closest points

problems : ① very slow at test time
② Distance metrics on pixels are not informative

★ L2 distance Euclidean distance $d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$



Hyper parameters : choices bout the algorithm that we set rather than learn 人为设定

accuracy and performance

train val test set → evaluate on test

↓

choose hyperparameters on val

cross-validation (split data into folds try each fold as validation and average the results.

↓

useful for small dataset but not frequently in deep learning

Linear Classification ✓

Image Example $32 \times 32 \times 3$ → $f(x, W)$ → 10 number giving class scores

W parameters of weights

$f(x, W) = Wx + b$

Lecture 3 Loss Function and optimization

To Do: ① Define a loss function that quantifies our unhappiness with the scores across the training data

② Come up with a way of efficiency finding the parameters that minimize the loss function. (optimization)

★ $\{(x_i, y_i)\}_{i=1}^N \Rightarrow L = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i)$

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

⇒ if we're looping over all of the incorrect classes, so we're looping over $C-1$ we will loss one.

↑ sum and mean have same effect

Loss function different choice. hing loss not wrong then increase.
square loss can enhance loss ② You should try different loss function.

$L = \frac{1}{N}\sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$ ⇒ find W for training data test. val

↓ update

$L(W) = \frac{1}{N}\sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$

Data loss: Model prediction should match training data

→ Regularization: Model should be "simple", so it works on test data.

overfitting

Occam's Razor: "Among competing hypotheses, the simplest is the best" — William of Ockham

Regularization: $L = \frac{1}{N}\sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$

L_2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L_1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net ($L_1 + L_2$) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Max norm regularization

Dropout

Fancier: Batch normalization, stochastic depth.

Softmax Classifier $P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$

Recap: $L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$ Softmax

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$ SVM

$L = \frac{1}{N}\sum_{i=1}^N L_i + R(W)$ Full loss

Optimization

follow the slope

SGD (Stochastic Gradient Descent)

$L(W) = \frac{1}{N}\sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$

$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$

Lecture 4 Backpropagation and Neural Networks

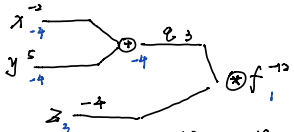
Gradient descent

$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$ Numerical gradient

Analytic gradient

Backpropagation: a simple example

$f(x, y, z) = (x+y) \cdot z$



$q = x + y \Rightarrow \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$

$f = qz \Rightarrow \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial z} = q$

$\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \quad \frac{\partial f}{\partial z}$

$f(x) = e^x \quad \frac{df}{dx} = e^x$

$f(x) = \frac{1}{x} \quad \frac{df}{dx} = -\frac{1}{x^2}$

$f_a(x) = ax \quad \frac{df}{dx} = a$

$f_c(x) = c + x \quad \frac{df}{dx} = 1$

local gradient × upstream gradient

$\sigma(x) = \frac{1}{1+e^{-x}} \quad \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)$

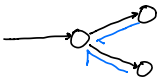
sigmoid function

$= (1 - \sigma(x)) \cdot \sigma(x)$

$\frac{\partial f}{\partial f} = 1 \quad \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial x} = 1$

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}$ chain rule

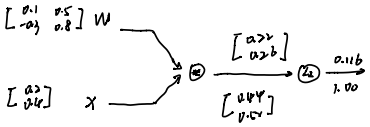
Gradients add at branches



they come back and influence it

$\frac{\partial f}{\partial x} = \sum_i \frac{\partial f}{\partial q_i} \frac{\partial q_i}{\partial x}$

Example: A vectorized example $f(x,W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



Neural Network

Linear score function $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$

$q = W \cdot x = \begin{pmatrix} W_{1,1} x_1 + \cdots + W_{1,n} x_n \\ \vdots \\ W_{n,1} x_1 + \cdots + W_{n,n} x_n \end{pmatrix}$

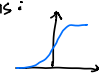
$f(q) = \|q\|^2 = q_1^2 + q_2^2 + \cdots + q_n^2$

$\frac{\partial q_k}{\partial W_{ij}} = 1_{k=i} x_j$

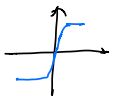
$\frac{\partial f}{\partial W_{ij}} = \sum_k \frac{\partial f}{\partial q_k} \cdot \frac{\partial q_k}{\partial W_{ij}} = \sum_k 2 q_k 1_{k=i} x_j = 2 q_i x_j$

Activation functions:

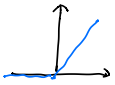
Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$



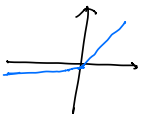
tanh $\tanh(x)$



ReLU $\max(0, x)$

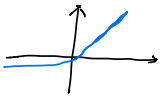


Leaky ReLU $\max(0.1x, x)$



Max out $\max(w_1^T x + b_1, w_2^T x + b_2)$

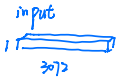
ELU $\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Lecture 5 Convolutional Neural Networks

$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$

Topographical mapping in the cortex



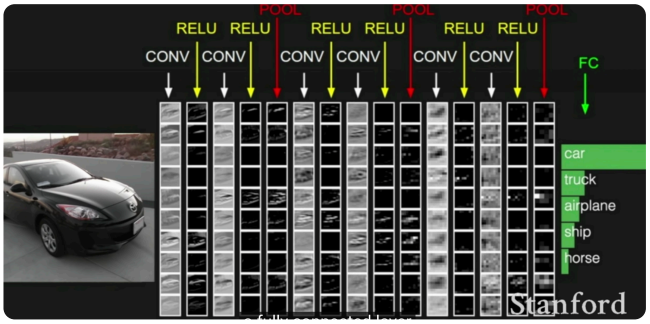
$\rightarrow$ Wx 10×30720 weights $\rightarrow$ activation 1×10

$32 \times 32 \times 3$ $5 \times 5 \times 3$ filter output

$\rightarrow$ $28 \times 28 \times 1$

input $\rightarrow$ low-level features $\rightarrow$ mid-level features

$\rightarrow$ high-level features $\rightarrow$ linearly separable classifier



stride one or two or three

input 7x7 · fliter 3x3

stride 1 ⇒ (7-3)/1+1 = 5

stride 2 ⇒ (7-3)/2+1 = 3

stride 3 ⇒ (7-3)/3+1 = 2.33

$(N-F)/stride+1$

Common to zero pad the border

zero padding has influence !!

32x32x3 ⟶ 10 5x5 filters with stride 1, pad 2

↓ output volumn size (32+2x2-5)/1+1 = 32 spatially, so 32x32x10

Conclusion: Accepts a volume of size $W_1 \times H_1 \times D_1$

Requires four hyperparameters $W_2 \times H_2 \times D_2$

$W_2 = (W_1 - F + 2P)/S + 1$

$H_2 = (H_1 - F + 2P)/S + 1$

$D_2 = K$

With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases

[x] convolution layers make perfect performance

in the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias

Pooling layer

- make the representations smaller and more manageable
- operates over each activation map independently

Max Pooling ✓

downsampling

Fully Connected Layer (FC layer)

Lecture 6 Training Neural Networks I

Mini-batch SGD

1. One time setup
2. Training dynamics
3. Evaluation

Part I 1.1 Activation Functions

Activation Functions

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

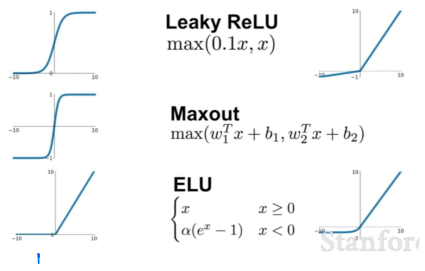
tanh $\tanh(x)$

ReLU $\max(0, x)$

Leaky ReLU $\max(0.1x, x)$

Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



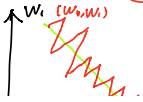
Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ when $|x|$ is large, the gradient is killed.

$f(\sum_i w_i x_i + b)$ 1. Saturated neurons "kill" the gradient

激活函数以0为中心的原因

权重更新方式是发生在反向更新阶段，过程：，对于神经元A来说，wi更新的方向和后几项都有关系，先看超参数!! 他是人为规定的，默认值，所以学习率可以不考虑；再看最后两位乘积项$\frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial x}$这个对于神经元A来说，它是神经元A的误差表示，在某一次反向传播也是不变的，也不考虑，所以w值的更新方向只与值有关，这个值是上一层神经元的输入值，即经过了sigmod函数激活过，所以肯定是正值，那么可以得出结论：在某一次反向传播时，对于神经元A来说，w1、w2..改变的方向是统一的，或正或负。如果你的最优值是需要w1增加，w2减少，那么只能走z字形才能实现。

$w_i \leftarrow w_i - \eta x_i \frac{\partial E}{\partial y} \frac{\partial y}{\partial x}$



2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

tanh
- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated

ReLU Computes $f(x) = \max(0, x)$ ($x>0$ region)
- Does not saturate (in + region)
- very computationally efficient
- Converges much faster than sigmoid/tanh in practice
- Actually more biologically plausible than sigmoid

dead ReLU will not activate, not update!

Leaky ReLU $f(x) = \max(0.01x, x)$

[$f(x) = \max(\alpha x, x)$ [

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice
- will not die

ELU (Exponential Linear Units)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

☆ Computation require exp()

1.2 Data preprocess

original data ⇒ zero-centered data ⇒ normalized data

`X -= np.mean(X, axis=0)` `X /= np.std(X, axis=0)`

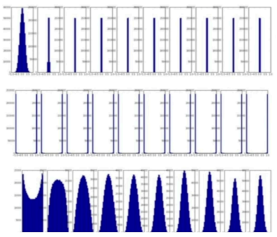
Subtract the mean image or channel ☆

1.3 Weight Initialization

idea 1 Small random numbers

`W = 0.01 * np.random.(D, H)`

⇒



Initialization too small: Activations go to zero, gradients also zero, No learning

Initialization too big: Activations saturate (for tanh), Gradients zero, no learning

Initialization just right: Nice distribution of activations at all layers, Learning proceeds nicely

Proper initialization is hard.

1.4 Batch Normalization

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$ this is a vanilla differentiable function

usually inserted after Fully Connected or Convolutional layers and before non linearity.

Batch Normalization

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta = BN_{\gamma,\beta}(x_i)$$

1.5 Babysitting the Learning Process

Double check the loss.

⇒ start with small training data

{ loss not going down — learning rate too slow
  loss exploding — learning rate too fast

1.6 Hyperparameter Optimization

Cross-validation strategy — First stage: only a few epoch
Second stage: long time

1.7 monitor and visualize the loss curve

training and validation accuracy → big gap: overfitting → increase regularization strength
→ no gap: increase model capacity

Lecture 7 Training Neural Networks

Fancier optimization
Regularization
Transfer Learning

saddle point → step small, slow progress

$x_{t+1} = x_t - \alpha \nabla f(x_t)$

SGD: stochastic gradient descent

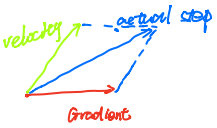
↓ come from mini batches so they can be noisy.

SGD + Momentum

$v_{t+1} = \rho v_t + \nabla f(x_t)$

$x_{t+1} = x_t - \alpha v_{t+1}$

↓ handle local minima and saddle points 梯度很小



Nesterov Momentum

$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$

$x_{t+1} = x_t + v_{t+1}$

$v_{t+1} = \rho v_t - \alpha f(\tilde{x}_t)$ $\quad \tilde{x}_t = x_t + \rho v_t$

$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1}$

$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$

AdaGrad (Added element-wise scaling of the gradient based on the historical sum of squares in each dimension) think past

RMSProp think past and now

Adam Momentum AdaGrad / RMSProp Bias correction

Adam is good on most case. but it is batch training.

Regularization: Add term to loss

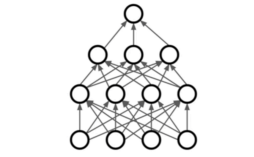
$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \lambda R(W)$$

In common use:

L2 regularization	$R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)
L1 regularization	$R(W) = \sum_k \sum_l \lvert W_{k,l}\rvert$
Elastic net (L1 + L2)	$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + \lvert W_{k,l}\rvert$

Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



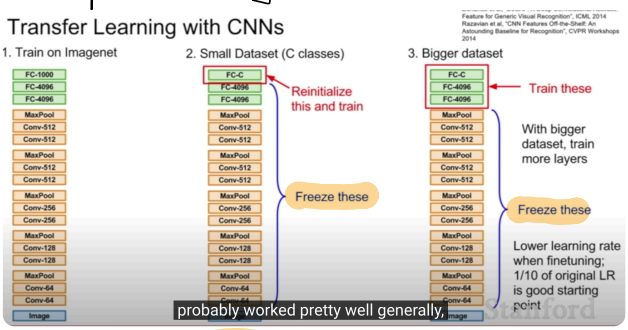
Stanford

Dropout is training a large ensemble of models. and make output random

Dropout , Batch Normalization Data Argumentation
DropConnect, Fractional Max pooling Stochastic Depth

Transfer Learning.

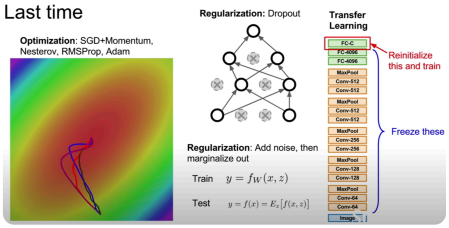
Transfer Learning with CNNs



more specific

more generic

Last time



Lecture 8 Deep Learning Software

CPU vs GPU Graphics Processing Unit

Caffe Pytorch Tensorflow

Pytorch : three level of Abstraction
Tensor, Varible, Module

Lecture 9 CNN Architectures

Case Study : AlexNet

Input : 227 x 227 x 3

First layer (CONV1): 96 11x11 filters applied at stride 4 ⇒ Hint (227-11)/4+1 = 55

↓ 11x11x96 parameters: 11x11x3x96 = 35K

Second layer pooling 3x3 stride2 ⇒ output 27x27x96

no parameters

★ first use of ReLU
use Norm layers
heavy Data augmentation
dropout 0.5
batch size 128

VGG use smaller filter

Stack of three 3x3 conv layers (stride 1)

layers has same effective receptive field as 7x7. $3\times(3^2C^2) < 7^2\cdot C^2$ ⇒

- use ensembles for best results
- Similar training procedure as Krizhevsky
- FCN features generalize well to other tasks.
- ...

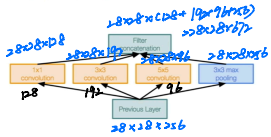
GoogleNet

- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters
- ...

Inception module

Case Study: GoogLeNet

[Szegedy et al., 2014]



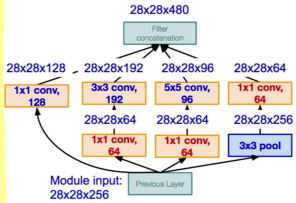
Naive Inception module

Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

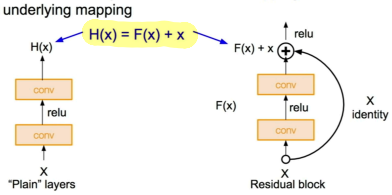
Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

Case Study: ResNet very deep by using residual connection

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

⇒

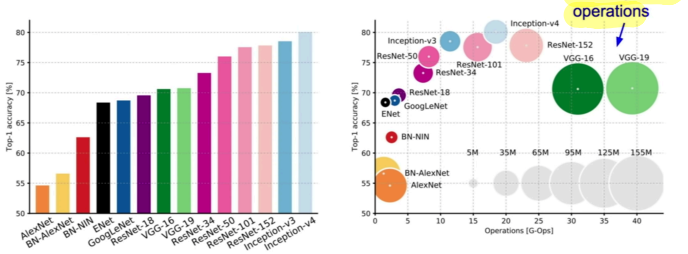
Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Comparing complexity

Comparing complexity...

VGG: Highest memory, most operations



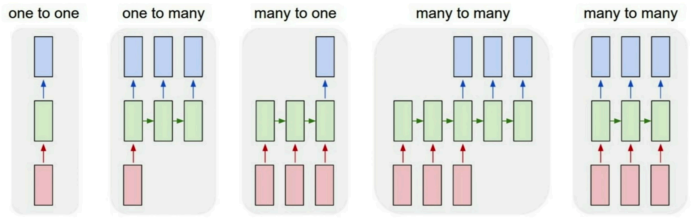
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Summary: CNN Architectures

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections

- Next time: Recurrent neural networks

Lecture 10 Recurrent Neural Network : Process Sequences

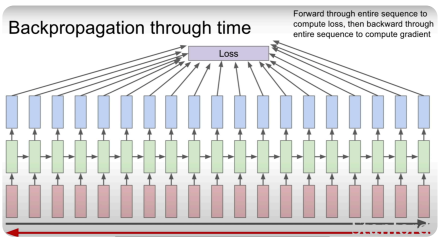


$h(t) = f_w(h_{t-1}, x_t)$

$h_t = f_w(h_{t-1}, x_t)$

$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$

$y_t = W_{hy} h_t$



Back propagation through time

Image Captioning with Attention.
RNN

Multilayer RNNs

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $\quad W^l \ [n \times 2n]$

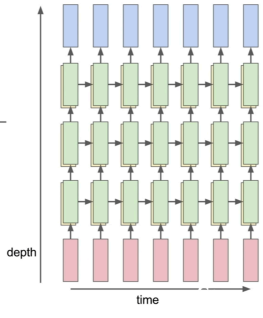
LSTM:

$W^l \ [4n \times 2n]$

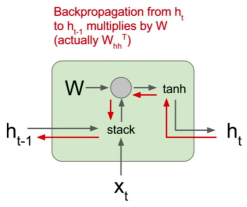
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$



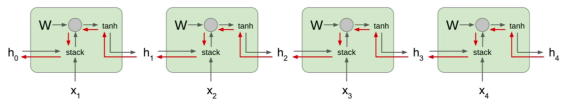
Vanilla RNN Gradient Flow



$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

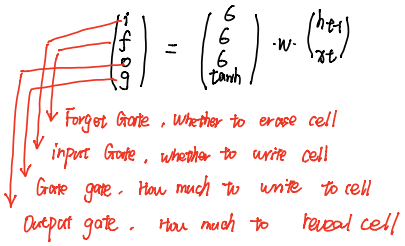
Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



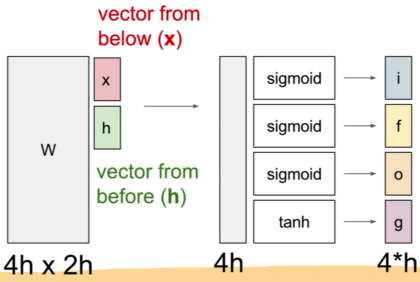
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Long Short Term Memory



$c_t = f \odot c_{t-1} + i \odot g$

$h_t = o \odot \tanh(c_t)$



Lecture 11 Detection and Segmentation

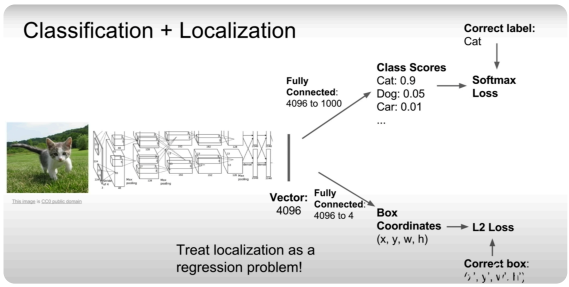
Unpooling 2x2 → 4x4

max pooling → max unpooling

Remember which element was max → Use positions from pooling layer

Transpose Convolution

strided convolution



object detection

RCNN

Fast RCNN → RoI

① RPN classify object / not object
② RPN regress box coordinates
③ Final classification score
④ Final box coordinates

Detection without Proposals: YOLO / SSD

with each grid cell

1. Regress from each of the B base boxes to a final box with 5 numbers (dx, dy, dh, dw, confidence)
2. Predict scores for each of C classes (including background as a class)

⇒ output $H \times W \times (5 \times B + C)$

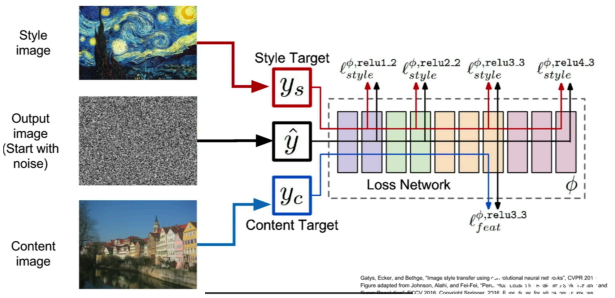
Object Detection: Lots of variables ...

Base Network	Object Detection architecture	Takeaways
VGG16	Faster R-CNN	Faster R-CNN is slower but more accurate
ResNet-101	R-FCN	
Inception V2	SSD	SSD is much faster but not as accurate
Inception V3		
Inception ResNet	**Image Size**	
MobileNet	**# Region Proposals**	
	...	

Lecture 12 Visualizing and Understanding

Visualizing CNN is an important task ⇒ Gradient Ascent

Neural Texture Synthesis



More weight to content or style loss

Style transfer very slow due to many forward / backward pass through VGG

Summary

Many methods for understanding CNN representations

Activations: Nearest neighbors, Dimensionality reduction, maximal patches, occlusion
Gradients: Saliency maps, class visualization, fooling images, feature inversion
Fun: DeepDream, Style Transfer.

Lecture 13 Generative Model

Supervised Learning

Data (x, y)

Goal: Learn a function to map x → y

Example: Classification, regression, object detection, Semantic segmentation, image captioning

Unsupervised Learning

Data: x

Goal: Learn some underlying hidden structure

Example: Clustering, dimensionality reduction, feature learning, density estimation

首先区分生成/判别方法和生成/判别模型。
有监督机器学习方法可以分为生成方法和判别方法（常见的生成方法有LDA主题模型、朴素贝叶斯算法和隐式马尔科夫模型等，常见的判别方法有SVM、LR等），生成方法学习出的是生成模型，判别方法学习出的是判别模型。

Generative Models

Given training data, generate new samples from same distribution


Training data ~ $p_{data}(x)$ Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Addresses density estimation, a core problem in unsupervised learning ★

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

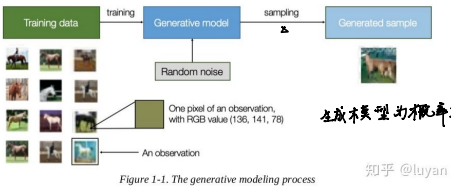
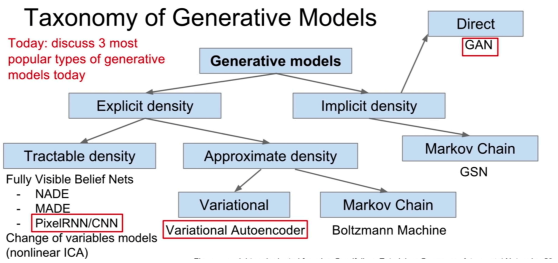


Figure 1-1. The generative modeling process

生成模型是概率模型，即非判别模型。

判别模型：P(y|x) 即给定观测 x 得到 y 的概率。
生成模型：P(x) 即观测 x 的现的概率



Component-by-component

Generative Model: PixelRNN, WaveNet, Video Pixel Networks

Fully visible belief network

$p(x) = \prod_{i=1}^{n} P(x_i | x_1, \ldots, x_{i-1})$

Likelihood of image x

probability of ith pixel value given all previous pixels

PixelRNN Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN

PixelCNN Generate image pixels starting from corner

Dependency on previous pixels modeled using an CNN over context region

Softmax loss at each pixel

PixelRNN and PixelCNN

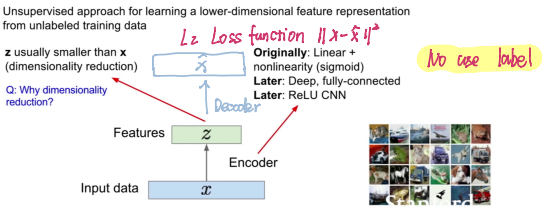
Pros:
- Can explicitly compute likelihood p(x)
- Explicit likelihood of training data gives good evaluation metric
- Good samples

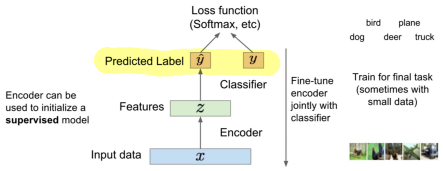
Con:
- Sequential generation => slow

Improving PixelCNN performance
- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc…

See
- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

Some background first: Autoencoders





Variational Autoencoders ☆

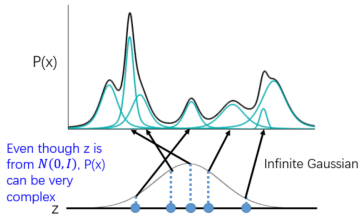
Data likelihood $P(x) = \int P(z) P(x|z) dz$

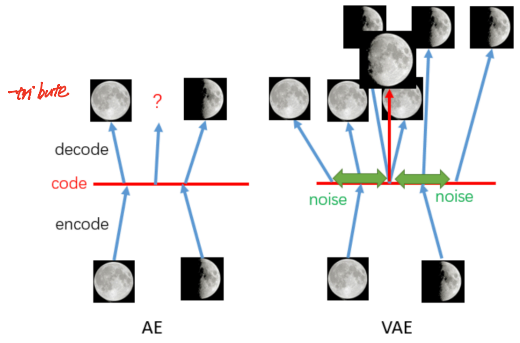
VAE是一个深度生成模型，是最终目的生成数据本身分布 $P(x)$，对输入数据。

VAE变分自动编码器作为AE的变体，它主要的变动是对编码(code)的生成上。编码(code)不再像AE中是唯一映射的，而是具有某种分布，使得编码(code)在某范围内波动时都可产生对应输出。借助下面这个例子进行理解：

VAE 通过高斯混合模型（Gaussian Mixture Model）来生成 $P(x)$，也就是说 $P(x)$ 由一系列高斯分布叠加而成，每一个高斯分布都有自己的参数 μ 和 σ

$\begin{cases} z \sim N(0, I) & z \text{ is a vector from normal distribution} \\ x|z \sim N(\mu(z), \sigma(z)) & \text{Each dimension of } z \text{ represents an attribute} \end{cases}$





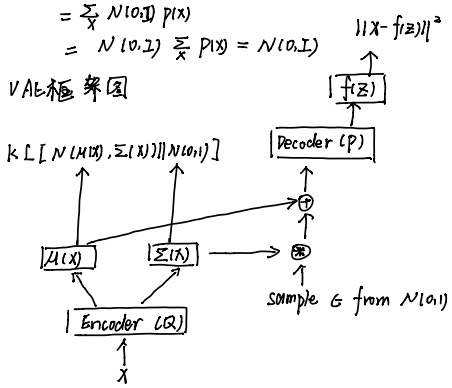
$X = \{x_1, x_2, \cdots, x_n\}$

$P(z) = \sum_x P(z|x) \cdot P(x)$

$= \sum_x N(0,1)\, P(x)$

$= N(0,1) \sum_x P(x) = N(0,I)$

VAE框架图

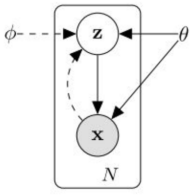


$L = \log(P(x)) = \sum_z q(z|x) \log(P(x))$

$= \sum_z q(z|x) \log\left(\frac{P(z,x)}{P(z|x)}\right)$

VAE是分布的映射关系 即 $D_X \to D_Z$

AE是值映射关系

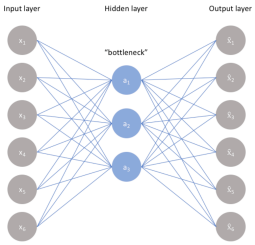


VAE的图模型。我们能观测到的数据是[公式]，而[公式]由隐变量[公式]产生，由[公式]是生成模型[公式]，从自编码器（auto-encoder）的角度来看，就是解码器；而由[公式]是识别模型（recognition model）[公式]，类似于自编码器的编码器。

$P_\theta(x|z)$

$q_\phi(z|x)$

Auto Encoder （异常检测. 数据去噪. 数据降维. 图像修复. 信息检索）



（1）压缩编码的数据维度一定要比元时输入数据更少。

（2）不管是编码器，还是解码器，本质上都是神经网络层，神经网络层一定要具有一定的“容量”。

最小化重构误差 $L(x, \hat{x})$

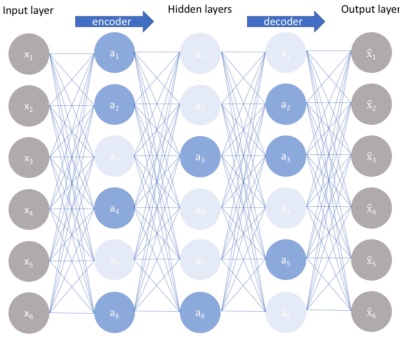
常见的 AE

① 欠完备的自编码器 — 普通一般自编码器

欠完备自编码器没有明确的正则化项，只是根据重构损失来训练我们的模型。因此，确保模型不记忆输入数据的唯一方法就是确保我们已经充分限制了隐藏层中的节点数量。

② 稀疏自编码器 — 隐藏层选择性的激活

稀疏自编码器为我们提供了一种不需要减少我们隐藏层的节点数量，就可以引入信息瓶颈的的方法。相反，我们将构造我们的损失函数，以惩罚层中的激活。对于任何给定的观察，我们都会鼓励我们的网络学习只依赖激活少量神经元进行编码和解码。值得注意的是，这是一种比较特殊的正则化实现方法，因为我们通常调整网络的权重，而不是激活。



Two approaches to enhance 稀疏性约束

L1正则化：我们可以添加一个对损失函数的正则化项，在h层中为观察i惩罚激活a的向量值的绝对值，使用微调参数λ进行缩放。 $L(x,\hat{x}) + \lambda \sum_i |a_i^{(h)}|$

KL -散度（相对熵）:本质上，KL散度是两个概率分布差异的度量。我们可以定义一个参数ρ稀疏，它表示一个神经元在样本集合上的平均激活。这种期望可以计算为 $\hat{\rho}_j = \frac{1}{m}\sum_i [a_i^{(h)}(x)]$

下标j表示表示层h中特定的神经元,对m个训练观察的表征x的激活求和。本质上，通过限制一个神经元在样本集合上的平均激活，我们鼓励神经元只对观测的一个子集进行激活。我们可以将ρ描述为一个伯努利随机变量分布，我们可以利用KL散度(下展开)来比较理想的分布在所有隐藏层节点上的观察分布。

$L(x,\hat{x}) + \sum_j KL(\rho \| \hat{\rho}_j)$

③ 降噪自编码器 → 给数据添加噪音

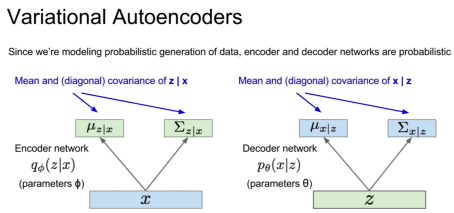
采用这种方法，我们的模型不能简单地开发一个记忆训练数据的映射，因为我们的输入和目标输出不再相同。更确切的说，该模型学习矢量场以将输入数据映射到较低维流形；如果这个流形精确地描述了自然数据，我们就有效地“消除”了多余的噪声。

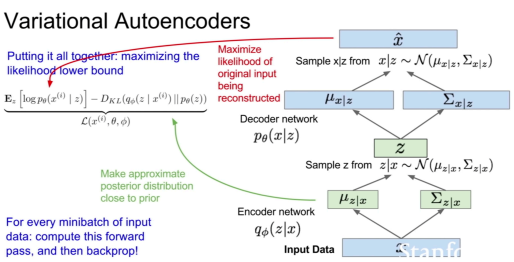
④ 压缩自编码器

人们会期望对于非常相似的输入，学习的编码也会非常相似。我们可以为此训练我们的模型，以便通过要求隐藏层激活的导数相对于输入而言很小。换句话说，对于输入比较小的改动，我们仍然应该保持一个非常类似的编码状态。这与降噪自编码器相似，因为输入的小扰动本质上被认为是噪声，并且我们希望我们的模型对噪声具有很强的鲁棒性。“降噪自编码器使重构函数(解码器)抵抗输入有限小的扰动，而压缩自编码器使特征提取函数(编码器)抵抗输入无限小的扰动。“

VAE: Probabilistic spin on autoencoders

Data likelihood: $p_\theta(x) = \int p_\theta(z)\, p_\theta(x|z)\, dz$





GAN: Generator Discriminator

$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_\theta(z)}[\log(1 - D(G(z)))]$

Variational Autoencoders

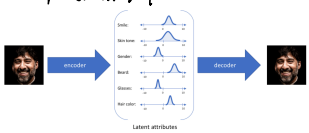
Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a (variational) lower bound

Pros:
- Principled approach to generative models
- Allows inference of q(z|x), can be useful feature representation for other tasks

Cons:
- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:
- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian
- Incorporating structure in latent variables

VAE: 当每个给定的隐属性都为概率分布，自从隐状态解码时，我们将从每个隐状态分布中随机采样，来生成向量作为解码器的输入。



Training GANs

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images.

整个式子由两项构成。x表示真实图片，z表示输入G网络的噪声，而G(z)表示G网络生成的图片。
D(x)表示D网络判断真实图片是否真实的概率（因为x就是真实的，所以对于D来说，这个值越接近1越好）。而D(G(z))是D网络判断G生成的图片的是否真实的概率。
G的目的：上面提到过，D(G(z))是D网络判断G生成的图片是否真实的概率，G应该希望自己生成的图片“越接近真实越好”。也就是说，G希望D(G(z))尽可能得大，这时V(D, G)会变小。因此我们看到式子的最前面的记号是min_G。
D的目的：D的能力越强，D(x)应该越大，D(G(x))应该越小。这时V(D,G)会变大。因此式子对于D来说是求最大(max_D)

Train jointly in minimax game

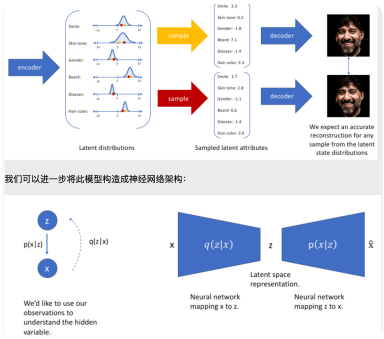
$$\min_G \max_D [E_{x\sim p_{data}} \log D_{\theta_d}(x) + E_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

Gradient on discriminator ↓

Gradient on generator ↑

Lecture 14 Reinforcement Learning

通过构造我们的编码器来输出一系列可能的值（统计分布），然后随机采样该值作为解码器的输入，我们能够学习到一个连续，平滑的隐空间。因此，在隐空间中彼此相邻的值应该与非常类似的重建相对应。而从隐分布中采样到的任何样本，我们都希望解码器理解，并准确重构出来。



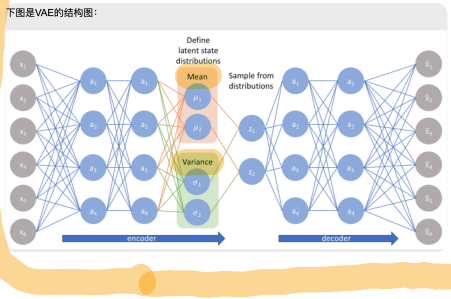
Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

下图是VAE的结构图：



Markov Decision Process

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(\,.\,|\,s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\,.\,|\,s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

- A policy π is a function from S to A that specifies what action to take in each state
- **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t\geq 0} \gamma^t r_t$

The optimal policy π^*

We want to find optimal policy π^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)? Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

马尔科夫性 Markov Property
某一状态信息包含了所有相关的历史，只要当前状态可知，所有的历史信息都不再需要，当前状态就可以决定未来，则认为该状态具有马尔科夫性。

可以用下面的状态转移概率公式来描述马尔科夫性：

$P_{ss'} = P[S_{t+1} = s' \mid S_t = s]$

状态转移矩阵

$$P = \text{from} \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \text{ to}$$

Markov Chain $\langle S, P \rangle$

Markov Reward Process $\langle S, P, R, \gamma \rangle$ 衰减系数

奖励系数

Definitions: Value function and Q-value function

$V^\pi(s) = E\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$ (How good is a state)

$Q^\pi(s,a) = E\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$ (How good is a state-action pair)

$R_s = E[R_{t+1} \mid S_t = s]$

Discount Factor 衰减系数

$\gamma \in [0, 1]$

Reinforce algorithm

$J(\theta) = E_{\tau\sim p(\tau;\theta)}[r(\tau)] = \int_\tau r(\tau)\, p(\tau;\theta)\, d\tau$

Q-value

policy → Action

Q-function → Critic

Algorithm

定义：收获 [公式] 为在一个马尔科夫奖励链上从t时刻开始往后所有的奖励的有衰减的总和。也有翻译成“收益”或"回报"。公式如下：

Gt

Return

$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ
For iteration=1, 2 … **do**
  Sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  For i=1, …, m **do**
    For t=1, … , T **do**
      $A_t = \sum_{t'>t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$
      $\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i|s_t^i)$
  $\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$
  $\theta \leftarrow \alpha\Delta\theta$
  $\phi \leftarrow \beta\Delta\phi$
End for

- **Policy gradients**: very general but suffer from high variance so requires a lot of samples. **Challenge**: sample-efficiency
- **Q-learning**: does not always work but when it works, usually more sample-efficient. **Challenge**: exploration

- Guarantees:
  - **Policy Gradients**: Converges to a local minima of J(θ), often good enough!
  - **Q-learning**: Zero guarantees since you are approximating Bellman equation with a complicated function approximator

价值函数给出了某一状态或某一行为的长期价值。

$\gamma \to 0$ 近期评估
$\gamma \to 1$ 远期考虑

定义：一个马尔科夫奖励过程中某一状态的价值函数为从该状态开始的马尔可夫链收获的期望

Value Function 价值函数

$v(s) = E[G_t | S_t = s]$ ★

收获是针对一个马尔科夫链中的某一个状态来说的。

Bellman Function – MRP

$$\begin{aligned} v(s) &= E[G_t | S_t = s] \\ &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] \\ &= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s] \\ &= E[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \end{aligned}$$

$v(s)$

通过方程可以看出 [公式] 由两部分组成，一是该状态的即时奖励期望，即时奖励期望等于即时奖励，因为根据即时奖励的定义，它与下一个状态无关；另一个是下一时刻状态的价值期望，可以根据下一时刻状态的概率分布得到其期望。如果用s'表示s状态下一时刻任一可能的状态，那么Bellman方程可以写成：$v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$

$V = R + \gamma P V$

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

$V = R + \gamma P V$
$[1 - \gamma P] V = R$
$V = (1 - \gamma P)^{-1} R$

Markov Decision Process
行为集合 A $\langle S, A, P, R, \gamma \rangle$
$P_{ss'}^a = P[S_{t+1} = s' | S_t = s, A_t = a]$
$R_s^a = E[R_{t+1} | S_t = s, A_t = a]$

策略 Policy π 是概率集成或分布，$\pi(a|s)$ 为对过程中某一状态采取可能行为 a 的概率
$\pi(a|s) = P[A_t = a | S_t = s]$

$M = \langle S, A, P, R, \gamma \rangle$ $S_1, S_2, \ldots$

当给定一个MDP: [公式] 和一个策略π，那么状态序列 [公式] 是一个马尔科夫过程 [公式]；同样，状态和奖励序列 [公式] 是一个马尔科夫奖励过程 [公式]，并且在这个奖励过程中满足下面两个方程：

$$P_{s,s'}^{\pi} = \sum_{a \in A} \pi(a|s) P_{ss'}^a$$

用文字描述是这样的，在执行策略 π 时，状态从s转移至 s' 的概率等于一系列概率的和，这一系列概率指的是在执行当前策略时，执行某一个行为的概率与该行为能使状态从s转移至s'的概率的乘积。

奖励函数表示如下：

$$R_s^{\pi} = \sum_{a \in A} \pi(a|s) R_s^a$$

用文字表述是这样的：当前状态s下执行某一指定策略得到的即时奖励是该策略下所有可能行为得到的奖励与该行为发生的概率的乘积的和。

基于策略 π 的价值策略
$V_\pi(s) = E_\pi[G_t | S_t = s]$
行为价值函数 $q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a]$

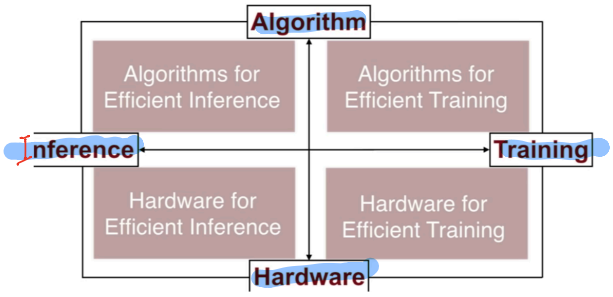
Lecture 15 Efficient Methods and hardware for Deep Learning

Google TPU

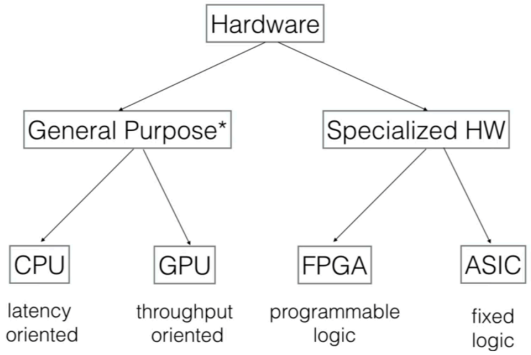
parallelize

hyper parameter parallel

Agenda



Hardware 101: the Family



Part I Algorithm for Efficient Inference

① Pruning (剪枝)

pruning and retraining

② weight sharing

③ Quantization (量化)

Cluster the weights

↓

Generate Code Book

↓

Quantize the weights with code book

↓

Retrain Code Book

Lecture 16 Adversarial Examples on Adversial Training

Adversarial Examples

$\tilde{J}(\tilde{x}, \theta) \approx J(x, \theta) + (\tilde{x} - x)^T \nabla_x J(x)$

Noise. ☆ sigmoid

Human Brain

Map of Adversial Cross section

Attack easy . defense difficult

0231. END.

2020.07.04

李文科