DCNN Q&A

Kessler, Somer, Itzhak

July 24, 2019

Unanswered Questions:

When to use StandardScaling vs Normalization?

SOLVE HW3?

Instance Normalization - don't we lose lots of information from normalzing each channel - "the fire channel"

Week 1

What are the inclusion relations between: AI, deep learning, machine learning and represntation learning?

Deep learning $\subset$ represntation learning $\subset$ machine learning $\subset$AI

What are the main problems in image classification?

Semantic gap (the computer sees only numbers in a grid), viewpoint variation, illumination, interclass variation, background clutter, occlusion (when an object is partly hidden)

Define nearest neighbor refering to training and predition.

Train - memorize all data and labels.
Predict - according to the most similar training image.

Define L1 distance for images.

$d_1(I_1, I_2) = \Sigma_P |I_1^P - I_2^P|$
P is pixel

Define L2 distance for images.

$d_2(I_1, I_2) = \sqrt{\Sigma_P (I_1^P - I_2^P)^2}$
P is pixel

With n examples what is the complexity of training and prediction?

Train $O(1)$
Predict $O(N)$

Why is this bad?

We want fast prediction, don't mind slow training.

Define K-nearest neighbor

Same as nearest neighbor only with majority vote from the K-nearestes images.

What are hyperparameters?

Choices about the algorithm that we set and not learned.

Why use validation dataset?

To choose hyperparameters (train for learning, test for testing).

What is cross-validation?

Split train data into folds, try each fold as validation (rest as train) and average the result.

Why K-nearest neighbor is never used on images?

1. Slow in test
2. Distance matrics on pixels are not informative

Week 2

What does it mean to assume i.d.d on a dataset?

Identically distributed - all of the samples are taken from same distrubtion D.
Independently distributed - $x_i \sim$ D is independent of $x_j \sim$ D

Why do we need to assume i.d.d on a dataset?

Identically distributed - to know that all of the samples are from the same domain.

Independently distributed - It is unrealistic to say that the second image in the training set depends on the tenth image of the training set.

What is the generalization error of learned function f, target function y on distribution D?

$L_D[f, y] = \mathbb{E}_{x \sim D}[loss(f(x), y(x))]$
with 1-0 loss:
$= \mathbb{E}_{x \sim D}[Indicator(f(x) \neq y(x))]$

Calculate:

What is the generalization error of *f*?

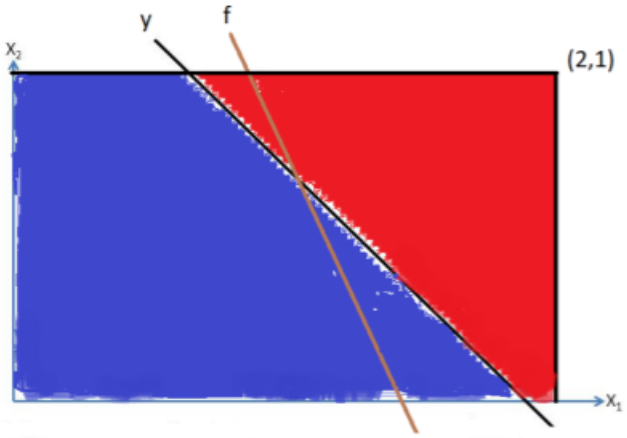


Figure: Assume *D* is a uniform distribution over the rectangle that is caged within the points $(0,0), (2,0), (0,1), (2,1)$. What is the generalization error of *f*?

slide 35 "Lecture 2 - Intro to ML"

How does machine learning differs from optimization?

In ML we want the generalization error to be low as well as the training error, treating ML problem as an optimization problem would lead to overfitting.

How can we control whether an algorithm is more likely to overfit or underfit?

By altering its capacity - it's ability to fit a wide variety of functions.

Algorithms with low capacity may struggle to fit the training set. Algorithms with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.

How can we control an algorithm capacity?

One way is to choose it's hypothesis space.

Week 3

Write linear regressor hypothesis class

$\mathcal{H} = \{f_w(x) = \langle w, x \rangle \,|w \in \mathbb{R}^n\}$

Write the solution for linear regression

$$\begin{aligned} f(w) &= \|y - Xw\|^2 \\ f'(w) &= \left(-X^T\right) \cdot 2 \cdot (y - Xw) \\ f'(w) &= 2 \cdot \left(X^T X w - X^T y\right) \\ 0 &= 2\left(X^T X w - X^T y\right) \\ X^T X w &= X^T y \\ w &= (X^T X)^{-1} X^T y \end{aligned}$$

$w = (X^T \cdot X)^{-1} \cdot X^T \cdot y$

Name one clustering algorithm

1. k-means

What is k-means objective function

$X =$ n points
$P =$ Partition of X into k sets
$P = \{1, ..., k\}^n$
$\arg\min_P \left(\sum_{i=1}^{k} \sum_{x \in P_i} \|x - \mu_i\|^2 \right)$

Give a huristic that sove k-means

Lloyd's algorithm:

1. Generate $\mu = \{\mu_1 ... \mu_k\}$
2. Repeat:
   (a) Assign each point in X to the closest point in μ
   (b) Update each μ_i to be the real mean of its current set
   (c) If μ did not change, stop

Always converges (The distances are only getting smaller)

Write the hypophesis class of SVM:

$\mathcal{H} = \{f_w(x) = sign(\langle w, x \rangle + b) \,|w \in \mathbb{R}^n, b \in \mathbb{R}\}$

What is a solution to SVM?

A hyper plane with max-margin or biggest separation

Write down SVM objective function:

$$\max_{w,b:\forall i:y_i(\langle w,x\rangle+b)\geq 0} \min_i \left(\left|\frac{\langle w,x\rangle+b}{\|w\|_2}\right|\right) =$$
$$\max_{w,b:\forall i:y_i(\langle w,x\rangle+b)\geq 1} \min_i \left(\left|\frac{1}{\|w\|_2}\right|\right) =$$
$$\max_{w,b:\forall i:y_i(\langle w,x\rangle+b)\geq 1} \frac{1}{\|w\|_2} \sim$$
$$\min_{w,b:\forall i:y_i(\langle w,x\rangle+b)\geq 1} \frac{1}{2}\|w\|_2^2$$

In cases where the data is not linear-speratable, how can we still use SVM?

Use kernels. Instead of trying to classify X, classify $\Phi(X)$, where $\Phi : \mathbb{R}^n \to \mathbb{R}^m$

Give 1 problem with choosing a general high dimention kernel (RBF)

1. Choosing general kernel might lead to good training loss, but poor generalization

Give 3 problem with choosing a specified kernel

1. Not automatic - require sometimes decades
2. Not generic - require tailor made kernel per dataset
3. Not transferable - different domains require different features

Week 5

Define convex function

$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$

Write down Jensen's inequality for f which is convex

$\forall x_1 ... x_T : f\left(\frac{1}{T}\sum_{i=1}^{T} x_i\right) \leq \frac{1}{T}\sum_{i=1}^{T} f(x_i)$

Write taylor first order approximation for function $g : \mathbb{R}^n \to \mathbb{R}$

$g(u) \approx g(w) + \langle u - w, \nabla g(w)\rangle$

Gradient Desent algorithm:

1. Parameter: μ - learning rate, T - iterations
2. Init: $w_0 = 0$
3. For $t = 1...T$:
   (a) $w_t = w_{t-1} - \mu \nabla g\left(w_{t-1}\right)$

What would be the output of gradient desent algorithm when optimizing convex and non-convex problems?

1. convex $\frac{1}{T}\sum_{i=1}^{T} w_i$
2. non-convex$\underset{w_i}{argming}\left(w_i\right)$

What is the optimization objective for each gradient decent iteration?

"Close with good improvement"
$w_t = \arg\min_{w}\left(\frac{1}{2}\left\|w - w_{t-1}\right\|_2^2 + \mu\left(g\left(w_{t-1}\right) + \left\langle w - w_{t-1}, \nabla g\left(w_{t-1}\right)\right\rangle\right)\right)$
The minimum is in $w_t = w_{t-1} - \mu \nabla g\left(w_{t-1}\right)$

Assuming that g is a convex-Lipschitz function, and let $\|w^*\| \leq B$, prove an upper bound on $g(\overline{w}) - g\left(w^*\right)$

First set $\mu = \sqrt{\frac{B^2}{\rho^2 T}}$, and run $T = \frac{B^2\rho^2}{\epsilon^2}$

$$\begin{aligned} g(\overline{w}) - g\left(w^*\right) &= g\left(\frac{1}{T}\sum_{t=0}^{T} w_t\right) - g\left(w^*\right) \\ &\leq \frac{1}{T}\sum_{t=0}^{T}\left(g\left(w_t\right)\right) - g\left(w^*\right) \\ &= \frac{1}{T}\sum_{t=0}^{T}\left(g\left(w_t\right) - g\left(w^*\right)\right) \end{aligned}$$

for every t: $g(\overline{w}) - g\left(w^*\right) \leq \left\langle w_t - w^*, \nabla g\left(w_t\right)\right\rangle$, as such from the previous part
$g(\overline{w}) - g\left(w^*\right) \leq \frac{1}{T}\sum_{t=0}^{T}\left\langle w_t - w^*, \nabla g\left(w_t\right)\right\rangle$

$$\begin{aligned}\langle w_t - w^*, \nabla g\left(w_t\right)\rangle &= \frac{1}{\mu}\left\langle w_t - w^*, \mu \nabla g\left(w_t\right)\right\rangle \\ &= \frac{1}{2\mu}\left(-\left\|w_t - w^* - \mu \nabla g\left(w_t\right)\right\|^2 + \left\|w_t - w^*\right\|^2 + \mu^2\left\|\nabla g\left(w_t\right)\right\|\right) \\ &= \frac{1}{2\mu}\left(-\left\|w_{t+1} - w^*\right\|^2 + \left\|w_t - w^*\right\|^2 + \mu^2\left\|\nabla g\left(w_t\right)\right\|\right)\end{aligned}$$

Replacing the values, we get:

$$\begin{aligned}g(\overline{w}) - g\left(w^*\right) &\leq \frac{1}{T}\sum_{t=0}^{T}\left\langle w_t - w^*, \nabla g\left(w_t\right)\right\rangle \\ &= \frac{1}{T}\sum_{t=0}^{T}\frac{1}{2\mu}\left(-\left\|w_{t+1} - w^*\right\|^2 + \left\|w_t - w^*\right\|^2 + \mu^2\left\|\nabla g\left(w_t\right)\right\|\right) \\ &= \frac{1}{2\mu T}\left(\left\|w_0 - w^*\right\|^2 - \left\|w_{T+1} - w^*\right\|^2\right) + \frac{1}{2\mu T}\sum_{t=0}^{T}\left(\mu^2\left\|\nabla g\left(w_t\right)\right\|\right) \\ &\leq \frac{1}{2\mu T}\left(\left\|w_0 - w^*\right\|^2\right) + \frac{1}{2\mu T}\sum_{t=0}^{T}\left(\mu^2\left\|\nabla g\left(w_t\right)\right\|\right)\end{aligned}$$

We know that $w_0 = 0$, so:

$$\begin{aligned}g(\overline{w}) - g\left(w^*\right) &\leq \frac{1}{2\mu T}\left(\left\|w_0 - w^*\right\|^2\right) + \frac{1}{2\mu T}\sum_{t=0}^{T}\left(\mu^2\left\|\nabla g\left(w_t\right)\right\|\right) \\ &= \frac{\left\|w^*\right\|^2}{2\mu T} + \frac{\mu}{2T}\sum_{t=0}^{T}\left(\left\|\nabla g\left(w_t\right)\right\|\right)\end{aligned}$$

$$
\begin{aligned}
\frac{\|w^*\|^2}{2\mu T} + \frac{\mu}{2T}\sum_{t=0}^{T}\left(\|\nabla g\left(w_t\right)\|\right) &\leq \frac{B^2}{2\mu T} + \frac{\mu}{2T}\sum_{t=0}^{T}\rho^2 \\
&= \frac{B^2}{2\mu T} + \frac{\mu\rho^2}{2} \\
&= \frac{B^2}{2\sqrt{\frac{B^2}{\rho^2 T}}T} + \frac{\sqrt{\frac{B^2}{\rho^2 T}}\rho^2}{2} \\
\frac{B^2\rho^2}{\epsilon^2} &= \frac{B\rho\sqrt{T}}{2T} + \frac{B\rho^2}{2\rho\sqrt{T}} \\
&= \frac{B\rho}{2\sqrt{T}} + \frac{B\rho}{2\sqrt{T}} = \frac{B\rho}{\sqrt{T}} \\
&= \frac{B\rho}{\frac{B\rho}{\epsilon}} = \epsilon
\end{aligned}
$$

What is the problem with Gradient Desent?

It's computatinaly expensive to compute the gradient over all X

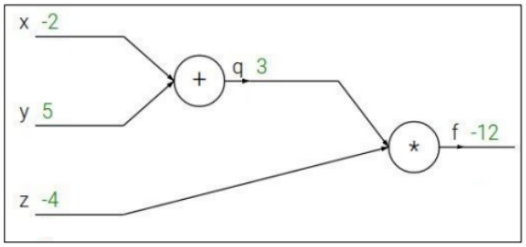
What is Stochastic Gradient Desent algorithm?

Like Gradient Desent algorithm, but instead of computing the gradient explicitly, we instead estimates it as $\nabla f\left(X\right) = \mathop{\mathbb{E}}_{x\in X}\left[\nabla f\left(x\right)\right]$

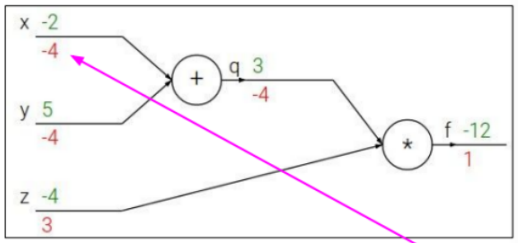
Write down the chain rule:

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$

Fill the derivative reletive to f of the following image:

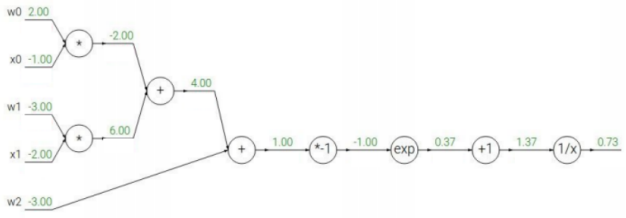


==========

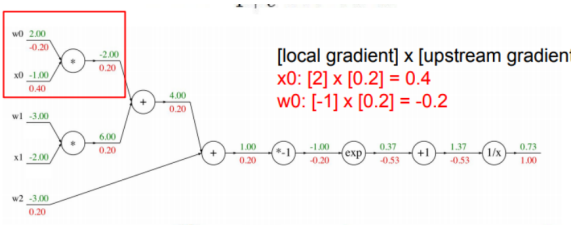


Fill the gradient reletiv to f:

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



==========

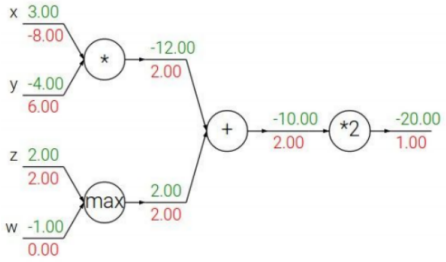


$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$ | $f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$ | $f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Explain add mul and max gates as to what they do to the gradient

1. max - routes
2. add - distributes
3. mul - switchs



Given X batch of size $N_{batch} \times D$ and w layer weights of size $D \times M$, and $y = f(x, w) = xw$, what would be the size of $\frac{\partial f}{\partial x}$?

y size is $N_{batch} \times M$, as such $\frac{\partial f}{\partial x}$ size is $(N_{batch} \times M) \times (N_{batch} \times D)$

Assuming that you have $\frac{\partial L}{\partial y}$ and you wish to compute $\frac{\partial L}{\partial x}$ without holding in the memory the whole $\frac{\partial y}{\partial x}$

you can just compute $\frac{\partial L}{\partial x_{(i,j)}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x_{(i,j)}}$

Week 6a - Neural Networks in Practice

Write down the multiclass SVM loss:

$L(f(x_i), y_i) = \sum_{j \neq y_i} \max\left(0, \left(f(x_i)_j + 1\right) - f(x_i)_{y_i}\right)$

In the multiclass SVM loss, what is the minimum achievable loss?

0

In the multiclass SVM loss, what is the maximum achievable loss?

$+\infty$

What is the initial loss, when W is set to small values, $s \approx 0$?

$c - 1$

What is the elastic net regularization method?

weighted sum of L_1 and L_2.

Develop the softmax classifier loss (multinomial logistic regression?):

1. Our neural network outputs the final activations, called - logits, unnormalized log-probabilities:

$$s$$

2. We raise them to the exponent so we have positive values

$$e^s$$

3. We normalize the values in order to get a distribution:

$$\hat{P}_i = P\left(Y = k | X = x_i\right) = \frac{e^{s_k}}{\sum_{j=1}^{k} e^{s_j}}$$

4. Our loss is the KL divergence between this distribution and the "indicator distribution for class k":

$$P_i = P\left(Y = c | X = x_i\right) = \begin{cases} 1 & c = k \\ 0 & \text{else} \end{cases}$$

The KL divergence is:

$$\begin{aligned} KL\left(\hat{P}_i \| P_i\right) &= \sum_{c=1}^{C} P_i\left(c\right) \log\left(\frac{P_i\left(c\right)}{\hat{P}_i\left(c\right)}\right) \\ &= P_i\left(y_i\right) \log\left(\frac{P_i\left(y_i\right)}{\hat{P}_i\left(y_i\right)}\right) \\ &= 1 \cdot \log\left(\frac{1}{\hat{P}_i\left(y_i\right)}\right) \\ &= -\log\left(\hat{P}_i\left(y_i\right)\right) \\ &= -\log\left(\frac{e^{s_{y_i}}}{\sum_{j=1}^{k} e^{s_j}}\right) \end{aligned}$$

What is the minimal loss?

0

What is the maximal loss?

∞

At initialization all scores are approximately equal, what is the loss?

$-\log\left(\frac{e^{s_{y_i}}}{\sum_{j=1}^{k} e^{s_j}}\right) = -\log\left(\frac{1}{C}\right) = \log\left(C\right)$

What is a "gradient check"?

The process of validation of an analytical gradient implementation.
The analytical method is fast and exact but implementation is error prone so we can test it using the numerical gradient which is very straightforward but slow and approximate.

State and describe 3 Old-School methods for image feature extraction:

1. Color histogram
2. Histogram of Oriented Gradients - Divide the image into 8×8 sized blocks - for each pixel, compute the gradient direction and magnitude. We then create a histogram of 9 "buckets", corresponding to 9 angles. We then add the relative amount of each gradient to each bucket. 50 will be split half and half to the 40,60 buckets.
   We concatenate these histograms to get the description.
3. Bag of words - Take a dataset of images, extract patches from them. Cluster the patches. Encode images by computing a histogram of counts by assigning each patch to a cluster.

Write down a brief history of convolutional neural networks, including dates, researchers etc.

the study of the human brain is thousands of years old. With the advent of modern electronics, it was only natural to try to harness this thinking process. The first step toward artificial neural networks came in 1943 when Warren McCulloch, a neurophysiologist, and a young mathematician, Walter Pitts, wrote a paper on how neurons might work. They modeled a simple neural network with electrical circuits. Reinforcing this concept of neurons and how they work was a book written by Donald Hebb. The Organization of Behavior was written in 1949. It pointed out that neural pathways are strengthened each time that they are used.

As computers advanced into their infancy of the 1950s, it became possible to begin to model the rudiments of these theories concerning human thought. Nathanial Rochester from the IBM research laboratories led the first effort to simulate a neural network. That first attempt failed. But later attempts were successful. It was during this time that traditional computing began to flower and, as it did, the emphasis in computing left the neural research in the background.

Yet, throughout this time, advocates of "thinking machines" continued to argue their cases. In 1956 the Dartmouth Summer Research Project on Artificial Intelligence provided a boost to both artificial intelligence and neural networks. One of the outcomes of this process was to stimulate research in both the intelligent side, AI, as it is known throughout the industry, and in the much lower level neural processing part of the brain.

In the years following the Dartmouth Project, John von Neumann suggested imitating simple neuron functions by using telegraph relays or vacuum tubes. Also, Frank Rosenblatt, a neuro-biologist of Cornell, began work on the Perceptron. He was intrigued with the operation of the eye of a fly. Much of the processing which tells a fly to flee is done in its eye. The Perceptron, which resulted from this research, was built in hardware and is the oldest neural network still in use today. A single-layer perceptron was found to be useful in classifying a continuous-valued set of inputs into one of two classes. The perceptron computes a weighted sum of the inputs, subtracts a threshold, and passes one of two possible values out as the result. Unfortunately, the perceptron is limited and was proven as such during the "disillusioned years" in Marvin Minsky and Seymour Papert's 1969 book Perceptrons.

Just kidding Ido and Itay.

We have a $32 \times 32 \times 3$ sized picture, what can we know for sure about the dimensions of any filter in the next layer?

$k \times k \times 3$ - filters always extend the full depth of the input volume

We have a $32 \times 32 \times 3$ sized picture, what is the number of paramaters of a $k \times k$ filter?

5*5*3 + 1 (bias)

We have a 7×7 image. Apply a 3×3 filter, stride 1. What is the size of the output?

$((7-3)/1) + 1 = 5 \rightarrow 5 \times 5$

We have a 7×7 image. Apply a 3×3 filter, stride 2. What is the size of the output?

$((7-3)/2) + 1 = 3 \rightarrow 3 \times 3$

What is a common way to end up with the same size image?

3×3 conv with zero padding of 1, with stride 1.

$$(N + 2 - 3)/1 + 1 = N - 1 + 1 = N$$

We have a $32 \times 32 \times 3$ image, 10 5×5 filters, stride 1 pad 2. What is the total output size? $(32 + 2 \cdot 2 - 5)/1 + 1 = 32$
$32 \times 32 \times 10$

What does a 1×1 convolution do?

A simple dot product over 1 pixel, through all channels.

We have a $32 \times 32 \times 3$ image followed by a Fully connected Layer. What is the receptive field of a neuron in that layer? The entire image.

Write down the typical CNN architecture structure Regex:

$[((CONV - RELU) * N - POOL) * M - (FC - RELU) * K, SOFTMAX]$

Week 7

Define the Sigmoid Function, state its value range:

$\frac{1}{1+e^{-x}}$, value from [0,1]

State the 3 problems of the Sigmoid function:

1. At high value of input $|x|$ the gradient approaches 0.
2. Outputs only positive values (leads to the "synchronized w gradients" zig zag issue)
3. e^x is compute expensive

Take the derivative of $\sigma(x)$:

$\sigma(x)\,(1 - \sigma(x))$

Define the tanh Function, state its value range:

$\frac{e^x - e^{-x}}{e^x + e^{-x}}$, value from [-1,1]

State an 1 advantage and 2 disadvantages of Tanh:

Advantages:

1. $(-1, 1)$ range is zero centered.

Disadvantages:

1. At high value of input $|x|$ the gradient approaches 0.
2. e^x is compute expensive.

Define the ReLU Function

$max(0, x)$, negative are 0, and otherwise x.

State 4 advantages of Relu:

1. Doesn't saturate in + region
2. Computationally efficient
3. Converges faster than sigmoid and tanh
4. More biologically plausible than sigmoid and tanh

State 2 dis-advantages of Relu:

1. Not zero centerd
2. "Dead Relu" (zero gradient for negative input)

State a solution and best practice for the Dead ReLu Problem:

Initialize ReLu units with slightly positive Bias (0.01)

Define the Leaky ReLU Function

$max(0.01x, x)$.

State 4 advantages of Leaky Relu:

1. Doesn't saturate at all
2. Computationally efficient
3. Converges faster than sigmoid and tanh
4. Does not die (the dead ReLU problem)

Define the Paramatric ReLU Function (PReLU)

$max(\alpha x, x)$ where $\alpha \in [0, 1)$.

Define the Maxout Function

$max(w_1 \cdot x + b_1, w_2 \cdot x + b_2)$, just check 2 different models

Define the ELU Function

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

What are the 5 advantages of ELU?

1. Doesn't saturate in + region
2. Converges faster than sigmoid and tanh
3. More biologically plausible than sigmoid and tanh
4. Closer to zero mean (compared with ReLU)
5. Some robustness to noise due to negative value gradient saturation.

What are the dis-advantages of ELU?

Computationally expensive

State 2 advantages of Maxout:

1. Generalizes ReLU/Leaky ReLU
2. Linear! No death.

State a disadvantage of Maxout:

Double number of paramaters.

Best practice for activation functions:

ReLU (low learning rate)-> Leaky/ELU/Maxout -> Tanh (never sigmoid)

Data PreProcessing

What is the typical method for preprocessing data?

Subtract column means, divide by column std.

Which 2 methods for preprocessing images are commonly used?

Center only (now division by STD):

1. Remove mean image (per pixel mean $H \times W \times C$)
2. Remove mean per channel (per channel mean C)

Weight Initialization:

What is the problem with constant weight initialization?

All nodes treated symetrically.

What is the formula for $VAR(X \cdot Y)$ for independent X, Y:

$\sigma_X^2\sigma_Y^2 + \sigma_X^2\mu_Y^2 + \sigma_Y^2\mu_X^2$

10 layers, 500 neurons per each, tanh activation. What is the problem with initializing all layers with a 0.01 standard deviation, 0 mean normal distribution? (describe the corresponding experiment)

All layer mean activations go to 0, all layers std go to 0. The reason is that we arrive at a 0 mean distribution with very low STD.

10 layers, 500 neurons per each, tanh activation. What is the problem with initializing all layers with a 1 standard deviation, 0 mean normal distribution? (describe the corresponding experiment)

All neurons are activated at $\{-1, 1\}$ (saturated) due to large variance in the input (500ish).

Describe the "Xavier Initialization" method:

$W = np.randn(\text{fan\_in}, \text{fan\_out}) \cdot \frac{1}{\sqrt{\text{fan\_in}}}$, justification - $\sum_{i=1}^{\text{fan in}} w_i x_i \sim N(0, 1)$

Why does this fail with ReLU, and in what way?

We divide by $\sqrt{\text{fan in}}$,but due to ReLU half of the weights are 0. So we should divide by $\frac{\sqrt{\text{fan in}}}{2}$

How is the above initialization called?

He

Batch Normalization

Describe the Batch Normalization method, before which layers do we usually place a BN layers?:

During training: Before the activation function, usually after FC or Convolutional layer:

1. Compute batch mean (at layer) μ_B
2. Compute batch std (at layer) σ_B
3. Per sample activation x_i compute a normalized version of the sample's activation: $\hat{x_i} = \frac{x_i - \mu_B}{\sigma + \epsilon}$
4. Scale: $\hat{x_i} \cdot \gamma + \beta \equiv BN_{\gamma,\beta}(x_i)$

4 Advantages of BN?

1. The gradient flow improve (because of the gradient's scale)
2. Allow higher learning rates (because of the gradient's scale)
3. Reduce dependency on initialization
4. Regularization: each batch is normalized reletive to itself, harder to overfit

Describe BN at test time:

Instead of normalizing to the batch, normalize reletive to an estimated mean and varance of the input data (which is computed during the training time)

Babysitting the learning process:

Describe the generic processes of creating and training a neural net:

1. Pre-process data
2. Choose architecture
3. See reasonable loss + loss increase after adding regularization
4. Tweak learning rate:
   (a) No improvement = too small
   (b) Explodes = too big

Hyperparamater Tuning:

What is the general principle for searching for optimal paramaters?

Start with a coarse search (find correct order of magnitude), then a more fine search.

Name two methods for hyperparamter searching

Grid Search, Random Search

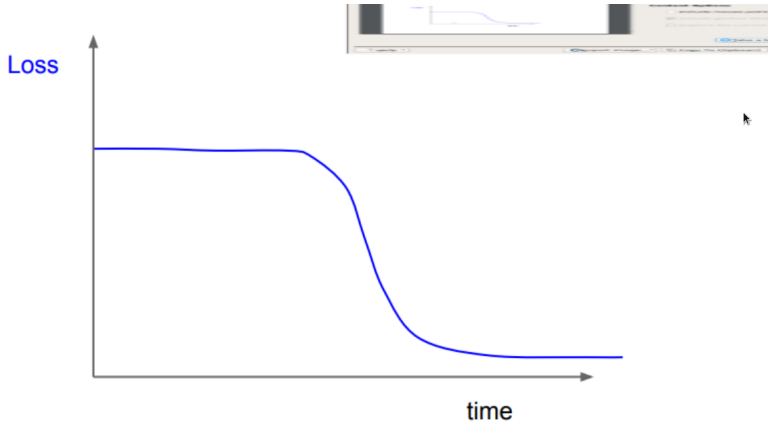
In what setting is the Random Search particularly effective?

One important paramater, one non-important. Instead of checking a small number of values of the important paramater against all values of the non-important, we get a random spread of values of the important one.

Give examples of hyperparamaters:

1. Architecture
2. Learning rate + update algos etc.
3. Regularization

What does the following image indicate?



Bad initialization

You see a large gap between validation and train error, what you do?

Overfitting! Increase regularization.

You see no gap between validation and train error, what you do?

Increase model capacity?

What will we never think of, as a method for validating correct learning rate?

See that weight updates are around 0.01 of the weight magnitude. (norm of current weights, norm of entire update vector)

Week 8a

How do we batch normalize images?

Per channel.

What is Layer Normalization?

Normalize each example independently using all of its features:

$$\gamma, \beta \in 1 \times D$$
$$\mu, \sigma \in 1 \times N$$
$$y = \gamma(x - \mu)/\sigma + \beta$$

Same in train and test.

When is Layer Normalization needed, why?

RNNs. Learning constant γ, β can lead to bad behavior in RNNs (explode/implode).

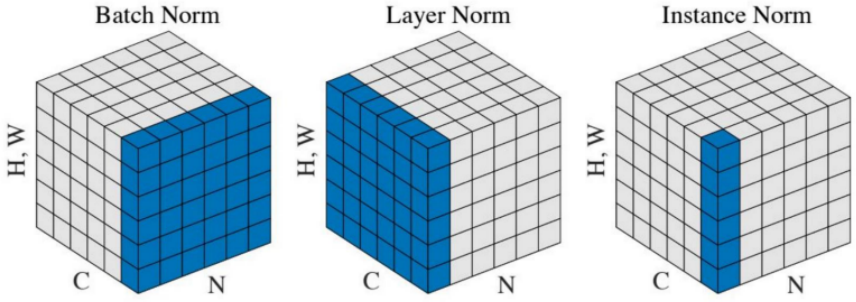
What is Instance Normalization?

"Layer Normalization" for images. Normalize each example independently, per channel using all pixels of that channel:

$$\gamma, \beta \in 1 \times C$$
$$\mu, \sigma \in N \times C$$
$$y = \gamma(x - \mu)/\sigma + \beta$$

Same in train and test.

Draw the 3 cubes of Normalization for images:



What is group norm?

Same as instance norm except we use a group of channels (arbitrarily chosen?)

What is DBN?

Decorrelated Batch Norm. Take the decorrelated x, remove mean, divide each feature by it's standard deviation. Now we have "whitened data"

$$\Sigma^{-\frac{1}{2}} (x_i - \mu)$$

State 3 problems of Vanilla SGD?

1. There can be large changes in one direction relative to others so learning rate does not fit all directions equally well.
2. Presence of local minima or saddle points
3. Gradients are noisy due to use of mini-batches.

What happens in SGD if in one direction we have a large gradient and in another a small gradient?

Jumps in large direction, slow in small direction.

What is the condition number of a hessian matrix? <LEARN MORE ABOUT THIS?>

Ratio of largest to smallest singular value.

Write down the vanilla sgd update rule of the weights?

$W_{t+1} = W_t - \alpha \nabla f(W_t)$

State a SGD update rule which enables the algorithm to overcome saddle points and local minima (zero gradients)?

Momentum!

$$\begin{aligned} v_0 &= 0 \\ v_{t+1} &= \rho v_t + \nabla f(W_t) \\ W_{t+1} &= W_t - \alpha v_{t+1} \end{aligned}$$

Give an alternative formulation to the momentum formula:

$$\begin{aligned} v_0 &= 0 \\ v_{t+1} &= \rho v_t - \alpha \nabla f(W_t) \\ W_{t+1} &= W_t + v_{t+1} \end{aligned}$$

In the Momentum method, what is the problem with adding the gradient at W_t?

We are moving, the gradient at W_t might not be relevant if we will move far from it due to v_t.

What is the solution to the previous issue (develops second formulation of Momentum formula)?

Nesterov momentum.

$$\begin{aligned} v_0 &= 0 \\ v_{t+1} &= \rho v_t - \alpha \nabla f(W_t + \rho v_t) \\ W_{t+1} &= W_t + v_{t+1} \end{aligned}$$

What is a common re-paramatization of the Nesterov rule?

Denote:

$$\tilde{W}_t = W_t + \rho v_t$$

The update rule becomes:

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

How do we overcome the issue of different sized gradients in different directions?

AdaGrad! We sum the square of the gradients at each step, and divide each gradient by the square root of this sum. (the norm of the vector of all gradients seen thus far for this direction)
Algorithm:

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

What happens for directions with large gradient?

Smaller steps.

What happens for directions with small gradient?

Large steps initially.

What happens to the step size over long time?

Goes to zero.

How do you prevent this from happening?

RMSprop - apply a decay to the sum of square gradients from AdaGrad (typically 0.9).

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Why is this decay computation not exactly a weighted average?

Initial gradient is 0!

How is the combination of RMSprop and Momentum called?

Adam!

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Explain the unbiasing term:

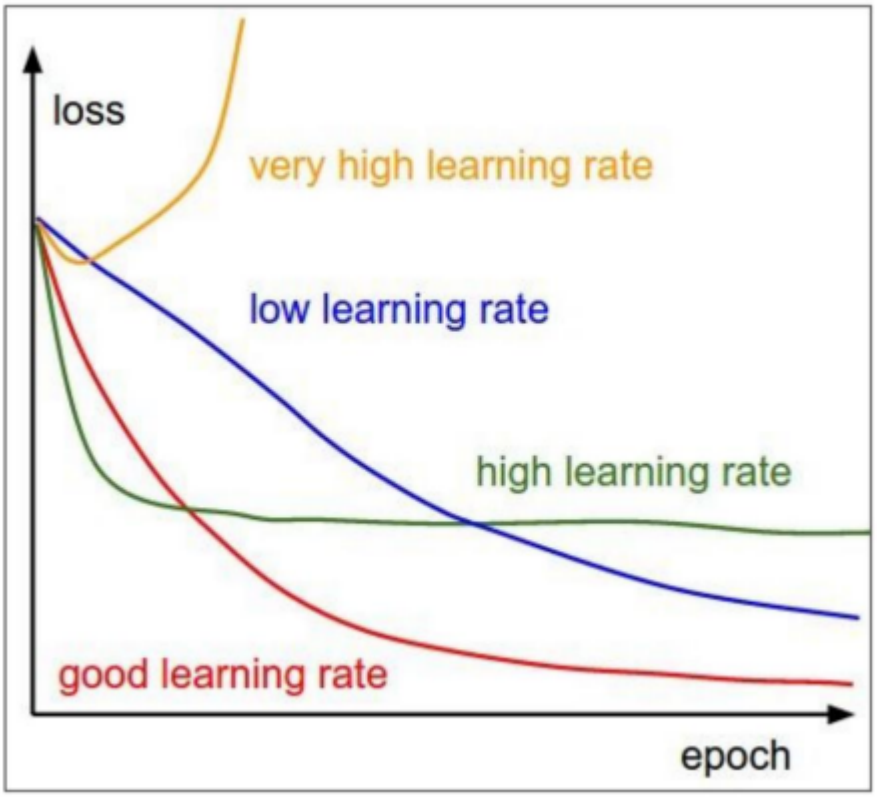
After unfolding we have the following weighted average (assuming first moment is initialized to 0):

$$\hat{m}_t = \frac{\beta^{t-1} g_1 + \beta^{t-2} g_2 + ...g_t}{\beta^{t-1} + \beta^{t-2} + ... + 1}$$

What hyperparamater do all of these methods have?

Learning Rate

Draw a graph with 4 cases for constant learning rate:



State 3 Methods for learning rate variation:

1. Step: divide by half every n epochs.
2. Exponential decay: $\alpha_t = \alpha_0 e^{-kt}$
3. 1/t decay: $\alpha_t = \frac{\alpha_0}{(1+kt)}$

Why is learning rate decay less common with Adam than with SGD + Momentum:

Adam has the learning rate adjusted per paramater according to size of the gradient.

What is the difference in the step taken using second order optimization?

In first order we take a small step in the direction of decrease. In second order optimization we take a guess at the minimum directly.

Write down the second order multi-variate taylor series:

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

$\frac{\partial x^T A x}{\partial x}$**=?**

$\left(A + A^T\right) x$

What is the critical point, according to the above expansion:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= \nabla J(\theta_0) + \mathbf{H}(\theta - \theta_0) = 0 \\ \mathbf{H}(\theta - \theta_0) &= -\nabla J(\theta_0) \\ \theta &= \theta_0 - \mathbf{H}^{-1} \nabla J(\theta_0) \end{aligned}$$

How many hyperparamaters does a second order optimization scheme have?

None!

Why can't we use this in practice?

Complexity of computing $\mathbf{H}^{-1}$ is $O\left(n^3\right)$.

What is the name of an algorithm for Solving this in $O\left(n^2\right)$ time?

BFGS or L-BFGS

In what setting is L-BFGS useful?

Full batch, deterministic (consistent $f(x)$ to be optimized)

How do we know how many epochs to use?

Early stopping - stop when validation error starts increasing, or train for a lot of epochs and keep a snapshot of best performing model.

What are model ensembles?

A method for reducing variance via averaging of many models.

How can we use model ensembles in the Deep Learning setting without training many different models from scratch?

Use different snapshots of the same model.

A specific model seems to just improve its performance over time. How can we still use snapshots while having relatively varying models?

Cyclic Learning rate. Such as cosine annealing - start with $\alpha = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min})\left(1 + \cos\left(\pi \frac{T \mod N + 1}{N}\right)\right)$

How can we implement snapshot model ensembles without holding many paramater vectors?

Polyak Averaging. Compute a running average of the paramaters:

$$W_{\text{test}} = 0.995 W_{\text{test}} + 0.005 W_t$$

State 7 methods for Regularization?

1. Add weight decay term to loss: $L_2, L_1,$ or a weighted mix of both (elastic net)
2. Dropout
3. Randomness - add randomness in training, average out in testing (see next questions)
4. Data Augmentation
5. DropConnect
6. Fractional Max Pooling
7. Stochastic Depth

Explain Dropout:

In the forward pass set each neuron to zero with probability p.

State two interpretations to why Dropout could be useful:

1. Force redundancy of features and prevent co-adaptation
2. Train an ensemble of models, each with slightly different architecture, all share some paramaters.

What is the issue with using models trained with Droput in the test phase?

Droupout makes output random:

$$y = f_W(\underbrace{x}_{\text{input image}}, \underbrace{z}_{\text{random mask}})$$

What is the output we would like to approximate?

$$\mathbb{E}_z\left[f_W(x,z)\right] = \int_z p\left(z\right) f_W(x,z) dz$$

This is the randomness method "template from before "

Why can't we simply activate all neurons in test time?

If we have trained with only p of the neurons in each layer activated, we will have a larger expected value in test time.

What are 2 possible solutions for this?

In both solutions we activate all neurons in test time. The solutions differ in the time in which we scale the outputs. Assuming a neuron is kept with probability p.

1. In test time, scale all activations of each layer by p.
2. In training time, divide the activations of each layer by p. (testing becomes a simple feedforward)

How is the second method called?

Inverted Dropout.

State _ methods for data augmentation:

1. Flips
2. Random crops and scales
3. Color jittering - randomize contrast and brightness
4. Distortions - lenses, stretch, translation, shearing etc.

How does transfer learning differ when using large/small datasets (both relatively similar to the original one)?

Small - finetune only last layer. Large - maybe finetune a few layers

How does transfer learning differ when the target dataset is very different from the original one. Explain for small/large amount of data?

Lots of data - finetune lots of layers. Little data - try linear classifiers from different layers, probably can't finetune many layers on a huge net using little data.

Week 8b - CNN Architectures

Given image of input $227 \times 227 \times 3$, and Conv2d layer with 96 filters of size11×11 with stride 4, what would be the output size:

The width and height is $(227 - 11)/4 + 1 = 55$, so $55 \times 55 \times 96$

Given image of input $227 \times 227 \times 3$, and Conv2d layer with 96 filters of size11×11 with stride 4, how many parameters are there:

$11 \cdot 11 \cdot 3$ for every filter, so $11 \cdot 11 \cdot 3 \cdot 96 = 34848$

Given input $55 \times 55 \times 96$, what will be the output of a $Max\,Pool$ layer with 3×3 filter size and stride 2:

The width and height of the new input will be: $(55-3)/2+1 = 27$, so $27 \times 27 \times 96$

Given image of input $13 \times 13 \times 256$, and Conv2d layer with 96 filters of size 3×3 with stride 1, and 1 padding what would be the output size:

$((13 + 2 \cdot 1) - 3)/1 + 1 = 13$, $13 \times 13 \times 96$

How is ZFNet different from AlexNet?

ZFNet use the same architecture of AlexNet, but keep higher resolution at the beginning(smaller stride at start), and use more channels at the deeper layers.

What are VGGNet building blocks?

VGGNet is builded with only 3×3 filter convolution layers, 2×2 max-pool stide 2 layers, and at the end fully connected layers

What were the 2 main changes that VGG net presented reletive to AlexNet and ZFNet?

1. Very small fillter right from the start that does not change the images sizse ($(3, 3)$, stride 1, padding 1)

2. More layers (deeper net)

What are the benefits of smaller fillter in VGGNet, for example a 3×3 filter reletive to 7×7 filter?

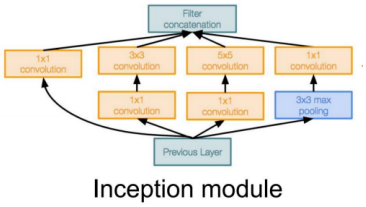
1. **Deeper:** You can stack 3 Conv layer with filter 3×3 of each other to get the same receptive field of a 7×7 filter. This allow deeper layers = more non-linearity.
   (Each stack of 3×3 filter enlarge the receptive field of the neuron by 1 to both sides, so after 3 stacks you get $3 \times 3 \to 5 \times 5 \to 7 \times 7$)

2. **less parameters**
   (Each 3×3 uses $3 \cdot 3 \cdot C$ parameters, so $3 \cdot \left(3^2 \cdot C\right) = 27 \cdot C$, on the other hand $7 \cdot 7 \cdot C = 49 \cdot C$)

What is the "Network within a network" feature of the inception model?

Design a good network topology and stack those onto each others.

What is the structure of the inception model block?



Inception module

What are 1×1 Conv filter used for?

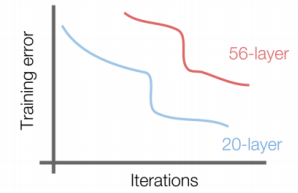
To reduce the number of channels without much parameters and without losing resolution

What do you add additinal gradient at lower layers?

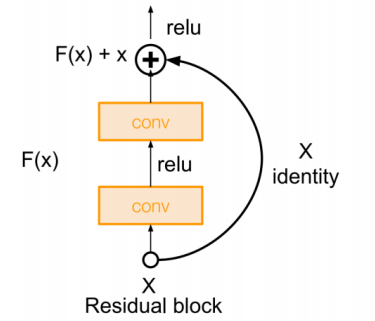
You can add auxilary outputs that try to classify the images at earlier stages.

When stacking convolutional layers onto each other, what happend to the losses as the depth get larger?

You get higher training and test error →You don't succeed in over-fitting, even thougth the model is more complex



Describe the ResNet blocks:



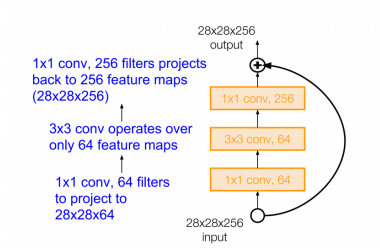
Essentialy $F(X) + X$

What does the ResNet block try to solve?

When building deeper network the input get distorted after many layers, and the gradient vanish when going all the way back to the first layers. The addition of the input at each block keep the data from distorting too much, and allow the gradient to flow more easily back to the first layers.

State 1 technique that allow you to reduce computation cost in ResNet:

Use "bottleneck" layers:

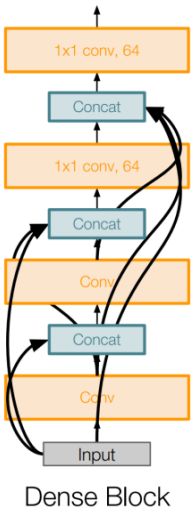


This avoid doing filters when both the input number of chanels and the output number of chanels is high, instead it reduce the dimention of the data first to 64 and then increase it back to 256

State 4 ways to improve ResNet?

1. Wide ResNet (More channels)
2. Argregated ResNet (each block contain mutiple version that are agragated at the end)
3. Regularization: Stochastic Depth
4. Squeeze-and-Excitation networks (SENet) - Before adding back the X, learn how much to "scale" each channel

How does a Dense Block (from dense net) look like?



Dense Block

Week 9 - Recurrent Neural Nets

What is the Recurrent Neural Net function:

$$
\begin{aligned}
h_t &= \text{State at time } t \\
x_t &= \text{Input for time } t \\
F_W(h, x) &= \text{Function with parameters } W \text{ that recive state and input} \\
h_t &= F_W(h_{t-1}, x_t)
\end{aligned}
$$

Name 3 RNN structures 'types':

1. Many to One
2. Many to Many
3. One to Many - "generate sequence from single input"

Give an example of "Many to One" RNN

Sequence of words -> Sentiment

Give an example of "One to Many" RNN

Image -> Sequence of words

Give an example of "Many to Many" RNN

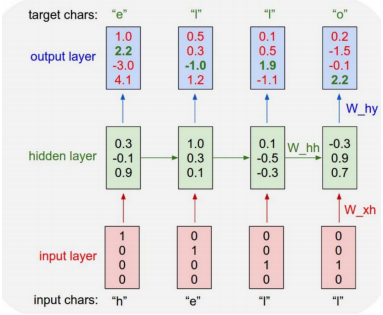
Video -> Classification on frame level

Write down te equation for Vanilla RNN

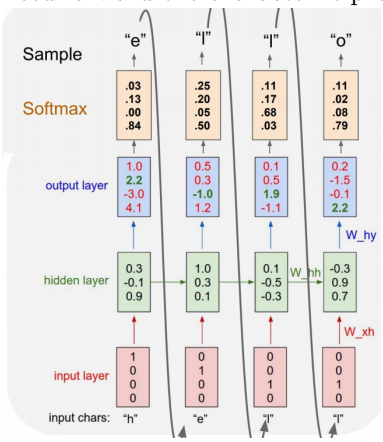
$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
$y_t = W_{hy}h_t$

How do you train a character level RNN on some language model, and output same random text from it?

Train the model such that at every step it should predict the next character

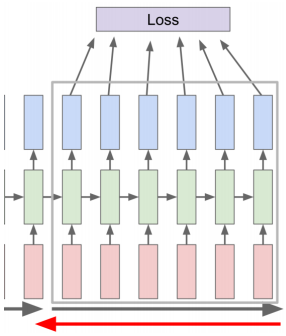


Feed forward the character it predict to generate:



What 2 ways are there to backpropogate RNN:

1. **Backpropogate through time:** Unfold the layer repetition as different layers, backpropogate throuth all of them
2. **Truncated backpropogation through time:** Unfold the layer repetition in chunks and back propogate on only those chunks



What are 2 problems that Truncated backpropogate help solve?

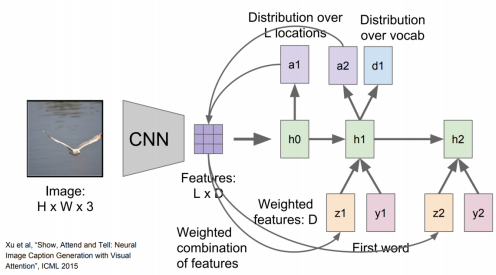
1. Single gradient computation require many interation unfolding
2. Gradient vanish after many iterations

How would you build a neural network that generate text descripion of the image?

Encode image into a latent vector (multiple conv and pooling)
Import the vector as a hidden state layer to RNN and generate words from it

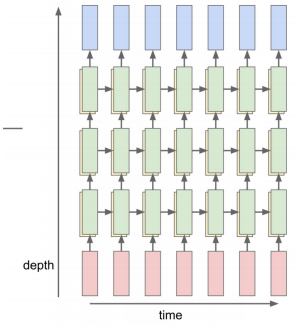
Describe how would you add an attention model to the previous example

Encode the image into latent matrix of size $v = Locations \times Features$.
At each RNN step output weights vector $a = Location \times 1$ which represent weight per location and feed the RNN with single feature vector $z = \sum_{i=1}^{Locations} a_i v_i$



What is multilayer RNN?

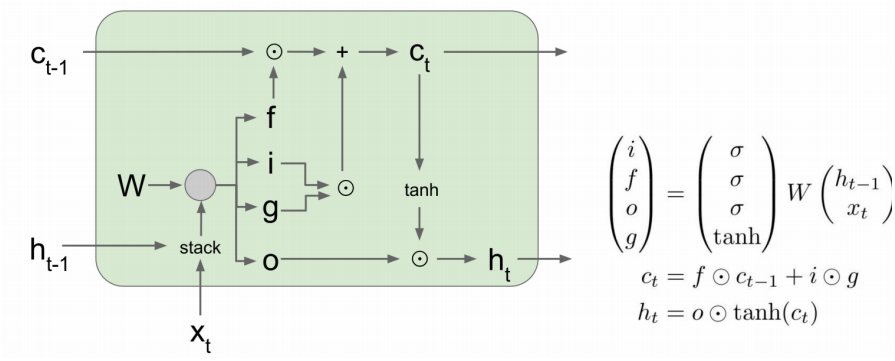
running the input of each timestamp of a RNN as a sequence that goes through another RNN



Give one way to handle vanilla RNN gradient explosion problem:

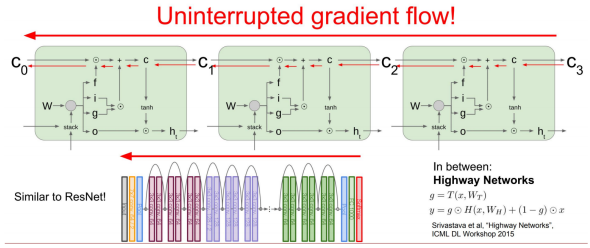
gradient clipping

What is the structure of LSTM?



What is the problem LSTM network solves and how does it do that?

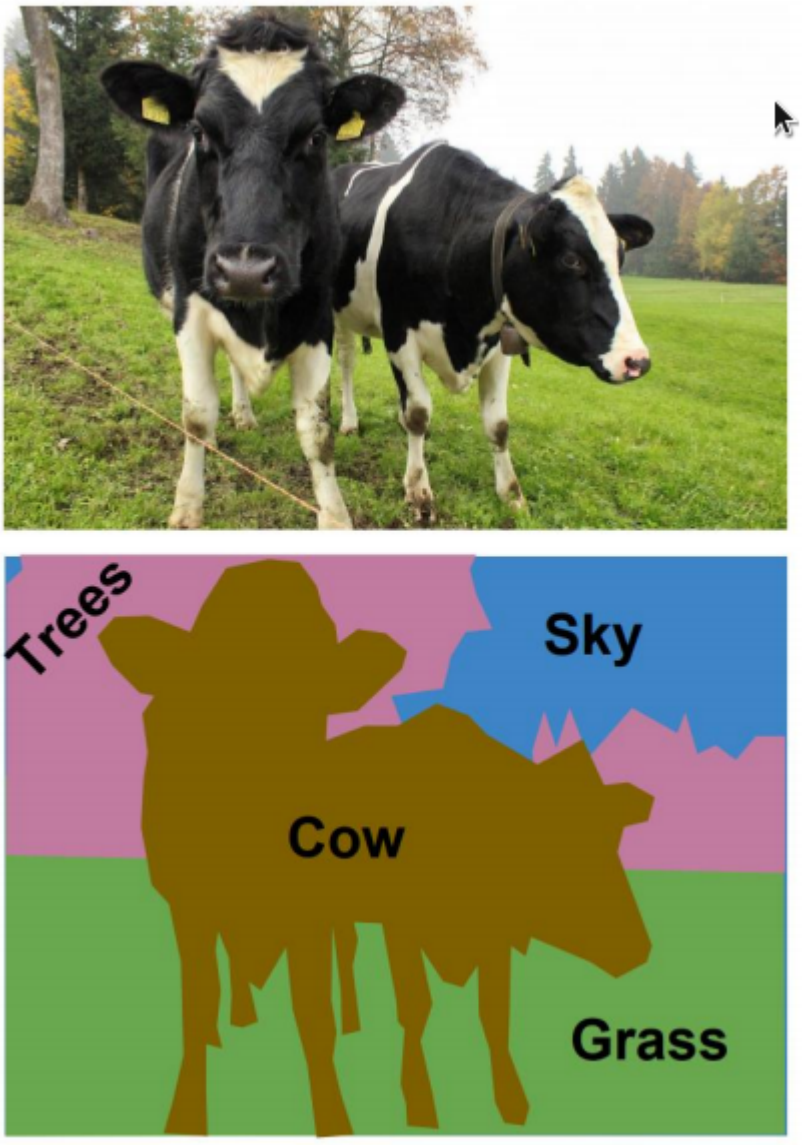
Vanilla RNN gradients "vanish", because of the repetative activation function and matrix multipication and finite-precision numbers.
LSTM fix this issue by creating an easier way for the gradient to flow through uninterrupted, which allow the gradient an easier way not to “vanish”.



Week 11a - Detection and Segmentation

Define Semantic Segmentation:

Tag each pixel with the type of element inside it. No differentiation between objects of the same type:



State 2 techniques for semantic segmentation:

1. Sliding Window
2. Fully Convolutional

What is the problem with the sliding window method?

Inefficient. Not sharing features between intersecting windows.

What is the problem with the second method?

complexity of convolutions at original resolution.

Solution?

Downsampling (pooling, and strided convolutions) and then upsampling (un-pooling and strided transpose convolution)

State 3 methods for un-Pooling:

1. Nearest Neighbor - if max was a 2, set entire region to 2.
2. Bed of Nails - if max was a 2, set top left corner to 2, all others to the minimal value, say 0.
3. Smart Bed of Nails - keep track of max locations during pooling and place the max value back in that location.

Describe the transpose convolution:

Use the input as a weight applied to the convolution filter, use strides to re-build the image.

Write down the 1D convolution and transpose convolution. Using stride 1 and stride >1:

Stride 1:

$$\overrightarrow{x} * \overrightarrow{a} = X\overrightarrow{a}$$

$$\begin{pmatrix} x & y & z & 0 & 0 \\ 0 & x & y & z & 0 \\ 0 & 0 & x & y & z \end{pmatrix} \begin{pmatrix} 0 \\ a \\ b \\ c \\ 0 \end{pmatrix} = \begin{pmatrix} ay + bz \\ ax + by + cz \\ bx + cy \end{pmatrix}$$

Stride 1 - transpose:

$$\overrightarrow{x} *^T \overrightarrow{a} = X^T\overrightarrow{a}$$

$$\begin{pmatrix} x & 0 & 0 \\ y & x & 0 \\ z & y & x \\ 0 & z & y \\ 0 & 0 & z \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy \\ cz \end{pmatrix}$$

Stride = 2:

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{pmatrix} x & y & z & 0 & 0 \\ 0 & 0 & x & y & z \end{pmatrix} \begin{pmatrix} 0 \\ a \\ b \\ c \\ 0 \end{pmatrix} = \begin{pmatrix} ay + bz \\ bx + cy \end{pmatrix}$$

Stride 1 - transpose:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{pmatrix} x & 0 & 0 \\ y & 0 & 0 \\ z & x & 0 \\ 0 & y & 0 \\ 0 & z & x \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} ax \\ ay \\ az + bx \\ by \\ bz \end{pmatrix}$$

Describe the 3D reconstruction setting and architecure.

From set of images to 3D map of points in space.
2D Downsampling (pooling, and strided convolutions), 3D convolutional LSTM, and then 3D upsampling (unpooling and strided transpose convolution).

Describe _ methods for 2D object detection:

1. Classification + Localization: a CNN (pretrained on imagenet usually) FC to classification softmax loss + FC to box (x, y, w, h) with L2 loss.
2. Classification sliding window.
3. R-CNN
4. Fast R-CNN
5. Faster R-CNN

State the issue with the Classification + Localization method:

If we have more than one object in the image, we need a different number of outputs. BAD

State the issue with sliding window:

complexity.

State an optimization for classification window

Region proposals / Selective search.

Explain the R-CNN method:

Training:

1. Obtain a feature extractor network, which can also classify images (AlexNet + fine-tune for example)
2. Per image in training dataset:
   (a) Run ROI algorithm. Per ROI $\sim 2K$:
      i. Feedforward warped version through network to obtain features. Save in large dataset of

$$x = (\text{feature}, \text{RoI BBOX})$$
$$y = (\text{is object}, \text{real BBOX})$$

3. Train SVM to identify objects.
4. Train regression models to warp predicted bbox into true bbox

Test:

1. Run image through RoI algorithm
2. Per RoI feedforward warped version through network.
3. Predict if object using SVM.
4. If object, predict which and predict true BBOX using regression models.

Main 2 problems with R-CNN:

Slow in train and test (2K ROIs), many ad-hoc training objectives (SVM, regression etc)

Explain the Fast-R-CNN method:

Training:

1. Obtain a feature extractor network, which can also classify images (AlexNet + fine-tune for example)
2. Per image in training dataset:
   (a) Run ROI algorithm on image to obtain BBOX.
   (b) Feedforward image through first n layers of network. Per RoI:
      i. Project RoI onto this representation.
      ii. Warp RoI - divide into 7×7 grid and max pool each part.

iii. Feedforward this warped RoI through FC layers to obtain features. Save in large dataset of

$$x = (\text{feature}, \text{RoI BBOX})$$
$$y = (\text{is object}, \text{real BBOX})$$

3. Backprop through: Train classification w/Softmax + linear regression w/Smooth L1 loss

Test:

1. Run image through RoI algorithm
2. Feedforward image through first n layers of network. Per RoI:
   (a) Project RoI onto this representation.
   (b) Warp RoI - divide into 7×7 grid and max pool each part.
   (c) Feedforward this warped RoI through FC layers to obtain features. Per tuple of

$$x = (\text{feature}, \text{RoI BBOX})$$
$$y = (\text{is object}, \text{real BBOX})$$

   Predict true BBOX + presence of object.

Write down the smooth L_1 loss function:

$$\begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & \text{else} \end{cases}$$

What is the difference between the Fast R-CNN and the Faster R-CNN?

Faster has an additional Region Proposal Network before the RoI pooling (two more losses - classification obj/not obj + bbox smooth L1, in total 4).

State a method for instance segmentation:

Mask R-CNN.

What is YOLO/SSD?

You only look once, single shot multibox detection.

How does the method for YOLO/SSD work?

Divide image into a 7×7 grid. For each square: give a score per class of the C classes and regress from each of B base boxes - to final box: $(x, y, w, h, \text{confidence})$

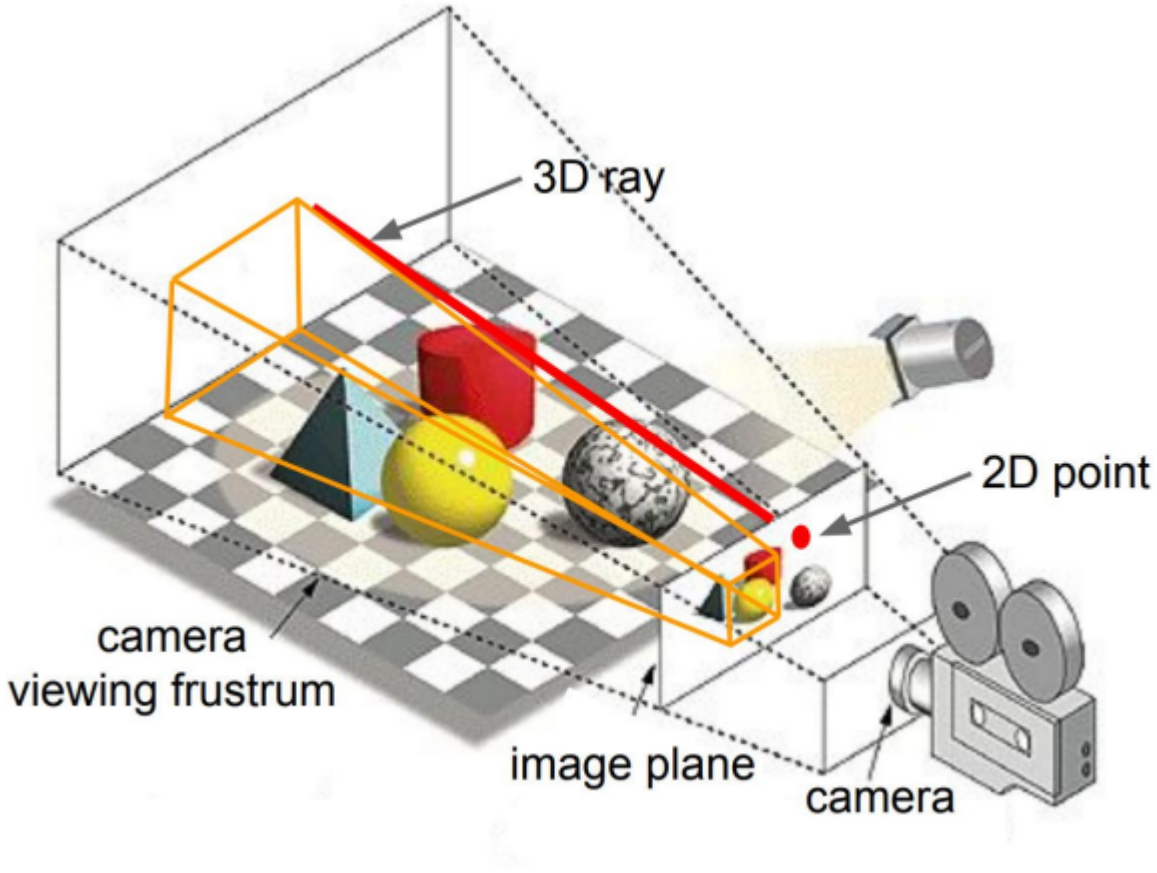
You need to design an object detection network, speed is crucial accuracy not so much. Which architecture?

SSD

You need to design an object detection network, accuracy is crucial speed not so much. Which architecture?

Faster R-CNN

Describe the 3D camera model, give names. Specifically, the wierd one!



What is the target of a 3D object detection task?

$(x, y, z, w, h, l, r, p, y)$ (roll, pitch yaw)

What is the simplified 3D bbox setting?

No roll and pitch.

What is the general method for 3D object detection?

Similar structure to Faster R-CNN. combine 3D proposals from many views and sensors. regress bbox numbers.

Week 11 - generative models

What are the two types of density estimation?

1. explicit - explicitly solve for and define $p_{\text{model}}(x) \sim p_{\text{data}}(x)$
2. implicit - can only sample from some $p_{\text{model}}(x) \sim p_{\text{data}}(x)$

Draw the taxonomy graph of generative models

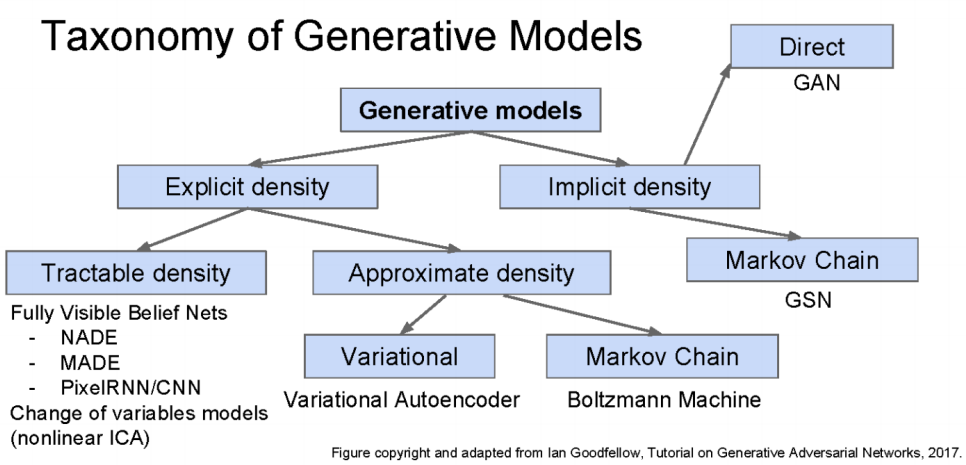


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Describe the main concept behind Fully Visible Belief Networks:

Generate an entire picture via 1D distributions:

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

Each p is expressed via NN. Optimization via Maximum Likelihood estimation on training data. Softmax loss on distribution outputed by NN.

Describe the PixelRNN

Generate image starting from corner. Spatial LSTM.

DrawBack?

Slow. Must work sequentially in training.

Describe the PixelCNN

Row by row, pixel by pixel. Generate via CNN over context region.

Which is faster? State 2 reasons.

PixelCNN:

1. Bound yourself to a smaller context region (no entire top-left hand part of image).
2. Can parralelize training (but not generation)! In the LSTM version we get a hidden state based on all previous images. In the PixelCNN version we can choose any pixel and use the context from it's surrounding in the training image.

State 3 Pros of these two models:

1. Explicitly compute $p(x)$ (how exactly?)
2. Empirical likelihood of training data can serve as validation
3. Good samples (huh?)

State a con of these methods

Sequential generation of pixels is slow.

Describe how to use an Autoencoder for a supervised learning task with few samples:

1. Train the autoencoder.
2. Drop the decoder
3. Fine tune the encoder + FC classifier part on the few labelled examples.

Describe how to use an autoencoder for generating data:

We assume that data is generated in the following manner:
Sample z from true prior

$$p_{\theta^*}(z)$$

Sample x from true conditional:

$$p_{\theta^*}(x|z)$$

How do we usually represent $p_{\theta^*}(z)$, and why?

A simple distribution, say Gaussian. It is reasonable to assume that the latent variables are distributed such (cat images - size of cat in image, pose, head rotation etc.).

How can we make independent features?

Diagonal Σ

State the function for expressing likelihood of sample x, in the latent variable setting:

$p_\theta(x) = \int_z p_\theta(z) \cdot p_\theta(x|z)\, dz$

What is the problem with trying to compute θ which maximizes likelihood in this setting?

Intractable, no way we can optimize for all z

The solution involves using an encoder and decoder network, state what they compute:

Encoder network computes traits of the distribution of z given x. Density function expressed as:$q_\phi(z|x)$. Samples:

$$z|x \sim \mathcal{N}\left(\mu_{z|x}, \Sigma_{z|x}\right)$$

Decoder network computes traits of the distribution of x given z. Density function expressed as:$p_\theta(x|z)$. Samples:

$$x|z \sim \mathcal{N}\left(\mu_{x|z}, \Sigma_{x|z}\right)$$

What is ELBO?

Evidence lower bound. It is a lower bound on the likelihood of our training data given paramaters θ, ϕ of our network. We optimize θ, ϕ such that this lower bound is maximized.

State the KL divergence of two distributions P, Q:

$$D_{KL}(P||Q) = \Sigma_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) = -\Sigma_{x \in X} P(x) \log\left(\frac{Q(x)}{P(x)}\right)$$

Develop the ELBO expression:

For some sample x:

$$\log\left(p_\theta\left(x\right)\right) = \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta\left(x\right)\right)\right]$$

True for any pair of random variables which do not depend on eachother. Now we use bayes' to express $p_\theta\left(x\right)$ differently:

$$\begin{aligned} p_\theta\left(x\right) &= \frac{p_\theta\left(x \wedge z\right)}{\frac{p_\theta(x\wedge z)}{p_\theta(x)}} \\ &= \frac{p_\theta\left(z\right)\cdot p_\theta\left(x|z\right)}{p_\theta\left(z|x\right)} \end{aligned}$$

Let's resume developing the lower bound:

$$\begin{aligned} \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta\left(x\right)\right)\right] &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(\frac{p_\theta\left(z\right)\cdot p_\theta\left(x|z\right)}{p_\theta\left(z|x\right)}\right)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(\frac{p_\theta\left(z\right)\cdot p_\theta\left(x|z\right)}{p_\theta\left(z|x\right)}\frac{q_\phi\left(z|x\right)}{q_\phi\left(z|x\right)}\right)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta\left(x|z\right)\right) - \log\left(\frac{q_\phi\left(z|x\right)}{p_\theta\left(z\right)}\right) + \log\left(\frac{q_\phi\left(z|x\right)}{p_\theta\left(z|x\right)}\right)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta\left(x|z\right)\right)\right] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(\frac{q_\phi\left(z|x\right)}{p_\theta\left(z\right)}\right)\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(\frac{q_\phi\left(z|x\right)}{p_\theta\left(z|x\right)}\right)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta\left(x|z\right)\right)\right] - D_{KL}\left(q_\phi\left(z|x\right)||p_\theta\left(z\right)\right) + D_{KL}\left(q_\phi\left(z|x\right)||p_\theta\left(z|x\right)\right) \\ &\geq \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\left(p_\theta\left(x|z\right)\right)\right] - D_{KL}\left(q_\phi\left(z|x\right)||p_\theta\left(z\right)\right) \\ &:= \mathcal{L}\left(x,\theta,\phi\right) \end{aligned}$$

Last inequality because $D_{KL} \geq 0$ always.

State the optimization objective of training a VAE using the above:

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}\left(x^{(i)},\theta,\phi\right)$$

This term is differentiable!

State 2 pros and 2 cons of VAEs:

Pros:

1. Principled approach to generation
2. Get a feature extractor on the way

Cons:

1. Maximize a lower bound..
2. Generally blurrier and lower quality images than GANs, which are state of the art.

Why are they called Variational Autoencoders?

Still don't know.

What are the two building blocks of GANs, and what does they represent?

Generator - a transformation from a random normal latent vector z into a image
Discriminator - a classifier of real and generated images

What is the GAN objective function?

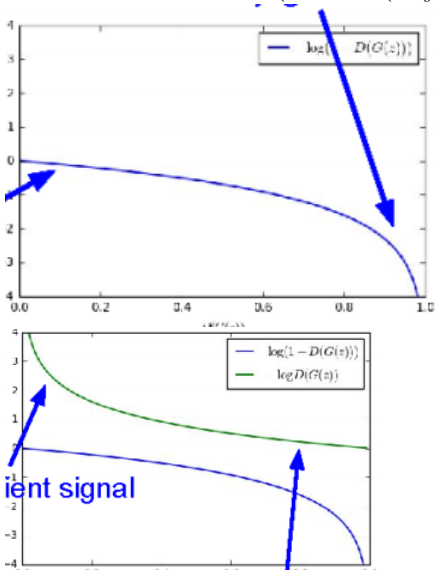
$$\min_{\theta_g}\max_{\theta_d}\left[\mathbb{E}_{x\sim p_{data}}\left[\log\left(D_{\theta_d}\left(x\right)\right)\right]+\mathbb{E}_{z\sim p(z)}\left[\log\left(1-D_{\theta_d}\left(G_{\theta_g}\left(z\right)\right)\right)\right]\right]$$

In practive what are the two function we optimize in GAN?

$$\max_{\theta_D}\left[\mathbb{E}_{x\sim p_{data}}\left[\log\left(D_{\theta_d}\left(x\right)\right)\right]+\mathbb{E}_{z\sim p(z)}\left[\log\left(1-D_{\theta_d}\left(G_{\theta_g}\left(z\right)\right)\right)\right]\right]$$

$$\max_{\theta_G}\left[\mathbb{E}_{z\sim p(z)}\left[\log\left(D_{\theta_d}\left(G_{\theta_g}\left(z\right)\right)\right)\right]\right]$$

*reminder why not $\log\left(1-D_{\theta_d}\left(G_{\theta_g}\left(z\right)\right)\right)$:



Good practices for architecture in GANs:

1. Replace pooling layers with strided conv(discriminator)\strided conv(generator)
2. Use batchnorms in both networks
3. remove fully connected hidden layers in the depth of the network
4. Use ReLU in the generator all layers, except in the output(use tanh)
5. Use Leaky ReLU in discriminator layers

What is the problem with the

Week 14 - Domain Adaptation

What is Domain Adaptation?

A ML settings in which we have little data on the target domain. We are training on a source domain (D_S, y_S) and testing on a target domain $(D_T y_T)$.

examples: (source, tatget): (high quality images, low quality images), (daylight, night time), (posed, "in the wild")

State the two cases of Domain Adaptation.

1. We have labeld source samples.
2. We have labeld source samples and unlabeled target samples.

What is the differnce between Domain Adaptation and transfer learning?

In Domain Adaptation we do not have access to y_T at all, it can be anything $(y_T = y_S \text{or } y_T = -y_S)$.

What assumptions we would have to make on the relationship between the source and the target domains?

That the distributions $D_S, D_T$and the functions y_S, y_T are close in some sense.

$R_{D_T}[h, y_T] \leq R_{D_S}[h, y_S] + dist(D_S, D_T) + dist(y_S, y_T)$

Desribe to first attempt for such assumption:

What is the total variation distance?

The total variation distance: $d_{TV}(D_S, D_T) := 2max_{A \in M} |\mathbb{P}_{D_S}[A] - \mathbb{P}_{D_T}[A]|$

$\mathbb{P}_D[A]$- the probability of $x \sim D$ being in A

M - All possible subsets of the sample space over $D_T$and D_S.

What is the Theorem (Ben-David et al. 2010)?

for any hypothesis h it holds that:
$R_{D_T}[h, y_T] \leq R_{D_S}[h, y_S] + d_{TV}(D_S, D_T) + min\{\mathbb{E}_{D_S}[|y_S(x) - y_T(x)|], \mathbb{E}_{D_T}[|y_S(x) - y_T(x)|]\}$

What does the theorem (Ben-David et al. 2010) gives us?

An upper bound on the argument we would like to minimize $R_{D_T}[h, y_T]$ by summing the source error, the variation distance between $D_S, D_T$and differnces in y_S, y_T.

State the disadvantages of the theorem (Ben-David et al. 2010).

1. Impractical - the second term $min\{\mathbb{E}_{D_S}[|y_S(x) - y_T(x)|], \mathbb{E}_{D_T}[|y_S(x) - y_T(x)|]\}$ cannot be estimated correctly.
2. It's a worst case distance measure
3. It holds for any h, we need only a bound on members oh H.

Desribe the second attempt for an assumption on the relationship between the source and the target domains:

What is the C-divergence?

the C-divergence between D_S and $D_T$is:
$d_C(D_S, D_T) = 2sup_{c \in C} |\mathbb{P}_{D_S}[I(c)] - \mathbb{P}_{D_T}[I(c)]|$
where:
$C = H\Delta H := \{h \, xor \, h' | h, h' \in H\}$
every function $c \in C$ is the set of disagreement between two members of H.
$I(c)$- the set of all x s.t $c(x) = 1$

What is Theorem 2 (Ben-David et al. 2010)?

for any hypothesis h it holds that:
$R_{D_T}[h, y_T] \leq R_{D_S}[h, y_S] + d_{H\Delta H}(D_S, D_T) + \lambda$
where:
$\lambda := min_{h^* \in H}\{R_{D_T}[h^*, y_T] + R_{D_S}[h^*, y_S]\}$

What is the triangle inequality for classification error?

for any labeling function f_1, f_2, f_3:
$R_D[f_1, f_2] \leq R_D[f_1, f_3] + R_D[f_2, f_3]$

Proof Theorem 2 (Ben-David et al. 2010).

We start with the triangle inequality for h, y_t, h^* :
$R_{D_T}[h, y_T] \leq R_{D_T}[h, h^*] + R_{D_T}[h^*, y_T]$
We'll devolpe the only RHS:

$R_{D_T}[h,h^*]+R_{D_T}[h^*,y_T] = R_{D_T}[h,h^*]+R_{D_T}[h^*,y_T]+(R_{D_S}[h,h^*]-R_{D_S}[h,h^*])$
$= R_{D_T}[h,h^*] + R_{D_T}[h^*,y_T] + (R_{D_S}[h,h^*] - R_{D_S}[h,h^*])$
$\leq R_{D_S}[h,h^*] + |R_{D_T}[h,h^*] - R_{D_S}[h,h^*]| + R_{D_T}[h^*,y_T]$

We'll observe for now only the expressions in the absolute value $|R_{D_T}[h,h^*] - R_{D_S}[h,h^*]|$.

Recall that by definition and expected value of bernoulli variable it holds:
$R_D[f_1,f_2] = \mathbb{E}_D[I[f_1(x) \neq f_2(x)]] = \mathbb{P}_D[I[f_1(x) \neq f_2(x)]]$

Addtionly it holds that:
$I[f_1(x) \neq f_2(x)] = [f_1 \, xor \, f_2(x) = 1]$

Now well use this on the expressions in the absolute value taking $f_1 = h, f_2 h^*$:
$|R_{D_T}[h,h^*] - R_{D_S}[h,h^*]| = |\mathbb{P}_{D_T}[h \, xor \, h^*(x) = 1] - \mathbb{P}_{D_S}[h \, xor \, h^*(x) = 1]|$

Now we can take the supremum on all possibe $h \, xor \, h^* = H\Delta H$ as an upper bound:
$|\mathbb{P}_{D_T}[h \, xor \, h^*(x) = 1] - \mathbb{P}_{D_S}[h \, xor \, h^*(x) = 1]| \leq sup_{c \in H\Delta H} |\mathbb{P}_{D_T}[I(c)] - \mathbb{P}_{D_S}[I(c)]|$

And this was defined eariler as:
$d_{H\Delta H}(D_S,D_T) = sup_{c \in H\Delta H} |\mathbb{P}_{D_T}[I(c)] - \mathbb{P}_{D_S}[I(c)]|$

putting in all togather in original RHS:
$R_{D_S}[h,h^*] + |R_{D_T}[h,h^*] - R_{D_S}[h,h^*]| + R_{D_T}[h^*,y_T]$
$\leq R_{D_S}[h,h^*] + d_{H\Delta H}(D_S,D_T) + R_{D_T}[h^*,y_T]$
$\leq R_{D_S}[h,h^*] + d_{H\Delta H}(D_S,D_T) + R_{D_T}[h^*,y_T]$

Again using the triangle inequality it holds:
$R_{D_S}[h,h^*] \leq R_{D_S}[h,y_S] + R_{D_S}[h^*,y_S]$

so we get:
$R_{D_S}[h,h^*] + d_{H\Delta H}(D_S,D_T) + R_{D_T}[h^*,y_T]$
$\leq R_{D_S}[h,y_S] + d_{H\Delta H}(D_S,D_T) + R_{D_T}[h^*,y_T] + R_{D_S}[h^*,y_S]$
$= R_{D_S}[h,y_S] + d_{H\Delta H}(D_S,D_T) + \lambda$

Q.E.D

State the advantages of theorem2 (Ben-David et al. 2010)

1. The $d_{H\Delta H}(D_S,D_T)$ measure is potentially smaller.
2. It can be estimated using samples from D_S, D_T

Prove that the C-divergence $d_{H\Delta H}(D_S,D_T)$ measure is no larger than total variation distance $d_{TV}(D_S,D_T)$.

$d_{H\Delta H}(D_S,D_T) = 2sup_{c \in C} |\mathbb{P}_{D_T}[I(c)] - \mathbb{P}_{D_S}[I(c)]| \leq 2sup_{A \in M} |\mathbb{P}_{D_T}[I(A)] - \mathbb{P}_{D_S}[I(A)]| = d_{TV}(D_S,D_T)$

What is the maximal gap between them?

The maximal gap is 2 ($d_{H\Delta H}(D_S,D_T) = 0$, $d_{TV}(D_S,D_T) = 2$).

Desribe a scenario in which it happens

This happens in a scenario where D_S and $D_T$are over disjoint sets and any discriminator $c \in H\Delta H$ cannot distinguish between them.

$D_S = \{x_1, x_2, x_3\}$ $D_T = \{y_1, y_2, y_3\}$

$d_{TV}(D_S, D_T) = 2sup_{A \in M}|\mathbb{P}_{D_T}[I(A)] - \mathbb{P}_{D_S}[I(A)]| = 2 \cdot (1 - 0) = 2$

for $A = \{x_1, x_2, x_3\}$.

And since any discriminator $c \in H\Delta H$ cannot distinguish between them it holds:

$|I(c) \cap \{x_1, x_2, x_3\}| = |I(c) \cap \{y_1, y_2, y_3\}|$

So assuming D_S and D_T are uniform distributions we have for all $c \in C$:

$\mathbb{P}_{D_S}[I(c)] = \frac{|I(c) \cap \{x_1, x_2, x_3\}|}{3} = \frac{|I(c) \cap \{y_1, y_2, y_3\}|}{3} = \mathbb{P}_{D_T}[I(c)]$

so

$d_{H\Delta H}(D_S, D_T) = 2sup_{c \in C}|\mathbb{P}_{D_T}[I(c)] - \mathbb{P}_{D_S}[I(c)]| = 0$

How can the C-divergence be estimated?

$d_{H\Delta H}(D_S, D_T) \approx d_{H\Delta H}(S_1, S_2)$

for two sets of samples $S_1 \sim D_S^m$ and $S_2 \sim D_T^m$ with large enough m and depnding on H's VC dimension.

What is the Lemma for the C-divergence estimation?

For a symmetric hypothesis class C (one where for every $c \in C$, the inverse hypothesis $1 - c$ is also in C) and samples S_1, S_2 of size m:

$d_C(S_1, S_2) = 2(1 - min_{c \in C}[\frac{1}{m}\Sigma_{x \in S_1} I[c(x) = 0] + \frac{1}{m}\Sigma_{x \in S_2} I[c(x) = 1]])$

What is the procedure for computing the C-divergence with this Lemma?

We find a function in C that has minimum error for the binary classification problem of distinguishing source from target instances. meaning finding c that minimize:

$min_{c \in C}[\frac{1}{m}\Sigma_{x \in S_1} I[c(x) = 0] + \frac{1}{m}\Sigma_{x \in S_2} I[c(x) = 1]]$

Reminds you of something?

GAN

But this is not the "full GAN" yet, it's just the discriminator.

$max_D[\frac{1}{m}\Sigma_{x \in S_1} log(D(x)) + \frac{1}{m}\Sigma_{x \in S_2} log(1 - D(x))]$

What assumption did we make on the two domains that we can maybe drop?

That the two domains are close to each other.

What can we use to drop that assumption?

Use unlabeled samples from the target domain to assume D_S and D_T not close by distance, but share "common characteristics".

What can we assume to say they common characteristics?

Namly assume that there exists a function f that maps D_S and D_T to a features space in which they behave similarly:

$f \circ D_S \approx f \circ D_T$

For instance, for the daylight and sunset domains (dolphins), $f(x)$ can take an image and keep all of its features except the brightness.

How would the learned classifier h would like?

$h = g \circ f$

where f is representation - the first layers of a neural network.

And g is classification - - the last few layers of the neural network.

What is Theorem 3 (Ben-David et al. 2010)?

for any $g \in G$ and $f \in F$:

$R_{D_T}[g \circ f, y_T] \leq R_{D_S}[g \circ f, y_S] + d_{G\Delta G}(f \circ D_S, f \circ D_T) + \lambda_f$

where

$\lambda_f = min_{g \in G}\{R_{D_S}[g \circ f, y_S] + R_{D_T}[g \circ f, y_T]\}$

What can we say when comparing the third λ term in theorm2 and theorm3?

We assume in both cases that it's small, but in theorm3 the assumption is more restrictive, since we assume that the learned representation f is "good".

What is the objective function for an algorithm for domain adaptation?

Minimizing $R_{D_S}[g \circ f, y_S] + d_{G\Delta G}(f \circ D_S, f \circ D_T)$.

formally, for unlabed target data $\{x_i\}_{i=1}^m$ and labeled source data $\{(\hat{x_i}, y_S(\hat{x_i}))\}_{i=1}^m$

$min_{f \in F, g \in G} \frac{1}{m} \Sigma_{i=1}^m I[g \circ f(\hat{x_i}) = y_S(\hat{x_i})]$

$+2(1 - min_{c \in G\Delta G}[\frac{1}{m} \Sigma_{i=1}^m I[c(f(\hat{x_i})) = 0] + \frac{1}{m} \Sigma_{i=1}^m I[c(f(x)) = 1]])$

How can we make our objective both generic and differentiable?

1. Break the objective into two separate losses.

2. Replace $G\Delta G$ with a generic set of discriminators C that return a value in [0, 1].

3. Replace the loss with the CE loss

Rewrite our objective in the generic and differentiable form

Source training error:

$min_{f\in F, g\in G}\frac{1}{m}\Sigma_{i=1}^{m} loss(g \circ f(\hat{x_i}), y_S(\hat{x_i}))$

Domain confusion:

$max_{f\in F} min_{c\in C}[\frac{1}{m}\Sigma_{i=1}^{m} loss(c(f(\hat{x_i})), 0) + \frac{1}{m}\Sigma_{i=1}^{m} loss(c(f(x)) = 1)]$

The second term reminds you of something?

This is a GAN in the latent space.

How can we optimize the objective?

Apply Backpropagation in 3 neural network, one for each function to be optimize: f, g, c. we'll denote each network parameters $\theta_f, \theta_g, \theta_c$.

What is the gradient step for θ_g?

$\theta_g \leftarrow \theta_g - \mu_1 \frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_g} loss(g\circ f(\hat{x_i}), y_S(\hat{x_i}))}{\theta_g}$

What is the gradient step for θ_c?

$\theta_c \leftarrow \theta_c - \mu_2[\frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_c} loss(c(f(\hat{x_i}),0)}{\theta_c} + \frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_c} loss(c(f(x_i),1)}{\theta_c}]$

What is the gradient step for θ_f?

It has two steps.

$\theta_f \leftarrow \theta_f - \mu_3 \frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_f} loss(g\circ f(\hat{x_i}), y_S(\hat{x_i}))}{\theta_f}$

and:

$\theta_f \leftarrow \theta_f + \mu_4[\frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_c} loss(c(f(\hat{x_i}),0)}{\theta_c} + \frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_c} loss(c(f(x_i),1)}{\theta_c}]$

What is the problem with the relationship between f and c in the current optimization method?

f can fool c "too much" and overkill c by making c predict 0 for all $f(x_i)$ and 1 for every $f(\hat{x_i})$

What can c do in such a case in the currect optimization method?

Flip its prediction.

What can be done to avoid this problem?

change the optimization of f.

$\theta_f \leftarrow \theta_f - \mu_4[\frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_c} loss(c(f(\hat{x_i}),\mathbf{1})}{\theta_c} + \frac{1}{m}\Sigma_{i=1}^{m} \frac{\nabla_{\theta_c} loss(c(f(x_i),\mathbf{1})}{\theta_c}]$

Note: the second part is also unnecessary.

Variational Autoencoders - Last Lecture

Write down $P(x)$ using the law of total probability, given latent variable z, paramaterized by θ:

$$p(x) = \int_z P(x|z,\theta)\, p(z)\, dz$$

What do we choose $P(X|z,\theta)$ to be in VAEs?

$\mathcal{N}\left(f(z,\theta), \sigma^2 I\right)$

Why?

1. The log likelihood of X given z becomes proportional to the euclidian distance between X and $f(z,\theta)$.
2. The likelihood of X is differential w.r.t. θ so can perform gradient descent.

What is the justification for always using $z \sim \mathcal{N}\left(0, \sigma^2 I\right)$ for some hyperparameter σ?

We can assume that in it's first layers, the network will learn a mapping from this noise to the "true" latent variables, and from there compute the image.

Write down an approximation for $p(x) = \int_z P(x|z,\theta)\, p(z)\, dz$?

$$p(x) \approx \frac{1}{n} \sum_{i=1}^{n} P(x|z_i)$$

What is the problem with this?

Assuming we choose a small σ (and we generally do, recall the 3 different 2s example), it is very rare to sample a z which leads to $f(z,\theta)$ close to X so we will likely have a sum of zeros.

How does the VAE method overcome this challenge?

We don't compute $p(x)$ through $\mathbb{E}_{z\sim P}[p(x|z)]$ which is hard, but compute $\log p(x)$ through $\mathbb{E}_{z\sim Q(z|X)}[\log(p(x|z))]$ (which is easier due to the more likely presence of z which can lead to X) along with a correction, based on the distance between $Q(X|z)$ and $P(z)$.

Write down the ELBO:

Start with the equality:

$$\log\left(p\left(X\right)\right) - KL\left(Q\left(z|X\right)||P\left(z|X\right)\right) = \mathbb{E}_{z\sim Q(z|X)}\left[\log\left(P\left(X|z\right)\right)\right] - KL\left(Q\left(z|X\right)||P\left(z\right)\right)$$

Now the ELBO:

$$\log\left(p\left(X\right)\right) \geq \mathbb{E}_{z\sim Q(z|X)}\left[\log\left(P\left(X|z\right)\right)\right] - KL\left(Q\left(z|X\right)||P\left(z\right)\right)$$

In the equality above, why can we assume that maximizing the right side will maximize the likelihood of x?

Some claim about "if you have a neural net with high enough capacity we will have $KL\left(Q\left(z|X\right)||P\left(z|X\right)\right) = 0$ and we then optimize the likelihood of x"

What are the paramaters of $Q\left(z|X\right)$?

The paramaters of a neural network which computes two functions:

$$\left(\mu\left(X,\omega\right), \Sigma\left(X,\omega\right)\right)$$

Σ is constrained to be diagonal.

What is the closed form solution of $KL\left(Q\left(z|X\right)||P\left(z\right)\right)$?

$$\frac{1}{2}\left(\text{trace}\left(\Sigma\left(x\right)\right) + \left\|\mu\right\|^2 - k - \det\left(\Sigma\left(x\right)\right)\right)$$

This is derivable.

Describe the feedforward phase for VAEs:

Feedforward:

1. Sample some X
2. Run the encoder $Q\left(z|X\right)$ in order to obtain $\mu\left(X\right), \Sigma\left(X\right)$
3. Sample z using the reparamatization trick: sample $\epsilon \sim \mathcal{N}\left(0, I\right)$, then compute $z = \mu\left(X\right) + \Sigma^{\frac{1}{2}}\left(X\right) \cdot \epsilon$. This z distributes: $\mathcal{N}\left(\mu\left(X\right), \Sigma\left(X\right)\right)$.
4. Run the encoder on z generating $f\left(z\right)$.

Write down the PDF of a multivariate normal function:

$$\left(2\pi\right)^{\frac{-k}{2}} \det\left(\Sigma\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\left(x-\mu\right)^T \Sigma^{-1}\left(x-\mu\right)}$$

Describe the Backpropogation procedure in VAEs:

After the feedforward we backprop in order to maximize:

$$\log\left(P\left(X|z\right)\right) - KL\left(Q\left(z|X\right)||P\left(z\right)\right)$$

We optimize both the encoder and decoder on both losses. We have a closed form solution for the KL divergence.

The log likelihood can be computed as such:

$$\begin{aligned}
\log\left(P\left(X|z\right)\right) &= \log\left(\left(2\pi\right)^{\frac{-k}{2}}\det\left(\sigma^2 I\right)^{-\frac{1}{2}}e^{-\frac{1}{2}\left(x-f(z)\right)^T\left(\sigma^2 I\right)^{-1}\left(x-f(z)\right)}\right) \\
&= \log\left(\left(2\pi\right)^{\frac{-k}{2}}\det\left(\sigma^2 I\right)^{-\frac{1}{2}}\right) + \log\left(e^{-\frac{1}{2}\left(x-f(z)\right)^T\left(\sigma^2 I\right)^{-1}\left(x-f(z)\right)}\right) \\
&= C - \frac{1}{2\sigma}\left(x-f\left(z\right)\right)^T\left(x-f\left(z\right)\right) \\
&= C - O\left(\left\|x-f\left(z\right)\right\|_2^2\right)
\end{aligned}$$

This is maximized when $\left\|x-f\left(z\right)\right\|_2^2$ is minimized, so we simply backprop from this norm.