Topic #1:CS 7150 NEU

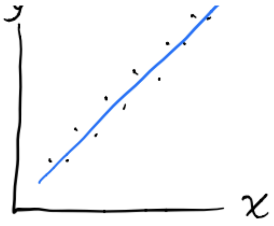
Slide: Supervised Machine Learning Review

- Regression: Predict a continuous value
  - Let $f: \mathbb{R}^d \to \mathbb{R}$

    $y = f(x) + \text{noise}$

    Given: $\{(x_i, y_i)\}_{i=1 \dots n}$

    Find: f

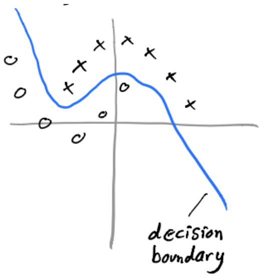
    

- Classification: Predict membership in a category
  - Let $f: \mathbb{R}^d \to \begin{Bmatrix} \text{cat } 1 \\ \vdots \\ \text{cat } m \end{Bmatrix}$

    $y = f(x) + \text{noise}$

    Given: $\{(x_i, y_i)\}_{i=1 \dots n}$

    Find: f

Term:

x – input variables, predictors, independent vars, features

y – response, dependent variable, output variable

f – model, predictor, hypothesis

Statistical Framework for ML(Supervised)

- Assume: (x,y) sampled from a joint probability distribution
- Training data are independent and identically distributed random variables
- Test data are also independent and identically distributed random variables
- Then we can estimate the model by maximum likelihood estimation(MLE).
  - Results in an optimization problem are usually empirical risk minimization(ERM)

When estimating the data's parameter, we usually use MLE or MAP(Maximum A Posterior Estimation).

- MLE is in contrast to MAP

Assume we chose to minimize square loss for a binary classification problem

- Erm does not guarantee to give you a good predictor
    - Could get a local minimum instead of the global minimum
    - Doing well on training data but not on test data leading to overfitting

- Property we want in the learned predictor
    - Good performance on test data
    - Minimize risk (expected loss) under the test distribution

Is risk minimization biased (cultural sense) when applied to real problems where {xi} correspond to people?

- Biased toward the training data when a group is underrepresented in training data, then performance on that group may be worse
- Data itself could have historical biases

Linear Regression and Square Loss

- $Let\ a \in R^d,\ x \in R^d$
- $Model:\ y_i = x_i^t a + \varepsilon_i\ with\ \varepsilon_i \sim N(0, \sigma^2)$
- $Data:\ D = \{(x_{i,} y_i)\}_{i=1\ldots n}$
- Estimate a by maximum likelihood
    - pdf of $\varepsilon_i\ is\ \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-z^2}{2\sigma^2}}\ over\ Z \in R$
    - Likelihood of data $using\ \epsilon_i = y_i - x_i^t a$
    - $L(a) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}}\ e^{-(y_i - x_i^t a)^2 / 2\sigma^2}$
    - $log\ L(a) = -\sum_{i=1}^{n} \frac{(y_i - x_i^t a)^2}{2\sigma^2} + terms\ constant\ in\ a$
    - $maximimizing\ data\ likelihood \Leftrightarrow minimizing\ square\ loss$
    - $max\ L(a) \Leftrightarrow min\ \underbrace{\sum_{i=1}^{n} (x_i^t a - y_i)^2}_{\text{Square loss } \ell(\hat{y}, y) = |\hat{y} - y|^2}$

- ○ Typically Mean Squared Error is used for a regression problem.

Logistic Regression and Cross Entropy Loss

- Model: $Let\ a\epsilon R^{d}$

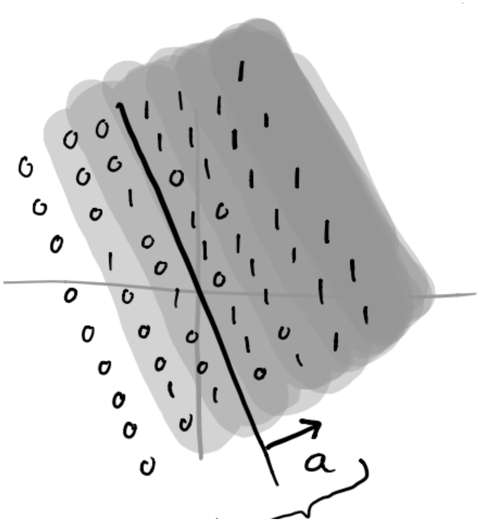
$$p(y\ =\ 1|x)\ =\ \sigma(x^{t}a) \qquad with\ \sigma(z)\ =\ \frac{e^{z}}{e^{z}+1} = \frac{1}{1+e^{-z}}$$

$$p(y\ =\ 0|x)\ =\ 1\ -\ \sigma(x^{t}a) \quad \text{**}\ (x^{t}a\ is\ a\ logit\ )$$

Data: $\{(x_{i}, y_{i})\}$

Logit -> probability of an event occurring

Visual:



-$width\ of\ region\ of\ uncertainty\ \approx \frac{1}{||a||_{2}}$

$Estimate\ a\ by\ maximum\ likelihood$

- $L(a)\ =\ \prod_{i=1}^{n} p(y_{i}\ =\ 0|x_{i})^{1-y_{i}}\ P(y_{i}\ =\ 1|x_{i})^{y_{i}}$
- $log\ L(a)\ =\ \sum_{i=1}^{n}(1\ -\ y_{i})log\ P(y_{i}\ =\ 0|x_{i})\ +\ y_{i}logP(y_{i}\ =\ 1|x_{i})$

Cross Entropy loss

$$l_{CE}(p, q) \;=\; -\sum_{z \epsilon Z} p(z) log\, q(z) \;=\; -E_{p}(log\, q)$$

p,q = discrete r.v.s(random variables) over Z

$$Maximizing\ data\ likelihood \Leftrightarrow minimizing\ cross\ entropy\ loss$$

$$max\ L(a) \Leftrightarrow min\ -\underbrace{\sum_{i=1}^{n}(y_i\, log(\sigma(x_t^{\ t} a)) + (1-y_i)\, log(1-\sigma\,(x_i^{\ t} a)))}_{\ell_{CE}\left(\begin{pmatrix} y_i \\ 1-y_i \end{pmatrix}, \begin{pmatrix} \sigma(x_i^t a) \\ 1-\sigma(x_i^t a) \end{pmatrix}\right)}$$

Cross-Entropy is an asymmetric measure of the distance between two distributions

Logistic Regression is like a simple version of binary classification with neural nets

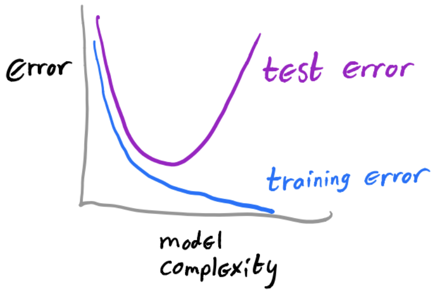
Cross Entropy loss

- Penalizes data points of a observed category to which the model assigns a very low probability

Bias-Variance Tradeoff

-Standard Statstical ML story:

- Higher Complexity modes -> lower bias but higher variance



- If complexity is too high -> overfits data -> variance term dominates test error
    - After certain threshold -> larger models are worse
- As the search space of larger complexity models is larger-> Training error monotonically decreasing
- Test error initially decreasing because if it is too low, it undefits the data and cannot represent the true model

Ie) $how\ complex\ of\ a\ \ data\ model\ would\ you\ consider\ when\ you\ have\ \ 10^3\ data\ samples,$

- Less than $10^3\ could\ be\ 30\ ,\ 100$

Tradeoff matter since it help us to select the right level of complexity and look for evidence of overfitting

Bias-Varaince Decomposition

- Consider Regression model

  $y\ =\ f(x)\ +\ \varepsilon \quad with\ E[\varepsilon|x]\ =\ 0$

  D = $\{(x_i, y_i)\}_{i=1...n}$ be independent and identically distributed random variables

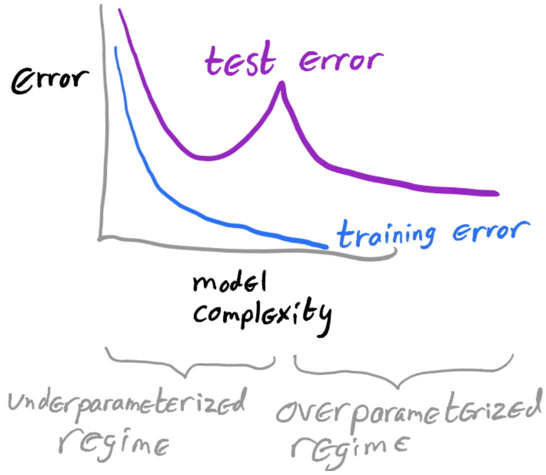
  Estimate f by an algorithm producing f_D

  $Evalulate\ f_D\ by\ expected\ loss\ on\ a\ new\ sample$

  $$R(f_D)\ =\ E_{x,y}(f_D(x)\ -\ y)^2$$

  $$E_D R(f)\ =\ \underbrace{E_x(f(x)\ -\ E_D f_D(x))^2}_{\text{expected squared bias of estimate}} + \underbrace{E_x Var_D f_D(x)}_{\text{expected variance of estimate}}\ +\ \underbrace{var(\varepsilon)}_{\text{irreducible error}}$$

Modern Story based on Neural Nets:



- Test error can decrease as model complexity continues increasing. It can be lower than in underparmeterized regime
    - Double Descent
- If you have10^3 $data\ samples$, how complex of a data model would you consider
    - Choose a neural network with 10000 or 100000 parameters
- Critically parameterize is bad for generatization
    - Critically parameterize : when number of parameters equal to number of data points
- Neural net must contort itself to fit the exact data.
- Infinite amount of model parameters that fit data exactly. Gradient descent will find one of them. Therefore, there would be solutions that do not generalize well.

Good generalization possible in the overparameterized regime

- Parameters that you find from running Gradient Descent from a reasonable initialization have small norm
    - Has a regularizing effect

Slide: Architectural Elements of Neural Networks

Logistic Function and Softmax

- Logistic(Z) = $\frac{e^{Z}}{e^{Z}+1} = \frac{1}{1+e^{-Z}} \, for\ Z \, \epsilon R$
- Softmax = $R^{N} -> R^{N}$

$$\{Z_i\}_{i=1..N}| -> \{\frac{e^{Z_i}}{\sum_{j=1}^{N} e^{Z_j}}\}_{j=1...N}$$

- Output of softmax is a probability distribution

Activation Functions ->Adds non-linearity on Neural Net

:Allows to

- Rectified Linear Unit (ReLU)
    - $\sigma(z) = max(0, z)$
- Absolute value rectification
    - $\sigma(z) = |z|$
    - Used sometimes in object recognition in images -> want invariance to image reversal
- Leaky ReLu
    - $\sigma(z) = \alpha_i min(0, z_i) + max(0, z_i)$
    - $\alpha_i \, could\ be\ set\ to\ a\ fixed\ value\ or\ could\ be\ learned$
- Sigmoid
    - $\sigma(z) = tanh(z)\ or\ logistic(z)$
    - Not recommended for internal/hidden layers due to saturation
    - Usually used for binary classification T,F, last layer's activation function-> consider which of the following among two is closer

Channels and Batching for images

- Image : 3-D matrix Tensor
    - Channels:

        - Gray = 1
        - RGB = 3

A minibatch of images

- Collection of N images a 4-D tensor
    - Pytorch= [N,C,H,W]
    - TensorFlow = [N,H,W,C]

Convolutional Layers

- Math: $S = x * w$

$$S(t) = \int x(a)w(t - a)da$$

Inner product of x with shifted and flipped W.

Term = x- Input, w- Kernel, filter, s- feature map, activation map

Discrete 1-D convolution :

$$s(i) = x * w(i) = \sum_{m} x(m)w(i - m)$$

Discrete 2-D convolution :

$$s(i, j) = x * w(i, j) = \sum_{m}\sum_{n} x(i - m, j - n)w(m, n)$$

Insead of convolution, Machine Learning Libraries implement cross correlation where it does not flip the kernel.

$$s(i, j) = x * w(i, j) = \sum_{m}\sum_{n} x(i + m, j + n)w(m, n)$$

Principles of Conv Layers: Locality, translation invariance

Conv Layers:

    - Pro:
        - Fewer Parameters than Fully-Connected Layers
            - Cheaper

        - Easier to optimize
    - Equivariance to translation
        - Features of image in top left should be treated same as in bottom right
- Visually they are feature detectors

Batch Normalization

- Used for preventing Gradient-Vanising
- Performed in each layers before enters the activation function.
- Improving Speed, reliability, and performance of optimization of Neural Networks

For Input X:

$$y = \frac{x - E[x]}{\sqrt{var(x) + \varepsilon}} * \gamma + \beta$$

- E[x]= Computed from input

Whitens the input

- Var(x)= computed from input
- ε: $Fixed, small$

$\gamma, \beta = Learned$

Why does BatchNormalization Work:

- Internal Covariate shift
    - Change in distribution of network activations due to the change in net parameters during training
- Idea:
    - Modifications in an early layer of net could cause a large change downstream. Batch Noramlization would decouple effects

Gradient Descent and Stochastic Gradient Descent

-GD: used to optimize the weight parameter, loss function's current weight's gradient(slope), train by minimizing the loss

Optimization and machine learning

$Data\,\{(x_i, y_i)\}_{i=1\ldots n},\ Model\ y_\theta(x_i)$

- $min_\theta \sum_{i=1}^{n} l(y_\theta(x_i), y_i)$

Optimization in general

- $min_x f(x)$
- Gradient Descent: Take successive steps downhill
  - $x^{i+1} = x^i - \alpha \nabla f(x^i),$

    $\alpha = Step\ size,\ learning\ Rate,\ \nabla f\ points\ in\ direction\ of\ steepest\ descent$

Alternatives to gradient descent

- Linear Regression
  - Could solve normal equations. Analytical formula, solve it. Closed form formula
- Derivative Free method.
  - Have access to evaluations of the objective f(x).
- Second Order Methods
  - Have access to f(x),grad f(x) and Hessian f(x)
    - $H_{ij} = \frac{d^2 f}{dx_i dx_j}$

If learning rate is large, then there is the possibility of divergence

We do not always need to have a decreasing learning rate in order for Gradient Descent to converge.

Learning Rate qualitatively affect behavior

- Too big
  - Can cause divergence

- Too big
  - Can miss a local well
- Too small
  - Can take a long time to converge

Challenges of gradient descent in deep learning

$$min\theta \underbrace{\frac{1}{n}\sum_{i=1}^{n} l(y_{\theta}(x_i), y_i)}_{f(\theta)}$$

$$\theta^{k+1} = \theta^{k} - \alpha\nabla f(\theta) = \theta^{k} - \alpha\frac{1}{n}\sum_{i=1}^{n} \nabla_{\varepsilon} l(y_{\theta}(x_i), y_i)$$

To evaluate ∇(f(θ), needs to loop through all data

- Downside: Expensive, not possible in some contexts

Way out:

- Use Minibatches
  - Select a minibatch $B \subset \{1, 2,..., n\}$
  - $\theta^{k+1} = \theta^{k} - \underbrace{\alpha\frac{1}{|B|}\sum_{i\epsilon B} \Downarrow_{\theta} l(y_{\theta}(x_i), y_i)}_{\text{use as approximation of } \nabla_{\theta} f(\theta)}$

Stochastic Gradient Descent

- Want to solve $min_{x} f(x)$
- Instead of having access to $\nabla f(x)$, suppose we only have $G(x)\ w/\ E[G(x)] = \nabla f(x)$
- Write SGD as

    - $x^{k+1} = x^{k} - \alpha_{r}G(x^{k})$
        - On average, move in direction of steepest descent
        - May move further from minimizer
- Simple model : additive noise
    - $G(x) = \nabla f(x) + w,\ w\sim N(0, \sigma^{2}I)$
- Use in Machine Learning: Minibatches
    - $f(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_{\theta}(x_{i}), y_{i})$
    - $G(\theta) = \frac{1}{|B|}\sum_{i\epsilon B} \nabla_{\theta}l(y_{\theta}(x_{i}), y_{i})\ \ for\, random\, subset\, B$

Qualitatively

- Fixed step size

  $\alpha,\ x^{k}\, will\, move\, close\, to\, x^{*}\, but\, will\, bounce\, around\, due\, to\, stochasticity$
- Large α

  $Fast\, initial\, convergence\, large\, error$
- Small α

  Slow initial convergence smaller error

Anaylsis of SGD

- Consider a convex f:$R^{d} -> R$

  Supposed $E(G(x)) = \nabla f(x)$

  We say the stochastic gradient is

  $(M, B) - \ bounded$ if

  $$E\, ||G(x)||^{2} \leq M^{2}||x - x^{*}\, ||^{2} + B^{2}$$

  Where $x^{*}\, is\, a\, minimizer\, of\, f$

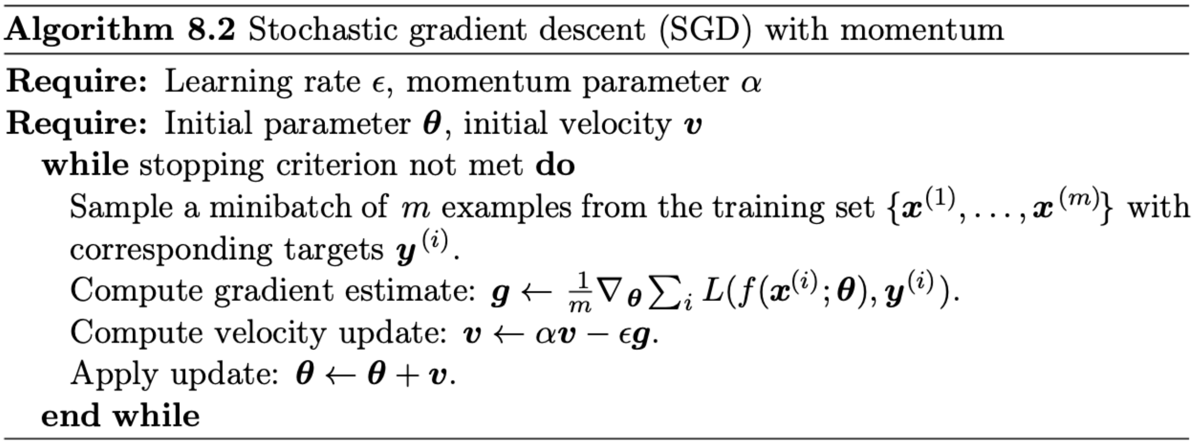
Optimization :Momentum, AdaGrad,RMSProp,Adam.

- Challenges
    - Avoid getting stuck in local minimum

- Need adaptive learning rates
- The objective function may not be stationary

Momentum

- Used for faster learning and improving local minima, applied inertia(m) on SGD.



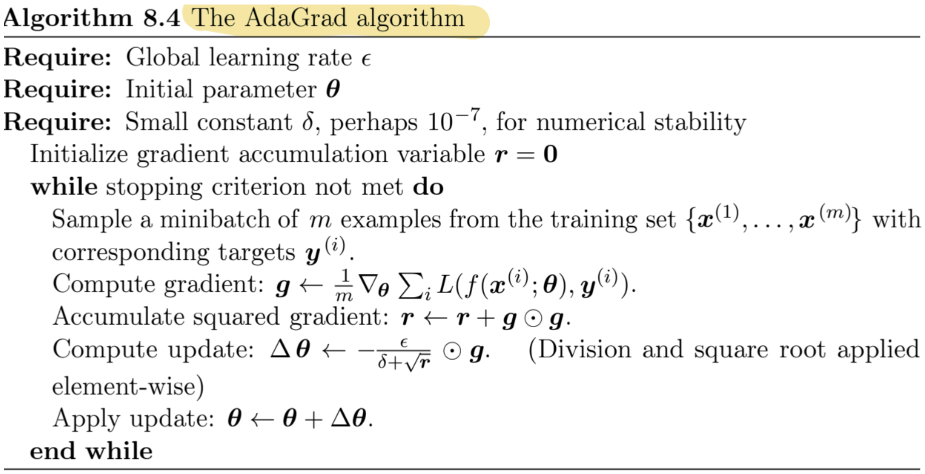
Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while
```

-

AdaGrad

- Use individual learning rates for each parameter
- Set each learning rate inversely proportional to the sum of the square of previous gradients.



Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.    (Division and square root applied
    element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
  end while
```

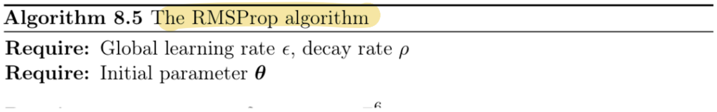
-

RMSProp

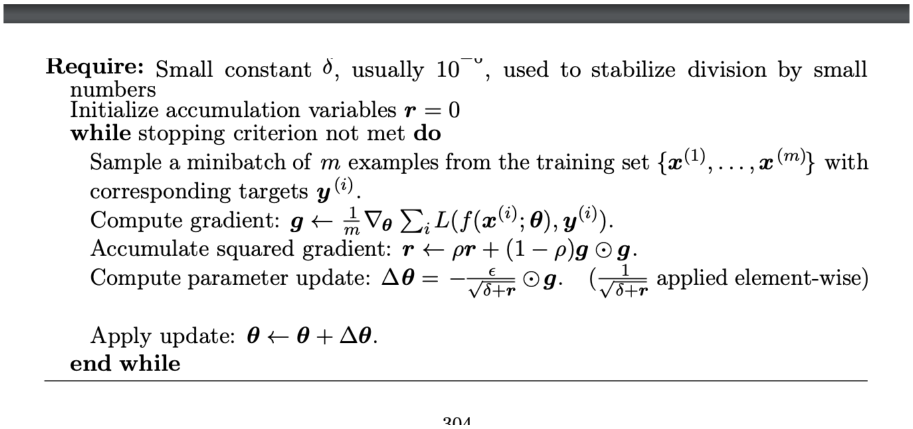
->Root Mean Squared Propagation

- Use of exponential moving average.

- RMS: Root Mean Square
- How
    - Divide the learning rate of each parameter by the root mean square value of its past gradients



```
Algorithm 8.5 The RMSProp algorithm
Require: Global learning rate $\epsilon$, decay rate $\rho$
Require: Initial parameter $\boldsymbol{\theta}$
```



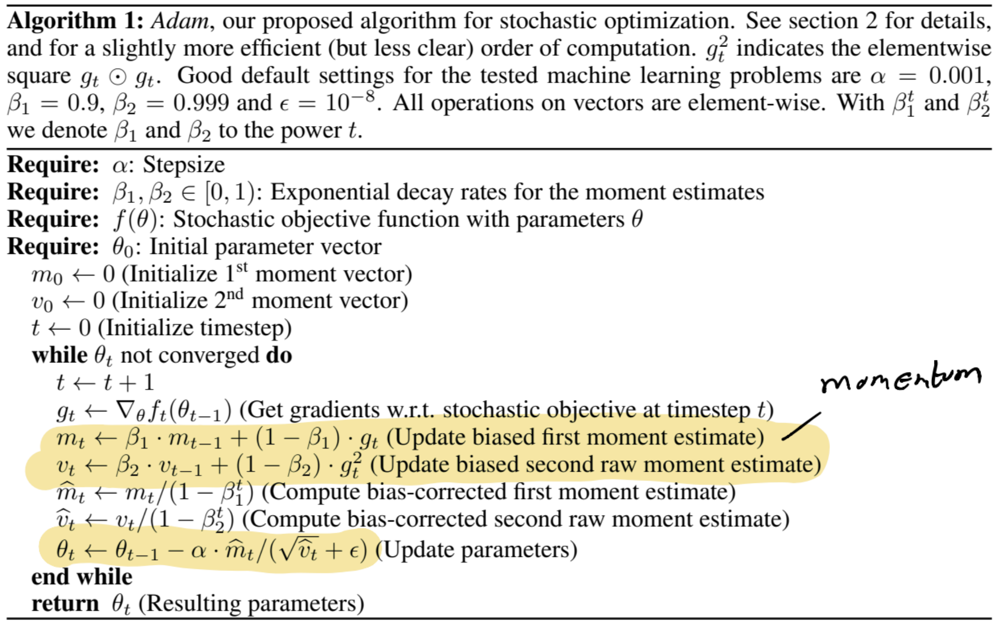
```
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
  end while
```

-

Adam

- -adapts learning rates of each parameter in terms
- -exponentially weighted average of squared gradients (for each parameter)
- -Has momentum(exponential moving average of gradients)
- -Bias Correction term, corrects for fact that the initial value of the moving average is 0
- -Updates with Bias-corrected velocity divided by root bias corrected second moments
- Stepsizes are approximately bounded by step size hyperparameter
- Does not require a stationary objective
    - Adagrad: maintains running sum of g^2
    - RMSProp&Adam: moving exponential average of g^2
-

- 

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)   momentum
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

CNN works a lot better for an image processing compare to MLP

- Why?
  - MLP (Multi-layer Perceptron), takes an input of image, flattens 3-D to 1-D. Does not care about the relationship in between pixel's inside of image
  - CNN leverages locality/geometry of images, enforce translation invariannce, fewer parameters than MLP, can handle images of arbitrary size.

Exercise: Larger or Small filters

- D convolutions layers with filter size k*k
  - Width of the receptive of a neuron field depend on d and k.
    - F(receptive Field) ≈ d*k
  - Parameter cunt depend on d and k
    - $k^2 d = F * k$
  - Are larger or smalll filters more economical in parameter count for a given receptive field?
    - For a fixed receptive field, smaller k uses smaller parameter count

Image Classification: Only one object

Classification with localization: one object and a bounding box for object

Detection: multiple object

Classification with localization:

Defining the target label y

Need to output $b_x, b_y, b_h, b_w,\ class\ label(1\ -\ 4)$

1- pedestrian

2- car

3- motorcycle

4- background

- What would the loss function be if you used cross-entropy loss for classification outputs and MSE for regression outputs
    - Labels y = $(p_c, b_x, b_y, b_w, b_h, c_1, c_2, c_3, c_4)$

      $p_c\ =\ 1\ if\ object\ is\ present,\ 0\ if\ not$

      $c_{1,} c_2, c_3, c_4\ =\ one\ hot\ encoding\ of\ class\ of\ of\ object$

      Output (estimated) y = $(p_c, b_x, b_y, b_w, b_h, c_1, c_2, c_3, c_4)$

      $p_c = classification(Binary\ CE),\ b_{X...h} =\ regression(MSE),\ c_{1..3} = Classification(CE)$

$\hat{y}_i$

Let variable’ = denotes some variable x,y,w,c,z.. ‘s hat(estimator)

$$l(y, y') = \begin{cases} -\ log(1\ -\ p'_c)\ if\ p_c\ =\ 0 & \\ & \text{if } p_c\ =\ 1 \\ -\ log(p'_c) & \\ +\ \sum_i -\ 1_{c_i=1} log(c'_i) & \\ +\ ||\ b_x\ -\ b'_y||^2\ +\ ||b_y\ -\ b'_y||^2 & \\ +\ ||b_w\ -\ b'_w||^2\ +\ ||b_h\ -\ b'_h||^2 & \end{cases}$$

Theoretical underpinning(groundw of this loss?

- Binary Classification
  - $l(y', y)\ =\ log\ y'\ \ if\ y\ =\ 1$

    $=-\ log(1\ -\ y')\ if\ y\ =\ 0$

    $=L_{y=1} log\ y'\ +\ L_{y=0} log(1\ -\ y')$

Output

$y'\ =\ (p'_c, b'_x, b'_y, b'_w, b'_h, c'_1, c'_2, c'_3)$

$p'_c =\ p(object\ is\ there),\ c'_1\ =\ p(object\ is\ of\ class\ 1|object\ present)$

Model(: Given x, there is a prob. Dist of y

$P(class\ i\ \&\ object\ present\ |x)\ =\ p'_c c'_i = p(class\ i\ |\ object_1 x)\ \ *\ P(object|x)$

$b_x|x\ =\ b'_y +\ \epsilon\ N(0, 1)$

Max Likelihood:

$$L_{likelihood}(\theta_{params}) =_i \prod [p'_c c'_1{}^{l_{c_1=1}} c'_2{}^{l_{c_2=1}} c'_3{}^{l_{c_3=1}} e^{-||b_x - b'_x||^2 - ||b_y - b'_y||^2 \dots}]^{p_c} (1\ -\ p'_c)^{1-p_c}$$

Adversarial Examples for Deep Neural Networks

- Formulation
  - Trained Neural Net Classifier
    - $f_{\theta}(x) = y$

      $f = net,\ \theta = parameters,\ x = image,\ y = prob\ dist\ over\ classes$

    $Networks\ Assigns\ to\ x\ the\ class\ c(x) = argmax_{i}y_{i}$

  - For some image x, find perturbation δ

    Such that $c(x + \delta) \neq c(x)$ ->untargeted

    or

    $c(x + \delta) = t$ ->targeted

    goal : x+δ to appear to a human as class c(x)

    Examples:

    - Targeted vs untargeted
    - White box vs black box vs no box
    - Imperceptible vs perceptible
    - Digital vs Phyiscal
    - Specific vs universal
    - Attack vs defense

White Box Attacks

- Projected Gradient Descent
  - $Solve\ x * = argmax_{x' \epsilon P_{x}} L(f(x'), y)$
  - $argmax = maximization,\ P_{x} = set\ of\ images\ w/small\ perturbations\ from\ x$

    $P_{x} = Given\ imae\ x,\ model\ \theta$

    $L = cross\ entropy\ loss,\ y = true\ label\ of\ x,$

Where $P_X = \{x' | \ ||x' - x||_\infty < \varepsilon\}\, for\ example$

$\infty : could\ be\ other\ p - norms,\ \varepsilon = small\ constant$

$||x||_\infty = max_i |x_i|$

- Maximize loss with respect to the true label in order to make system misclassify the image.

Constraining the optimization with the optimization with P_x

- Ensures that each pixel does not change by more than epsilon

Formulation above is untargeted since it never provided a target class as a parameter

Targeted formulation

- $Target\ class\ t,\quad min_{x' \epsilon P_x} L(f(x'|t)$

Carlini-Wagner Attack

- One of the most potent attacks, achieved almost 100% attack success rates for fooling DNN while keeping the same visual quality
  - Want:$min_\delta ||\delta||_P \quad st \quad c(x + \delta) = t$

$$||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$$

$c(x + \delta) = t \quad --> \quad hard\ to\ work\ with\ this\ constraint$

$Suppose\ we\ have\ access\ to\ classifier\ f.\ Notations:\ z = f(x)\ is\ class\ logits$

$Solve: min_\delta\ ||\delta||_p \quad st(max_{i \neq t} z(x + \delta)_i - z(x + \delta)_t)^+ \leq 0$

$max_{i \neq t} z(x + \delta)_i = largest\ logits\ other\ than\ class\ t$

Penalized form : $min_\delta ||\delta||_p + \lambda(max_{i \neq t} z(x + \delta)_i - z(x + \delta)_t)^+$

- Targeted since we constrain the problem to output a perturbation that gets classified as t
- Variant that is untargeted
  - $Let\ y\ be\ True\ Class$

    $min_{\delta}||\delta||_p - \lambda(max_{i \neq y} z(x + \delta)_i - z(x + \delta)_y)^{-}$

FGSM = Fast Gradient Sign Method

- $x^{*} = x + \varepsilon\, sgn\, \nabla_x L(x, y)$

  $x = special\ case\ of\ Projected\ Gradient\ Descent\ ,\ sgn = roughly\ projecting\ on\ l_{\infty}\ ball$

Build a Model for Transformations:

$A(adversarial\ patch(\delta),\ image(x),\ location(l), rotation(T), scale(T))$

Find adversarial patch $\delta\ by\ choosing\ target\ class\ t$:

$$argmax_{\delta} E_{x,l,T}\ logP(image(\ A(\delta, x, l, T)\ is\ class\ t)$$

Adversarial Training

- Train a classifier to classify adversarial perturbations
- FGSM ie):
  - Instead of optimizing $L(\theta, x, y)\ at\ train\ time$

    Optimize

    $\alpha L(\theta, x, y) + (1 - \alpha)\, L(\theta, x + \varepsilon sgn \nabla_x L(\theta, x, y), y)$

    Where $\alpha L(\theta, x, y) + (1 - \alpha) = weighted\ combination$

    And $x + \varepsilon sgn \nabla_x L(\theta, x, y) = FGSM\ adversarial\ example$

Process for computing an adversarial example using the FGSM.

IN:$model : \theta,\ image: x,\ loss: J,\ percepibilit\ threshold : \varepsilon,\ true\ Label : y$

Out:$New\ Image : x + \delta,\ Perturbation = \delta$

$\delta = \varepsilon sgn \nabla_x J(x, y)$

$J(x, y) \ = \ L\,(f_{\theta}(x), y)$

Goal: Increase loss so to achieve a misclassification. Gradient J points in the direction of increased J

Inerpretability:

Dimension 1	**Passive V. Active Approaches**
Passive	Post-hoc explain trained neural networks
Active	Actively change the network architecture or Training process for Better interpetability
Dimension 2	**Type of Explanations(Order of increasing explanatory power)**
Examples	Provide examples which could be considered similar/ as prototype
Attribution	Assign credit to the input features
Hidden Semantics	Make sense of certain hidden neurons/layers
Rules	Extract logic rules
Dimenison 3	**Local V. Global Interpretability(in terms of the Input Space)**
Local	Explain network's predictions on individual samples
Semi-Local	In between, ie) explain group of similar inputs together
Global	Explain the network as a whole

LIME -Local Interpretable Model-Agnostic Explanations

- Unlike most models DNN, Random Forest, Gradient Boosting where the decision tree is black-box. LIME gives an explanation why such predictions are made across prediction model.

$$Explanation(x) = argmin_{g\epsilon G} L(f, g, \pi_x) + \Omega(g)$$

$$Loss\ Function: L(f, g, \pi_x) = \sum_{z,z'\epsilon Z} \pi_x(z)(f(z) - g(z'))^2$$

Learning without Forgetting (LWF)

- Consider predictor with shared parameters across task and some task specific parameters
- At new task, update shared parameters, new parameters, and old parameters
- Output of old task on new data does not change too much

$Start$: θ_s: $shared\ Parameter,\ \theta_o$: $task\ sepific\ param\ for\ each\ old\ task,\ X_{n,}\ Y_n$: $training\ data\ and\ ground\ truth\ on\ the\ new\ task$

$Initialize$: $Y_o <- CNN(X_n, \theta_s, \theta_o)//Comput\ output\ of\ old\ tasks\ for\ new\ data$

$\theta_n <- RANDINT(|\theta_n|)\ //\ Randomly\ Initialize\ new\ parameters$

$Train$: $Define\ Y'_o \equiv CNN(X_n, \theta'_s, \theta'_o)//\ old\ task\ output$

$Define\ Y'_n \equiv CNN(X_{n,} \theta'_s, \theta'_n)\ //\ New\ task\ Output$

$\theta^*_s, \theta^*_o, \theta^*_n <- argmin_{\theta'_s,\theta'_o,\theta'_n}(\lambda_o L_{old}(Y_o, Y'_o) + L_{new}(Y_{n,} Y'_n) + R(\theta'_s, \theta'_o, \theta'_n))$

$\lambda_o = parameter\ for\ plasticity - stability$

$L_{old} = modified\ cross - entrophy\ loss$

$L_{new} = cross - entrophy\ loss$

$R = weight\ decoy$

Elastic Weight Consolidation (EWC)

- When training on task B, identify weights that were important to A and penalize updates to these weights. Try to stay in low error region for A

- Miniminze

  $$L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

  $\frac{\lambda}{2} = parameter.\ \theta^*_{A,i} = Solution\ to\ task\ A$

  Where

  $F_i = E_x(\delta_{\theta_i} log\ p_\theta(x))^2$

  $E_x = diagonal\ entry\ of\ Fisher\ information\ matrix\ of\ predictor\ at\ \theta^*_A$

Generative Adversarial Networks

- Model trained in a game-theoretic adversarial way

  $G: R^k -> R^n\ \ st\ \ if\ z \sim N(0, I_k)\ then\ G(z)\ samples\ from\ a\ learned\ data\ distribution$

  $R^k: laten\ space,\ R^n: image\ space$

  While G induces a distribution on

  $R^n,\ we\ will\ not\ attempt\ to\ maimize\ data\ likehlihood$

- Idea: Generator trying to fool discriminator (concurrently trained)
- Unsupervised, it only had samples

  $\{x_i\}\ of\ a\ training\ distribution,\ No\ one\ labeled$
- Fromulation of GAN training as minimax optimization
  - $Let\ P_d:\ data\ destribution,\ P_Z:\ N(0,\ I_k)$

    $Let\ G:\ R^k -> R^n\ be\ the\ generator$

    $D:\ R^n -> [0, 1]\ be\ P(input\ is\ real)$

    Value function:

    $$V(D, G) = E_{x \sim P_d} log\ D(x) + E_{z \sim P_z} log(1 - D(G(z)))$$

Why optimize this

- negative cross-entropy loss but label=real when $x \sim p_d$

  And label $\neq real$ when $z \sim p_z$

Cross Entropy Loss

$$l_{CE}(p, q) = -\sum_{s \epsilon S'} p(s) log\, q(s) = -E_p(log\, q)$$

Where (p,q) = r.v.s over S

Minimax Formulation

$$min_G max_D E_{x \sim P_d} log\, D(x) + E_{z \sim P_z}(1 - D(G(z)))$$

Max_D: $wants\ to\ maximize\ negative\ cross - entrophy$

Min_G: $wants\ the\ opposite$

- Q: Intractable to compute $E_{x \sim P_d} logD(x)$
  - Perform sampling, Run SGD, estimate that is exact in expectation
- Gan Value function is the right thing to optimize
  - Claim :For fixed G, optimal D is

$$D_G^*(x) = \frac{P_d(x)}{P_d(x) + P_g(x)}$$

Proof:

$$V(D, G) = E_{x \sim P_d} log\, D(x) + E_{z \sim P_z} log(1 - D(G(z)))$$

$$= E_{x \sim P_d} log\, D(x) + E_{x \sim P_g} log(1 - D(G(x)))$$

log(1-D(G(x)):distribution induced by generator

$$= \int_X P_d(x) logD(x) + P_g(x) log(1 - D(x)))dx$$

To find max over D:

Variational calculus and differentiate with respect to D and Set equal to 0

$$\frac{P_d(x)}{D(x)} - \frac{P_g(x)}{1-D(x)} \equiv 0$$

$$D^*(x) = \frac{P_d(x)}{P_d(x)+P_g(x)}$$

- Global minimum of C(G) = $max_D V(G,D)\ is\ unique\ and\ achieved\ \Leftrightarrow P_G = P_D$

  Proof: By previous claim

$$C(G) = E_{x\sim P_d} log\, D_G^*(x) + E_{x\sim P_g} log(1 - D_G^*(x))$$

$$=E_{x\sim P_d} logP_d \frac{2}{p_d+p_g} + E_{x\sim p_g} log\, P_g \frac{2}{p_d+p_g} - log4$$

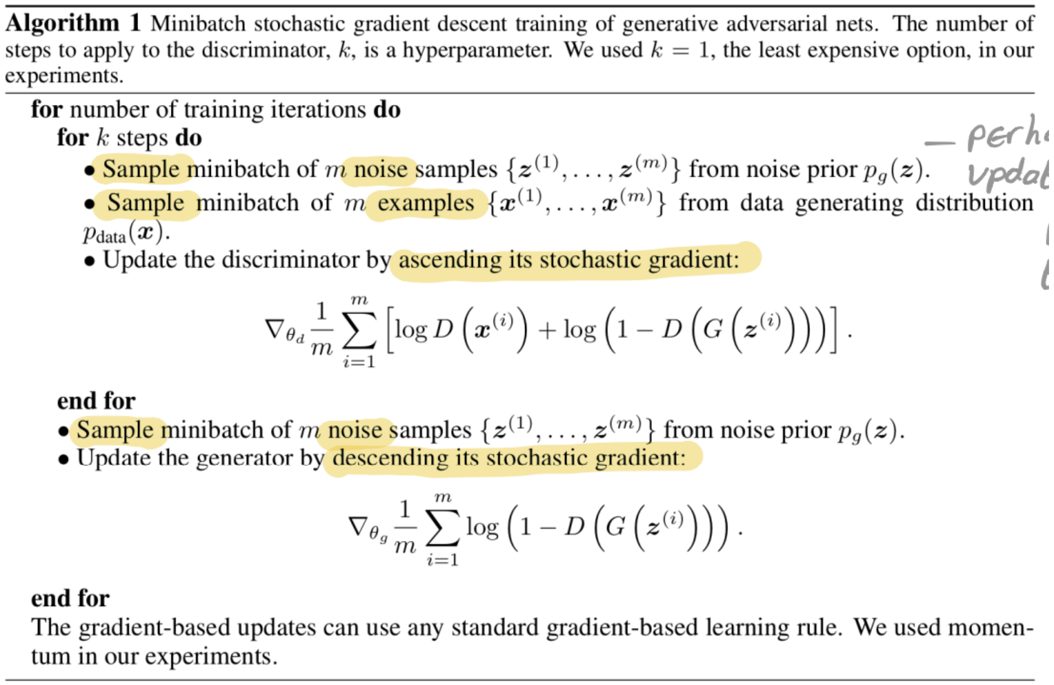
$$= -\ log4 + D_{KL}(P_d||\frac{p_d+p_g}{2}) + D_{KL}(P_g||\frac{p_d+p_g}{2})$$

$$\uparrow\ Nonenegative\ and\ 0\ iff\ \uparrow$$

$$p_d = p_g$$

  - Limits: Non-parametric, infinite capacity models, does not assure the minimax problem, can be solved to global optimality

Minibatch Stochastic Gradient Descent Algorithm



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

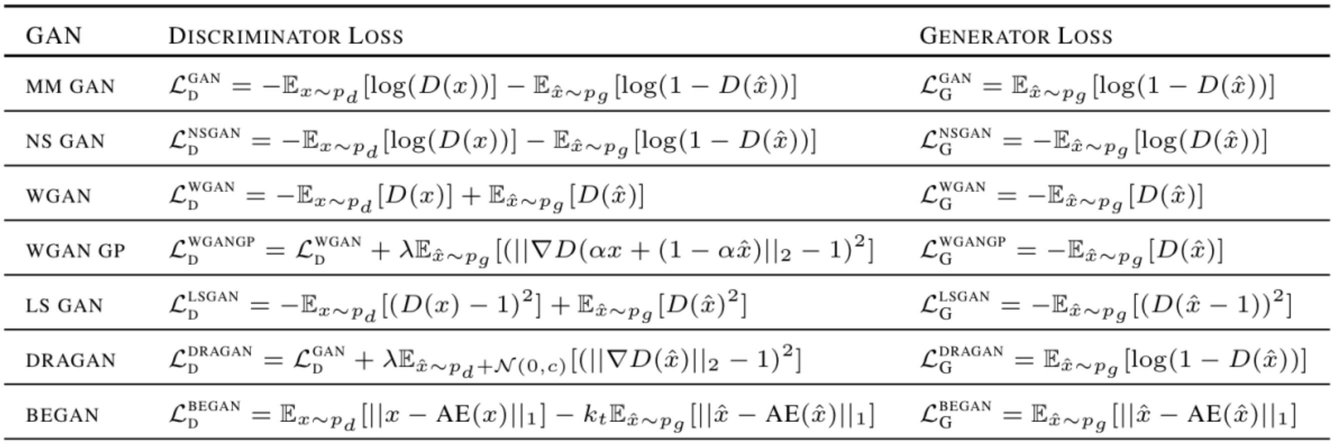
```
for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
        $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

— perh
updat

- Different # of update steps for Discriminator than for G to balance the performance of discriminator and generators.

GAN LOSS



GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\text{D}}^{\text{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\text{D}}^{\text{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\text{D}}^{\text{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\text{G}}^{\text{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\text{D}}^{\text{WGANGP}} = \mathcal{L}_{\text{D}}^{\text{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\|\nabla D(\alpha x + (1-\alpha\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\text{D}}^{\text{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\text{G}}^{\text{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1))^2]$
DRAGAN	$\mathcal{L}_{\text{D}}^{\text{DRAGAN}} = \mathcal{L}_{\text{D}}^{\text{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\|\nabla D(\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\text{D}}^{\text{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|\|x - \text{AE}(x)\|\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\text{G}}^{\text{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$

- Why use NS-GAN instead of MM GAN?
  - NS: Non Saturating. MM: minimax “Vanilla”
  - Vanishing gradients early in training
  - $log\,(1 - D(x'))$ ‘: used to represent “hat” notation

    $\nabla_\theta log(1 - D(x')) = \frac{\nabla D(x')}{1-D(x')}$ $\nabla log(D(x'))$

    $= \frac{\nabla D(x')}{D(x')}$

Wasserstein GAN

- Goal: minimize the distance between P_d and P_g, use Earth mover distance(Waserstein-1 distance)

  $Formally\; W(P_d, P_g) = inf_{\gamma \epsilon \prod(P_d, P_g)} E_{(x,y)\sim\gamma}||x - y||$

  w/$\prod(P_d, P_g) = joint\; distributions\; on\; (x, y)\; s.t.\; marginals\; are\; P_d\, and\; P_g$

Minimize EMD(Earth Mover Distance)?

- Earlier GAN roughly minimizes
  - $D_{KL}(P_d||\frac{p_d+p_g}{2}) + D_{KL}(P_g||\frac{p_d+p_g}{2})$

    $= JS(P_d, P_g)\ \ where\, JS\ =\ JensenShannon\, Divergence$
  - Is not continuous $in\ P_d and P_g,\ but\ EMD\ is$

Approximating EMD with Nets

- Kantorovich-Rubinstein Duality
  - $W(P_d, P_g)\ =\ sup_{||f||_L \leq 1} E_{x\sim p_d} f(x)\ -\ E_{x\sim p_g} f(x)$
  - $||f||_L \leq 1 :\ Lipschitz\ constant{:}\ ||f||_L =\ sup_{x\neq y}\frac{||f(x)-f(y)||}{||x-y||}$

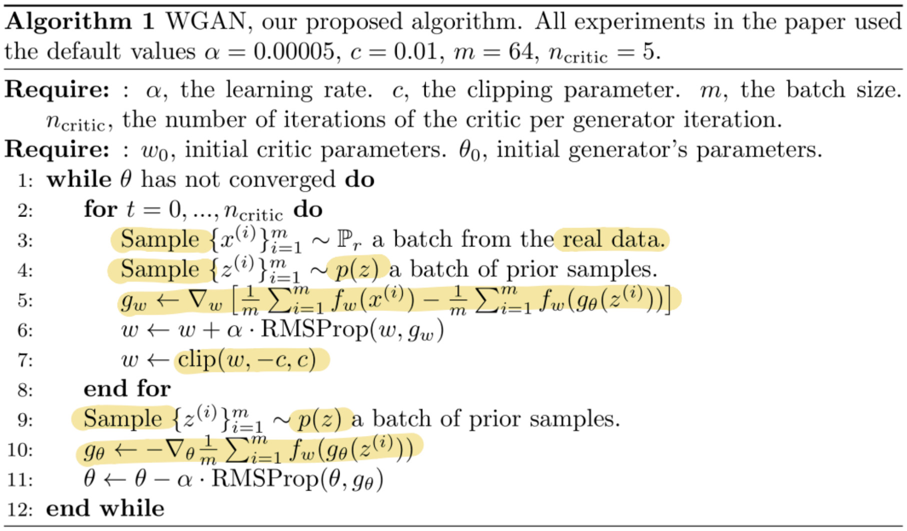
    At the expense of a factor of k, can take sup over $||f||_L \leq K$
- To estimate $W(P_d, P_g)$:
  - $max_{w\epsilon W} E_{x\sim P_d} f_w(x)\ -\ E_{z\sim P_z} f_w(G_\theta(z))$

    Where $f_w\, are\ neural\ nets\ with\ parameters\ w\ in\ a\ compact\ set\ W.$

    $ie)\ each\ w\ is\ in\ [-\ 0.01, 0.01]$

WGAN formulation

- $min_w max_\theta E_{x\sim p_d} f_w(x) - E_{z\sim p_z} f_w(G_\theta(z)) \quad /call\, f_w\, the\, critic$



Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
while $\theta$ has not converged do
    for $t = 0, ..., n_{\text{critic}}$ do
        Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
        Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
        $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
        $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
        $w \leftarrow \text{clip}(w, -c, c)$
    end for
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
    $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while
```

VAE

- Generative model
  - Model that can sample from a learned distribution

Auto-Encoders:

- Attempt to reconstruct the input by learning mappings to and from a code
- Want: $x' = D_\theta(E_\emptyset(x)) \approx x$
- $min_{\theta,\emptyset} \sum_{i=1}^{n} || D_\theta(E_\emptyset(x_i)) - x_i||^2 \quad with\, \{x_i\}_{i=1...n}\, is\, dataset$
- Plain AE is not gen model since it does not define a distribution

$Train\ a\ low\ latent - Dimensional\ gen\ model\ \ by\ likelihood$

- Given Data

  $\{x_i\}_{i=1...n}$, $train\ a\ gen\ model\ to\ maximize\ the\ likelihood\ of\ the\ observed\ data$

If gen model G_θ: $R^k -> R^d\ with\ k < d,$

$$z \mapsto x$$

Then p(x) = 0 almost everywhere,

so cannot directly optimize likelihood

To have nonzero likelihood everywhere,

Define noisy observation model

$$P_\theta(x|z) \;=\; N(x;\; G_\theta(z), I)$$

Under a simple prior p(z), this induces a joint distribution $P_\theta(x, z)$

$$p(x) \;=\; \int p(z)\, p(x|z)\, dz$$

$\uparrow$ *Intractable to evaluate at each iteration, optimize a lower bound instead*

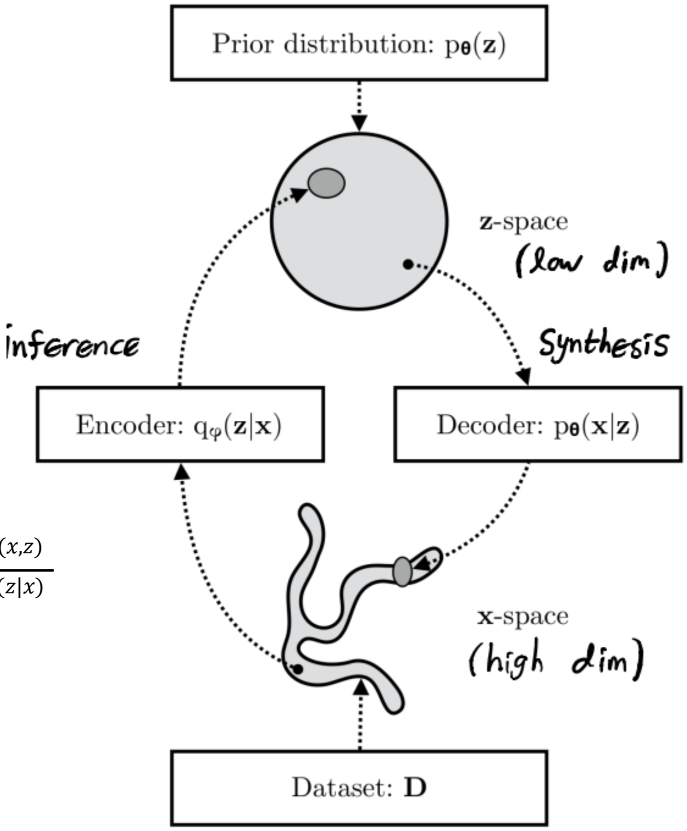
Varaiational Lower Bound

- Setup:
  - Deta generated by

    $z \sim p(z) \qquad prior$

    $x \sim p_\theta(x|z)$

    $Use\ q_\varphi(z|x)\ as\ proxy\ for\ p_\theta(z|x)$



Find: Lower bound to $P_\theta(x)$

$$log\, P_\theta(x) \;=\; E_{z\sim q_\varphi(z|x)} log P_\theta(x) \;=\; E_{z\sim q_\varphi(z|x)} log \frac{P_\theta(x,z)}{P_\theta(z|x)}$$

$$= E_{z\sim q_\varphi(z|x)} log\left(\frac{p_\theta(x,z)}{q_\varphi(z|x)} * \frac{q_\varphi(z|x)}{p_\theta(z|x)}\right)$$

$$= E_{z\sim q_\varphi(z|x)} log \frac{p_\theta(x,z)}{q_\varphi(z|x)} + E_{z\sim q_\varphi(z|x)} log \frac{q_\varphi\ (x,z)}{p_\theta(z|x)}$$

Where $E_{z\sim q_\varphi(z|x)} log \frac{p_\theta(x,z)}{q_\varphi(z|x)} \;=\; L_{\theta,\varphi}(x)$

= variational lower bound(vlb)/ Evidence Lower bound(ELBO)

and $E_{z\sim q_\varphi(z|x)} log \frac{q_\varphi\ (x,z)}{p_\theta(z|x)}$

= $D_{KL}(q_\varphi(z|x)||p_\theta(z|x))$

VLB interpretation:

$$L_{\theta,\varphi}(x) \ = \ \underset{z\sim q_{\varphi}(z|x)}{E} log\frac{p_{\theta}(x,z)}{q_{\varphi}(z|x)} \ = \ E_{z\sim q_{\varphi}(z|x)} log\, P_{\theta}(x|z)\,\frac{p_{\theta}(z)}{q_{\varphi}(z|x)}$$

$$= E_{z\sim q_{\varphi}(z|x)} log\, P_{\theta}(x|z) \ + \ E_{z\sim q_{\varphi}(z|x)} log\,\frac{p(z)}{q_{\varphi}(z|x)}$$

First term : reconstruction Error, Second: regularization

Maximizing VLB

$L_{\theta,y}$: *roughly maximizes* $p(x)$, *minimizes KL divergence of* $q_{\varphi}(z|x)$ *and* $p_{\theta}(z|x)$, *making* q_{φ} *better*

Optimizing Variational Lower Bound

$$max_{\theta,\varphi}\sum_{i=1}^{n} L_{\theta,\varphi}(x_i)$$

- Learn an inference net $x \longmapsto (\mu, \sum)\ with\ q_{\varphi}(z|x) \ = \ N(z; \mu(x), \sum(x))$

$$\uparrow \ or\ \sigma(x)I$$

Optimize VAE parameters with Stochastic gradients