Deep Learning - basics
深度学习 - 基础知识

李建  Jian Li
IIIS, Tsinghua University
Optimization Basics

优化基础知识
Linear Algebra 线性代数

• Vector 向量 $\mathbf{v} = [0, 1, 2, 4, 0, 2]$
• Matrix 矩阵 $\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (2 × 2 matrix)
• Tensor 张量 $\mathbf{T} = [[[1, 0], [1, 1]], [[1, 0], [0, 1]]]$ (2 × 2 × 2 tensor)
• Inner product of two vectors 内积 $\mathbf{x} \cdot \mathbf{y} = <\mathbf{x}, \mathbf{y}> = \mathbf{x}^T \mathbf{y} = \sum_i x_i y_i$
• L2 Norm $||\mathbf{x}||_2 = \sqrt{<\mathbf{x}, \mathbf{x}>} = \sqrt{\sum_i x_i^2}$ (x的长度)
• L1 Norm $||\mathbf{x}||_1 = \sum_i |x_i|$
• General Norm:
  (1) $||\mathbf{x}|| \geq 0$; (2) $||\mathbf{x}|| = 0$ if $\mathbf{x} = 0$ (3) $||c\mathbf{x}|| = c ||\mathbf{x}||$ (4) $||\mathbf{x} - \mathbf{y}|| \leq ||\mathbf{x} - \mathbf{z}|| + ||\mathbf{z} - \mathbf{y}||$
Linear Algebra 线性代数

• Eigenvalue 特征值: $Av = \lambda v$ ($v$是对应特征值$\lambda$的特征向量)

• Positive definite 正定: $x^TAx > 0$ for any $x$
  (equivalently $A$ is symmetric and all eigenvalues are positive)
  A 是对称阵且所有特征值为正

• Semi-Positive definite 半正定: $x^TAx \geq 0$ for any $x$
  (equivalently $A$ is symmetric and all eigenvalues are nonnegative)
  A 是对称阵且所有特征值为非负
Linear Algebra 线性代数

- Two vector $x$, $y$ are orthogonal 两个向量正交
  \[
  \langle x, y \rangle = \sum_i x_i y_i = 0
  \]

- Orthonormal basis \( \{x_i\}_i \) 正交基
  \[
  \langle x_i, x_j \rangle = 0 \text{ if } i \neq j, \quad \langle x_i, x_i \rangle = \|x_i\| = 1
  \]

- Projection of a vector $z$ to a subspace $K$ (of dim $k$) 将向量$z$投影到$k$维子空间$K$
  Suppose $X = \{x_i\}_{i=1,\ldots,k}$ (each $x_i$ is a col of $X$) is an orthonormal basis of $K$
  \[
  P_K(z) = X X^T z \text{ (projection in original space)}
  \]
  $X^T z$ is the projection point in the subspace

\[
\mu_1: \|\mu_1\| = 1 \\
\mu_2: \|\mu_2\| = 1 \\
P_{Xt} = \mu_1 \mu_1^T t + \mu_2 \mu_2^T t
\]
Calculus 微积分

• Gradient 梯度 $\nabla f(x) = \left[ \frac{\partial f}{\partial x_i} \right]_i$

• Hessian 矩阵 $\nabla^2 f(x) = \left[ \frac{\partial^2 f}{\partial x_i \partial x_j} \right]_{i,j}$

Chain rule 链式法则

$$F = f(g_1(x, y, z), g_2(x, y, z))$$

$$\frac{\partial F}{\partial x} = \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x}$$
Math Basics
数学基础知识
凸函数 Convex Functions

• $f \left( \frac{x+y}{2} \right) \leq \frac{1}{2} (f(x) + f(y))$

• $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$, $\forall t \in [0,1]$

(the above two definitions are equivalent for continuous functions)
凹函数 Concave functions

• $f$ is concave if $-f$ is convex
非凸函数 (Nonconvex Functions)
凸函数 Convex Functions

• 一阶条件 First order condition (对可导 for differentiable $f$):
  - $f(y) \geq f(x) + \nabla f(x)(y - x)$

• $\nabla f(x) = \left[ \frac{\partial f}{\partial x_i} \right]_i$

• 二阶条件 Second order condition (对二次可导 for twice-differentiable $f$):
  - Hessian matrix $\nabla^2 f(x)$ is positive semidefinite (psd 半正定)
  - $\nabla^2 f(x) = \left[ \frac{\partial^2 f}{\partial x_i \partial x_j} \right]_{i,j}$
  - $\nabla^2 f(x)$ 刻画了 $f$ 局部的二阶形状 $\nabla^2 f(x)$ specifies the local 2nd order shape of $f$ (how $f$ matches a quadratic function locally)

• 泰勒展开 Recall Tayler expansion:

$$f(y) = f(x) + \nabla f(x)(y - x) + \frac{1}{2} (y - x)^T \nabla^2 f(x)(y - x) + \cdots$$
凸函数 Convex Functions

• 如果 $f$ 不可导呢？What if $f$ is non-differentiable (but still convex)?

• Subgradient
  
  First order condition: $g$ is a subgradient at $x$ if $f(y) \geq f(x) + g^T(y - x) \forall y$

Both $g_2$ and $g_3$ are sub-grad
凸优化 Convex Optimization

• Convex optimization: $f_0, f_1, \ldots$ are convex functions, $h_j$ are linear functions

$$\min_x f_0(x) \quad \text{subject to } f_i(x) \leq 0 \ \forall i$$

$$h_j(x) = 0 \ \forall j$$

$f_0$: objective function 凸的目标函数

All $x$ satisfying the constraints defines a convex region (feasible region).

凸的约束条件（可行域）
凸优化 Convex Optimization

• Linear Programming: （线性规划）

$$\text{minimize } c^T x \text{ subject to } Ax \geq b, x \geq 0$$
凸优化 Convex Optimization

- Quadratic Programming （二次规划）: $P$ is positive semi-definite （$P$是半正定）

$$
\text{minimize } \frac{1}{2} x^T P x + q^T x + r \\
\text{subject to } A x \geq b, x \geq 0
$$

Note that if $P$ is not psd, the objective function is not convex (it can be even concave). 如果$P$不是半正定，那目标函数不是凸的
凸优化Convex Optimization

• 二次锥规划Second Order Cone Program (SOCP)

• 几何规划Geometric Programming (GP)

• 半定规划Semidefinite Programming (SDP)

See the classic book [Convex Optimization] by Stephen Boyd and Lieven Vandenberghe
Sub-gradient Descent for unconstrained minimization:
（无约束求最小值的次梯度下降）

• Iterate until converge: \( x^{(k+1)} = x^{(k)} - \alpha_k g_k \)
• \( \alpha_k \): step size
  • Constant step size: \( \alpha_1 = \alpha_2 = \alpha_3 = \cdots \)
  • Decreasing step size: \( \alpha_k = O(1/k), \alpha_k = O(1/\sqrt{k}), \cdots \)
• Testing convergence
  • \(|f(x^{(k+1)}) - f(x^{(k)})| \) is small enough
  • ....
次梯度下降 Sub-gradient Descent

- Guarantee to converge for convex function 对凸函数来说可以保证收敛
- May converge to local optimal or saddle point for nonconvex functions （非凸函数：局部最优）
随机梯度下降 Stochastic Gradient Descent

- 机器学习中常见的损失函数 Common loss functions in ML
  - $\text{loss}(w) = (1/n) \sum_{i=1}^{n} \ell_i(w)$ (each $\ell_i$ corresponds to a data point, $w$ is the parameter we want to learn)
    - E.g. $\ell_i(w) = (w^T x_i - y_i)^2$

- SGD: Iterate until converge: $w^{(k+1)} = w^{(k)} - \alpha_k h_k$ 迭代到收敛
  - $h_k$ is a random vector such that $\mathbb{E}[h_k] = g_k$
  - For $\text{loss}(w) = (1/n) \sum_{i=1}^{n} \ell_i(w)$, we can choose $h_k = \nabla \ell_i(w^{(k)})$ where $i$ is chosen uniformly at random from $[n]$
    - It is easy to see that $\mathbb{E}[h_k] = \mathbb{E}[\nabla \ell_i(w^{(k)})] = \left(\frac{1}{n}\right) \sum_i \nabla \ell_i(w^{(k)}) = \nabla \text{loss}(w^{(k)})$
    - Hence, in each iteration, we only need one data point
GD vs SGD

Batch: gradient

Stochastic: single-example gradient
SGD

• How to implement SGD (to make it run faster and on larger data sets)

  • Parallel algorithm (Synchronous vs Asynchronous)
    • Analyzing the convergence for asynchronous algorithm can be tricky
      • Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent F. Niu, B. Recht, C. Ré, and S. J. Wright. NIPS, 2011

  • Reduce the variance (减少方差)

  • Mini-batch
    • Instead of computing gradient for each single data point, we do it for a mini-batch (which contains more than 1 points (e.g., 5-20). 每次处理一个mini-batch的样本点

  • System level optimization（系统级优化）
逻辑回归Logistic Regression
Logistic Regression

- Two classes: G=0,1
- \( p(x) = \Pr[G=1 \mid x] \)

\[ \log \frac{p(x)}{1 - p(x)} = \beta_0 + x \cdot \beta \]

\[ p(x; \beta, \beta_0) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}} \]

\( p(x; \beta, \beta_0) \): 模型参数为\( \beta, \beta_0 \)时，点\( x \)属于class 1的概率
Logistic Regression 逻辑回归

• Given training data \( \{x_i, y_i\}_{i=1}^n \), find the parameter \( \beta, \beta_0 \) that maximize Likelihood 找到最大化似然函数的参数:

\[
L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}
\]

• 目标-最大化对数似然 Objective – maximize the log-likelihood

\[
\ell(\beta_0, \beta) = \sum_{i=1}^{n} y_i \log p(x_i) + (1 - y_i) \log 1 - p(x_i)
\]

\[
= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1 - p(x_i)}
\]

\[
= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i (\beta_0 + x_i \cdot \beta)
\]

\[
= \sum_{i=1}^{n} -\log 1 + e^{\beta_0 + x_i \cdot \beta} + \sum_{i=1}^{n} y_i (\beta_0 + x_i \cdot \beta)
\]
Logistic Regression 逻辑回归

• 梯度 The gradient:
  \[
  \frac{\partial l}{\partial \beta_j} = - \sum_{i=1}^{n} \frac{1}{1 + e^{\beta_0 + x_i \cdot \beta}} e^{\beta_0 + x_i \cdot \beta} x_{ij} + \sum_{i=1}^{n} y_i x_{ij}
  \]
  \[
  = \sum_{i=1}^{n} (y_i - p(x_i; \beta_0, \beta)) x_{ij}
  \]

• Cross Entropy (between two distributions p and q):
  \[
  H(p, q) = - \sum_x p(x) \log q(x).
  \]

• The objective of RL is in fact minimizing the cross entropy（逻辑回归的目标事实上是最小化cross entropy）
• 这个是分类问题里最常用的一种loss function
Logistic Regression 逻辑回归

• Multiclass 多分类逻辑回归:

\[
Pr(Y = c | X = x) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}
\]

• Easy exercises 习题:

(1) Compute the log-likelihood for multiclass LR and its gradient
计算多分类逻辑回归的对数似然函数和其梯度

(2) Express the objective as the cross entropy function
将目标函数表示成cross entropy函数的形式

Softmax function
Logistic Regression 逻辑回归

• Essentially linear decision boundary

\[ p(x; b, w) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}} \]

Feature is combined linearly 特征是线性的综合起来

• We can apply a nonlinear function (e.g., a deep neural network) to the feature

我们可以作用非线性函数（如深度神经网络）
支持向量机和最大Margin
SVM and The Idea of Max Margin
Support Vector Machines (SVM)

- Derived from statistical learning theory by Vapnik and Chervonenkis (COLT-92) （来源于统计学习理论， Vapnik and Chervonenkis）
- Base on convex optimization. Solid theoretical foundation. （基于凸优化，理论基础强）
- Mainstream machine learning method. Very successful for many problems for many years. （主流方法之一，广泛应用）
- The ideas and algorithms are very important in the development of machine learning. （其中的想法对于机器学习发展极其重要）
- The max-margin idea (the hinge loss) extends to many many learning problems （可以拓展到很多其他问题）
  - Can be incorporated in deep learning （可以和深度学习结合）
Two Class Problem: Linear Separable Case
2分类问题：线性可分

• Many decision boundaries can separate these two classes
  有很多分界面

• Which one should we choose?
  我们到底选哪个?

Note: Perceptron algorithm can also be used to find a decision boundary between class 1 and class 2
Example of Bad Decision Boundaries
不好分界线的例子
Good Decision Boundary: Maximizing the Margin

- The decision boundary should be as far away from the data of both classes as possible (分界面离两类数据尽可能远)

We should maximize the margin, \( m \) 即最大化margin

This is the simplest kind of SVM (Called an Linear SVM)
这是线性SVM（分界面是线性函数）
好的分界面：最大化margin
Good Decision Boundary: Maximizing the Margin

• Maximize the margin, $m$
好的分界线：最大化margin
Good Decision Boundary: **Maximizing the Margin**

- Maximize the margin, $m$

\[
m = \frac{2}{\|w\|} \quad \|x\| := \sqrt{x_1^2 + \cdots + x_n^2}.
\]
The Optimization Problem

- Let \( \{x_1, \ldots, x_n\} \) be our data set and let \( y_i \in \{1, -1\} \) be the class label of \( x_i \)
- The decision boundary should classify all points correctly \( \Rightarrow \)
  \[
y_i(w^T x_i + b) \geq 1, \quad \forall i
  \]
- A constrained optimization (Quadratic Programming) problem
  
  \[
  \text{Minimize} \quad \frac{1}{2}||w||^2
  \]
  
  subject to \( y_i(w^T x_i + b) \geq 1 \quad \forall i \)
A Geometrical Interpretation
几何解释

The boundary does not depend on the internal pts (can be seen from complementary slackness)
线性不可分问题 Non-linearly Separable Problems

- 允许分类错误 $\xi_i$ We allow “error” $\xi_i$ in classification; it is based on the output of the discriminant function $\mathbf{w}^T \mathbf{x} + b$
- $\xi_i$ 近似误分类样本数 $\xi_i$ approximates the number of misclassified samples

New QP:

$$\frac{1}{2} || \mathbf{w} ||^2 + C \sum_{i=1}^{n} \xi_i$$

Subject to

$y_i (\mathbf{w} \mathbf{x}_i + b) \geq 1 - \xi_i$ for all $i$

$\xi_i \geq 0$ for all $i$

$C$ : tradeoff parameter

What happens for very large $C$?

For very small $C$?
Hinge Loss

- Hinge Loss:
  \[ \ell(t) = \max(0, 1 - t) \]
- Think it as a Convexification of 0-1 loss.
  可以把hinge loss当成0-1 loss的凸函数替代
- A reformulation of non-separable SVM using Hinge Loss:

\[
\begin{align*}
\min & \quad \frac{1}{2} \|w\|^2 + C \sum_i \ell(y_i(wx_i + b)) \\
\text{S.t.} & \quad y_i(wx_i + b) \geq 1 - \xi_i \text{ for all } i,
\xi_i \geq 0 \text{ for all } i
\end{align*}
\]
Hinge损失函数 Hinge Loss

• More generally, for classification function, the hinge loss of point $x_i$:
  我们可以对一般的分类问题定义Hinge loss
  
  $$\ell(f(x_i), y_i) = \max(0, 1 - y_i f(x_i))$$

• In linear SVM, $f(x) = wx + b$

• If $f(x_i)$ has the same sign as $y_i$, and $|f(x_i)| \geq 1$, the loss $\ell(f(x_i), y_i) = 0$
  如果$f(x_i)$和$y_i$同符号, 并且$|f(x_i)| \geq 1$, 损失函数 $\ell(f(x_i), y_i) = 0$

• $f$ can be a function defined by deep neural network
  $f$可以是由深度神经网络定义的

• Hinge loss is a convex loss function (but not differentiable). Hence, we can use standard sub-gradient descent algorithm to solve it.
  该损失函数是一个不可微的凸函数，所以我们需要使用次梯度下降
Other (surrogate) Loss Functions

- Log (logistic) loss:
  \[ \ell(f(x_i), y_i) = \log_2 (1 + e^{-y_i f(x_i)}) \]

The loss for logistic regression \((y_i = \pm 1)\)

(the loss in previous slides was for \(y_i = 0, 1\))
Other (surrogate) Loss Functions

- Modified Huber Loss:
  \[
  \ell(f(x_i), y_i) = \begin{cases} 
  -2y_if(x_i) + 1 & y_if(x_i) \leq 0 \\
  (y_if(x_i) - 1)^2 & 0 < y_if(x_i) \leq 1 \\
  0 & \text{otherwise}
  \end{cases}
  \]

- Exponential loss (in boosting)
- Sigmoid loss (nonconvex)
正则化项 Regularization

- SVM: \( \min \frac{1}{2} \|w\|^2 + C \sum_i \ell(f(x_i), y_i) \)
- More generally:

\[
\underset{w}{\text{minimize}} \quad \text{penalty}(w) + C \sum_i \elloss(f(x_i), y)
\]

Regularization: What do we want about \( w \)
- \( \ell_2: \|w\|_2^2 \)
- \( \ell_1 (\text{LASSO}): \|1\|_1 \)
  
  (it encourages the sparsity of \( w \))

Loss: how well the function fits the training data

Why regularization?
- One important reason: prevent overfitting. 防止过拟合
- Better generalization to new data points 泛化性更好
支持向量机实现SVM implementations

• Svmlight:  http://svmlight.joachims.org/
• LIBSVM and LIBLINEAR
• Implemented in many machine learning libraries:
  • sofia-ml (google)
  • scikit-learn (python)
  • matlab
次梯度下降 Sub-gradient Descent

• Hinge损失的次梯度 Subgradient for Hinge loss:

- \[ \frac{\partial \ell}{\partial w_i} = -y_i x_i \text{ if } y_i f(x_i) < 1 \]
- \[ \frac{\partial \ell}{\partial w_i} = 0 \text{ if } y_i f(x_i) \geq 1 \]

• Coding HW:
Implement the subgradient descent algorithm for SVM (create a simple 2-d example and visualize it)
Imagine that the search engine wants to rank a collection of documents for a query. To train the model, we use training data 

\((q_1, r_1^*), (q_2, r_2^*), \ldots, (q_n, r_n^*)\).

For each document \(d\) and a query \(q\), we can produce a feature vector \(\Phi(q, d)\).

We want to learn a good ranking function. Assume linear ranking functions: \(d_i\) is better than \(d_j\) iff 

\[ \overline{w} \Phi(q, d_i) > \overline{w} \Phi(q, d_j) \]

The weight vector we want to learn.
支持向量机排名 SVM-Rank

- Training data (query, ranking): \((q_1, r_1^*), (q_2, r_2^*), \ldots, (q_n, r_n^*)\).
- We want to following set of inequalities hold for the training data

对于训练数据，希望以下不等式成立

\[ \forall (d_i, d_j) \in r_1^* : \quad \bar{w} \Phi(q_1, d_i) > \bar{w} \Phi(q_1, d_j) \]

\[ \vdots \]

\[ \forall (d_i, d_j) \in r_n^* : \quad \bar{w} \Phi(q_n, d_i) > \bar{w} \Phi(q_n, d_j) \]
支持向量机排名 SVM-Rank

- Training data (query, ranking): \((q_1, r_1^*), (q_2, r_2^*), \ldots, (q_n, r_n^*)\).
- Natural Idea: Maximize the margin 最大化margin

**Optimization Problem 1. (Ranking SVM)**

\[
\begin{align*}
\text{minimize:} & \quad V(\vec{w}, \vec{\xi}) = \frac{1}{2} \vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k} \\
\text{subject to:} & \quad \forall (d_i, d_j) \in r_1^* : \vec{w} \Phi(q_1, d_i) \geq \vec{w} \Phi(q_1, d_j) + 1 - \xi_{i,j,1} \\
& \quad \ldots \\
& \quad \forall (d_i, d_j) \in r_n^* : \vec{w} \Phi(q_n, d_i) \geq \vec{w} \Phi(q_n, d_j) + 1 - \xi_{i,j,n} \\
& \quad \forall i \forall j \forall k : \xi_{i,j,k} \geq 0
\end{align*}
\]

Optimizing Search Engines using Clickthrough Data, Joachims
神经网络基础 Neural Network Basics
人工神经网络 Artificial Neural Networks
人工神经网络Artificial Neural Networks

- First proposed by Warren McCulloch and Walter Pitts (in 1940s)
  首先被Warren McCulloch and Walter Pitts和Walter Pitts提出
人工神经网络 Artificial Neural Networks
激活函数 Activation function

**Sigmoid**
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**  \( \tanh(x) \)

**ReLU**  \( \max(0, x) \)

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

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

**ELU**
\[ f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \]
把神经网络看成函数 View NN as functions

\[ M(w, x) = \sum_{j=1}^{N_h} w_j^{(2)} \phi \left( \sum_{i=1}^{d} w_{ji}^{(1)} x_i \right) \]

\[ M(w, x) = \sum_{j(i(\lambda))}^{N_{i(\lambda)}} w_{j(i(\lambda))}^{(\lambda)} \phi \left( \sum_{j(i(\lambda-1))=1}^{N_{i(\lambda-1)}} w_{j(i(\lambda-1), j(i(\lambda-1)))}^{(\lambda-1)} \phi \left( \sum_{j(i(\lambda-2))=1}^{N_{i(\lambda-2)}} w_{j(i(\lambda-2), j(i(\lambda-2)))}^{(\lambda-2)} \cdots \phi \left( \sum_{j(i(1))=1}^{N_{i(1)}} w_{j(i(1))}^{(1)} x_{j(i(1))} \right) \right) \]
前向计算 Forward Computation

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)
计算梯度 Compute the gradient (Back Propagation)

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]
Compute the gradient (Back Propagation)

\[ f(x, y, z) = (x + y)z \]

\[ \text{e.g. } x = -2, y = 5, z = -4 \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]
Compute the gradient (Back Propagation)

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \)
计算梯度 Compute the gradient (Back Propagation)

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:
\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}
\]
Compute the gradient (Back Propagation)

链式法则 Chain rule (to compute $\frac{\partial f}{\partial \omega}$)

$$F = f(g_1(x, y, z), g_2(x, y, z))$$

$$\frac{\partial F}{\partial x} = \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x}$$

e.g. consider $f(\omega, x) = \frac{1}{1+e^{-1(\omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2)}} = \delta(\omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2)$. sigmoid function $\delta(x) = \frac{e^{-x}}{1+e^{-x}}$

Note: $\frac{d\delta(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = (1 - \delta(x))\delta(x)$

$$\frac{\partial f}{\partial \omega_1} = \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial \omega_1} = (1 - \delta(y))\delta(y)x_1 \text{ where } y = \omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2$$
Compute the gradient (Back Propagation)

Backprop tries to compute $\frac{\partial f}{\partial \omega}$ backwards

$t, t_1, t_2$ are the output variables of corresponding gates

Suppose we have already computed $\frac{\partial f}{\partial t_1'}, \frac{\partial f}{\partial t_2'}, \cdots$

$t_1' = T_1(t_1, t_2, \ldots), t_2' = T_2(t_1, t_2, \ldots)$

$$\frac{\partial f}{\partial t_1} = \frac{\partial f}{\partial t_1'} \cdot \frac{\partial t_1'}{\partial t_1} + \frac{\partial f}{\partial t_2'} \cdot \frac{\partial t_2'}{\partial t_1} + \cdots$$
后馈传播 Back Propagation

\[ f(x, y) = \frac{x + \delta(y)}{\delta(x) + (x + y)^2} \]

```plaintext
x = 3 # example values
y = -4

# forward pass
sigy = 1.0 / (1 + math.exp(-y)) # sigmoid in numerator   #(1)
num = x + sigy # numerator        #(2)
sigx = 1.0 / (1 + math.exp(-x)) # sigmoid in denominator #(3)

xpy = x + y                        #(4)
xpysqr = xpy**2                     #(5)

den = sigx + xpysqr # denominator  #(6)
invden = 1.0 / den                  #(7)
f = num * invden # done!           #(8)
```
We better cache the forward variables as they are useful in computing gradient as well.
后馈传播 Back Propagation

ReLU Gate

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \max(0, x) \]

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial x} \text{ if } x \geq 0 \]

\[ \frac{\partial f}{\partial y} = 0 \text{ if } x < 0 \]
Training NN

Preprocessing:
• 零均值化 Zero mean
• 白噪声化 Whitening: equalize the top-k principle directions

Images by top-144 eigenvectors
After equalizing the top-144 directions

Note: not used so much in CNN
(see PCA. next time)
白噪声化Whitening
Training: Mini-batch SGD  

Sample a batch of data  

Forward (compute all forward variables)  

$$L = \sum_{i \in \text{minibatch}} L(f^{(i)}, y^{(i)})$$  

Backward to compute the gradient  

$$\nabla_{\omega} L = \left( \frac{\partial L}{\partial \omega_1}, \frac{\partial L}{\partial \omega_2}, \ldots \right)^T$$  

Update the weight  

$$\omega \leftarrow \omega + \eta \nabla_{\omega} L$$
损失函数Loss functions

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i \]

- \( L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1) \) (multiclass SVM loss) recall \( \max(0, x) \) hinge loss
  
  \[ f = (f_1, ..., f_j, ..., f_d) \] output of NN
  
  e.g. \( f = (3, -5, 2.5) \) the true label is \( y_i = 0 \)

\[ L = \max(0, -5 - 3 + 1) + \max(0, 2.5 - 3 + 1) = 0 + 0.5 \]

如果是线性模型，对应SVM

- \( L_i = -\log(\frac{e^{f_{y_i}}}{\sum_{j} e^{f_j}}) \) (cross-entropy loss/soft-max)

如果是线性模型，对应逻辑回归

因此，我们可以将深度神经网络的最后一层（loss）看成一个传统的机器学习模型（LG或SVM）
而前面的层看成对原始输入特征的非线性变换
而深度神经网络的作用是使得经过非线性变换的高阶特征变得更容易学习
权重新初始化

- 全为 0 Zeros
- 小的随机数 Small random numbers $\sim N(0, 0.01)$
- 矫正方差 Calibrating the variance

$$\omega \sim \frac{1}{\sqrt{n}} N(0, 1)$$

确保输出相似分布

$$Var\left(\sum_{i=1}^{n} \omega_i x_i\right) = \sum_{i=1}^{n} Var(\omega_i) Var(x_i)$$

(assuming $E[x_i] = 0$)
Training NN

Tricks:

• Choose successive examples from different classes 从不同类中选择连续的样本
  --Try to update gradient fast 快速更新梯度
• Choose the data with large error first 选择错得多的样本
  --careful: outliers would be disastrous 注意 outliers
• Momentum 动量更新方法

\[
\Delta \omega^{t+1} \leftarrow \eta \nabla_{\omega} L + \mu \Delta \omega^t \\
\omega \leftarrow \omega + \Delta \omega^{t+1}
\]

Useful in the directions with low curvature 在曲率小的方向上更有效
Adagrad [Duchi, Hazan, Singer JMLR 11]

Projected grad descent 投影梯度下降

\[ x_{t+1} = \Pi_X(x_t - \eta g_t) = \arg\min_{x \in X} \|x - (x_t - \eta g_t)\|_2^2 \]

Steepest descent in Mahalanobis norm \[ \|x\|_A = \sqrt{x^T A x} \]

\[ x_{t+1} = \arg\min_{x \in X} \left| x - \left( x_t - \eta G_t^{-\frac{1}{2}} \right) \right|_{G_t^{1/2}} \]

\[ G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T \]

Unit norm of \( G_t^{-1/2} \):
\[ x^T G_t^{-1/2} x = 1 \]
\[ x^T \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} x = 1 \]

The above is computation expensive
Adagrad

Diagonal adaptation

\[ x_{t+1} = \arg\min_{x \in X} \left\| x - (x_t - \eta \text{diag}(G_t)^{-1/2} g_t) \right\|_{\text{diag}(G_t)^{1/2}} \]

Try to make the first order method better conditioned

Related to FTRL, dual averaging, proximal method

In ADAGRAD, after a few iteration, the variance in y direction accumulate
RMSProp

[Tieleman, Hinto 2012]

\[ V_t = \text{decayrate} \times v_{t-1} + (1 - \text{decayrate}) \times \begin{pmatrix} g_{t1}^2 \\ \vdots \\ g_{ti}^2 \\ \vdots \end{pmatrix} \]

\[ x_{t+1} \leftarrow x_t - \eta(v_t)^{-1/2} g_t \]

Tensorflow Code:
```
tf.train.RMSPropOptimizer(learning_rate, decay=0.9, momentum=0.0, epsilon=1e-10)
```

Keras Code:
```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
```
ADAM

\[ g_t \leftarrow \nabla f \]

\[ m_t \leftarrow \beta_1 \times m_{t-1} + (1 - \beta_1) \times g_t \]

\[ v_t \leftarrow \beta_2 \times v_t \]

\[ \hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t} \]

\[ \hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t} \]

\[ x_{t+1} \leftarrow x_t - \eta (\hat{v}_t)^{-\frac{1}{2}} \hat{m}_t \]

Tensorflow Code:
```
tf.train.RMSPropOptimizer(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08,)```

Keras Code:
```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)```
Batch Normalization [Ioffe, Szegedy]

For a layer of input vector \( x \):

**Normalize:**

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

\[
y(k) = \gamma(k) \hat{x}(k) + \beta(k)
\]

- BP needs to be modified to account for the change 反向传播过程需要修正
- Improves gradient flow through the network 可以改善网络的梯度流
- Allows higher learning rates 允许更高的学习速率
- Reduces the strong dependence on initialization 减少对初始化的依赖
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe 在某种程度上是正则化

Additional parameters to learn (thru BP)

**Tensorflow Code:**

\[
tf.nn.batch_normalization(x, mean, variance, offset, scale, variance_epsilon, name=None)
\]

**Keras Code:**

\[
k keras.layers.normalization.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True)
\]
训练神经网络 Training NN

• Dropout:
  • An effective way to prevent overfitting 可以有效防止过拟合
  • In each iteration, drop each node with probability $p$, and train the remaining network. 每次迭代中，以概率$p$丢掉每个节点，训练剩下的网络
  • In some sense, it has the effect of regularization 有正则化的效应
  • Make the training faster 可以使训练更快
  • Can be seen as an ensemble of many network structures (in a loose sense) 可以被视为很多个网络结构的系综

```python
keras.layers.core.Dropout(rate, noise_shape=None, seed=None)
```

rate: float between 0 and 1. Fraction of the input units to drop.
训练神经网络Training NN

• Step size schedule
  • First larger step size, then smaller step size 先用大步长，再用小步长

• Data Augmentation 数据增广
  • E.g., images – flip, rotate, shift the images, delete some (rows or col) pixels

• Lots Lots of other tricks [Book: Neural Networks: Tricks of the Trade]

• Fancier: Learning to learn
  • Using deep neural network to learning how to do gradient descent 用深度神经网络去学习如果做梯度下降

Learning to learn by gradient descent by gradient descent, NIPS16
Learning Gradient Descent: Better Generalization and Longer Horizons, ICML17
Open Source DL platforms

- Caffe
- Theano (compatible with python)
- TensorFlow (Google, support multi-GPU)
- Mxnet (https://github.com/dmlc/mxnet) (support multi-GPU)
- Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano
  - https://keras.io/
Tensorflow 简介
TensorFlow 教程

• 张量 Tensor

The central unit of data in TensorFlow is the tensor. A tensor consists of a set of primitive values shaped into an array of any number of dimensions. A tensor's rank is its number of dimensions. Here are some examples of tensors:

```
3 # a rank 0 tensor; this is a scalar with shape []
[1., 2., 3.] # a rank 1 tensor; this is a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

• Placeholder: a placeholder is a promise to provide a value later (can also be viewed as tensor)

```
a = tf.placeholder(tf.float32)
```
Tensorflow教程

• 计算图Computational Graph

You might think of TensorFlow Core programs as consisting of two discrete sections:

1. Building the computational graph.  
2. Running the computational graph.

A computational graph is a series of TensorFlow operations arranged into a graph of nodes. Let's build a simple computational graph. Each node takes zero or more tensors as inputs and produces a tensor as an output. One type of node is a constant. Like all TensorFlow constants, it takes no inputs, and it outputs a value it stores internally. We can create two floating point Tensors node1
and node2 as follows: 计算图的输入输出是tensor，中间节点是TensorFlow的操作。还有其他一类节点是常量。下面是两个常量节点。

```python
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0)  # also tf.float32 implicitly
print(node1, node2)
```
Session: something that encapsulates the control and state of the Tensorflow runtime. To actually evaluate the nodes, we must run the computational graph within a session. 我们需要在session中运行计算图

```python
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)

(<tf.Tensor 'Const:0' shape=() dtype=float32>, <tf.Tensor 'Const_1:0' shape=() dtype=float32>)
```

Notice that printing the nodes does not output the values 3.0 and 4.0 as you might expect. Instead, they are nodes that, when evaluated, would produce 3.0 and 4.0, respectively. And the following code creates a Session object and then invokes its run method to run enough of the computational graph to evaluate node1 and node2. By running the computational graph in a session as follows:

```python
sess = tf.Session()
print(sess.run([node1, node2]))
```

```
[3.0, 4.0]
```

只有在session中运行计算图，才会得到节点的具体的值
Tensorflow教程

• 更加复杂的计算 More complicated computations

```python
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0)  # also tf.float32 implicitly
node3 = tf.add(node1, node2)
print("node3: ", node3)
print("sess.run(node3): ", sess.run(node3))

('node3: ', <tf.Tensor 'Add:0' shape=() dtype=float32>)
('sess.run(node3): ', 7.0)
```
• 更加复杂的计算（加入placeholder）More complicated computations: Placeholder makes the computational graph accept external inputs possible

placeholder可以使计算图接受外来输入

```python
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)
```

The preceding three lines are a bit like a function or a lambda in which we define two input parameters (a and b) and then an operation on them. We can evaluate this graph with multiple inputs by using the feed_dict parameter to specify Tensors that provide concrete values to these placeholders:

```python
print(sess.run(adder_node, feed_dict={a: 3, b: 4.5}))
print(sess.run(adder_node, feed_dict={a: [1,3], b: [2, 4]}))
```

7.5
[ 3. 7.]
Tensorflow 教程

- 更加复杂的计算 More complicated computations

```python
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
add_node = a + b  # + provides a shortcut for tf.add(a, b)
add_and_triple = add_node * 3.
print(sess.run(add_and_triple, {a: 3, b: 4.5}))
```

22.5
Tensorflow教程

• 更加复杂的计算More complicated computations

In machine learning we will typically want a model that can take arbitrary inputs, such as the one above. To make the model trainable, we need to be able to modify the graph to get new outputs with the same input. **Variables** allow us to add trainable parameters to a graph. They are constructed with a type and initial value:

```python
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W * x + b
```

**Variable**变量，可以变化，一般**Variable**代表模型参数
Tensorflow教程 Tutorial for Tensorflow

- tf.constant: tensors whose value never change and they are initialized when you call tf.constant
- tf.Variable: tensors whose value can change and they are initialized when you call a special operation as follows:

```python
init = tf.global_variables_initializer()
sess.run(init)
```

It is important to realize `init` is a handle to the TensorFlow sub-graph that initializes all the global variables. Until we call `sess.run`, the variables are uninitialized.
Tensorflow 教程

• 结合前面，一个Tensorflow的计算线性函数的程序
Combine all the previous:

```python
code
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W * x + b
init = tf.global_variables_initializer()
sess.run(init)
print(sess.run(linear_model, {x:[1,2,3,4]}))
```

```
[ 0.  0.3  0.6  0.9]
```
Tensorflow教程

• 损失函数Loss function in Tensorflow

A loss function measures how far apart the current model is from the provided data. We'll use a standard loss model for linear regression, which sums the squares of the deltas between the current model and the provided data. `linear_model - y` creates a vector where each element is the corresponding example's error delta. We call `tf.square` to square that error. Then, we sum all the squared errors to create a single scalar that abstracts the error of all examples using `tf.reduce_sum`:

```python
y = tf.placeholder(tf.float32)
squared_deltas = tf.square(linear_model - y)
loss = tf.reduce_sum(squared_deltas)
print(sess.run(loss, {x:[1,2,3,4], y:[0,-1,-2,-3]}))
```

producing the loss value

23.66
Tensorflow教程

- Training: TensorFlow provides optimizers that slowly change each variable in order to minimize the loss function. The simplest optimizer is gradient descent. Tensorflow 提供已经实现好的优化方法，如 GD，SGD，等等

```python
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

```python
sess.run(init)  # reset values to incorrect defaults.
for i in range(1000):
    sess.run(train, {x:[1,2,3,4], y:[0,-1,-2,-3]})

print(sess.run([W, b]))
```

results in the final model parameters:

```
[array([-0.9999969], dtype=float32), array([ 0.99999082], dtype=float32)]
```
Tensorflow教程

• 完整的程序  A complete program

```python
import numpy as np
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)  # 模型参数用Variable
b = tf.Variable([-.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)          # 输入输出用placeholder
linear_model = W * x + b                # 在构建计算图阶段，目前输入输出并没有实际的值
y = tf.placeholder(tf.float32)

# loss
loss = tf.reduce_sum(tf.square(linear_model - y))  # sum of the squares  
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)  
train = optimizer.minimize(loss)
```
Tensorflow教程

```python
# training data
x_train = [1, 2, 3, 4]  
y_train = [0, -1, -2, -3]

# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)  # reset values to wrong
for i in range(1000):
    sess.run(train, {x:x_train, y:y_train})  # 实际计算图必须在session中运行

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x:x_train, y:y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

Results:

W: [-0.9999969] b: [ 0.99999082] loss: 5.69997e-11
Tensorflow for MLP (multi layer perceptron)
Tensorflow for MLP

```python
# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])

# Create model
def multilayer_perceptron(x, weights, biases):
    # Hidden layer with RELU activation
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)
    # Hidden layer with RELU activation
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.relu(layer_2)
    # Output layer with linear activation
    out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
    return out_layer
```

Create 2 placeholders
输入输出定义placeholder

$x \text{ weights['h1'] + biases['b1']}'$

$ReLU(\text{layer}_1)$

$layer_1 \text{ weights['h2'] + biases['b2']}'$

$ReLU(\text{layer}_2)$

$layer_2 \text{ weights['out'] + biases['out']}'$
Tensorflow for MLP

# Store layers weight & bias
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes]))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}

# Construct model
pred = multilayer_perceptron(x, weights, biases)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# Initializing the variables
init = tf.global_variables_initializer()
Tensorflow for MLP

```python
# Launch the graph
with tf.Session() as sess:
    sess.run(init)

# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop) and cost op (to get loss value)
        _, c = sess.run([optimizer, cost], feed_dict={x: batch_x, y: batch_y})
    # Compute average loss
    avg_cost += c / total_batch
```

Create a session
Initialize all the variables
Train `training_epochs` epochs
Set loss as 0
Compute the number of iterations in every epoch
Get training data
运营计算图，训练这个batch
Keras 简介
Keras for MLP

```python
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.summary()
```

配置模型来训练
```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

训练固定的epoch数
```
history = model.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs=epochs,
                     verbose=1,
                     validation_data=(x_test, y_test))
```

返回loss值和metric的值
```
score = model.evaluate(x_test, y_test, verbose=0)
```

严格来说输入维数784×batchsize，输出维数512×batchsize

全连接层，输入维数784，输出维数512

全连接层，输入维数512，输出维数512

全连接层，输入维数512，输出维数10

Loss是crosentropy
Many other loss functions:
mean_squared_error, mean_absolute_error, hinge, kullback_leibler_divergence, cosine_proximity, ...

优化方法是RMSprop
其他optimizer：SGD, Adagrad, adam, adadelta, ...

Verbosity mode. 0 = silent, 1 = verbose, 2 = one log line per epoch.
Thanks