Machine Learning

Liyao Tang

March 28, 2019

Contents

1 Math **1**
- 1.1 Convolution . . . 1
- 1.2 Linear Algebra . . . 1
  - 1.2.1 Essence . . . 1
  - 1.2.2 Interchanging Coordinates . . . 8
- 1.3 Calculus . . . 11
  - 1.3.1 Integral . . . 11
- 1.4 Probability Theory . . . 13
  - 1.4.1 Introduction . . . 13
  - 1.4.2 Expectations and Covariances . . . 14
  - 1.4.3 Transformations of Random Variables . . . 15
  - 1.4.4 Gaussian Distribution . . . 16
  - 1.4.5 Bayesian Interpretation of Probability . . . 19

2 Introduction **22**
- 2.1 General Concern . . . 22
  - 2.1.1 Types of Learning . . . 22
- 2.2 Decision Theory . . . 23
  - 2.2.1 . . . 23
- 2.3 Information Theory . . . 23
- 2.4 Recommended Practice . . . 23
  - 2.4.1 Data . . . 23
  - 2.4.2 Dataset . . . 24
  - 2.4.3 Orthogonalization Practice . . . 26
  - 2.4.4 Tunning Hyperparameters . . . 28
- 2.5 Model Analysis . . . 29
  - 2.5.1 Measurements of Problem . . . 29
  - 2.5.2 Improving Model . . . 32
  - 2.5.3 Evaluating Hypothesis . . . 35
  - 2.5.4 Skewed classes . . . 35
- 2.6 Supervised Learning . . . 36
- 2.7 Linear Regression . . . 36
- 2.8 Bayesian Regression . . . 37
- 2.9 Logistic Regression (Classification) . . . 39
- 2.10 Latent Variable Analysis . . . 43
  - 2.10.1 Principal Component Analysis (PCA) . . . 43
  - 2.10.2 Independent Component Analysis (ICA) . . . 46
  - 2.10.3 t-SNE . . . 46
  - 2.10.4 Anomaly Detection . . . 47
  - 2.10.5 Recommender System . . . 47
- 2.11 Large Scale Machine Learning . . . 49
  - 2.11.1 Online Learning . . . 49

2.11.2 Map-reduce 49

3 Linear Regression 50

4 Linear Classification 51

5 Kernel Methods 52

6 Graphical Models 53

7 Mixture Models and EM 54

8 Approximate Inference 55

9 Sampling Methods 56

10 Continuous Latent Variable 57

11 Sequential Data 58

12 Deep Learning 59
12.1 Interview of Fame 59
12.1.1 Geoffrey Hinton 59
12.1.2 Pieter Abbeel 60
12.1.3 Ian Goodfellow 61
12.1.4 Yoshua Bengio 61
12.1.5 Yuanqing Lin 62
12.1.6 Andrej Karpathy 62
12.1.7 Ruslan Salakhutdinov 63
12.1.8 63
12.1.9 Research 63
12.2 Basic Neutral Network 66
12.2.1 Advantages 66
12.2.2 Problem 67
12.2.3 Learning 68
12.3 Operations & Layers Structure 69
12.3.1 Operations in Network 69
12.3.2 Operations on Network 71
12.3.3 Cost 76
12.3.4 Layers 76
12.4 Architectures 84
12.4.1 Convolutional Networks 84
12.4.2 Recurrent Neural Network 85
12.4.3 Encoder-Decoder Architecture 86
12.4.4 Generative Network 87
12.5 Computer Vision 87
12.5.1 Objects Detection 87
12.5.2 Face Recognition 91
12.5.3 Image Style Transfer 92
12.5.4 Point Cloud Recognition 93
12.6 Natural Language Processing 93
12.6.1 Language Representation 93
12.6.2 Language Modeling 97
12.6.3 Name-Entity Recognition 97
12.6.4 Sentiment Classification 98

12.6.5 Neural Machine Translation 98
12.6.6 Speech Recognition 99
12.6.7 Machine Reading Comprehension 100
12.6.8 Image Caption 100
12.6.9 Referring Segmentation 101
12.7 Special Learning 105
12.7.1 Transfer Learning 105
12.7.2 Multi-task Learning 106
12.7.3 K-shot Learning 107

List of Figures

12.2 (using LSTM cell) 84

List of Tables

Chapter 1

Math

1.1 Convolution

- Definition
  - $f * g(z) = \int_{\mathbb{R}} f(x)g(z-x)dx$, where $f(x), g(x)$ are functions in $\mathbb{R}$
- Statistical Meaning
  - Notation
    - X,Y: independent random variables, with pdf's given by f and g
    - $Z = X + Y$, with pdf given by $h(z)$:
  - $\Rightarrow h(z) = f * g(z)$
    - derivation

$$
\begin{aligned}
H(z) = P(Z < z) &= P(X + Y < z) \\
&= \int_x P(X = x)P(X + Y < z | X = x)dx \\
&= \int_x f(x)P(Y < z - x)dx \\
&= \int_x f(x)G_Y(z - x)dx \\
\Rightarrow h(x) = \frac{d}{dz}H(z) &= \frac{d}{dz}\int_x f(x)G_Y(z - x)dx \\
&= \int_x f(x)\frac{dG_Y(z - x)}{dz}dx \\
&= \int_x f(x)g(z - x)dx \\
&= f * g(z)
\end{aligned}
$$

1.2 Linear Algebra

1.2.1 Essence

Vector

- Interpretation
  - Movement

        - direction
        - distance
    - Numeric in High Dimensions
        - in 1-D: +/- represents direction
        - in n-D: +/- alone each dimension combined to represent an overall direction (direction of the n-D numeric - vector)

- Numerics Multiplication
    - Scaling
        - the number scales the distance of vector (direction remains)
          $\Rightarrow$ such number thus also called scalar
        - $\Rightarrow$ scale alone each axis by that scalar
          $2\mathbf{x} = 2x_1e_2 + ... + 2x_ne_n$,
          where $e_1, ..., e_N$ are vector defining coordinates

- Linear Combination
    - Vector Adding: Generalization of Numerical Adding
        - in 1-D: joint movement along single axis
        - in n-D: joint movement along each axis $\Rightarrow$ a joint movement in n-D space
    - Definition: $\mathbf{x} = a_1\mathbf{x}_1 + ... + a_n\mathbf{x}_n$
        - the vectors $\mathbf{x}_1, ..., \mathbf{x}_n$ only altered linearly (as only being scaled)
          $\Rightarrow \mathbf{x}$ direction & size are linear combination of that in $\mathbf{x}_1, ..., \mathbf{x}_n$
    - Span of $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$
        - the n-D space S constructed by linear combination of $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$
    - x_0 linearly dependent on $\{x_1, ., , , x_n\}$
        - x_0 can be constructed by linear combination of $\{x_1, ., , , x_n\}$
          (already in the span space S)
        - $\Leftrightarrow$ function $a_0\mathbf{x}_0 + ... + a_n\mathbf{x}_n = 0$ has other solution than $a_0 = ... = a_n = 0$
    - x_0 linearly INdependent with $\{x_1, ., , , x_n\}$
        - x_0 can NOT be constructed by linear combination of $\{x_1, ., , , x_n\}$
          (not in the span space S, will increase the dimension of S if adopted)
        - $\Leftrightarrow$ function $a_0\mathbf{x}_0 + ... + a_n\mathbf{x}_n = 0$ has and only has solution $a_0 = ... = a_n = 0$

- Special Vectors
    - n-D Zero Vector $\mathbf{x}$
        -
    - Unit Vector
    - Basis of Vector Space S^n
        - general basis: a set of linearly independent vectors that span the space
          (i.e. a set of linearly independent n-D vectors)
        - unit basis: a general basis where every vector is unit vector
        - orthogonal basis: a general basis where all vectors are orthogonal with each other
        - unit orthogonal basis: a basis that is also a unit basis and an orthogonal basis

      $\Rightarrow$ coordinate: the scaler to composite a vector given a specific basis

Linear Transformation and Maps

- Linear Transformation
  - Transformation
    - a function mapping: vector → vector
    - a vector movement: scale & rotate all possible input vectors (i.e. a vector space)
  - Transformation with Linearity $L(\cdot)$
    - intuition: lines remain lines & origin remains origin
    - definition: a transformation $L(\cdot)$ is linear if
      1. additivity: $L(\mathbf{x}_1 + \mathbf{x}_2) = L(\mathbf{x}_1) + L(\mathbf{x}_2)$
      2. scaling: $L(a\mathbf{x}) = aL(\mathbf{x})$, where a is scaler
  - Features of $L(\cdot)$ given $\mathbf{x} = x_1e_1 + ... + x_ne_n$
    - same scaler for coordinates
      $\Rightarrow L(\mathbf{x}) = L(x_1e_1 + ... + x_ne_n)$
      $= L(x_1e_1) + ... + L(x_ne_n)$
      $= x_1L(e_1) + ... + x_nL(e_n)$
      $\Rightarrow$ transformed vector $\mathbf{x}' = L(\mathbf{x})$ has the same coord under the transformed basis
- Linear Map
  - Definition
    - map $F : V \to X$ is a linear map if it is a linear transformation, where V, X are vector spaces
- Multilinear Maps
  - Definition
    - map $F : \underbrace{V \times ... \times V}_{k \text{ copies}} \to X$ is multilinear/k-linear if it is linear in each slot
      i.e. $F(\mathbf{v}_1, ..., a\mathbf{v}_i + b\mathbf{v}_i', ..., \mathbf{v}_k) = aF(\mathbf{v}_1, ..., \mathbf{v}_i, ..., \mathbf{v}_k) + bF(\mathbf{v}_1, ..., b\mathbf{v}_i', ..., \mathbf{v}_k)$
      i.e. for fixed $\mathbf{v}_1, ..., \mathbf{v}_{i-1}, \mathbf{v}_{i+1}, ..., \mathbf{v}_k, F$ reduced to linear map with $\mathbf{v}_i$ as variable
      (where V, X are vector spaces)
  - Alternating Maps
    - map F is alternating if, its output is $\mathbf{0}$ whenever two vectors in inputs are identical
  - for Multilinear Map F: F Alternating $\Leftrightarrow F(..., \mathbf{v}, ..., \mathbf{w}, ...) = -F(..., \mathbf{w}, ..., \mathbf{v}, ...)$
    i.e. for multilinear map F, F alternating $\Leftrightarrow$ swapping two inputs flips sign of output
    - proof: given multilinear and alternating map F, for any $\mathbf{v}, \mathbf{w}$
      $0 = F(..., (\mathbf{v} + \mathbf{w}), ..., (\mathbf{v} + \mathbf{w}), ...)$
      $= F(..., \mathbf{v}, ..., \mathbf{v}, ...) + F(..., \mathbf{w}, ..., \mathbf{w}, ...) + F(..., \mathbf{v}, ..., \mathbf{w}, ...) + F(..., \mathbf{w}, ..., \mathbf{v}, ...)$
      $= F(..., \mathbf{v}, ..., \mathbf{w}, ...) + F(..., \mathbf{w}, ..., \mathbf{v}, ...)$
      $\Rightarrow F(..., \mathbf{v}, ..., \mathbf{w}, ...) = -F(..., \mathbf{w}, ..., \mathbf{v}, ...)$
    - proof: given multilinear map $F : F(..., \mathbf{v}, ..., \mathbf{w}, ...) = -F(..., \mathbf{w}, ..., \mathbf{v}, ...)$
      $\Rightarrow F(..., \mathbf{v}, ..., \mathbf{v}, ...) = -F(..., \mathbf{v}, ..., \mathbf{v}, ...)$
      $\Rightarrow F(..., \mathbf{v}, ..., \mathbf{v}, ...) = 0$, hence alternating

Matrix

- Matrix for Linear Transformation $L : S \to S'$
  - Representing Space Transformation
    - package the transformed basis under the original basis using matrix M
      i.e. represent the $e'_1, ..., e'_n = L(e_1), ..., L(e_n)$ under the original basis $e_1, ..., e_n$
      $\Rightarrow M = [e'_1, ..., e'_n]$, with all transformed basis as column vectors
      $\Rightarrow M$ represent the result of linear transformation for the basis of S
    - hence, determine a linear space transformation $L : S \to S'$ using $e_1, ..., e_n$
      for $M_{m \times n}$ matrix: a linear transformation from n-D space to m-D space
      1. $m < n$: projecting to subspace
      2. $m > n$: expanding into a hyper-plane/-line/etc (constrained in hyper-space)
  - Performing Space Transformation
    - $\forall i \in \{1, ..., n\}, x'_i = i^{\text{th}}$ component of $\mathbf{x}' = L(\mathbf{x})$, then $x'_i = (e'_{1i} + ... + e'_{n_i})x_i$
      (as proved above)
    - $\Rightarrow$ output vector $\mathbf{x}' = L(\mathbf{x}) = M\mathbf{x}$ under the original basis $e_1, ..., e_n$
      hence the rule for matrix multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = x\begin{bmatrix} a \\ c \end{bmatrix} + y\begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ax+by \\ cx+dy \end{bmatrix}$$

  where green and red columns the $\{e'_1, e'_2\}$ under $\{e_1, e_2\}$, $[x, y]$ the input vector $\mathbf{x}$

- Matrices for Composition of Linear Transformation
  - Transformation L_1, L_2 as Matrix M_1, M_2
    - as easy to prove $M_2 \cdot (M_1 \cdot \mathbf{x}) = (M_1 \cdot M_2) \cdot \mathbf{x}$
      $\Rightarrow L_2(L_1(\cdot)) \Leftrightarrow$ the composition of transformation defined by $M_2 \cdot M_1$
    - $\Rightarrow \mathbf{x}$ first transformed by L_1 then $L_2 \Leftrightarrow$ transformed by $M_2 \cdot M_1$
  - Linear Transformation on Space
    - given a basis matrix $E = [e_1, ..., e_n]$, a transformed basis $L_1(E) = L_1(e_1), ..., L_1(e_n)$
      ($e_1, ..., e_n$ as column vectors)
      $\Rightarrow L_2(L_1(e_k))$ performs L_2 transformation on the k^{th} vector of $L_1(E)$
    - hence to perform L_2 on the transformed space $L_1(E) \Rightarrow L_2(L_1(E))$
    - given that $L_1(E) = M_1 \Rightarrow L_2(L_1(E)) = M_2 \cdot M_1$
      (due to the derivation of linear transformation as matrix)
    - hence, $M_2 \cdot M_1$ denotes
      1. a further L_2 transformation on a transformed space $L_1(E)$
         (all result represented under the original basis E)
      2. the final transformed basis (first L_1 then L_2) under original basis E
         $\Rightarrow$ denotes a composite transformation of $L_2(L_1(\cdot))$
  - Multiplication between Matrices
    - $\Rightarrow$ composite linear transformation**s** into single linear transformation
    - $\Rightarrow$ linear transformation on vector**s**/space (generalized from single vector)
- Understanding Properties
  - $(AB)C = A(BC)$
    - both meaning apply transformation C then B then A...

$$\overbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}^{M_2}\overbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}^{M_1} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

where $M_1 = [L_1(e_1), L_1(e_2)], M_2 = [L_2(e_1), L_2(e_2)]$, the result $= [L_2(L_1(e_1)), L_2(L_1(e_2))]$
(all under basis $E = [e_1, e_2]$)

Dot Product

- Projecting to 1-D
  - Unit Orthogonal Basis
    - $\mathbf{x} \cdot \mathbf{y} = x_1y_1 + ... + x_ny_n = \mathbf{x}^T \cdot \mathbf{y} = \mathbf{y}^T \cdot \mathbf{x}$
      ($\mathbf{x}, \mathbf{y}$ assumed to be column vector / matrix with single column)
  - General Basis $E = [e_1, ..., e_n]$
    - $\Rightarrow \mathbf{x} = x_1e_1 + ... + x_ne_n = E\mathbf{x}, \mathbf{y} = y_1e_1 + ... + y_ne_n = E\mathbf{y}$
      $\Rightarrow \mathbf{x} \cdot \mathbf{y} = (E\mathbf{x}) \cdot (E\mathbf{y}) = \mathbf{x}^T E^T E\mathbf{y}$
    - understanding:
      1. $\mathbf{x} = E\mathbf{x}$: transfer back to a representation under unit orthogonal basis
      2. for $E = I$, back to the unit orthogonal case

  $\Rightarrow \mathbf{x} \cdot \mathbf{y}$: projecting $\mathbf{x}/\mathbf{y}$ to 1-D line using transformation $\mathbf{y}/\mathbf{x}$
  (direction/scaling effect of projection can be alter by choice of basis E though)

- Duality
  - Dual Vector
    - the vector represent a projection (linear transformation) to 1-D line
      (hence the vector equivalent to the matrix with 1 row defining the projection)
    - $\Rightarrow$ performing transformation on a vector $\Leftrightarrow$ taking product with the dual vector

Exterior (Wedge) Product

- Introduction
  - Definition
    - kth exterior product $\wedge^k V$ is a vector space, with a map $\psi : \underbrace{V \times ... \times V}_{k \text{ times}} \to \wedge^k V$
      note: map ψ called exterior multiplication, element $\psi(\mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k)$ called k-blade
  - Key Properties for Pair $(\wedge^k V, \psi)$
    - ψ is alternating multilinear map
    - for basis $\{e_1, ..., e_n\}$ of $V \in \mathbb{R}^n \Rightarrow \{e_{i_1} \wedge ... \wedge e_{i_k} | 1 \le i_1 \le i_k \le n\}$ a basis for $\wedge^k V$
      (due to its alternating multilinearity)
      $\Rightarrow e_{i_1} \wedge ... \wedge e_{i_k}$ can form any permutation of the same input by swapping order
      e.g. for $\wedge^2 V : e_1 \wedge e_1 = e_2 \wedge e_2 = \mathbf{0}, e_1 \wedge e_2 = -e_2 \wedge e_1 \Rightarrow$ not linearly independent
    - dim $\wedge^k V = \binom{n}{k} = C_n^k$ (due to the form of its basis),
      where $n = \dim V$
    - sum of k-wedge is still in the vector space $\wedge^k V$
    - any alternating multilinear map $F : \underbrace{V \times ... \times V}_{k} \to X$ factors uniquely into:
      $\underbrace{V \times ... \times V}_{k} \xrightarrow{\psi} \wedge^k V \xrightarrow{\underline{F}} X$, where ψ exterior multiplication, $\underline{F}$ a linear map

Determinant

- Signed Volume
-
- Measuring Linear Transformation on Volume
  - Unit Volume
    - unit volume $v = \|e_1 \bigwedge ... \bigwedge e_n\|$ with basis $e_1, ..., e_n$
      (for orthogonal basis, $v = \|e_1\| \times ... \times \|e_n\|$)
    - after transformation: $L(v) = \|L(e_1) \bigwedge ... \bigwedge L(e_n)\| = \|Me_1 \bigwedge ... \bigwedge Me_n\|$

    where $\bigwedge$ is the exterior product
  - Measuring Change of Unit Volume
    - $\Rightarrow \det(M) = \frac{L(v)}{v} = \|M_0 \times ... \times M_n\|$,
      assuming unit orthogonal basis $E = I$, where I is identity matrix
      (note: interpretation of $\det(\cdot) \leftrightarrow$ spacial interpretation of cross product $\times$)
    - hence, the rule of calculating $\det(\cdot)$

- Measuring Change of Orientation
  - Orientation of Space
    - jointly defined by the direction & order of the sequence of basis vector
      i.e. the positive/negative part of each axis in sequence
  - Measuring the Change
    - $\det(M) < 0$ if axises flipped over once (for an odd times)
      $\det(M) > 0$ if flipped for even times
    - interpretation: the flipped axis approaches 0 then expanded into the negative
      (measured by original basis E)

- Linear Dependency
  - $\det(M) = 0$
    - volume in current space becomes 0
      $\Rightarrow$ dimensions decreases after transformation applied
      $\Rightarrow$ i.e. transformed basis not able to span the current space
    - $\Rightarrow$ basis in M NOT linearly INdependent! $\Leftrightarrow \det(M) = 0$
  - $\det(M) \neq 0$
    - volume still exist
      $\Rightarrow$ dimensions remain & transformed basis still span the space
    - $\Rightarrow$ basis in M is linearly INdependent $\Leftrightarrow \det(M) \neq 0$

- Understanding Properties
  - $\det(M_1 M_2) = \det(M_1) \det(M_2)$:
    - left: final volume & orientation changed after transformation M_2 then M_1
    - right: the volume scaled by one transformation, then further by the other;
      (similar for orientation, as measured by sign)

Cross Product

- Determinant
  - $\|\mathbf{v} \times \mathbf{w}\| = |\det([\mathbf{v}, \mathbf{w}])|$
    - as determinant measuring the change of unit basis
      $\Rightarrow$ constructing matrix with target vector $\mathbf{v}, \mathbf{w}$ as column
    - $\Rightarrow$ matrix $[\mathbf{v}, \mathbf{w}]$ as the transformation altering the space
  - Direction of $\mathbf{v} \times \mathbf{w}$
    - expanding in the perpendicular direction w.r.t the hyper-plane defined by $\mathbf{v}, \mathbf{w}$ (obeying the "right hand rule")
- Linear Transformation
  -

Rank

- Measuring Linear Transformation on Dimension
  - Measuring the Transformed Space
    - rank of $M = r$: basis in M span a r-dimension (column) space
      note: the column space of M: transformed space defined by column basis
    - full rank: the dimension of column space is as high as possible
- Null Space (Kernel) of M
  - Vectors in Null Space
    - $\forall \mathbf{x}, M\mathbf{x} = \mathbf{0}$
      (i.e. all the vectors in null space transformed into $\mathbf{0}$ by M, hence the name)
    - $\mathbf{0}$ always in null space
  - Dimension of Null Space
    - D(null space) = num of column in M − rank of M
      note: as focusing on column space $\Rightarrow$ num of cols = dimension of input vectors
- Understanding Properties
  - $R(M_{m\times n}) \leq \min\{m, n\}$
    - $m < n$: projecting a n-D vector to a m-D subspace, hence at most of rank m
    - $m > n$: M consists of n vectors of m dimensions, define at most n-D space
  - $R(M_{m\times n}) = \min\{m, n\} \Leftrightarrow M$ Full Rank
    - from definition, it is as high as possible
    - m vectors with n dimensions, span at most a $\min\{m, n\}$ space
      either linearly dependent ($m < n$), or not enough independent vectors ($m > n$)

Inverse of Matrix

- Inverse of Linear Transformation
  - Interpretation
    - a transformation L^{-1} to inverse the effect of another transformation L
      $\Rightarrow \forall \mathbf{x}, L^{-1}L(\mathbf{x}) = \mathbf{x}$
    - $\Rightarrow M^{-1}M = I$, where M for L, M^{-1} for L^{-1}, I the identity matrix

- Requirement
  - transformed basis in M still span the space!
    otherwise, transformed vectors projected onto subspace $\Rightarrow$ information lost!
  - hence, following equivalent requirements:
    1. transformed basis in M linearly INdependent
    2. M is square and full rank
    3. $\det(M) \neq 0$

Linear System of Equations

- Linearity
  - Linear Combination of Variables
    - coefficients matrix A: holding the coefficient for each equation (in row)
    - variables vector $\mathbf{x}$: holding variables as column vector
    - constants vector $\mathbf{v}$: holding target constant for each equations as column vector

    $\Rightarrow A\mathbf{x} = \mathbf{v}$ for a set of linear equations
  - Linear Transformation Perspective
    - $\mathbf{x}/\mathbf{v}$ as original/transformed vectors
    - columns of A (coefficients for the same variable) as transformed basis

    $\Rightarrow$ finding a start position $\mathbf{x}$ which, after transformation A, lands on $\mathbf{v}$
- Solutions
  - Transformed Basis Linearly INdependent ($\det(A) \neq 0$)
    - hence, $A\mathbf{x} = \mathbf{v} \Leftrightarrow \mathbf{x} = A^{-1}\mathbf{v}$
      (use the inverse transformation A^{-1} to find the input $\mathbf{x}$ using output $\mathbf{v}$)

    $\Rightarrow$ single unique $\mathbf{x}$ found
  - Transformed Basis Linearly Dependent ($\det(A) = 0$)
    - information lost ($\mathbf{x}$ projected to subspace) $\Rightarrow$
      1. multiple solutions: basis in A linearly dependent with $\mathbf{v}$
         $\Rightarrow$ i.e. $\mathbf{v}$ in the column space of A
         (special case where $\mathbf{v} = \mathbf{0}$: solution space = null space of A)
      2. no solution: basis in A linearly INdependent with $\mathbf{v}$
         $\Rightarrow$ i.e. can NOT possibly be described by basis in A

1.2.2 Interchanging Coordinates

N-dimensional Spherical Coordinates

- Notation
  - N-dim Euclidean Space E_N
    - $e_1, e_2, ..., e_N$: a group of orthonormal basis of E_N
    - $\mathbf{x} = (x_1, x_2, ..., x_N)$: vector in E_n
    - $\mathbf{x}_{i-N}$: projection of $\mathbf{x}$ onto subspace spanned by $e_i, ..., e_N$

      $\Rightarrow \mathbf{x}_{i-N} = \sum_{n=i}^{N} x_n e_n$
  - Spherical Coordinates

- $r = \|\mathbf{x}\|$: the norm of $\mathbf{x}$
- $\phi_i \in [0, \pi]$: angle between $\mathbf{x}_{i-N}$ and e_i
- $r_i = \|\mathbf{x}_{i-N}\|$: norm of projection $\mathbf{x}_{i-N}$, with $r_1 = r$

- Observation
  - Space $e_1, ..., e_N$:
    - $\cos\phi_1 = \frac{\mathbf{x}e_1}{\|\mathbf{x}\|\|e_1\|} = \frac{x_1}{r}$
      $\Rightarrow x_1 = r\cos\phi_1$
      $\Rightarrow \mathbf{x} = r\cos\phi_1 e_1 + \sum_{n=2}^{N} x_n e_n$
  - Space $e_2, ..., e_N$:
    - from above: $\mathbf{x}^2 = r^2\cos^2\phi_1 + \sum_{n=2}^{N} x_n^2 = r^2$
      $\Rightarrow \sum_{n=2}^{N} x_n^2 = r^2\sin^2\phi_1$
    - $\mathbf{x}_{2-N} = \sum_{n=2}^{N} x_n e_n$
      $$\Rightarrow \begin{cases} \|\mathbf{x}_{2-N}\|^2 = \sum_{n=2}^{N} x_n^2 = r^2\sin^2\phi_1 = r_2^2 \\ \cos\phi_2 = \frac{\mathbf{x}_{2-N}\cdot e_2}{\|\mathbf{x}_{2-N}\|\|e_2\|} = \frac{x_2}{r_2} \end{cases}$$
      $$\Rightarrow \begin{cases} r_2 = r\sin\phi_1 & (\text{ as } \phi_1 \in [0,\pi]) \\ x_2 = r_2\cos\phi_2 = r\sin\phi_1\cos\phi_2 \end{cases}$$
      $\Rightarrow \mathbf{x}_{2-N} = r\sin\phi_1\cos\phi_2 e_2 + \sum_{n=3}^{N} x_n e_n$
  - Space $e_3, ..., e_N$:
    - from above: $\mathbf{x}_{2-N}^2 = r^2\sin^2\phi_1\cos^2\phi_2 + \sum_{n=3}^{N} x_n^2 = r^2\sin^2\phi_1$
      $\Rightarrow \sum_{n=3}^{N} x_n^2 = r^2\sin^2\phi_1\sin^2\phi_2$
    - $\mathbf{x}_{3-N} = \sum_{n=3}^{N}$
      $$\Rightarrow \begin{cases} \|\mathbf{x}_{3-N}\|^2 = \sum_{n=3}^{N} x_n^2 = r^2\sin^2\phi_1\sin^2\phi_2 = r_3^2 \\ \cos\phi_3 = \frac{\mathbf{x}_{3-N}\cdot e_3}{\|\mathbf{x}_{3-N}\|\|e_3\|} = \frac{x_3}{r_3} \end{cases}$$
      $$\Rightarrow \begin{cases} r_3 = r\sin\phi_1\sin\phi_2 & (\text{ as } \phi_1, \phi_2 \in [0,\pi]) \\ x_3 = r_3\cos\phi_3 \end{cases}$$
      $\Rightarrow \mathbf{x}_{3-N} = r\sin\phi_1\sin\phi_2 e_3 + \sum_{n=4}^{N} x_n e_n$

- Proof for x_i
  - Procedure

- $\mathbf{x}_{i-N} = \sum_{n=i}^{N} x_n e_n$
  $\Rightarrow \cos\phi_i = \frac{\mathbf{x}_{i-N} \cdot e_i}{\|\mathbf{x}_{i-N}\|\|e_i\|} = \frac{x_i}{r_i}$
  $\Rightarrow x_i = r_i \cos\phi_i$

- Induction
  - Goal
    - $\forall i \geq 2, r_i = r\prod_{j=1}^{i-1} \sin\phi_j$
  - Base Case ($i = 2$)
    - as in observation, $r_2 = r\sin\phi_1 = r\prod_{j=1}^{2-1} \sin\phi_j$
  - Step Case
    - assumption $r_i = r\prod_{j=1}^{i-1} \sin\phi_j$
    - procedure: $\mathbf{x}_{i-N} = \sum_{n=i}^{N} x_n e_n = r_i\cos\phi_i + \sum_{n=i+1}^{N} x_n e_n$

    $\Rightarrow \|x_{i-N}\|^2 = r_i^2\cos^2\phi_i + \sum_{n=i+1}^{N} x_n^2 = r_i^2$

    $\Rightarrow \|x_{i+1-N}\|^2 = \sum_{n=i+1}^{N} x_n^2 = r_i^2\sin^2\phi = r_{i+1}^2$

    $\Rightarrow r_{i+1} = r_i\sin\phi_i = r\prod_{j=1}^{i} \sin\phi_j$

- Derivation
  - x_i from Combined Proofs
    - $x_i = \begin{cases} r\cos\phi_1 & i = 1 \\ r\cos\phi_i\prod_{j=1}^{i-1}\sin\phi_j & 2 \leq i \leq N \end{cases}$
  - Last 2 Dimensions
    - $\mathbf{x}_{(N-1)-N} = x_{N-1}\cdot e_{N-1} + x_N \cdot e_N$, with $r_{N-1} = r\prod_{j=1}^{N-2}\sin\phi_j$

    $\Rightarrow \|\mathbf{x}_{(N-1)-N}\| = f(\phi_{N-1}, \phi_N) = r_{N-1}$
    $\Rightarrow \phi_{N-1}, \phi_N$ not independent**!**
    (actually, if $e_N = e_{N-1} + \frac{\pi}{2}$, then $\phi_N = \phi_{N-1} - \frac{\pi}{2}$)

    $\Rightarrow$ define $\theta \in [0, 2\pi)$ instead of $\phi_{N-1}, \phi_N \in [0, \pi]$
    $\Rightarrow x_{N-1} = r_{N-1}\sin\theta, x_N = r_{N-1}\cos\theta$ (interchangeable)
  - Final Spherical Coordinates

- $x_i = \begin{cases} r\cos\phi_1 & i = 1 \\ r\cos\phi_i \prod_{j=1}^{i-1} \sin\phi_j & 2 \le i \le N-1 \\ r\sin\theta \prod_{j=1}^{N-2} \sin\phi_j & i = N-1 \\ r\cos\theta \prod_{j=1}^{N-2} \sin\phi_j & i = N \end{cases}$

1.3 Calculus

1.3.1 Integral

Interchanging Coordinates in Integral

- General Theory
  - Notation
    - (x, y): coordinate under Field D
    - (u, v): coordinate under Field D'
    - $T : \begin{cases} x = x(u, v), \\ y = y(u, v) \end{cases}$: transformation from D to D'
  - Assumption
    - $f(x, y)$ continuous in D
    - transformation T's partial 1^{st} order derivatives continuous on D'
    - transformation T's Jacobian $J(u, v) = \frac{\partial(x,y)}{\partial(u,v)} \neq 0$
    - transformation $T : D \to D'$ is 1-1 mapping
  - Derivation
    - take infinitely small square in D' :
      $M_4'(u, v + \delta v)$, $M_3'(u + \delta u, v + \delta v)$,
      $M_1'(u, v)$, $M_2'(u + \delta u, v)$

      $\Rightarrow$ after transformation to D :
      $M_4(x(u, v + \delta v), y(u, v + \delta v))$, $M_3(x(u + \delta u, v + \delta v), y(u + \delta u, v + \delta v))$,
      $M_1(x(u, v), y(u, v))$, $M_2(x(u + \delta u, v), y(u + \delta u, v))$

      $\Rightarrow x_2 - x_1 = x(u + \delta u, v) - x(u, v) = \frac{\partial x}{\partial u}|_{(u,v)}\delta u$

      $x_4 - x_1 = x(u, v + \delta v) - x(u, v) = \frac{\partial x}{\partial v}|_{(u,v)}\delta v$

      $y_2 - y_1 = y(u + \delta u, v) - y(u, v) = \frac{\partial y}{\partial u}|_{(u,v)}\delta u$

      $y_4 - y_1 = y(u, v + \delta v) - y(u, v) = \frac{\partial y}{\partial v}|_{(u,v)}\delta v$

      as $\delta u, \delta v \to 0$, curvilinear boundary quadrilateral $M_1M_2M_3M_4 \to$ parallelogram

      $\Rightarrow S_{M_1M_2M_3M_4} = |\overrightarrow{M_1M_2} \times \overrightarrow{M_1M_4}| = |\begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_4 - x_1 & y_4 - y_1 \end{vmatrix}|$

      $= |\begin{vmatrix} \frac{\partial x}{\partial u}\delta u & \frac{\partial y}{\partial u}\delta u \\ \frac{\partial x}{\partial v}\delta v & \frac{\partial y}{\partial v}\delta v \end{vmatrix}| = |\begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} \end{vmatrix}|\delta u\delta v$

      $= |J(u, v)|\delta u\delta v$

- $\Rightarrow$ infinitely small area $\delta\sigma = dxdy = |J(u,v)|\delta u\delta v$
  $\Rightarrow \int\int_D f(x,y)dxdy = \int\int_{D'} f(x(u,v),y(u,v))|J(u,v)|dudv$

- Integral in Cartesian $\rightarrow$ Polar
  - Result
    - $dxdy = rdrd\theta$
  - Derivation
    - from general transformation: $x = r\cos(\theta), y = r\sin(\theta)$
      $\Rightarrow dxdy = |J(r,\theta)|drd\theta = rdrd\theta$
    - from direct calculation of infinite small area in polar coordinate
      $\Rightarrow d\sigma = \frac{1}{2}(r+dr)^2d\theta - \frac{1}{2}r^2d\theta = rdrd\theta + \frac{1}{2}(dr)^2d\theta$
      $\Rightarrow d\sigma = rdrd\theta$, when $dr, d\theta \rightarrow 0$

Gaussian Integral

- Gaussian Function
  - $f(x) = \mathrm{e}^{-\mathrm{a(x+b)^2}}$
    - special form: $f(x) = \mathrm{e}^{-(\mathrm{x})^2}$
    - alternative form: $f(x) = \mathrm{e}^{\mathrm{ax^2+bx+c}}$
    - no indefinite integral $\int_a^b \mathrm{e}^{-\mathrm{x}^2}$
    - only definite integral $\int_{-\infty}^{+\infty} \mathrm{e}^{-\mathrm{x}^2}$

- Direct Integral
  - $$\begin{aligned}(\int_{-\infty}^{+\infty} \mathrm{e}^{-\mathrm{a(x+b)^2}}\mathrm{dx})^2 &= \int_{-\infty}^{+\infty} \mathrm{e}^{-\mathrm{a(x+b)^2}}\mathrm{dx}\int_{-\infty}^{+\infty} \mathrm{e}^{-\mathrm{a(y+b)^2}}\mathrm{dy} \\ &= \int\int_{-\infty}^{+\infty} \mathrm{e}^{-\mathrm{a[(x+b)^2+(y+b)^2]}}\mathrm{d(x+b)d(y+b)} = \int\int_{-\infty}^{+\infty} \mathrm{e}^{-\mathrm{a(x^2+y^2)}}\mathrm{dxdy} \\ &= \int_0^{2\pi}\int_0^{+\infty} \mathrm{e}^{-\mathrm{ar}^2}\mathrm{rdrd}\theta \\ &= \frac{\pi}{a}\end{aligned}$$
    $$\Rightarrow \int_{-\infty}^{+\infty} \mathrm{e}^{-\mathrm{a(x+b)^2}}\mathrm{dx} = \sqrt{\frac{\pi}{\mathrm{a}}},\ \text{alternatively}\ \int_{-\infty}^{+\infty} \mathrm{e}^{\mathrm{ax^2+bx+c}}\mathrm{dx} = \sqrt{\frac{\pi}{-\mathrm{a}}}\cdot \mathrm{e}^{\frac{\mathrm{b}^2}{4\mathrm{a}}+\mathrm{c}}$$

- Even Moment of Gaussian Function
  - $$\begin{aligned}\int_{-\infty}^{+\infty} x^{2n}\mathrm{e}^{-\mathrm{ax}^2}\mathrm{dx} &= (-1)^n\int_{-\infty}^{+\infty}\frac{\partial^n}{\partial a^n}\mathrm{e}^{-\mathrm{ax}^2}\mathrm{dx} \\ &= (-1)^n\frac{\partial^n}{\partial a^n}\int_{-\infty}^{+\infty}\mathrm{e}^{-\mathrm{ax}^2}\mathrm{dx} && \text{by parameter differeation} \\ &= (-1)^n\sqrt{\pi}\frac{\partial^n}{\partial a^n}a^{-\frac{1}{2}} \\ &= \sqrt{\frac{\pi}{a}}\frac{(2n-1)!!}{(2a)^n}, && \text{where !! is double factorial}\end{aligned}$$

1.4 Probability Theory

1.4.1 Introduction

Background

- Measuring Uncertainty
  - Source of Uncertainty
    - noise in reality & observation
    - finite size of data (limited information)
- Derivation
  - Quantifying Belief
    - by Cox (1946): if numerical values used to represent degrees of belief, a simple set of axioms encoding common sense properties of such beliefs will lead uniquely to a set of rules for manipulating degrees of belief that are equivalent to the sum and product rules of probability
  - Measuring Uncertainty
    - by Jaynes (2003): probability theory can be regarded as an extension of Boolean logic to situations involving uncertainty
  - Common Destination
    - numerical quantities to measure uncertainty, derived from different sets of properties/axioms, behave precisely according to the rules of probability

The Basic

- Notation
  - X, Y: random variable
- Discrete
  - $P(X, Y)$: joint probability of X, Y taking their values
  - $P(X)$: marginal probability of X taking its value
  - $P(X|Y)$: conditional probability of X taking its value given Y observed / determined
- Continuous
  - $P(x) = P_X(x)$: cumulative probability of value for variable $X < x$
  - $p(x)$: probability density,
    - where $\lim_{\delta x \to 0} P(X \in (x, x + \delta x)) = \lim_{\delta x \to 0} p(x)\delta x \Rightarrow P(X \in (a, b)) = \int_a^b p(x)dx$
    - $\Rightarrow p(x) \geq 0$ and $\int_{-\infty}^{+\infty} p(x) = 1$
    - $\Rightarrow P(z) = \int_{-\infty}^{z} p(x)dx$
- Basic Rules
  - Sum Rule
    - $P(X) = \sum_Y P(X, Y)$, where X, Y are discrete

        - $P(X) = \int_Y P(X, Y)$, where X, Y are continuous
          (formal justification requires measure theory)
    - Product Rule
        - $P(X, Y) = P(Y|X)P(X)$
        - $P(X, Y) = P(Y)P(X)$, where X, Y are independent
    - $\Rightarrow$ Bayes' Rule
        - $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{P(X|Y)P(Y)}{\sum_Y P(X|Y)P(Y)}$, where Y are discrete
        - $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{P(X|Y)P(Y)}{\int_Y P(X|Y)P(Y)}$, where Y are continuous

- Interpretation of Bayes
    - Normalization
        - the $\sum$, $\int$ can be interpreted as a **normalization constant**
          $\Rightarrow$ **posterior** $\propto$ **likelihood** $\times$ **prior**
    - Prior
        - $P(Y)$: available probability of desired variable **before** anything observed
          $\Rightarrow$ Y usually model parameters
    - Posterior
        - $P(Y|X)$: obtained probability of desired variable **after** observation
          $\Rightarrow$ if X, Y independent, observation has no effect $\Rightarrow$ prior = posterior
    - Likelihood
        - $P(X|Y)$: how probable/likely of X being observed under different setting of Y
    - Prior $\rightarrow$ Posterior
        - a process of incorporating the evidence provided by observation

1.4.2 Expectations and Covariances

Expectation

- Definition
    - Expectation of $f(x)$ under $p(x)$
        - discrete x: $\mathbb{E}_p[f] = \sum_x p(x)f(x)$
        - continuous x: $\mathbb{E}_p[f] = \int p(x)f(x)dx$
        - approximation with N points drawn from $p(x)$: $\mathbb{E}_p[f] \simeq \frac{1}{N}\sum_{n=1}^{N} f(x_n)$
          (when $N \rightarrow \infty$, $\simeq$ becomes $=$)
    - Multivariate Expectation
        - Marginal Expectation of $f(x, y)$ on x: $\mathbb{E}_x[f(x, y)] = \sum_x p(x)f(x, y)$
          (hence a function of y)
        - Conditional Expectation $f(x)$ on $p(x|y)$: $\mathbb{E}[f|y] = \sum_x p(x|y)f(x)$

- Independence

- Independent x, y
  - $\mathbb{E}_{xy}[x, y] = \sum_{x,y} p(x, y)xy = \sum_{x,y} p(x)p(y)xy = \mathbb{E}[x]\mathbb{E}[y]$

Variance

- Definition
  - Variance of $f(x)$:
    - $\text{var}[f] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$
      $= \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2$
  - Covariance

Covariance

- Definition
  - between Variables x, y
    - $\text{cov}[x, y] = \mathbb{E}_{x,y}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])]$
      $= \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y]$
  - between Vectors $\mathbf{x}, \mathbf{y}$ (column vectors)
    - $\text{cov}[\mathbf{x}, \mathbf{y}] = \mathbb{E}_{\mathbf{x},\mathbf{y}}[(\mathbf{x} - \mathbb{E}[\mathbf{x}]) \cdot (\mathbf{y}^T - \mathbb{E}[\mathbf{y}^T])]$
      $= \mathbb{E}_{\mathbf{x},\mathbf{y}}[\mathbf{x}\mathbf{y}^T] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}^T]$
      (pairwise covariance between components of $\mathbf{x}, \mathbf{y}$)
  - within Vector $\mathbf{x}$
    - $\text{cov}[\mathbf{x}] \equiv \text{conv}[\mathbf{x}, \mathbf{x}]$
      (pairwise covariance between its components)
- Independence Variable
  - Independent x, y
    - $\text{cov}[x, y] = \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y] = 0$

1.4.3 Transformations of Random Variables

Inverse Image

- Definition
  - Notation
    - function $g : \mathbb{R} \to \mathbb{R}$
    - set A in $\mathbb{R}$
  - Inverse Image on Set A
    - $g^{-1}(A) = \{x \in \mathbb{R} | g(x) \in A\}$
      $\Leftrightarrow x \in g^{-1}(A)$ if and only if $g(x) \in A$
      interpretation: for each element in A, get its original value before g applied
- Properties
  - $g^{-1}(\mathbb{R}) = \mathbb{R}$, as g is defined on $\mathbb{R}$
  - $\forall$ set $A, g^{-1}(A^c) = g^{-1}(A)^c$, where A^c is the complement of set A

- $\forall$ collection of sets $\{A_\lambda | \lambda \in \Lambda\}, g^{-1}\left(\bigcup_\lambda A_\lambda\right) = \bigcup_\lambda g^{-1}(A_\lambda)$
- General Transformation $Y = g(X)$
  - $P(Y \in A) = P(g(X) \in A) = P(X \in g^{-1}(A))$

Discrete Variable

- Variable
  - X: random variable with probability mass function $P_X(x)$
  - $Y = g(X)$, with probability mass function $P_Y(y)$
- Probability Mass Function
  - $P_Y(y) = \sum_{x \in g^{-1}(y)} P_X(x)$, as $X = x$ is independent and mutually exclusive
    note: $g^{-1}(y)$ denotes $g^{-1}(\{y\})$
  - Example
    - uniform random variable X on $\{-n, ..., n\}$ with $Y = |X|$
      $\Rightarrow P_X(x) = \frac{1}{2n+1}$
      $\Rightarrow P_Y(y) = \begin{cases} \frac{1}{2n+1}, & x = 0, \\ \frac{2}{2n+1}, & x \neq 0. \end{cases}$

Continuous

- Variable
  - X: random variable with cumulative distribution $P_X(x)$, density $p_X(x)$
  - $Y = g(X)$, with cumulative distribution $P_Y(y)$, density $p_Y(y)$
- Cumulative Distribution
  - Strictly Monotone Increasing g
    - $P_Y(y) = P(Y \leq y) = P(g(X) \leq y) = P(X \leq g^{-1}(y)) = P_X(g^{-1}(y))$
  - Strictly Monotone Decreasing g
    - $P_Y(y) = P(Y \leq y) = P(g(X) \leq y) = P(X \geq g^{-1}(y)) = 1 - P_X(g^{-1}(y))$
- Probability Density
  - Strictly Monotone g (an one-to-one function)
    - $p_Y(y) = \frac{d}{dy} P_Y(y) = \frac{dP_Y(y)}{dg^{-1}(y)} \frac{dg^{-1}(y)}{dy} = p_X(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y) \right|$,
      as g^{-1} has the same monotony as g, combined with the sign in P_Y to give the $|\cdot|$

1.4.4 Gaussian Distribution

Definition

- Univariate Gaussian
  - Variable
    - mean: μ
    - variance: $\sigma^2 \Rightarrow$ reciprocal of the variance $\beta = \frac{1}{\sigma^2}$ (also called precision)

- Probability Dense Function (PDF)
  - $\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$
    $\Rightarrow$ satisfying probability axioms: $\mathcal{N}(x|\mu,\sigma^2) > 0$ and $\int_{-\infty}^{+\infty} \mathcal{N}(x|\mu,\sigma^2)dx = 1$
- Expectation
  - $\mathbb{E}[x] = \int_{-\infty}^{+\infty} \mathcal{N}(x|\mu,\sigma^2)xdx = \mu$

$$\begin{aligned}
\Rightarrow \mathbb{E}[x^2] &= \int_{-\infty}^{+\infty} \mathcal{N}(x|\mu,\sigma^2)x^2dx \\
&= \frac{1}{(2\pi\sigma^2)^{1/2}} \int_{-\infty}^{+\infty} x^2 \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} dx \\
&= \pi^{-\frac{1}{2}} \int_{-\infty}^{+\infty} (\sqrt{2\sigma^2}x+\mu)^2 \exp(-x^2)dx, \text{ substituting } a = \frac{x-\mu}{\sqrt{2\sigma^2}} \\
&= \pi^{-\frac{1}{2}}(2\sigma^2 \int_R x^2 \mathrm{e}^{-\mathrm{x}^2}\mathrm{dx} + 2\mu\sqrt{2\sigma^2} \int_R \mathrm{x}\mathrm{e}^{-\mathrm{x}^2}\mathrm{dx} + \mu^2 \int_R \mathrm{e}^{-\mathrm{x}^2}\mathrm{dx}) \\
&= \pi^{-\frac{1}{2}}(2\sigma^2 \int_R x^2 \mathrm{e}^{-\mathrm{x}^2}\mathrm{dx} + 2\mu\sqrt{2\sigma^2} \cdot 0 + \mu^2\sqrt{\pi}) \\
&= 2\sigma^2\pi^{-\frac{1}{2}} \int_R x^2 \mathrm{e}^{-\mathrm{x}^2}\mathrm{dx} + \mu^2 \\
&= \sigma^2 + \mu^2, \text{ by 2nd moment of Guassian or } (x\mathrm{e}^{-\mathrm{x}^2})' = \mathrm{e}^{-\mathrm{x}^2} - 2\mathrm{x}^2\mathrm{e}^{-\mathrm{x}^2}
\end{aligned}$$

- Variance
  - $\mathrm{var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \sigma^2$

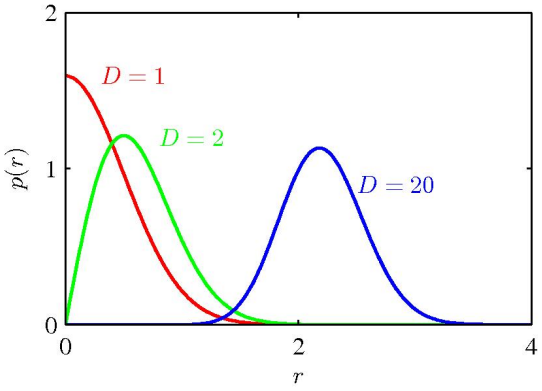
- Multivariate (d-dimensional) Gaussian
  - Variable
    - mean: $\mu \in \mathbb{R}^d$
    - covariance matrix: $\Sigma_{d\times d}$
  - Probability Dense Function (PDF)
    - $\mathcal{N}_d(\mathbf{x}|\mu,\mathbf{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$,
      noted as $X \sim \mathcal{N}_d(x|\mu,\Sigma)$

Multivariate Gaussian

- Dimensionality
  - Volume of High Dimensional Sphere
    - for $n = 2k, k \in N^+, V_{2k}(R) = \frac{\pi^k}{k!}R^{2k}$
    - for $n = 2k+1, k \in N, V_{2k}(R) = \frac{2^{k+1}\pi^k}{(2k+1)!!}R^{2k+1}$
    - $\Rightarrow \lim_{D\to+\infty} \frac{V_D(1) - V_D(1-\epsilon)}{V_D(1)} = \lim_{D\to+\infty} 1 - 1(1-\epsilon)^D = 1 \Rightarrow$ the volume of a D-sphere concentrate in a thin shell near the surface!
      (actually, in the corner of a high dimensional cube as shown below)
  - Volume of High Dimensional Cube
    -

- ⇒ volume ratio of hyper sphere and hyper cube: ⇒ the volume of a D-cube concentrates in its corner! ⇒ distance function in high dimensional space CAN be useless

- High Dimensional Distribution
- High Dimensional Gaussian
  - probability density with respect to radius r for various dimension D
    ⇒ most density are in a thin shell at a specific r



- Facing High Dimensionality

- Convolution of Gaussian
  - Integral of Gaussians $\int G_1 G_2 dx$
    - $G_1 \sim \mathcal{N}_d(x|a, A), G_2 \sim \mathcal{N}_d(x|b, B)$

$$\Rightarrow \int \mathcal{N}_d(x|a,A)N_d(x|b,B)dx$$
$$= \int \frac{1}{(2\pi)^{d/2}|A|^{1/2}} e^{-\frac{1}{2}(x-a)^T A^{-1}(x-a)} \frac{1}{(2\pi)^{d/2}|B|^{1/2}} e^{-\frac{1}{2}(x-b)^T B^{-1}(x-b)} dx$$
$$= \int \frac{1}{(2\pi)^{d/2}|A|^{1/2}} \frac{1}{(2\pi)^{d/2}|B|^{1/2}} e^{-\frac{1}{2}[(x-a)^T A^{-1}(x-a)+(x-b)^T B^{-1}(x-b)]}$$
$$= \int \frac{1}{(2\pi)^{d/2}|A|^{1/2}} \frac{1}{(2\pi)^{d/2}|B|^{1/2}} e^{-\frac{1}{2}[(x-c)^T (A^{-1}+B^{-1})(x-c)+(a-b)^T C(a-b)]},$$
where $c = (A^{-1}+B^{-1})^{-1}(A^{-1}a + B^{-1}b), C = A^{-1}(A^{-1}+B^{-1})^{-1}B^{-1} = (A+B)^{-1}$
$$= \frac{\left|(A^{-1}+B^{-1})^{-1}\right|^{1/2}}{(2\pi)^{d/2}|A|^{1/2}|B|^{1/2}} e^{-\frac{1}{2}(a-b)^T C(a-b)} \int \frac{1}{(2\pi)^{d/2}\left|(A^{-1}+B^{-1})^{-1}\right|^{1/2}} e^{-\frac{1}{2}(x-c)^T (A^{-1}+B^{-1})(x-c)} dx$$
$$= \frac{\left|(A^{-1}+B^{-1})^{-1}\right|^{1/2}}{(2\pi)^{d/2}|A|^{1/2}|B|^{1/2}} e^{-\frac{1}{2}(a-b)^T C(a-b)}$$
$$= \frac{1}{(2\pi)^{d/2}(|A||B||A^{-1}+B^{-1}|)^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)}$$
$$= \frac{1}{(2\pi)^{d/2}|ABA^{-1}+ABB^{-1}|^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)}$$
$$= \frac{1}{(2\pi)^{d/2}|ABA^{-1}+A|} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)}$$
$$= \frac{1}{(2\pi)^{d/2}|A(B+A)A^{-1}|} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)}$$
$$= \frac{1}{(2\pi)^{d/2}|A+B|^{1/2}} e^{-\frac{1}{2}(a-b)^T (A+B)^{-1}(a-b)}$$

- $\Rightarrow$ Convolution of Gaussians $G1 * G2$
  - $G_1 \sim \mathcal{N}_d(a,A), G_2 \sim \mathcal{N}_d(b,B)$

$$G_1 * G_2(z) = \int G_1(x)G_2(z-x)dx$$
$$= \int \mathcal{N}_d(x|a,A)\mathcal{N}_d(z-x|b,B)dx$$
$$= \int \mathcal{N}_d(x|a,A) \cdot \frac{1}{(2\pi)^{d/2}|B|^{1/2}} e^{-\frac{1}{2}(z-x-b)^T B^{-1}(z-x-b)} dx$$
$$= \int \mathcal{N}_d(x|a,A)\mathcal{N}_d(x|z-b,B)dx$$
$$= \frac{1}{(2\pi)^{d/2}|A+B|^{1/2}} e^{-\frac{1}{2}(z-(a+b))^T (A+B)^{-1}(z-(a+b))}$$
$$= \boldsymbol{\mathcal{N}_d(z|a+b, A+B)}$$

1.4.5 Bayesian Interpretation of Probability

Contrasting Frequentist Estimator

- Posterior $p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$
  - Notation
    - $\mathcal{D}$ the observed dataset
    - $\mathbf{w}$ the vector for model parameters
  - Bayesian

        - only one single dataset $\mathcal{D}$ (the observed one)
        - uncertainty expressed as distribution over $\mathbf{w}$
        - model's error: use likelihood / posterior directly (or after taking log)
        - pros
            1. naturally incorporating prior knowledge as prior distribution (of $\mathbf{w}$)
        - cons
            1. prior usually selected for mathematic convenience
    - Frequentist Estimator
        - parameters $\mathbf{w}$ already determined / fixed by 'estimator' (model)
        - error bars of the model obtained by considering the distribution over $\mathcal{D}$
        - model's error: bootstrap procedure
            1. generate dataset(s) by drawing from the observed $\mathcal{D}$ with replacement
            2. sampling L datasets with the same size as $\mathcal{D}$
            3. error = variability of predictions between the sampled datasets
        - pros
            1. protect the conclusion from false prior knowledge
        - cons
            1. sensitive to observation (extreme cases), especially under small dataset

Parameter Estimation

- Bias vs. Variance
- Taking Logarithm
    - Reason
        - monotonically increasing function $\Rightarrow \arg\max_\theta f(x;\theta) = \arg\max_\theta \log f(x;\theta)$
        - simplify mathematical analysis $\Rightarrow \prod \to \sum$
        - numerical stability $\Rightarrow$ avoid $\prod$(small probabilities)
          (may otherwise underflow the numerical precision)
- Maximum Likelihood Estimation for Gaussian
    - Notation
        - $X = \{x_1, ..., x_N\}$: observed N data points
    - Assumption
        - data points are i.i.d. (identically and independently distributed)
        - underlying distribution is Gaussian $\mathcal{N}(\mu, \sigma)$
    - Likelihood
        - $p(X|\mu,\sigma^2) = \prod_{n=1}^{N} (x_n|\mu,\sigma^2)$
        - $\Rightarrow \ln p(X|\mu,\sigma) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi)$
    - Solution
        - let $\frac{\partial}{\partial\mu}\ln p(X|\mu,\sigma^2) = 0 \Rightarrow \mu_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} x_n$
        - let $\frac{\partial}{\partial\sigma^2}\ln p(X|\mu,\sigma^2) = 0 \Rightarrow \sigma^2_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu_{\mathrm{ML}})$

- Analysis
  - $\mathbb{E}[\mu_{\mathrm{ML}}] = \mathbb{E}[\frac{1}{N}\sum_{n=1}^{N} x_n] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[x_n] = \mu$
    (as $x_1, ..., x_N$ i.i.d, drawn from $\mathcal{N}(\mu, \sigma^2)$, thus $\sim \mathcal{N}(\mu, \sigma^2)$)
    $\Rightarrow$ unbiased estimation of mean
  - $\mathbb{E}[\sigma^2_{\mathrm{ML}}] = \mathbb{E}[\frac{1}{N}\sum_{n=1}^{N}(x_n - \mu_{\mathrm{ML}})^2]$

$$= \mathbb{E}[\frac{1}{N}\sum_{n=1}^{N}(x_n^2 - 2\mu_{\mathrm{ML}}x_n + \mu^2_{\mathrm{ML}})]$$

$$= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[x_n^2] - \mathbb{E}[\frac{1}{N}\sum_{n=1}^{N}2x_n\mu_{\mathrm{ML}}] + \mathbb{E}[\mu^2_{\mathrm{ML}}]$$

$$= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[x_n^2] - 2\mathbb{E}[\mu^2_{\mathrm{ML}}] + \mathbb{E}[\mu^2_{\mathrm{ML}}]$$

$$= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[x_n^2] - \frac{1}{N^2}\sum_{i,j=1}^{N}\mathbb{E}[x_i x_j]$$

$$= \frac{1}{N}\sum_{n=1}^{N}(\sigma^2 + \mu^2) - \frac{1}{N^2}[N(N-1)\mu^2 + N(\sigma^2 + \mu^2)]$$

(by 2nd moment of Gaussian $\mathbb{E}[x^2]$ and i.i.d assumption)

$$= \left(\frac{N-1}{N}\right)\sigma^2$$

$\Rightarrow$ biased variance **!**

$\Rightarrow$ unbiased variance $\hat{\sigma}^2 = \frac{N}{N-1}\sigma^2_{\mathrm{ML}} = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \mu_{\mathrm{ML}})^2$

interpretation: $N-1$ degree of freedom,
(as calculating σ^2 needs μ, which help pin down x_N given $x_1, ..., x_{N-1}$)

Predictive Distribution

- Probabilistic Prediction
  - Notation
    - $\mathbf{x}, \mathbf{t}$: vector of data examples and corresponding ground truth
    - $\mathbf{w}$: model parameters
    - x, t: new data example for prediction and its ground truth
  - Prediction by Model
    - $p(t|x, \mathbf{w}')$, where $\mathbf{w}'$ is the best fit parameters founded
  - Prediction by Data
    - $p(t|x, \mathbf{x}, \mathbf{t}) = \int p(t|x, \mathbf{w})p(\mathbf{w}|\mathbf{x}, \mathbf{t})d\mathbf{w}$, where $\mathbf{w}$ marginalized over its posterior

Chapter 2

Introduction

2.1 General Concern

2.1.1 Types of Learning

Supervised Learning

- Overview
  - training data comprises examples of input vectors with corresponding target vectors
- Regression
  - output one or more continuous variable
- Classification
  - assign input to one of a finite number of discrete categories

Unsupervised Learning

- Overview
  - training data consists of a set of input vectors without target vectors
- Clustering
  - Goal: discover groups of similar examples
- Density Estimation
  - Goal: determine the distribution of data within the input space
- Dimension Reduction
  - Goal: project data into low dimension, for the purpose of such as visualization

Reinforcement Learning

- Overview
  - input with time series & discover optimal output by a process of trial and error
- Goal
  - find actions to take under given circumstance to maximize a reward

2.2 Decision Theory

2.2.1

2.3 Information Theory

2.4 Recommended Practice

2.4.1 Data

Data Augmentation

- Artificial Data Synthesis
  - Practice
    - easily prepare a large amount of similar (yet, different) data
    - add canonical noise to the similar data
  - Understanding
    - convert similar data to the desired distribution
  - Caution
    - collected canonical noise may only represent a subset of all possible noise
      ⇒ may overfit to those collected noise
      (e.g. distortion on image from game for car detection: too less unique cars)
  - Examples
    - in the task of classifying image uploaded by users
      ⇒ collect web image & blur the image
    - in the task of in-car NLP interaction
      ⇒ collect well-recorded sentence audio & add in-car noise
- Distortion
  - Practice in Computer Vision
    - mirroring horizontally/vertically, rotation, shirring, etc.
    - random crop a reasonably large subset of image
    - color shifting: e.g. PCA color augmentation
  -

Data Preprocessing

- Mean Centering
  - Practice
    - for all training examples, compute mean (on each features) $\mu = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$,
      where $\{\mathbf{x}_1, ..., \mathbf{x}_N\} = X_{\text{train}}$ the training set
    - preprocess each $\mathbf{x} \in X_{\text{train}}, X_{\text{val}}, X_{\text{test}}$ to be $\mathbf{x}' = \mathbf{x} - \mu$ (all data go through the same process)
  - Pros
    - (training) data has a zero mean (statistically, most data close to 0)
  - Cons

        - different features may reside in various scales

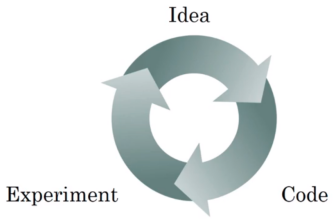
- Standardizing
    - Practice
        - compute mean μ, standard deviation $\sigma = \left(\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \mu)^2\right)^{1/2}$, where $\{\mathbf{x}_1, ..., \mathbf{x}_N\} = X_{\text{train}}$ the training set
        - preprocess each $\mathbf{x} \in X_{\text{train}}, X_{\text{val}}, X_{\text{test}}$ to be $\mathbf{x}' = \frac{\mathbf{x}-\mu}{\sigma}$ (all data go through the same process)
        - note: with big data, usually computed iteratively due to limited memory
    - Pros
        - (training) data has zero mean & unit variance ⇒ approximated to normal distribution
        - for deep learning: different features in same small range close to 0 ⇒ weights for different features are in roughly the same scale ⇒ easier to train

- Cleaning Incorrect Label
    - Practice
        - before cleaning: measure its contribution to the error rate & its cause
        - random error (e.g. occasional mistake, etc.) with a big dataset: okay to ignore
        - systematic error: should be corrected, at least for val&test set ⇒ to evaluate the model on the target data distribution
        - if mislabeled data cause inability to evaluate&compare model: must be cleaned
        - val&test set should be cleaned together ⇒ to have the same distribution
    - Understanding
        - in train set: statistic model (deep net etc.) quite robust to random errors while model can learn the systematic error ⇒ not able to generalize
        - in val set: random error can cause inability ⇒

2.4.2 Dataset

Train-Val-Test

- Reason
    - Iterative Process



        - intuition usually do NOT transfer across domains (NLP, CV, Search, etc.)
        - do NOT hope to have the correct hyperparameters at the first try

⇒ need feedback from experiment result
⇒ make sure the feedback is CORRECT and FAST

- Recommended Usage
  - Splitting
    - classic split for small dataset ⇒ train:val:test = 60 : 20 : 20, or K-fold
    - in big data (e.g. 100 million) ⇒ train:val:test = 98 : 1 : 1

    (as long as val-test sets cover enough data variance)
  - Training Set
    - to find the model parameter estimation (used for learning process of model)
      ⇒ over-fit by complex model
    - can incorporate methods to train the model to have desired property
      (where augment data goes)
  - Validation Set (Val)
    - to indicate generalization ability of a range of trained models on target data
      (correct if enough various input covered)
      ⇒ for model comparison, selection & hyperparameters tunning
    - should have consistent distribution with test set
      (as val set is also evaluating the generalization ability)
  - Test Set
    - to evaluate the **generalization ability** of final model on target data
      (correct if enough various input covered)
    - should represent the distribution of target data
      i.e. data that the deployed model will need to handle
  - Training-Validation Set (Train-Val)
    - another val set split from original training set
    - used if training set are from different distribution then the val/test set
      (e.g. due to augmented data etc.)
    - performance gap between train&val set: variance + distribution mismatch
      ⇒ separate each measurement
    - performance gap between train&train-val set: measuring variance
      ⇒ performance gap between train-val&val set: measuring distribution mismatch
- Potential Problem
  - Mismatched Distribution across Sets
    - classic supervised learning assumption: all sets drawn from SAME distribution
      (yet transfer/adaptive learning focus on violation of such assumption)
    - measured by train-val set
    - should ensure at least that val&test set have the SAME distribution

    ⇒ yet, make sure val&test set from the SAME distribution as the desired one
  - **Overfitting Val Set**
    - iteratively tunning model is a processing of learning (fitting to the val set)
      ⇒ with enough iteration, val set can be overfit
    - may consider test set as 2^{nd} val set, and further have $3^{\text{rd}}, 4^{\text{th}}$... val sets
  - Limited Data
    - better model ⇒ more training data
    - ⇒ less validation ⇒ noisy estimation of generalization ability

Train-Test

- No Val Set
  - Practice
    - may use the "test" set as val set ⇒ generalization ability NOT reported
    - should be confident in that dataset cover/represent true distribution of data (yet, not recommended)
  - Understanding
    - to utilize as many data as possible for ultimate performance
- K-fold Cross Validation
  - Procedure
    - split all data into K folds, $K-1$ folds for train, 1 for validation
    - ⇒ average over all C_K^1 combination to indicate the generalization ability
    - extreme case: leave-out-one ⇒ $K = N$, where N is number of all data
  - Cons:
    - $\mathcal{O}(K)$ ⇒ slow, especially if training process already slow
      ⇒ trade off between time vs. constraint on validation
    - hence, **not** often used in big data era

2.4.3 Orthogonalization Practice

Definition

- Decoupling Goals and Models
  - Designing Metrics
    - to evaluate the models
      ⇒ capture how well the problem solved as desired
    - decouple different aspect of concern into different metrics
  - Designing Models
    - to do well on the previously chosen metrics
      (including training & tunning hyperparameters)
- Decoupling Hyperparameters
  - disjoint set of hyperparameters to optimize for train-val-test set
  - hyperparameter taking effect on single goal
    at least, NOT to impose negatively related effect on multiple goals
    (e.g. early stopping on performances on train and val sets ⇒ NOT preferred)
  - ⇒ clearer control on model behaviors

Practice

- Designing Metrics (Goals)
  - Single Metric Reporting Overall Performance
    - a metric accounting for multiple metrics
      e.g. F1 score instead of precision and recall
    - weighted average over metrics
      (capturing tendency by different weights)

    - Satisficing Metrics
        - optimizing single metric with some minimum requirement must being satisfied
          ⇒ single optimizing metric + several satisficing metrics
          ⇒ optimizing under constraints
          e.g. optimizing accuracy with false positive rate < 0.2 satisfied

- Tunning Hyperparameters (Designing Model)
    - Inherent Separation for Set-level Goals
        - fit model on train set for good fitting
          ⇒ tune model/network structure, optimization, preprocessing, etc.
        - evaluate on val set for good generalization ability
          ⇒ tune regularization, etc.
        - evaluate on test set for hopefully good generalization ability reported
          ⇒ consider bigger val set (if an overfit val set indicated)
        - apply in real world hopping model to generalize well indeed
          ⇒ consider mismatched data distribution, redesign cost function / metrics etc. (if failed to generalize)

      note: size of dataset can be hyperparameter sometimes
    - Separating Tunning for Performance Metrics
        -

Orthogonalization

- Definition
    - Decoupling Tunning for Different Goals
        - disjoint set of hyperparameters to optimize on train-val-test set
        - hyperparameter taking effect on single goal
          (at least, NOT to impose negatively related effect on multiple goals)
    - Single Metric Reporting Performance
        - a metric accounting for multiple metrics
          e.g. F1 score instead of precision and recall
        - optimizing under constraints (must-satisfied metrics)
          e.g. optimizing accuracy with false positive rate < 0.2 satisfied

- Practice
    - Separating Tuning for Set-level Goals
        - for train set: model/network structure, optimization, preprocessing, etc.
        - for val set: regularization, etc.
        - for test set: bigger val set (as indicating an overfit val set)
        - for real world: mismatched cost function / data distribution, etc.
    - Separating Tunning for Performance Metrics
        -

- Understanding
    - Inherently Separated Goals
        - fit model on train set: adjust model for good fitting

        - evaluate on val set: adjust model for good result on metrics (indicating generalization ability)
        - evaluate on test set: hope to report good generalization ability
        - apply in real world: hope to generalize well indeed
    - Clear Control on Behaviors
        - form iterative process among various goals
        - prevent tunning practice with unaware negatively coupled effect on different goals (e.g. early stopping on performances on train and val sets ⇒ NOT preferred)

2.4.4 Tunning Hyperparameters

especially for deep learning

Hyperparameters

- Overview
    - Structures and Architectures
        - type of layers and size of layers
        - type of activation
        - depth of networks
    - Learning
        - learning rate
        - optimizer (learning process)
    - Robustness and Generalizability
        - regularization(s)
        - data preprocessing/augmentation
- Challenges
    - NO Consistent Prescience
        - popular choices from one domain usually NOT carry over to other domains
    - NOT Predictable Effect
        - hyperparameter does NOT have predictable effect on specific model behavior ⇒ need a search for the best one

Systematic Searching

- Random Sampling
    - Reason
        - NOT able to know the importance of different hyperparameters ⇒ not wasting grid search step on the unimportant
        - NOT able to know the effective range of a hyperparameters ⇒ may be skipped by grid search step
        - decouple the search for different hyperparameters ⇒ more richly explore (whereas grid search fix one while searching on others)
    - Coarse to Fine Scheme
        - explore whole space uniformly (equally random)
        - exploit region where good results show up (with more densely sampled)

    - Sampling on Scale
        - instead of sampling the value of hyperparameter, sample the scale of it
          e.g. sampling learning rate $\alpha = 10^r, r \sim U(-4, 0)$
        - $\Rightarrow$ distribute the density across desired scale**s**
          (by using transfered scale, e.g. applying $\log, e^x$ e.t.c)
        - reason: depends on the
            - use of hyperparameter e.g. in an exponential/linear/log way
            - whether intend to sample on scale e.g. across one or more scales

- Swarm Optimization
    - Intuition
        - searching over a space with continuous$\times$discrete across various scale
          $\Rightarrow$ encoded into a list
        - search using permutation / group behavior
          $\Rightarrow$ inherently imposing explore-exploit strategy
    - Popular Framework
        - genetic algorithm (GA)
        - particle swarm optimization (PSO)

Tunning Practice

- Single Model
    - Practice
        - supervising one model at a time
        - interactively justify the hyperparameter in training process
          $\Rightarrow$ gain knowledge through interaction & ensure a good performance
          $\Rightarrow$ early feedback
    - Reason
        - too many data (online advertisement, computer vision etc.)
        - few computing resource
- Parallel Training
    - Practice
        - shoot out multiple model with various settings
        - compare at the end (after trained & evaluated)
    - Reason
        - small data/model, enough computability / fast training process

2.5 Model Analysis

2.5.1 Measurements of Problem

Performance Metrics

- Intersection over Union (IoU)
    - Input
        - two regions, from prediction and label
          e.g. bounding box, segmentation mask

- Definition
  - I = intersection of the regions
  - U = union of the regions
  - $\Rightarrow$ IoU = $\frac{I}{U}$
- Use Case
  - evaluation of object detection/segmentation
  - post-processing in object detection/segmentation
    (e.g. non-max suppress)
- Understanding
  - measure how well two regions overlap
    $\Rightarrow$ usually IoU > 0.5 considered a decent match
  - average IoU over each prediction class & scale of threshold for an overview report
    (e.g. average over all obstacle types, threshold range [0.05 0.95] with step 0.05)
- Extension
  - intersection over label/prediction region $\Rightarrow$ in case of unstable label/prediction region
    (while matching relation considered important in the task)

- Bilingual Evaluation Understudy (BLEU)
  - Input
    - one sequence as prediction; other sequence(s) as references
      (can be more than one) i.e. label
  - Definition
    - n-gram g_n: n continuous tokens
    - count of g_n: the number of appearance of g_n in a sequence
      $\Rightarrow$ num of $g_n = \sum_{g_n}$(count of g_n) $= l - n + 1$, in sequence of len l
    - ref count: $\sum_{g_n \in \text{pred}}$ (1 if it appears in any reference; else 0)
      $\Rightarrow$ count of g_n in prediction that also appears in a reference
      $\Rightarrow$ $g_n \in$ pred matched as long as $g_n \in$ a reference
    - $\Rightarrow$ precision $= \dfrac{\text{true positive}}{\text{positive pred}} = \dfrac{\text{ref count}}{\text{num of } g_n \in \text{pred}}$
      (yet, "the the the the" has high score when $n = 1$, as "the" almost always appear)
    - clipped ref count: min{ count of g_n in pred , max(num of g_n in a reference)}
      $\Rightarrow$ count only unique n-gram in pred
      $\Rightarrow$ $g_n \in$ pred matched up to the max num of $g_n \in$ reference
    - $\Rightarrow$ clipped precision $p_n = \dfrac{\text{clipped true positive}}{\text{positive pred}} = \dfrac{\text{clipped ref count}}{\text{num of } g_n \in \text{pred}}$
    - $\Rightarrow$ BLUE score $= BP \exp\left(\dfrac{1}{N}\sum_{n=1}^{N} p_n\right)$,
      where $BP = 1$ if $\text{len}_{\text{pred}} > \text{len}_{\text{ref}}$; else $\exp(1 - \text{len}_{\text{pred}}/\text{len}_{\text{ref}})$
      $\Rightarrow$ the brevity penalty to penalize too short pred
      (as short pred has larger chance to have all its gram contained in ref)
  - Use Case
    - machine translation
    - image caption

- Understanding
  - clipped ref count: has a maximal number for g_n appearance, given the reference
    $\Rightarrow$ same g_n exceeding the number becomes false positive, as can NOT be matched
    (analogy: metrics in obj detection with bounding box)
  - evaluate the appearance of generated tokens (prediction) in references (label)
  - highly correlated with human evaluation
  - bias towards statistical model (when compared against rule-based model)

Expected Generalization Ability Measurement

- Bayes Optimal Performance
  - Measurement
    - the theoretically best performance on all data
      $\Rightarrow$ modeling only the indent mapping without the noise (a perfect model)
      (note: in practice, only approximated bayes performance available)
  - Understanding
    - measure the inherent noise in data as the best possible performance on all data
    - $\Rightarrow$ measure **avoidable** bias: indicate the upper-bound performance
    - $\Rightarrow$ measure degree of **overfitting**: indicate how much the model fit to the noise
- Human Performance: Approximation to Bayes Performance
  - Definition
    - for best approximation: best achievable performance by human
      (e.g. group of experts, as bayes optimal performance is even better)
    - for specific focus: depends on use case
      e.g. for self-diagnose model, may define as the performance of a normal doctor
  - Practice
    - usually done in supervised learning
      note: the label (i.e. 0-error) is NOT bayes performance
    - on unstructured data, human almost achieves bayes performance
      (as human good at natural perception task, like cv, nlp)
  - Cons
    - hard to distinguish surpassing human performance from overfitting training set
- (Avoidable) Bias
  - Measurement
    - gap between train set performance and bayes performance
      (note: in practice, only approximated bayes performance available)
  - Understanding
    - measure the model capacity of handling given (train) data
      as (approximately) measuring the gap between the theoretically best model
- Variance
  - Measurement
    - performance gap between val set & train set (if under same distribution)
    - if different distribution for train&val set $\Rightarrow$ train-val set instead of val set
  - Understanding

        - measure the generalization ability:
          as measuring how much model can cope with unseen data
          ⇒ model the indent mapping, instead of the noise

- Mismatch Distribution
  - Measurement
    - performance gap between train-val set & val set
  - Understanding
    - train-val set contains the unseen train data; val set the unseen target data
      ⇒ gap only caused by different distribution between sets

2. Interaction with regularization:
- Improper λ: - large λ =¿ high bias - small λ =¿ high variance - Choosing λ: - try $\lambda = 0, 0.01, 0.02, 0.04, ..., 10$ - select the model with lowest $J_{cv}(\theta)$ without regularization term
3. Interaction with training set size:
- Normal Learning curve:
![Normal learning curve](../../Machine
- Learning curve with high bias:
- where getting more training data **doesn't** help
![Learning curve with high bias](../../Machine
- Learning curve with high variance:
- where getting more training data **helps**
![Learning curve with high variance](../../Machine
4. Ways to fix:
- High bias: - more features / more polunomial terms of features - decreasing λ
- High variance:
- larger data setOrtho - fewer features - increasing λ
- **In neural network:**
- High bias =¿ larger neural networks (more hidden layers / more units in one layer)
- High variance =¿ smaller neural networks
Larger network with regularization (λ) is more powerful

2.5.2 Improving Model

Bias-Variance Guideline

- Solving High (Avoidable) Bias
  - Increasing Model Capability
    - increase complexity: more weights, latent variable / hidden layer etc.
    - use more suitable model specifically designed for the data (e.g. CNN for image)

  ⇒ until fitting training set well

- Solving High Variance
  - Data Augment
    - get/simulate more training data (via crowd sourcing, distortion, GANs, etc.)
  - Model Regularization
    - control the complexity of model (e.g. L0/1/2 normalization)

- Solving Trade-off
  - Iterative Process

        - solve bias, then solve variance, iteratively
    - Complexity + Data/Regularization
        - increase complexity to solve bias
          without hurting variance (via more data/regularization)
        - more data/regularization to solve variance
          without hurting bias (with enough complexity)

Behavior Detail

- Low Bias, High Variance (Over-fitting)
    - Symptom
        - good performance on train set & poor generalization (bad on val)
        - ⇒ good at fitting train set; bad at representing/modeling underlying data source
    - Cause
        - too much representation ability (to fit even the noise)
        - directly model the likelihood instead of posterior
    - Remedy
        - larger dataset
        - regularization (model the posterior by accounting prior)
- High Bias, Low Variance (Under-fitting)
    - Symptom
        - bad at fitting training examples & modeling underlying data source
          (bad at train & val)
        - ⇒ poor performance on train set & good generalization (though meaningless)
    - Cause
        - lack of representation ability (not enough flexibility)
    - Remedy
        - try model with better representative ability (more complexity, flexibility)
- High Bias, High Variance (Over&Under-fitting)
    - Symptom
        - bad at fitting some general cases; while good at some rare and special cases
          (especially in high dimensional space)
          ⇒ fitting largely noise
    - Cause
        - model probably not suitable for the dataset
    - Remedy
        - switch to other types of model
        - dataset preprocessing
- Low Variance, High Mismatch
    - Symptom
        - good at generalizing (on train-val); bad at target data (on val)

    - Cause
        - model not able to generalize across mismatch distribution
          (yet generalized well in the same distribution: as good at train-val)
    - Remedy
        - transform train data towards (more like) target data
          e.g. data synthesis: adding noise that is special in target data, etc.
        - ensure train set contains enough / assign larger weight to, the desired target data
        - transfer learning, adaptive learning, etc.

- Low Bias, Low Variance, Low Mismatch, High val-test Variance
    - Symptom
        - model with specific hyperparameter overfitting the val set
          ⇒ val set NO longer reveal model generalizability
    - Cause
        - val set overfit by iteration of hyperparameter tunning (as a practice of fitting)
    - Remedy
        - more data for val&test set
        - re-design/choose the model after val set overfitting fixed
          (after true generalizability reported)

- Low Bias, Low Variance
    - Behavior
        - good at fitting training examples & modeling underlying data source
          (good at train & val)
        - ⇒ good performance on training set & good generalization

- High Bias, Low Variance, Lower/Negative Mismatch
    - Behavior
        - perform better on val& test set then on train set
          ⇒ target data distribution easier than train set distribution
          ⇒ able to do well on desired data, even if not good on train set
          (better convinced by measuring human performance on both distribution)

Error Analysis

- Categorizing Error Source
    - Practice
        - create histogram on val set reflecting data categories
          (e.g. for image: blurry, rotated, incorrect label, etc...)
          ⇒ categorize data first
        - ⇒ data leading to error scattered into different categories
          ⇒ evaluate the contribution to error from different categories
        - note:
    - Understanding
        - find out the most important error source
          ⇒ prioritize the direction of tunning model

- Ceiling Analysis
    - Definition & Practice
        -
    - Understanding

Approaches Analysis

- End-to-End Approach
  - Definition
    - use single network to learn the mapping from input directly to desired output (no intermediate result)
  - Pros
    - reveal the data statics: avoid any specific prior
    - large & auto feature extraction
  - Cons
    - need enough data for effective end-to-end model
    - hard to inject effective prior into model
      ⇒ exclude potentially hand-designed component/knowledge
  - Understanding
    - end-to-end model works only when enough data to reveal the problem complexity

1. Aim:
- Decide which modules might be the best use of time to improve
2. Procedure:
- Draw a table with 2 column (Component - Accuracy)
- Component: the modules simulated to be perfect (100- Accuracy: the accuracy of the entire system on the test set (define by chosen evaluation matrix)
- — **Perfect Component** — **Accuracy** — — :————————: — :————————: — — none — f — — module 1 — $f + \epsilon_1$ — — module $1, 2$ — $f + \epsilon_1 + \epsilon_2$ — — · · · — · · — module $1, 2, ..., n$ — $f + \epsilon_1 + ... + \epsilon_n = 100\%$ —
- =¿ Improving module x will gain at most ϵ_x improvement in the overall performance
- Choose the module with most significant ϵ to improve

1. Procedure: - Algorithm (trained) misclassifies n data in cross validation set - Classify these n data and rank them - Maybe more features are found 2. Feature selection =¿ Numerical evaluation - =¿ test algorithm with / without this feature on **CV set** (compare error rate)

2.5.3 Evaluating Hypothesis

4. Choosing procedure:
- Minimize traning error $J_{train}(\theta)$ - Select a model with lowest $J_{cv}(\theta)$ - Estimate generalization error as $J_{test}(\theta)$

2.5.4 Skewed classes

1. Precision / Recall
- — — **Actual 1** — **Actual 0** — — ————— — ————— — ————— — — **Predicted 1** — True positive — False positive — — **Predicted 0** — False negative — True negative —
- **Precision** $= \dfrac{\text{True positive}}{\text{Predicted positive}} = \dfrac{\text{True positive}}{\text{True pos + False pos}}$
- **Recall** $= \dfrac{\text{True positive}}{\text{Actual positive}} = \dfrac{\text{True positive}}{\text{True pos + False neg}}$

2. Evaluation with precision/recall

- Predict 1 if $h_\theta(x) \geq \epsilon$, 0 if $h_\theta(x) < \epsilon$

- larger ϵ =¿ higher precision, lower recall (more confident) - smaller ϵ =¿ lower pecision, higher recall (avoid missing)

![Posiible Precision -Recall curev](../../Machine Learning/Statistical 0Machine Learning/Posiible Precision -Recall curev.png)

3. Compare precision/recall num

- $\mathrm{F}_1 Score = 2\frac{PR}{P+R}$, P as precision, R as recall - higher better, on cross validation set

4. High precision & high recall:

- **large num of features (low bias) + large sets of data (low variance)**

2.6 Supervised Learning

- Feature normalization: $\forall x_{ij} \in X, x_{ij} = \frac{x_{ij}-\mu_j}{\sigma_j^2}$, $X : [instance][feature]$, without $[1...1]^T$ in 1st column $X = [x_1, x_2, ..., x_m]$, m instances in total
- Regularization: add penalty for θ being large into cost function
- $J(\theta) = ... + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$, **bias θ_0 shouldn't be penalized**

2.7 Linear Regression

- Notation
  - t: observed data
  - $y(\mathbf{x}, \mathbf{w}) = \sum_{i=0}^{M}\phi_i(\mathbf{x})w_i = \mathbf{w}^T\phi(\mathbf{x})$: model generating ground truth, with
    - $\mathbf{w}$: weight vector
    - $\phi(\mathbf{x})$: basis function for feature vector $\mathbf{x}$, with usually $\phi_0(\mathbf{x}) = 1$ as bias
- Assumption
  - Deterministic Model with Observation Noise
    - $t = y(x, w) + \epsilon$, where
      $\epsilon \sim \mathcal{N}(0, \beta^{-1})$ is Gaussian noise where precision (inverse variance) β
    - $\Rightarrow$ consequence
      1. likelihood $p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$
      2. $\mathbb{E}[t|\mathbf{x}] = \int t \cdot p(t|\mathbf{x})dt = y(\mathbf{x}, \mathbf{w})$
      3. unimodal distribution $p(t|\mathbf{x}) \Rightarrow$ extended by conditional mixture model
- Joint Likelihood
  - $P(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N}\mathcal{N}(t_n|\mathbf{w}^T\phi(\mathbf{x}_n), \beta^{-1})$, where
    - $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}, \mathbf{t} = \{t_1, ..., t_N\}$
  - Log Likelihood
    - $\ln P(\boldsymbol{y}|X, \theta, \beta) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$
- Log Posterior leads to regularization

- Maximizing the likelihood function ⇒ (often) excessively complex models & over-fitting
- Regularization term comes from the Prior:
  - assume Prior $p(\theta) = \mathcal{N}(\theta|0, \alpha^{-1}I)$, so that Posterior & Prior are of the same distribution to maximize log Posterior :
    $\Rightarrow \ln p(\theta|X) \propto -\frac{\beta}{2}\sum_{i=1}^{n}(y^i - h_\theta(x))^2 - \frac{\alpha}{2}\theta^T\theta + const$
  - If $\alpha \to 0$ (Prior is most useless), maximise Posterior is equivalent to maximizing likelihood
  - Maximize Posterior ⇔ Minimize cost function with regularization, where $\lambda = \alpha/\beta$

- Predictive Distribution: $p(y|x, X, Y)$
  - $p(y|x, X, Y) = \int p(y, \theta|x, X, Y)d\theta = \int p(y|\theta, x, X, Y)p(\theta|x, X, Y)d\theta$
  - $p(y|\theta, x, X, Y) = p(y|\theta, x) = \mathcal{N}(y|h(x, \theta), \beta^{-1})$
    based on assumption: $y = y(x, \theta) + \epsilon$, where ϵ is Guassian noise
    $p(\theta|x, X, Y) = p(\theta|X, Y) =$ posterior
  - $\Rightarrow p(y|x, X, Y) = \int p(y|\theta, x)p(\theta|X, Y)d\theta$
  - Expected Lost $= (bias)^2 + variance + noise$

- Notation:
  - $t = y(x, w) + \epsilon,$ where ϵ is Gaussian noise
  - $\hat{y}$ is prediction function to approximate $y = y(x, w)$

- Procedure:
  - $\mathbb{E}[(t - \hat{y})^2] = \mathbb{E}[t^2 - 2t\hat{y} + \hat{y}^2]$
    $= \mathbb{E}[t^2] + \mathbb{E}[\hat{y}^2] - \mathbb{E}[2t\hat{y}]$
    $= \mathrm{Var}[t] + \mathbb{E}[t]^2 + \mathrm{Var}[\hat{y}] + \mathbb{E}[\hat{y}]^2 - 2y\mathbb{E}[\hat{y}]$
    $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + (y^2 - 2y\mathbb{E}[\hat{y}] + \mathbb{E}[\hat{y}]^2)$
    $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + (y - \mathbb{E}[\hat{y}])^2$
    $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + \mathbb{E}[t - \hat{y}]^2$
    $= \sigma^2 + \mathrm{Var}[\hat{y}] + \mathrm{Bias}[\hat{y}]^2$
    where $\sigma^2 = \mathrm{Var}[\epsilon]$ is the noise
    (formula used: $\mathrm{Var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \Leftrightarrow \mathbb{E}[x^2] = \mathrm{Var}[x] + \mathbb{E}[x]^2$)
  - Matrix inverse can be evil & avoid inverse operation:
    $A = U\Lambda U^T$, where Λ is diagonal matirx
    $=> A^{-1} = U\Lambda^{-1}U^T$
    but number on the diagonal line of Λ can be small =¿ maybe 0 depending on accuracy of computer

2.8 Bayesian Regression

- Assumption:
  - $t = y(x, w) + \epsilon,$ where ϵ is Gaussian noise; $y(x, w)$ approximated by $\phi(x)w$

- Bayesian view:

- Gaussian Prior : $p(w) = \mathcal{N}(w|m_0, S_0)$
  Reason: to be conjugate
- Likelihood : $p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|w^T\phi(x_n), \beta^{-1}) = \mathcal{N}(\boldsymbol{t}|\Phi w, \beta^{-1}I)$
- $\Rightarrow$ Posterior : $p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N)$
  where $m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\boldsymbol{t}), S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$
- Maximum Likelihood:
  - Likelihood : $p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1})$
    - meaning: how probable the observed dataset is w.r.t the model setting (under parameter w)
  - $\ln$ Likelihood $= \sum_{n=1}^{N}[-\ln\frac{\beta}{\sqrt{2\pi}} - \frac{\beta}{2}(t_n - \phi(x)w)^2]$
  - $\frac{\partial}{\partial w}\ln$ Likelihood $= \beta\Phi^T(\boldsymbol{t} - \Phi w)$
    let $\frac{\partial}{\partial w}\ln$ Likelihood $= 0$
    $\Rightarrow w_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$
  - $\frac{\partial}{\partial \beta}\ln$ Likelihood $= -N\beta^{\frac{1}{2}} + \beta^{\frac{3}{2}}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$
    let $\frac{\partial}{\partial \beta}\ln$ Likelihood $= 0$
    $\Rightarrow \beta^{-1} = \frac{1}{N}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$
    Note: solve $w = w_{ML}$ first
- Maximum Posterior:
  - Posterior $= p(w|\boldsymbol{t})$, Prior $= p(w)$, Likelihood $= p(\boldsymbol{t}|w)$, Normalization $= p(t)$
    $\Rightarrow$ Posterior $= \frac{\text{Likelihood*Prior}}{\text{Normalization}}$
    $\Rightarrow$ Posterior $\propto$ Likelihood*Prior
  - assume Prior $p(w) = \mathcal{N}(w|m_0, S_0)$,
    so that Prior & Likelihood are conjugate $\Rightarrow$ Gaussian Posterior
  - Likelihood $p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1}) = \mathcal{N}(\boldsymbol{t}|\Phi w, \beta^{-1}I)$
  - $\Rightarrow$ Posterior $p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N)$,
    where $m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\boldsymbol{t}), S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$
    $\Rightarrow w_{MAP}$ = mean of the Gaussian $= m_N$
    Note: can also get w_{MAP} from taking gradient
- Simple Prior:
  Prior $p(w) = \mathcal{N}(w|0, \alpha^{-1}I)$
  $\Rightarrow$ Posterior $p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N)$,
  where $m_N = \beta(\alpha I + \beta\Phi^T\Phi)\Phi^T\boldsymbol{t}, S_N^{-1} = \alpha I + \beta\Phi^T\Phi$
  $w_{MAP} \to w_{ML}$, when $\alpha \to 0$ (most useless Prior)

- Maximize Posterior ⇔ Minimize cost function with regularization:
  Simple Prior ⇒ $\ln p(w|\boldsymbol{t}) = -\frac{\beta}{2}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w) - \frac{\alpha}{2}w^T w + const$
- If $\alpha \to 0$ (Prior is most useless), maximize Posterior is equivalent to maximizing likelihood
- Maximize Posterior equal to minimize sum-of-squares error function with the addition of a quadratic regularization term with $\lambda = \alpha/\beta$
- Regularization term comes from the Prior
- Predictive Distribution:
- Assume: Prior : $p(x|\alpha) = \mathcal{N}(x|0, \alpha^{-1}I),\ (m_0 = 0, S_0 = \alpha^{-1}I)$
- $p(t|x, X, \boldsymbol{t}) = \int p(t|w, x)p(w|X, \boldsymbol{t})dw$
- $\Rightarrow p(t|x, X, \boldsymbol{t}) = \mathcal{N}(t|m_N^T\phi(x), \sigma_N^2(x))$
  where $\sigma_N^2(x) = \frac{1}{\beta} + \phi(x)^T S_N \phi(x); m_N, S_N$ from Posterior$(m_N = w_{MAP})$
- Sequential data:
  - Posterior from previous data ⇔ the Prior for the arriving data
  - Sequential data and data in one go is equivalent when finfding the Porsterior
- Gradient descent
  - Hypothesis function:
    - $h_\theta(x) = x\theta,\ \theta = [\theta_0, \theta_1, ..., \theta_n]^T,\ x = [x_0, x_1, ..., x_n], x_0 = 1$
  - x is one instance
  - Cost function: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$
  - Update rule: $\forall\theta_j \in \theta, \theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial\theta_j},\ \frac{\partial J(\theta)}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^i) - y^i)x_j^i] + \frac{\lambda}{m}\theta_j$ - $\frac{d}{d\theta}J(\theta) = \frac{1}{m}((X\theta - y)^T X)^T + \frac{\lambda}{m}[0, \theta_1, ..., \theta_m]^T$ (θ_0 shouldn't be penalized)
  - simultaneously for all $\theta_j \in \theta$
  - Normal equation (mathematical solution)
    - $\theta = (X^T X)^{-1}X^T y$

2.9 Logistic Regression (Classification)

- Decision Theory:

- classes $C_1, ..., C_K$, decision regions $\mathcal{R}_1, ..., \mathcal{R}_K$ - Minimze $p(mistake) = \sum_{k=1}^{K}(\int_{\mathcal{R}_k}\sum_{i\neq k}p(x, C_i)dx)$
(can have weight on each misclassfication though) - Maximize $p(correct) = \sum_{k=1}^{K}\int_{\mathcal{R}_k}p(x, C_k)dx$

- Models for Decision Problems:
- Find a discriminant function - Discriminative Models: less powerful, yet less parameter =¿ easier to learn
- Infer **posterior** $p(C_k|x)$, C_k: $x \in C_k$, x is examples in trainning set - Use decision theory to

assign a new x - Generative Models: more powerful, but computationally expensive - Infer conditional probabilities $p(x|C_k)$ - Infer prior $p(C_k)$ - Find **either** **posterior** $p(C_k|x)$, **or** **joint distribution** $p(x, C_k)$ (using Bayes' theorem) - Use decision theory to assign a new x - **=¿ Able to create synthetic data using $p(x)$**

- Naive Bayes on Discrete Features:

- Assumption:

- Discrete Features: data point $x \in \{0, 1\}^D$

- Naive Bayes: all features conditioned on class C_k are independent with each other

$\Rightarrow p(x|C_k) = \prod_{i=1}^{D} \mu_{ki}^{x_i}(1 - \mu_{ki})^{1-x_i}$

1. Linear Discriminant (Least Squares Approach)

- Prediction:

- $y(x) = xw + w_0$, with bias = w_0, where $w = [w_1, ..., w_n]^T, x = [x_1, ..., x_n]$ - $y(x) > 0$: positive confidence to assign x to current class - $-w_0$ called threshold sometimes

- Decision Boundary $y(x) = w^T x + w_0 = 0$:

- w is orthogonal to vectors on the boundary:

assume x_1, x_2 on the boundary

$\Rightarrow 0 = y(x_1) - y_(x_2) = (x_1 - x_2)w$

- Distance from origin to boundary is $-\frac{w_0}{\|w\|}$:

assume distance is k

$\Rightarrow k\frac{w}{\|w\|}$on boundary, thus $k\frac{w}{\|w\|}w + w_0 = 0$

$\Rightarrow k = -\frac{w_0}{\|w\|}$

- $y(x)$ is a signed measure of distance from point x to boundary:

assume distance is r

$\Rightarrow y(x) = \overbrace{(x_\perp + r\frac{w}{\|w\|})}^{x} w + w_0 = \overbrace{x_\perp w + w_0}^{0} + r\|w\| = r\|w\|$

$\Rightarrow r = \frac{y(x)}{\|w\|}$

- Multi-class (k-classes):

- prediction: x is of class $\mathcal{C}_k$ if $\forall j \neq k, y_k(x) > y_j(x)$

$\Rightarrow y(x) = xW$, where $W = [w_1, ..., w_k], \forall x_i \in X, x_{i0} = 1$ (bias), $y(x)$ is 1-of-k coding

- sum-of-squares error: $E_D(W) = \frac{1}{2}\text{tr}\{(XW - T)(XW - T)^T\}$

$\Rightarrow$ optimal $W = (X^T X)^{-1} X^T T$

note that $tr\{AB\} = A^T B^T$

2. Fisher's Linear Discriminant

- Basic idea:

- Take linear classification $y = w^T x$ as dimensionality reduction (projection onto 1-D) - =¿ find a projection (denoted by vector w) which maximally preserves the class separation - =¿ if $y > -w_0$ then class C_1, otherwise C_2

- Goal:

- Distance within one class is small - Distance between classes is large

- Mean & Variance of Projected Data:

- Mean: $\widetilde{m}_k = w^T m_k$, where $m_k = \frac{1}{N_k}\sum_{x \in C_k} x$ - Variance: $\widetilde{s}_k = \sum_{x \in C_k} (w^T x - w^T m_k)^2 = w^T [\sum_{x \in C_k} (x - m_k)(x - m_k)^T] w$

- 2-Classes to 1-D line:

- Maximize Fisher criterion: $J(w) = \frac{|\widetilde{m}_1 - \widetilde{m}_2|^2}{\widetilde{s}_1^2 + \widetilde{s}_2^2}$

- Between-class covariance: $S_B = (m_1 - m_2)(m_1 - m_2)^T$

- Within-class covariance: $S_k = \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T$

$\Rightarrow J(w) = \frac{w^T S_B w}{w^T S_W w}$

- Lagrangian: $L(w, \lambda) = w^T S_B w + \lambda(1 - w^T S_W w)$

fix $w^T S_W w$ to 1 to avoid infinite solution (any multiple of a solution is a solution)

$\Rightarrow \frac{\partial}{\partial w} L = 2S_B w - 2\lambda S_W w = 0$

$\Rightarrow S_B w = \lambda S_W w$

$\Rightarrow ({S_W}^{-1} S_B) w = \lambda w$

To maximize $J(w)$, w is the largest eigenvector of ${S_W}^{-1} S_B$ if S_W invertible

- K-classes to a d-D subspace: N_k is num in class k, N is the total example num

- Between-class covariance: $S_B = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T$, where $m = \frac{1}{N} \sum_{n=1}^{N} x_n$

reduce to $(m_1 - m_2)(m_1 - m_2)^T$ when K=2 (constant ignored)

- Within-class covariance: $S_W = \sum_{k=1}^{K} S_k$, where $S_k = \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T, m_k = \frac{1}{N_k} \sum_{n \in C_k} x_n$

- Maximize Fisher criterion: $J(w) = \frac{tr\{W^T S_B W\}}{tr\{W^T S_W W\}}$

- Lagrangian:

Solve for each $w_i \in W \Rightarrow ({S_W}^{-1} S_B) w_i = \lambda_i w_i$

$\Rightarrow W$ conosists of the largest d eigenvectors

${S_W}^{-1} S_B$ is not guaranteed to be symmetric $\Rightarrow W$ might not be orthogonal

Need to minimize the whole covariance matrix ($J(w)$ as a matrix) $\Rightarrow$ not choosing same eigenvectors twice

- Maximum Possible Projection Directions = $K - 1$:

$r({S_W}^{-1} S_B) \leq \min(r({S_W}^{-1}), r(S_B)) \leq r(S_B)$

$r(S_B) \leq \sum_{K} r((m_k - m)(m_k - m)^T) = K$, as $r(m_k - m) = 1$

$\sum_{K} m_k = m \Rightarrow r(m_1 - m, ..., m_K - m) = K - 1$

$\Rightarrow r(S_B) \leq K - 1$

$\Rightarrow r({S_W}^{-1} S_B) \leq K - 1$

3. Perceptron Algorithm

- Generalised linear model $y = f(w^T \phi(x))$, where $\phi(x)$ is basis function; $\phi_0(x) = 1$

- Nonlinear activation funtion: $f(a) = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}$

- Target coding: $t = \begin{cases} 1, & \text{if } C_1 \\ -1, & \text{if } C_2 \end{cases}$

- Cost function:

- All correctly classified patterns: $w^T \phi(x_n) t_n > 0$

- Add the errors for all misclassified patterns (denoted as set $\mathcal{M}$):

$\Rightarrow E_P(w) = -\sum_{n \in \mathcal{M}} w^T \phi(x_n) t_n$

- Algorithm: (Aim: minimize total num of misclassified patterns)

- loop

choose a traning pair (x_n, t_n)

update the weight vector w: $w = w - \eta \nabla E_p(w) = w + \phi_n t_n$

where η=1 because $y = f(\cdot)$ does not depend on $\|w\|$

- Perceptron Convergence Theorem:

- If the training set is linearly separable, the perceptron algorithm is guaranteed to find a solution in a finite number of steps

(Also is the algorithm to find whether the set is linearly separable =¿ Halting Problem)

4. Maximum Likelihood

- Assumption: - $p(x|C_k) \sim \mathcal{N}(\mu_k, \Sigma)$, and all $p(x|C_k)$ share the same Σ - $p(C_1) = \pi, p(C_2) = 1-\pi$, π unknown - Likelihood of whole data set $\boldsymbol{X}, \boldsymbol{t}$, N is the num of data - $p(\boldsymbol{X}, \boldsymbol{t}|\pi, \mu_1, \mu_2, \Sigma) = \prod_{n=1}^{N} [\pi\mathcal{N}(x_n|\mu_1, \Sigma)]^{t_n}[(1-\pi)\mathcal{N}(x_n|\mu_2, \Sigma)^{1-t_n}]$ $\rightarrow$ when info of label t lost: mixture of Gaussian - $\ln(\text{Likelihood}) = \sum_{n=1}^{N}[t_n(\ln\pi + \ln\mathcal{N}(x_n|\mu_1, \Sigma)) + (1-t_n)(\ln(1-\pi) + \ln\mathcal{N}(x_n|\mu_2, \Sigma))]$ - Parameters when maximum reached: - $\pi = \frac{N_1}{N_1+N_2}$, N_1 is the num of class C_1 - $\mu_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n x_n, \mu_2 = \frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)x_n$, (mean of each class) - $\Sigma = \frac{N_1}{N}S_1 + \frac{N_2}{N}S_2$, where $S_k = \frac{1}{N_k}\sum_{n\in C_k}(x_n - \mu_k)(x_n - \mu_k)^T$

5. Logistic Regression

- Sigmoid function: $\sigma(a) = \frac{1}{1+e^{-a}}$
- $p(x|C_k) \sim \mathcal{N} \implies p(C_k|x) = \sigma(w^T x + w_0)$ (2-classes) (Generative model)
- Assumption:

$p(x|C_k) = \mathcal{N}(\mu_k, \Sigma)$ (can also be a number of other distributions)

$\forall k, p(x|C_k)$ shares the same Σ

-

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \sigma(a),$$

$$\begin{aligned}
\text{where } a &= \ln\frac{p(x, C_1)}{p(x, C_2)} \\
&= \ln\frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)} \\
&= \ldots \text{(assumption applied)} \\
&= \ln\frac{\exp(\mu_1^T\Sigma^{-1}x - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1)}{\exp(\mu_2^T\Sigma^{-1}x - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2)} + \ln\frac{p(C_1)}{p(C_2)} \\
\implies a &= w^T x + w_0 \text{ where,} \\
w &= \Sigma^{-1}(\mu_1 - \mu_2) \\
w_0 &= -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \ln\frac{p(C_1)}{p(C_2)}
\end{aligned}$$

- $\implies p(C_1|x) = \sigma(w^T x + w_0)$

$\Rightarrow$ Find parameters in Gaussian model using Maximal Likelihood Sulotion as: $p(C_1|x) \propto p(x|C_1)p(C_1) = p(x, C_1)$

- Generalize to K-classes:

$a_k(x) = \ln[p(x|C_k)p(C_k)], p(C_k|x) = \frac{\exp(a_k)}{\sum_i \exp(a_i)}$

$\Rightarrow a_k(x) = w_k^T x + w_{k0}, \text{where } w_k = \Sigma^{-1}\mu_k; w_{k0} = -\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + p(C_k)$

- Assume directly $p(C_k|x) = \sigma(w^T x + w_0)$ (2-classes) (Discriminative model)
- Assume directly: $p(C_1|w, x) = \sigma(w^T x), x_0 = 1$

$\Rightarrow$ less parameters to fit (compared to Gaussian)

- Likelihood function:

$p(\boldsymbol{t}|w, X) = \prod_{n=1}^{N} p_n^{t_n}(1-p_n)^{1-t_n}$, where, $p_n = p(C_1|x_n), t_n$ is the class of x_n

Define error function :

$E(w) = -\ln(Likelihood) = -\sum_{n=1}^{N}[t_n \ln p_n + (1-t_n)\ln(1-p_n)]$

$\Rightarrow \nabla E(w) = \sum_{n=1}^{N}(p_n - t_n)x_n$

- Find Posterior $p(w|\boldsymbol{t})$:

Likelihood is product of sigmoid

Conjugate Prior for "sigmoid distribution" is unknown

$\Rightarrow$ Assume Prior $p(w) = \mathcal{N}(w|m_0, S_0)$

$\Rightarrow \ln p(w|\boldsymbol{t}) \propto -\frac{1}{2}(w-m_0)^T S_0^{-1}(w-m_0) + \sum_{n=1}^{N}[t_n \ln p_n + (1-t_n)\ln(1-p_n)]$

find w_{MAP}, calculate $S_N = -\nabla\nabla \ln p(w|\boldsymbol{t}) = S_o^{-1} + \sum_{n=1}^{N} p_n(1-p_n)\phi_n\phi_n^T$

$\Rightarrow p(w|\boldsymbol{t}) \simeq \mathcal{N}(w|w_{MAP}, S_N)$, via Laplace Approximation

- Laplace Approximation:

- Fit a guassian to $p(z)$ at its **mode** (mode of $p(z)$: point where $p'(z) = 0$)

- Assume $p(z) = \frac{1}{Z}f(z)$, with normalization $Z = \int f(z)dz$

Taylor expansion of $\ln f(z)$ at z_0: $\ln f(z) \simeq \ln f(z_0) - \frac{1}{2}A(z-z_0)^2$,

where $f'(z_0) = 0, A = -\frac{d^2}{dz^2}\ln f(z)|_{z=z_0}$

Take its exponentiating: $f(z) \simeq f(z_0)\exp -\frac{A}{2}(z-z_0)^2$

$\Rightarrow$ Laplace Approximation $= (\frac{A}{2\pi})^{1/2}\exp -\frac{A}{2}(z-z_0)^2$, where $A = \frac{1}{\sigma^2}$

- Requirement:

$f(z)$ differentiable to find a critical point

$f''(z_0) < 0$ to have a maximum & so that $\nabla\nabla \ln f(z_0) = A > 0$ as $A = \frac{1}{\sigma^2}$

- In Vector Space: approximate $p(z)$ for $z \in \mathcal{R}^M$

Assume $p(z) = \frac{1}{Z}f(z)$

Taylor expansion: $\ln f(z) \simeq \ln f(z_0) - \frac{1}{2}(z-z_0)^T A(z-z_0)$,

Hessian $A = -\nabla\nabla \ln f(z)|_{z=z_0}$

$$\Rightarrow \text{Laplace approximation} = \frac{|A|^{1/2}}{(2\pi^{M/2})}\exp -\frac{1}{2}(z-z_0)^T A(z-z_0) \tag{2.1}$$

$$= \mathcal{N}(z|z_0, A^{-1}) \tag{2.2}$$

- Gradient descent:

- Hypothesis function: $h_\theta(x) = \sigma(x\theta) = \frac{1}{1+e^{-x\theta}}$

- Cost function:

$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^i \ln(h_\theta(x^i)) - (1-y^i)\ln(1-h_\theta(x^i))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

- Update rule: $\forall\theta_j \in \theta, \theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial\theta_j}$, $\frac{\partial J(\theta)}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^i) - y^i)x_j^i] + \frac{\lambda}{m}\theta_j$

2.10 Latent Variable Analysis

2.10.1 Principal Component Analysis (PCA)

1. Motivation:
   - Data compression (reduce highly related features) - Data visualization
   2. Assumption:
   - Gaussian distributions for both the latent and observed variables
   3. Two Equivalent Definition of PCA:

- Linear projection of data onto lower dimensional linear space (principal subspace) such that:

⇒ variance of projected data is maximized

⇒ distortion error from projection is minimized

4. Maximum Variance Formulation

- Goal:

- project data from D dimension to M while maximizing the variance of projected data

- Eigenvalues λ of covariance matrix S express the variance of data set X in direction of corresponding eigenvectors

- Projection Vectors:

- $U = (u_1, ..., u_M)$, where $\forall i \in \{1, .., M\}, u_i \in \mathbb{R}^D$ s.t. $u_i^T u_i = 1$(only consider direction)

- Projected Data:

- Mean = $\bar{x}^T U$, where $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x^i$ - Variance = $tr\{U^T S U\}$, where $S = \sum_{i=1}^{N}(x^i - \bar{x})(x^i - \bar{x})^T$ (outer product)

- Lagrangian to maximize Variance:

- $L(U, \lambda) = tr\{U^T S U\} + tr\{(I - U^T U)\lambda\}$

constraint $u_i^T u_i = 1$ to prevent $u_i \to +\infty$

$$\text{For each } u_i \in U, \ \frac{\partial}{\partial u_i} L = 2Su_i - 2\lambda_i u_i = 0 \tag{2.3}$$

$$\Rightarrow Su_i = \lambda_i u_i \tag{2.4}$$

$$\Rightarrow U \text{ consists of eigenvectors corresponding to the first M large eigenvalue of } S \tag{2.5}$$

(S symmetric $\Rightarrow U$ orthogonal)

5. Minimum Error Formulation:

- Introduce Orthogonal Basis Vector for D dimension:

- $U = (u_1, ..., u_D)$

- Data representation:

- Original: $x^n = \sum_{i=1}^{D} \alpha_i^n u_i$ - Projected: $\widetilde{x^n} = \sum_{i=1}^{M} z_i^n u_i + \sum_{i=M+1}^{D} b_i u_i$

$(z_1^n, ..., z_M^n)$ is different for different x^n, $(b_{M+1}, ..., b_D)$ is the same for all x^n

- Cost function: $J = \frac{1}{N}\sum_{n=1}^{N} \|x^n - \widetilde{x^n}\|^2$, where $\widetilde{x^n} = \sum_{i=1}^{M} z_i^n u_i + \sum_{i=M+1}^{D} b_i u_i$

- Let $\begin{cases} \frac{\partial}{\partial z_j^n} J = 0 \\ \frac{\partial}{\partial b_j} J = 0 \end{cases} \Rightarrow \begin{cases} \frac{1}{N} 2(x^n - \widetilde{x^n})^T(-u_j) = \frac{2}{N}(z_j - (x^n)^T u_j) = 0 \\ \frac{1}{N}\sum_{n=1}^{N} 2(x^n - \widetilde{x^n})^T(-u_j) = \frac{2}{N}\sum_{n=1}^{N}(b_j - (x^n)^T u_j) = 0 \end{cases}$

$$\Rightarrow \begin{cases} z_j = (x^n)^T u_j & j \in \{1, ..., M\} \\ b_j = \overline{x}^T u_j & j \in \{M+1, ..., D\} \end{cases}$$

Noticing $(x^n)^T u_j = (\sum_{i=1}^{D} \alpha_i^n u_i^T) u_j = a_j \Rightarrow a_j = (x^n)^T u_j$

$$\Rightarrow x^n - \widetilde{x^n} = \sum_{i=M+1}^{D} [(x^n - \overline{x})^T u_i] u_i$$

-

$$\Rightarrow J = \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{i=M+1}^{D}[(x^n-\overline{x})^T u_i]u_i\right)^T\left(\sum_{i=M+1}^{D}[(x^n-\overline{x})^T u_i]u_i\right) \tag{2.6}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{i=M+1}^{D}u_i^T((x^n-\overline{x})^T u_i)\right)\left(\sum_{i=M+1}^{D}((x^n-\overline{x})^T u_i)u_i\right) \tag{2.7}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}u_i^T(x^n-\overline{x})^T u_i u_i^T(x^n-\overline{x})u_i \qquad u_i \text{ orthogonal to each other} \tag{2.8}$$

$$= \sum_{i=M+1}^{D}u_i^T\left(\frac{1}{N}\sum_{n=1}^{N}(x^n-\overline{x})^T(x^n-\overline{x})\right)u_i \qquad \|u_i\| = 1 \tag{2.9}$$

$$\tag{2.10}$$

$\Rightarrow J = \sum_{i=M+1}^{D}u_i^T S u_i$, where $S = \frac{1}{N}\sum_{n=1}^{N}(x^n-\overline{x})^T(x^n-\overline{x})$

- Lagrangian to Minimize J:

- $L(u_{M+1},...,u_D,\lambda_{M+1},...,\lambda_D) = \sum_{i=M+1}^{D}u_i^T S u_i + \sum_{i=M_1}^{D}\lambda_i(1-u_i^T u_i)$

constraint $\|u_i\| = 1$ to prevent $u_i = 0$

For each u_i, $\frac{\partial}{\partial u_i}L = 2Su_i - 2\lambda_i u_i = 0$

$\Rightarrow Su_i = \lambda_i u_i$

$\Rightarrow$ To minmize J, take eigenvectors with the first $(D-M)$ small eigenvalue orthogonal to (out of) subspace

$\Leftrightarrow$ define subspace with eigenvectors with the first M large eigenvalue

-

Intuition: $\widetilde{x_n} = \sum_{i=1}^{M}((x^n)^T u_i)u_i + \sum_{i=M+!}^{D}(\overline{x}^T u_i)u_i$ (2.11)

$$= \overline{x} + \sum_{i=1}^{M}[(x^n-\overline{x})^T u_i]u_i \tag{2.12}$$

1. Singular Value Decomposition - SVD:

- Intorduce matrix $A_{m\times n}$

- $(A^T A)_{n\times n}$ symmetric matrix (actually, Gram matrix $\rightarrow$ semi-definite) -

eigenvalue decomposition: (2.13)

$A^T A = VDV^T$, V is normalized ($v_i^T v_i = 1$) with column as eigenvector (2.14)

- $AV = (Av_1,...,Av_n)_{m\times n}$

- Let $r(A) = r$

$\Rightarrow r(A^T A) = r(A) = r$ (2.15)

$r(AV) = \min\{r(A), r(V)\} = \min\{r, n\} = r$ (2.16)

- Reduce AV to basis $(Av_1,...,Av_r)$

- Let $U = (u_1,...,u_r) = (\frac{Av_1}{\sqrt{\lambda_1}},...,\frac{Av_r}{\sqrt{\lambda_r}})$, λ_i is i-thh eigenvalue of $A^T A$

- Orthogonal: $\forall i \neq j, u_i^T u_j = \frac{1}{\sqrt{\lambda_i\lambda_j}}v_i^T A^T A v_j = \frac{\lambda_j}{\sqrt{\lambda_i\lambda_j}}v_i^T v_j = 0$

- Unit: $\|u_i\| = \frac{\|Av_i\|}{\sqrt{\lambda_i}} = \frac{\sqrt{<Av_i, Av_i>}}{\sqrt{\lambda_i}} = 1$

$\Rightarrow U$ is standard orthogonal (orthonormal) basis

- $AV = U\Sigma$, where $\Sigma = D^{\frac{1}{2}}$

- Expand U to orthonormal in $\mathbb{R}^m : (u_i, ..., u_m)$
- Epand corresponding part in Σ with 0
- $A = U\Sigma V^T$, with singular value in Σ in decreasing order
2. SVD with PCA:
- X is data matrix in row (centered - zero mean)
- Eigenvectors of corvariance matrix $S = X^T X$ are in V, where $X = U\Sigma V^T$
- When using $S = U\Sigma V^T \Rightarrow U = V \wedge S = V\Sigma V^T$
reduced to eigenvalue decomposition
- $S = VDV^T$ with V orthonormal:
Eigenvalues λ of covariance matrix S express the variance of data set X in direction of corresponding eigenvectors
- Projection:
- $\widetilde{X} = XV_M$, where V_M contains first M-large eigenvectors - Projection direction is **not** unique
3. Reconstruction (approximate):
- Data is projected onto k dimension using SVD with $S = U\Sigma V^T$ - $x_{approx} = U_{reduce} \cdot z$, U_{reduce} is n*k matrix, z is k*1 vector - ![Reconsturction from data Compression](../../Machine
4. Choosing k (num of principal components):
- choose the **smallest** k making $\frac{J}{V} \leq 0.01$ =¿ 99
- $[U, S, V]$ = svd(Sigma) =¿ $\frac{J}{V} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$, S is diagonal matrix
=¿ check $\frac{J}{V}$ before compress data
5. Data Preprocessing:
- PCA *vs.* Normalization: - Normalization: Individually normalized but still correlated - PCA: create decorrelated data − whitening - Whitening: projection with normalization - $S = VDV^T$, where S is Gram matrix over X^T - $\forall n, y_n = D^{-\frac{1}{2}} V^T (x^n - \overline{x})$, where $\overline{x}$ is the mean of X

$$\Rightarrow y^n \text{ has zero mean} \tag{2.17}$$

$$cov(\{y^n\}) = \frac{1}{N} \sum_{n=1}^{N} y_n y_n^T = D^{\frac{-1}{2}} V^T S V D^{\frac{-1}{2}} = I \tag{2.18}$$

6. Tips for PCA:
- Do NOT use PCA to prevent overfitting, use regularization instead - Try original data before implement PCA - Train PCA only on trainning set

2.10.2 Independent Component Analysis (ICA)

1. Goal: - Recover original signals from a mixed observed data - Source signal $S \in \mathbb{R}^{N \times K}$; mixing matrix A; Observed data $X = SA$ - Maximizes statistical independence - Find A^{-1} to maximizes independence of columns of S 2. Assumption: - At most one signal is Gaussian distributed - Ignorde amplitude and order of recovered signals - Have at least as many observed mixtures as signals - A invertible 3. Independence *vs.* Uncorrelatedness - Independence ⇒ Uncorrelatedness - $p(x_1, x_2) = p(x_1)p(x_2) \Rightarrow \mathbb{E}(x_1 x_2) - \mathbb{E}(x_1)\mathbb{E}(x_2) = 0$ 4. Central Limit Theorem 5. FastICA algorithm

2.10.3 t-SNE

1. Problem & Focus 2. Compared to PCA: - No whitening function to use for new data - PCA can only capture linear structure inside the data - t-SNE preserves the ¡u¿local distances¡/u¿ in the original data

2.10.4 Anomaly Detection

1. Problem to solve:

- Given dataset $x^1, x^2, ..., x^m$, build density estimation model $p(x)$ - $p(x^{test} < \epsilon)$ =¿ x^{test} anomaly

2. Hypothesis function:

- $p(x) = \prod_{i=1}^{n} p(x_i), x \in R^n, \forall i \in [1, n], x_i \sim N(\mu_i, \sigma_i^2)$ - $\mu = \frac{1}{m}\sum_{i=1}^{m} x^i, \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)^2$ - assume $x_1, ..., x_n$ independent from each other

3. Multivariate Gaussian:

- $p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$,

$x \in R^n, \mu \in R^n, \Sigma \in R^{n \times n}$, where Σ is covariance matrix

- $\mu = \frac{1}{m}\sum_{i=1}^{m} x^i, \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^i-\mu)(x^i-\mu)^T$ - $x_1, ...x_n$ can be correlated but **not** linearly dependent - need $m > n$ ($m \geq 10 nsuggested$) or elas Σ non-invertible

4. Algorithm:

- choose features - compute μ, σ - compute $p(x)$ for new example, anomaly if $p(x) < \epsilon$

5. Evaluation (real-number):

- Labeled data into normal/anomalous set

(okay if some anomalies slip into normal set)

- training set: unlabeled data from normal set (60- CV set: labeled data from normal (20- test set: labeled data from normal (20

- Use evaluation metrics (skewed data)

6. When to use:

- Anomaly detection: - Very small num of positive data (0-20 commonly); Large num of negative data - Difficult to learn from positive data (not enough data, too many features...) - Future anomalies may look nothing like given data - Supervised Learning: - Larger num of positive & negative data - Enough positive data for algorithm to learn - Future positive example is likely to be similar to given data

7. Example:

- Anomaly detection: - Fraud detection, Manufacturing, Monitoring machines in data center... - Supervised learning: - Email spam classification (enough data), Weather prediction (sunny/rainy/etc), Cancer classification...

8. Tips:

- Non-guassian feature: transformation / using other distribution - Choosing features: compare anomaly data with normal data

2.10.5 Recommender System

1. Problem Formulation:

- $r_{i,j} = 1$ if item i is rated by user j
- $y_{i,j}$ = rating of item i given by user j
- θ^j = parameter vector for user j
- x^i = feature vector for movie i

=¿ for user j, movie i, ($r_{i,j} = 0$), predict rating $x^i\theta^j$

2. Content Based Recommendations:

- Treat each user as a seperate linear regression problem with the feature vectors of its rated items as traning set

Assume features for each items (x^i) are available and known

=¿ given X estimate Θ

- Cost Function for θ_j:

$J(\theta^j) = \frac{1}{2}\sum_{i:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(\theta_k^j)^2, \theta^j \in R^{n+1}(\theta_0$ not regularized)

- Cost Function for Θ:

$J(\Theta) = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^j)^2,$

$\theta^j \in R^{n+1}(\theta_0$ not regularized), n_u is num of users

- Update Rule: $\forall\theta_k^j \in \theta^j, \theta_k^j := \theta_k^j - \alpha\frac{\partial J(\Theta)}{\partial\theta_k^j}$, $\frac{\partial J(\Theta)}{\partial\theta_k^j} = \sum_{i:r_{i,j}=1}(x^i\theta^j - y_{i,j})x_k^i + \lambda\theta_k^j$, for $k \neq 0$ ($\theta^j \in R^{n+1}$)

3. Collaborative Filtering

- **Assume preference of each users (θ^j) are available and known**

=¿ given Θ estimate X

- Cost Function for x^i: $J(x^i) = \frac{1}{2}\sum_{j:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(x_k^i)^2$ - Cost Function for X:

$J(X) = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^i)^2$

$x^j \in R^{n+1}(x_0$ not regularized), n_m is num of items - Update Rule: $\forall x_k^i \in x^i, x_k^i := x_k^i - \alpha\frac{\partial J(X)}{\partial x_k^i}$,

$\frac{\partial J(X)}{\partial x_k^i} = \sum_{j:r_{i,j}=1}(\theta^j x^i - y_{i,j})\theta_k^j + \lambda x_k^i$, for $k \neq 0$ ($x^i \in R^{n+1}$)

- Basic Idea:
- Randomly initialize Θ
- loop:

Estimate X

Estimate Θ

- Cost Function:

$J(X,\Theta) = \frac{1}{2}\sum_{(i,j):r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^i)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^j)^2, x \in R^n, \theta \in R^n$

(the sum term in $J(\Theta)$, $J(X)$, and $J(X,\Theta)$ is the same)

- Update Rule:

- $\forall x_k^i \in x^i, x_k^i := x_k^i - \alpha\frac{\partial J(X,\Theta)}{\partial x_k^i}$, $\frac{\partial J(X,\Theta)}{\partial x_k^i} = \frac{\partial J(X)}{\partial x_k^i} = \sum_{j:r_{i,j}=1}(\theta^j x^i - y_{i,j})\theta_k^j + \lambda x_k^i, x^i \in R^n$

- $\forall\theta_k^j \in \theta^j, \theta_k^j := \theta_k^j - \alpha\frac{\partial J(X,\Theta)}{\partial\theta_k^j}$, $\frac{\partial J(X,\Theta)}{\partial\theta_k^j} = \frac{\partial J(\Theta)}{\partial\theta_k^j} = \sum_{i:r_{i,j}=1}(\theta^j x^i - y_{i,j})x_k^i + \lambda\theta_k^j, \theta^j \in R^n$

- **Algorithm**
- Initialize X, Θ to **small random values**

=¿ for symmetry breaking (similar to random initialization in neural network)

=¿ so that algorithm learns features $x^1, ..., x^{n_m}$ that are different from each other

- Minimize $J(X,\Theta)$
- Predict $y_{i,j} = x^i\theta^j$ ($Y = X\Theta$)
- Finding Related Item to Recommend
- $||x^i - x^j||$ is samll =¿ item i and j is similar
- Mean Normalization:
- Problem: if user j hasn't rated any movie, $\theta^j = [0, ..., 0]$

=¿ predicted rating of user j on all item $= 0$

=¿ useless prediction

- Algorithm (row version):

compute vector $\mu, \forall\mu_i \in \mu, \mu_i =$ mean of Y_i, where Y_i is the i^{th} row in Y

manipulate Y: $\forall y_{i,j} \in Y \wedge r_{i,j} = 1, y_{i,j} -= \mu_i$ =¿ the mean of each row in Y is 0

predict rating for user j on item $i = x^i\theta^j + \mu_i$

- For item i with no rating
=¿ apply column version of mean normalization
(but user with no rating is generally more important)

2.11 Large Scale Machine Learning

- Compute $cost(\theta, (x^i, y^i))$ before updating
For every k update iterations, plot average $cost(\theta, (x^i, y^i))$ over the last k examples
- Checking curves:
Increasing k result in smoother line and less noise, but the result is more delayed

2.11.1 Online Learning

1. Situation: - Has too many data (can be considered as infinite) - When data comes in as a continuous stream - Can adapt to changing user preference 2. Procedure: - Use one example only once (Similar to stochastic gradient decent in this sense

2.11.2 Map-reduce

1. In Batch Gradient Descent:
- Update rule $\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i$ - Parallelize the computation of $\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i$ by dividing the data set into multiple sections
2. Ability to reduce:
- Contain operation over the whole data set
(Neural Network can be map-reduced)

Chapter 3

Linear Regression

Chapter 4

Linear Classification

Chapter 5

Kernel Methods

Chapter 6

Graphical Models

Chapter 7

Mixture Models and EM

Chapter 8

Approximate Inference

Chapter 9

Sampling Methods

Chapter 10

Continuous Latent Variable

Chapter 11

Sequential Data

Chapter 12

Deep Learning

12.1 Interview of Fame

12.1.1 Geoffrey Hinton

Knowledge Embedding

- BP
  - psychology view: knowledge in vectors
  - semantic AI: knowledge graph
  - BP algorithm can interpret & convert between feature vector and graph representation (with some embedding)
- Boltzmann Machine
  - Leaning Algorithm on Density Net
    - same information in forward & backward propagation to learn feature embedding
  - Restricted Boltzmann Machine (RBM)
    - ways of learning in deep dense net with fast inference
    - iterative learning (adding layer after the above trained)
    - ReLU $\Leftrightarrow$ a stack of sigmoid functions (approximately) in RBM
    - ReLU units initialized to identity for efficient learning
- EM
  - EM with Approximate E Step
- vs. Symbolic AI
  - Symbolic AI: symbolic logic-like expression to do reasoning
  - yet, maybe state vector to represent knowledge

Brain Science

- Brain: Nets Implemented by Evolution
  - trying to train without BP
  - doing BP (get derivatives) with re-construction error (auto-encoder)

Memory in Nets

- Fast Weights for Short-term Memory
- Capsule Net
  - structured knowledge representation in each unit (feature with sets of property)
  - $\Rightarrow$ enable nets to vote rather than filtering - thus better generalization
  - now working: published in **2017 NIPS**

Unsupervised Learning

- Importance
  - better than human eventually (as supervised learning has limited maximum)
  - GAN as a breakthrough

"Slow" Feature

- Non-linear Transform to Find Linear Transform
  - find a latent representation containing linear transform to do the work
  - e.g. change viewpoints: pixels $\rightarrow$ coordinates $\rightarrow$ linear transform $\rightarrow$ back to pixels

Relations between Computers

- showing computer data to work
  - instead of programming it to work

12.1.2 Pieter Abbeel

Deep Reinforcement Learning

- Overall Challenge
  - Representation
  - Exploration Problem
  - Credit Assignment
  - Worst Case Performance
- Advantage (Deep Nets in RL)
  - network capturing the representation (state vector)
- Question in DRL
  - how to learn safely
  - how to keep learning (under small negative samples) e.g. better than human
  - can we learn the reinforcement learning program (RL in the RL)
  - long time horizon
  - use experience across tasks
- Success of DRL
  - simulated robot inventing walking... $\Rightarrow$ single general algorithms to learn

12.1.3 Ian Goodfellow

Generative Adversarial Networks

- Generative Models
  - Resembling
    - trained to optimized the distribution behind training data
      (then sampled from that distribution to get more imaginary training data)
    - ⇒ produce data to resemble the training data
  - Usage
    - semi-supervised learning
    - data augmentation
    - simulating scientific experiment
  - Previous Ways
    - Boltzmann Machine
    - Sparse Coding
  - Now: Generative Adversarial Networks (GANs)
  - Future
    - increase reliability of GANs (stabilizing)

12.1.4 Yoshua Bengio

Thoughts

- Fallacy
  - Smoothness in Nonlinearity
    - to ensure non-zero gradients every where
- Surprising Fact
  - ReLU in Deep Net
    - inspired initially by biological connection
- **Distribution v.s. Symbolic Representation**
  - Distributed Representation
    - distributed in lots of units, instead of a symbolic representation in a single cell
      (agree on Geoffrey Hinton)
  - Curse of Dimensionality
    - neural net's distributed representation for joint distribution over random variables
  - ⇒ Word Embedding
    - generalized to joint distribution over sequence of words

Works

- Piecewise Linear Activation (PLU)
- Unsupervised Learning
  - Focus
    - Denoising auto-encoder
    - GANs
  - Importance
    - human ability: self-teaching, building world-model from perception
  - Unsupervised Learning + Reinforcement Learning
    - underlying concept across two fields: machine can learn through interactions ⇒ learning "good" representation (yet, what is "good")
  - Possible Directions
    - loss function: not even defined for each task (not knowing which is good for what?)
- Attention
  - Machine Translation (Founder)
  - Generalized into Other Fields
- Back-prop in Brains (Neural Science
  - Reasons for Efficiency of Backprop
  - Larger Family behind Credit Assignment

)

12.1.5 Yuanqing Lin

National Deep Learning Lab

- Paddle Paddle
- Baidu Lab

12.1.6 Andrej Karpathy

Human Benchmark

- Programming by Showing
  - Requirement
    - input + output as specification
    - metric as goal
  - Writer
    - the optimizer
- Understanding Importance of Benchmark
  - importance to do better given the current performance on the dataset (as important increase after passing human error)
- Understanding Network Behavior
  - compared to the process of human decision

Transfer Learning

- Image Task
  - feature extractor + fine tune/modification onto various task

12.1.7 Ruslan Salakhutdinov

Restricted Boltzmann Machine

- Auto Encoder
  - Encoding All Kind of Data
    - from digit to face, document, etc...
    - deeper and deeper structure
- Training Boltzmann Machine
  - Pretraining
    - increase the low boundary by training the previous layer
    - then add another layer to train, ...
  - Direct Training (with GPU)
    - similar, or better result
- Boltzmann Machine Ability
  - Generative Model
    - model coupling distributions in data
      $\Rightarrow$ scalable (more scalable than current model&operation)
    - only way to train the model in the early age
- Progress on Generative Model
  - probabilistic max pooling
  - variational encoder
  - deep energy model
  - semi-supervised Model

12.1.8

title

-

12.1.9 Research

Topics

- Point Cloud
  - Operations on Points: how to embed location in operation
    - select fixed number of points via coord?: then take weighted average (conv) / max (pooling) on them
    - need a "select input points" op: like deformable conv?

- Bounding Box Directly from Points: no voxel
  - clustering + regression on each cluster?

- Unsupervised Learning
  - Deep Belief Nets

- Reinforcement Learning
  - Deep Reinforcement Learning
    - scalable system
    - communicative& cooperating agents

- One-shot / Transfer Learning
  - Learning the Ability to Learn

- General AI
  - Structure for General Task
    - neural network or other structure, shared for multiple tasks
      (instead of breaking down to different parts like segmentation, detection, etc.)
      (instead of the split of cv, nlp, plaining, etc.)
    - $\Rightarrow$ a full agent (instead of decomposed function)
      $\Rightarrow$ optimization method/objective need to be carefully defined
  - Attempt for General AL
    - scaling up supervised learning: imitating human
    - unsupervised learning: AIXI, artificial evolution, etc.

- AI Security
  - Anti Inducing
    - NOT to be fooled/induced to do unappropriated things
      (even if algorithm is right)
  - Built-in Security

- Fairness in AI
  - Dealing Societal Issue
  - Reflecting Preferred Bias

- Auto Optimization (Hyperparameter Tunning)
  - Swarm Optimization
  - Expectation Maximization
    - target variable θ = hyperparameters
    - hidden variable Z = weights of network
    - data X = dataset

    $\Rightarrow$

    - E-step: evaluate $\mathbb{E}_{Z|\theta_n,X}(\ln P(Z,X|\theta))$
      - $\ln P(Z,X|\theta)$: log likelihood of hyperparam θ (for weights & data to be observed)
      - $P(Z|\theta_n,X)$: posterior of weights Z

⇒ evaluate (approximate) the expectation of the log likelihood of hyperprarm θ
(from a functional view, train with $\theta_0 - \theta_N$, evaluate model M times in training,
thus with weights $Z_{00} - Z_{NM}$)
⇒ a matrix with n as row entry, m as column entry, mapping to both $\ln P(Z, X|\theta), P(Z|\theta_n, X)$
⇒ then marginalize (taking the expectation) over Z, to get a (sampled) function
over θ

- M-step: maximize the result function from E-step
  - fit a curve & maximize w.r.t hyperparams θ

- World Understanding: after perception
  - Unsupervised Learning + Reinforcement Learning
    - machine learns from interactions
    - machine builds a representation of world (like human ability, without fine label)

    ⇒ building world-model from perception
  - Causality Mining

- Model Interpretation
  - Logical Formalization
    - deep learning can be understood logically
      e.g. what make deep net training harder? understand the limit of current algorithm/model and **why**

Advises

- Learning Direction
  - Math
    - statistic
    - linear algebra
    - calculus
    - optimization

- Reading
  - read a little bit & find somewhere intuitively not right
    - good intuition: eventually work;
      bad intuition: not working no matter what it is doing
    - if other doubts your idea as bullshit ⇒ a sign for real good result
  - a supervisor with similar belief
  - PhD vs. Company
    - amount of mentoring
    - faster if dedicated supervisor available
    - resource

- Practice
  - open-source learning resource
  - open source contribution
    - contribute to open source framework (e.g. conv on sparse matrix in TF)
    - implement the paper, the open source it (as a tool for other)

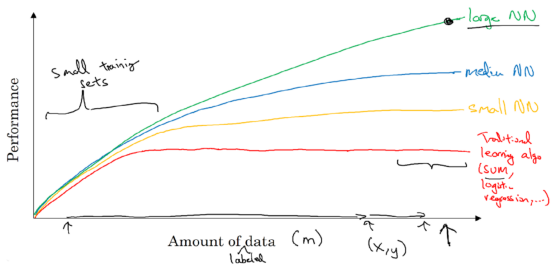
- work on a projected and open source it
  ⇒ the stage (e.g. github) will bring people to you
- implement the tools: to find out how & why it works
  ⇒ derive theories from the
- full stack of understanding
  ⇒ understand the implementation under the deep learning framework

12.2 Basic Neutral Network

12.2.1 Advantages

Large/Big Data

- Larger Maximum Capability
  - Curve given Amount of Data



  - Reasons
    - the scale of data (labeled)
    - the scale of neural network (computability)
    - the scale of efficiency: e.g. ReLu, faster parallel algorithm

Flexibility

- Different Structures for Different Tasks
  - Same Data & Task
    - changing settings/structures of deep learning model can make a difference (v.s. SVM, etc.)
- Ability to Choose Basis Functions
  - Functional View
    - $y(\mathbf{x}, \mathbf{w}) = f(\mathbf{w}^T \phi(\mathbf{x}))$, where ϕ is basis function , $f(\cdot)$ is net as a function
  - Learning ϕ: choose embedding ⇒ choose basis function
  - Learning $\mathbf{w}$: choose which feature / basis functions more useful
- Solving Bias-Variance Trade-off
  - Complexity + Data/Regularization
    - easy complexity via depth, size
      ⇒ reduce bias, without hurting variance by utilizing big data
    - easy regularization via L2 ant etc.
      ⇒ prevent high variance without hurting bias much in a deep/big net

Power of Depth

- Deep Representation
  - Low-level → High-level
    - multiple layers to choose & combine useful information (creating new feature/basis)
      ⇒ next layer use chosen/combined simple basis to build more complex one
    - ⇒ an hierarchy from low-level information to high-level information
- Circuit Theory
  - Power of Combination
    - functions that can be compactly represented by a depth k architecture might require an exponential number of computational nodes using a depth $k-1$ architecture
      (from the perspective of factorization)

Yet, start from the SHALLOW (logistic regression) before trying the deep

12.2.2 Problem

(n units in one hidden layer)

Weight-space Symmetries

- Symmetries in Activation Function
  - $\mathcal{O}(2^n)$, e.g. $\arctan(-x) = -\arctan(x)$ ⇒ changing signs of all input & output has the same mapping (reduce effective data)
- Positional Combination in One Layer
  - $\mathcal{O}(n!)$ exchange unit with each other (together with their input output weights) ⇒ mapping stay the same

⇒ $\mathcal{O}(n!2^n)$ overall weight-space symmetries

High-Dimension Search Space

- Multiple Critical Points
  - Symmetries
    - at least $\mathcal{O}(n!2^n)$ critical points ($\nabla E(w) = 0$), where $E(w)$ is error function due to weight-space symmetries
  - Saddle Points
    - both the bottom (in one dimension) and the top for another
    - due to high-dimension weight space
      ⇒ more likely to have functions being convey/convex in different dimensions
  - Local Optima
    - less then saddle points in amount, due to high-dimension weight space
      e.g. usually $\geq 10^4$-D for modern deep nets
- Plateaus
  - a large flat region where gradient → 0
    ⇒ gradient descent slowly down the flat surface (before exiting)

- ⇒ slow down gradient descent significantly

- Expensive in Finding Critical Point
  - expensive for even local optima with gradient decent
  - as expensive as $\mathcal{O}(n^3)$ if using Laplace approximation

Gradient Vanishing/Exploding

- Gradient Vanishing
  - Saturated Function
    - sigmoid/tanh function: gradient $\to 0$ when input $\to \pm\infty$
  - Exponential Effect
    - with depth L, each activation (e.g. tanh) output $a^l < 1$ and weight $\mathbf{w}^l < 1$
      $\Rightarrow y(\mathbf{x}, W) \approx w^L \mathbf{w}'^{L-1}\mathbf{x}$, with $\mathbf{w}' < 1$
      $\Rightarrow$ all the gradient along the way get multiplied by number < 1
      $\Rightarrow$ gradient exponentially decayed in back-prop

- Gradient Exploding
  - Exponential Effect
    - similarly, each activation (e.g. ReLU) output $a^l > 1$ and weight $\mathbf{w}^l > 1$
      $\Rightarrow y(\mathbf{x}, W) \approx w^L \mathbf{w}'^{L-1}\mathbf{x}$, with $\mathbf{w}' > 1$
      $\Rightarrow$ all the gradient along the way get multiplied by number > 1
      $\Rightarrow$ gradient exponentially augmented in back-prop

- Possible Solutions
  - Random Initialization
    - Xavier Initialization: for gradient vanishing & exploding
  - Activation
    - ReLU: for gradient vanishing
  - Skip/Concat Connection
    - residual block
    -

12.2.3 Learning

Forward-Backward Propagation

- Representation
  - Layers
    - input layer
    - hidden layer(s): layer with NO ground truth (for the associated weights) available
      note: input & hidden layers have associated biases as well (usually)
    - output layer
  - Neuron (Unit)
    - s_l: num of units in layer l
    - w^l: weight matrix of mapping from layer l to $l+1$, with shape of $(s_{l+1}, s_l + 1)$
    - $h(\cdot)$: activation function (usually shared)

        - a_j^l: activation output of unit j at layer l
        - z_j^l: output of unit j at layer l
          (represent parameterized basis, also the input for layer $l+1$)
    - Intuition
        - all stacked vertically (vertical vector)
          $\Rightarrow$ horizontally for different examples; vertically for different units

- Forward Propagation (Inference)
    - Activation $a^{j+1} = w^j \cdot [z_0^j, ..., z_{s_j}^j]^T$, with $z_0 = 1$
    - Unit Output $z^{j+1} = h(a^{j+1}) = [z_1^1, ..., z_{s_{j+1}}^{j+1}]^T$

- Backward Propagation
    - Loss $\mathcal{L}(W) =$

- Practice of Back Prop
    - Caching Intermediate Result
        - naturally cached: input $a^0 = x$, weights matrix w and bias b
        - activation input/output a/z
          (since will be used in back-prop)
    - Auto Difference
        - achievement: calculate the derivatives along the forward prop **!**

12.3 Operations & Layers Structure

12.3.1 Operations in Network

Activations

- Sigmoid $a = \sigma(z)$
    - Pros
        - mapping to $(0, 1)$, with $\sigma(0) = 0.5$
    - Cons
        - gradient vanishing: $\sigma(z)' = \sigma(z)(1 - \sigma(z)) \Rightarrow \lim_{z \to \pm\infty} \sigma(z)' \to 0$
          (as the gradient passed through (via chain rule) $= \frac{a}{z}\frac{z}{w}$)

- Tangent $a = \tanh(z)$
    - Pros
        - empirically, almost always better than sigmoid (in hidden layers)
        - maps to $(-1, 1)$, with $\tan(0) = 0 \Rightarrow$ help centering data (0-mean)
          $\Rightarrow$ make the learning of next layer easier
    - Cons
        - still, gradient vanishing when $z \to \pm\infty$

- Rectified Linear Unit (ReLU) $\max(0, z)$
    - Derivation: approximated by a stack of sigmoid
        -

  - Pros
    - mitigate gradient vanishing: $\forall z > 0, a = z \Rightarrow$ learn much faster

    $\Rightarrow$ the default choice!
  - Cons
    - undefined behavior at $x = 0$ (actually, gradient becomes the sub-gradient)
    - gradient totally vanished for $x < 0$
    - $\Rightarrow$ dead units: weights learned/initialized to always output negatives
      $\Rightarrow$ activation always output 0
      $\Rightarrow$ the unit always output 0

- Leaky Relu $a = \max(\alpha z, z), \alpha \to 0^+$ (e.g. $\alpha = 0.01$)
  - Pros
    - mitigate the gradient vanishing problem for $(-\infty, +\infty)$
    - avoid dead units problem

    (yet not that popular as ReLU)

- Piecewise Linear Unit (PLU) $a = \max(\alpha(z + \beta) - \beta, \min(\alpha(z - \beta) + \beta, z)$
  - Pros
    - hybrid of tanh & ReLU: three linear pieces approximating tanh in a given range
    - more expressive than ReLU: more nonlinear, better to fit smooth nonlinear function
    - mitigate gradient vanishing problem: due to linearity
  - Cons

- Linear (Identity) Activation $a = z$
  - Pros
    - used in regression to output real number $\in (-\infty, +\infty)$
    - used in compression net
  - Cons
    - stacked units with linear activation $\Leftrightarrow$ single linear transformation
    - logistic regression with linear activation in hidden layer is NO more expressive than logistic regression with no hidden layer **!**

Normalization in Network

- Batch Normalization
  - Definition
    - for an activation in hidden layer with input z, a batch with size N_b
    - calculate the mean of current batch $\mu = \frac{1}{N_b} \sum_n z_n$, where z_n for the n^{th} example
    - calculate the deviation of current batch $\sigma = \sqrt{\frac{1}{N_b} \sum_n (z_n - \mu)^2}$
    - normalize to be $z'_n = \frac{z_n - \mu}{\sigma}$
    - allow model to recover/manipulate original distribution: $\hat{z}_n = \gamma z'_n + \beta$,
      where γ, β being trainable (updated by optimizer using gradients)

- Implementation
  - preferred to apply batch norm on z (before activation), instead of after it
  - for math stability, $z'_n = \frac{z_n - \mu}{\sigma + \epsilon}$, with $\epsilon \to 0+$
  - (usually) with mini-batch, calculate the mean & variance from only the mini-batch
  - with batch norm, original bias b in calculating $z = wx + b$ becomes pointless
    $\Rightarrow$ integrated into the β in batch norm
  - at test time (1 example a time): need an estimation for μ, σ
    $\Rightarrow$ exponentially weighted average over β, σ in training time
- Understanding
  - normalize the intermediate data to have 0 mean, unit variance
    $\Rightarrow$ to speed up the training from some hidden layers (as normalization does)
  - remain the ability to transfer the data to have other mean & variance
    (controlled by γ, β)
  - control the distribution of data in hidden layer
    $\Rightarrow$ suppress the change of input data distribution for the layer after it
    $\Rightarrow$ increase robustness for later layers, against covariate shift
    (from both the weight update in early layers and the input data change)
  - regularize the net by adding noise to the input data of hidden layer
    (due to computing mean/variance only on mini-batch)
    $\Rightarrow$ enforce robustness against noise, hence unintended slight regularization effect

12.3.2 Operations on Network

Initialization

- Random Initialization for Weights
  - Practice
    - weights initialized to a random variable in a small range e.g. $(-0.03, 0.03)$
  - Pros
    - avoid symmetry problem:
      if identical initialization for weights $\Rightarrow$ units in same layer computing exactly same function
      $\Rightarrow$ get the same learning step propagated back
      $\Rightarrow$ then always compute exactly the same function (by induction)
    - avoid gradient vanishing: especially for gradient of sigmoid/tanh activation
  - Cons
    - NOT concern various nets: sampling in a fixed range may not work for all nets
- Xavier Initialization for Weights
  - Practice
    - set $\forall l \in [1, L], \mathrm{Var}(w^l) = \frac{1}{n_l}$ for tanh, $\frac{2}{n_l}$ for ReLU,
      where n_l is the number of unit in layer l
  - Implementation
    - draw random variable $r \sim \mathcal{N}(0, 1)$
    - set each of $w^l = r \cdot \sqrt{\frac{2}{n_l}}$ for ReLU, $r \cdot \sqrt{\frac{2}{n_l}}$ for tanh
      or $r \cdot \sqrt{\frac{2}{n_{l-1}+n_l}}$ proposed by

- Pros
  - theoretically justified to initialized weights to be around ± 1
    $\Rightarrow$ mitigate gradient vanishing& exploding problem statistically

- Zero Initialization for Bias
  - Reason
    - default to use 0 bias
      (can NOT used for weights as explained)

Regularization

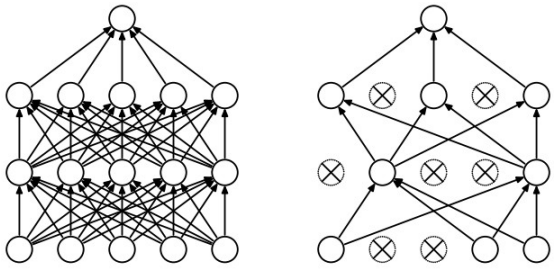
- $L2$ Regularization
  - Definition
    - $\Rightarrow$ also called "weight decay"
      (as in gradient decent, weight is multiplied by a < 1 number due to L2 term)
  - Understanding
    - forcing weights to be smaller
      - single node has smaller effect
      - input of activation closer to 0
        $\Rightarrow$ activation becomes more linear-alike (e.g. sigmoid, tanh)
        $\Rightarrow$ layers perform more linear-alike transformation

      $\Rightarrow$ simpler network, less able to fit extreme curly decision boundary
      (hence less able to overfit)

- $L1$ Regularization
  - Definition
    - for each weight ww we add the term $\lambda|w|$ to the objective. It is possible to combine the L1 regularization with the L2 regularization: $\lambda_1|w| + \lambda_2 w^2$ (this is called Elastic net regularization). The L1 regularization has the intriguing property that it leads the weight vectors to become sparse during optimization (i.e. very close to exactly zero). In other words, neurons with L1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to the "noisy" inputs. In comparison, final weight vectors from L2 regularization are usually diffuse, small numbers. In practice, if you are not concerned with explicit feature selection, L2 regularization can be expected to give superior performance over L1.
  - Practice
    -
  - Understanding
    -
  - Cons
    -

- Dropout Regularization
  - Definition
    - for each of selected units, set a drop probability
      i.e. for each forward/back-prop, nodes are "dropped" according to the probability
      $\Rightarrow$ for each time, a randomly reduced net is trained

- Implementation: Inverted Dropout
  - set a keep prob k instead of drop prob, for a selected layer
  - generate random numbers for all units & turned into a boolean "keep" vector $\mathbf{k}$
  - dropped activation $\mathbf{d} = \mathbf{a} \times \mathbf{k}$ (element-wise),
    where $\mathbf{a}$ is original activation output vector from the layer
  - $\Rightarrow$ activation becomes 0 for dropped units in $\mathbf{d}$
  - scaling up by dividing the keep prob: $\mathbf{d}/k$
    $\Rightarrow$ so that expected output value of each activation remains the same
  - test time: no dropout $\Rightarrow$ no random output & consider all robust features learned
    (randomness in training, mitigated by big data)
- Understanding
  - can NOT rely on any one feature $\Rightarrow$ have to spread out weights
    $\Rightarrow$ results in shrinking the squared norm of weights (as $L2$)
  - used on layers with enormous features as input (e.g. computer vision)
    $\Rightarrow$ reduce the chance of relying on small set of features



(a) Standard Neural Net (b) After applying dropout.

- Cons
  - training loss may have bigger glitch $\Rightarrow$ harder to debug
    (make sure loss decreasing before introduced dropout)

- Max norm constraints
  - Definition
    - enforce an absolute upper bound on the magnitude of the weight vector for every neuron and use projected gradient descent to enforce the constraint. In practice, this corresponds to performing the parameter update as normal, and then enforcing the constraint by clamping the weight vector w_n^l of every neuron to satisfy (). Typical values of cc are on orders of 3 or 4. Some people report improvements when using this form of regularization. One of its appealing properties is that network cannot "explode" even when the learning rates are set too high because the updates are always bounded.
  - Practice
    -
  - Understanding
    -
  - Cons
    -
- Early Stopping

- Definition
  - stop the training at lowest validation loss (with training loss decreasing)
    ⇒ at the start point of overfitting
- Practice
  - evaluate both train & val loss, saving models along the way
    ⇒ use the model corresponding to the start of overfitting
- Understanding
  - at relatively early stage, weights are still relatively small
    (due to random initialization in $[0^-, 0^+]$)
- Cons
  - couples task of optimizing loss and task of not overfitting
    ⇒ no longer one task at a time

Optimization

- Batch Descent
  - Practice
    - evaluate on entire training set; then update weights
  - Pros
    - largest optimization every time
  - Cons
    - greedy optimizing
    - slow & memory demanding on large dataset
- Stochastic Gradient Descent
  - Practice
    - shuffle data to have training set X_{train}, further split into $X^1_{\text{train}}, ..., X^T_{\text{train}}$
    - train the net iteratively with $\forall t \in [1, T], X^t_{\text{train}}$
      i.e. one mini-batch for a gradient descent (weights update)
    - after training through all T batches, an epoch of training is finished
      ⇒ 1 epoch = 1 full scan of training set
  - Pros
    - faster: more weight upgrade over the same amount of data
    - better chance to reach global change: not greedy anymore
    - more affordable for training in GPU memory

    ⇒ preferred choice
  - Cons
    - observing noisy loss: not monotonically decreasing (but overall decreasing)
- Gradient Descent with Momentum
  - Definition: exponentially weighted average
    - calculate the gradient for weight update: $dW'_t = \beta dW'_{t-1} + (1-\beta)dW_t$,
      where dW the original gradient
    - ⇒ average over past gradients with exponentially decaying weight,
      ⇒ for past $k \in [0, K]$ gradient, coefficient becomes $\beta(1-\beta)^k$
      (with $k = 0$ denoting current gradient)

        - bias correction: avoid slow start
          (due to: gradient dW_0 initialized to 0 & not enough gradients for averaging)
          $\Rightarrow$ set $dW_t = \frac{dW_t}{1-\beta^t}$ in the early stage
          (after starting stage, bias correction $\to 0$ for large t)
    - Implementation
        - approximation: weighted average over past $K = \frac{1}{1-\beta}$ gradients
          due to $(1-\epsilon)^{1/\epsilon} \approx \frac{1}{e}$, recognized as small enough
          $\Rightarrow$ discard gradients with further exponentially small weights
        - apply element-wise multiplication on gradients and pre-calculated coefficient
        - sum up to be the gradient for weight update
          (include bias correction term if necessary, yet often omitted)
        - note: $dW'_t = \beta dW'_{t-1} + dW_t$ is another version, yet discouraged
          (coupling momentum β with learning rate α, as α needs to cooperate)
    - Understanding
        - averaging/smoothing out the regular oscillation in stochastic gradient descent
          $\Rightarrow \beta$ popularly chosen to be 0.9 (averaging over last 10 gradients)
    - Pros
        - avoid some regular oscillation (slowing down the training & not true randomness)

- Root Mean Square Propagation (RMS prop)
    - Definition
        - compute $S_t = \beta S_{t-1} + (1-\beta) dW_t^2$ (S_0 initialized to 0),
          where dW^2 the original gradient being element-wisely squared
          $\Rightarrow$ exponentially weighted square of gradients
        - calculate the gradient for weight update $dW'_t = \frac{dW_t}{\sqrt{S_t}}$
    - Implementation
        - calculate S_t similarly (as an exponentially weighted average)
        - $\sqrt{S_t}$ becomes $\sqrt{S_t + \epsilon}$, where $\epsilon \to 0^+$ for mathematical stability
    - Understanding
        - for gradients with large variance in training $\Rightarrow S_t$ large $\Rightarrow \frac{1}{\sqrt{S_t}}$ small
          $\Rightarrow$ weighted less, hence stabilized (as it should be noisy & taking smaller step)
        - for gradients with small variance
          $\Rightarrow$ weighted more, encouraged (as it should be on the "trend" towards optimum)
    - Pros
        - recognize trend from noise via variance of their gradient $\Rightarrow$ speedup training
        - auto-fixing learning rate for each weight given the recorded behavior
          (protect learning process from a too large learning rate)

- Adaptive Momentum (Adam) Optimization Optimization
    - Definition
        - compute $M_t = \beta_1 M_{t-1} + (1-\beta_1) M_t$ as momentum
        - compute $S_t = \beta_2 S_{t-1} + (1-\beta_2) dW_t^2$ as root mean square
        - apply bias correction on both: $M'_t = \frac{M_t}{1-\beta_1^t}, S'_t = \frac{S_t}{1-\beta_2^t}$
        - $\Rightarrow$ calculate gradient for update $dW'_t = \frac{M'_t}{\sqrt{S'_t + \epsilon}}$, where $\epsilon \to 0^+$
    - Implementation

        - implement M_t, S_t as momentum and root mean square
          (popular choice: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)
        - do implement bias correction
    - Understanding
        - combine momentum with root mean square
          $\Rightarrow$ for each weight
            - smooth out regular oscillation
            - encourage the trend & adapt learning rate given history record
    - Pros
        - effective for a large range of problem

- Learning Rate Decay
    - Definition
        - update learning rate $\alpha = \frac{1}{1+r \cdot e}$, where r the decay rate, e the epoch number
        - other decay formula:
            - exponential decay: $\alpha = r^e \cdot \alpha_0$, where α_0 the base learning rate
            - $\alpha = \frac{k}{\sqrt{e}} * \alpha_0$, where k a constant
    - Implementation
        - set learning rate for each epoch, or after some global steps
    - Understanding
        - fast learning at the beginning, more cautious when approaching the optimum
          $\Rightarrow$ in order to finally converge

12.3.3 Cost

Probabilistic Cost

- Log Maximum Likelihood / Posterior
    - Definition
        - convert the logits into probability-alike prediction
          $\Rightarrow$ then interpreted as predicted likelihood $p(\mathbf{y}|\mathbf{w}, \mathbf{x})$
        - bayesian regression $L = -\frac{1}{2} \sum_{\mathbf{y} \in \mathbf{Y}} (\mathbf{y} - \hat{\mathbf{y}})^2$
          (for $\mathbf{y}$ real number vector label, $\hat{\mathbf{y}}$ real number vector prediction)
        - classification with logistic assumption $L = -\sum_{\mathbf{y} \in \mathbf{Y}} (\mathbf{y}^T \cdot \log \hat{\mathbf{y}})$
          (t one-hot encoded label, $\hat{y}$ one-hot encoded prediction)
        - to use posterior with Gaussian distribution: add $L2$ regularization term

12.3.4 Layers

Prediction

- Sigmoid
- Softmax
    - Input

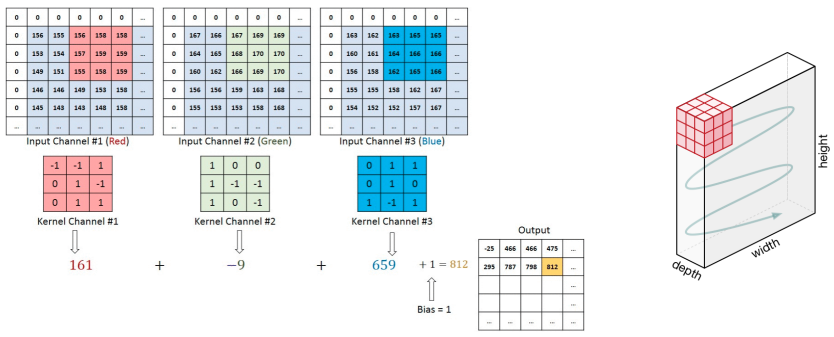
        - arbitrary input $\mathbf{z}^L$ being logits, containing multiple multi-class predictions z^L
          $\Rightarrow$ each prediction being the same dimension as one-hot encoded label
    - Output
        - probabilistic-alike prediction $\mathbf{a}^L$, with the same shape as the input (logits)
    - Operation
        - for K classes to predict $\Rightarrow \dim(z^L) = K$
        - for each dimension $k \in [1, K]$, compute $a_k^L = \dfrac{e^{(z_k^L)}}{\sum_{k=1}^{K} e^{(z_k^L)}}$
    - Implementation
        - vecotrize the exponential computation $\hat{z}^L = \exp(z^L)$
        - compute normalization $N = \sum_{k=1}^{K} \hat{z}_k^L$
        - normalize as $a^L = \frac{1}{N}\hat{z}^L$
        - maximum likelihood with softmax: $L = \dfrac{1}{N}\sum_{\mathbf{Y}} -\mathbf{y}^T \cdot \log \hat{\mathbf{y}}$,
          where $\mathbf{y}$ the one-hot encoded label, $\hat{\mathbf{y}}$ the prediction
          $\Rightarrow$ easy gradients: $dz^L = \hat{\mathbf{y}} - \mathbf{y}$, where z^L the logits (vector)
    - Understanding
        - contrasting the hard-max function (non differentiable): $a_k = 1$ if $\arg\max_k(z)$; else 0
        - exponentially normalizing the output of arbitrary net into probabilistic form
          (reduced to logistic for binary class i.e. $K = 2$)
          $\Rightarrow$ generalize logistic prediction to K-class prediction
        - for maximum likelihood loss, only the gap with true class generate gradients
          (due to one-hot encoding)
          $\Rightarrow$ trying to predict the class true with higher probability

- Hierarchical Softmax
    - uses a binary tree representation of the output layer with the W words as its leaves
      and, for each node, explicitly represents the relative probabilities of its child nodes
      $\Rightarrow$ a random walk that assigns probabilities to classes

- Normalization

Convolution Layer

- Convolution $2D$
    - Input
        - spatially $2D$ feature maps, usually with multiple channels
    - Operation
        - given hyperparameter: kernel/filter size, stride, padding
        - kernel (weights) structured as a matrix (for each input channel)
        - input maps padded if required
        - an element-wise weighted sum on the spatially corresponding position
        - sum across channels: sum over kernel output from each channel + optional bias



- kernel strides spatially across the image, with stride along each axis defined
  ⇒ to calculate **1** channel in the output feature maps
  ⇒ for multi-channels output: multiple sets of kernels
- activation then taken after convolution operation, element-wisely

◦ Padding

- reason
  · prevent output feature maps from spatially shrinking
  · prevent info lost on the edge&corner of image
  (compared to the central part of feature map, multiplied less with the kernel)
- convention: 0-padding on both directions of an axis
- valid conv: no padding
- same conv: pad so that output size same as input size

◦ Kernel Size

- odd square matrix: avoid asymmetric padding & a central pixel for filter location
  (even becomes a convention)

◦ Stride

- the step for kernel to move its location as striding over feature maps
- kernel striding over the edge (after padding): NOT convoluted

◦ Output

- a feature maps with defined output channel & size
  (output channel depends on the number of sets of kernels)
- on a $2D$ square feature map, with padding at each edge p, stride on all axises s
  kernel size $k \times k$, input size $i \times i$, output size $o \times o$
  $\Rightarrow o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$

◦ Understanding

- computer vision filter: learn the weights in filters, instead of hand design
  ⇒ guided by data statistics
- weights in the l^{th} conv layer: $n_c^{l-1} \times k \times k \times n_c^l$, where n_c the channel number
  (to output n_c^l channels with n_c^{l-1} input channels from previous layer)
  ⇒ invariant to the input size (number of trainable variables fixed on design)
  ⇒ less weights (then dense layer), more generalizability, hence less overfitting
- sharing weights spatially: apply same weights over the whole space
  ⇒ NO need for special design at each location
  ⇒ as need to handle spatial variance in processing images
- sparse connection: output connected only to the local input ⇒ as a high-pass bandwidth
  ⇒ robust to spatial variance

    - Back Propagation
    - Implementation
        - implement cross-correlation instead of convolution
          (skip the flipping operation: as their results are symmetric)
          yet, convolution is associative, due to the flipping

- 1×1 Convolution
    - Input
        - multi-dimension feature maps with multiple channels
    - Operation
        - integrate channels at each spatial location together
          (a weighted sum with bias, as conv definition)
    - Output
        - feature maps with same dimensions, but different channels
    - Understanding
        - shrink the number of channels
        - add more non-linearity & info combination (more representability)
        -

- Atrous Convolution
- Deconvolution

Pooling Layer

- Normal Pooling
    - Input Data
        - feature maps
    - Operation
        - given hyperparameter: kernel size, stride (usually no padding)
        - compute the max/average of the elements covered by kernel
        - kernel strides along each axis over the feature maps (like conv)
          $\Rightarrow$ does NOT change the channel
          $\Rightarrow$ one kernel per channel (compared to conv)
    - Output
        - a downsampled feature maps
        - given a $2D$ feature map with kernel size $k \times k$, strides along each axis $s \times s$
          input size $i \times i$, output size $o \times o$
          $\Rightarrow o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1$ (same as conv)
    - Understanding
        - downsampling the feature maps (NO weights to learn)
          $\Rightarrow$ if desired features detected anywhere, represent it by local max/average

- Unpooling
- Spatial Pyramid Pooling (SPP)
- Region of Interest Pooling (RoI Pooling)

    - Input
        - feature maps from CNN
        - RoIs i.e. proposal region (from selective search etc.) projected on feature map
    - Operation
        - divide each RoI with grid of desired size (proportional to the RoI size)
        - max pooling from each cell

      $\Rightarrow$ single-size SPP for each RoI
    - Output
        - a fixed size feature maps for each RoI
- Probabilistic Max Pooling

Residual Block

- Structure
    - Definition
        - for the $l+2$ layer, $a^{l+2} = g(z^{l+2} + a^l)$,
          where $g(\cdot)$ the activation, $z^{l+2} = W^{l+2}a^{l+1} + b^{l+2}$ from layer $l+1$
    - Main Path
        - the usual passing of a^l to layer $l+1$, then the z^{l+2} at layer $l+2$
    - Skip Connection (Shortcut)
        - the passing of a^l directly to the layer $l+2$
    - Join
        - for different size a^l, a^{l+2}: adjust a^l by linear transformation or padding
          (can be implemented as matrix/conv with trainable or fixed weights)
        - add the result from two paths $\Rightarrow a^{l+2} = g(z^{l+2} + a^l)$
- Understanding
    - Guaranteed Baseline
        - with ReLU as activation, easy to learn identity function
          $\Rightarrow$ layer $l+2$ only need to make $z^{l+2} = 0$ (as weights & bias initialized near 0)
        - $\Rightarrow$ deeper net can easily guarantee to be at least as good as its shallow version
          (then search for luck to surpass baseline)
    - More Gradient
        - gradient more easily passed to the early layers
          (as shortcuts not attenuating gradients)
        - $\Rightarrow$ early layers settle down faster $\Rightarrow$ late layers get a more consistent input

Inception Block

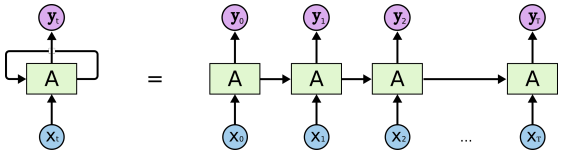
- Basic Inception
    - Input
        - feature maps with multiple channels
    - Operation
        - op1 = 1×1 conv
        - op2 = 1×1 conv, 3×3 conv with same padding

        - op3 = 1×1 conv, 5×5 conv with same padding
        - op4 = max pooling with same padding and stride 1, 1×1 conv
        - channel concate: concatenate the output channel from each op (as output size ensured to be the same in each op)
    - Output
        - feature maps with the same spatial size as input
    - Understanding
        - 1×1 conv to shrink
            - less computation for afterwards larger kernel (e.g. $3 \times 3, 5 \times 5$)
            - prevent output of other ops from being overwhelmed by pooling
        - enable network to learn the desired combination of info (instead of predefined hyperparamter)
          $\Rightarrow$ more representability

- Xception
    -

RNN Layer

- Overview
    - Input Data
        - sequence data: includes time/precedence (conventionally arrived from left to right)
        - for each time step, data can be vector, feature maps, etc...
    - RNN Cell
        - consume the input of current time step & the hidden state from last time step (hidden state usually initialized to $\mathbf{0}$)
        - calculate a hidden state at each time step
    - Operation
        - RNN cell at time t, calculate hidden state (activation) $h^t = g_h(w_h[h^{t-1}, x^t] + b_h)$, where $g_h(\cdot)$ the activation function, $g_h = \tanh$ by convention
          $[h^{t-1}, x^t]$ the concat of h^{t-1} (hidden state of time $t-1$), x^t (input at time t)
        - expose its hidden state at each time steps
        - calculate its output $y^t = g_y(w_y h^t + b_y)$, where $g_y = \sigma, softmax$ or $identity$

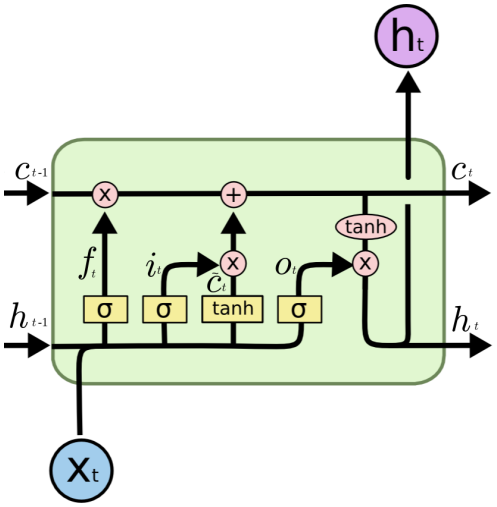


    - Types of RNN Mapping
        - many-to-one: encoder scans through the input, only the last output considered
        - one-to-many: decoder with single input x^1, take $x^t = y^{t-1}$, till $y^{t'} = \text{stop}$
        - many-to-many: a many-to-one encoder, followed by a one-to-many decoder
          $\Rightarrow$ able to map between various length
    - Back Propagation through Time
        - unroll the recurrent operation into a sequential network with length T

- given the loss for each time step $L^1, ..., L^T \Rightarrow L = \sum_{t=1}^{T} L^t$
- for time $t = 1, ..., T-1$, $\frac{\partial}{\partial a^t} L = \frac{\partial}{\partial a^t} L^t + \sum_{t'=t+1}^{T} \frac{\partial L^{t'}}{\partial a^{t+1}} \frac{\partial a^{t+1}}{\partial a^t} = \sum_{t'=t}^{T} \frac{\partial}{\partial a^t} L^{t'}$

- Truncated Back Propagation through Time
- Challenge
  - bad at modeling longterm dependency due to gradient vanishing problem
    $\Rightarrow$ loss at late time needs to go through multiple activations to the early time (similar to the deep plain net, after unrolled)
    $\Rightarrow$ loss at late time are hard to affect weights when evaluated at early time (i.e. larger the t', smaller the $\frac{\partial}{\partial a^t} L^{t'}$)
    $\Rightarrow$ hard to find out error in late time due to observation in early time
    $\Rightarrow$ hard to represent longterm dependency (e.g. Car...is fast vs. Cars...are fast)
  - easily affect by local dependency (as longterm dependency lost)
  - gradient exploding, due to multiple / too many updates on the same weights (solved by gradient clipping)
- Understanding
  - sharing weight across time: same weights used on each time step
    $\Rightarrow$ solve various length input by applying weights recurrently on each of them
  - information early in the sequence reserved & passed through in the hidden state

- Long Short Term Memory (LSTM)
  - Memory Cell
    - c^t the memory maintained by LSTM cell at time t
  - Gates
    - forget gate $G_f = \sigma(w_f[h^{t-1}, x^t] + b_f)$
    - input gate $G_i = \sigma(w_i[h^{t-1}, x^t] + b_i)$
      $\Rightarrow$ together control the memory update (how past&current info fused)
    - output gate $G_o = \sigma(w_o[h^{t-1}, x^t] + b_o)$
      $\Rightarrow$ control the generation of hidden state
  - Fusing Info
    - propose candidate $\hat{c}^t = \tanh(w_c[c^{t-1}, x^t] + b_c)$ for memory update
    - update memory as $c^t = G_f c^{t-1} + G_i \hat{c}^t$
  - Hidden State (Activation)
    - generate as $h^t = G_o \cdot \tanh(c^t)$

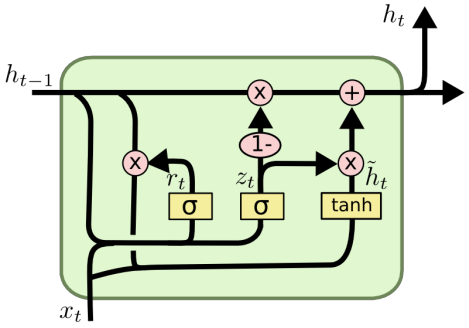


where,
joint arrows denotes a concatenation
$c_{t-1}, \tilde{c}_t, c_t$ the previous/candidate/current memory;
h_{t-1}, h_t the previous/current hidden state;
f_t, i_t, o_t the forget/input/output gate

- Understanding
  - easy to learn an identity mapping $c^{t-1} \to c^t$
    $\Rightarrow$ memory (info) generated early can last for long term
    $\Rightarrow$ better model the long-term dependency

- Gated Recurrent Unit (GRU)
  - Memory Cell
    - c^t the memory maintained by GRU cell at time t
  - Gates
    - relevance gate $G_r = \sigma(w_r[c^{t-1}, x^t] + b_r)$, a mask $\in [0, 1]$
      $\Rightarrow$ control how memory candidate proposed (fusion of last memory & input)
    - update gate $G_u = \sigma(w_u[c^{t-1}, x^t] + b_u)$, a mask $\in [0, 1]$
      $\Rightarrow$ control how memory update happen (fusion of last memory & candidate)

    (two gates computed with the same input, though with different weights)
  - Fusing Info
    - propose memory candidate $\hat{c}^t = \tanh(w_c[G_r c^{t-1}, x^t] + b_c)$ for the update
      $\Rightarrow$ decide whether the previous memory useful (relevant) with the context x^t
    - update memory as: $c^{t+1} = G_u \hat{c}^{t+1} + (1 - G_u)c^{t-1}$
      $\Rightarrow$ decide how the memory updated & remained
  - Hidden State (Activation)
    - $h^t = c^t$

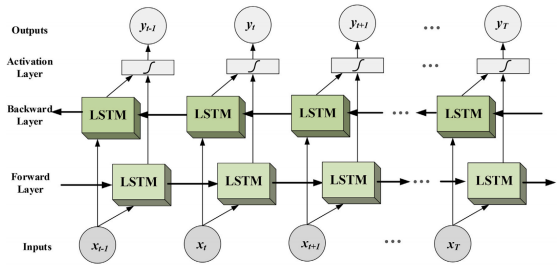


where,
joint arrows denotes a concatenation
$h_{t-1}, \widetilde{h}_t, h_t$ the previous/candidate/current memory;
r_t, z_t the relevance/update gate

  - Understanding
    - single gate control the generation of current memory
    - memory directly as hidden state
    - $\Rightarrow$ less weights, simpler structure $\Rightarrow$ faster
      (basic logic inherent from LSTM $\Rightarrow$ similar performance)

- Bidirectional RNN (BRNN/BiRNN)
  - Structure
    - one RNN layer scanning as $t = 1 \to T$, hidden state exposed as h_1^t
    - another RNN layer scanning from $t = T \to 1$, hidden state exposed as h_2^t
    - generate output $y^t = g(w_y[h_1^t, h_2^t] + b_y)$
      (concat all hidden states at the same time step from each RNN)
  - Understanding
    - account for the info from both previous & latter time step
      $\Rightarrow$ global context info acquired
    - cons: need the entire input sequence before processing
      $\Rightarrow$ NOT the case in real-time speech recognition etc.

- Convolutional LSTM (ConvLSTM)



(using LSTM cell)

12.4 Architectures

12.4.1 Convolutional Networks

Classic Convolutional Networks

- Overview
  - Convolution Part
    - one or multiple "same" conv layer(s) with stride $s = 1$
    - followed by a max (rarely average) pooling
    - apply on input & repeat on feature maps afterwards
      $\Rightarrow$ usually decreasing spacial size (width, height), increasing channel
  - Fully Connected (Dense) Part
    - apply flattening / average pooling on last feature maps
      $\Rightarrow$ generate a single vector as input
    - apply fully connected layers & repeat for 1 2 times
      $\Rightarrow$ usually with decreasing size
    - finally output prediction probability
  - Understanding
    - trainable weights are mostly from dense layer
      $\Rightarrow$ conv layers, though large in number, contains far less weights
    - conv stride $s = 1$ (compare to $s > 1$)
      $\Rightarrow$ decouple downsampling into the pooling
      $\Rightarrow$ account for more info/possibility before downsampling
      (also trade-off between required computability)
- Receptive Fields
  -
- VGG-16
  - Building Blocks
    - fixed conv: 3×3 kernel, stride $s = 1$, same padding, ReLU activation
    - fixed pooling: 2×2 kernel, stride $s = 2$, max pooling
    - Notation
      - [conv64] to denote a conv layer with 64 output channels
      - pool to denote max pooling
      - [fc4096] to denote a fully connected layer with output vector of length 4096
    - Structure

        - input image: $224 \times 224 \times 3$
        - ([conv64]×2, pool) → ([conv128]×2, pool) → ([conv256]×3, pool) → ([conv512]×3+pool)×2
        - last feature maps (7×512) flatten into a input vector of length 4096
        - [fc4096] → [fc4096] → logits → softmax prediction for 1000 categories
    - Understanding
        - fixed conv & pooling operation
          ⇒ few hyperparameters, yet large in parameters (∼138 million)
        - decrease spatial size by 2 & increase channel by 2 on each step
          (till small or deep enough e.g. 7×7 size, 512 channels)
- ResNets
    - Building Blocks
        - residual block
        - batchnorm applied in the conv(s) inside residual block
    - Structure
        - stack of residual block, for hundreds, even thousands of layers
- Inception Net
    - Building Blocks
        - inception block
    - Structure
        - stack of inception blocks
        - two auxiliary loss from hidden layers

Fully Convolutional Networks (FCN)

- FCN for Classification & Regression
    - Convolution as Flattening
        - given input feature maps $i \times i \times c$, with c channels
        - perform valid conv with kernel $i \times i \times o$
          ⇒ output size $1 \times 1 \times o$
    - 1×1 Convolution as Fully Connected Layer
        - for flatten vector $1 \times 1 \times i$, perform 1×1 conv with kernel size $1 \times 1 \times 0$
          ⇒ output size $1 \times 1 \times o$, with same computation as fc layer
    - Use Case
        - object detection
-

12.4.2 Recurrent Neural Network

Classic RNN

- Overview
    - Stacked RNN Layers
        - the output of layer l becomes the input for layer $l + 1$
          ⇒ RNN scanning the output of previous RNN layer
        - $3 \sim 5$ layer considered deep: as RNN can be unrolled into a deep plain net
    - Encoder-Decoder RNN
        - encoder RNN scans the input sequence, last hidden layer as sequence encoding
        - decoder RNN takes encoding as initial hidden state, unrolled the output sequence

Attention

- Overview
  - Goal
    - overcome long-term dependency, via considering all input with different attention
  - Practice
    - a bi-RNN encode rich input context at each hidden state: $h^1, ..., h^{T_x}$
    - global context after attention at time t: $c^t = \sum_{i=1}^{T_x} \alpha_i^t h^i = [h^1, ..., h^{T_x}]\alpha^t$,
      where hidden states $h^1, ..., h^{T_x}$ and attention α^t column vectors
      $\Rightarrow$ a weighted sum over all hidden states as context
    - a small (1-layer) net mapping $[c^{t-1}, h^i] \xrightarrow{\text{dense}}$ logits $\xrightarrow{\text{softmax}} \alpha_i^t$
      $\Rightarrow$ attention on h^i depends on previous global context c^{t-1} & h^i itself
      (softmax to ensure $\sum \alpha^t = 1$)
    - a decoder RNN taking c^t as input, until stop sign generated
      (as c^t can be calculated infinite times)
  - Understanding
    - use the sequence of hidden states as encoding
      (as sentence encoded in last hidden states can be bias & have info lost)
      $\Rightarrow$ account input sequence (with different attention at different places) when generating output sequence
    - quadratic cost: attention α a $T_x \times T_y$ matrix to be calculated
- Attention on Image
  - Goal
    - focus on correct spatial context when generating corresponding caption/sequence
  - Practice
    - calculate weights for a sum over all spatial location of the image encoding
      $\Rightarrow$ calculate a 2-D mask over a 3-D tensor

Transformer

12.4.3 Encoder-Decoder Architecture

Basic

- Encoder
  - Functionality
    - downsample/encode input into rich feature maps/vectors
  - Implementation
    - visual input: CNN backbone
    - natural expression input: RNN backbone
- Decoder
  - Functionality
    - upsample/decode rich feature maps back to the original size
    - actually, impose requirement onto the encoder

    - Implementation
        - visual output: CNN backbone
        - natural expression output: RNN backbone
- Connection
    - Functionality
        - combine high level information with low level information
        - image → image: outline refinement ...
        - language → language: sentence style capturing
    - Implementation
        - concatenation

Extension

- Multiple Encoder
    - Functionality
        - project different information into the same space
        - combine those information via some shared layers at the end
- Multiple Decoder
    - Functionality
        - impose multiple requirements to the encoder (via auxiliary loss)
- Variational Auto-Encoder

12.4.4 Generative Network

Generative Adversarial Network

12.5 Computer Vision

12.5.1 Objects Detection

Problem Formulation

- Input Data
    - Spatial Information
        - image maps as 3-D tensor: width, height, channel (rgb, etc.)
        - multi/stereo -images for 3-D detection
- Goal
    - Bounding Box Prediction
        - localization as regression task on box attributes (x, y, h, w)
        - classification on object classes & object existence
    - Landmark (Key Point) Prediction
        - landmarks: the coordinates of key points (of each type) in image
        - label of landmark should be consistent across image
        - output real number as regression task
          ⇒ used in pose detection, bounding box detection, etc.
- Understanding
    - multi-object localization & classification

Common Postprocessing

- Non-max Suppression
  - Input
    - multiple predicted bounding boxes, with probability of existence (p_e) relatively large
  - Goal
    - clean up & account for duplicate bounding boxes on the same object
      ⇒ for each (predicted) object, finalize prediction to be a single bounding box
  - Naive Operation
    - choose the bounding boxes with highest (maximal) p_e (for each object classes)
    - remove those bounding boxes whose IoU with it are large
      ⇒ remove (suppress) bounding boxes with large overlap
    - repeat until all bounding boxes have a low IoU with each other
      ⇒ remaining boxes are the final predictions

Classic Approaches

- Sliding Window Detection
  - Bounding Box Proposal
    - slide the window with various size, across the image
      ⇒ propose bounding boxes with predefined size & location
    - feed the window into CNN for classification
      ⇒ CNN as classifier, window as bounding box
  - Fast Implementation: Sliding Window as Convolution
    - implement CNN classification as FCN
      ⇒ as conv independent from input size
    - run directly the CNN on the input image (instead of on each sliding window)
      ⇒ output size $m \times n \times k$,
      where $m \times n$ the total number of sliding windows on the image, k the class number
    - ⇒ one times running for all sliding windows
      (as sliding window essentially is a crop from image)
  - Understanding
    - though with fast implementation, still not promising regarding result
      ⇒ sliding window propose a fixed set of window
      ⇒ fail if not matching window size; or missing location/rotation
- You Only Look Once (YOLO)
  - Preprocessing
    - grid the input image into disjoint cells
      (apply usually fine grid, e.g. 19×19 girds)
    - define a serious of anchor box: covering most of the interested objects
      (e.g. tall&thin box for pedestrian, low&fat box for car)
    - assign the objects to a tuple (cell, anchor box)
      - assign to cell by its central point ⇒ ensure maximal **1** object in a grid
      - assign to one from B anchor boxes depending on its IoU with all B boxes
  - Anchor Box Selection

- run K-means on all clabel box; use cluster center as the anchor box

- Bounding Box Encoding
  - p_e probability of existence for current anchor box in a cell
  - b_x, b_y the box location offset relative to the top-left coord of the cell
    $\Rightarrow$ easier to learn, as dense layer not confused by varying locations
  - b_h, b_w the box size relative to the size of the cell (allowed to go outside the grid)
    $\Rightarrow$ cope with various girding strategy
  - $p_1, ..., p_C$ the classification for object inside current box
  - $\Rightarrow a_b = [p_e, b_x, b_y, b_h, b_w, p_1, ..., p_C]$ the anchor box b
    (note: localization&classification considered only when $p_e = 1$)
  - $\Rightarrow [a_1, ..., a_B]^T$ the prediction/label in each cell
    (hence, able to match multiple anchor boxes at the same time in one cell)

- Prediction
  - each cell predict bboxes with their p_e
  - each cell predict conditional probability p(class|obj)
    i.e. the class probability of object, given there is (part of) an object in the cell
  - final prediction of each bbox in each cell: $p(class, obj) = p_e \times p$(class|obj)

- Bounding Box Prediction
  - apply FCN on each cell for classification&localization
    (implemented as fast sliding window)
    $\Rightarrow$ output spatial shape = grid shape; channel = bounding box len
  - get some predicted bounding boxes in each cell (e.g. fixed to n box per cell)
  - non-max suppression used to finalize predicted bounding box (for each object class)

- Structure
  - passthrough layer (skip connection) as downsampling
    $\Rightarrow$ early layer ($26 \times 26 \times 512$) stacks its spatially adjacent features together $\Rightarrow$ form a $13 \times 13 \times 2048$ feature map; then concat to late layer
    (a modest 1% performance increase)
  -

- Training
  - optimize the sum-squared error (due to its simplicity)
  - emphasize the label containing box (object)
    $\Rightarrow$ avoid large num of labels without object pushing weights to 0
  - predict the $\sqrt{b_w}, \sqrt{b_h}$, instead of b_w, b_h
    $\Rightarrow$ reflect that small deviations in large boxes matter less than in small boxes
  - for each bbox predicted by a cell
    - increase the confidence & adjust coords for box overlapping the most with label
    - decrease the confidence for box not overlapping with any label
  -
  - data augmentation: mix classification & detection dataset
    $\Rightarrow$ train the same backbone with different prediction branch

- Understanding
  - multiple anchor boxes: able to predict multiple boxes

        - decoupled anchor boxes & object class: able to predict multiple same-type objects (have to use different anchor box)
        - ⇒ able to handle: multiple objects, each in a different anchor box in one cell
        - yet, NOT able to handle: multiple objects in same-type anchor box in one cell (NOR when objects more than total anchor boxes)
          ⇒ suffer from small objects in group (e.g. birds)
        - griding: to increase the chance of predicting
          ⇒ avoid the failed case (i.e. multiple object in same-type anchor box in one cell)
        - with FCN implementation, receptive fields not restricted by grid
          ⇒ global knowledge utilized for prediction corresponding to each cell
        - one scan to have predictions for all cell ⇒ **fast** & enable real-time application

- Region Proposal
    - Goal
        - effectively propose a various bounding boxes on the image for potential focused object
          ⇒ recall all focused object with preferably fewer boxes (e.g. $\sim$ 1k)
    - Input
        - segmentation mask by a segmentation network
          ⇒ to place bboxes on high-probability blobs

- R-CNN
    - Overview
        - region proposal by selective search
        - CNN to predict class & a bounding box (to refine)

- Fast R-CNN
    - Overview
        - based on r-cnn
        - convolution implementation to predict all proposed regions (similar to fast sliding window)

- Faster R-CNN
    - Overview
        - based on fast r-cnn
        - use CNN for segmentation & region proposal (faster than selective search)

Research Direction

- Faster YOLO
    - Dynamic Grid
        - similar to spatial pyramid pooling: griding with respect to input image size
          ⇒ larger image more grid
        - ⇒ ultimately, pixel grid: each pixel as a grid cell
    - ⇒ Segmentation as Detection
        - for output mask, giving classification (p_e) + localization (regression) (instead of giving class probability as per-pixel classification)
        - at most as many objects as pixel number
          ⇒ not possible to lose detection due to griding

12.5.2 Face Recognition

Problem Formulation

- Input Data
  - Image
    - image from camera
- Goal
  - Identity Recognition
    - recognize the identity of the face in image
    - refuse to recognize if the face does NOT belongs to any stored identity
  - Liveness Detection
    - make sure the face in image are from a live human (instead of picture etc.)
- Face Verification
  - Input
    - image from camera & identity
  - Goal
    - true/false, regarding whether the image content belongs to the identity
- Challenge
  - One-shot Learning
    - given only single (at most, few) face-identity pair for each identity
    - still, need to build a robust system for recognition task

Classic Approach

- Siamese Network as Encoder
  - Structure
    - CNN + dense layer to encode the input image x^i as a vector $f(x^i)$
  - Learning Goal
    - minimize $\|f(x^i) - f(x^j)\|^2$ if x^i, x^j from same identity
    - maximize $\|f(x^i) - f(x^j)\|^2$ if x^i, x^j from different identity
    - $\Rightarrow$ learning encoding given a fixed distance function $d(x_1, x_2) \geq 0$ (here, $d(x_1, x_2) = \|x_1 - x_2\|^2$)
  - Triplet Loss
    - given an anchor image A representing the identity I
    - take a positive image P from identity I; an negative image N not from identity I
    - $\Rightarrow L(A, P, N) = \max\left(d(f(A), f(P)) - d(f(A), f(N)) + \alpha, 0\right)$, where α an hyperparamter to make sure the net differentiate them by a margin; max() to make the loss = 0 as long as the requirement satisfied
  - Training: Hard Negative Mining
    - due to large variance in the dataset $\Rightarrow d(A, P) << d(A, N)$ in most case
    - due to large number of identities $\Rightarrow$ permutation explosion

        - ⇒ evaluate current net on dataset, use mistakes for the next epoch
    - Understanding
        - learn a encoder towards a selected distance function
          ⇒ use permutation to have more training examples
        - able to precomputing the encoded vector for fast recognition

- Encoder with Binary Classification
    - Structure
        - still, CNN + dense layer to encode input image
    - Learning Goal
        - given two encoded vectors, another net (or logistic regression) to perform binary classification
          ⇒ 1 for two image has same identity; 0 for different identities
    - Understanding
        - still, utilize permutation for larger training set
          (use pair, instead of triplet)
        - learn the similarity function as well: output directly the result of comparison

12.5.3 Image Style Transfer

Problem Formulation

- Input Data
    - Content Image C
        - the image containing the spatial info (content)
    - Style Image S
        - the image containing the style of presenting the content
- Goal
    - Generated Image G
        - a image with content from C drawn in style of S

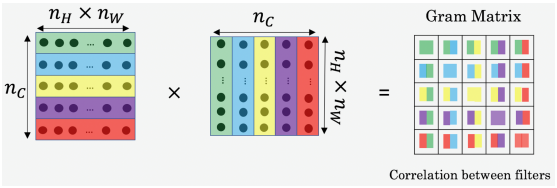
Classic Approach

- Neural Style Transfer
    - Learning Goal
        - given input C, S with output G, loss $L = \sum_l [\alpha L_{\text{content}}(C, G) + \beta L_{\text{style}}(C, S)]$, where l is sum over chosen hidden layers of the CNN
        - ⇒ minimize content & style difference
    - Content

- given input, the activations from a set of (hidden) layers of the net
- $\Rightarrow$ similarity of C, G measured as $\sum_l d(a^{l(C)}, a^{l(G)})$,
  where $a^{l(\cdot)}$ the feature maps at layer l given the input, $d(\cdot)$ a distance function
  (e.g. $d(x_1, x_2) = \|x_1 - x_2\|^2$)

◦ Style

- given input, the correlation between activations across channels, for chosen layers
  $\Rightarrow$ correlation matrix across feature map at each channel as style matrix
  (actually, gram matrix)
- let $a^l_{i,j,k}$ the activation at a $h \times w \times c$ conv kernel location i, j, k in layer l
  $\Rightarrow$ style (gram) matrix $M^l_{k,k'} = \sum_{i,j} a^l_{ijk} \cdot a^l_{ijk'}$, for all $k, k' \in \{1, ..., c\}$
- $\Rightarrow$ similarity of S, G measured as $\sum_l \left[\frac{1}{(2h^l w^l c^l)^2} d(M^{l(S)}, M^{l(G)})\right]$
  where $M^{l(\cdot)}$ the gram matrix at layer l given the input, $d(\cdot)$ a distance function,
  with normalization term $\frac{1}{(2h^l w^l c^l)^2}$
  (e.g. $d(x_1, x_2) = \|x_1 - x_2\|^2_F$, the euclidean norm between matrices)



◦ z

12.5.4 Point Cloud Recognition

12.6 Natural Language Processing

12.6.1 Language Representation

Problem Formulation

- Input
  - Language Token/Corpus
    - words, sentences, paragraphs, ... $\Rightarrow$ can be language at various level
- Goal
  - Distributed Vector Representations as Embedding Matrix $E_{M \times N} = [e_1, ..., e_N]$
    - e the column vectors, M the desired embedding length, N the total tokens num
      $\Rightarrow$ look up for the desired embedding
      (NOT using matrix multiplication due to sparsity from one-hot encoding)
    - distributed representation: decomposed yet meaningful
      $\Rightarrow$ fight the curse of dimensionality
  - $\Rightarrow$ Meaningful Vector
    - able to measure the (dis-)similarity of between tokens (words)
      $\Rightarrow$ semantic meaning: "Germany"-"Berlin" & "France"-"Paris"
      $\Rightarrow$ syntactic meaning: "quick"-"quickly" & "slow"-"slowly"
      (e.g. $e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{queen}}$, where e_{text} the embedding for word "text")
    - $\Rightarrow$ allow NLP model to be more robust & generalize better

- Challenge
  - Problem of Bias
    - word embedding reflect biases of text used to train the model
      e.g. "father-doctor" as "mother-nurse" ⇒ gender bias
    - ⇒ can cause discrimination when making decision

Overview

- Character Embedding
  - One-hot Encoding
    - a one-hot vector with length 26
- Word Embedding
  - Word Dictionary
    - a collection of high-frequency word, embedded as one-hot vector
    - special token <UNK> for unknown word
  - Features from Rule Model
    - number at each vector location denotes the score for the word matching a rule
      (e.g. location for "is_food" contains score $s \to 1$ for "apple", $s \to 0$ for "man")
  - Part-of-Speech (POS) Tag
    -
  - Word2Vec Embedding
    - construct supervised learning from UNlabeled corpus
  - Global Vector for Word Embedding (GloVe)
    - linear model with simple optimization goal
  - RNN Encoder
    - apply RNN model as encoder on characters in the word
      ⇒ no more ¡UNK¿ or unknown word
- Sentence/paragraph Embedding
  - RNN Encoder
    - apply RNN model as encoder on words in the sentence
      (last hidden layer as encoding)

Word2Vec Embedding

- N-gram Model
  - Learning Objective Setup
    - simple network to predict the $N + 1^{\text{th}}$ word given previous N words as input
      (e.g. using single softmax layer)
  - Practice
    - E randomly initialized & all words in corpus encoded in one-hot vector
    - forward prop: word in one-hot $\to$ lookup $E \to$ linear layer $\to$ softmax to predict
    - training: update linear layer parameters & matrix E as weights
      (with cross-entropy loss)

- Understanding
  - learn $P(t|c)$, where t the target word, c the previous N context words
  - setup a even larger training set from a large corpus
- Generalization
  - more context: take input from both previous and after words
  - less & close context: take only the last word as input ⇒ 1-gram model

- Skip-gram Model
  - Learning Objective Setup
    - choose only 1 single word as context word
      ⇒ balance sampling w.r.t. word frequency (e.g. prevent tons of "the", "a", ...)
    - randomly choose other word(s) in the sentence as target word(s)
    - ⇒ to predict target word(s) given only context word as input
      ⇒ learn word vector representations that are good at predicting the nearby words
  - Practice
    - same as N-gram model
  - Understanding
    - harder supervised learning task, yet goal is to learn E
    - better reflect the statistic: similar word appear in similar context
      (e.g. "soviet"-"union" appears much more often than "soviet"-"sasquatch")
      ⇒ embedding for similar target word adjusted with similar gradients
      ⇒ lie closer in vector space
    - cons: softmax over large word dict ⇒ low computation
      ⇒ mitigated by hierarchical softmax, noise contrastive estimation (NCE)

- Skip-gram with Negative Sampling
  - Learning Objective Setup
    - choose a pair of context and target word (c, t) as positive example
    - generate k negative examples by: same context word c & random word t' as target
    - given a pair of words, binary classification: is a (context, target) pair?
      ⇒ distinguish valid target word from k draws from noise distribution
  - Practice
    - detect meaningful (c, t) pair/phrase by heuristic method
      e.g. if c, t co-appear within 10-words distance more than a threshold, etc.
    - $k = 5 - 20$ for small train set; $k = 2 - 5$ for large train set
      (larger noise to avoid overfitting)
    - sample random word t' from modified uniform distribution $\frac{1}{Z}U(t)^{3/4}$ over words
    - subsample frequent words: sampled t' discarded by probability $P(t') = 1 - \sqrt{\frac{thr}{f(t')}}$
      where thr a threshold, $f(t')$ the frequency of t' in corpus
      (to avoid meaningless words like "the", "a", etc.)
    - E randomly initialized & all words in corpus encoded in one-hot vector
      (forward prop similarly)
  - Understanding
    - learn $P(y = 1|c, t)$ via logistic regression
      ⇒ much computationally affordable compared to giant softmax (less weights)
      ⇒ much simpler approach than hierarchical softmax & NCE
    - non-linear model (logistic reg) also prefers linear structure of word embedding
      ⇒ cosine distance still measures (dis-)similarity

Global Vector for Word Embedding (GloVe)

- Overview
  - Context-Target Matrix X
    - x_{ij}: the count of times word w_i appear in the context of word w_j (context definition can be non-symmetric)
  - Learning Objective Setup
    - given embedding matrix E, minimize $\sum_{i,j} f(x_{ij})(\theta_i^T e_j - \log x_{i,j})^2$, where e_j the embedding for w_j, θ_i the weights associated with w_i
    - $f(x_{ij})$ a weighting term to balance infrequent-frequent words ($f(x_{ij})$ for $x_{i,j} = 0$, preventing $-\inf$ from $\log 0$)
  - Practice
    - gradient decent directly optimize the simple objective
    - final embedding for word $w, w_e = \frac{1}{2}(e_w + \theta_w)$ $\Rightarrow$ as θ_w, e_w in objective interchangeable
  - Understanding
    - directly model the linear structure in word representation (project input e directly to output $\theta^T e$)
    - final linear structure probably NOT align with human interpretable axis $\Rightarrow$ yet probably a combination of them (from a higher view)

Addressing Bias in Word Embedding

- Overview
  - Input Data
    - a trained word embedding
  - Goal
    - identify the bias in embedding
    - eliminate the bias if it appears in undesired places
  - Identify Bias Direction
    - singular value decomposition to identify the axises where biases lie (similar to a PCA)
    - e.g. principle component of $e_{\text{man}} - e_{\text{woman}}, e_{\text{male}} - e_{\text{female}}, \ldots$
  - Neutralize
    - for all NOT definitional word (where bias should NOT appear) $\Rightarrow$ project to axises orthogonal to bias axises (to get rid of bias)
    - e.g. project e_{doctor} to the axises to reduce component in bias axises
  - Equalize Pairs
    - for all definitional word (where bias should appear) $\Rightarrow$ adjust their distance towards non-definitional word to be the same (may train/handpick all definitional words, which is only a small set)
    - e.g. make sure $d(e_{\text{boy}}, e_{\text{doctor}}) = d(e_{\text{girl}}, e_{\text{doctor}})$

12.6.2 Language Modeling

Problem Formulation

- Input Data
  - Sequence
    - a word with sequence of characters
    - a sentence with sequence of words/characters
    - a paragraph with sequence of sentences/words/characters
- Goal
  - Probability Distribution
    - model the appearance probability of the sequence $P(\mathrm{z}^1, ..., \mathrm{z}^t)$, where z^t the token at time t

Classic Approach

- RNN
  - Practice
    - at each time, output token distribution conditional on previous token(s) $\Rightarrow y^t = p(z^t|z^1, ..., z^{t-1})$, where z^t the token at time t
    - $\Rightarrow$ sequence probability $P(z^1, ..., z^t) = P(z^1)P(z^2|z^1)...P(z^T|z^1, ..., z^{T-1}) = \prod_t^T y^t$
  - Inference
    - $\mathbf{0}$ vector as both (initial) hidden state & input at time 0 $\Rightarrow$ estimate $y^1 = p(z^1)$, the distribution for being the 1^{st} token
    - for time $t = 2, ..., T$, take input $x^2, ..., x^T$ with hidden state $h^1, ..., h^{T-1}$ $\Rightarrow$ estimate each conditional distribution (conditioning by passing hidden state)
  - Training
    - for time 1, input $x^1 = \mathbf{0}$, previous hidden state $h^0 = \mathbf{0}$
    - for time $t = 2, ..., T$, input $x^t = z^{*t-1}$ the true token of $t-1$ in the given sequence
  - Generative Model: Sampling New Sequence
    - sampling the first token $\hat{z}^1$ according to the distribution y^1
    - for $t = 2, ...$, take input $x^t = \hat{\mathrm{z}}^{t-1}$, the token sampled from y^{t-1}
    - until $t > T$ or the end signal sampled (e.g. the period "." in a sentence)

Masked Language Model

12.6.3 Name-Entity Recognition

-

12.6.4 Sentiment Classification

Problem Formulation

- Input Data
  - a sentence / paragraph
- Goal
  - predict the degree of positive/negative attitude
- Challenge
  - small training set

Classic Approach

- Many-to-one RNN
  - Practice
    - each word encoded by word embedding
    - RNN scanning through paragraphs
    - last hidden layer as paragraph representation & used to classify/regress

12.6.5 Neural Machine Translation

Problem Formulation

- Input Data
  - Sequence
    - typically sentence, can be also multiple sentences (paragraph)
- Goal
  - Sequence
    - generated sentences in desired language

Classic Approach

- RNN Encoder-Decoder
  - Overview
    - an RNN encodes the input sequence by its last hidden layer
    - input encoding used as initial hidden state for decoder RNN
    - decoder RNN generates (conditional) distribution over words at each step
      $\Rightarrow$ for time $t, y^t = p(z^t|x^1, ..., x^{T_x}, \hat{z}^1, ..., \hat{z}^{t-1})$,
      where z^t the token at time t, $\hat{z}^1, ..., \hat{z}^{t-1}$ the tokens chosen from $y^1, ..., y^{t-1}$
      (z^t a random variable, $\hat{z}^t$ a concrete assignment, y^t a conditional distribution)
    - unroll until stop sign generated
  - Understanding: Conditional Language Model
    - decoder functions like language modeling, only different in its initial hidden state
    - $\Rightarrow$ measure the conditional distribution $p(z^1, ..., z^{T_y}|x^1, ..., x^{T_x}) = \prod_{t=1}^{T_y} y_t$,
      where $z^1, ..., z^{T_y}$ the generated sequence, $x^1, ..., x^{T_x}$ the input sequence
  - Improvement
    - combined with attention model

Choosing Output Sequence

- Greedy Search
  - Practice
    - choose the words of highest (conditional) probability at each time step
- Beam Search
  - Practice
    - with a vocabulary size of N, a bean with size b, input sequence $\mathbf{x} = x^1, ..., x^{T_x}$
    - to start, choose top-b tokens (among N tokens) at the 1^{st} step
    - for step t, input each previous stored b tokens to have b conditional distributions
    - choose the top-b token pairs (among $b \times N$ pairs) regarding joint probability $\Rightarrow P(z^1, ..., z^t|\mathbf{x}) = P(z^t|z^1, ..., z^{t-1}, \mathbf{x})P(z^1, ..., z^{t-1}|\mathbf{x})$
  - Normalization by Length
    - reason: short sequence with less $y^t \in [0, 1] \Rightarrow$ larger in general
    - choose t^{th} pair regarding the normalized probability $\frac{1}{t}P(z^1, ..., z^t|\mathbf{x})$
    - $\Rightarrow$ more numerically stable
  - Understanding
    - approximately search the sequence with highest joint (conditional) probability $\Rightarrow$ try to maximize $P(z^1, ..., z^{T_y}|x^1, ...x^{T_x})$
    - similar to viterbi algorithm in HMM $\Rightarrow b = 1$ reduce to greedy search
    - B usually chosen in 10 in research, > 1000 in commercial system (still faster than BFS/DFS, yet no guarantee on finding best result)
-

12.6.6 Speech Recognition

Problem Formulation

- Input Data
  - Sequence
    - an audio sample, with each frame as a time step
- Goal
  - Sequence
    - text (words/sentences) corresponding to the audio
- Challenge
  - Variable Timing
    - output (letter/words) usually has much less time steps than input (audio frames) $\Rightarrow$ multiple input time steps corresponding to same output time step

Learning Objective

- Connectionist Temporal Classification (CTC) Loss
  - Goal
    - avoid learning boundaries and timings
  - Practice
    - two sequences considered equivalent if they differ only in alignment, ignoring blanks
      ⇒ remove duplicate token (e.g. letters) from both sequence before comparison

Classic Approach

- RNN Encoder-Decoder
  - Overview
    - encoder scan through audio frames & decoder output letter/punctuation/"blank"/"space"
    - "blank": no symbol v.s. "space": delimiter for letters → words

Trigger Word Detection

- Goal
  - trigger word: a specific predefined audio signal to invoke system (e.g. xiaodu xiaodu)
  - detect where trigger word included in an audio (if any)
- Labeling
  - 0 for frames not corresponding to trigger word; 1 for frames consisting trigger words
  - upsampling positive example: extends 1 label a few frames after the trigger words (as trigger word often appears once in an interaction with system)
- Classic Approach
  - RNN Encoder-Decoder
    - encoder scan through audio & decoder output 0-1 classification at each step
  - Conv RNN
    - a fixed window to better capture context for detecting trigger word

12.6.7 Machine Reading Comprehension

RNN with Attention

Convolution with Self-attention - QAnet

12.6.8 Image Caption

Problem Formulation

- Input Data
  - Image
    - visual input as the target of description
- Goal
  - Natural Expression
    - description of the image in natural language, e.g. English

Baseline Approach & Previous Work

- Neural Image Caption
  - Visual Information
    - encoded by CNN backbone into a 1-D vector
  - Word Information
    - a set of word selected beforehand
    - word embedding performed
  - Language Generation
    - generated by an LSTM decoder
    - combining info: visual encoding as initial state of LSTM
    - process: LSTM gives each word a to-be-selected probability at each time step
  - Inference
    - sampling: sample each word according to the distribution given by LSTM
    - beam search: iteratively consider extending k best sentence of length t to $t+1$ $\Rightarrow$ select k best sentence of length $t+1$ from all resulted sentences

    (beam search selected in the paper)

12.6.9 Referring Segmentation

Problem Formulation

- Input
  - Image
    - visual input for segmentation
  - Natural Language Expression
    - expression to denote the interested object(s)/stuff(s)
- Goal
  - Segmentation Mask of Referred Object(s)
    - currently (till early 2019), mostly binary segmentation
- Related Area
  - NLP + CV
    - referring localization
    - image caption

Baseline Approach & Previous Work

- Segmentation from Natural Language Expressions
  - Spatial Info
    - FCN-32s to encode the image into 2-D feature maps (the last conv layer)
  - Language Info
    - LSTM to encode the sentence into 1-D vector (the last hidden state)
  - Combining Info and Output
    - per-pixel info: concat [coordinates of current pixel (coord info), language info]

        - tile the per-pixel info into a feature map, then concat to the spatial info
          (per-pixel info concatenated at every pixel of spatial info)
        - followed by a series of conv and finally a deconv for upsampling
    - Training
        - per-pixel cross-entropy loss
    - Pros
        - special spatial info: coord of each pixel
        - standard info combination: concatenation
    - Cons
        - no powerful spatial info encoder: FCN-32s instead of Resnet/Unet...
        - weak upsampler, compared to encoder-decoder architecture
        - language info comes late: after downsampling
        - weak language info: only integrated once

- Recurrent Multimodal Interaction for Referring Image Segmentation
    - Spatial Info
        - DeepLab-101 as encoder (Resnet as backbone, with atrous conv)
        - then tiled (concat at every pixel) by coord info (coordinate of current pixel)
    - Language Info
        - word embedding w_t for $t = 1, ..., T$
        - LSTM scanning the sentence, with hidden state h_t at time t
        - language info l_t = concat $[h_t, w_t]$
    - Combining Info
        - l_t tiled to spatial info, at each time step
          $\Rightarrow$ creating combined feature maps F_t (of shape [height, wide, channel])
        - combined feature maps $F_1, ..., F_T$ fed to an convolutional LSTM,
          where the ConvLSTM shares weight over both space and time
          $\Rightarrow$ feature vector of $F_t[i, j]$ is the input of the ConvLSTM at time t
          $\Rightarrow$ conv in ConvLSTM implemented as 1×1 conv
        - a series of conv following the last hidden state of the ConvLSTM
    - Output
        - bilinear interpolated to original input size
        - optionally post-processed by dense CRF, using pydensecrf (hence inference only)
    - Pros
        - more powerful spatial info extractor: DeepLab-101
        - better language info: integrated at every time step, maintained by an ConvLSTM
    - Cons
        - weak architecture for spatial info: still no upsampling (blur segmentation)
        - no spatial relation considered in ConvLSTM (?)
        - weak language representation
          (better with pos tag, word2vec, word dict, biLSTM, and maybe even attention)
        - language info still comes late: still after downsampling

Current State-of-The-Art (early 2019)

- Key-Word-Aware Network for Referring Expression Image Segmentation
  - Spatial Info
    - DeepLab-101 as encoder for comparability
    - then tiled by coord info (coordinate of each pixel)
  - Language Info
    - LSTM scanning sentence, each hidden state as word info
  - Combining Info
    - attention mask from combined info (spatial info with language info tiled) (at each time step)
    - attention weighting over spatial info at each time step
      $\Rightarrow$ an 1-D global encoding for each time step (via weighted mean over space)
      $\Rightarrow$ filling feature maps: global encoding if attention here $>$ threshold; else $\mathbf{0}$
      $\Rightarrow$ summing filled feature maps over time for the global spatial maps c
    - attention weighting over tiled language info at each time step, correspondingly
      $\Rightarrow$ tiled language info maps summed over time for the global language maps q
    - concat [spatial info, c, q], followed by 1×1 conv
  - Output
    - upsampling performed
  - Pros
    - attention introduced: from combined info
    - better combination: attention masked interact with both spatial & language info
  - Cons
    - blur segmentation: no encoder-decoder architecture
    - attention mask obtained sequentially: only last mask has complete language info
    - language info comes late: after downsampling
- Referring Image Segmentation via Recurrent Refinement Networks
  - Spatial Info
    - DeepLabl ResNet-101 as encoder
    - last feature maps tiled (concat at each pixel) with coord info
  - Language Info
    - LSTM scanning sentence, generating word info at each time step
    - last hidden layer as language info
  - Combining Info
    - combined info = spatial info tiled with language info
    - selecting set of feature maps from downsampling stages
    - all selected feature maps resized and fed to 1×1 conv
      $\Rightarrow$ to match the dimensions of combined info
    - convolutional LSTM applied to refine the combined info
      (with matched selected feature maps as input at each time step)
  - Output
    - a conv after final hidden state of ConvLSTM for segmentation
    - upsampled to original image size

- Pros
  - ConvLSTM integrating info at dowsampling stage ⇒ segmentation refined
- Cons
  - no upsampling: blur segmentation, mitigated by ConvLSTM though (yet no language info introduced in refinement)
  - CNN fixed during training: relying on ConvLSTM
  - single info combination: only by tiling (though, currently performing the best in all dataset)

Research Direction

- Integrating Encoder-Decoder Architecture
  - Upsampling
    - similar to Unet, concat low-level spatial info
    - introduce language info as well (e.g. early combination, explicit introducing, ...)
- Early Info Combination
  - Tiling at First Conv
    - downsampling more responsible for language info processing ⇒ hopefully get more fine-tuning alone with conv filters
    - can be used with pre-trained net: $ReLU(conv_1 * X_1 + conv_2 * X_2) = ReLU([conv_1, conv_2] * [X_1, X_2])$
  - Multiple Entries
    - combining info at different stages of downsampling / upsampling
- Attention
  - Attention from Combined Info
    - as key-word-aware net
  - Attention on Language Info
    - 1-D spatial pyramid pooling / attention mask on the sentence encoding
- Language Info Throughout Network
  - Encoder-Decoder for Language Info
    - network asked to recover language info after processing combined info (potentially via a separate branch only at training time) ⇒ auxiliary loss
  - Language as Conv Filter
    - Language Info, through a subnet, becoming a set of conv filters ⇒ then imposed in downsampling, tunnel, upsampling or bridge stage(s)
- Data Augmentation
  - Translation Module
    - using the same image
    - expression translated to a middle language and then back to English ⇒ language info trained more finely

12.7 Special Learning

12.7.1 Transfer Learning

Problem Formulation

- Input Data
  - Source Data
    - a large amount of labeled data
    - having different distribution then the desired target data
  - Target Data
    - a small amount of labeled data, with a large amount of unlabeled data (due to hardness of labeling, etc.)
    - from the distribution where model need to handle
- Goal
  - Model Performance
    - good performance on val&test set (containing target data)
    - good generalization ability on the target distribution

Standard Baseline

- Pre-training & Fine-tunning
  - Assumption
    - distribution of source & target data share some common features ⇒ different task shares some common knowledge
  - Practice
    - train network on source data only
    - swap/modify the last few layers (including prediction layer)
    - retrain the last layer (limited target data) / all net (enough target data)
  - Guideline
    - small dataset: freeze pretrained network & use it as fixed feature extractor ⇒ only train the last prediction layer
    - medium dataset: freeze fewer layers, design some own last layers
    - exceptionally large dataset with large computation budget: train from scratch ⇒ pretrained weights as initialization (nor preferred in most cases)
  - Understanding
    - sharing weights/structure: low level feature extraction useful for both ⇒ based on model ability
- Transfer Ada Boost (trAdaBoost)
  - Assumption
    - distribution of target & source overlap more or less ⇒ able to extract helpful guides from source data
  - Practice
    - setup train set with mixed target & source data

        - weighting example from target & source differently:
          for source data weight = $\frac{1}{N_{\text{source}}}$; target data weight = $\frac{1}{N_{\text{target}}}$
          ⇒ target data more important (as smaller in number)
        - for each weight-update iteration (may contain multiple epochs), update the weight:
          ⇒ shift the weight (importance) towards target data & normalize all the weight

      ⇒ based on data distribution

    - Understanding
        - learn the shared feature/knowledge with the help of source data
        - focus more on target as making progress

- Feature Projection
    - Assumption
        - few or NO overlap between source & target (as data examples directly)
        - source & target can be mapped onto a shared feature space, where overlap can be discovered
    - Practice
        - project/map the source & target data onto the same feature space
        - transfer learning in the shared space

      ⇒ based on distribution transformation
    - Understanding
        - try discover common feature through transformation
          (may need a decoder to map back to desired output space)
    - Example
        - word embedding: learn word relation as unsupervised learning
          ⇒ a shared feature space to represent word

12.7.2 Multi-task Learning

Problem Formulation

- Input Data
    - Multi-labeled Data
        - one input data corresponds to multiple desired outputs
        - ⇒ require similarity/common knowledge in different tasks
          (e.g. object detection for multiple object types)
    - Partial-labeled Data
        - desired outputs may not be all labeled in the input
          (i.e. some may be missed)
- Goal
    - General Solution to Multi-task
        - give all desired outputs from a single network

Standard Baseline

- Single Networks with Multiple Predictions
  - Sharing
    - shared low-level layers to extract features from the input
    - shared loss as a sum over all prediction for corresponding label
    - shared training as back-prop computed as a single network
    - shared input data as trained together
      $\Rightarrow$ shared knowledge discovered when training on data for other tasks
  - Understanding
    - each task help each other, by contributing to the common knowledge
    - overcome data shortage: augmented by data for other tasks
    - partial labeled still useful: help train the shared layers

12.7.3 K-shot Learning

Problem Formulation

- Training Set
  -