Machine Learning

Liyao Tang

January 4, 2020

Contents

List of Figures

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

$$
\begin{aligned}
H(z) = P(Z < z) &= P(X + Y < z) \\
&= \int_x P(X = x)P(X + Y < z | X = x)dx \\
&= \int_x f(x)P(Y < z - x)dx \\
&= \int_x f(x)G_Y(z - x)dx
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow h(x) = \frac{d}{dz}H(z) &= \frac{d}{dz}\int_x f(x)G_Y(z-x)dx \\
&= \int_x f(x)\frac{dG_Y(z-x)}{dz}dx \\
&= \int_x f(x)g(z-x)dx \\
&= f * g(z)
\end{aligned}
$$

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)
- Abstraction
  - a vector space with operations that can be applied on different math concept (e.g. function derivatives is actually in vector space)

- 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
      - additivity: $L(\mathbf{x}_1 + \mathbf{x}_2) = L(\mathbf{x}_1) + L(\mathbf{x}_2)$
      - 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

    $$\begin{aligned}\Rightarrow L(\mathbf{x}) &= L(x_1e_1 + ... + x_ne_n)\\ &= L(x_1e_1) + ... + L(x_ne_n)\\ &= x_1L(e_1) + ... + x_nL(e_n)\end{aligned}$$

    $\Rightarrow$ 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 $\mathbf{v}, \mathbf{w}$

      $$\begin{aligned}0 &= F(..., (\mathbf{v} + \mathbf{w}), ..., (\mathbf{v} + \mathbf{w}), ...)\\ &= F(..., \mathbf{v}, ..., \mathbf{v}, ...) + F(..., \mathbf{w}, ..., \mathbf{w}, ...) + F(..., \mathbf{v}, ..., \mathbf{w}, ...) + F(..., \mathbf{w}, ..., \mathbf{v}, ...)\\ &= F(..., \mathbf{v}, ..., \mathbf{w}, ...) + F(..., \mathbf{w}, ..., \mathbf{v}, ...)\end{aligned}$$

      $\Rightarrow F(..., \mathbf{v}, ..., \mathbf{w}, ...) = -F(..., \mathbf{w}, ..., \mathbf{v}, ...)$
    - 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
      - $m < n$: projecting to subspace
      - $m > n$: 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 $x'_i = \sum_{j=1}^{n} e'_{ji} x_j$
      (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 cols 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
      - a further L_2 transformation on a transformed space $L_1(E)$
        (all result represented under the original basis E)
      - 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 $E = [e_1, e_2]$)

Inverse of Matrix

- Inverse of Linear Transformation in Square Matrix
  - 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 for $M_{n\times n}$
    - transformed basis in M still span the space!
      (otherwise, transformed vectors projected onto subspace & can NOT be reversed)
    - $\Rightarrow$ column vectors (transformed basis) in $M_{n\times n}$ linearly INdependent
- Non-Square Matrix
  - Definition
    - left inversion: $M^{-1}_{n\times m} \cdot M_{m\times n} = I_{n\times n}$
    - right inversion: $M_{m\times n} \cdot M^{-1}_{n\times m} = I_{m\times m}$

Rank and Dimension

- Dimension
  - Definition
    - the number of linearly INdependent vectors in a vector space
      $\Rightarrow$ the least number of vector required to span the space
  - Dimension of Column Space
    - $\Rightarrow$ the linearly independent col vectors
      i.e. the rank of M
- Measuring Linear Transformation on Dimension
  - Measuring the Transformed Space
    - rank of $M : R(M) = r$: 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 $M_{m\times n}$
  - 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

        - $D(\text{null space}) = n - R(M)$

        - understanding: $R(M)$ vectors chosen for basis of col space of M
          $\Rightarrow \mathbf{x}$ able to freely combine the remaining col vectors, with $M\mathbf{x} = 0$ still satisfied

- 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 ($m > n$)

- Linear Dependency
    - $R(M_{m\times n}) = \min\{m, n\}$ (Full Rank)
        - $m = n$: M col vectors linear INdependent as n vector spans an N-D space
        - $m < n$: M col vectors linear dependent as n vector spans an M-D subspace
        - $m > n$: M col vectors linear INdependent as n vector spans at least an N-D space
    - $R(M_{m\times n}) < \min\{m, n\}$
        - $m \leq n$: M col vector linear dependent
        - $m > n$: M col vector MAY be linear independent/dependent

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
    - Understanding
        - each basis $e_i \in E$ projected to 1-D space (a scalar) by transformation $\mathbf{x}/\mathbf{y}$
        - $\Rightarrow \mathbf{x} \cdot \mathbf{y}$: 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 $\in \mathbb{R}^n$ 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

- Overview
  - Definition
    - kth exterior product $\wedge^k V$ is a vector space, with a map $\psi : \underbrace{V \times ... \times V}_{k \text{ times}} \to \wedge^k V$,
      where $V \in \mathbb{R}^n$ a vector space, map ψ the exterior multiplication
      (note: $\psi(\mathbf{v}_1, ..., \mathbf{v}_k) = \mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k$, also called k-blade or k-form)
  - Properties of Space $\wedge^k V$
    - ψ is alternating multilinear map
    - for basis $\{e_1, ..., e_n\}$ of $V \in \mathbb{R}^n$
      $\Rightarrow \{e_{i_1} \wedge ... \wedge e_{i_k} | 1 \leq i_1 < ... < i_k \leq n\}$ a basis of $\wedge^k V$
      $\Rightarrow e_{i_1} \wedge ... \wedge e_{i_k}$ can form any permutation of the same input by swapping order (due to its alternating multilinearity)
      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
    - sum of k-wedge is still in the vector space $\wedge^k V$ (as can be expressed by its basis)
    - $F : \underbrace{V \times ... \times V}_{k} \to X$ uniquely factors into $\underbrace{V \times ... \times V}_{k} \xrightarrow{\psi} \wedge^k V \xrightarrow{\underline{F}} X$,
      where F any alternating multilinear map, ψ exterior multiplication, $\underline{F}$ linear map
      (since ψ also an alternating multilinear map & $\underline{F}$ to preserve linearity)
  - Inferred Propositions
    - with $V \in \mathbb{R}^n$, dim $\wedge^k V = \binom{n}{k} = C_n^k$, due to the form of its basis
      note: for $k > n$, $\wedge^k V = \{0\}$
      (top exterior power)
    - k-blade $\mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k = 0 \Leftrightarrow \mathbf{v}_1, ..., \mathbf{v}_k$ linearly dependent
      (k-blade $\mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k \neq 0 \Leftrightarrow \mathbf{v}_1, ..., \mathbf{v}_k$ linearly INdependent)
      - if $\mathbf{v}_1, ..., \mathbf{v}_k$ linearly dependent $\Rightarrow \mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k = 0$ as ψ alternating
      - if $\mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k = 0$, assume $\mathbf{v}_1, ..., \mathbf{v}_k$ INdependent
        $\Rightarrow \mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k$ a basis of $\wedge^k V \Rightarrow$ basis $= 0$
        $\Rightarrow$ assumption failed, $\mathbf{v}_1, ..., \mathbf{v}_k$ linearly dependent

      (wedge dependence lemma)
    - $\mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k = c(\mathbf{w}_1 \wedge ... \wedge \mathbf{w}_k) \Leftrightarrow \text{span}(\mathbf{v}_1, ..., \mathbf{v}_k) = \text{span}(\mathbf{w}_1, ..., \mathbf{w}_k)$,
      where $\mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k, \mathbf{w}_1 \wedge ... \wedge \mathbf{w}_k \in \wedge^k V$ & non-zero, scalar $c \neq 0$
      - if $\text{span}(\mathbf{v}_1, ..., \mathbf{v}_k) = \text{span}(\mathbf{w}_1, ..., \mathbf{w}_k)$
        $\Rightarrow$ each $\mathbf{v}_1, ..., \mathbf{v}_k$ expressed by a linear combination of $\mathbf{w}_1, ..., \mathbf{w}_k$
        $\Rightarrow \mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k = c(\mathbf{w}_1 \wedge ... \wedge \mathbf{w}_k)$ as ψ alternating multilinear
        ($c \neq 0$ as both pure wedges non-zero)
      - if $\mathbf{v}_1 \wedge ... \wedge \mathbf{v}_k = c(\mathbf{w}_1 \wedge ... \wedge \mathbf{w}_k)$
        let $U_v = \text{span}(\mathbf{v}_1, ..., \mathbf{v}_k), U_w = \text{span}(\mathbf{w}_1, ..., \mathbf{w}_k), U_{vw} = U_v \cap U_w$
        $\Rightarrow$ assume $U_v \neq U_w$, U_{vw} has dimension l
        $\Rightarrow \exists l$ vectors in $\{\mathbf{v}_1, ..., \mathbf{v}_k\}, \{\mathbf{w}_1, ..., \mathbf{w}_k\}$ as 2 sets of basis of U_{vw}
        $\Rightarrow$ change l vectors to be a common basis for U_{vw}: $\{\mathbf{u}_1, ..., \mathbf{u}_l\}$
        $\Rightarrow$ now $\mathbf{v}_1, ..., \mathbf{v}_{k-l}, \mathbf{u}_1, ..., \mathbf{u}_l, \mathbf{w}_1, ..., \mathbf{w}_{k-l}$ linearly independent
        $\Rightarrow \psi(v_1, ..., \mathbf{v}_{k-l}, \mathbf{u}_1, ..., \mathbf{u}_l)\psi(\mathbf{w}_1, ..., \mathbf{w}_{k-l}, \mathbf{u}_1, ..., \mathbf{u}_l)$ 2 different basis for $\wedge^k V$
        while $\psi(\mathbf{v}_1, ..., \mathbf{v}_{k-l}, \mathbf{u}_1, ..., \mathbf{u}_l) = c_v \psi(\mathbf{v}_1, ..., \mathbf{v}_k)$
        similarly $\psi(\mathbf{w}_1, ..., \mathbf{w}_{k-l}, \mathbf{u}_1, ..., \mathbf{u}_l) = c_w \psi(\mathbf{w}_1, ..., \mathbf{w}_k)$
        $\Rightarrow \psi(\mathbf{v}_1, ..., \mathbf{v}_k) \neq c\psi(\mathbf{w}_1, ..., \mathbf{w}_k), (c \neq 0)$ (as basis linearly independent)
        $\Rightarrow$ conflict, assumption failed, hence $\text{span}(\mathbf{v}_1, ..., \mathbf{v}_k) = \text{span}(\mathbf{w}_1, ..., \mathbf{w}_k)$

- Induced Maps

-

- Understanding
  - Construction from Geometry
    - e.g. $V \in \mathbb{R}^3, \wedge^3 V$ has basis for:
      each x, y, z axis, each xy, xz, yz plane and finally xyz volume

Determinant

- Construction from Exterior Product
  - Definition
  - Independence of Basis
  - Multiplicativity
  - Relationship to Invertibility
- 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

- Calculation via Determinant
    - Practice
        - for $\mathbf{v}_1, ..., \mathbf{v}_{n-1} \in \mathbb{R}^n, \mathbf{v}_1 \times ... \times \mathbf{v}_{n-1} = \det \begin{pmatrix} — & \mathbf{v}_1 & — \\ & \vdots & \\ — & \mathbf{v}_{n-1} & — \\ e_1 & ... & e_n \end{pmatrix}$
        - construct matrix with each vectors as row & basis $e_1, ..., e_n$ as the last row
          $\Rightarrow$ an $N \times N$ matrix (fake $e_1, ..., e_n$ as number)
          $\Rightarrow$ re-combined into a vector $\mathbf{w} = \mathbf{v}_1 \times ... \times \mathbf{v}_{n-1} \in \mathbf{R}^n$
        - actually, carry out the hodge star operator implicitly (via determinant)
    - Direction
        - perpendicular to the $\text{span}(\mathbf{v}_1, ..., \mathbf{v}_{n-1})$

- Understanding of Calculation via Determinant
    - Equivalent Linear Transformation
        - view the constructed $\det(\cdot)$ as transformation $F : (\mathbf{v}_1, ..., \mathbf{v}_{n-1}, \mathbf{x}) \to y \in R$
          $\Rightarrow$ a linear transformation to 1-D number line
          $\Rightarrow \exists \mathbf{p} \in \mathbb{R}^n, y = F(\mathbf{x}) = \mathbf{p} \cdot \mathbf{x}$ (due to duality of dot product)
        - note that: transformation $\mathbf{p}$ (or F) is generated by vector $\mathbf{v}_1, ..., \mathbf{v}_{n-1}$
    - Algebraic View
        - view variable $\mathbf{x} = x_1, ..., x_n$ as the coefficient for each basis $e_1, ..., e_n$
        - $\Rightarrow \mathbf{p}$ describes the linear transformation of F on the vector space (on arbitrary vector $\mathbf{x}$)
        - $\Rightarrow$ dual vector $\mathbf{p} = \mathbf{v}_1 \times ... \times \mathbf{v}_{n-1}$
    - Geometric View
        - as $\det(\mathbf{v}_1, ..., \mathbf{v}_{n-1}, \mathbf{x})$ measures their high-dimension volume y
          $\Rightarrow y = \mathbf{p} \cdot \mathbf{x} = \|\mathbf{p}\| \cdot \mathbf{x}_{\|\mathbf{p}}$,
          where $\mathbf{x}_{\|\mathbf{p}}$ the component of $\mathbf{x}$ parallel to $\mathbf{p}$
        - while $y = \text{volume}(\mathbf{v}_1, ..., \mathbf{v}_{n-1}) \cdot \mathbf{x}_{\perp \text{span}(\mathbf{v}_1, ..., \mathbf{v}_{n-1})}$ (integration perspective),
          where $\mathbf{x}_{\perp \text{span}(\mathbf{v}_1, ..., \mathbf{v}_{n-1})}$ the component of $\mathbf{x}$ perpendicular to $\text{span}(\mathbf{v}_1, ..., \mathbf{v}_{n-1})$
        - $\Rightarrow$ find a $\mathbf{p}$, s.t. $\forall \mathbf{x}, \det(\mathbf{v}_1, ..., \mathbf{v}_{n-1}) \cdot \mathbf{x}_{\perp \text{span}(\mathbf{v}_1, ..., \mathbf{v}_{n-1})} = \|\mathbf{p}\| \cdot \mathbf{x}_{\|\mathbf{p}}$
        - $\Rightarrow$ only solution: $\mathbf{p} \perp \text{span}(\mathbf{v}_1, ..., \mathbf{v}_{n-1})$ and $\|\mathbf{p}\| = \det(\mathbf{v}_1, ..., \mathbf{v}_{n-1})$
          (naturally accounting for signed volume via direction of $\mathbf{p}$)

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 $\mathbf{v}$

- Existence of Solution(s)
  - Transformed Basis Linearly INdependent
    - single solution exists, as $A\mathbf{x} = \mathbf{v} \Leftrightarrow \mathbf{x} = A^{-1}\mathbf{v}$
      (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 ($\det(A) = 0$)
    - multiple solutions: transformed 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)
    - no solution: basis in A linearly INdependent with $\mathbf{v}$
      $\Rightarrow$ i.e. can NOT possibly be described by transformed basis in A

- Cramer's Rule
  - Basis Represented by Determinant
    - $\forall i = 1, ..., n, \det(e_1, ..., e_{i-1}, \mathbf{x}, e_{i+1}, ...e_n) = \mathbf{x}_{\|e_i}$,
      where $\mathbf{x}_{\|e_i}$ the component of $\mathbf{x}$ on basis e_i (as the i^{th} coord of $\mathbf{x}$)
    - similarly, $\det(a_1, ..., a_{i-1}, \mathbf{v}, a_{i+1}, ..., a_n) = \mathbf{v}_{\|a_n}$,
      where a_i the i^{th} col vector of A
      $\Rightarrow$ represent the i^{th} coord of transformed vector $\mathbf{v}$ under transformed basis A
  - Transformation Described by Determinant
    - $\det(A)$ measures the change of unit volume from I to A
    - $\det(e_1, ..., e_{i-1}, \mathbf{x}, e_{i+1}, ...e_n)$ the volume before transformation
    - $\det(a_1, ..., a_{i-1}, \mathbf{v}, a_{i+1}, ..., a_n)$ the volume after transformation
    - $\Rightarrow \det(a_1, ..., a_{i-1}, \mathbf{v}, a_{i+1}, ..., a_n) = \det(e_1, ..., e_{i-1}, \mathbf{x}, e_{i+1}, ...e_n) \cdot \det(A)$
  - Solving $\mathbf{x}$
    - hence, $\mathbf{x}_{\|e_i} = \det(e_1, ..., e_{i-1}, \mathbf{x}, e_{i+1}, ...e_n) = \dfrac{\det(a_1, ..., a_{i-1}, \mathbf{v}, a_{i+1}, ..., a_n)}{\det(A)}$

Eigenvector and Eigenvalue

- Invariability in Linear Transformation
  - Invariable Line
    - vector before and after transformation A are on the same **line**
      $\Rightarrow A\mathbf{x} = \lambda\mathbf{x}$ (under the same basis)
      $\Rightarrow$ transformation A NOT rotate those $\mathbf{x}$, but only stretch them, by a factor λ
    - $\mathbf{x}$ the eigenvector for those invariable line;
      λ the eigenvalue for corresponding stretching factor
  - Representing Transformation
    - for rotation, the eigenvector&value can be more expressive than a matrix

- Solving Eigenvector&value
  - solving $A\mathbf{x} = \lambda\mathbf{x}$ for $\mathbf{x}$ and λ
    $\Rightarrow$ equivalent with solving $(A - \lambda I)\mathbf{x} = \mathbf{0}$
  - a necessary condition for non-zero solution $\mathbf{x}$: $\det(A - \lambda I) = 0$
    (understanding 0: as solution existence condition for linear system of equations)
    (understanding 1: non-zero $\mathbf{x}$ transformed by $A - \lambda I$ into $\mathbf{0}$)
    (understanding 2: non-zero $\mathbf{x}$ perpendicular to all row vectors in $A - \lambda I$)
    (understanding 3: otherwise, $\mathbf{x} = (A - \lambda I)^{-1}\mathbf{0} = \mathbf{0}$)
  - $\Rightarrow$ with no/single/multiple $\lambda \in \mathbb{R}$ solved, bring back to relation $(A - \lambda I)\mathbf{x} = 0$
    $\Rightarrow$ to solve for a line/span, consisting of eigenvector(s)
    (note: no $\lambda \in \mathbb{R}$ solved, no eigenvector then)

Translation of Basis

- Translation between Coordinate
  - Notation
    - $\mathbf{x} = (x_1, ..., x_n)$: a coordinate under basis $A_{n\times n}$
    - $\mathbf{b} = (b_1, ..., b_n)$: a coordinate under basis $E = I$
    - $\overrightarrow{v}$: the same vector described by $\mathbf{x}$ & $\mathbf{b}$
    - $A_{n\times n}$: the transformed basis described by basis before transformation (i.e. E)
      (with col vectors of A linearly INdependent)
    - $A^{-1}_{n\times n}$: the transformed basis s.t. $A^{-1}AE = E$
      $\Rightarrow$ describe transformation $A \to E$ by basis before transformation (i.e. $AE = A$)
    - $M_{n\times n}$: a matrix for arbitrary transformation $L : E \to ?$, described by basis E
  - Translating Coordinate
    - $A\mathbf{x}$: describe $\overrightarrow{v}$ by basis E
      $\Rightarrow$ translate the vector coord from basis A to basis E
    - $A^{-1}\mathbf{b}$: describe $\overrightarrow{v}$ by basis A
      $\Rightarrow$ translate the vector coord from basis E to basis A
    - $\Rightarrow A\mathbf{x} = \mathbf{b}$
  - Translating Transformation
    - translate coord $\mathbf{x}$ (under A) to be under $E = I$: $A\mathbf{x}$
      (translate to the basis where M known)
    - transform the vector by M: $MA\mathbf{x}$ (under basis E)
    - translate the vector back to be under basis A: $A^{-1}MA\mathbf{x}$
    - $\Rightarrow \mathbf{x}' = A^{-1}MA\mathbf{x}$, where $A^{-1}MA$ the transformation L described under basis A

Diagonal Matrix

- Overview
  - Definition
    - must be a square matrix
    - non-zero values allowed only on the diagonal of a matrix
- Transformation in Diagonal Matrix
  - Interpretation
    - as the transformation only stretch original basis I by diagonal value
      $\Rightarrow$ all transformed basis is eigenvectors, with eigenvalue the diagonal value!

- Eigenbasis
  - the basis formed by eigenvectors of a matrix $A_{n\times n}$

- Diagonalizing Matrix
  - Practice
    - for transformation $A_{n\times n}$ under basis I, solve all eigenvalues $\lambda = [\lambda_1, ..., \lambda_m]$
    - if $m = n$, check all eigenvectors $Q = [\mathbf{v}_1, ..., \mathbf{v}_m]$ to be linearly INdependent (i.e. check if eigenvectors able to form a valid basis for n-D space)
      $\Leftrightarrow$ check there are n distinct eigenvalues u
    - if so, A said to be **diagonalizable**
    - $\Rightarrow$ reconstruct transformation A under eigenbasis (view V as transformed basis)
      $\Rightarrow Q^{-1}AQ = \Lambda$, or $A = Q\Lambda Q^{-1}$, where $\Lambda = \text{diag}[\lambda_1, ..., \lambda_n]$
  - Understanding
    - numerical proof: $Q^{-1}AQ = Q^{-1}[\lambda_1\mathbf{v}_1, ..., \lambda_n\mathbf{v}_n] = \Lambda$
    - translating transformation A into a transformation A' under eigenbasis
      $\Rightarrow$ for $\mathbf{x}$, transformed into eigenbasis $Q^{-1}\mathbf{x}$
      $\Rightarrow$ perform k transformation in eigenbasis $\Lambda^k Q^{-1}\mathbf{x}$
      $\Rightarrow$ then transformed back $Q\Lambda^k Q^{-1}\mathbf{x}$
    - $\Rightarrow$ easy to perform any diagonalizable transformation

1.2.2 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: $\mathbf{x}^2 = r^2\cos^2\phi_1 + \sum_{n=2}^{N} x_n^2 = r^2$
      $\Rightarrow \sum_{n=2}^{N} x_n^2 = r^2\sin^2\phi_1$

- $\mathbf{x}_{2-N} = \sum_{n=2}^{N} x_n e_n$

$$\Rightarrow \begin{cases} \|\mathbf{x}_{2-N}\|^2 = \sum_{n=2}^{N} x_n^2 = r^2 \sin^2\phi_1 = r_2^2 \\ \cos\phi_2 = \dfrac{\mathbf{x}_{2-N} \cdot e_2}{\|\mathbf{x}_{2-N}\|\|e_2\|} = \dfrac{x_2}{r_2} \end{cases}$$

$$\Rightarrow \begin{cases} r_2 = r\sin\phi_1 & (\text{ as } \phi_1 \in [0,\pi]) \\ x_2 = r_2\cos\phi_2 = r\sin\phi_1\cos\phi_2 \end{cases}$$

$$\Rightarrow \mathbf{x}_{2-N} = r\sin\phi_1\cos\phi_2 e_2 + \sum_{n=3}^{N} x_n e_n$$

- Space $e_3, ..., e_N$:
  - from above: $\mathbf{x}_{2-N}^2 = r^2\sin^2\phi_1\cos^2\phi_2 + \sum_{n=3}^{N} x_n^2 = r^2\sin^2\phi_1$

$$\Rightarrow \sum_{n=3}^{N} x_n^2 = r^2\sin^2\phi_1\sin^2\phi_2$$

  - $\mathbf{x}_{3-N} = \sum_{n=3}^{N}$

$$\Rightarrow \begin{cases} \|\mathbf{x}_{3-N}\|^2 = \sum_{n=3}^{N} x_n^2 = r^2\sin^2\phi_1\sin^2\phi_2 = r_3^2 \\ \cos\phi_3 = \dfrac{\mathbf{x}_{3-N} \cdot e_3}{\|\mathbf{x}_{3-N}\|\|e_3\|} = \dfrac{x_3}{r_3} \end{cases}$$

$$\Rightarrow \begin{cases} r_3 = r\sin\phi_1\sin\phi_2 & (\text{ as } \phi_1, \phi_2 \in [0,\pi]) \\ x_3 = r_3\cos\phi_3 \end{cases}$$

$$\Rightarrow \mathbf{x}_{3-N} = r\sin\phi_1\sin\phi_2 e_3 + \sum_{n=4}^{N} x_n e_n$$

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

Homogeneous Coordinate

- Derivation
  - Perspective Geometry
    - solving the limitation of euclidean geometry
      (actually euclidean geometry is a subset)
    - $\Rightarrow$ describe parallel lines intersect at an infinite point in camera / human eyes
    - create a projective space & establish transformation between euclidean space
- Definition

- Point
  - $p' = (x', y')$ in Cartesian coord $\Rightarrow p = (x, y, w)$ in homogeneous coord, where $x' = \frac{x}{w}, y' = \frac{y}{w}$
  - $(x, y, 0)$ for points at infinity, instead of tedious (∞, ∞) ($(0, 0, 1)$ for origin)
  - homogeneous due to: for any normal point, w takes any value except for 0
- Vector
  - $v' = (a, b)$ in Cartesian coord $\Rightarrow v = (a, b, 0)$ in homogeneous coord

- Describing Space
  - Line
    - line $l' = ax' + by' + c = 0$ in Cartesian coord $\Rightarrow l = ax + by + cw = 0$ (by replacing x', y' with x, y)
    - $\Rightarrow$ parallel lines intersect at points at infinity
      $\begin{cases} ax + by + c_0 w = 0 \\ ax + by + c_1 w = 0 \end{cases}$ has solutions $(x, y, 0)$, where $x, y \in \mathbb{R}$
  - Affine Transformation
    - definition: $y' = A'x' + b'$ in Cartesian space
    - translating into homogeneous space
      $\Rightarrow x = [x'^T, 1]^T, A = \begin{bmatrix} A', \mathbf{0} \\ \mathbf{0}^T, 1 \end{bmatrix}$ where $\mathbf{0}$ a col vector $[0, ..., 0]^T$
      $\Rightarrow$ hence Ax(homo) $\Leftrightarrow A'x'$(Cart)
      $\Rightarrow$ construct $T = \begin{bmatrix} A', b \\ \mathbf{0}^T, 1 \end{bmatrix}$
      $\Rightarrow y = Tx$(homo) $\Leftrightarrow y' = A'x' + b'$(Cart)
    - $\Rightarrow$ single matrix for affine transformation

- Understanding
  - Distinguishing Point vs. Vector
    - normal Cartesian point $p' = (x, y) \Rightarrow (x, y, 1)$
    - normal Cartesian vector $v' = (x, y) \Rightarrow (x, y, 0)$, as an infinite point (as for vector, absolute position does NOT matter)
  - Unifed Expression for Affine & Linear Transformation
    - easier, simpler & faster computation

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 :

    $M_4(x(u,v+\delta v), y(u,v+\delta v))$, $M_3(x(u+\delta u, v+\delta v), y(u+\delta u, v+\delta v))$,
    $M_1(x(u,v), y(u,v))$, $M_2(x(u+\delta u, v), y(u+\delta u, v))$

    $$\begin{aligned}
    \Rightarrow x_2 - x_1 &= x(u+\delta u, v) - x(u,v) = \frac{\partial x}{\partial u}|_{(u,v)}\delta u \\
    x_4 - x_1 &= x(u, v+\delta v) - x(u,v) = \frac{\partial x}{\partial v}|_{(u,v)}\delta v \\
    y_2 - y_1 &= y(u+\delta u, v) - y(u,v) = \frac{\partial y}{\partial u}|_{(u,v)}\delta u \\
    y_4 - y_1 &= y(u, v+\delta v) - y(u,v) = \frac{\partial y}{\partial v}|_{(u,v)}\delta v
    \end{aligned}$$

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

    $$\begin{aligned}
    \Rightarrow S_{M_1M_2M_3M_4} &= |\overrightarrow{M_1M_2} \times \overrightarrow{M_1M_4}| = |\begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_4 - x_1 & y_4 - y_1 \end{vmatrix}| \\
    &= |\begin{vmatrix} \frac{\partial x}{\partial u}\delta u & \frac{\partial y}{\partial u}\delta u \\ \frac{\partial x}{\partial v}\delta v & \frac{\partial y}{\partial v}\delta v \end{vmatrix}| = |\begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} \end{vmatrix}|\delta u \delta v \\
    &= |J(u,v)|\delta u \delta v
    \end{aligned}$$

  - $\Rightarrow$ 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 $\to$ 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
      $\Rightarrow d\sigma = \frac{1}{2}(r+dr)^2 d\theta - \frac{1}{2}r^2 d\theta = rdrd\theta + \frac{1}{2}(dr)^2 d\theta$
      $\Rightarrow d\sigma = rdrd\theta$, when $dr, d\theta \to 0$

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

- $\mathbb{E}[x] = \int_{-\infty}^{+\infty} \mathcal{N}(x|\mu, \sigma^2) x dx = \mu$

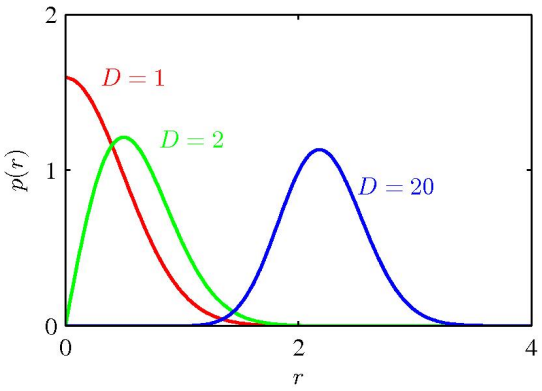
$$\Rightarrow \mathbb{E}[x^2] = \int_{-\infty}^{+\infty} \mathcal{N}(x|\mu, \sigma^2) x^2 dx$$
$$= \frac{1}{(2\pi\sigma^2)^{1/2}} \int_{-\infty}^{+\infty} x^2 \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} dx$$
$$= \pi^{-\frac{1}{2}} \int_{-\infty}^{+\infty} (\sqrt{2\sigma^2}x + \mu)^2 \exp(-x^2) dx,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

  - 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
    -
    - $\Rightarrow$ volume ratio of hyper sphere and hyper cube: $\Rightarrow$ the volume of a D-cube concentrates in its corner! $\Rightarrow$ 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
      $\Rightarrow$ 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 $G1 * G2$
    - $G_1 \sim \mathcal{N}_d(a, A), G_2 \sim \mathcal{N}_d(b, B)$

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

1.4.5 Bayesian Interpretation of Probability

Contrasting Frequentist Estimator

- Posterior $p(\mathbf{w}|\mathcal{D}) = \dfrac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$
  - 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 $\Rightarrow \prod \rightarrow \sum$
- 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
    - $p(X|\mu, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \sigma^2)$
    - $\Rightarrow \ln p(X|\mu, \sigma) = -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (x_n - \mu)^2 - \frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi)$
  - Solution
    - let $\frac{\partial}{\partial \mu} \ln p(X|\mu, \sigma^2) = 0 \Rightarrow \mu_{\text{ML}} = \frac{1}{N} \sum_{n=1}^{N} x_n$
    - let $\frac{\partial}{\partial \sigma^2} \ln p(X|\mu, \sigma^2) = 0 \Rightarrow \sigma^2_{\text{ML}} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{\text{ML}})$
  - Analysis
    - $\mathbb{E}[\mu_{\text{ML}}] = \mathbb{E}[\frac{1}{N} \sum_{n=1}^{N} x_n] = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[x_n] = \mu$
      (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
    - $$\begin{aligned}
      \mathbb{E}[\sigma^2_{\text{ML}}] &= \mathbb{E}[\frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{\text{ML}})^2] \\
      &= \mathbb{E}[\frac{1}{N} \sum_{n=1}^{N} (x_n^2 - 2\mu_{\text{ML}} x_n + \mu^2_{\text{ML}})] \\
      &= \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[x_n^2] - \mathbb{E}[\frac{1}{N} \sum_{n=1}^{N} 2x_n \mu_{\text{ML}}] + \mathbb{E}[\mu^2_{\text{ML}}] \\
      &= \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[x_n^2] - 2\mathbb{E}[\mu^2_{\text{ML}}] + \mathbb{E}[\mu^2_{\text{ML}}] \\
      &= \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[x_n^2] - \frac{1}{N^2} \sum_{i,j=1}^{N} \mathbb{E}[x_i x_j] \\
      &= \frac{1}{N} \sum_{n=1}^{N} (\sigma^2 + \mu^2) - \frac{1}{N^2} [N(N-1)\mu^2 + N(\sigma^2 + \mu^2)] \\
      &\qquad \text{(by 2nd moment of Gaussian } \mathbb{E}[x^2] \text{ and i.i.d assumption)} \\
      &= \left(\frac{N-1}{N}\right) \sigma^2
      \end{aligned}$$

$\Rightarrow$ biased variance **!**

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

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 (consider all possible $\mathbf{w}$)

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 Overview

Formulation

- Analysis Target
  - Human Behavior
    - goal-directed behaviors in the presence of options (alternative actions)
- Goal
  - Normative Decision Theory
    - how the decision should be made, in order to be national, etc.
      ⇒
    - an normative theory is *weakly falsified*, if ∃ a decision problem in which an agent can perform in contradiction with the theory without being irrational
    - an normative theory is *strictly falsified*, if ∃ a decision problem in which an agent performing in accordance with the theory cannot be a rational agent
  - Descriptive Decision Theory
    - how decisions are actually made in practice
  - Specific Concern
    - interaction of agents: collective decision-making
    - behavior of individuals in decisions
    - rationality in decisions
    - coordinating decision over time, in a changing environment
- Preference in Decision Making
  - Preference Logic
    - defined relation $\geq$ for two options a, b, according to agent's preference
      ⇒ form a partially ordered set $(S, \geq)$, where S the set of options
    - other relation $<, \equiv$ can be derived from $\geq$
  - Completeness
    - complete: $\forall i, j \in X$, a relation is defined
      (otherwise incomplete preference)
    - yet, usually incomplete preference, as determining preference takes effort
  - Transitivity
    - $a \geq b \wedge b \geq c \rightarrow a \geq c$ holds true for all a, b, c
      (hence all other relation $<, \equiv$)
    - ⇒ problem of accumulated negligible indifference ($\equiv$)
      e.g. a_0 as no sugar, a_{100} as full sugar, given $a_i \equiv a_{i+1}$ due to negligible difference
      ⇒ finally $a_0 \equiv a_{100}$ due to transitivity
  - Guideline
    - given the constructed partial order set $(S, \geq)$, choose one of its upper bound
      (with prevalent assumption of transitivity and completeness)
- Utility in Decision Making
  - Numerical Representation

        - using pre-defined relation in number, with a preference of maximal utility
          ⇒ naturally transitive & complete
        - relative number ⇒ meaningful in comparison within current option set
          (i.e. not comparable across different decision making process)
    - **Comparability**
        - only comparable between options in the same decision process
    - Guideline
        - maximization: choose one of the options with maximal utility
          (employed by default)
        - satisfying: choose one of the options with sufficient utility

- Decision Matrices
    - Outcome $O_{n\times m}$ = actions $\{a_1, ..., a_n\} \times$ states $\{s_1, ..., s_m\}$
        - with $\{a_1, a_2\}$ = {umbrella, no umbrella}; $\{s_1, s_2\}$ = {rain, no rain}

	rain	no rain
umbrella	dry&heavy	dry&heavy
no umbrella	wet&light	dry&light

    - Utility Assignment
        - requirement: shared perspective
          ⇒ all participants share a common concern
          (otherwise, different incomparable utility assigned to the same outcome)
        - assign each outcomes with utility ⇒ utility matrices U

	rain	no rain
umbrella	15	15
no umbrella	0	18

- Probability Estimation of Environment
    - Modeling State
        - certainty: deterministic knowledge of the environment after each action
          ⇒ thus deterministic outcomes for each action
        - risk: complete probabilistic knowledge of environment
          (may conditioned on action)
        - uncertainty: partial probabilistic knowledge
        - ignorance: no probabilistic knowledge, or not meaningful
    - Objective Probability
        - based on empirical known frequencies
        - indirect estimation (e.g. similar experiment), with calibration
          may use subjective probability, which is unreliable due to psycho-effect
          ⇒ better not use subjective estimates of prob
    - Bayesian Description
        - a coherent and complete set of probabilistic beliefs
          (in compliance with laws of probability)
        - probability subjected to the observation

- Action Discovering
    - Category
        - closure: settle down with current available actions

        - active postponement: searching for other possible actions (postpone the decision)
        - semi-closure: consider only reversible actions & searching for other actions
    - Guideline
        - all current actions have severe drawbacks?
        - is searching possible actions costly?
        - is problem aggregating over time?
          etc...
    - Understanding
        - a meta-decision problem before the scenario-specific decision

- $\Rightarrow$ Assumption
    - Closed Set of Actions
        - no new alternative action allowed to be added
          (v.s. open: new actions can be discovered and taken into consideration)
    - Mutually Exclusive Actions
        - no two actions can be both realized
    - Defined States
        - all possible states of environment are recognized
    - Outcome & Utility
        - the joint result of environment (state) and chosen action
        - utility assigned to each outcome (similar to rewards in RL)
          (though the utility for each result is usually subject to agent)
          $\Rightarrow$ completeness & transitivity assumed

Decision under Risk

- Maximizing Expected Utility
    - Procedure
        - for action a, expected utility $u = \mathbb{E}_{p(s)}U(a,s) = \sum_{s\in S} p(s)U(a,s)$,
          where $U(a,s)$ the utility assigned to outcome $O(a,s)$
    - Objective Utility
        - utility assigned according to the objective rewards (e.g. money)
        - $\Rightarrow$ St, Petersburg paradox: a fair coin tossed, if its head comes up at n^{th} toss, 2^n gold coins are rewarded, stop tossing if its back comes up
          $\Rightarrow$ expected wealth $= \lim_{n\to\infty} \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + ... + \frac{1}{2^n}\cdot 2^{n-1} = +\infty$
          $\Rightarrow$ should play the game for any finite entry fee
    - Subjective Utility
        - in economic: utility of next increment of wealth $\propto \frac{1}{w}$, where w the current wealth
          $\Rightarrow$ utility of wealth $= \log(w)$
    - Understanding
        - objective utility to maintain inter-subjective validity
          (hence expert advice remains effective)
        - maximizing utility to maximize long-term outcome
          (suitable for large-scale / long-term decision)
          $\Rightarrow$ law of large number to level out randomness

        - ⇒ suitability depends on the scale (the leveling-out effect)
          (e.g. isolated cases may not suitable to maximize expected utility)
        - may not suitable in extreme effect, to maintain fairness
          (e.g. avoid imposing high-prob risk on individuals)

- Variations
    - Conditionalized Expected Utility
        - prob of state $p(s)$ described conditional on action
          $\Rightarrow u = \mathbb{E}_{p(s|a)}U(a,s) = \sum_{s\in S} p(s|a)U(a,s)$
        - model the influence of actions on states
    - Generalized Expected Utility
        - process utility: accounts for attitude towards risk and certainty
        - maximize expected (utility of outcome + process utility)
    - Regret Theory
        - regret: the gap between reward received and highest possible reward from other actions
        - maximize expected utility of outcome & minimize regret
    - Prospect Theory
        - function $\mathrm{f}(0,1) \to (0,1)$ to adjust probability
          (description to be merely weights, not satisfying prob law) ⇒ overweight the prob close to 0&1, as tendentious to certainty
        - maximize weighted utility

- Causal Decision

Decision under Uncertainty

- Estimation of Environment
    - Binary Measure
        - segment out a range for probability describing the state
          e.g. $p(s) \in (0.05, 0.2)$
    - Second-Order Probability
        - a Bayesian probability for each state s
        - a further measurement of the reliability of probability estimates $p(s)$
          (may consider as a meta subjective prob)
    - Fuzzy Set Membership
        - vagueness, instead of randomness, to describe uncertainty
          ⇒ assign a degree of membership for each $p(s)$
          ⇒ based on laws of fuzzy membership (non-statistical concept)
        - measure the degree of "$p(s)$ is the prob of state s happening" being true
    - Epistemic Reliability
        - a weight $\in (0,1)$ as the measure of reliability of a prob,
          with NO mathematical properties pre-defined

- Decision Criteria
    - Maxmin Expected Utility

        - maximize the minimal expected utility
          ⇒ an extremely prudent/pessimistic criterion
    - Reliability-weighted Expected Utility
        - use the weighted average probability as true probability
          (then reduced to maximizing expected utility)
          ⇒ an optimistic criterion
    - Index
        - p' the best prob estimate and ρ as its degree of confidence
          ⇒ calculate u_{best} using p'; $u_{\min}$ the minimal expected utility
        - summarize as $u = \rho u_{\text{best}} + (1-\rho)u_{\min}$
          ⇒ maximize the u (choose the action with maximal u)
        - $\rho \in (0, 1)$ as a balance factor between pessimism & optimism
    - Clipped Maxmin Expected Utility
        - discard any prob with a reliability lower than a threshold
          (then max-min expected utility with remaining prob)
    - Filtered Maxmin Expected Utility
        - 3 filters applied in sequential order on actions (rather than prob)
        - E-test: action a passes, iff $\exists$ reliable prob $p(s) > 0$ & $\forall a' \in A, U(a, s) \geq U(a', s)$
          (action a is possible to be the best)
        - P-test: passes, iff it can be the best in all candidates
        - S-test: passes, iff it is the best under maxmin expected utility in all candidates
          ⇒ directly produce action to choose

Decision under Ignorance

- Category
    - Known States
        - the prob for each state $p(s) > 0 \Rightarrow$ unknown non-zero prob
    - Unknown States
        - the prob $p(s) \geq 0 \Rightarrow$ unknown prob
- Unknown Non-zero Prob
    - Max-Min Utility
        - maximize the minimal utility (extremely pessimistic)
        - lexicographic maximin: consider their 2^{nd} worst outcome
    - Max-Max Utility
        - maximize the maximal utility (extremely optimistic)
    - Index
        - calculate $u_{\max}, u_{\min}$ by max-max, max-min
        - $u = \alpha u + (1-\alpha)u_{\min}$
          ⇒ α as tge degree of optimism
    - Min-Max Regret
        - produce a regret matrix R from utility matrix U,
          where $R(a, s) =$ maximal possible utility in state $s - U(a, s)$
        - then, minimize the maximal regret, according to R
    - Common Prior

        - assign uniform distribution over states, thus reduced to decision under risk
        - yet, depends on how the states are partitioned
          i.e. the num of states affect the prob

- Unknown Prob
    - Challenge
        - action may lead to catastrophic outcomes (unforeseen outcomes)
          $\Rightarrow$ hard to decide if the chance of a severe consequence is negligible or not
          $\Rightarrow$ uncertainty towards severe consequence
    - Empirical Guideline
        - choosing / not choosing an action may have different uncertainty
          (asymmetry of uncertainty)
        - novelty: empirically, novelty brings in more uncertainty
        - spatial and temporal limitations: more limitation, less uncertainty
        - more interference with complex system, the more uncertainty

Social Decision

- Assumption
    - Conflicting Concern
        - participants usually have different/conflicting goals&concerns
- Cyclic Preference
    - No Stable Actions
        - the directed set (actions, preference) forms a directed graph
          $\Rightarrow \forall$ action a, $\exists$ action $a', a' > a$
    - Arrow's Theorem
        - 4 rationality criteria are satisfied by decision rule
          $\Rightarrow$ then cyclic preference unavoidable
        - understanding: some rationality demands are NOT compatible

Decision for Statistical Model

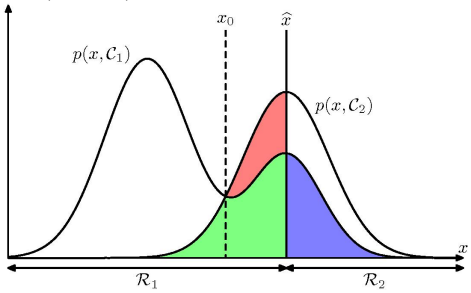
- Formulation
    - Probability Description
        - known distribution $p(\mathbf{t}|\mathbf{x})$ from model inference, where $\mathbf{x}$ the input, $\mathbf{t}$ the label
          $\Rightarrow$ all states (label) estimated with probability
- Decision Criteria
    - Minimizing Misclassification Rate (Maximizing Correct-classification Rate)
        - action: for each class k, assign a region $\mathcal{R}_k$ s.t. if $\mathbf{x} \in \mathcal{R}_k$, predict $\hat{\mathbf{t}} := \mathcal{C}_k$

        $$\Rightarrow P(\text{mistake}) = \sum_{k=1}^{K} P(\mathbf{x} \notin \mathcal{R}_k, \mathbf{t} = \mathcal{C}_k) = 1 - \underbrace{\sum_{k=1}^{K} P(\mathbf{x} \in \mathcal{R}_k, \mathbf{t} = \mathcal{C}_k)}_{P(\text{correct})}$$

        $$= 1 - \sum_k \int_{\mathcal{R}_k} p(\mathcal{C}_k|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

        $\Rightarrow$ segment $\mathbf{x}$ to be in region $\mathcal{R}_k$ governed by the maximal $p(\mathcal{C}_k|\mathbf{x})$
        i.e. predict $\hat{\mathbf{t}}$ to be the $\mathcal{C}_k$ with maximal $p(\mathcal{C}_k|\mathbf{x})$

- in binary classification ($K = 2$) with $\mathbf{x} = x$ being a scalar:



where $\hat{x}$ a sub-optimal decision region segment, x_0 the optimal one

- Minimizing Expected Loss (Maximizing Expected Utility)
  - loss for a kind of misclassification can vary from other kinds
    (e.g. recall v.s. precision)
    $\Rightarrow$ a loss function (utility function) f : (label $\mathbf{t} = \mathcal{C}_k$, pred $\mathbf{x} \in \mathcal{R}_j$) $\rightarrow$ loss L_{kj}
    $\Rightarrow$ L_{kj} as the loss of mis-classify $\mathcal{C}_k$ into $\mathcal{C}_j$
    ($\forall k, L_{kk} = 0$ & utility as negative loss)
  - expected loss $\mathbb{E}[L] = \sum_{k,j} L_{kj} P(\mathbf{x} \in \mathcal{R}_j, \mathbf{t} = \mathcal{C}_k) = \sum_{k,j} \int_{\mathcal{R}_j} L_{kj} p(\mathcal{C}_k|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$

    $\Rightarrow$ expected loss from $\mathcal{R}_j$: $\sum_k \int_{\mathcal{R}_j} L_{kj} p(\mathcal{C}_k|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$

    $\Rightarrow$ segment $\mathbf{x}$ into region $\mathcal{R}_j$ governed by the minimal $\sum_k L_{kj} p(\mathcal{C}_k|\mathbf{x})$

    i.e. predict $\hat{\mathbf{t}}$ to be the $\mathcal{C}_j$ with minimal $\sum_k L_{kj} p(\mathcal{C}_k|\mathbf{x})$

- Regret Option
  - Decision Error Source
    - the largest posterior $p(\mathcal{C}_k|\mathbf{x}) << 1$
      $\Rightarrow$ large $p(\mathbf{t} \neq \mathcal{C}_k)$ on region $\mathcal{R}_k$
    - $\Leftrightarrow$ all posteriors are comparable
  - Reject by Thresholding
    - reject to make decision when the maximal posterior $p(\mathcal{C}_k|\mathbf{x}) <$ threshold θ
      ($\theta = frac1k$ to not reject any example)
  - Reject Criterion to Minimize Expected Loss
    - consider the expected loss instead of raw posterior

2.3 Information Theory

2.3.1

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

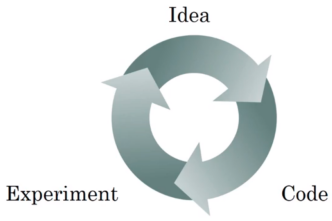
        - for deep learning: different features in same small range close to 0
          ⇒ weights for different features are in roughly the same scale
          ⇒ easier to train

- 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
        - $\Rightarrow$ average over all C_K^1 combination to indicate the generalization ability
        - extreme case: leave-out-one $\Rightarrow K = N$, where N is number of all data
    - Cons:
        - $\mathcal{O}(K) \Rightarrow$ slow, especially if training process already slow
          $\Rightarrow$ 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
          $\Rightarrow$ 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 $\Rightarrow$ NOT preferred)
    - $\Rightarrow$ 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
          $\Rightarrow$ single optimizing metric + several satisficing metrics
          $\Rightarrow$ 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
          $\Rightarrow$ tune model/network structure, optimization, preprocessing, etc.
        - evaluate on val set for good generalization ability
          $\Rightarrow$ 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
      $\Rightarrow$ need a search for the best one

Systematic Searching

- Random Sampling
  - Reason
    - NOT able to know the importance of different hyperparameters
      $\Rightarrow$ not wasting grid search step on the unimportant
    - NOT able to know the effective range of a hyperparameters
      $\Rightarrow$ may be skipped by grid search step
    - decouple the search for different hyperparameters $\Rightarrow$ 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
          ⇒ 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
          ⇒ gain knowledge through interaction & ensure a good performance
          ⇒ 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
        - ⇒ 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
          ⇒ 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 ⇒ 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
          ⇒ 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)
          ⇒ count of g_n in prediction that also appears in a reference
          ⇒ $g_n \in$ pred matched as long as $g_n \in$ a reference
        - ⇒ 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)}
          ⇒ count only unique n-gram in pred
          ⇒ $g_n \in$ pred matched up to the max num of $g_n \in$ reference
        - ⇒ clipped precision $p_n = \dfrac{\text{clipped true positive}}{\text{positive pred}} = \dfrac{\text{clipped ref count}}{\text{num of } g_n \in \text{pred}}$
        - ⇒ 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 $\exp(1 - \text{len}_{\text{pred}}/\text{len}_{\text{ref}})$
          ⇒ 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
          ⇒ 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
          ⇒ 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
        - ⇒ measure **avoidable** bias: indicate the upper-bound performance
        - ⇒ 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 ⇒ 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)

  $\Rightarrow$ 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
  - $P(\mathbf{t}|X, \mathbf{w}, \beta) = \prod_{n=1}^{N}\mathcal{N}(t_n|\mathbf{w}^T\phi(\mathbf{x}_n), \beta^{-1})$, where
    - $X = \{\mathbf{x}_1, ..., \mathbf{x}_N\}, \mathbf{t} = \{t_1, ..., t_N\}$
  - 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)
  - $\ln \text{Likelihood} = \sum_{n=1}^{N}[-\ln\frac{\beta}{\sqrt{2\pi}} - \frac{\beta}{2}(t_n - \phi(x)w)^2]$
  - $\frac{\partial}{\partial w}\ln \text{Likelihood} = \beta\Phi^T(\boldsymbol{t} - \Phi w)$
    let $\frac{\partial}{\partial w}\ln \text{Likelihood} = 0$
    $\Rightarrow w_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$
  - $\frac{\partial}{\partial \beta}\ln \text{Likelihood} = -N\beta^{\frac{1}{2}} + \beta^{\frac{3}{2}}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$
    let $\frac{\partial}{\partial \beta}\ln \text{Likelihood} = 0$
    $\Rightarrow \beta^{-1} = \frac{1}{N}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$
    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. Fishers 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 $p(w)=\mathcal{N}(w|m_0,S_0)$

$\Rightarrow \ln p(w|\boldsymbol{t})\propto -\frac{1}{2}(w-m_0)^TS_0^{-1}(w-m_0)+\sum_{n=1}^{N}[t_n\ln p_n+(1-t_n)\ln(1-p_n)]$

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: $\ln f(z)\simeq\ln f(z_0)-\frac{1}{2}(z-z_0)^TA(z-z_0)$,

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

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

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

- 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 $u_i \to +\infty$

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

$\Rightarrow Su_i = \lambda_i u_i$ (2.4)

$\Rightarrow U$ consists of eigenvectors corresponding to the first M large eigenvalue of S (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

-

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

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

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 $r(A) = r$

$$\Rightarrow r(A^T A) = r(A) = r \tag{2.15}$$

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

- 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 - $\forall n, y_n = D^{-\frac{1}{2}} V^T (x^n - \overline{x})$, where $\overline{x}$ is the mean of X

$\Rightarrow y^n$ has zero mean (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:

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

$\theta^j \in R^{n+1}$(θ_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

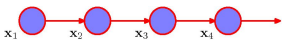
Sequential Data

- Challenge
  - Violated Identically Independent Draw Assumption
    - current data depends on previous data ⇒ i.i.d. assumption NOT hold
    - ⇒ distribution changing while drawing data
- Assumption
  - yet Gaussian still usually assumed s.t. model complexity remains in iterations (as exponential family's prior-posterior in the same family)
- Overview
  - Stationary
    - data evolves in time; generative distribution stays the same
  - Non-stationary
    - generative distribution changes along the time

11.1 Markov Model

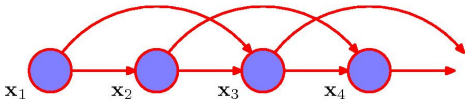
11.1.1 Markov Chain

First-order Markov Chain

- Assumption
  - each data only depends on the most recent node (direct predecessor)
- Description
  - Bayesian Networks
    - 

  - ⇒ Distribution
    - joint distribution: $p(x_1, ..., x_N) = p(x_1) \prod_{n=2}^{N} p(x_n|x_{n-1})$
    - conditional dependency: $p(x_n|x_1, ..., x_{n-1}) = \cdots = p(x_n|x_{n-1})$

Second-order Markov Chain

- Motivation
  - Modeling Trend
    - at least 2 observations to model a trend
- Description
  - Bayesian Networks
    - 

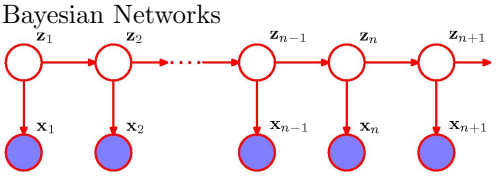
  - Distribution
    - joint distribution: $p(x_1, ..., x_N) = p(x_1)p(x_2|x_1) \prod_{n=3}^{N} p(x_n|x_{n-1}, x_{n-2})$
- Generalization: M^{th}-order Markov Chain
  - Distribution
    - joint distribution: $p(x_1, ..., x_N) = p(x_1)p(x_2|x_1) \cdots p(x_M|x_1, ..., x_{M-1}) \times \prod_{n=M+1}^{N} p(x_n|x_{n-1}, ..., x_{n-M})$
- Understanding
  - Analysis of M (assume K states for an observation)
    - $M = 0$: no markov parameter, data drawn i.i.d.
    - $M = 1$: first-order chain: $K-1$ for previous observation $\Rightarrow K(K-1)$ parameters
    - M : M^{th}-order chain: $K-1$ for each of M observation $\Rightarrow K^M(K-1)$ parameters
      $\Rightarrow$ num of parameters grows exponentially with M

11.1.2 Hidden Markov Model

Overview

- Motivation
  - Improving High-Order Markov Chain
    - avoid exponentially growing parameters in high-order Markov chain
    - avoid fixed & restricted length of dependence
  - Sequential Correlation
    - model sequential correlations in data as an extension of mixture models
- Assumption
  - Latent Variable
    - latent variables are <u>discrete</u> & form a Markov chain
      $\Rightarrow$ assumption on conditional independence
    - one latent variable z_n for each observation x_n
- Description

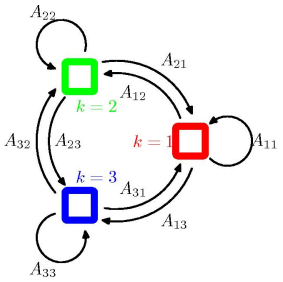
- Bayesian Networks

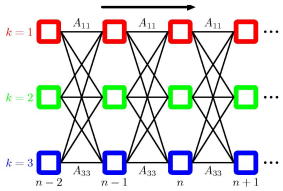


- Conditional Independence
  - $z_{n+1} \perp\!\!\!\perp z_{n-1} \mid z_n$
  - no blocked path between any two observed x_i, x_j
    $\Rightarrow$ prediction depends on all previous observation
- Distribution
  - joint distribution

$$p(x_1, ..., x_N, z_1, ..., z_N) = p(z_1)[\prod_{n=2}^{N} p(z_n|z_{n-1})] \prod_{n=1}^{N} p(x_n|z_n)$$
$$= p(z_1)p(x_1|z_1) \prod_{n=2}^{N} p(z_n|z_{n-1})p(x_n|z_n)$$

  - $\Rightarrow$ as an extension of mixture distribution:
    component densities $= p(x|z)$
    choice of component depends on previous state $= p(z_n|z_{n-1})$
- Transition Probabilities
  - encoding: latent variable z with K state, in one-hot encoding
  - $p(z_n|z_{n-1}) = \mathbf{A}_{K\times K} \in [0,1]^{K\times K}$
    where $A_{ik} = p(z_{n,k} = 1 \mid z_{n-1,i} = 1), 0 \le A_{ik} \le 1$ and $\sum_{k=1}^{K} A_{ik} = 1$
    $\Rightarrow$ num of independent parameters $= K(K-1)$
    $\Rightarrow p(z_n|z_{n-1}, A) = \prod_{k=1}^{K}\prod_{i=1}^{K} A_{ik}^{z_{n-1,i}\times z_{n,k}}$
  - $p(z_1) = \pi$
    where $\pi_k = p(z_{1k} = 1), 0 \le \pi_k \le 1$ and $\sum_k \pi_k = 1$
    $\Rightarrow p(z_1|\pi) = \prod_{k=1}^{K} \pi_k^{z_{1k}}$
  - transition diagram & unfolded transition diagram (K=3)



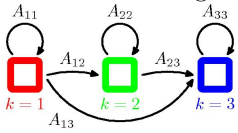


- Emission Probabilities
  - $p(x_n|z_n, \phi) = \prod_{k=1}^{K} p(x_n|\phi_k)^{z_{n,k}}$
    where ϕ is a set of parameters governing the conditional distribution

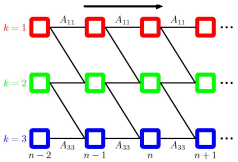
- Variants of Hidden Markov Model
  - Homogeneous HMM
    - homogeneous assumption: $\forall n = 3, ..., N, p(z_n|z_{n-1}) = p(z_{n-1}|z_{n-2})$ (transition probabilities are the same for all)
    - $\Rightarrow$ joint distribution

$$p(X, Z|\theta) = p(z_1|\pi) \left[\prod_{n=2}^{N} p(z_n|z_{n-1}, A) \right] \prod_{n=1}^{N} p(x_n|z_n, \phi)$$
$$= \prod_{k=1}^{K} \pi_k^{z_{1k}} \left[\prod_{n=2}^{N} \prod_{k=1}^{K} \prod_{i=1}^{K} A_{ik}^{z_{n-1,i} \times z_{n,k}} \right] \prod_{n=1}^{N} \prod_{k=1}^{K} p(x_n|\phi_k)^{z_{nk}}$$

    where $X = (x_1, ..., x_N), Z = (z_1, ..., z_N)$ and $\theta = \{\pi, A, \phi\}$
  - Left-to-Right HMM
    - $\forall i > j, A_{ij} = 0 \Rightarrow$ can only go to larger num state
      $\Rightarrow$ transition diagram:



    - constraint over the maximal change of state Δ, in one step
      $\Rightarrow$ unfolded transition diagram, with $\Delta = 1$:



- Maximum Likelihood Solution (via EM)
  - Goal
    - $\arg\max_{\theta} p(X|\theta)$, where likelihood $p(X|\theta) = \sum_{Z} p(X, Z|\theta)$
      $\Rightarrow$ determine model parameter θ given observations X
  - Notation
    - $\gamma(z_n) = p(z_n|X, \theta)$,
      where $\gamma(z_n) \in \mathbb{R}^K, \sum_{k=1}^{K} \gamma(z_{nk}) = 1, \forall k, \gamma(z_{nk}) \geq 0$
      $\Rightarrow \gamma(z_{nk}) = p(z_{nk} = 1|X, \theta) = \sum_{z_n} \gamma(z_n) z_{nk}$
    - $\xi(z_{n-1}, z_n) = p(z_{n-1}, z_n|X, \theta)$,
      where $\xi(z_{n-1}, z_n) \in \mathbb{R}^{K \times K}, \sum_{i,j=1}^{K} \xi(z_{n-1,i}, z_{nj}) = 1, \forall i, j, \xi(z_{n-1,i}, z_{nj}) \geq 0$
      $\Rightarrow \xi(z_{n-1,i}, z_{nk}) = p(z_{n-1,i} = 1, z_{nk} = 1|X, \theta) = \sum_{z_{n-1}, z_n} \xi(z_{n-1}, z_n) z_{n-1,i} z_{nk}$
  - E-step
    - evaluate posterior $P(Z|X, \theta) \Rightarrow$ evaluate $\gamma(z_n), \xi(z_{n-1}, z_n)$
  - M-step

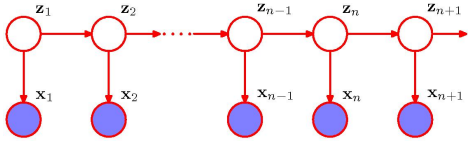
- $\theta = \arg\max_\theta \ln P(X|\theta) = \arg\max_\theta \sum_Z p(Z|X,\theta^{\text{old}}) \ln p(X,Z|\theta)$, where $\theta = \{\pi, A, \phi\}$
- $\arg\max_\theta Q = \sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k + \sum_{n=2}^{N}\sum_{i=1}^{K}\sum_{k=1}^{K} \xi(z_{n-1,i}, z_{nk}) \ln A_{ik} + \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk}) \ln p(x_n|\phi_k)$
- where full combination $\sum_Z \prod_{\text{each combination of } z_{1-n}} = \prod_X \sum_{\text{all possibility of each } x}$

◦ Critical Points

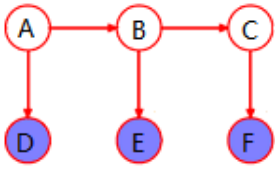
- $\pi_k = \dfrac{\gamma(z_{1k})}{\sum_{i=1}^{K} \gamma(z_{1i})}$
- $A_{ik} = \dfrac{\sum_{n=2}^{N} \xi(z_{n-1,i}, z_{nk})}{\sum_{n=2}^{N}\sum_{j=1}^{K} \xi(z_{n-1,i}, z_{nj})}$
- assume independent $\phi_k \Rightarrow$ if Gaussian $\begin{cases} \mu_k = \dfrac{\sum_{n=1}^{N} \gamma(z_{nk}) x_n}{\sum_{n=1}^{N} \gamma(z_{nk})} \\ \Sigma_k = \dfrac{\sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})} \end{cases}$

◦ Solving $\gamma(z_{nk}), \xi(z_{n-1,i}, z_{nk})$

- sum-product: based on message passing tree structure in HMM
- alpha-beta algorithm: known as forward-backward algorithm
  (two approaches are equivalent i.e. derive the same recursion formula)
- with Bayesian networks as



$\Rightarrow$ all paths through z_n is blocked conditioned on z_n
$\Rightarrow p(X|z_n) = p(x_1, ..., x_n|z_n)p(x_{n+1}, ..., x_N|z_n)$
illustration:



$$P(D,B) = \sum_A P(A,B)P(D|A)$$
$$P(F,B) = \sum_A \left(P(A,B) \sum_C P(C|B)P(F|C) \right)$$
$$P(D,F,B) = \sum_A \left(P(A,B)P(D|A) \sum_C P(C|B)P(F|C) \right)$$
$$\Rightarrow P(D|B)P(F|B) = P(D,F|B) \Rightarrow D \perp\!\!\!\perp F|B$$

6. Maximum Likelihood − EM algorithm:
- alpha-beta algorithm:
- Let $\alpha(z_n) = p(x_1, ..., x_n, z_n)$

$$\Rightarrow \alpha(z_n) = p(x_n|z_n)p(x_1,...,x_{n-1}|z_n)p(z_n) \tag{11.1}$$
$$= p(x_n|z_n)\sum_{z_{n-1}} p(x_1,...,x_{n-1},z_{n-1},z_n) \tag{11.2}$$
$$= p(x_n|z_n)\sum_{z_{n-1}} p(x_1,...,x_{n-1}|z_{n-1})p(z_n|z_{n-1})p(z_{n-1}) \tag{11.3}$$
$$= p(x_n|z_n)\sum_{z_{n-1}} \alpha(z_{n-1})p(z_n|z_{n-1}) \tag{11.4}$$

$\Rightarrow \alpha(z_1) = \prod_{k=1}^{K}[\pi_k p(x_1|\phi_k)]^{z_{1k}}$

- Let $\beta(z_n) = p(x_{n+1},...,x_N|z_n)$

$$\Rightarrow \beta(z_n) = \frac{1}{p(z_n)}\sum_{z_{n+1}} p(x_{n+1},...,x_N,z_n|z_{n+1})p(z_{n+1}) \tag{11.5}$$
$$= \frac{1}{p(z_n)}\sum_{z_{n+1}} p(x_{n+1},...,x_N|z_{n+1})p(z_n|z_{n+1})p(z_{n+1}) \tag{11.6}$$
$$= \sum_{z_{n+1}} p(x_{n+2},...,x_N|z_{n+1})p(x_{n+1}|z_{n+1})\frac{p(z_n|z_{n+1})p(z_{n+1})}{p(z_n)} \tag{11.7}$$
$$= \sum_{z_{n+1}} \beta(z_{n+1})p(x_{n+1}|z_{n+1})p(z_{n+1}|z_n) \tag{11.8}$$

$\Rightarrow \beta(z_N) = 1$, solved from $\gamma(z_N) = \frac{\alpha(z_N)\beta(z_N)}{p(X)}$

- $\gamma(z_n) = p(z_n|X) = \frac{P(X|z_n)p(z_n)}{p(X)} = \frac{\alpha(z_n)\beta(z_n)}{p(X)}$

- $1 = \sum_{z_n}\gamma(z_n) = \frac{\sum_{z_n}\alpha(z_n)\beta(z_n)}{p(X)}$

$\Rightarrow p(X) = \sum_{z_n}\alpha(z_n)\beta(z_n)$

More conveniently, let $z_n = z_N \Rightarrow p(X) = \sum_{z_N}\alpha(z_N)$

$\Rightarrow$ complexity: $\mathcal{O}(n)$, instead of $\mathcal{O}(2^n)$, where n is the length of the chain

- $\xi(z_{n-1},z_n) = \frac{\alpha(z_{n-1})p(x_n|z_n)p(z_n|z_{n-1})\beta(z_n)}{P(X)}$:

$$\xi(z_{n-1},z_n) = p(z_{n-1},z_n|X) = \frac{p(X|z_{n-1},z_n)p(z_{n-1},z_n)}{p(X)} \tag{11.9}$$
$$= \frac{p(x_1,....,x_{n-1}|z_{n-1},z_n)p(x_n,...,x_N|z_{n-1},z_n)\times p(z_n|z_{n-1})p(z_{n-1})}{p(X)} \tag{11.10}$$
$$= \frac{p(x_1,...,x_{n-1}|z_{n-1})p(x_n|z_n)p(x_{n+1},...,x_N|z_n)\times p(z_n|z_{n-1})p(z_{n-1})}{p(X)} \tag{11.11}$$
$$= \frac{\alpha(z_{n-1})p(x_n|z_n)p(z_n|z_{n-1})\beta(z_n)}{p(X)} \tag{11.12}$$

where factorizing using conditional independence : $\begin{cases}[x_1,...,x_{n-1}],[z_n] & \text{on } z_{n-1}\\ [x_n,...,x_N],[z_{n-1}] & \text{on } z_n\\ [x_n],[x_{n+1},...,x_N] & \text{on } z_n\end{cases}$

($p(A|B) = p(A)$ when $A \perp\!\!\!\perp B$)
- Algorithm description:
- Initialize $\theta = \{\pi, A, \phi\}$
- E step:
Forward recursion for $\alpha(z_n)$
Backward recursion for $\beta(z_n)$
Calculate $\gamma(z_n), \xi(z_{n-1}, z_n)$
- M step:
Maximize $Q(\theta, \theta^{\text{old}})$ using critical points
- Regularized EM
- add the prior of π, A, ϕ, in the form of $\log p(\theta)$, into $Q(\theta, \theta^{\text{old}})$ before maximization
7. Prediction
- Goal
- predict x_{N+1} given $X = \{x_1, ..., x_N\}$
- Algorithm
- α recusion + summing over z_N

$$p(x_{N+1}) = \sum_{z_{N+1}} p(x_{N+1}, z_{N+1}|X) \tag{11.13}$$
$$= \sum_{z_{N+1}} p(x_{N+1}|z_{N+1,X}) p(z_{N+1}|X) \tag{11.14}$$
$$= \sum_{z_{N+1}} p(x_{N+1}|z_{N+1}) \sum_{z_N} p(z_{N+1}|z_N, X) p(z_N|X) \tag{11.15}$$
$$= \sum_{z_{N+1}} p(x_{N+1}|z_{N+1}) \sum_{z_N} p(z_{N+1}|z_N) p(z_N|X) \tag{11.16}$$
$$= \frac{1}{P(X)} \sum_{z_{N+1}} p(x_{N+1}|z_{N+1}) \sum_{Z_N} p(z_{N+1}|z_N) \alpha(z_N) \tag{11.17}$$

$\Rightarrow$ store the $\alpha(z_t)$ for predicting x_{t+1} **and** computing $\alpha(z_{t+1})$ once x_{t+1} observed
- Intuition: information in $x_1, ..., x_N$ stored in $\alpha(z_N)$
$\Rightarrow$ enable real-time application
8. Scaling Factors
- Original $\alpha - \beta$ Recursion
- $\alpha(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n|z_{n-1})$ - $\beta(z_n) = \sum_{z_{n+1}} \beta(z_{n+1}) p(x_{n+1}|z_{n+1}) p(z_{n+1}|z_n)$
$\Rightarrow \alpha \to 0$ exponentially quickly to length of chain
$\Rightarrow$ for long chain (100), α can exceed the dynamic range of computer
- Normalized and Rescaled α
- normalize: $\hat{\alpha}(z_n) = \dfrac{\alpha(z_n)}{p(x_1, ..., x_n)} = p(z_n|x_1, ..., x_n)$
$\Rightarrow$ stay in the dynamic range
- let rescale factor $c_n = p(x_n|x_1, ..., x_{n-1})$

$$\Rightarrow p(x_1, ..., x_n) = \prod_{m=1}^{n} c_m \tag{11.18}$$
$$\Rightarrow \alpha(z_n) = (\prod_{m=1}^{n} c_m) \cdot \hat{\alpha}(z_n) \tag{11.19}$$
$$\Rightarrow c_n \hat{\alpha}(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \hat{\alpha}(z_{n-1}) p(z_n|z_{n-1}) \tag{11.20}$$

- Rescaled β

- let normalization $\hat{\beta}(z_n) = \dfrac{\beta(z_n)}{\prod_{m=n+1}^{N} c_m} = \dfrac{p(x_{n+1},...,x_N|z_n)}{p(x_{n+1},...,x_N|x_1,...,x_n)}$

$\Rightarrow c_{n+1}\hat{\beta}(z_n) = \sum_{z_{n+1}} \hat{\beta}(z_{n+1})p(x_{n+1}|z_{n+1})p(z_{n+1}|z_n)$ note: c_{n+1} can be re-used from α recursion

- EM under Rescaled $\alpha - \beta$

- monitoring likelihood: $p(X) = \prod_{n=1}^{N} c_n$

- E step:

$$\Rightarrow \gamma(z_n) = \hat{\alpha}(z_n)\hat{\beta}(z_n) \tag{11.21}$$

$$\xi(z_{n-1}, z_n) = c_n\hat{\alpha}(z_n)p(x_n|z_n)p(z_n|z_{n+1})\hat{\beta}(z_n) \tag{11.22}$$

9. Viterbi Algorithm (max-sum algorithm)

- Goal:

- find the most probable sequence of latent variable

$\Rightarrow$ find $\max_Z p(Z|X,\theta)$, where Z is sequence of latent states

- compare with: set of states being individually most probable $\Rightarrow \forall n$, find $\max_{z_n} p(z_n|X,\theta)$

$\Rightarrow$ maximize $\gamma(z_n)$ for all n

- efficiency: searches space of paths efficiently ($\mathcal{O}(n)$ to the length of chain)

- Notation:

- $w(z_n) = \max_{z_1,...,z_{n-1}} \ln p(x_1,...,x_n,z_1,...,z_n)$

note: $w(z_n)$ is a function of z_n, with log probability maximized over $z_1,...,z_{n-1}$

- Recursion from Joint Distribution of HMM:

- $w(z_{n+1}) = \ln p(x_{n+1}|z_{n+1}) + \max_{z_n}\{\ln p(z_{n+1}|z_n) + w(z_n)\}$

$w(z_1) = \ln p(x_1, z_1) = \ln p(z_1) + \ln p(x_1|z_1)$

- Backtrack

- maximization over $z_n \Rightarrow$ individually done for each of the K states of z_{n+1}

- maintain a matrix record for each maximization:

let $\phi(k_n)$ be the state of z_n when $w(z_{n+1})$ getting maximum given $z_{n+1} = k$ (in state k)

$\Rightarrow k_n^{\max} = \phi(k_{n+1}^{\max})$, where $k_n^{\max}$ is the desired state of z_n

$\Rightarrow k_{N-1}^{\max} = \phi(k_N^{\max}) = \phi(\arg\max_{z_N} w(z_N))$

11.2 Linear Dynamic System

1. Goal

- Continuous Latent Variable

- sum becomes integral

- practical sense $\Rightarrow$ multivariated Gaussian distribution assumed

(so that complexity of posterior dose NOT increase)

- Senquential Correlation in Contiuous Data

- an extension to continous latent variable model (such as probablistic PCA)

2. Notation

- Underlain Procedure - transition: $z_n = Az_{n-1} + w_n$, where noise $w \sim \mathcal{N}(w|0,\Gamma)$ - emission: $x_n = Cz_n + v_n$, where noise $v \sim \mathcal{N}(v|0,\Sigma)$ - initialization: $z_1 = \mu_0 + \mu$, where noise $\mu \sim \mathcal{N}(\mu|0,V_0)$ - Probabilities - transition: $p(z_n|z_{n-1}) = \mathcal{N}(z_n|Az_{n-1},\Gamma)$ - emission: $p(x_n|z_n) = \mathcal{N}(x_n|Cz_n,\Sigma)$ - initialization: $p(z_1) = \mathcal{N}(z_1|\mu_0,V_0)$ - Model Parameters - $\theta = \{A,\Gamma,C,\Sigma,\mu_0,V_0\}$

3. Maximum Likelihood - EM
- E step
- Inference problem - determine the local posterior marginals for latent variables (sum-product algorithm)
- M step

4. Linear-Gaussian Model Features
- sequence of individually most probable latent variable $\Leftrightarrow$ the most probable latent sequence
$\Rightarrow$ no need for Viterbi algorithm
- Joint Distribution
- $P(X,Z) = p(z_1)\left[\sum_{n=2}^{N} p(z_n|z_{n-1})\right]\sum_{n=1}^{N} p(x_n|z_n)$ - same form as HMM
$\Rightarrow$ an Gaussian (product of Gaussians)
$\Rightarrow$ standard result available for its marginals and conditionals
$\Rightarrow$ sum-product algorithm for faster computation

5. Inference
- Goal
- determine marginal distribution $P(Z|X)$ - prediction: $P(z_n, x_n|x_1, ..., x_{n-1}.\theta)$ - used in real-time application
- Sum-Product Algorithm (Kalman Filter + Kalman Smoother)
- analogous to alpha-beta algorithm in HMM
$\Rightarrow \hat{\alpha}(z_n) = p(z_n|x_1, ..., x_n)$, factor $c_n = p(x_n|x_1, ..., x_{n-1})$
$\Rightarrow c_n\hat{\alpha}(z_n) = p(x_n|z_n)\int_{z_{n-1}} \hat{\alpha}(z_{n-1})p(z_n|z_{n-1})dz_{n-1}$

$$\Rightarrow c_n\hat{\alpha}(z_n) = c_n\mathcal{N}(z_n|\mu_n, V_n) \tag{11.23}$$
$$= \mathcal{N}(x_n|Cz_n, \Sigma)\int \mathcal{N}(z_{n-1}|\mu_{n-1}, V_{n-1})\mathcal{N}(z_n|Az_{n-1}, \Gamma)dz_{n-1} \tag{11.24}$$
$$= \mathcal{N}(x_n|Cz_n, \Sigma)\mathcal{N}(z_n|A\mu_{n-1}, P_{n-1}), \tag{11.25}$$
$$\text{where } P_{n-1} = AV_{n-1}A^T + \Gamma(\text{ by integral of } \mathcal{N}\cdot\mathcal{N}) \tag{11.26}$$

$$\Rightarrow \mu_n = A\mu_{n-1} + K_n(x_n - CA\mu_{n-1}) \tag{11.27}$$
$$V_n = (I - K_nC)P_{n-1} \tag{11.28}$$
$$c_n = \mathcal{N}(x_n|CA\mu_{n-1}, CP_{n-1}C^T + \Sigma), \tag{11.29}$$
$$\text{where } K_n = P_{n-1}C^T(CP_{n-1}C^T + \Sigma)^{-1}, \text{ known as Kalman gain matrix} \tag{11.30}$$

- Initial Condition
$\Rightarrow c_1\hat{\alpha}(z_1) = p(z_1)p(x_1|z_1)$, where $z_1 = p(x_1)$

$$\Rightarrow \mu_1 = \mu_0 + K_1(x_1 - C\mu_0) \tag{11.31}$$
$$V_1 = (I - K_1C)V_0 \tag{11.32}$$
$$c_1 = \mathcal{N}(x_1|C\mu_0, CV_0C^T + \Sigma), \tag{11.33}$$
$$\text{where } K_1 = V_0C^T(CV_0C^T + \Sigma)^{-1} \tag{11.34}$$

- Interpretation
- $A\mu_{n-1}$: predicted mean of z_n, by projecting mean of z_{n-1} one step forward - $CA\mu_{n-1}$: predicted observation x_n, by emitting from predicted mean of z_n - $x_n - CA\mu_{n-1}$: error between predicted observation and actual observation - K_n : coefficient of error, giving a correction to the predicted mean of z_n
$\Rightarrow$ making successive predictions & correcting them in the light of new observation

6. EM - Learning Model Parameters

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 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, planning, 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

      $\Rightarrow$ 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 $Z_{00} - Z_{NM}$)
      $\Rightarrow$ a matrix with n as row entry, m as column entry, mapping to both $\ln P(Z,X|\theta), P(Z|\theta_n,X)$
      $\Rightarrow$ 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

Direction - Segmentation

- Multi-scale in Segmentation
  - Per-pixel Classification
    - not only skipping from encoder to decoder, but also skipping from tunnel to decoder final output layer, by e.g. tiling.

Direction - Detection

- Faster One-stage
  - 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
      $\Rightarrow$ not possible to lose detection due to griding
    - one-step further from ExtremeNet:
      directly segmentation, then use boundary pixel to re-organize into bbox
- Two-stage in One-stage
  - Attention
    - attention map as 1st box proposal, as extreme net (instead of RoI)
- **No Non-Maximum Suppression**
  - Auto Filtering
    - regress threshold of objectness as well
      $\Rightarrow$ objectness under threshold not considered by encoder-decoder
      (under the GAN framework ?)
      (under multi-tasking ?)
  - CRF
    - still not end2end, yet trainable
  - RNN Encoder-Decoder
    - encode all bbox / spatial feature, then decode (generate) till end-of-sequence
      (as YOLOv1 still use dense layer...)

Direction - Tracking

- Current State - 2019
  - NN in Data Association
    - pred association prob of prediction-detection in JPDA
    - pred prob of detection being true / false alarm (the existence prob for bbox), used in MHT
  - Tracking Initiation & Deletion
    - currently, by rules...
      e.g. 3 observations in 4 consecutive frames; observation missed for 4-7 frames
  - Prediction Encoding
    - crop the image according the predicted area (gating)
  - Deep Backbone

        - deeper wider siamrpn

- NN in Gating
    - Region Proposal Network
        - propose the search region (gating) by RPN
          (instead of using rule based on current track location)
          ⇒ predict a large region with RPN based on track info
          (down by siamRPN?)

- End-to-end / unified framework
    - End-to-End Regression for Data Association in MOT
        - cite: Online Multi-Target Tracking Using Recurrent Neural Networks
        - directly output association decision
        - input: raw image (semantic), current detection, track prediction, track history
        - track-level info: extended Siamese for relation of: track - current detection
          (instead of only det-det in 2 frames)
        - fully NN approach for JPDA
          ⇒ track - all det: concat (tile) track+pred to each det ⇒ use fcn, then pooling to regress to a bbox for update
          (direct regress the box, instead of Siamese net + weighted sum)
          (cite: Data-Driven Approximations to NP-Hard Problems)
        - ⇒ fast JPDA: single RNN encode all detections as D, then regress for each track-D for update
          (instead of num of track × num of det, now only num of track + num of det)
        - fully NN approach for MHT ???
          trace-back available ... encoding past detection as well? (in the tracker?)
          N-scan pruning?
    - Track Initiation/Deletion
        - directly output decision result
          ⇒ regress the decision boundary as well (instead of universal 0.5)
    - Interesting Predicted Bounding Box Encoding: attention instead of cropping
        - direct encoding: x,y,w,h, o, + encoding of full image
        - mask: bounding box plot onto a separate mask, concat to the (encoding of) full image
          further, attention mechanism
          (soft-crop ???)
          ⇒ naturally develop into **instance tracking**, even tracking in point cloud data
        - ⇒ **crop with extension** can significantly draw back the wall time
          (due to: 1. crop in cpu & for each object; 2. crop is not generally gpu accelerated)
          especially, when object is large in the image ⇒ need to pad a lot
          (e.g. baidu field-end tracking: car emerging/leaving right under the camera)
        - two masks: one for all bbox from detector, one for current track

      ⇒ enable global track for multiple obj track (single RNN for all track in an image)
    - Scale Invariance
        - current siamRPN++: trained with a fixed-scale resized image crop as input
          ⇒ if object is non-rigid: pixel size can change significantly between frames
          ⇒ crop image according to last frame would contain crop out far more/less background than expected
          ⇒ wrong percentage of fore-/back-ground in the image crop

⇒ model input (as resized to fixed size) has different object scale than training
⇒ failed to track & fall back to only detection (as containing RPN)
(reproduced by having different context_amount in inference-training)

- siamFC: in test time, extract template at multiple (e.g. 3) scales for each target
- HENCE, try to have scale invariance to address abrupt size change in target

- SOT
  - directly use cnn + conv-lstm
- single Obj Tracking for MOT
  - common cnn encodes image t as f_t, detected bbox as D_t (given or trained)
    (prefer to train, as auxiliary loss for multi-tasking ?)
  - encode each detection $d_t \in D_t$ as $c(d_t) \in c(D_t)$
    empty detection always $\in D_t$
    ⇒ track may always have NO compatible det
  - one rnn tracker for each $c(d_t)$
  - rnn takes in encoded previous prediction $c(p_{t-1})$ (produced by itself), with f_t, and each encoded det $c(d_t) \in c(D_t)$
    ⇒ produce as many track as D_t, each with a probability
    (control the track's death)
    ⇒ only the highest remains, others deleted
  - remaining track rnn regress pred bbox p_{t+1} (an encoded bbox as well)
    (a decoder at t trying to induce the d_{t+1})
    (GAN ?)
- MOT as SOT
  - attention mask contain multiple interest
    ⇒ not able to handle birth/death of independent object
    (as modeled all as a whole)
- End-to-End MOT
  - design requirement
    - arbitrary start of track
    - arbitrary num of track
    - arbitrary end of track
- End-to-end MOT with States (Markov Decision Process / Deterministic Finite Automaton)
  - action at each state given by NN
  - history of track in RNN ⇒ provided when making decision
    ⇒ RNN modeling all the state ???
    (update with different set of weights on different chosen actions ?)
    (design transition instead of states: states as the closure of all actions ?)
  - design state for each challenge scene separately (out-of scene, occlusion, long-term lost, etc.)
    ⇒ directly tackle each scene
    (state growing: auto-discover state ???)
  - RNN for actions ?
    output score for all actions, only legal actions (given current state) considered&selected, then transfer state accordingly
  - trained with RL

- Training

- MDP as Hard-Negative Mining
  - yet, hard negative discovered in DaSiam / siam cascaded-RPN
    (in terms of data imbalance)
- Sparse Detection
  - feed only a few detection, demand NN to fill up using track history&prediction
- Other NLP Training
  - training of n-gram model used on rnn traker ?

- Fixable Track
  - Fixing After Lost
    - backward rnn for re-associated track after lost
      ⇒ to fix the previous prediction given current observation
      (then recompute the forward rnn for consecutive tracking)

- SiamRPN Tracking
  - further branches after xcorr?
    - as currently, xcorr to directly produce regression
      (followed by a 1x1 conv to adjust channel for classification)
      ⇒ equip with a further conv pipe? (including 3x3) to further utilize semantic similarity?
      (enable deep power after xcorr)
  - padding removal for deep power
    - siamRPN++: cropping for template branch & data augmentation for search branch
    - siamDW: crop-inside unit: as early as where the padding is introduced
    - ⇒ crop before xcorr & before final regression

    ⇒ try to break the 20-layer limits in siamDW
    (20-layer limit not happened in siamRPN++, possibly due to cropping???)
    (or, is it data augmentation always demanded to embrace deep power???)

  - ⇒ **review another factor for central bias**
    - data aug really solves: central bias is the thing
    - crop NOT solve completely ⇒ other factors causing central bias
    - perhaps due to the intrinsic central bias in conv ???
  - ⇒ avg padding (use channel-wise mean, instead of 0, to pad)
    - conv contains intrinsic central bias: center pixel get more convolution
      ⇒ padding actually **reserve** border info
      ⇒ keep the upside & remove the downsid of padding
      i.e. NOT introduce location signal (0), but channel-wise avg
    - if it works: proof that central bias is introduced in deep tracker:
      - intrinsically from conv
      - by 0-signal from 0-padding & mostly centralized label box
  - Removing Post-process
    - cosine window: seems to be solved in siamrpn++ (by deep power?)
    - score penalty: to be tested

- GAN to Improve Transformation Learning
  - auxilary loss in siam tracker to distinguish from true search target, or a fake search region
  - may train the image patch generator first with a simple classifier
  - then append the classifier to the siam-tracker ⇒
    - use the gradient from GAN classifier branch to boost siam performance
    - use the generated data for more diverse training data

Directoin - Referring Seg

- Integrating Encoder-Decoder Architecture
  - Upsampling
    - similar to Unet, concat low-level spatial info
    - introduce language info as well
      (e.g. early combination, explicit introducing, ...)
- Info Early Fusion
  - Tiling at First Conv
    - as siamese net for joint input
    - 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

Directoin - Tactile Perception

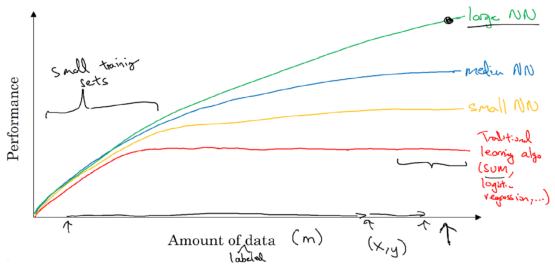
- Close-range Camera as Tactile Sensor
  - directly location mapping between camera & tactile sensor via IMU
    ⇒ jointly process in cnn

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 **!**

- Maxout Activation $a = \max(Wz + b, \text{axis=0})$
  - Operation
    - a/z the col vector for input/output with dimension of d_i/d_o
      (k a hyper parameter)
      $\Rightarrow W_{[k \times d_o \times d_i]}$ & $b_{k \times d_o}$
      $\Rightarrow W \cdot z + b \to [k, d_o]$
    - finally, take element-wise max over k candidate
  - Pros
    - multi-linear approximation
      $\Rightarrow$ use k linear functions to approximate convex function
      (due to the max ops)
    - i.e. can learn a piecewise linear, convex function with up to k pieces
  - Cons

        - more params & hyper param
    - Understanding
        - actually, append one more linear layer
        - in conv ops: a channel-wise maxpool as the activation

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
        - $\Rightarrow$ address gradient exploding & vanishing
          (as responses are controlled, thus backward gradients are also regularized accordingly)

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 ⇒ units in same layer computing exactly same function
          ⇒ get the same learning step propagated back
          ⇒ 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 $\forall l \in [1, L], \text{Var}(w^l) = \frac{1}{n_l}$ 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
          ⇒ mitigate gradient vanishing& exploding problem statistically

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

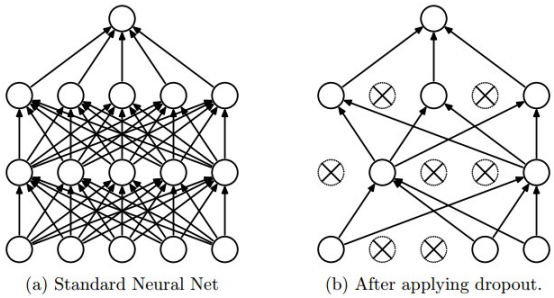
Regularization

- $L2$ Regularization
    - Definition
        - ⇒ 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
              ⇒ activation becomes more linear-alike (e.g. sigmoid, tanh)
              ⇒ layers perform more linear-alike transformation

          ⇒ 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 ⇒ 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
    $\Rightarrow$ 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

  $\Rightarrow$ 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
    - $\Rightarrow$ average over past gradients with exponentially decaying weight,
      $\Rightarrow$ 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
      $\Rightarrow$ β 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 Loss

Probabilistic Loss

- 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
- Smooth L1 Loss
  -

Metric Loss

- Triplet Loss
-

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 $a_k^L = \frac{e^{(z_k^L)}}{\sum_{k=1}^{K} e^{(z_k^L)}}$
  - 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: $L = \frac{1}{N} \sum_{\mathbf{Y}} -\mathbf{y}^T \cdot \log \hat{\mathbf{y}}$,
          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
    - Input
        - arbitrary input $\mathbf{z}^L$ being logits, considered as a flatten tree
    - Output
        - a tree structure, flatten into 1-D array
    - Operation
        - for each node (if not leaf), a softmax to predict prob for its direct children
          $\Rightarrow$ multiple softmax connected to various input nodes in $\mathbf{z}^L$
    - Implementation
        - create a Bayesian network with a single root node (e.g. "obj")
          $\Rightarrow$ each node being a conditional probability conditioned on its direct parent
        - for absolute probability of each class (represented by a node)
          $\Rightarrow$ apply sum rule & product rule accordingly
    - Understanding
        - NOT assuming mutual exclusion between class
        - $\Rightarrow$ enable graceful degrade in classification
          e.g. can still recognize by p(animal), if failed with p(cat), p(dog), etc.
        - enable joint training with multiple datasets (involving classification)
          e.g. YOLOv2(YOLO9000)

- Normalization

Convolution Layer

- Convolution $1D$
    - Input
        - $1D$ vector of size i, can have multiple channels at each location
    - Setting
        - kernel size k: another 1D vector (weight vec)
        - padding p: the size to pad at one direction of an axis
          (usually symmetric padding at both direction $\Rightarrow 2p$ in total)
        - stride s: the step for kernel to move its location as sliding across input
    - Operation

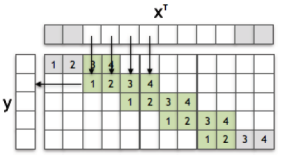
- $s(t) = (f * g)(t) = \sum_{x=-\infty}^{\infty} f(x)g(t-x)$
- padding: to extend the x range where $f(x)$ is defined, by filling default values
- strides: the step / gap between 2 consecutive kernel locations
- flip the kernel vector $g(x)$ & slides it across input vector $f(x)$ with stride s
- for each position t:
  an element-wise weighted sum on the spatially corresponding position
  (overlap = range where $f(x), g(t-x)$ are both defined)
- sum across channels: sum over kernel output from each channel + optional bias
- kernel strides spatially across the image, with stride along each axis defined
  $\Rightarrow$ to produce a **1**-channel output
  $\Rightarrow$ for multi-channels output: multiple sets of kernels

◦ Output

- output size o = possible placements of kernel on input
  (channel depends on the num of sets of kernels)
  $\Rightarrow o = \left\lfloor \frac{i+2p-k}{s} \right\rfloor + 1$
- for $s > 1$, all input size $\{j = i + a | a = 0, ..., s-1\}$ will produce same output size
  (complicate the analysis of transposed conv)

◦ Implementation - Img-to-Col

- pre-calc the function $g(t-x)$ for all possible t (as output size known)
  $\Rightarrow$ fill in the table that corresponds to input size:



$$= \begin{pmatrix} w_1 & w_2 & w_3 & w_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & w_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 \end{pmatrix}$$

  where $\mathbf{x}$ = input $f(x)$ after padding (gray cells), $\mathbf{y} = s(t)$ the output,
  with $i = 8, k = 4, p = 2, s = 2$, thus $o = 5$
- $\Rightarrow$ each row of the matrix = kernel $g(t-x)$ with a given t
  i.e. each row represents kernel at a specific position
  - row num = possible kernel position = output size
  - col num = input size (after padding)

◦ Back Propagation (TODO: study matrix derivatives)

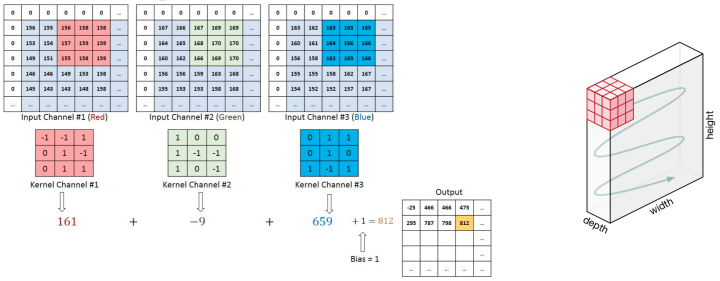
- with above matrix (denoted as C): $\mathbf{y} = C\mathbf{x}$
- to update kernel weights: $\frac{\partial}{\partial C} L = \frac{\partial L}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial C} = \frac{\partial L}{\partial \mathbf{y}} \cdot \mathbf{x}^T$
  as $\frac{\partial \mathbf{y}}{\partial C} = \begin{bmatrix} - & \mathbf{x}^T & - \\ & \vdots & \\ - & \mathbf{x}^T & - \end{bmatrix}$ & $\frac{\partial L}{\partial C} = \begin{bmatrix} - & \frac{\partial L}{\partial \mathbf{y}}[1] \cdot \mathbf{x}^T & - \\ & \vdots & \\ - & \frac{\partial L}{\partial \mathbf{y}}[5] \cdot \mathbf{x}^T & - \end{bmatrix}$
  $\Rightarrow$ equivalent to $\frac{\partial L}{\partial \mathbf{y}} \cdot \mathbf{x}^T$, as $\frac{\partial L}{\partial \mathbf{y}}$ a col vec as $\mathbf{y}$
  (still, needs to sum up grad for the same weights)
- to backprop: $\frac{\partial}{\partial \mathbf{x}} L = \frac{\partial L}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = C^T \cdot \frac{\partial L}{\partial \mathbf{y}}$
  as $\frac{\mathbf{y}_i}{\mathbf{x}_j} = C_{i,j}$

◦ Understanding

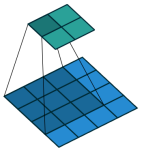
- the basis for N-D conv: as the relations are analogy across axises
- kernel defines both forward & backward pass
  $\Rightarrow$ depending on how to use the C and C^T

- Convolution $2D$

- Input
  - spatially $2D$ feature maps, usually with multiple channels
- Setting
  - kernel size k: a 2D matrix (weights) to be convoluted
    $\Rightarrow$ usually odd square matrix - even becomes a convention
    (to avoid asymmetric padding & a central pixel for filter location)
    $\Rightarrow k \times k$ kernel
  - padding p: as in 1D conv (usually, same p for all axises)
  - stride s: as in 1D conv (usually, same s for all axises)
- Operation (similar to 1D conv)
  - $S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$
    $\Rightarrow$ slide along both axises & produce a 2D function
    (hence 2D conv)
  - kernel (weights) K structured as a matrix (for each input channel)
  - slide & sum on 2D spaces



  - each kernel a input channel & a set of channel for 1-channel output
    $\Rightarrow$ multiple sets of kernels for multi-channel output
  - activation then taken after convolution operation, element-wisely
- Output
  - as in 1D conv: $o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$
    for $s > 1$, all input size $\{j = i + a | a = 0, ..., s-1\}$ will produce same output size
- Implementation
  - implement cross-correlation instead of convolution
    (skip the flipping operation: as their results are symmetric)
    yet, convolution is associative, due to the flipping
  - create a matrix with: each row = unrolled kernel at a specific location
    (similar to 1D conv)



with $i = 4, k = 3, p = s = 0$, construct matrix:

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

- Back Propagation
  - similar to 1D conv above
- Understanding - Padding
  - prevent output feature maps from spatially shrinking

        - prevent info lost on the border & corner of image
          (compared to the central part of feature map, multiplied less with the kernel)
          ⇒ ensure border feature get convoluted by the same times as central feature
        - YET, may introduce noise if using large kernel
          (as need to pad a lot & introduce overwhelming useless info, e.g. 0s)
        - convention: 0-padding on both directions of an axis
        - padding style:
          - valid (no) padding: $p = 0$
          - half (same) padding for odd kernal $k = 2n + 1$ and $s = 1$
            ⇒ $p = n$ to have $o == i$
          - full padding: $p = k - 1$ to account all possible overlaps of kernel & feature map
    - Understanding - Kernel Size
        -
    - Understanding - Stride
        - a form of subsampling (how much output is retained)
          ⇒ subsample every 1 in s locations from the stride-1 feature map along each axis
    - Understanding
        - a linear transformation that preserves topological info & ordering
        - v.s. hand-designed filter: learn & optimize the goal with the help of 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
          ⇒ robust to spatial variance (as a high-pass bandwidth)

- 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)
        - channel-wise interaction

- Transposed Convolution
    - Input
        - feature vec/map/volume, etc.

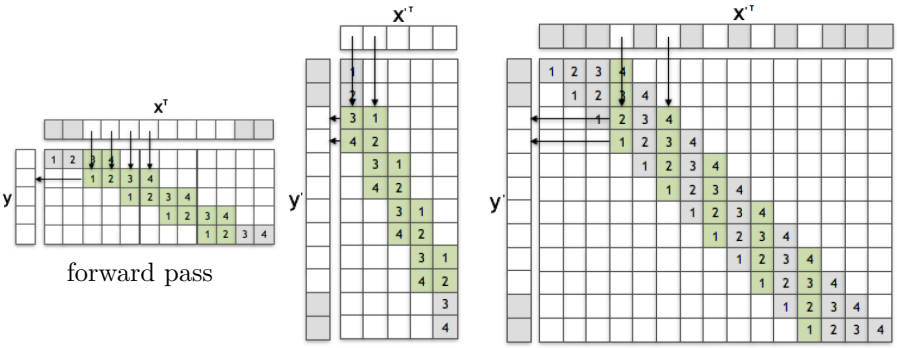
- Setting
  - consider only one axis
    (can generalize to N-D input/kernel with all axises sharing the same setting)
  - original conv: i, a, k, s, p, o, where $i + 2p - k \mod s = 0$ & $a = 0, ..., s - 1$
    (to unleash the constraint from $\lfloor\rfloor$ ops)
  - input size i', output size o'
  - kernel size k', stride s', padding at one direction p'
  - f the num of location to be inserted into 2 consecutive input location
  - a' the num of padding to add to **1** side of the axis (additional padding)
- Operation
  - given a direct conv with setting i, k, s, p, o
    $\Rightarrow$ its transposed conv is the ops to have $i' = o$ and $o' = i$
    i.e. to recover the original input shape & maintain the correspondence
  - assume $s' = 1, k' = k$, then set $p' + p = k - 1, a' = a, f = s - 1$
  - conv as usual on output $o = i'$ (after manipulated by p', a', f)
- Output
  - feature with size of $o' = i$
- Derivation - Mathematically
  - assume $s' = s = 1, k' = k$
    $$\Rightarrow \begin{cases} \lfloor \frac{i + 2p - k}{s} \rfloor + 1 \ = i + 2p - k + 1 \ = o & \text{direct conv} \\ \lfloor \frac{o + 2p' - k}{s} \rfloor + 1 = o + 2p' - k + 1 = i & \text{transposed conv} \end{cases}$$
    $\Rightarrow p + p' = k - 1$
    thus, under special case of $s = 1$:
    - zero-full padding are transposed conv for each other
    - same padding is transposed conv for itself (as shape does NOT change at all)
  - more general case: assume $s' = 1, k' = k$
    $$\Rightarrow \begin{cases} \lfloor \frac{i + a + 2p - k}{s} \rfloor + 1 = \frac{i + 2p - k}{s} + 1 = o & \text{direct conv} \\ \lfloor \frac{o + 2p' + a' + f(o-1) - k}{s'} \rfloor + 1 = o + a' + 2p' + f(o-1) - k + 1 = i + a & \text{transposed conv} \end{cases}$$
    $$\Rightarrow \text{given } \begin{cases} 2p' + a' + f(o-1) = a + 2k - 2 - 2p + (s-1)(o-1) \\ 0 \le a \le s - 1 \\ 0 \le p \le k - 1 \end{cases}$$
    find p', a', f
    s.t. $0 \le p' \le k - 1$ to have valid overlap with kernel of transposed conv
    $0 \le a' \le k - 1$
    $0 \le p' + a' \le k - 1$
    $0 \le f \le k - 1$
    $\Rightarrow$ one intuitive solution is thus: $p + p' = k - 1, a' = a, f = s - 1$
- Derivation - Intuitively
  - to maintain the correspondence of the original direct conv (in the opposite way):
  - recover the original output $o_{s=1} = i + a + 2p - k + 1$
    - stride s as subsampling on $o_{s=1} \Rightarrow f = s - 1$
    - $a \le l - 1$ as skipped input at the tail of an axis/dim of $o_{s=1} \Rightarrow a' = a$

    (besides, $o_{s=1} = o + (s-1)(o-1) + a$)

- ⇒ then, could have $s = 1$ assumption
- p as additional border info/mapping that is not required to account
  ⇒ the border output (e.g. $o[0] = i'[0]$) corresponds to original $k - p$ input
  (while the border location recovers $\lfloor \frac{p'+1}{s'} \rfloor = p' + 1$ location via trans conv)
  ⇒ $p' + 1 = k - p$, i.e. $p' + p = k - 1$

◦ Implementation

- using the backward of a conv: transposed conv via C^T
- given direct conv, where $x = i, y = o$
  ⇒ have its backward & transposed conv with $x' = y, y' = x$ (in shape & location)



where gray in x^T the padding,
gray in y' the places to be cropped,
gray in x'^T the compensated location & padding

- ⇒ each transposed conv equivalent to a conv backward (after a kernel re-arrange)
  (since kernel are learnable) ⇒ implement as swapping back/forward-prop of normal conv

◦ Understanding

- a learnable version of inverse conv
  ⇒ to maintain the correspondence between original input & output
  ⇒ also, to recover the shape of the original input of the direct conv
  (yet, not recovers the content, but learning a new mapping)

• Dilated/Atrous Convolution

◦ Input

- 2D feature map

◦ Setting

- size: input i, output o, kernel k, padding p, stride s
- input skipped after the last conv at each axis: $a = \{0, ..., s - 1\}$
- dilation rate d

◦ Operation

- insert $d - 1$ spaces between two consecutive kernel elements
  ⇒ actual kernel size become $k + (k - 1)(d - 1)$
- convolve with dilated conv kernel

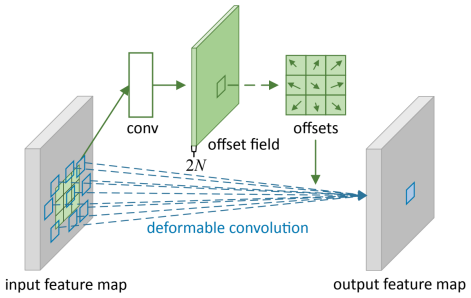
◦ Output

- $o = \lfloor \frac{i+a+2p-k-(k-1)(d-1)}{s} \rfloor + 1$

◦ Understanding

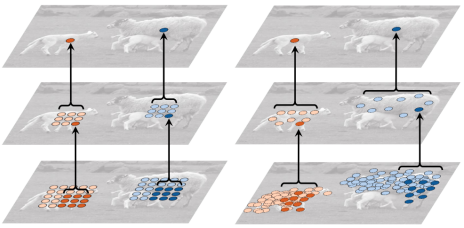
        - subsampling the kernel
          $\Rightarrow$ increase receptive field without increasing learnable params
        - whereas, deep net usually do NOT have enough receptive field for large obj on large img

- Deformable Convolution
    - Input
        - feature map
    - Setting
        - $\mathbf{w}$ the kernel of a 2D conv
        - $\mathcal{R}$ the offset of each kernel params to the conv center
          e.g. $\mathcal{R} = \{(-1,-1),(-1,0),...,(1,1))\}$ for 3x3 conv
        - $N = |\mathcal{R}|$ the number of conv kernel params
        - $\mathbf{p}_0$ the position of a conv:
            - $\mathbf{x}(\mathbf{p}_0)$ the feature at conv center on input feature map
            - $\mathbf{y}(\mathbf{p}_0)$ the resulting/corresponding feature on output map
    - Operation
        - original conv: $\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_0 \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n)$
          (single channel)
        - another same-setting conv kernel to produce offset field
          (same-size output feature map, but of $2N$ channels)
          $\Rightarrow N$ groups of x-y offset at each location
          $\Rightarrow$ each $\mathbf{y}(p_0)$ has a corresponding offset prediction $\mathbf{y}_p(\mathbf{p}_0)$
        - deformable conv: $\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_0 \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$, where $\Delta\mathbf{p}_n$ the n^{th} x-y offset in $\mathbf{y}_p(\mathbf{p}_0)$
          (still, single channel)

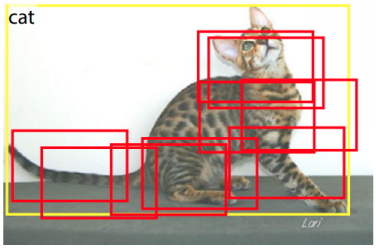


        - multi-channel input: different offset for conv on each location of each channel
        - bilinear interpolation to obtain $\mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$ as $\Delta\mathbf{p}_n$ predicted by network
    - Back Propagation
        - let $G(\mathbf{q}, \mathbf{p}) = g(q_x, p_x) \cdot g(q_y, p_y)$, with $g(a,b) = \max(0, 1-|a-b|)$
          $\Rightarrow G$ the bilinear interpolation to have $\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q})$
        - $$\frac{\partial}{\partial \Delta\mathbf{p}_n}\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \frac{\partial \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta\mathbf{p}_n}$$
          $$= \sum_{\mathbf{p}_n \in \mathcal{R}} \left[\mathbf{w}(\mathbf{p}_n) \cdot \sum_{\mathbf{q}} \frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta p_n} \mathbf{x}(\mathbf{q}) \right]$$
    - Understanding

- loosen the assumption of conv: learning the sample location for each conv
  (conv assume strict local correspondence prior)
  ⇒ more effective receptive fields
  (as bbox-size receptive fields can contain lots of background)



standard convolution deformable convolution



red box: receptive field from deformable conv
(as it is preceded by normal conv)

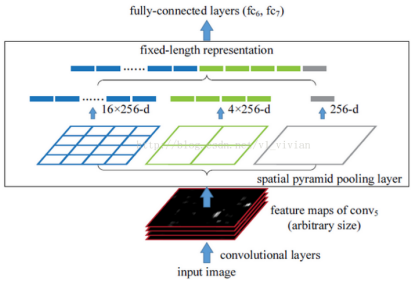
  ⇒ better sampling on non-rigid object
- attention by learning sampling location:
  - conditioned on the input ⇒ dynamically determination
  - weighting the input by sampling times ⇒ able to focus/ignore

  ⇒ each conv location has a simplified attention map (offset)
- i.e. dynamically determine the sample location for each conv at each location
  ⇒ adaptive receptive fields (based on semantic)
  - less background noise
  - can have large enough receptive fields for background / large object
    (more freedom than dilated conv)
- recognize spatial transformation / geometry variance based on semantic
  (no embedded prior & conditioned on the input)
- effective dilation are correlated with target object size
  (evidence to have learned meaningful result)

Pooling Layer

- Max/Average 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)
      ⇒ does NOT change the channel
      ⇒ 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 $o \times o$
      ⇒ $o = \lfloor \frac{i-k}{s} \rfloor + 1$ (same as conv)
  - Understanding
    - downsampling the feature maps (NO weights to learn)
      ⇒ if desired features detected anywhere, represent it by local max/average
      ⇒ summarize subregions
    - invariance to small translations of input

- Unpooling
- Spatial Pyramid Pooling (SPP)
  - Input
    - feature maps from CNN
  - Operation
    - apply on feature map a series of grids with cell number predefined
    - perform pooling on each cell, across all grids, then concat all output
      ⇒ gird defined as a proportional slicing
      ⇒ actual gird solved at runtime (w.r.t feature map size)
    - example: three grids with different cell number; each cell a max pooling

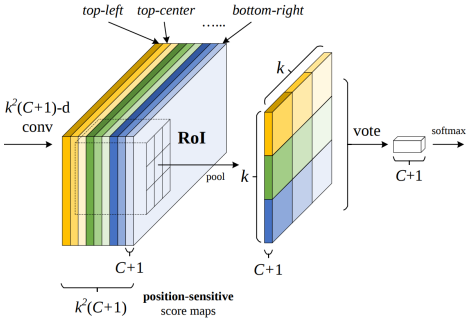


  - Output
    - a fixed size feature maps
  - Understanding
    - output shape of pooling layer pre-defined ⇒ map arbitrary input size to fixed size
      ⇒ no more crop/resize on input image
      ⇒ conv layer need only to handle normal ratio (less burden)
      (decouple conv from requirement of dense layer)
- Region of Interest Pooling (RoI Pooling)
  - Input
    - feature maps from CNN
    - RoIs: proposals (from selective search, RPN etc.) of $[x, y, w, h]$
      (translated back to be under image coordinates)
      ⇒ floating-point bbox
  - Operation
    - project RoI to feature map & quantize RoI to align with indexes
      ⇒ $x' = \lfloor \frac{x}{s} \rfloor$, where s the network stride (same applied to $[x, y, w, h]$)
    - divide each RoI with grid of desired size (proportional to the RoI size)
    - max pooling from each cell/bin

    ⇒ single-size SPP for each RoI
  - Output
    - a fixed size feature maps for each RoI
- Positional Sensitive RoI Polling (PS-RoI Pooling)
  - Input
    - RoIs (floating-point bbox)

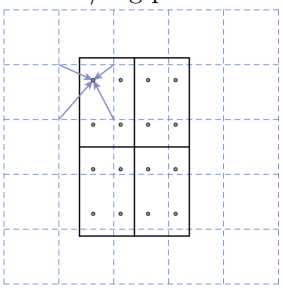
- k^2n-channel feature maps: divided into k^2 groups, each group n channel, where k^2 the bin/cell num of RoI, n related to downstream task

- Operation
  - project & quantize RoI onto feature map
  - selective (over channel) RoI pooling:
    $\Rightarrow$ for i-th bin/cell $\in k \times k$ bins/cells, select i-th group of feature maps
  - average pooling on the bin location of the selected group
- Output
  - $k \times k$ feature maps with n channels
  - n-d vector with a further average pooling
    e.g. $n = 4$ for (x, y, w, h) regression & $n = C + 1$ for C-class classification
  - example: 3×3 bins RoI for C-class classification
    (average pooling as voting to produce final classification pred)

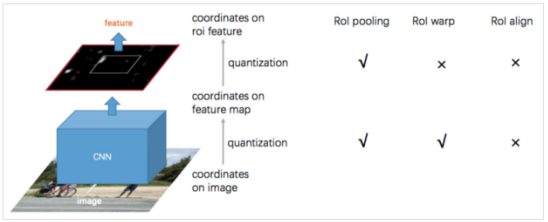


- Understanding
  - position sensitive: dedicated conv kernels & feature maps for each bin location
    (each feature map group accounts for a spatial location of RoI bin)
  - no need of post-RoI process: share all computation
    $\Rightarrow$ produces prediction from each bins before RoI
    (parallel proposal generation & prediction in R-FCN)
  - $\Rightarrow$ more robust prediction
    (as $k \times k$ prediction made for each proposals)

- RoI Align
  - Input
    - feature map & RoI (floating-point bbox)
  - Operation
    - project the RoI to feature map without quantization
      i.e. $x' = \frac{x}{s}$, where s the network stride $\Rightarrow$ still floating-point number
    - calculate all the desired sampling point for the RoI
    - bilinear interpolate to obtain the feature at each point
      (using the 4 closest features)
    - $\Rightarrow$ fetch out the RoI feature & max/avg pool at each bin (as usual)



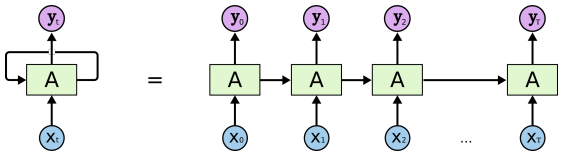
- Output
  - fixed-size feature
- Understanding
  - no quantization
    ⇒ remove mis-alignment between RoI location & extracted feature
    ⇒ preserve exact localization (i.e. per-pixel spatial correspondence)
  - RoI pooling v.s. align



- Probabilistic Max Pooling

RNN Layer

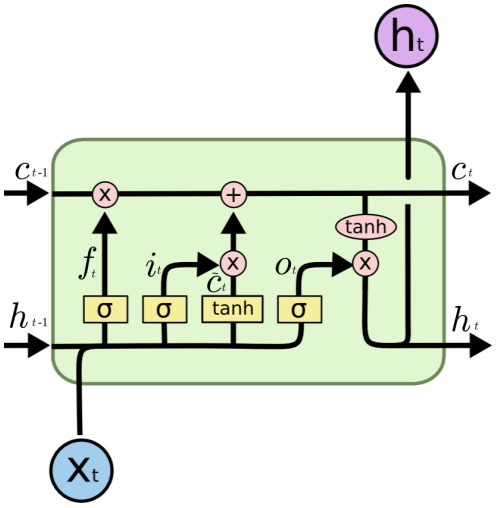
- Overview
  - Input Data
    - sequence data: includes time/precedence (conventionally arrived from left to right)
    - for each time step, data can be vector, feature maps, etc...
  - RNN Cell
    - consume the input of current time step & the hidden state from last time step (hidden state usually 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
      ⇒ 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)
        - hard to converge, due to fluctuating gradient in an unroll (noisy intermediate stage)
          $\Rightarrow$ initial short sequence to overcome plateaus, then long sequence for dependency
    - 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 $h^t = G_o \cdot \tanh(c^t)$

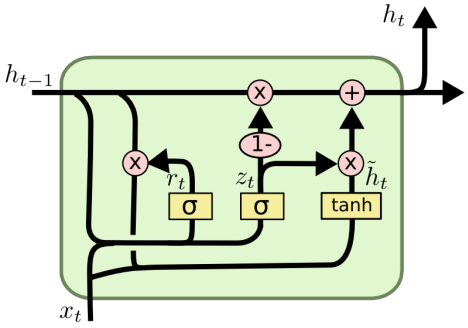


where,
joint arrows denotes a concatenation
$c_{t-1}, \widetilde{c}_t, c_t$ the previous/candidate/current memory;
h_{t-1}, h_t the previous/current hidden state;
f_t, i_t, o_t the forget/input/output gate

- 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)
    - $h^t = c^t$

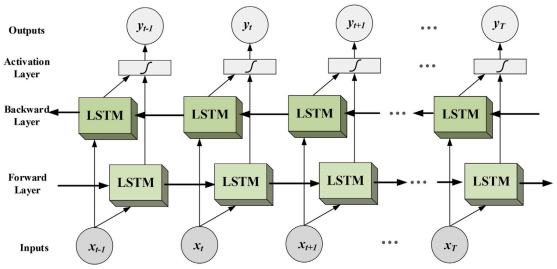


where,
joint arrows denotes a concatenation
$h_{t-1}, \widetilde{h}_t, h_t$ the previous/candidate/current memory;
r_t, z_t the relevance/update gate

  - 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/Bi-RNN)
    - Structure
        - one RNN layer scanning as $t = 1 \rightarrow T$, hidden state exposed as h_1^t
        - another RNN layer scanning from $t = T \rightarrow 1$, 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.



(using LSTM cell)

- Convolutional LSTM (ConvLSTM)

Attention

- Motivation
    - Long-Term Dependency Modeling
        - consider all input at each step
          $\Rightarrow$ reduce maximal num of ops before 2 positions interact to $\mathcal{O}(1)$
        - v.s. in rnn/conv/dialated conv, ops between positions grow $\mathcal{O}(n)/\mathcal{O}(\frac{n}{k})/\mathcal{O}(\log_k^n)$,
          where n the distance between positions, k the conv kernel size
    - Interpretable Model
        - can visualize & interpret how the network models various relations
          (via attention distributions)
- Encoder-Decoder Attention
    - Motivation
        - model relationship between input-output & overcome long-term dependency
          (e.g. 1st input position - last output position)
        - focus on different attributes/features based on various requirement
    - Input
        - sequence of input vector

        - current output location & previous context (attention output)
    - Operation
        - a function over all spatial location: $[f_i, \mathbf{r}] \rightarrow \text{logits}_i$,
          where f_i the feature at location i, $\mathbf{r}$ task specific requirement
        - construct a distribution over all location: logits $\xrightarrow{\text{softmax}} \alpha$,
          where α the attention over all location i
    - Output
        - a probability distribution over all location of the input
        - $\Rightarrow$ weighted-sum on all input vector to obtain a context vector
          (which is specifically designed for current output location)
    - Practice in Seq-to-Seq RNN
        - a bi-RNN encode each input as hidden state: $h^1, ..., h^{T_x}$
        - context from attention at time t: $c^t = \sum_{i=1}^{T_x} \alpha_i^t h^i = [h^1, ..., h^{T_x}]\alpha^t$,
          where attention α^t being a column vector
          $\Rightarrow$ a weighted sum over all hidden states as context
        - a small (1-layer) net mapping $[c^{t-1}, h^i] \xrightarrow{\text{dense}} \text{logits}_i \xrightarrow{\text{softmax}} \alpha_i^t$
          $\Rightarrow$ attention on h^i depends on previous global context c^{t-1} & h^i itself
          (softmax to ensure $\sum_{i=1}^{T_x} \alpha_i^t = 1$)
        - decoder RNN takes c^t as input, until stop sign generated
          (as c^t can be calculated infinite times)
        - or, decoder RNN process as usual, but concat $[c^t, h^t]$ before output y^t
          (still takes its previous output y^{t-1} as input)
    - Understanding
        - **essence**: re-weight certain features in the network
        - at each step, amplify relevant input hidden state (state with more scores)
          $\Rightarrow$ capture the token-level relationship between input-output sequence
        - use the whole sequence of hidden states as sequence encoding
          (instead of only last hidden state as context, which can be bias & have info lost)
          $\Rightarrow$ at each step, provide both sequence- & word-level info
          (aggregate all location of input seq for decoder rnn at each step)
          $\Rightarrow$ every output location can directly interact with input seq $\Rightarrow$ overcome RNN encoder-decoder failure in longterm dependency
        - quadratic cost: attention α a $T_x \times T_y$ matrix to be calculated

- Self Attention
    - Motivation
        - model not only sentence-sentence relation, but also relation within the sentence
          $\Rightarrow$ each word recognizes all other words in its sentence
    - Input
        - sequence of a vector & current location
    - Operation
        - with matrix for Query W^Q, Key W^K and Value W^V & a word embedding e
          $\Rightarrow$ generate query $q = W^Q e$, key $k = W^K e$ and value $v = W^V e$, where
          W^Q, W^K, W^V to be optimized & e a col vector

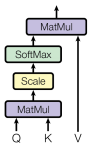
- given current word t, for each word i, obtain score (scalar) $s_t = (q^t)^T \cdot k^i$
  i.e. use its "query" to multiply the "key" of other word in the sequence
- normalize (standardize) by the square root of the length of the "key" vector
  $\Rightarrow s'_i = \frac{1}{\sqrt{d_k}} s_i$, where d_k the dimension of "key"
  (to obtain a more stable gradient)
- softmax to obtain attention α^t as a col vector
- "self" context $c^t = \sum_{i=1}^{T} \alpha_i^t v_i = [v^1, ..., v^T]\alpha^t$
  i.e. weighted-sum over "value" vectors

◦ Output

- a distribution over "value" vector of each word in the current sequence
  $\Rightarrow$ to obtain an aggregation (weighted-sum) of "self" sequence

◦ Speed-up

- use matrix calculation: $E = [e^1, ..., e^T]$, with each e a col vector
  $\Rightarrow Q = W^Q E, K = W^K E, V = W^V E$, with each q, k, v as col vector
- $\Rightarrow A = \text{softmax}\left(\frac{1}{\sqrt{d_k}} Q^T K\right)$,
  where A_i the attention for i^{th} word (as a row vector)
- $\Rightarrow C = VA^T$, where each context vector as a col vector



◦ Understanding

- in seq2seq: capture the relationship inside input/output seq
  e.g. let model associate "it" with "animal" in sentence:
  "The animal didn't cross the street because it was too tired"
- normalization to obtain more stable gradient:
  e.g. with q, k vector of random variables with mean 0 & variance 1
  $\Rightarrow q \cdot k = \sum_{i=0}^{d_k} q_i k_i$ has mean 0 variance d_k
  $\Rightarrow$ actually, standardization within attention layer
- contextualized word-embedding / sequence representation
  $\Rightarrow$ determine word embedding / representation with sequence context considered
- could be dense/conv layers, instead of matrices
  $\Rightarrow$ a layer / network branch shared across all location
  (for 2D input: map $m \times n$ input feature map to $(m \times n)^2$ attention map...)
  (if directly apply self attention)

• Multi-headed Self Attention

◦ Motivation

- provide multi-modal attention to focus on various location

◦ Input

- same as self attention

◦ Operation

- the "batch-mode" of self attention to get N separate results (context matrices)
- concat & further project into a single context matrix $C = [C_0, ...C_{N-1}] \cdot W^O$

◦ Output

- same as self attention
  $\Rightarrow$ transient to layers before & after

◦ Understanding

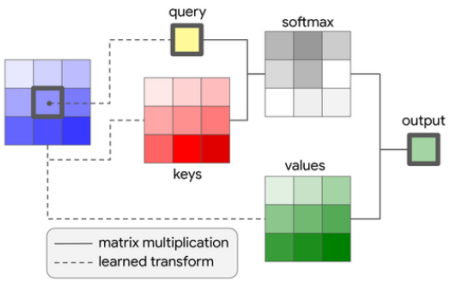
- avoid attention to focus on the current location itself
  $\Rightarrow$ able to focus on multiple position

        - multiple sets of W^Q, W^K, W^V to optimize
          $\Rightarrow$ multiple representation subspace to project the input into

- Visual/Spatial Attention
    - Motivation
        - focus on correct spatial context when generating corresponding caption/sequence
    - Operation
        - given feature map $\mathbf{a} = \mathbf{a}_1, ...\mathbf{a}_L$
          $\Rightarrow$ attention $\alpha = \text{softmax}[\phi(\mathbf{a}, h)]$, where h other info to guide the attention
        - e.g. use high-level semantic feature to generate attention
          $\Rightarrow$ ensure enough receptive fields to accounts for global context
          (may need to upsample attended low-level feature to fit spatial dimension)
        - e.g. use previous RNN hidden state to generate attention
          $\Rightarrow$ ensure sequential conditioning
          (as in "show attend and tell")
    - Understanding
        - enable model to better capture low-level info
          (from different location & on demand)

- Channel-Wise Self Attention in Conv
    - Motivation
        - better examine inter-dependency between channels
          (with the help of current global context info)
    - Input
        - feature maps
    - Operation
        - squeeze: spatially aggregate global feature to summarize each channel
          e.g. channel-wise average pooling
          $[H, W, C] \rightarrow [1, 1, C]$
        - excite: perform inter-channel interaction to get attention α
          e.g. dense (1x1 conv) layer(s)
          $[1, 1, C] \rightarrow [1, 1, C]$
        - scale: channel-wisely multiply α back to input feature map X
    - Output
        - re-weighted feature maps
    - Understanding
        - break the limited receptive fields of conv ops
          $\Rightarrow$ include global information to augment conv net
    - Analysis
        - lower layers tend to have more uniform attention
          $\Rightarrow$ correlates with "lower layers learn more general feature"
        - high layers tend to have more attention at some specific channels
          $\Rightarrow$ correlates with "higher layers are more specific & discriminative"
        - attention becomes almost "all 1s" at last stage
          $\Rightarrow$ already contains global feature at last/latter stage

- Stand-Along Self Attention as Conv
  - Motivation
    - provide alternatives to conv ops
      Stand-Alone Self-Attention in Vision Models (https://arxiv.org/abs/1906.05909)
  - Operation



    - stand-along attention
  - Understanding
    - less FLOPS & less params than standard dense conv
- Attention in Attention

12.3.5 Blocks

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
  -

Residual Block

- Basic 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 $l+2 \Rightarrow$ two 3x3 conv (with batch-norm, relu)
  - Skip/Passthrough Connection (Shourtcut)
    - $s(x^l)$: the passing of x^l, from layer l directly to the layer $l+2$
    - different spatial size /channel in $l, l+2$: adjust x^l by
      - padding: pad to the size in layer $l+2$ (pad a volume)
      - spatial stack: merge local channels into single channel by stacking (e.g. $26 \times 26 \times 512 \to 13 \times 13 \times 2048$)
      - stride-1 1x1 conv to adjust channel, stride-2 1x1 conv to also downsample (used in original paper)
- Bottleneck Structure
  - Main Path
    - 1x1 conv to shrink the channel
    - 3x3 conv to process
    - 1x1 conv to expand the channel
- Understanding
  - Bottleneck
    - to reduce parameters in conv kernel (used in deep net, e.g. Resnet-50, -101)
  - 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)
  - Ensemble
    - contains much more shallow paths than real deep paths
      $\Rightarrow$ ensemble those shallow paths at various level
  - Easier Optimization
    - provide a prior function $y = x$, which is closer to the target function to learn (compared with $y = 0$, as params rand init to be closed to 0)
      $\Rightarrow$ learning by reference is easier
      (shown by smaller response & params closer to 0 & faster convergence)
    - progressive learning by design: each block learn a modification
      $\Rightarrow$ stacked blocks progressively fit the target function
      $\Rightarrow$ guaranteed baseline
    - makes optimization for deep (≥ 20) net possible & unleash deep power

12.3.6 Training

- Layer-Wise Fine-Tuning
  - Overview
    - freeze the low-level layers of pretrained net & only train the output part
    - gradually (e.g. layer-by-layer, block-by-block) unfreeze lower layers
  - Understanding
    - low-level layer capture generic & task-independent info
    - high-level layer capture task-specific & semantic info
      (e.g. conv net)
    - high-level can settle down only after its input are more consistent
    - low-level can get effective update only if its gradient are more consistent
    - $\Rightarrow$ train high-level first (more directly beneficial)
      (then jointly optimization)
- Multi-Tasking Training
  - Overview
    - optimization goal composes of losses from multiple tasks
      (usually a sum over all losses)
  - Understanding
  - joint optimization find the shared representation suitable for all tasks
    (e.g. classification representation NOT suitable for localization)
  - could prevent net from using false evidence by constraints from other tasks
    $\Rightarrow$ thus may improve per-class performance
  - able to design loss more aligned with final evaluation matrices
    (though, really depends on loss design)

12.4 Architectures

12.4.1 Convolutional Networks

Classic Backbone

- 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
          ⇒ conv layers, though large in number, contains far less weights
        - conv stride $s = 1$ (compare to $s > 1$)
          ⇒ decouple downsampling into the pooling
          ⇒ 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)
        - introduce clean & neat philosophy
          ⇒ if not downsampling: keep to have same channel num
          ⇒ at each downsample stage: halve the spatial size & double the channel number
          (the complexity of each layer is preserved)
          (till small or deep enough e.g. 7×7 size, 512 channels)

- Inception Net
    - Building Blocks
        - inception block
    - Structure
        - stack of inception blocks
        - auxiliary loss from hidden layers ⇒ to avoid gradient vanishing

- **ResNets**
    - Building Blocks
        - residual block (basic/bottleneck)
        - batchnorm applied in the conv(s) inside residual block
          (enormous batchnorm to address gradient exploding&vanishing)
    - Structure
        - stack of residual block, for hundreds, even thousands of layers
        - average pooling (channel-wise) at the last layer

        - classification: softmax classifier
        - detection: regressor
          ... for other tasks
    - Training
        - warm-up the training with small learning rate till net starts to converge
          (avoid randomness at the beginning)
          ⇒ explore the right optimization direction first
    - Understanding
        - notice directly building deep-net cause degradation:
            - high train error & high test error ⇒ no over-fitting
            - batch-norm ⇒ no gradient vanishing/exploding

          ⇒ current optimizer (SGD) fails to optimize deep net
        - provide a prior for optimizer: start with a identity mapping
          ⇒ $y = x$ are closer to target function than $y = x$
          (easier to push params to 0, than fit a identity mapping from scratch)
          (if no modification needed)
          ⇒ learning with reference
        - ⇒ progressively fitting target function by design
          ⇒ deep net can at least be optimized based on the result of shallow net
          (each residual block gradually modify the signal to the desired one)
        - ensemble nets with shallow & medium depth ⇒ essentially a wide net
          (as the real deep net takes only few paths)
          ⇒ avoid the training of real deep net (instead of solving it)
        - compared with inception net: less choice for network (easier for optimizer)
          ⇒ less params and complexity, but deeper & better performance (depth wins!)
        - res-net makes SGD optimization of deep net (≥ 20) possible
          (plain net with depth ≥ 20 NOT converges even if more time/epochs given)
        - resnet has lower response & params closer to 0 & converges faster
          (compared to non-residual plain net)
          ⇒ empirically prove resnet is easier for optimization

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

Interpretation and Visualization

- Activation Testing
- Procedure

- use a fixed set of bounding box proposals (with different images)
- treat a single hidden unit in network as a detector/classifier
  $\Rightarrow$ use its activation as scoring
- for detection: after non-max suppression, rank the boxes from highest to lowest activation
  $\Rightarrow$ test out which visual mode the feature is sensing
- Understanding on Network
  - each feature sensing a different object
    e.g. face, red blob, square objects, light reflectance, array of dots, text (like OCR), etc.
  - able to learn: class-specific shape & abstract attributes
    $\Rightarrow$ construct a distributed representation of object
    (e.g. shape, color, texture, text, material properties, etc.)
  - $\Rightarrow$ later dense layer can model a large range of object
    (as combining different distributed object attributes)

- Layer Ablation
  - Procedure
    - after trained, reduce the num of dense layer, until fully conv
      i.e. using previous layer output as prediction at test time
    - on test set: evaluate performance drop
      on other dataset: evaluate generalization ability drop
  - Understanding on Network
    - CNN ability relies on convolution layer, instead of dense layer
      $\Rightarrow$ conv layer as feature map extractor for other domain-model
    - fine-tuning improves a lot $\Rightarrow$ fine-tunning the dense layer
      (while conv layer are learned to be quite generic)

12.4.2 Recurrent Neural Network

Classic RNN

- Overview
  - Stacked RNN Layers
    - the output of layer l becomes the input for layer $l+1$
      $\Rightarrow$ 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

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 $\rightarrow$ image: outline refinement ...
        - language $\rightarrow$ language: sentence style capturing
    - Implementation
        - concatenation

Extension

- Siamese Network
    - Input
        - image pair (p_1, p_2)
    - Structure
        - CNN as feature encoder
          $\Rightarrow$ same CNN process each image, yielding (f_1, f_2) as extracted feature
        - jointly process concat $[f_1, f_2]$
    - Variants
        - joint process from the beginning
          $\Rightarrow$ concat $[p_1, p_2]$, then CNN till the end
          $\Rightarrow$ generally **better performance**, especially when detailed structure are vital
- Multiple Encoder (Pseudo Siamese Network)
    - Understanding
        - project different information into the same space
          (jointly process those information after feature projection/extraction)
- Multiple Decoder
    - Functionality
        - impose multiple requirements to the encoder (via auxiliary loss)
          $\Rightarrow$ multi-task training (multiple output branch)

12.4.4 Generative Network

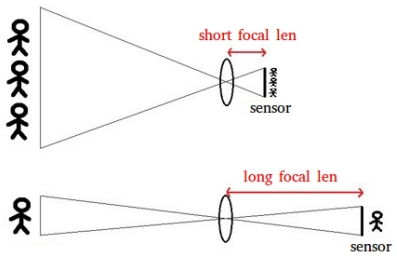
Generative Adversarial Network

12.5 Computer Vision

12.5.1 Visual Sensor

Camera

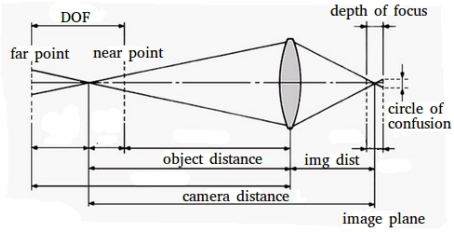
- Basic Image-forming Theory
  - Camera Coordinate
    - image axises: same as axises for matrix index
    - x, y align with the image axises, z align with focal line
  - Resolution
    - the minimum feature size of the object under inspection
      $\Rightarrow$ finally, represented as the pixel in the image
      (hence, jointly controlled by pixel size, lens, object distance and sensor size)
    - pixel count: total number of pixels in the row/col of image
    - pixel density: dots/pixels oer inch (dpi)
    - sensor resolution: pixel count on sensor and their physical size and spreading
    - lens resolution: the impact of optics (diffraction, etc) on light info towards sensor
  - ISO: Light Sensitivity
    - used to be the light sensitivity of the film
    - $\Rightarrow$ in digital camera: the amplification/attenuation for raw sensor value
      $\Rightarrow$ can add noise if amplified too much, due to dark current
    - base ISO:
  - Exposure Time
    - control incoming light (denoted by shutter speed)
  - Focal Length F
    - the distance from last lens central point to focal point, determined by lens physics
    - given a fixed object distance, larger F, larger formed object
      (object distance need to be larger than focal length)



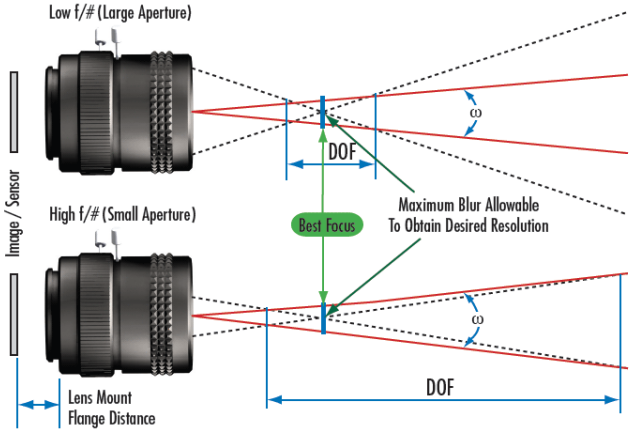
  - Aperture
    - diameter of entrance pupil D: control the size of aperture
      $\Rightarrow$ aperture stop: aperture setting to restrict input pupil size and hence brightness
    - larger size (D), more incoming light
  - F-Number $N = \frac{F}{D}$
    - reveal the signal to noise ratio, contrast, image brightness

- larger N, smaller D, less incoming light, larger DOF, larger depth of focus

- Depth of Field (DOF) & Depth of Focus
  - definition: the range on z axis where resolution can be maintained
    before the lens: depth of field (in the field)
    after the lens: depth of focus (inside camera)

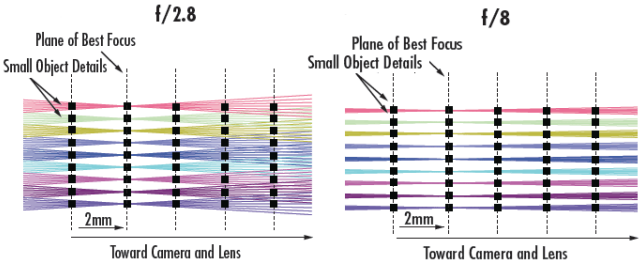


- Depth of Field
  - DOF as the range where 1 pixel can hold all info from 1 position



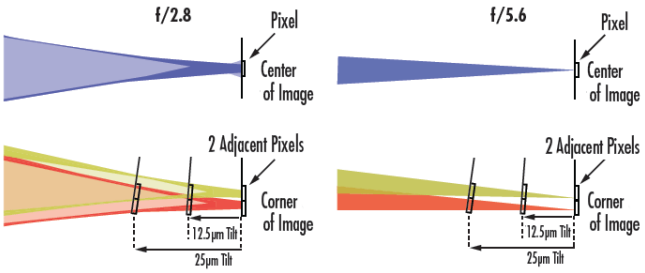
where, dashed line the incoming light;
red line the system resolution (the pixel size, changing with object dist)
⇒ DOF defined by intersection of two type of lines
⇒ with large enough DOF, resolution can be bad at the end of DOF
(pixel size too large)

  - resolution with DOF



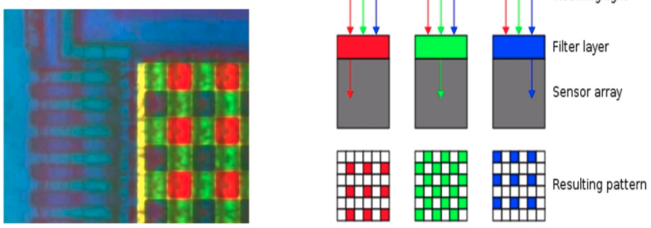
where black boxes a series of pixels, beams in colors as light info from an object
⇒ the placement of box on beams shows the information contained in a pixel
⇒ larger DOF, large depth range for acceptable resolution
(as pixel contain more separable info)
⇒ out of DOF: pixel can not distinguish beams from diff location

- Depth of Focus
  - depth of focus as the range where 1 pixel can hold all info from 1 position
  - resulted from tilted sensor ⇒ may not in the ideal position to receive light info



where right-most line as ideal focal plane, left lines as sensor tilted by $12.5/25\mu m$
$\Rightarrow$ tilt not influence pixel in the center, but the pixel in the corner
$\Rightarrow$ out of depth of focus: pixel can not distinguish beams from diff location

- Camera System
  - Optics
    - lens: determine focal length, control f-number, thus DOF
    - filter: filter out the influence of infer-red
  - Image Sensor
    - tilted sensor for optic-electric conversion
    - determine the pixel-size, thus influence resolution
    - dynamic range: range of full electric capacity - smallest signal-to-noise ratio
  - Color Filter Array
    - a filter for each sensor to allow only light of specified color to reach sensor
    - interpolation to recover color info for each pixel



  - ISP
    - optimize the raw sensor data to account for:, e.g.
      white balance, motion compensation, tone mapping, denoising, etc.
  - Serdes
    - serialized output the displayed image
- Noise
  - Rolling Shutter
    - due to image sensor scanning through across the scene
      (instead of snapshot the whole scene)
      $\Rightarrow$ speed of shutter + object movement speed: skewed scene
    - scanning scene takes time & each time step only scanning a roll of sensor
    - trade-off: able to continue gather photons, thus more sensitive
      (v.s. global shutter)

Lidar

- Time of Flight (ToF) System
  - Light Source
    - wavelength: 905nm vs. 1550nm
    - laser peak power: need to avoid eye damage
    - repetition rate: positively related to fps and points intensity
    - pulse width
    - beam quality
    - background noise: more noise for 905nm than 1550nm
  - Scanning
    - spinning: emit single beam & scan emit angle across 3D space
      $\Rightarrow$ do need motion compensation if sensing when moving
    - optical phase array: use electricity to control beam angle (solid state streamer)
    - flash light: emit laser like flashlight (consume more power)
  - Receiver
    - sensitivity: determine effective detect distance
    - saturability: determine noise and dynamic range
  - A/D Converter
    - convert received laser to electricity signal
    - response curve: usually non-linear
-

Radar

- Radar System
  - Waves
    - range resolution inversely $\propto$ bandwidth
    - frequency $\in (76, 81)$ Ghz
  - Pulse Radar
    - transmit antenna G_t send pulse signal
    - receiver antenna A_t receive signal
    - measure range by ToF
  - FMCW Radar
    - send frequency modulated continuous-wave signal, instead of pulse
      $\Rightarrow$ measure speed (vector) via observing Doppler frequency shift
- Understanding
  - Ability
    - work under extreme condition (-40 to 80°, rain, dust, etc.)
  - Features
    - sensitive to metal
    - multiple radar interference: receive signal sent by others
    - multiple reflectance: waves bounce between parallel metal objects

12.5.2 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
      $\Rightarrow$ used in pose detection, bounding box detection, etc.
- Evaluation Metrics
  - Recall-to-IoU
    - the plot of recall at each IoU threshold, similar to average precision curve
    - should use only for diagnosis, instead of performance evaluation
      (as only loosely related to accuracy, etc.)
- Understanding
  - multi-object localization & classification
  - Two-Stage Detection
    - 1^{st} stage: objectness classification (filter out simple neg)
      (trained with imbalanced data)
    - 2^{nd} stage: fine k-class classification & location refinement
      (trained with more balanced data)

Common Processing

- 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
      $\Rightarrow$ 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
      $\Rightarrow$ remove (suppress) bounding boxes with large overlap
    - repeat until all bounding boxes have a low IoU with each other
      $\Rightarrow$ remaining boxes are the final predictions

- Understanding
  - usually not harm performance (as preserving high-score pred)
  - help reduce false positive

- Selective Search
  - Input
    - image
  - Goal
    - class independent regions to distinguish regions of different objects
      $\Rightarrow$ object segmentation: potential region containing object (with classification waived)
  - Hierarchical Grouping
    - initialize the image with a region proposals method
    - compute similarities for each adjacent region pairs
    - group the most similar region pair into one region
      (similarity include: size, color in color space, sift, etc.)
    - iterate through the process, until whole image grouped into a single region
    - diverse settings: color space, similarity measurement, region initialization method
  - Region Scoring
    - the earliest merged (grouped) images get the highest score in current run
      e.g. last merge (whole image) scored 1, 2^{nd} merge scored 2
    - propose region with diverse settings & accumulate score for the same region
    - select regions in first n^{th} high score
  - Understand
    -

- Localization Loss Normalization
  - Input
    - proposal box $b = \{b_x, b_y, b_w, b_h\}$
    - pred box $p = \{p_x, p_y, p_w, p_h\}$
      $\Rightarrow$ network output $\{\frac{p_x - b_x}{b_w}, \frac{p_x - b_y}{b_h}, \log \frac{p_x}{b_w}, \log \frac{p_x}{b_h}\}$
    - ground truth box $g = \{g_x, g_y, g_w, g_h\}$
      $\Rightarrow$ label $l = \{\frac{g_x - b_x}{b_w}, \frac{g_y - b_y}{b_h}, \log \frac{g_w}{b_w}, \log \frac{g_h}{b_h}\}$
      (under classic encoding: predict the adjustment)
    - $\Rightarrow$ diff $\Delta = \{\delta_x, \delta_y, \delta_w, \delta_h\}$,
      where $\delta_x = \frac{g_x - p_x}{b_w}, \delta_y = \frac{g_y - p_y}{b_h}, \delta_w = \log \frac{g_w}{b_w}, \delta_h = \log \frac{g_h}{b_h}$
  - Goal
    - avoid too small Δ localization loss (compared to classification loss)
      $\Rightarrow$ normalization to avoid small numerics
  - Operation
    - normalize as $\forall \delta \in \Delta, \delta := \frac{\delta - \mu}{\sigma}$, where μ the mean; σ the standard deviation
  - Understand
    - ensure balanced loss $\Rightarrow$ more effective multi-tasking

Two-Stage Detection

- 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
- R-CNN: Regions with CNN Features
  - Bounding Box Encoding
    - tightly crop/wrap over regions from region proposal methods
    - regressor output refinement of x, y, w, h
  - Inference
    - selective search ⇒$\sim$ 2000 regions potentially with object
    - each region edged with a tight bounding box, resized according to CNN
    - CNN to extract feature for each bounding box, into 4096-length vector
    - class-specific SVMs (one for each class) to classify each box (feature vec)
    - per-class linear regressor to refine the proposed box
      (better localization mAP 4% ↑)
  - Training CNN
    - pre-train on classification
    - fine-tine on detection: classify resized proposed box into $N + 1$ classes
      (+1 for non-objectness)
    - box cropping: the tightest box + some space for context
    - 32 box with object & 96 box without object ⇒ bias to positive example
    - neg example: box proposal with $\text{IoU}^{\text{region proposal}}_{\text{label box}} < 0.5$
    - pos example: box proposal with $\text{IoU}^{\text{region proposal}}_{\text{label box}} > 0.5$
      (assigned to the label box with max IoU if multiple matched)
    - rely on val set, with train set as auxiliary source for pos example
      ⇒ regenerate more class-balanced data split
      (avoid annotation noise in train set)
  - Training SVM
    - input 4096-length vector, output binary classification for a class
    - neg example: box proposal with $\text{IoU}^{\text{region proposal}}_{\text{label box}} < 0.3$

        - pos example: box proposal with $\text{IoU}^{\text{region proposal}}_{\text{label box}} > 0.5$
        - box in-between are ignored
        - all label box as pos example, hard neg mining on 5k images
          $\Rightarrow$ control data volume, speed up SVM inference
    - Training Linear Regressor
        - input the feature at CNN logits layer, regress location offset
        - regress x, y directly, w, h in the log space
        - only regress box nearby: $\text{IoU}^{\text{region proposal}}_{\text{label box}} > 0.6$
          (assigned to label box with max IoU if multiple matched)
        - heavily regularized (as only performing nearby regression)
    - Extension in Segmentation
        - region proposal switched to that for segmentation
        - wrapped bounding box on proposed region
        - get conv features with & without background in box removed (set to 0)
        - output classification for the region
    - Understanding
        - first to bridge the gap between classification & detection
        - first to show supervised pre-training on auxiliary dataset is effective
          $\Rightarrow$ multi-tasking is good
        - important to distinguish pos-neg examples
          $\Rightarrow mAP5\% \uparrow$ when thr= 0.3, (v.s. thr= 0.5)
          $\Rightarrow$ pos-neg balance important
        - context in box matters
        - by ablation study reveals: CNN ability comes from conv layer
          $\Rightarrow$ CNN as feature map extractor for other domain-specific algorithm
        - by visualization reveals: CNN is able to learn distributed representation
          $\Rightarrow$ recognize abstract attributes e.g. shape, texture, color, material etc.
        - speed legged by region proposal
        - performance legged by localization (instead of classification, thanks to CNN)
        - jittered example: box proposal with $\text{IoU}^{\text{region proposal}}_{\text{label box}} \in (0.5, 1)$ but is not ground truth
          $\Rightarrow$ need more careful fine-tunning to avoid SVM (which is used for hard negative mining)
          $\Rightarrow$ problems arise from NOT being end-to-end trained

- Fast R-CNN
    - Bounding Box Encoding
        - similar to SPP & R-CNN: predict adjustment/refinement for each proposal
          (focusing on the 2nd stage)
        - classifier for all $k + 1$ class & each k class has its own location regressor
          (1 for the background class)
    - Inference
        - input: an image & a list of region of interest (RoI) in the image
          (RoI pre-calculated from proposals)
        - one-pass inference: shared computation for proposals from backbone CNN
          (similar to SPP-net & improved over R-CNN)

- RoI pooling for each projected proposal
  (a special SPP with only one pyramid level)
  ⇒ extract fix-size feature from each proposal
- further 2 dense layers generate RoI feature vector
- fed into classification & localization branches (each with 1 dense layer)
- perform per-class NMS on refined proposals to generate final bbox output
  ⇒ all bboxes are included in NMS (as independent NMS for each class)

◦ Structure

- VGG to extract feature of image
- RoI pooling to extract proposal (RoI) region into fixed-len feature
- discard SVM: no need to cache tons of feature (support vector) on disk
- post-RoI dense layers could be implemented as 1-D conv over RoI feature vectors

◦ Training

- in R-CNN/SPP: uniformly sample RoI, each RoI has an associated image
  ⇒ NOT able to share computation if each RoI is from a different image
- discard the concern of
  "RoI from one image are correlated and can cause slow convergence"
  (no harm ... as each example treated independently anyway)
- enforced computation sharing:
  sample N images, then sample R/N RoIs for each image
  (R the size of batch: each RoI considered as an example)
  ⇒ much faster training as computation shared over R/N gradient updates
  (in paper, $N = 2, R = 128$)
- multi-task loss: $L(p, u, t^u, v) = L_{\text{cls}}(p, t^u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$,

  where p/u the pred / label for classification (0 being background class)
  t^u/v the pred / label bbox for true class u

- classification loss $L_{\text{cls}}(p, u) = -\log p_u$ the cross-entropy
- localization loss: $L_{\text{loc}}(t^u, v) = \sum_{i \in \{x,y,\log w,\log h\}} \text{smooth}_{L_1}(t^u_i - v_i)$,
  where: $\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & |x| < 1, \\ |x| - 0.5 & \text{otherwise.} \end{cases}$
- normalize location label v to be 0-mean & unit variance
  ⇒ help setting $\lambda = 1$
- $IoU^{\text{proposal}}_{\text{labe}} > 0.5$ to be positive example, $0.1 < IoU^{\text{proposal}}_{\text{label}} < 0.5$ to be negative
  (some data discarded due to GPU limit; pos example ensured to be $< 25\%$)
- joint end-to-end training

◦ Analysis: v.s. Stage-Wise Training

- first train classification & then train location branch with other params frozen
  ⇒ worse than joint multi-task training
- joint optimization find the shared representation suitable for both tasks
  (classification representation NOT suitable for localization)
- could prevent net from using false evidence by constraints from other tasks
  ⇒ thus may improve per-class performance

◦ Analysis: Scale Invariance

- conv net able to achieve

◦ Analysis: Proposal Density

- Fast R-CNN cannot handle dense proposals

        - average precision fall & average recall not able to review true ability of net
          (as recall due to more input, rather than ability of net)
    - Understanding
        - better performance from multi-tasking & end-to-end joint optimization
          (by exploring a more robust shared representation & no SVM)
        - faster training: sharing computation in batch & no SVM
        - faster inference: sharing convolution across proposals
          (together with, network compression by SVD on weight matrix of dense layer)
        - smooth L_1 is less sensitive to outliers than L_2
          $\Rightarrow$ thus less sensitive to learning rate tunning
          (as L_2 needs to avoid gradients exploding under unbounded location pred)
        - per-class regressor: big output volume & require enough data for all classes
          $\Rightarrow$ hard to scale to detection with thousands of classes

- RPN: Faster R-CNN
    - Bounding Box Encoding
        - anchor box $[x_a, y_a, w_a, h_a]$: predefined boxes of various {scale} $\times$ {ratio}
          (a grid of reference boxes - k anchors at each location on output feature map)
        - proposal: similar to Fast R-CNN, but adjustment predicted w.r.t anchors
          $\Rightarrow t_x = \frac{x-x_a}{w_a}, t_y = \frac{y-y_a}{h_a}; t_w = \log(\frac{w}{w_a}), t_h = \log(\frac{h}{h_a})$
          $\Rightarrow$ used as reference box for 2nd-stage detector
        - Fast R-CNN as detector & its box encoding used at final pred
    - Inference
        - RPN predicts proposals based on anchor box, with only objectness classification
          (binary fore- vs. back-ground classification)
          $\Rightarrow$ channel-size $= 2k$ (k if using sigmoid) for cls; $4k$ for reg
        - non-maximal suppression (NMS) to reduce proposals
        - project & quantize floating-point $[x, y, w, h]$ box onto the feature map
          ($[x, y, w, h]$ box is under original image coord, after translated by anchor box)
          $\Rightarrow x' = \lfloor \frac{x}{s} \rfloor$, where s the network stride
        - detector (Fast R-CNN) predicts final boxes & multi-class classification
          (with class-specific NMS)
    - Structure
        - backbone CNN (VGG, Resnet-50, etc.) extract feature map
        - RPN branch = 3x3 conv + 2 sibling 1x1 conv (for cls & reg)
        - RoI pooling on the extracted feature map, according to each adopted proposal
          (followed by rest of the Fast R-CNN)
          $\Rightarrow$ proposal generator & detector shares the same backbone
    - Training RPN
        - RPN loss $L(\{p_i\}\{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{N_{\text{reg}}} p_i^* L_{\text{reg}}(t_i, t_i^*)$,
          where $*$ denotes label, p_i^*/t_i^* the cls/reg label for i-th anchor
        - L_{cls} the log loss on objectness of proposal (binary classification)
          $\Rightarrow p_i^* = \begin{cases} 1 & \text{IoU}_{\text{label}}^{\text{anchor}} > 0.7 \text{ or anchor overlap with a label with highest IoU,} \\ 0 & \text{IoU}_{\text{label}}^{\text{anchor}} < 0.3 \end{cases}$
            · ignore confusing anchors with $\text{IoU}_{\text{label}}^{\text{anchor}} \in [0.3, 0.7]$ (not included in loss)
            · ensure at least 1 (can have multiple) anchor for each label
        - L_{reg} the smooth L_1 loss on location regression
          (only include positive anchor as guarded by p_i^*)

- $N_{\text{cls/reg}}$ normalizing the cls/reg loss by num of involved example
  $\Rightarrow N_{\text{cls}}$ = batch size; N_{reg} = num of positive anchor
- λ the balancing term, default to 10 & robust in a wide range $(1 - 100)$
- each batch sample 256 anchors from the same image & ensure pos : neg $\leq 1:1$
  (share computation across anchors as much as possible)
- cross-border anchors can match to label (thus positive)
  $\Rightarrow$ predict boxes from 0-padding influenced feature (noisy feature)
  $\Rightarrow$ NOT converge
  $\Rightarrow$ cross-border anchors are discarded

◦ Joint Training

- final loss = loss of 1st stage (RPN) + loss of 2nd stage (Fast R-CNN)
  $\Rightarrow$ shared backbone updated by gradient from both branch
- YET, stop the gradient from 2nd stage to RPN
  (as gradient from RPN loss & from 2nd stage can diverge)
  (though still converge after enough training)

◦ Alternating (4-step) Training

- train RPN: pretrain on ImageNet & finetune on detection dataset
- train Fast R-CNN: pretrain on ImageNet & finetune on detection dataset
- finetune 1st stage: using Fast R-CNN backbone as fixed & update RPN branch
- finetune 2nd stage: backbone and RPN fixed & update Fast R-CNN branch
  (could alternatively fine-tune for more times...)

◦ Understanding

- RPN: high-quality region proposal by learning:
  · fully-conv net as sliding window proposal generator
  (use same spatial window at each reg location)
  (use diff regressor for each of the k anchor)
  · benefit from deep model & increasing data
  (e.g. resnet-101 & coco leads to better result)
  · more stable proposal due to fully-conv net setting (translation invariant, upto the network stride)

  $\Rightarrow$ less proposal (≤ 300) required & less burden on 2nd stage
  (top-ranked RPN proposals are accurate)
- $\Rightarrow$ faster & unified end-to-end detection framework

  · enable shared conv between proposal generation & refinement
  · shared backbone optimized with multi-tasking & more end-to-end framework
- anchors as multi-scale prior $\Rightarrow$ detection reference pyramid
  (as predict the adjustment of referenced/anchor box)
  · avoid resource-consuming input image / kernel size pyramid
  · address scale-ratio variance with dedicated params (k regressor for k anchors)

  $\Rightarrow$ multi-scale reference is efficient & effective
- faster inference & training:
  · no image/filter pyramid, but anchors, for multi scale
  · sharing conv for proposals generator & detector
- 2-stage framework:
  · RPN generates proposals by only objectness, with highly imbalanced pos-neg
  $\Rightarrow$ filters out easy negative & bad prior by objectness
  $\Rightarrow$ more balanced training data & good prior for 2nd stage

        · detector further regress & classify (class-specific detection)
      ⇒ 2-stage hierarchical refinement on both regression & classification
      (RPN has its own loss: thus 2-stage, not 1-stage network with a sub-net)
    - improve on hard case, indicated by recall/precision @ high IoU threshold
    - still a 16-stride (= downsample rate) detection network
      (could improve by smaller stride)
    - indeed, 4-branch network & sequential inference
      ⇒ still need to design well-aligned loss for the 1st-stage proposal
      (as no gradient from 2nd stage)

- R-FCN: Region-based Fully Conv Net
  - Bounding Box Encoding
    - similar to Faster R-CNN (anchor-proposal-pred)
  - Inference
    - RPN head generate proposals
    - R-FCN head generate k^2 group of feature maps, each group n feature maps
      (in total k^2n: $n = C + 1$ for cls & $n = 4$ for reg),
      where $k \times k$ the bins num of RoI pooling
    - positional-sensitive RoI pooling
      ⇒ for i-th bin, slice & average the feature maps of i-th group
      ⇒ producing n feature maps of size $k \times k$
    - voting (another average pooling) on RoI features to obtain final cls/reg prediction
    - NMS with IoU threshold = 0.3
  - Structure
    - RPN & detector (conv + ps-roi pooling) sharing backbone
    - atrous ResNet-101 used as backbone (stride reduced $32 \to 16$)
      (1x1 conv to reduce last feature maps channel num to 1024)
    - conv to produce feature maps for $k \times k = 7 \times 7$ bins ps-roi pooling
  - Training
    - loss similar to Faster R-CNN: L_{cls} = cross entropy; L_{reg} = smooth L1
    - alternating training RPN & R-FCN branch
    - $\text{IoU}^{\text{RoI}}_{\text{label}} \geq 0.5$ to be positive & ≤ 0.5 to be negative
    - online hard examples mining (OHEM)
      ⇒ evaluate N=300 proposal & select B=128 RoIs with highest loss for back-prop
  - Understanding
    - location info demanded by detection v.s. translation invariance FCN backbone
      (motivation: solving dilemma & maintain simplicity of FCN)
      ⇒ encode spatial info into different semantic feature maps
      ⇒ demands dedicated conv kernel to infer cls&reg from each bin location
      (e.g. $k = 3$: can have positions of top-left, ..., bottom-right)
      ⇒ dedicated feture maps strongly activated at specific relative position of an obj
    - position-sensitivity are necessary for detection
      (if $k = 1$, i.e. remove spatial info ⇒ NOT even converge)
    - thus, an FCN detection net:
      ⇒ sharing all computation (including cls/reg prediction) across proposals
      ⇒ moving params forward & more params for less computation by FCN fashion
      ⇒ produce dense prediction from each bin before RoI:
        · $\mathcal{O}(1)$ inference time w.r.t. proposals ⇒ enable costless OHEM

        - parallel proposal generation & prediction
    - ⇒ more robust prediction
      (as $k \times k$ prediction made for each proposal)
    - transfer deep classifier into deep detector by adding conv for ps-roi pooling
      ⇒ without introducing per-RoI process (thus no efficiency burden)
    - cls branch: {bin location} × {class}
      ⇒ infeasible for large-category classification (e.g. 1000 classes)

- Feature Pyramid Network (FPN)
  - Bounding Box Encoding
    - same as Faster R-CNN & R-FCN
    - anchors assigned to different pred head according to anchor scale P_k,
      where P_k denote a feature map of stride 2^k (i.e. downsampled by 2^k times)
  - Inference
    - infer as 2-stage detection, with backbone network replaced by / extended to FPN
    - shared network head (e.g. RPN head) predict at each pyramid level
    - assign RoIs to scale/pyramid level k, P_k (similar to anchor assignment),
      with $k = \lfloor k_0 + \log_2(\frac{\sqrt{w_r h_r}}{244}) \rfloor$,

      where 244 the canonical imagenet pretraining size

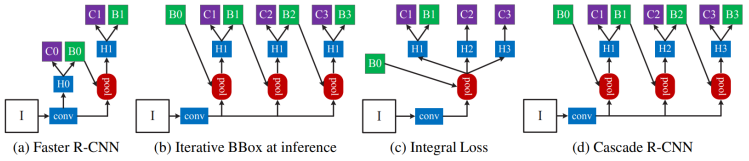
      $k_0 = 4$ the level to which RoI of $w \times h = 244^2$ should be assigned

      $\frac{\sqrt{wh}}{244}$ the downsample rate ($\log_2$ as stride-2 downsample)

      ⇒ assign large RoI to low-res (e.g. P_5), small RoI to high-res (e.g. P_2)
    - 7x7 RoI pooling for each RoI at its scale level
    - prediction made by a pred head shared across all levels
    - convert pred at each scale/pyramid level, using anchor assigned to that level
  - Structure
    - downsample (bottom-up): downsample at each ResNet stage
      ⇒ collect last layer of each stage $\{C_2, C_3, C_4, C_5\}$
    - upsample (top-down): interpolation (e.g. nearest neighbor, bilinear)
      ⇒ results in corresponding $\{P_2, P_3, P_4, P_5\}$ (generate P_5 first)
    - skip connection: 1x1 conv to match channel to $d = 256$ & element-wise adding
      ⇒ residual connection to associate across resolution & semantic level
  - Training
    - 15 anchors = $\{32^2, 64^2, 128^2, 256^2, 512^2\} \times$ {1:2, 1:1, 2:1}
      (pos-neg assignment as in Faster R-CNN, i.e. RPN)
      ⇒ assigned to $\{P_2, P_3, P_4, P_5\}$ according to scale
      (e.g. P_2 accounts for 32^2 anchor, P_5 accounts for 512^2 anchor)
    - associate label box to anchor as usual (thus to each pyramid/scale level)
    - train as Fast/Faster R-CNN
  - Understanding
    - provide both high-resolution & high-level semantic info from a single scale input
      ⇒ provide feature pyramid with rich semantic info
      (dedicated feature map for each scale)
      (instead of a single high- res, high-level feature map)

- prediction independently made at each scale
  (analogs to using image pyramid: easily generate to more/less-level pyramid)
  ⇒ alleviate the mismatch between: RPN receptive field & actual object size
- ⇒ explicitly address multi-scale problem by design
  (effective pyramid from learning)
  ⇒ much better in detection small object
- remove the need of image pyramid at test time
  (still needed as deep feature / anchors are not enough for multi-scale problem)
  ⇒ much faster inference & consistent train-test
- yet, still slow inference in industrial deployment: slow post-processing
  (too many pred due to large final feature map & per-box post-processing)
- skip connection
  - residual adding (v.s. concat in U-net) to ensure effective depth increase
  - mitigate the large semantic gap between resolutions (in downsample path)
  - fix the location of upsampled feature

- Cascade R-CNN
  - Bounding Box Encoding
    - same as Faster / Fast R-CNN: encode as the adjustment on proposals
  - Inference
    - extract feature map from backbone (e.g. ResNet)
    - stage-0 (RPN head) generates pred boxes
    - sample pred boxes into pos-neg (class-specific) examples according to IoU threshold
      ⇒ construct proposals from pred of stage $t-1$
    - stage-t head (classifier & regressor) predict (with RoI pooling) based on previous proposals
    - ensemble pred for all stages with $t \geq 1$, i.e. except RPN
      (as NOT all proposals can pass to the last stage: can get filtered out by classifiers)
  - Structure
    - shared backbone (e.g. ResNet, FCN)
    - RPN head to generate initial proposal
    - cascaded multiple R-FCN head for class-specific classification & regression
    - comparison:



      where "pool" the roi pooling, "B0" the proposals, "H" the network head
  - Training
    - loss at stage t:
      $L(\mathbf{b}^t, \mathbf{g}) = L_{\text{cls}}(h_t(x^t), y^t) + \lambda[y^t \geq 1] L_{\text{loc}}(f_t(x^t, \mathbf{b}^t), \mathbf{g})$,
      where h_t, f_t the stage t head for classifier, regressor

      $\mathbf{b}^t = f_{t-1}(x^{t-1}, \mathbf{b}^{t-1})$, with x^t the image (feature map) patch cropped by $\mathbf{b}^t$

      $\mathbf{g}$ the corresponding label box of current examples

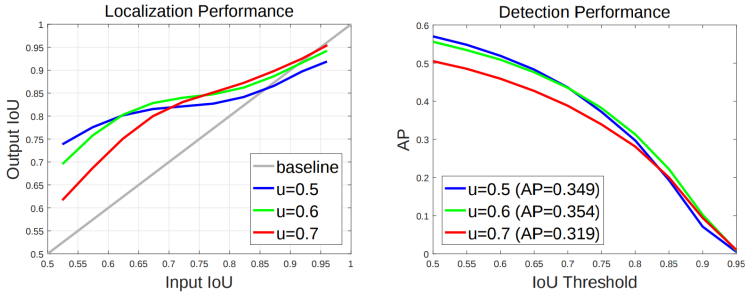
      y^t the class label for $\mathbf{b}^t$, considering $\text{IoU}_{\mathbf{g}}^{\mathbf{b}^t}$ under the threshold u^t

      (y^t = class of $\mathbf{g}$ if IoU $> u^t$; else 0)

- $\Rightarrow L = \sum_t L(\mathbf{b}^t, \mathbf{g})$, with $\{u^t\} = \{0.5, 0.6, 0.7\}$ for $t \geq 1$
  (stage-0 is RPN: use its default sampling strategy)
- sequential, stage-by-stage training: improve based on a fixed input distribution

◦ Analysis: Ineffectiveness of Single Detector - Performance Perspective

- optimization of different points of ROC requires different loss function
  $\Rightarrow$ detector performs best
  · with the proposals IoU close to its training IoU threshold (left)
  · when evaluated under IoU threshold close to its training IoU threshold (right)

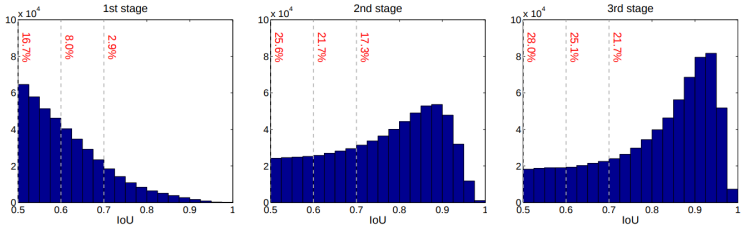


- $\Rightarrow$ detector performs best when NO train-test mismatch
  i.e. can NOT be best
  · for proposal of all input IoU
  · under all evaluation IoU threshold

◦ Analysis: Ineffectiveness of Single Detector - Data Distribution Perspective

- quality: the threshold u for $\text{IoU}^{\text{proposal}}_{\text{label}}$ to consider a proposal to be positive
  $\Rightarrow$ quality of detector = the IoU threshold u under which it is trained
  = the quality of its input proposals

- increase quality leads to exponential decrease in positive proposal num
  $\Rightarrow$ delimma:
  · low quality allows close false positive $\Rightarrow$ noisy positive
  · high quality allows little positive $\Rightarrow$ overfit the positive & extreme imbalance
  (also need to operate on low-quality proposal at test time)



where red number the positive percentage of examples higher than that IoU
$\Rightarrow$ distribution at each IoU quality are different

◦ Analysis: Effectiveness of Cascading

- observe that output IoU are better than input IoU (if detector operates on its quality)
  $\Rightarrow$ enable boostrapping
- each stage improves the average IoU (quality) of all proposals
  $\Rightarrow$ more balanced dataset under higher quality (for the next stage)
  (as proposals are improved by previous stages)

◦ Understanding

        - v.s. iterative bbox regression
            - not a post-processing process, but involved in training
            - use specialized regressor, instead of the same one applied multiple times

          ⇒ each regressor operates on its training quality & distribution: no mismatch
        - v.s. integral loss
            - not sharing regressor: each head a complete detector
            - avoid operating on the same input distribution (resample by producing new proposal for next stage)

          ⇒ more balanced pos-neg for classifier at each IoU threshold
        - sequentially resample & improve proposals to higher quality
            - each stage optimizes for a specified IoU quality: specialized pred head
            - each stage improves the distribution of higher quality to be more balanced

          ⇒ sequentially more selective against close false negative
          ⇒ enable precise detection at high quality (i.e. high IoU threshold)
        - input distribution of t stage is ensured by $t-1$ stage
          ⇒ NO train-test mismatch
        - multi-stage & multi-task: stage-1 already improves over original Faster R-CNN (multi-tasking at various IoU threshold)
          ⇒ end-to-end model ensemble
        - more stages lead to better AP at higher IoU threshold (yet, 3-stage achieves a good trade-off)

- Deformable Conv Net (v1)
    - Bounding Box Encoding
        - same as Fast / Faster R-CNN
    - Inference
        - backbone (with deformable conv) extracts feature & RPN predicts proposals
        - RoI pooling sampling location is also augmented by learnt offset (similar to deformable conv)
        - 2nd-stage produce final pred
    - Structure
        - ResNet-101 with last 3 layers (in the 5^{th}/last stage) replaced by deformable conv
          ⇒ performance gain saturate if using deformable conv in ≥ 3 layer
        - deformable RoI pooling (could also be deformable ps-roi pooling)
          ⇒ another roi produce normalized offset $\Delta\hat{\mathbf{p}}_{ij} = \frac{1}{\gamma}(\frac{\mathbf{p}_{ij}[x]}{w}, \frac{\mathbf{p}_{ij}[y]}{h})$,
          where $\gamma = 0.1$ a scalar, w-h the size of roi, $\mathbf{p}_{ij}$ the offset for (i,j)-th bin
    - Training
        - similar to Faster R-CNN
    - Understanding
        - more in deformable conv

One-stage Detection

- You Only Look Once (YOLO)
    - Preprocessing

- grid the input image into disjoint $S \times S$ cells
  (apply usually fine grid, e.g. $S = 19$)
- assign each object to a cell by its central point
  $\Rightarrow$ try ensuring maximal **1** object in a grid

◦ Bounding Box Encoding

- $conf = p(\text{obj}) \times \text{IoU}^{\text{label}}_{\text{pred}}$ confidence of current box containing an object
  $\Rightarrow$ 0 desired if none, IoU between pred and label desired if exist
  where, $p(\text{ob}j)$ the probability of object existence in the CELL
- $x, y \in [0, 1]$ the box center in the cell, normalized by cell size
  $\Rightarrow$ easier to learn, as dense layer not confused by varying locations
- $h, w \in [0, 1]$ the box size normalized by image size
  $\Rightarrow$ easier to learn for dense layer

◦ Classification Encoding

- $p_1, ..., p_C$ the conditional probability $p(\text{class}_c|\text{obj})$ for object inside the CELL
  i.e. the class probability of object, given object existing & assigned to the cell

◦ Inference

- for each cell, predict both $[p_1, ..., p_C]$ and B bounding boxes $[conf, x, y, w, h]$
  $\Rightarrow$ final prediction as a $S \times S \times (C + 5B)$ tensor
- $p_c \cdot conf = p(\text{class}_c|\text{obj}) \cdot p(\text{obj}) \cdot \text{IoU}^{\text{label}}_{\text{pred}} = p(\text{class}_c, \text{obj}) \cdot \text{IoU}^{\text{label}}_{\text{pred}}$
  where p_c from the cell, $conf$ from each box in the cell
  $\Rightarrow$ combine the decoupled pred for various class-specific boxes prediction
- non-max suppression to fix multiple detection box (at test time)

◦ Structure

- 24 conv layer downsampling from $448 \times 448 \times 3$ to $S \times S \times 1024$
- followed by: $\rightarrow$dense with dropout$\rightarrow$ 4096 $\rightarrow$dense$\rightarrow$ $S \times S \times (C + 5B)$
  (dropout to prevent co-adaption, i.e. dependent units between 2 dense layers)
- leaky ReLu $a = \max(z, 0.1z)$ used, except for linear activation in final layer

◦ Optimization Goal

- $$\begin{aligned}\text{loss} =& \lambda_{\text{obj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \\ & \lambda_{\text{obj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] + \\ & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{obj} (conf_{ij} - \hat{conf}_{ij})^2 + \lambda_{\text{none}} \cdot \mathbf{1}_{ij}^{none} (conf_{ij} - \hat{conf}_{ij})^2 + \\ & \sum_{i=0}^{S^2} \mathbf{1}_{i}^{obj} \sum_{c=0}^{C} (p_{ci} - \hat{p}_{ci})^2\end{aligned}$$
- $\lambda_{\text{obj}} = 5$ to up-weight box with obj, $\lambda_{\text{none}} = 0.5$ for empty box
  - balance the dataset: avoid large num of empty box pushing $conf$ pred to 0
  - emphasize on predicting box responsible for object
    (adjust its pred x, y, w, h)
- $\mathbf{1}_{ij}^{obj}$ the indicator: 1 if obj exists in j box of i cell, similar for $\mathbf{1}_{ij}^{none}$
- minimize $\sqrt{w_i}, \sqrt{h_i}$: mitigate various box size into similar scale
  (as same error occurred in small box matter more than that in big box)
- p_{ci} the classification prob for i cell (only trained if it contains obj)

- Training
  - optimize the sum-squared error (due to its simplicity)
  - for each cell: correct its classification if it contains obj
  - for each box:
    - increase $conf$ & adjust box pred if it overlaps the most with label of its cell
    - decrease $conf$ for other box (not responsible for having obj)
  - pre-training: pretrained first 20 conv layers on classification dataset
  - data augmentation: random scaling size, adjusting exposure and saturation
- Limitation
  - use only coarse feature to predict box
    (due to multiple downsampling & produced by dense layer)
  - NOT able to handle: multiple objects of the same type in a cell
    (NOR when same-type objects more than total pred boxes)
    $\Rightarrow$ suffer from small objects in group (e.g. birds)
  - struggle in box localization
- Understanding
  - bounding box localization & classification as regression problem
    (predicted totally by a dense layer)
  - end-to-end optimizing directly the detection score
    (vs. separate modules: region proposal, bounding box localization, classification)
  - global information for each bounding box localization & classification
    (instead of taking only local info to classify boxes)
    $\Rightarrow$ less false alarm (mistake background as object)
  - generalizable representation for detection
    (as feature extraction embedded in the network)
    $\Rightarrow$ able to generalized to art work
  - much less predicted box (compared with 2000 boxes in R-CNN framework)
  - fast $\Rightarrow$ ensemble with (fast) R-CNN framework with no overhead
    $\Rightarrow$ mitigate the false alarm in R-CNN framework

- YOLOv2, YOLO9000
  - Preprocessing
    - no more explicit grid, implicitly set by downsampling factor
      (similar to spatial pyramid pooling)
    - select N_A anchor boxes: covering most of the interested objects
      (e.g. tall-thin box for pedestrian, low-wide box for car)
    - assign the objects to a tuple (cell, anchor/prior box)
      - assign to cell by its central point $\Rightarrow$ try ensuring maximal N_A object in a cell
      - assign to one from N_A anchor boxes depending on its IoU with all N_A boxes
        (NOT assigned, if all $\text{IoU}^{\text{label}}_{\text{anchor}} < 0.5$)
  - Anchor Box Selection (Prior)
    - run K-means on all label box in train set
      with distance $\text{dist}(\text{box}, \text{centroid}) = 1 - \text{IoU}(\text{box}, \text{centroid})$
    - use cluster centroid as the anchor box
      $\Rightarrow$ for anchor box $a \in A, a = [w_a, h_a]$
    - num of anchor box (centroid) N_A reflect recall $\leftrightarrow$ model complexity trade-off
  - Bounding Box Encoding

- $t_x, t_y \in (0, 1)$ the box center in the cell, normalized by cell size
  $\Rightarrow$ true pred center $x = c_w\sigma(t_x) + c_x, y = c_h\sigma(t_y) + c_y$ where c_x, c_y, c_w, c_h the cell
  $\Rightarrow$ easier to learn, as YOLO
- t_w, t_h coefficient to adjust anchor box size
  $\Rightarrow$ true pred size $w = w_a e^{t_w}, h = h_a e^{t_h}$ where w_a, h_w the anchor box size
  $\Rightarrow$ more robust, avoid influence from anchor prior
- $p_1, ..., p_C$ classification pred $\Rightarrow$ decouple from cell, each box a classification

◦ Classification with Hierarchical Encoding

- tree-structure for multi-label: a hierarchical softmax
  (hence, NOT assuming mutual exclusion between class)
- lookup WordNet for class label & its path to the selected root "physical object"
  (if multi-path, choose the one with least new word involved)
  $\Rightarrow$ build a WordTree
- multiple softmax: one for each set of direct children of a node (if not leaf)
  e.g. at node "dog": predict $p(\text{malamute dog}|\text{dog}), p(\text{terrier dog}|\text{dog}), ...$
  $\Rightarrow$ pred a Bayesian network, according to WordTree
- $\Rightarrow t_o$ objectiveness by summing conditional prob accordingly
  $\Rightarrow$ true objectiveness: $conf = p(\text{obj}) \cdot \text{IoU}^{\text{label}}_{\text{pred}} = \sigma(t_o)$

◦ Inference

- for each cell, N_A bounding boxes: [spatial info$[t_x, t_y, t_w, t_h]$, a flatten WordTree$[p_1, ..., p_C]$]
  $\Rightarrow$ final prediction as a $\frac{I}{32} \times \frac{I}{32} \times N_A(4 + C)$ tensor (each box a WordTree!)
- objectiveness of each box: from WordTree of each box
- box classification: choose highest confidence path at every node
  (till a specified threshold or a leaf node reached)
- non-max suppression to fix multiple detection box (at test time)

◦ Structure

- fully conv net: downsample by 32, directly predict $\frac{I}{32} \times \frac{I}{32} \times (C + 4N_A)$ tensor
- passthrough layer with concat (1% $\uparrow$)
  e.g. spatially stack $26 \times 26 \times 512$ into $13 \times 13 \times 1024$ before concat
- batch norm in ALL conv layer ($> 2\%$ mAP$\uparrow$)

◦ Training

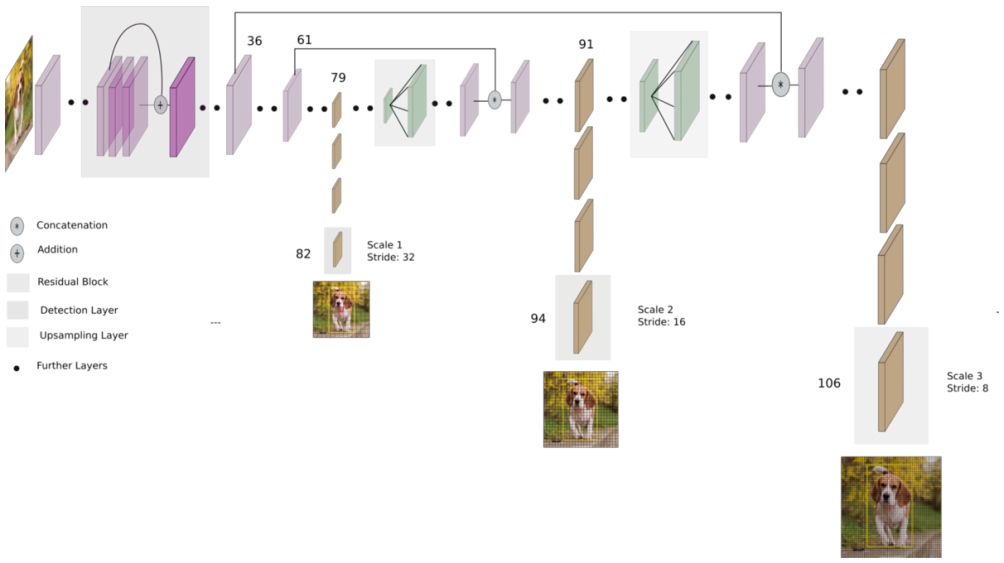
- pre-training on classification with data augmentation
  (e.g. random crop, rotation, hue, saturation and shifting shift)
- more time for model fine-tuning on detection dataset ($\sim 4\%$ mAP$\uparrow$)
  as model takes time to adjust to resolution change
- multi-scale training s.t. model able to cope with varying resolution
- joint training: mix detection & classification dataset ($\sim 5\% \uparrow$)
  $\Rightarrow$ for detection label, full loss available
  $\Rightarrow$ for classification: only classification & objectiveness loss considered
  - · select responsible pred box by the largest objectiveness
  - · objectiveness loss: considered if t_o of responsible box $<$ threshold (0.3)
  - · classification loss: only nodes in its WordTree above current label considered
    with $p(\text{physical object}) = 1$ for label
- balancing joint dataset: oversampling detection dataset
  $\Rightarrow$ s.t. detection data : classification data = 1:4

◦ Limitation

- each box a classification: too much output channel

        - still, detectable objects number restricted by anchor box num in a cell (e.g. group of small objects)
    - Understanding
        - still, fast & global context for box detection
        - auto-select prior, better than handpicked (examined by using anchor box directly as prediction)
        - adjustable speed↔accuracy trade-off via input resolution (enabled by being fully conv & multi-scale trained) ⇒ larger resolution, slower, more accurate
        - joint training with WordTree: multi-task (detection + classification) learning ⇒ enrich classes to detect & more robust in class-detection ⇒ graceful degrade on detecting new/unknown data
        - decoupled localization & classification ⇒ able to handle some multiple same-type objects in same cell

- YOLOv3
    - Preprocessing
        - same as YOLOv2
    - Bounding Box Encoding
        - t_x, t_y, t_w, t_h: same as YOLOv2
        - t_o the objectiveness, modeling directly $p(\text{obj})$, instead of $p(\text{obj}) \cdot \text{IoU}$
        - $p_1, ..., p_C$ the classification of obj: conditional prob $p(\text{class}_c|\text{obj})$, as YOLO
    - Classification Encoding
        - independent logistic regression for each class (instead of hierarchical softmax) ⇒ multiple binary classification (NOT assuming mutual exclusive classes) ⇒ multi-label approach (box may contain more than one class)
    - Inference
        - for each cell, N_A bounding boxes $[t_x, t_y, t_w, t_h, t_o, p_1, ..., p_C]$ (N_A the num of anchor box)
        - ⇒ final prediction as a $\frac{I}{N^l} \times \frac{I}{N^l} \times N_A(C+5)$ tensor where N^l the size of downsampled feature map (at pred layer l)
    - Structure
        - residual blocks & batch norm as YOLOv2
        - 2D encoder-decoder structure with concat connection, with 3 pred branches ⇒ final num of pred boxes $\sum_{l \in \text{pred layer}} (N^l \times N^l \times N_A)$
    - Optimization
        - sum of the binary cross-entropy for classification loss
    - Training
        - pretrain backbone (feature extractor) on classification
        - multi-scale training & data augmentation as YOLOv2
    - Failed Approach
        - box x, y as an offset to anchor box position x_a, y_a (stability decreased)
        - linear regression for x, y (worse than bounded by logistic)
        - focal loss (already solved by decoupled $p(\text{obj})$ & $p(\text{class}_c|\text{obj})$)
        - dual IoU threshold in assigning object (need more tunning for stabilized model)



- Understanding
  - real-time applicable detector
  - detection at multiple scale, with encoder-decoder structure
  - decent localization + great box classification
    $\Rightarrow$ emphasize on box classification, since human insensible to IoU change

12.5.3 Object Tracking

Problem Formulation

- Input Data
  - Raw Data
    - time-series observation from sensor $\Rightarrow$ need to embed detection
  - Object Representation
    - bounding box: 2/3-D size, with pose, motion, velocity, etc.
    - point model: centroid point with attributes denoting an object (including bbox)
    - silhouette: for non-rigid object
    - articulate: for articulated model, e.g. human skeletal model
  - Object Existence
    - probability density for existence
    - probability of existence as a feature in representation info
      (objectiveness in detection)
- Goal
  - Trajectory
    - recover the true trajectory of the object
      (including current location, potentially future prediction)
- Overview
  - Online Learning (Tracking)

        - trade off speed - model complexity, as train on arriving frames
        - adaptive, as accounting the history info of a track
          ⇒ may provide more info (e.g. covariance matrix from Kalman filter)
    - Offline Tracking
        - learn similarity functions between frames offline
        - fast, as no online training needed ⇒ but NO explicit adaptive
        - batch method: generate the track after all frames examined
    - Tracking-by-Detection
        - detect target(s) in each frame; link target into track
          ⇒ as a two-stage problem
        - ⇒ viewed as meta learning / one-shot learning
          (train on first frame, detect on subsequent frames)

- Challenge
    - Appearance Model
        - robust target-specific model
        - shape&color can be trap: changing clothes etc. ⇒ learning directly transformation by image pair (e.g. siamRPN)
    - Long Term Dependency
        - re-identification: people leaving and re-entering the scene
        - hot standby surveillance
    - Initialization & Termination Criteria
        - bad initialization: overlapped box due to overlapped object
          (e.g. two closely standing people)
        - termination vs. occlusion
          ⇒ need to tell leaving the view or temporally occluded
    - Fast Motion
        - large search area, hence more background noise, hence similarity can fail
        - motion blur (which can be modeled as appearance change)

- Evaluation Metrics
    - Success
        - overlap success rate: average pred-label box IoU
        - overlap success plot: the rate of pred-label box IoU $\geq$ a given threshold
          ⇒ can have AUC
        - orientation success: diff of pred-label yaw angle $\leq$ threshold $= 10^\circ$
        - other success...
          ⇒ review the overall estimated quality
    - Normalized Cumulative Sum of Success vs Normalized Time
        - a plot ⇒ review tracking quality over time
    - Trajectory Difference
        - compare the similarity of pred-label trajectory
          (may consider abrupt change, slowly drift, and etc.)
    - VOT Metrics
        - robustness: failure times (frame cnt of lost track)

        - accuracy: average overlap while tracking successfully
        - expected average overlap (EAO): accounts for both accuracy & robustness

- Dataset
    - Video (Single) Object Tracking
        - Youtube-BB
            - $> 100,000$ videos, annotated once every 30 frames
        - ILSVRC (ImageNet) VID
            - ~ 4000 videos annotated frame-by-frame

- State-of-the-Art
    - Data Balance
        - avoid simple negative (foreground-background in siamese tracker)
          $\Rightarrow$ use negative box with semantic instance (discussed in DaSiam)
          $\Rightarrow$ cascaded RPN to filter out hard negative
        - triplet loss: mine the relation of (template, positive instance, negative instance)
          $\Rightarrow$ focus on making correction decision on hard neg
        - scale balance: having enough detail for semantic similar distractor
          cascaded RPN: feature-transform-block for multi-scale feature
          siamgrpn++: multi-level feature (enough diversity from deepnet)
          (yet, in siamRPN, model overfits to object scale)
    - Deep Power
        - revealed to be central bias in deepnet, due to padding
          (network can realize input patch location by if feature is padding influenced)
          (padding on template prevent xcorr from effectively measuring similarity)
          $\Rightarrow$ siamRPN++: cropping for template branch & data augmentation for search branch
          $\Rightarrow$ siamDW: crop-inside unit: as early as where the padding is introduced
        - worse localization due to large receptive field & accumulated stride
    - $\Rightarrow$ Discriminative Ability
        - deep feature via deepnet
        - mining pos-neg for better discriminative ability
        - use background info in tracking stage, instead of only in training phase
          (introducing optimization approach?)
    - Finer Detail
        - instance tracking via multi-tasking with video segmentation
        - $\Rightarrow$ attention based cropping?

Detection in Tracking

- Single-frame Detection
    - Bounding Box Detection
        - YOLO, R-CNN, etc.
    - Point Detection
        - detect landmarks
    - Background Modeling

        - segmentation, ...
- Temporal Detection
    - Optic Flow
        - able to represent non-rigid, deformable object
        - yet, may failed in moving foreground (e.g. birds, fog, smoke...)
    - Motion Detection
    - Orientation
        - frame differencing of location, ...
    - Background Modeling
        - adaptive background, ...

Single-Object Tracking

- Kalman Filter
    - Understanding
        - predict covariance ⇒ better gating in association
        - cooperate with noise
          ⇒ able to modified to account association uncertainty
- Particle Filter
- Correlation Filter
    - Detector
        - online-training to learn the object appearance at previous frame
          ⇒ conv over test (current) frame for similarity response
    - Tracker
        - no explicitly modeled, though can augment online-training data with past images
    - Inference
        - $\mathbf{w}$ the weight to learn $\mathbf{x}$ input image of current frame, y the desire response
        - $\mathbf{x}[\tau]$ the circular shift of $\mathbf{x}$ by $\tau = [\tau_x, \tau_y]$ pixel
        - formulated as regression problem
          $\Rightarrow \arg_{\mathbf{w}} \min \frac{1}{2} \sum_{n \in N} \|\mathbf{x}[\tau_n]^T \star \mathbf{w} - y_{\tau_n}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$
          (mosse: minimum output sum of squared error, with regularization)
        - original solution for regression $(X^T X + \lambda I)^{-1}$, where $X = \{\mathbf{x}[\tau_n] \mid n = 1, ..., N\}$
          (corresponding $\mathbf{y} = \{y_{\tau_n} \mid n = 1, ..., N\}$)
          $\Rightarrow \mathcal{O}(N^3)$, where N the num of training example in X
        - since $\mathbf{x}[\tau_n]$ a circulant toeplitz matrix ⇒ fourier transform into diagonal matrix
    -
- GOTURN:
    - Detector
        - CNN
    - Tracker
        - no explicit tracker (not modeling temporal info)

- Inference
  - crop t^{th} image: centered at current bounding box with context
  - crop $t+1^{\text{th}}$ image: centered at bbox in t, doubled size (gating)
  - encode both image via CNN detector; concat encoding & feed to dense layer
  - regress the bbox in $t+1^{\text{th}}$ crop
- Tracking
  - arbitrary target object selected for tracking (treated as ground truth)
  - 1-step prediction of location, then predict further on...
- Training
  - train with consecutive image pair froxm sequences
  - $L1$ loss for exact match, as it does NOT smooth out in $(0^-, 0^+)$ as $L2$ loss
    $\Rightarrow$ mitigate the introduced motion noise
  - data augment: motion noise applied on the 2^{th} image; random crop on the 1^{st}
    (motion noise $\sim$ Laplace distribution and prefer smooth & small motion)
- Understanding
  - 2-frame model as a simple local generic detector, no explicit temporal tracking
    $\Rightarrow$ learn to locate the nearest similar object by comparing $t-1, t$ frame
    $\Rightarrow$ learn to discover relationship between object appearance
    (by the Siamese setting)
    $\Rightarrow$ refine bbox for nearest similar object with one proposal ($t-1$ bbox)
  - $\Rightarrow$ able to specialize tracker for specific object tracking
    e.g. fine-tune on car video for car tracker
  - can NOT possess intrinsic invariance of object movement at various direction
    $\Rightarrow$ detector needs to be trained by data augmentation for object at all positions
    (due to dense layer)
  - can NOT handle large drastic target change: preferring local & smooth change
  - Not good at exact frozen scene: training augmented with noisy motion
    (could be worse if NOT using $L1$ loss)
  - real-time timing (100fps), included in opencv $\Rightarrow$ stable performance

- SiamFC: Fully-Conv Siamese Network for Object Tracking
  - Detector
    - siamese CNN to extract feature from exemplar & search image
  - Tracker
    - no explicit tracker, yet cross-correlation to measure spatial similarity
  - Inference
    - exemplar and search image cropped with target in the center & feature extracted into z, x
      (at most T frames apart)
    - conv exemplar z on search image x for response map of similarity
    - maximum score in response map as object in cur frame & fit a box accordingly
  - Structure
    - AlexNet as CNN
  - Training
    - logistic loss for final response map from final cross-correlation
      (cosine window applied to penalize large displacements)

        - weighted for pos-neg balance
        - bbox encoded as a mask with a square area of $1s$
        - image cropped with padding & scaled to 127×127 & 255×255 for convenience
        - 50 epochs (50000 examples per epoch), batch size 8
    - Tracking
        - crop with padding on box of 1^{st} frame, crop with $t-1^{\text{th}}$ box padded to $4\times$ size on frame t
        - tracking with scale: processing multi-scaled version of search image (5 scale $1.025^{\{-2,-1,0,1,2\}}$)
        - produce response map of similarity & upsampled to 272×272 for better localization
    - Understanding
        - 2-frame model: measuring spatial similarity by correlation for localization
          $\Rightarrow$ an unrolled RNN trained with length 2
          $\Rightarrow$ strong initialization for RNN-based tracker
        - learning strong offline embedding for exemplar-search similarity measurement
          $\Rightarrow$ complementary to online tracker
        - updating exemplar z in time series NOT gaining much (as initial box assumed to be a great one)

- Re^3: Real-time Recurrent Regression Network
    - Detector
        - CNN to extracts multi-scale representation from image
          (more descriptive info e.g. human in red/blue shirt)
          $\Rightarrow$ a siamese net as encoding
    - Tracker
        - two-layer, factored LSTM, taking tracker input at both layer
          $\Rightarrow$ longer dependency & more complex object transformation with 2 layers
        - hidden state as tracker state
          $\Rightarrow$ forward-prop to update (no training)
    - Inference
        - crop t^{th} image: centered at current bounding box, extended to twice of box size
        - crop $t+1^{\text{th}}$ image: centered at box of t, doubled size (gating)
        - late fusion: concat CNN output from $t, t+1$ & fed into dense layer for fusion
        - LSTM tracker updates on fused info
        - dense layer regress the tracker (LSTM) output for bounding box at $t+1$
    - Training
        - bounding box defined by up-left, right-bottom coordinates in the crop
          (hence as a ratio of bounding box size)
        - $L1$ loss to encourage exact match
        - short sequence (2 unrolls) & multi-batch (64) to overcome plateaus
          $\Rightarrow$ slowly increase to 32 unrolls & batch size 4
        - use ground-truth crop when training with short sequence
          $\Rightarrow$ slowly increase probability to use predicted crop, when increasing unrolls
          $\Rightarrow$ to prevent from accumulating drifts
    - Data Augmentation
        - utilize detection dataset: crop the object with bounding box

        - random crop a patch from the same image as background
        - occluders randomly taken from the same image
        - object with box initialized with velocity with Gaussian noise
    - Tracking
        - an initial bounding box over arbitrary object given at the start
        - crops fed into net at each time step
        - LSTM state reset after each 32 frames (as maximally trained with 32 unrolls)
          ⇒ hot-start by using the state in 1^{st} forward pass (instead of $\mathbf{0}$)
          ⇒ preserve the initial encoding & recover from drift
    - Understanding
        - end-to-end training for both detector and tracker
        - model regress changes to the box ratio ⇒ easier as bbox refinement
        - LSTM tracker maintain track state ⇒ temporal fusion
          ⇒ overcome occlusion, update for variance/shape change
          (observation usefulness modeled by LSTM tracker)
        - LSTM tracker needs specialized training, and single-layer LSTM NOT enough
          (or, can hurt performance due to instability)
        - LSTM state reset: prevent drift, yet can fail if initial box overlaps other object
        - may still, drifted to similar nearby object, known object (e.g. face), large motion
          (comparison & failure: https://youtu.be/RByCiOLlxug?t=214)

- CFNet: Correlation Filter based Tracking (SiamFC v2)
    - Detector
        - siamese CNN
    - Tracker
        - correlation filter embedded in forward prop ⇒ online-learning 1-step tracker
    - Correlation Filter Block
        - input training img $x \in \mathbb{R}^{m \times m}$, test img z
        - $w \star x$ construct
    - Inference
        - crop t frame at box center of $t-1$ frame, with $4\times$ larger & siamese CNN extracts feature from
          ⇒ extracted last frame feature (train) & cur frame feature (test) into x, z
        - solving correlation filter optimization: get circular shift of x by circular linear matrix k
          ⇒ via Lagrangian dual, turn optimization into linear equations system
          ⇒ solve for equations system & construct back prop map
          (can even learn the desired response y instead of assuming it to be Gaussian)
        - crop CF output & convolve with test image z to regress box
    - Structure
        -
    - Training
    - Understanding
        - correlation filter embedded in forward prop
          ⇒ utilize online-learning to acquire prior for target (while tracking generic object)

- enable ultra-lightweight network comparative with deep net
  (though correlation filter layer does NOT improve the ceiling performance)
- updating template on each frame (v.s. using always 1^{st} frame)

- **SiamRPN: Tracking with Siamese Region Proposal Network**
  - Tracking Initialization
    - take initial frame & given bbox as template z
    - CNN (modified AlexNet) obtains a feature map $\varphi(z)$
      (crop $(w+p)\times(h+p)$ with padding $p=\frac{w+h}{2}$ & resized to 227×227)
  - Detector
    - in $t>1$ frames, each frame as detection frame x
    - same CNN (modified AlexNet) extracts feature maps into $\varphi(x)$
      (crop $2(w+p)\times2(h+p)$ & resized to 255×255)
    - 2 independent & identical branch in RPN: classification/regression branch
      $\Rightarrow$ conv to get $\zeta[\varphi(z)],\zeta[\varphi(x)]$ & conv with $\zeta[\varphi(z)]$ as kernel over $\zeta[\varphi(x)]$
      (with different output branch)
    - use kernel from template branch to convolute on feature map of detection branch
      (same channel num)
      $\Rightarrow$ xcorr directly regress bbox based on response map
      (template branch has a following 1x1 conv to adjust channel)
  - Tracker
    - tracking by (one-shot) detection
  - Bounding Box Encoding
    - anchor box $x^{\text{an}},y^{\text{an}},w^{\text{an}},h^{\text{an}}$, ground truth box $x^{\text{gt}},y^{\text{gt}},w^{\text{gt}},h^{\text{gt}}$
    - perfect prediction as $\delta x=\frac{x^{\text{gt}}-x^{\text{an}}}{w^{\text{an}}},\delta y=\frac{y^{\text{gt}}-y^{\text{an}}}{y^{\text{an}}},\delta w=\ln\frac{w^{\text{gt}}}{w^{\text{an}}},\delta h=\frac{h^{\text{gt}}}{h^{\text{an}}}$
      $\Rightarrow$ predict box as $x^{\text{p}}=\delta x w^{\text{an}}+x^{\text{an}},y^{\text{p}}=\delta y h^{\text{an}}+y^{\text{an}},w^{\text{p}}=e^{\delta w}w^{\text{an}},h^{\text{p}}=e^{\delta h}h^{\text{an}}$
  - Inference
    - crop on template & search img & extract feature volume
    - take template volume as kernel, convolute on feature map of current frame
      $\Rightarrow$ RPN takes resulted response map as proposal & regress bboxes
    - select top K response in classification branch as set of $\{(i_k,j_k,c_k)\mid k\in(0,K)\}$
      $\Rightarrow$ use the location $\{(i_k,j_k,c_k)\}$ to obtain corresponding anchor & pred
      $\Rightarrow$ generate K proposal as $\{x_k^{\text{pro}},y_k^{\text{pro}},w_k^{\text{pro}},h_k^{\text{pro}}\}$
    - box refinement: only central $g\times g$ anchors remain considered
      (use the center size of search region)
    - size change penalty $e^{k*\max(\frac{r_t}{r_{t-1}},\frac{r_{t-1}}{r_t})*\max(\frac{s_t}{s_{t-1}},\frac{s_t}{s_{t-1}})}$,
      where r_t the bbox $\frac{w}{h}$ ratio of frame t, $s^2=(w+p)(h+p)$ with padding $p=\frac{w+h}{2}$
      $\Rightarrow$ bbox score (objectness/classification response) multiplied by penalty
    - non-max suppression on re-ranked boxes for final bbox
      (target size updated by linear interpolation for smooth translation)
  - Tracking
    - arbitrary target object selected for tracking (treated as ground truth)
      $\Rightarrow$ extract the feature template $\varphi(z)\&\zeta[\varphi(z)]$
    - each $t>1$ frame, perform feature extraction, detection with kernel $\zeta[\varphi()z]$ & bbox regression
      $\Rightarrow$ formulate as task of a one-shot learning for detection
    - gating on frame t based on $t-1$ result

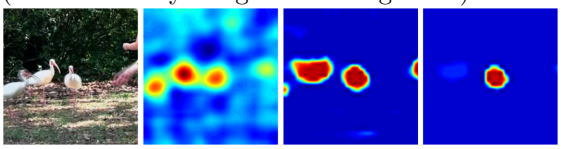
    - detect target within gated region of frame t
      (the best remained & adjusted anchor as the target to track)
  - Training
    - crop input by $\sqrt{(w+p)*(h+p)}$, where padding $p = \frac{w+h}{2}$ & then resize to fixed size
      $\Rightarrow$ trofeat box as square & has always the same percentage target in the input
      (crop based on each label box: NOT using previous label box)
    - cross-entropy loss for classification & smooth $L1$ loss for regression
    - pos-neg example: dual IoU threshold $(0, 0.3, 0.6)$
    - batch: size 64, with at most 16 positive examples
    - fine-tune only last 2 conv layers in AlexNet
    - dataset: image pair from imagenet VID & youtube-BB
  - Understanding
    - 2-frame model: tracking as one-shot detection (with response map of similarity)
      $\Rightarrow 1^{st}$ bbox as only examplar to learn the conv kernel in RPN
      (which is used in following frames for bbox proposal)
      $\Rightarrow$ Siamese net learns to efficiently map z to weights $[\varphi(z)]$ (learning to learn)
      $\Rightarrow$ has learned transformation of target, hence update template has mere improvement
    - template branch to extract feature discriminate fore-/back-ground
      $\Rightarrow$ predict detection kernel by input z and its weight in $[\varphi(\cdot)]$ as a meta learner
    - detection branch (RPN) to propose & refine a compact bbox (instead of online fine-tuning)
      $\Rightarrow$ no need for online learning, nor multi-scale test
    - meta-learning / oneshot detection: predict the last kernel for regression/classification (separately)
    - crop the kernel-branch: crop out edge case to avoid noise
      $\Rightarrow$ NOT able to handle large search region
      (use only limited center size in refinement)
    - yet, using both VID & youtube-bb dataset
      $\Rightarrow$ youtube-bb contain larger transform between frames
      (due to one labeled frame in every 30 frames)
      $\Rightarrow$ needs larger center-size & context to overcome large transformation
    - tracking based on <u>similarity & transformation & semantic</u>
      - similarity: from correlation
      - transformation: from using z-x pair from a large frame range (~ 100)
        (both template-search img are cropped by corresponding label box)
        (instead of box from previous frame as in Re^3)
        $\Rightarrow$ learning appearance transformation, instead of object motion
      - semantic: from using detection net (RPN)
        $\Rightarrow$ anchor for obejct different w-h ratio
    - fail in large & fast moving object, potentail reason:
      large search area contains too much noise & similarity get more response
    - fail with different image crop: training with only a specific crop strategy
      (trained with a fix 255-255 crop, with a fixed ratio of region being target object)
      $\Rightarrow$ overfit the chosen scale & NOT able to handle different scale
      $\Rightarrow$ not fully explore the discriminative ability from the semantic
    - may fall back to simple detector if backbone can not extract meaningful template
      (potential reason: not fully trained, training set do not have hard negative, ...)
      by meaningful: discriminative enough for two similar object
      (need to able to recognize tiny feature: similar to requirement in re-ID)

        - offline transformation + semantic learning to take the place of online learning
          ⇒ NOT using background info of each video
          (use only foreground info via similarity)

- DaSiamRPN: Distractor-aware siam network
    - Analysis
        - data imbalance in tracker training
        - lack of diversity in positive: too few category in tracking dataset
          ⇒ model can not generalize to real-generic object & bias to known object
        - most negative are sampled from background ⇒ no semantic
          ⇒ learn a simple fore-/back-ground classifier for objectness
          ⇒ easily drift to arbitrary object & high score even after target lost
    - Training
        - introduce detection dataset & use augmentation
          ⇒ more diverse classes in positive examples
        - use inter-class negative: semantic target from different class as negative
          ⇒ avoid drifting to other object
        - use intra-class: semantic target from same class as negative (diff track)
          ⇒ force tracker to be able to realize fine-grained feature
        - ⇒ improve intra-/inter-class discriminative ability & generalization ability
          ⇒ tracker able to produce meaningful objectness score
    - Inference
        - inference as siamRPN: $17 * 17 * 5$ proposal & apply non-max suppression
        - select the best-scored proposal z
        - collect distractors set $D = \{\forall d_i, f(d_i, z_t) > \text{thr} \wedge d_i \neq z_t\}$,
          where thr a pre-defined threshold, $f = \varphi(d_i) \star \varphi(z_t) + \mathbf{b}$
        - collect top-k scored proposal into P
        - select q from P s.t.

          $$q = \arg\max_{p_k \in P} f(z, p_k) - \hat{\alpha} \frac{\sum_{i=1}^{n} \alpha_i f(d_i, p_k)}{\sum_{i=1}^{n} \alpha_i}$$
          $$= \arg\max_{p_k \in P} (\varphi(z) - \hat{\alpha} \frac{\sum_{i=1}^{n} \alpha_i \varphi(d_i)}{\sum_{i=1}^{n} \alpha_i}) \star \varphi(p_k) \text{by xcorr being linear operation}$$

        - to online model distractors: incrementally learn the target templates
          (inspired by associative law)
          $\Rightarrow q_{T+1} = \arg\max_{p_k \in P_T} (\frac{\sum_{t=1}^{T} \beta_t \varphi(z_t)}{\sum_{t=1}^{T} \beta_t} - \hat{\alpha} \frac{\sum_{t=1}^{T} \beta_t \sum_{i=1}^{n} \alpha_i \varphi(d_{i,t})}{\sum_{t=1}^{T} \beta_t \sum_{i=1}^{n} \alpha_i}) \star \varphi(p_k)$, where
          $beta_t$ the learning rate at time t, set to be $\sum_{i=0}^{t-1} \frac{\eta}{1-\eta}$ with $\eta = 0.01$
          ⇒ model a online-updating classifier (by re-ranking proposals)
          (yet, also introduce a lot of hyper-params...)
    - Tracking
        - siamnet tracker
        - long-term tracking by re-detecting: use model objectness to denote target lost/recovery
          ⇒ larger search region if target lost (increase from 255 to 767)
    - Understanding

- using negative pair with semantic object ⇒ more discriminative
  ⇒ model realize difference between similar object (both inter- & intra- class)
  (instead of only foreground-background)
  
  (bird: siamFC vs siamRPN vs DaSiamRPN)
- ⇒ hence more robust & meaningful score
  ⇒ able to use score to denote lost-recovery in noisy & big search region
  (otherwise, may pick up random object to track with high score)
  ($\geq 10\%$ absolute improvement on most datasets...)

- C-RPN: Siamese Cascaded Region Proposal Network
  - Detector
    - modified AlexNet for encoding (multi-scale info utilized)
      ⇒ extracts info from search region x with gating
      (target template z already prepared)
      ⇒ $\phi_n(\cdot)$ for features at n^{th} layer, backwards
    - feature fusion $\Phi_l(\cdot) = f\left(\Phi_{l-1}, \phi_l\right)$, with $\Phi_1 = \phi_1$
      ⇒ recursively fuse semantic info with lower-level spatial infoz
    - $f(\Phi_{l-1}, \phi_l)$: feature transfer block for fusion
      ⇒ upsample Φ_{l-1} by deconv, further 2 convs on ϕ_l for channel matching
      ⇒ element-wise sum, then interpolation as downsampling
  - Tracker
    - 2-frame siamese net as tracker
  - Inference
    - same CNN extracts features from search region
    - l^{th} RPN convolute $\Phi_l(z)$ on $\Phi_l(x)$
      ⇒ regresses boxes based on response map & anchor boxes A_l
    - discard any boxes $\in A_l$ with confidence/objectness lower than a preset threshold
      ⇒ produce anchor boxes A_{l+1}
    - fuse semantic info with lower-level info, for both branch for x, z
    - $l+1^{\text{th}}$ RPN further refine anchors A_{l+1}
  - Loss
    - loss for l^{th} RPN $L_l = \sum_{a \in A_l} L_{\text{cls}}(c_a^l, \hat{c}_a^l) + \lambda \sum_{a \in A_l} \hat{c}_a^l \cdot L_{\text{loc}}(r_a^l, \hat{r}_a^l)$,
      where $c_a^l / \hat{c}_a^l$ the predict/label objectness for anchor a;
      with $r_a^l / \hat{r}_a^l$ the predict/label location for anchor a (encoded as YOLOv2)
      note: label based on current anchor $a \in A_l$ and ground truth $\hat{r}_a$
    - ⇒ total loss $L = \sum_l L_l$
  - Training
    - random sample image from a video
      ⇒ forming image pair (target template image, image with/without target)
  - Understanding
    - multi-stage tracking: each RPN sequentially refine bbox (size & location)

- hard negative mining by filtering out box proposal at each RPN stage
  ⇒ training samples sequentially more balanced
  ⇒ RPNs sequentially more discriminative
- ⇒ hence, more discriminative between similar nearby object
  (compare with Re^3 & SiamRPN)
- fusion of multi-level feature (spatial + semantic info) for RPN
  ⇒ provide detail for semantic similar distractor

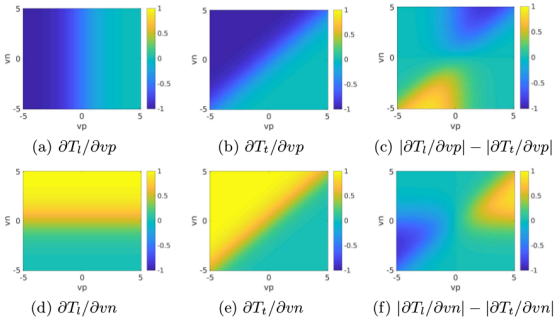
- Triplet Loss in Siamese Tracker
  - Tracker
    - two-frame siamese net as tracker
    - inference as siamese tracker: init & track
  - Loss
    - triplet: exemplar (template) z, positive instance x_i, negative instance x_j, where positive/negative instance the search image with/without target
    - matching prob: $prob(vp_i, vn_j) = \frac{e^{vp_i}}{e^{vp_i} + e^{vn_j}}$,
      where vp_i/vn_j the predicted prob of x_i/x_j to be pos-/neg-ative
      ⇒ a softmax over prediction on pos-neg instance (with different search img)
    - joint prob: $\frac{1}{MN}\prod_i^M \prod_j^N prob(vp_i, vn_j)$, where
      M/N the number of instance in the positive/negative set

      ⇒ triplet loss $L_t = -\frac{1}{MN}\sum_i^M \sum_j^N \log prob(vp_i, vn_j)$
      $= \frac{1}{MN}\sum_i^M \sum_j^N \log(1 + e^{vn_j - vp_i})$
  - Training
    - train another 10 epoch with the triplet loss after the original training
  - Analysis
    - scale the normal logistic loss (used due to binary classification) accordingly

      ⇒ $L_l = \sum_i^M \frac{1}{2M}\log(1 + e^{-vp_i}) + \sum_j^N \frac{1}{2N}\log(1 + vn_j)$
      $= \frac{1}{MN}\sum_i^M \sum_j^N \frac{1}{2}(\log(1 + e^{-vp_i}) + \log(1 + e^{vn_j}))$

      i.e. repeat N times for M positive instance; M times for N negative instance
    - thus, logistic v.s. triplet lies in their term inside sum
      i.e. $\underbrace{\frac{1}{2}(\log(1 + e^{-vp_i}) + \log(1 + e^{vn_j}))}_{T_l}$ v.s. $\underbrace{\log(1 + e^{vn_j - vp_i})}_{T_t}$
    - for logistic: $\frac{\partial T_l}{\partial vp} = \frac{-1}{2(1+e^{vp})}$, $\frac{\partial T_l}{\partial vn} = \frac{1}{2(1+e^{-vn})}$
    - for triplet: $\frac{\partial T_t}{\partial vp} = \frac{-1}{1+e^{vp-vn}}$, $\frac{\partial T_t}{\partial vn} = \frac{1}{1+e^{vp-vn}}$

- ⇒ gradient value:



(a) $\partial T_l/\partial vp$ (b) $\partial T_t/\partial vp$ (c) $|\partial T_l/\partial vp| - |\partial T_t/\partial vp|$

(d) $\partial T_l/\partial vn$ (e) $\partial T_t/\partial vn$ (f) $|\partial T_l/\partial vn| - |\partial T_t/\partial vn|$

⇒ triplet loss offers larger absolute gradient when $vp < vn$ (up-left triangular)
⇒ logistic loss: focus on making vp larger & vn smaller, separately
($\frac{\partial T_l}{\partial vp} \to 0$ when $vp \to +\infty$, similar for vn)
⇒ while, triplet loss: focus on having $vp > vn$ (to ensure the correct decision)

- Understanding
  - mine the relationship between exemplar-positive-negative
    ⇒ larger absolute gradient under wrong classification (predict $vp \leq vn$)
    ⇒ focus on correct prediction, instead of only high score
  - form more training examples: num of examples increases from $M + N$ to MN
    ⇒ more importantly, form a more diverse dataset
  - all model trained with triplet loss outperform its original logistic loss version
    (compared with training extra 10 epoch with logistic loss)
    (though not outperforming in every scenario, e.g. low resolution)

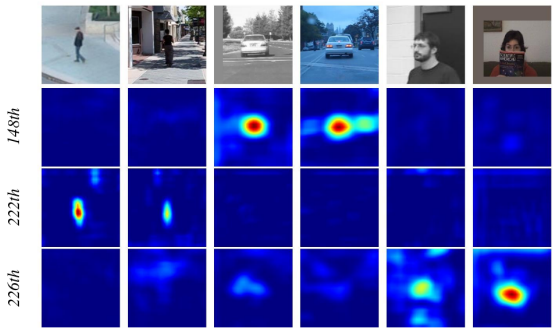
- Learning Discriminative Model Prediction for Tracking
  -

- SiamRPN++
  - Detector
    - deep backbone ⇒ multi-level feature with enough detail-semantic diversity
    - depth-wise separable correlation: xcorr on each corresponding channel-channel
      (similar to depthwise separable conv: 1x1 conv to replace summation)
      ⇒ each semantic info measure similarity separately (more semantic)
      ⇒ meanwhile, reduce parameters and thus faster convergence
    - neck: conv-bn block to adjust & align channel before xcorr
      (separate neck for classification - regression)
    - multi-RPN: corresponding to multi-level feature
      ⇒ depthwise xcorr applied on each level feature & each followed by an RPN
      ⇒ (trainable weight) weighted average directly on RPN result
  - Tracker
    - siamese RPN net tracker, with deeper backbone (e.g. resnet, rather than alexnet)
  - Analysis
    - siamese tracker: correlation to measure similarity
      ⇒ $f(\mathbf{z}, \mathbf{x}) = \phi(\mathbf{z}) * \phi(\mathbf{x}) + b$, where $\phi(\cdot)$ map into embedded space, b the bias
    - ⇒ need to have strict translation INvariance: $f(\mathbf{z}, \mathbf{x}[\Delta\tau_j]) = f(\mathbf{z}, \mathbf{x})[\Delta\tau_j]$, where $[\Delta\tau_j]$ the translation, i.e. shifting sub-window
      (i.e. conv kernel canNOT infer the input location by inspecting its feature)
    - ⇒ need to have structure symmetry: $f(\mathbf{z}, \mathbf{x}) = f(\mathbf{x}, \mathbf{z})$

- YET, deep net involves too much 0-padding, which creates unique response
  ⇒ conv kernel can recognize if its input is from center-border-corner
  ⇒ 0-padding introduce border signal & destroy the translation invariance
  (as network can now realize the input location from its content)
  (e.g. if/how the content is influenced by 0-padding)
  ⇒ introduce central bias (from in-model padding)
  (as most targets labeled in center - a wrong cue learned for objectness)
  ⇒ trained with shift augmentation
  (random shifting the crop region s.t. target not right in the center)
  ⇒ ensure network realize: border signal is UNreliable cue for objectness
  ⇒ destroy the central bias
- 0-padding in template & only border of search img get 0 padded
  ⇒ template with 0-padding hard to match the center of search img
  (as only the border get padded)
  ⇒ strong, yet wrong, signal for network to realize location
  ⇒ central bias again! (introduced by xcorr)
  ⇒ crop out padding-influenced part from template
  (e.g. use only the central 7x7 of template branch as template)
- 0-padding in preparing network input (input image patch)
  ⇒ may get network realize location again (from outside-model op)
  ⇒ use the image average (channel-wise) instead of 0 to pad

◦ Understanding

- depthwise xcorr with deepnet: tracking with semantic - similarity
  ⇒ extract useful semantic for similarity
  ⇒ each channel captures some specific semantic (enforced by depthwise xcorr)

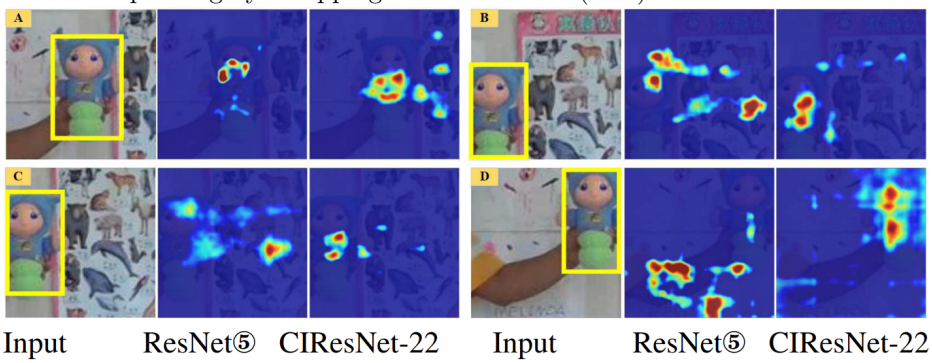


  (response of $xcorr(\phi(\mathbf{z}), \phi(\mathbf{x}))$ at different channel, only search image shown)
  ⇒ better tracking result in hard case (e.g. under high IoU threshold)
- depthwise xcorr: less params ⇒ more balanced params number between
  · template-search branch: avoid overfit/underfit template, as search branch are easier to be trained
  · backbone-rpn: more stable
  ⇒ more stable training procedure
- deep tracker can learn central bias, due to padding & target-centered cropping
  ⇒ network learn to locate target by wrong evidence (input patch location)
  (as padding introduces unique response on the border & corner)
  ⇒ shift augmentation & template central crop & average padding
  ⇒ force network to be location invariance & ignore response from padding
- RPN use supervision more than similarity
  ⇒ need asymmetric part to map similarity to RPN regression-classification task
  ⇒ separate xcorr for different class

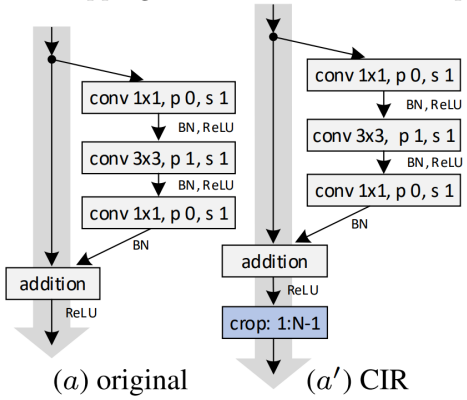
- multi-level feature may be meaningful only given enough detail-semantic diversity
  ⇒ only meaningful with deep-enough net

- SiamDW: Deeper and Wider Siam
  - Analysis
    - padding destroy the translation invariance
      (can know if a feature comes from corner-border-center by analyzing its content)
      ⇒ as all target labeled in center, thus central bias
    - padding results in INconsistent template-search feature map
      - template feature extracted with padding
      - search feature map has no such padding in non-border area

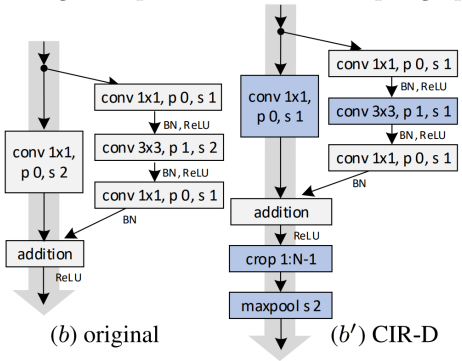
      ⇒ xcorr canNOT effectively measure similarity
    - ⇒ remove padding by "cropping-inside residual" (CIR) unit



  - ⇒ CIR Unit (to remove padding influence)
    - add cropping after addition ⇒ remove padding influenced feature

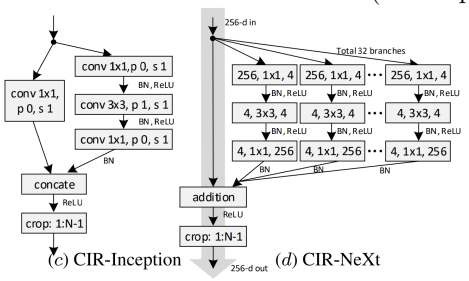


    - change the position of downsampling ops



⇒ use conv with stride 1 (instead of 2) & add 2x2 max pool after addition-crop
⇒ remove padding influence & considering info at the border (by max pool)
(directly crop after stride-2 conv: miss out potential strong response at the edge)
(has been also empirically proven direct-cropping has worse result)

- increase the width of each unit (as inception, resXnet)



- Backbone
  - modified Resnet with stacked CIR unit & downsampled by CIR-D unit
    ⇒ with CIResNet-22 being the best-performance net
    (CIResNet-43 perform almost the worst, due to in less weight CIResNet-43)
    really? less wright, yes, but being deeper not help?
- Training
  - gradually finetune the network from back to front
    (e.g. unfreeze a unit every 5 epoch)
- Tracking
  - siam net as tracker & tracking by detection
- Understanding
  - padding leads to central bias & degraded similarity measurement
    ⇒ cropping to remove padding influence
    (why not, crop only before xcorr & logit feature map to speedup?)
  - ensure receptive field to be 60% ∼ 80% size of the template image
    (empirical setting to have best performance for siam tracker)
    ⇒ as larger RF includes more context & thus insensitive to spatial location

Multi-Object Tracking & Data Association

- Gated Association
  - Procedure
    - given prediction, its variance and detection noise, filter out an interested area
    - consider only detection inside the interested area (satisfying requirements)
    - score each detection & associate detection (detection result) with tracker
      ⇒ which detection belongs to which trajectory
  - Global Nearest Neighbor
    - choose the best / most probable / nearest
      (under the constraint that an detection can associate with at most one track) ⇒ assume one detection is produced by single object
    - require accurate and sparse detection, with few false alarm ⇒ sensible to noise (easily fail in crowded scene)
  - Nearest Neighbor
    - choose the best / most probable / nearest (thought one detection may be used by multiple tracks)

    - Understanding
        -

- Joint Probability Data Association (JPDA)
    - Procedure
    - Understanding
        - probabilistic perspective for prediction-detection relation
          $\Rightarrow$ cooperate with uncertain association: weight all detections by probability
        - hence, crowded detections tends to pull multiple tracks together
          $\Rightarrow$ coalescence problem

- Multiple Hypothesis Tracking
    - Assumption
        - Gaussian models for target dynamics and noise in detection
        - uniform distribution for false alarm (false-positive detection)
    - Track Hypothesis
        - association result given track initiation, prediction and detection
          $\Rightarrow$ a sequence of selected detection
        - compatibility: tracks are compatible, if they do NOT share any detection
          $\Rightarrow$ any track update using the same detection are INcompatible
    - Track Tree (Clustering)
        - use incompatibility as edge, track as vertice, sort tracks by time
          $\Rightarrow$ each connected tree becomes a cluster (track family)
          (the tree level denotes time sequence)
        - each tree shares a common root node (the initial detection)
        - growing: whenever a new detection can be accounted for a track hypothesis (node)
          $\Rightarrow$ the node generates 2 children nodes (tracks): update / not update
    - Global Hypothesis
        - a global hypothesis contains only compatible track(s)
          $\Rightarrow$ the collection of track, with ≤ 1 track from each tree/family
    - Track Score
        - posterior ratio $r = \frac{p(D|T)p(T)}{p(D|F)p(F)} \triangleq \frac{p_T}{p_F}$,
          where $p(D|T), p(D|F)$ the likelihood given detection is true, false alarm
          with D the detections in current track
          $\Rightarrow$ log ratio $lr = \ln \frac{p_T}{p_F}$
        - use log ratio as score, at time $t, L(t) = L(t-1) + \Delta L(t)$,
          where $\Delta L(t) = \begin{cases} \ln(1 - \hat{P}(D)) & \text{no update} \\ \Delta L_u(t) & \text{update} \end{cases}$,
          with $\hat{P}(D)$ the expected probability of detection;
          and $\Delta L_u(t)$ the residual error between prediction and detection
          ($\Delta L_u(t)$ may include covariance, density, $\hat{P}_D$, etc.)
    - Global Hypothesis Score
        - $s_H = \sum_{k \in K} L_k(t)$,
          where K_H all (compatible) tracks in hypothesis H, L_k the score for track k
    - Global Hypothesis Probability

        - computed from hypothesis score (a maximum weighted independent set problem)
    - Track Probability
        - the sum of probability of all hypothesis that contains the track
    - N-scan Pruning
        - given detection at time t, eliminate IMplausible tracks originated at $t - N$
          $\Rightarrow$ suppress tree from exponentially growing;
        - N the time step buffer before decision (scan = time), usually $N \geq 5$
    - Understanding
        - probabilistic perspective towards the result of decisions for data association
          (a larger scope than JPDA)
        - defer critical decision into the future
          $\Rightarrow$ make decisions for the past after their observation available
        - model track alternatives, each with a probability, by track tree and hypothesis
          (for all possible tracks, model joint prob over all detections in a track)
        - model global joint probability of all tracks, by global hypothesis
        - similar to DP-longest substring: maintain a set of candidates
        - essentially, a bread-first search $\Rightarrow$ real-time ability constrained by tree size

- Maximum Net Flow
    - Input
        - detection
    - Output
        - association result
    - Inference
        - detection as node, possible association as edge
          $\Rightarrow$ construct a graph, with detection time as layer
        - $\Rightarrow$ solve as maximum flow / minimal cost
    - Understanding
        - another global optimization, vs. probabilistic perspective in MHT

- Inversed Reinforced Learning for Data Association with Markov Decision Process
    - Input
        - current detected bounding boxes, with objectness
        - previously predicted bounding boxes
    - Output
        - decision of data association between track & detection
    - Markov Decision Process (MDP) for Track Management
        - states: active, tracked, lost, inactive
          $\Rightarrow$ model the state of a track
        - probability at each state given by trained model for each state
    - Inference
        - binary classification for track start (given a detection)
        - optic flow to track & association
          (rule model to decide if lost)

        - regression model to associate lost track & current detection
          (measure similarity)
        - rule for track death: lost for consecutive 6 frames
    - Training: Inversed Reinforcement Learning for Lost Recovery
        - classifier for track start: trained offline
        - regressor for lost track association: trained only when MDP make wrong decision
          (similar to hard-example mining?)
    - Tracking ‘
        - one MDP for a track in multi-object tracking
          $\Rightarrow$ multiple tracks may update with same detection (need to tune optic flow)
    - Understanding
        - explicit expression for track state
          $\Rightarrow$ can design state for hard scenario
          $\Rightarrow$ enable explicit control over optimization
        - ugly crashed model for each state $\Rightarrow$ can be all unified to NN(s)

- Siamese CNN for Association
    - Input
        - image pair (I_1, I_2), with optic flow $I_1 \rightarrow I_2$ as (O_1, O_2)
          $\Rightarrow D = [I_1, I_2, O_1, O_2]$ (resized & channel concat)
    - Output
        - probability of data association between two detections
    - Inference & Structure
        - 3 conv layers, max pooling, 4 dense layers
        - examine spatio-temporal info: position change & relative velocity by difference
        - NN feature vector concat with handcraft feature
        - feed into gradient boosting classifier (with 400 trees)
          $\Rightarrow$ output as binary classification of (match, no match)
    - Training
        - true positive: associated ground truth detection in 2 frames
          (time gap $\leq$ 15 frames)
        - negative: wrong association to true detection of other track / false detection
        - data augmentation: false alarm, distortion on image
    - Tracking
        - given current data association probability for all track-detection pair
          $\Rightarrow$ construct a linear program problem (with constraints) $\Rightarrow$ a global optimization
          for association given probability
        - online tracker (e.g. kalman filter)
    - Understanding
        - NN approach for association probability
          (with fusion of NN & rule-model via GB classifier)
          $\Rightarrow$ fusion much better than pure NN $\Rightarrow$ spatio-temporal info important

- Online Multi-Target Tracking Using RNN
    - Input
        - D the dimension of bbox encoding (e.g. x, y, w, h, objectiveness, etc.)

- $x_t \in \mathbb{R}^{ND}$ all the N track state (bbox) at time t
- $x_t^\star \in \mathbb{R}^{ND}$ all the N predicted bboxes for time t, from time $t-1$
- $z_t \in \mathbb{R}^{MD}$ all the $M-1$ detected bbox at time t, with an empty detection
- $\varepsilon_t \in (0,1)^N \in \mathbb{R}^N$ the existence probability (liveness) for all tracks
- $A_t \in \mathbb{R}_{N \times M}$ the probability matrix for data association between track-detection
- h_t hidden state of track RNN at time t
- $C_t \in \mathbb{R}_{N \times M}$ the distance matrix between $x_t^\star$ and z_t
  i.e. $C_t[i,j] = dist(x_t^\star[i] - z_t[j])$
- available ground truth: $\widetilde{x}_t, \widetilde{A}_t, \widetilde{\varepsilon}_t$

◦ Inference

- given x_t, h_t, track RNN outputs state prediction $x_{t+1}^\star$, compute h_{t+1}
- based on pred $x_{t+1}^\star$ and detection z_{t+1}, compute C_{t+1}
- given C_{t+1} and hidden state for i^{th} track $h_{t+1}[i]$
  $\Rightarrow$ association LSTM scans over all detection z_{t+1}
  $\Rightarrow$ regress $A_{t+1}[i,:]$, the association prob for i^{th} track and each bbox in z_{t+1}
  (as part of track RNN process)
- given detection z_{t+1}, association prob A_{t+1}, latest liveness ε_t, with h_{t+1}
  $\Rightarrow$ update state to be x_{t+1}, estimate liveness ε_{t+1}

◦ Structure

- track RNN consists of a 2-layer association LSTM

◦ Loss

- prediction $L_{\text{pred}} = \frac{\lambda}{ND} \sum (x_{t+1}^\star - \widetilde{x}_{t+1})^2$
- updated state $L_{\text{update}} = \frac{\kappa}{ND} \sum (x_t - \widetilde{x}_{t+1})^2$
- liveness $L_\varepsilon = \widetilde{\varepsilon}_t \log \varepsilon_t + (1 - \widetilde{\varepsilon}_t) \log(1 - \varepsilon_t) + |\varepsilon_t - \varepsilon_{t-1}|$
  $\Rightarrow$ minimize the diff between consecutive liveness estimation $\Rightarrow$ smoothness
  (prevent track from termination for only a single detection lost)
- association $L_a = -\log(A_{t+1}[i, \widetilde{j}])$, where $\widetilde{j}$ the true association for i^{th} track

◦ Training

- data augmentation: sample synthetic trajectories from each labeled video
  (Gaussian distribution)

◦ Tracking

- forward unroll track RNN, if liveness ≤ 0.6, corresponding track ignored
- liveness ≥ 0.6 again, a new track initiated

◦ Understanding

- specialized RNN cell accounting for prediction, update, birth-death of all tracks
- another LSTM cell designed for data association
  $\Rightarrow$ able to learn $1-1$ association by scanning
  (yet, unnecessary, since N, M fixed i.e. a fixed size mapping)
- utilize given detector $\Rightarrow$ no appearance model (but only location & size)
- able to maintain at most N track with maximally M detection at a time

• Collaborative Deep Reinforcement Learning for MOT

◦ Notation

- I_t the t^{th} image frame
- $b_{i,t}^\star$ a bbox for i^{th} ground truth object p_i at frame t
- $B_{i,t}^\star$ a set of bboxes sampled around $b_{t,i}^\star$

- $g(a, b)$ cal the IoU between bbox a, b
- $p_{i,t} = \{b, f\}$, the i^{th} detected object at t,
  where $b = \{x, y, w, h\}$ the bbox; f the appearance model
  $\Rightarrow$ distance $d(p_1, p_2) = \alpha(1 - g(b_1, b_2)) + (1 - \underbrace{\frac{f_1^T f_2}{\|f_1\|\|f_2\|}}_{\text{(cos dist)}})$
- $H = \{b_1, ..., b_t\}$ the history trajectory of an object
  $\Rightarrow H^K = \{b_{t-K+1}, ..., b_t\}$ the history of past K frames
- an agent for each object $g = \{H, p\}$
- detections as environment $\hat{P}_t = \{\hat{p}_1, ..., \hat{p}_{n_t}\}$
- state at frame $t, s_t = \{G_t, \hat{P}_t\}$, where $A_t = \{g_1, ..., g_m\}$
- set of actions $\mathcal{A} = \{\text{update}, \text{ignore}, \text{block}, \text{delete}\}$

◦ Prediction Net Inference

- crop the frame $t + 1$ at the location of estimated bbox $b_{i,t}$ (of frame t)
- 3 conv layers, then dense, then concat with $H^{K=10}$ (fuse with temporal info)
- 2 dense layers to regress $b_{i,t+1}$, the bbox for object i of frame $t + 1$

◦ Prediction Net Training

- regression loss $L = \sum_{i,t} \sum_{b \in B_{i,t}} g(b^\star_{i,\mathbf{t+1}}, \phi(I_t, b, H_i^{K=10}))$,
  where I_t the image frame at time t, ϕ the mapping of pred net

◦ Action

- update: $f_{t+1} = (1 - \rho_f) \cdot f_t + \rho_f \cdot \hat{f}_{t+1}$, where $\hat{f}_{t+1} \in$ selected detection $\hat{p}_{t+1}$;
  $b_{t+1} = (1 - \rho_b) \cdot b_t + \rho_b \cdot b'_{t+1}$, where b'_{t+1} predicted position;
  $(\rho_f, \rho_b$ pre-selected)
- ignore: no detection suitable, use only prediction for update $(\rho_f = 0, \rho_b = 1)$
- block: same as ignore, no detection due to occlusion
- delete: remove the agent

◦ Reward

- for agent g at time t, with pred and ground-truth box $b'_{t+1}, b^\star_{t+1}$
- reward for agent g at time $t : r^\star_t = r_t + \beta r_{j,t}$, where
  r_t for itself; $r_{j,t}$ for its nearest neighbor, β a balance factor
  $(r_t, r_{j,t}$ calculated in the same manner)
  $\Rightarrow$ agents need to collaborate for better reward
- for action $a \in \{\text{update}, \text{ignore}, \text{block}\}$
  $\Rightarrow r_t = \begin{cases} 1 & \text{if } IoU \geq 0.7 \\ 0 & \text{if } 0.5 \leq IoU \leq 0.7 \text{ } (IoU \text{ calculated between } b'_{t+1} \text{ and } b^\star_{t+1}) \\ -1 & \text{otherwise} \end{cases}$
  for action $a =$ delete $\Rightarrow r_t = 1$ if object disappear; else -1
- $\Rightarrow Q(s_t, a_t) = r^\star_t + \gamma r^\star_{t+1} + \gamma^2 r^\star_{t+2} + \cdots$, where γ decaying param

◦ Decision Net Inference

- for each $g \in G_t$ with its current & predicted location b_t, b'_{t+1}
- select a neighbor agent $g_j \in G_t - \{g\}$, that is nearest to b_t
- select the detection $\hat{p} \in \hat{P}_{t+1}$ that is nearest to b'_{t+1}
- 3 feature maps $(p \in g, p_j \in g_j, \hat{p})$, flatten as 1-D vector input
  $\Rightarrow 3\times$ dense layer to output prob over actions $\pi(a|s, \theta)$, where θ the weights

◦ Decision Net Training

- goal $\arg\max_\theta L(\theta) = \mathbb{E}_{s,a} \log(\pi(a|s,\theta)) \cdot Q(s,a)$

$$\Rightarrow \frac{\partial}{\partial\theta} L = \mathbb{E}_{s,a} \frac{\partial}{\partial\theta} [\log(\pi(a|s,\theta)) \cdot Q(s,a)]$$
$$= \mathbb{E}_{s,a} [\frac{Q(s,a)}{\pi(a|s,\theta)} \cdot \frac{\partial}{\partial\theta} \pi(a|s,\theta)]$$

  $\Rightarrow$ increase probability for actions with $Q > 0$; decrease for those with $Q < 0$
- to speed up converge: value for state $s, V(s) = \frac{\sum_a p(a|s) Q(s,a)}{\sum_a p(a|s)}$
  $\Rightarrow$ advantage value $A(s,a) = Q(s,a) - V(s)$
  (in case all $Q > 0$, or all $Q < 0$ at the beginning: use expectation as zero-line)
- policy gradient as $L(\theta) = \mathbb{E}_{s,a} \log(\pi(a|s.\theta)) A(s,a)$
- pre-training: set $\gamma, \beta = 0$ before searching hyper-parameter β, γ
- training set: detection bbox = $\{\hat{b} \in \text{detection } \hat{P} |\ \text{IoU}^{\hat{b}}_{\text{label}} > 0.5\} + \{\text{label } b^\star\}$

◦ Tracking

- initiate a track for each initial detection
- for each agent, predict its location at $t+1$, by pred net
- for each agent g, select its closest detection $\hat{p} \in \hat{P}_{t+1}$, $p_j \in$ closest agent g_j
  $\Rightarrow$ decision net: $(p \in g, \hat{p}, p_j) \rightarrow$ action $\mathcal{A}$
- track terminates when decision net decides to "delete"

◦ Understanding

- prediction net as a detector to give more precise location
  (an implicit self-trained detector)
- decision net to eliminate false alarm & combine prediction
  $\Rightarrow$ robust to different detector/predictor
- decision net trained to collaboratively maximize utility
  $\Rightarrow$ mitigate the false negative
  $\Rightarrow$ introduce distractor in training for better discriminative ability
- may trapped in false appearance feature $\Rightarrow$ ID switch
  (e.g. blue box handed from one person to another person)

Instance Tracking

• SiamMask: Fast Online Object Tracking and Segmentation : A Unifying Approach

◦ Input

- initial bbox z as target template at first frame (exemplar)
- cropped search region x_t, centered at target location of $t-1$

◦ Output

- response map for object localization
- bbox regression for target (with resizing & rotation)
- binary segmentation mask i.e. pixel $\in$ (target, not target)

◦ Inference

- given z, x_t backbone CNN extract feature as $f(z), f(x_t)$
- cross-correlation response map $g(z, x_t) = f(z) \star f(x_t)$,
  where $f(z)$ used as kernel
  $\Rightarrow$ g as response of candidate window (RoW)
  $\Rightarrow$ encode similarity between z and each candidate window/bbox in $f(x_t)$

- Structure
  - backbone CNN ResNet-50 as feature extractor, with adjustment (dilated conv for better resolution)
  - cross-relation between two extracted feature maps ⇒ response map, each location an RoW
  - bbox regression: 2×(1-by-1 conv) for k anchors at each RoW
  - bbox classification: 2×(1-by-1 conv) for score of k anchors at each RoW
  - segmentation: 2×(1-by-1 conv) + upsampling with skip + per-pixel sigmoid ⇒ upsample into a mask for each RoW location
- Training
  - loss $L = \lambda_1 L_{\text{mask}} + \lambda_2 L_{\text{score}} + \lambda_3 L_{\text{box}}$ (L_{mask}, L_{box} considered only at location for ground truth)
  - trained different branch using corresponding dataset
  - data augmentation: random jitter, translation, rescaling
- Tracking
  - one-step update, may use mask to produce
  - multi-object: multiple initialization, each with a net as tracker
- Understanding
  - multi-task training improves all branches (semi-supervised video object segmentation, bbox tracking)
  - mask branch: output the mask in the context of x for that obj in z
  - simple starting point & fast speed for video object segmentation (compared to 0.1FPS)
  - more descriptive representation for tracked object ⇒ more detail in mask

12.5.4 Instance Segmentation

Problem Formulation

- Input Data
  - images
- Goal
  - obtain pixel mask for each instance
- Challenge
  - Within-Category Overlap
    - need to realize the boundary of multiple same-class instances
  - Semantic Segmentation + Detection
    -
- Evaluation Metrics
  - AP IoU
    - use mask IoU to classify an pred mask to be true/false positive
    - calc the average precision accordingly

Instance-first

- Mask R-CNN
  - Inference
    - backbone extract feature map
    - existing cls-reg head produce bbox detection & NMS
      ⇒ selecting out 100 highest-score boxes
      (apply mask head after NMS for less overhead)
    - for each corresponding RoI, mask head produce K mask, each of size $m \times m$,
      where K the num of classes
    - select the k^{th} mask, with k the predicted class of the RoI
    - resize mask into predicted bbox size
  - Structure
    - ResNet-FPN backbone
    - based on two-stage (Faster R-CNN) framework
    - RoI align (instead of RoI pooling) to avoid quantization
    - mask head: a small FCN to produce K binary masks via per-pixel sigmoid
  - Training
    - $L = L_{\text{cls}} + L_{\text{reg}} + L_{\text{mask}}$, where $L_{\text{cls}}, L_{\text{reg}}$ same as detection
    - L_{mask}: an average binary cross-entropy on one of the K mask
      ⇒ define only on k^{th} mask of a positive RoI
      (only consider the part of label mask inside the pred box)
    - ensure pos-net ratio = 1:3
    - joint training: still, stop gradient from RoI Align to RPN proposals
  - Understanding
    - RoI Align to remove quantization
      ⇒ remove mis-alignment between RoI location & extracted feature
      ⇒ preserve exact localization
        i.e. per-pixel spatial correspondence
    - ⇒ able to use large-stride network (e.g. 32-stride)
      (as quantization leads to larger misalignment with larger network stride)
    - ⇒ significant improvement in AP & bigger gain in AP at higher IoU
    - decouple mask & class pred
      · per-pixel sigmoid to produce each mask for each class v.s. per-pixel softmax
      · binary cross-entropy loss v.s. multinomial cross-entropy loss
      (could be further a class-agnostic mask pred of m^2 size)
      ⇒ avoid class competition when generating mask
      ⇒ much easier to learn
    - FCN as mask head: utilize pixel-to-pixel correspondence of conv ops
      (instead of dense layers) ⇒ more accurate
    - generalization ability: extend to keypoint detection
      ⇒ each mask produce a key point (spatial softmax over each mask)
    - able to multi-tasking all three tasks: detection, instance seg & key-point
      (improve on all tasks)
    - still, introduce $\sim 20\%$ overhead, even after box selection
      (due to heavy mask head)

Segmentation-first

12.5.5 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 $P \in$ identity I; an negative image $N \notin$ 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 ⇒ 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

- Encoding + 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
        - pre-compute the encoding of Siamese net
          ⇒ enable flexible deployment (device performs only bi-classification)

12.5.6 Stereo Vision

Problem Formulation

- Input
    - Visual Perception
        - image from mono-camera
        - images pair from stereo-/multi-cameras
        - video sequence from mono-/stereo-cameras
- Goal
    - Depth Perception
        - directly output a depth map
        - output a disparity map as indirect depth perception
- Approach Understanding
    - Unsupervised Learning
        - f : input $\rightarrow$ output hard & f^{-1} : output $\rightarrow$ input easy
          ⇒ analog to P-NP problem
          (easy to check/describe the error solution & hard to find solution)
        - designed error should be highly correlated with desired prediction error

Unsupervised Learning

- Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue
  - Understanding
    - avoid systematic error in collected image-disparity dataset
      ⇒ use deep net to jointly deal with motion blur, etc.
- Unsupervised Monocular Depth Estimation with Left-Right Consistency
  - Understanding
    - epipolar constraint with image reconstruction loss
      ⇒ unsupervised depth from mono-camera
      ⇒ avoid stereo-/multi-camera in practice use & avoid pixel-level labeling

12.5.7 Recognition at a Distance

Problem Formulation

- Input Data
  - Video
    - hot standby camera with stationary & active vision
    - stationary vision: wide field of view with low resolution for detection
    - active vision: narrow field of view with high resolution for detail analysis
- Goal
  - Recognition in the Wild
    - large coverage areas: > 100m range
      ⇒ scale beyond the theoretical analysis and practical design advice
    - with NO subject cooperation
- Challenge
  - None Cooperative Subject
    - not cooperating, may even be evading the system
  - Resolution
    - low resolution due to the very long object distance
    - restricted by lens resolution, which is then restricted by price
    - trade-off between wideness (coverage) & depth (zooming)
  - Illumination
    - dynamic illumination in the wild scene
    - maximum light intensity restricted by aperture size
      (given a fixed exposure time)
  - Distortion and Blur
    - amplified noise, due to:
      low brightness ⇒ low signal-to-noise ratio ⇒ ISO amplification
    - motion blur (if trade-off between ISO & exposure time)
    - blur from sensor tilt, as a hot standby system
    - fog, haze & atmosphere blur for very long distance recognition
  - Pose

        - the view angle due the camera position
          (usually overhead for less occlusion ⇒ downward tilt)
    - Multi-Object
        - schedule the limited high-resolution vision resource for multiple candidates
          ⇒ time window prediction, scheduling & resource allocation
    - Physical Coupling
        - expensive field test (mitigated by virtual environment)

- Application
    - Watch-list Recognition
        - an alert when person of interest appear/approach
    - Re-recognition
        - cross-camera tracking, long-range persistent tracking
    - Logging
        - catalog best recognized feature (e.g. face) for each person entering a region
        - marketing: understand customer activities and behavior

Stationary Vision

-

3D Imaging

- ◦

12.5.8 Re-Identification

Problem Formulation

- Input Data
    - Images
        - captured by camera networks across multiple areas
        - may assume to be a crop over interested target
          (e.g. a bounding box crop, instead of the whole image)
        - a single image or a images sequence of target
- Goal
    - Identification
        - retrieve images from gallery/database that has same identification as input
    - Ranking
        - list out the most probable ID, to ensure recall
          (e.g. in security scenario)
- Challenge
    - Varying Appearance
        - due to different view point, pose, background change (indoor v.s. outdoor), etc.
          ⇒ hard to realize inter-class similarity & intra-class difference
    - Limited Data

        - small dataset, hard to learn to use rare feature
          (though rare feature should be a strong identification sign)

- Metrics
    - Rank-1 Accuracy
        - measure the match of ID with highest predict prob
    - mean Average Precision
        - measure the accuracy of match of whole ranking
          $\Rightarrow$ account for the hard case (same ID can be multi-viewed in camera net)

- Trends
    - Local & Distributed Representation
        - feature on human body with human body model (e.g. skeleton model) as a prior
        - region proposal for different local area on human body
        - salient partitions with attention
    - Multi-scale Representation
        - concat local feature, features at multi-scales
        - spatial attention
    - Surpassing Human Performance

Re-Ranking

- Re-ranking Person Re-identification with k-reciprocal Encoding
    - Analysis & Derivation
        - given probe p, retrieved image $g \in$ gallery G, $|G| = N$
          $\Rightarrow N(p,k) = \{g_1, ..., g_k\}$ the k-nearest neighbors of p, where p the probe image
          (i.e. top-k sample in the ranking result)
        - yet, can often introduce false matches (noise)
        - $\Rightarrow$ require $p \in \text{k\_near}(g) \&\& g \in \text{k\_near}(p)$ (similar to the thought in re-projection error)
          i.e. both rank top-k when the other image is taken as probe
          (hence, reciprocal)
        - $\Rightarrow$ the k-nearest reciprocal neighbors $R(p,k) = \{g_i \mid g_i \in N(p,k) p \in N(g_i,k)\}$
        - yet, can also have positive image excluded

          $\Rightarrow \forall q \in R(p,k), R^* \leftarrow R(p,k) \cup R(q, \frac{1}{2}k)$

          $$\text{s.t. } |R(p,k) \cap R(q, \tfrac{1}{2}k)| \geq \frac{2}{3}|R(q, \tfrac{1}{2}k)|$$

          i.e. expand the set by considering the k-reciprocal neighbors of the set member
          (expansion confirmed by voting)
    - Distance Derivation
        - Jaccard distance

          $$\begin{aligned} d_J(p, g_i) &= 1 - \text{IoU}^{R^*(p,k)}_{R^*(g_i,k)} \\ &= 1 - \frac{|R^*(p,k) \cap R^*(g_i,k)|}{|R^*(p,k) \cup R^*(g_i,k)|} \end{aligned}$$

- to speed-up: encoding R^* into vector
  $\Rightarrow V_p = [V_{p,g_1}, ..., V_{p,g_N}]$,
  where $V_{p,g_i} = 1$ if $g_i \in R^*(p,k)$, else 0
- to consider original distance measurement (weighting image)
  $\Rightarrow V_p = [V_{p,g_1}, ..., V_{p,g_N}]$,
  where $V_{p,g_i} = e^{-d(p,g_i)}$ if $g_i \in R^*(p,k)$, else 0, with $d(p,g_i)$ the original distance
- $\Rightarrow |R^*(p,k) \cap R^*(g_i,k)| = \|\min(V_p, V_{g_i})\|_1$
  $|R^*(p,k) \cup R^*(g_i,k)| = \|\max(V_p.V_{g_i})\|$,

  where $\min, \max$ being element-wise
  (i.e. element-wise logical and & count the num of resulting true)
- $\Rightarrow d_J(p,g_i) = 1 - \dfrac{\sum_{j=1}^{N} \min(V_{p,g_j}, V_{g_i,g_j})}{\sum_{j=1}^{N} \max(V_{p,g_j,V_{g_i,g_j}})}$
- accounting for original distance (which contains context info)
  $\Rightarrow d^*(p,g_i) = (1-\lambda) d_J(p,g_i) + \lambda d(p,g_i)$, with $\lambda \in [0,1]$

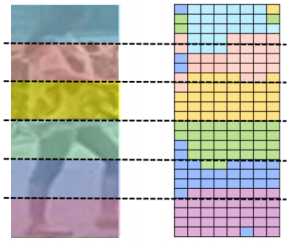
- Understanding
  - need to have ≥ 1 positive in the gallery to better improve result
    (yet, does not harm even in a single-shot setting, e.g. CUHK03)
  - a fully automated & unsupervised approach
  - significantly improve mAP & improve rank-1 accuracy, after re-ranking
    (verified with various model: robust to various distance metrics)
    (better than k-nearest neighbors as expected)
  - $k = 20$ in most dataset to avoid including false matches with too large k
  - $\lambda = 0.3$ (empirically) to further boost performance
    (original distance is also important in re-ranking)

People Re-ID

- PCB-RPP: Person Retrieval with Refined Part Pooling
  - Inference
    - CNN extracts feature map, vertically "soft" partitioned into p parts
    - each of p branch independently performs average pooling, processing & output
    - p output concat into together to measure overall similarity
  - PCB (Part-based Convolution Baseline) Structure
    - ResNet-50 as backbone, before the global average pooling
      (last downsampling removed $\Rightarrow$ richer feature granularity)
    - vertically partitioned into p parts, average pooling on each part
      $\Rightarrow$ part average pooling (centroid representation as part-level feature)
    - 1x1 conv over p vec to extract channel-wise info & dimension reduction
      (sharing weights between parts here)
    - each vec a separate dense layer for output
      $\Rightarrow p$ vector, each able to independently measure similarity / predict ID
  - RPP (Refined Part Pooling) Structure
    - observe within-part INconsistency in hard-uniform partition
      $\Rightarrow$ violate part-level feature assumption
      (violate: feature vec in same part are similar & dissimilar to vec in other parts)
    - $\Rightarrow$ a softmax classifier to predict the part a feature vec should belong to
      $\Rightarrow$ model $P(P_i \mid f)$, where P_i i^{th} part, f the feature vec

- feature representation for i^{th} part $= \sum_{f \in \{P_i\}} P(P_i \mid f) \cdot f$
  (weighted average, instead of simple part average pooling)
  ⇒ essentially, use attention for a "soft-adaptive" partition

- Training
  - train initial PCB net with all p cross-entropy loss to convergence
  - replace p-part average pooling with p part classifiers (attention generator)
  - with PCB net fixed, train only part classifiers to convergence
    ⇒ ensure part-based attention is learned
    (since there lacks direct supervision for attention to be part-focused)
    ⇒ force to collect part-consistent feature (close to original part feature)
  - train whole net to convergence for fine-tunning
    ⇒ further jointly refine consistency & performance
  - augmented by horizontal flip, 60+10+10 epochs
- Understanding
  - model prior on people: more diverse in vertical, more similar in horizontal (as symmetric)
    (can be visualized by activation testing on their corresponding final classifier - dense layer)
  - separate supervision & independent classifier on each branch is superior
    ⇒ vital to learn&use discriminative part-level features
  - waive the need of learning part partitioning algorithm
    ⇒ less noise source & NOT depends on other realms e.g. human pose estimation
    (as there are gaps between pose estimation & re-ID)
  - setting p needs validation ⇒ $p = 6$ empirically
    (small p: just global feature, large p: some redundant parts being repeated/empty)
  - within-part INconsistency observed in PCB:
    by clustering vec & compute similarity with the average pooling of each part



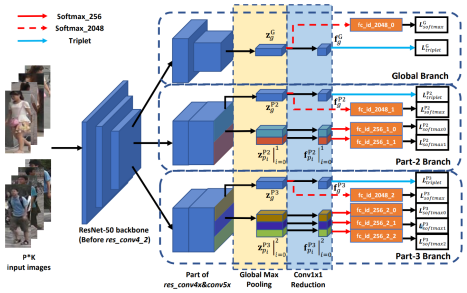
    where,
    partitioned to $p = 6$ parts,
    vec in feature map colored to its closest part
    (by similarity between vec & per-part average pooling)
    ⇒ overall consistent, yet with some outliers
  - RPP emphasize within-part consistency, by refining pre-partitioned parts
    ⇒ protect part-level feature assumption
    (s.t. feature vec within same part are similar & dissimilar to vec in other parts)
  - attention as "soft-adaptive" partition
    ⇒ over whole feature maps to aggregate feature for each part
    ⇒ avoid outliers in "hard-uniform" partition
    ⇒ hence better partition for deep part feature
  - : EM-like iterative training for desired effect (better than only joint training)
    (more supervision)

- MGNet: Discriminative Features with Multiple Granularities
  - Inference
    - CNN extracts feature map, each branch vertically partitioned with various granularity

- each branch output part/global-level representation accordingly
- all (global + multi-granularity) representation concat as final representation

◦ Structure

- ResNet-50 as backbone (before and including res_conv4_1)
- global branch: res_conv5_1 block + global max pooling + 1x1 conv
- part-n branch: split into n strips, each strip a max pooling + 1x1 conv
  (max pooling & 1x1 conv before split to generate branch-level global feature $f_g^{P_n}$)
  ⇒ employ part-2 & part-3 branch



res_conv5_1 block + max pooling + 1x1 conv

n strips, each a max pooling + 1x1 conv
+
max pooling & 1x1 conv before split into strips
(⇒ branch-level global feature $f_g^{P_n}$)

◦ Training

- pre-trained on ImageNet & horizontal flip for data augmentation
- each branch a softmax loss & triplet loss

- softmax: learning basic discrimination in ReID as multi-classification problem
  (classification to all number of class in dataset)
  ⇒ no bias/activation for better discrimination
- batch-hard triplet: metric learning & better ranking performance
  (embed on final feature after 1x1 conv on each branch)
- classfication-before-metric, as shown
  (softmax loss on part feature & before each global feature)
  (triplet loss after softmax & only on each global feature)

◦ Understanding

- multi-scale feature matters ⇒ global + multi-granularity local feature
  ⇒ global branch with downsampling: global & coarse feature
  ⇒ local branch with no strided conv: local & fine feature
  (better than PCB-RPP by a large margin, especially mAP and in hard scene)
- able to learn to focus on various part based on its split region
  e.g. global branch: main body, part-3: small salient feature & limbs
  ⇒ enhanced ability to notice infrequent yet discriminative feature
  (by fine-local feature)
- NOT applying triplet loss on local feature of any branch
  (local feature not enough for identification, hence not bother to confuse model)
- NOT using single feature map for different split of granularity
  ⇒ s.t. each branch can further mining better feature for its split setting
- extra weights are NOT main contributor (experimented)
- ensembling helps, yet still better if multi-branch jointly trained
  (sharing backbone for joint goal: branches mutually complement others)
- ⇒ overlap in split matter: branches part-2/4 worse than branches part-3/4
- triplet loss further help network to capture fine-local feature

Vehicle Re-ID

- Multi-Camera Vehicle Tracking and Re-Identification
  based on visual and spatial-temporal features
  - Inference
    - detection on each frame & multi-object tracking on each camera
    - extract visual feature for each track
    - match track across multi-camera ⇒ multi-cam tracking
    - re-ID: visual feature of given probe
    - re-ranking: use multi-tracking info
  - Detection
    - cascade R-CNN + FPN & soft NMS
  - Feature Extraction from Bbox
    - global feature: last conv layer & self-attention to focus on vehicle pixels (masking)
    - region feature from image parts: 2 vertical + 2 horizontal strips
      (from MGN: multi-granularity net)
    - key point feature: extract feature around key points with the help of heat map
      for key points pred
  - Single Camera Tracking
    - re-project detection to 3D (calibrated cam) ⇒ only 3D tracking
    - Kalman filter & hungarian association with Mahalanobis distance
      ⇒ formulate small tracks from frames of target
    - short tracks association/merge ⇒ long tracks
  - Multi-Camera Tracking
    - similarity $S = W_a S_a + W_l(S_l + S_v) + W_d S_d + W_t S_t$,
      where W_a, W_l, W_d, W_t the weight to balance terms,
      S_a the appearance similarity for each track,
      S_l, S_v the location&velocity similarity (if cam overlaps),
      S_d the similarity of vehicle motion direction (in a uniform coord),
      S_t the temporal constraints: estimated-vs-true travel time (if cam not overlaps)
    - clustering tracks from different cameras ⇒ each cluster is a 3D track
  - Video-based Re-ID
    - identify the track of query & prefer image also belongs to that track
    - meta data constraint: using car-type classification, etc.
  - Understanding
    - a systematic approach: assemble multiple models & constraints

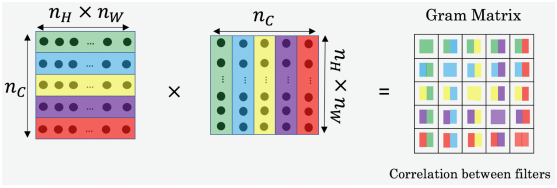
12.5.9 Image Style Transfer

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
    - $\Rightarrow$ 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)



12.5.10 Video Generation

Problem Formulation

- Input
  - Image
    - as the first frame of the video
  - Caption
    - specifying the video content
- Output

- a video sequence

- Metrics
  - Similarity / Distance
    - between generated & original video
      (original video usually uniformly sampled & with its image downsampled)
  - Use Study
    - unique human answering question
      e.g. which video more realistic / more suitable for the given caption? etc.

- Challenge
  - Consistency
    - consistent scenes between frames
  - Realistic Motion
    - especially human/animal motion due to their high complexity
  - Conditioning Video Generation
    - control the content, style, etc.
      (given initial frame: overlap with video prediction realm)
      (while caption provides more control)

Generation from Caption

- CFT-GAN: Conditional Video Generation Using Action-Appearance Captions
  - Inference (Generation)
    - encode caption with randomness: encode the description of subject, action & background, etc.
    - generate optical flow to represent action given $\psi, z_{\text{flow}} \sim p_z$
  - Caption Encoding
    - caption feature: ψ by Fisher Vectors with HGLMM
    - conditioning augmentation: $\mathbf{c} = \psi * (1 + \epsilon)$, where $*$ element-wise product
      $\Rightarrow$ avoid coarse distribution of ψ by adding noise $\epsilon \sim \mathcal{N}(0, 1)$
      $\Rightarrow \mathbf{c} \sim \mathcal{N}(\psi, 1)$
  - Latent Variables
    - random variable (vector) $z \sim \mathcal{N}(0, 1)$
  - Motion Generator FlowGAN
    - input: concat [sampled $\mathbf{c} = \mathbf{c}_{\text{flow}}$, sampled $z = z_{\text{flow}}$]
    - based on VGAN: generate optical flow $\mathbf{f} = m(z) * f(z)$, ($*$ element-wise product)
      where $m(z)$ a mask to fuse foreground $f(z)$ & background $b(z) = 0$
      ($b(z) = 0$ due to static camera, as background NOT generating optic flow)
    - for each time step, upsampled to 64×64 feature map & mask
      $\Rightarrow$ 3D conv for volume of $64 \times 64 \times t$
  - Motion Discriminator
    - concat an input flow & a corresponding real flow, downsample
    - dense layer to compress original caption feature ψ as ψ'
    - tile ψ' into downsampled feature map & discriminate if input flow is real or fake
      (further discriminate if flow related to caption)

- Appearance Generator TextureGAN
  - input: optic flow $\mathbf{f}$ & concat [sampled $\mathbf{c} = \mathbf{c}_{\text{tex}}$, sampled $z = z_{\text{tex}}$]
  - condition variables map $\mathbf{f}_c$: upsample $[\mathbf{c}, z]$ by 2 upsampling blocks
  - $\mathbf{f}$ as U-net input & $\mathbf{f}_c$ concat in downsampling stage
    $\Rightarrow$ retain spatial info: reflect edges in $\mathbf{f}$ as shape of moving target
    $\Rightarrow$ output both foreground f & mask m of shape $64 \times 64 \times t$
  - background b: upsampled from $[\mathbf{c}, z]$ for a single frame & replicated t times
  - fusion: video $v = f * m + (1 - m) * b$
- Video Discriminator
  - concat input video & a corresponding real video, downsample
  - downsample the optic flow for input video $\mathbf{f}$ as $\mathbf{f}'$ & further downsample
  - dense layer compress caption ψ as ψ' & concat to downsampled feature map
    (also discriminate if video related to caption)
- Training
  - textureGAN trained initially using real optic flow calculated from real video
    (until flowGAN more stable & trained to a extend, to avoid wasted training)
  - use existing video dataset & adding action-appearance caption
- Understanding
  - demonstrate ability for finer control on video content
    (not only action, but background, appearance)
    $\Rightarrow$ two-stage (action + appearance) generation for each frame for realistic complex scene
    $\Rightarrow$ caption for more descriptive control (v.s. initial frame)
  - more abstract & general control
    $\Rightarrow$ reflect more variety of complex actions and appearances by captions
    (compared to use image as initialization)

12.5.11 Point Cloud Data Processing

Problem Formulation

- Input Data
  - Point Cloud
    - from lidar $\Rightarrow$ 3-D position x,y,z & reflection intensity
    - from radar $\Rightarrow$ 2-D bird-view x,y & intensity (rcs) & speed (along radial direction)
- Goal
  - 3-D Environment Modeling
    - bounding box
    - segmentation (per-point classification)
    - instance segmentation, etc.
- Challenge
  - Sparsity
    - point gets much sparser in distance (e.g. $>$ 40m)
  - Varying Density
    - due to occlusion, distance etc.

Common Preprocessing

- Voxelization
  - Goal
    - create 3-D pixel, the voxel
  - Procedure
    - apply 3-D grid on the space
      $\Rightarrow$ each point resides in a spatial cell, the voxel
- Bird-view Projection
  - Goal
    - project onto 2-D map of a bird-view perspective
      $\Rightarrow$ better resolution due to reduced dimension
  - Procedure
    - apply 2-D grid on the ground
      $\Rightarrow$ each point resides in a cell, or, each cell consists of several point
    - extract height information from points in each cell
      $\Rightarrow$ each pixel with a height h
- Cylindrical Projection (Frontal View)
  - Goal
    - project onto 2-D map of pilot perspective
      $\Rightarrow$ better align with camera perspective
  - Procedure
    - given 3-D cartesian coord x, y, z
      $\Rightarrow$ spherical coord $r = \sqrt{x^2 + y^2 + z^2}, \theta = \arctan \frac{y}{x}, \phi = \arcsin \frac{z}{r}$
    - slicing on horizontal & vertical angle: each point resides in a 3-D slice
      $\Rightarrow$ normalized by angle resolution $\theta' = \lfloor \frac{\theta}{\delta\theta} \rfloor, \phi' = \left\lfloor \frac{\phi}{\delta\phi} \right\rfloor$
    - each pixel (θ', ϕ') extract depth $d = \sqrt{x^2 + y^2}$ & height z from its point(s)

Point Cloud

- PointNet
  -
- VolxelNet
  - Preprocessing
    - voxelization
    - random sampling points in each voxel to be at most T points a voxel
  - Bounding Box Encoding
    - x, y, z for center position; l, w, h for size; θ for orientation, the yaw rate
      $\Rightarrow x_g, y_g, z_g, l_g, w_g, h_g, \theta^g$ the ground truth box
      $\Rightarrow x_a, y_a, z_a, l_a, w_a, h_a, \theta^a$ an anchor box
    - normalized residual position: $\Delta x = \frac{x_g - x_a}{\sqrt{l_a^2 + w_a^2}}, \Delta y = \frac{y_g - y_a}{\sqrt{l_a^2 + w_a^2}}, \Delta z = \frac{z_g - z_a}{h_a}$
    - normalized size ratio: $\Delta l = \log(\frac{l_g}{l_a}), \Delta w = \log(\frac{w_g}{w_a}), \Delta h = \log(\frac{h_g}{h_a})$
    - residual orientation: $\Delta\theta = \theta_g - \theta_a$

- Inference
  - voxel map with raw points as input, output 3-D classification & regression map
- Voxel Feature Encoding Layer
  - voxel $V = \{p_i = [x_i, y_i, z_i, r_i] \in \mathbb{R}^4\}_{i=1,...t}, t < T$
  - augment each point with its offset to the centroid (v_x, v_y, v_z), the mean of $p \in V$
    $\Rightarrow p_i' = [x_i, y_i, z_i, r_i, x_i - v_x, y_i - v_y, z_i - v_z]$
  - 1×1 conv (with batch norm, ReLu) on each feature point: $p_i' \to f_i$
  - element-wise max pooling on $f_i \in V$: voxel-wise feature $\hat{f}$
  - augment each processed feature point with voxel feature: $f_i^{\text{out}} = [f_i, \hat{f}]$
- Structure
  - stacked voxel feature encoding layer: deep net to extract feature for each voxel
  - detection model: RPN modified for 3-D detection
- Training
  - per-voxel weighting: balanced according to num of positive example
  - smooth L1 for regression loss
  - further up-/down-sample positive/negative voxel in classification loss
  - predicted box considered positive, if its IoU with label box > 0.6 & is the highest
    (negative, if its IoU with any label box < 0.45)
    (not-care, if its IoU with any label box $\in [0.45, 0.6]$)
  - data augmentation: box (collision-free) perturbation in location, size, rotation
- Understanding
  - range: $[-3, 1] \times [-20, 20] \times [0, 48]$ in height-width-length
  - voxel feature extraction: voxel-level PointNet
    $\Rightarrow$ able to concat with various detection models (e.g. YOLO, etc.)

- LMNet
  - Preprocessing
    - cylindrical projection
  - Bounding Box Encoding
    - given a pixel (corresponding to a lidar point) p in a bounding box
      $\Rightarrow$ encode box under coord originated at p
      $\Rightarrow$ axises: x along radial direction; y parallel with horizontal plane; z accordingly
    - encode 8 corner $\Rightarrow$ 24 channel encoding for each box
  - Classification Encoding
    - per-pixel classification (car, pedestrian, cyclist, none)
  - Inference
    - per-pixel regression & classification $\Rightarrow$ standard CV detection
    - non-max suppression as postprocessing
      (score modified to be the num of nearby-box)
  - Structure
    - convs - max pooling - dilated convs - unpooling - branch for regression/classification
  - Training
    - point-wise weighting
      $\Rightarrow$ regression: consider box size for each class
      $\Rightarrow$ classification: downsample background pixel (standard)

- Understanding
  - real-time timing due to simple structure with dilated conv
    (though performance hurt...) ⇒ enable further fusion with image, pertaining real-time timing

Point Cloud + Image Fusion

- MV3D: Multi-view 3D Object Detection
  - Preprocessing
    -
  - Bounding Box Encoding
    - location
    - size
    - orientation
  - 3D Proposal Network
    - bird-view of point cloud ⇒ propose reliable 3D bbox
  - Region-based Fusion Network
    - project 3D proposal to feature maps of multi-view
  - Understanding
    -

12.6 Natural Language Processing

12.6.1 Language Representation

Problem Formulation

- Input
  - Language Token/Corpus
    - words, sentences, paragraphs, ... ⇒ 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
      ⇒ look up for the desired embedding
      (NOT using matrix multiplication due to sparsity from one-hot encoding)
    - distributed representation: decomposed yet meaningful
      ⇒ fight the curse of dimensionality
  - ⇒ Meaningful Vector
    - able to measure the (dis-)similarity of between tokens (words)
      ⇒ semantic meaning: "Germany"-"Berlin" & "France"-"Paris"
      ⇒ 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")
    - ⇒ 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
      $\Rightarrow$ 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 $\Rightarrow$ 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 $e_{\text{man}} - e_{\text{woman}}, e_{\text{male}} - e_{\text{female}}, ...$
    - 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 $d(e_{\text{boy}}, e_{\text{doctor}}) = d(e_{\text{girl}}, e_{\text{doctor}})$

Contextualized Word Embedding

- ELMo
- BERT

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

Approaches

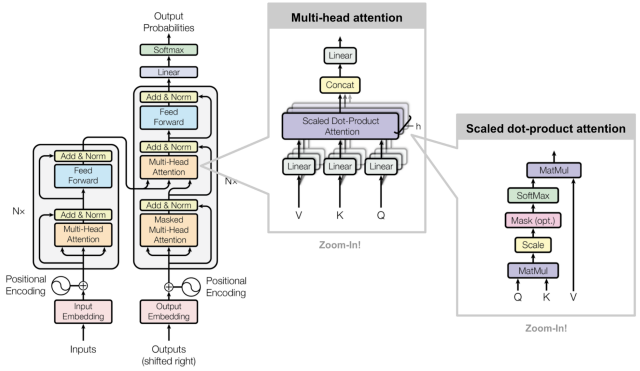
- RNN Encoder-Decoder
  - Inference
    - an RNN encodes the input sequence by its last hidden layer
    - input encoding used as initial hidden state for decoder RNN
    - decoder RNN generates (conditional) distribution over words at each step
      $\Rightarrow$ for time $t, y^t = p(z^t|x^1, ..., x^{T_x}, \hat{z}^1, ..., \hat{z}^{t-1})$,
      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

- Transformer: Attention is All You Need
  - Embedding
    - word embedding: classic embedding + positional embedding
    - positional embedding:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right),$$

      where $d_{\text{model}} = 512$ the embedding len, pos the word position, i the dim of vector
      $\Rightarrow$ any offset $k, PE(pos+k)$ can be represented by a linear func of $PE(pos)$
      $\Rightarrow$ provide positional info easy for model to use
      (also allow model to extrapolate sequence longer than training examples)
  - Inference
    - encoder maps input $(x_1, ..., x_n) \to \mathbf{z} = (z_1, ..., z_n)$ (in parallel)
    - decoder maps $\mathbf{z} \to (y_1, ..., y_m)$ one at a time (sequentially)
    - beam search with bean size = 4 & length penalty $\alpha = 0.6$
    - maximal output len = input len + 50
  - Structure
    - encoder: stack of $N = 6$ layers, each: [multi-headed self-attention, dense layer], with residual connection around both sub-layers (attention and dense) & layer normalization after each residual connection
    - multi-headed self-attention: 8 heads with 64-dim vector for key, query & output (vs. word embedding of dim $d_{\text{model}} = 512$: same complexity with more representability)
    - encoder-decoder attention:
      self-attention with key-value from encoder & query from last decoder
      $\Rightarrow$ mimic the classic encoder-decoder attention
    - decoder: a stack of layers similar to encoder + encoder-decoder attention
      $\Rightarrow$ restricted attention (via masking) to output sequentially
    - overview: (scaled dot-product attention = self attention)



  - Training
    - 8 NVIDIA P100 for 3.5 days...4000 steps warm up + 300,000 steps training (still, much faster training than other models...)

        - adam optimizer & batching based on approximate sentence length
        - drop-out (dropout rate = 0.1) after each sublayer (before the residual add)
        - label smoothing with value $\epsilon_{ls} = 0.1$???
        - final model = the average of the last 20 checkpoints
    - Understanding
        - no sequential component use in encoding
          (entirely replace the RNN with attention in encoder-decoder structure)
          $\Rightarrow$ parallelized encoding phase
        - the dimension of key-value should NOT be too small
          $\Rightarrow$ determining relations can be difficult
          $\Rightarrow$ may use non-linearity
        - model structure can generalize to other challenging NLP tasks
          e.g. constituency parsing: structural constraint & much longer output than input
          (also, little annotated data, $\sim 40K$ sentences)

- BERT
    -

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)
      ⇒ 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)
- Train Set Setup
  - Basic
    - 0 for frames not corresponding to trigger word; 1 for frames consisting trigger words
  - upsampling
    - 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

12.7 CV & NLP

12.7.1 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
- Challenge
  - Scene Understanding
    - basic: determine objects in image
    - further: realize relationships & connection inside the image
  - Descriptive Representation
    - able to map such understanding into a descriptive representation
      i.e. natural language

Approaches

- 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)

- Show, Attend and Tell: Neural Image Caption Generation with Visual Attention
  - Motivation & Analysis
    - does NOT explicitly use object detection net
      $\Rightarrow$ model visual relations beyond objectness
    - better capture low-level detail for descriptive language
      $\Rightarrow$ better represent visual perception than a single feature vector
      (which is usually used as starting state for RNN-decoder in previous works)
  - Inference
    - CNN downsample image to $n \times n = L$ feature map, with channel dimension D
      $\Rightarrow$ feature map $\mathbf{a} = \{\mathbf{a}_1, ..., \mathbf{a}_L\}, \mathbf{a}_i \in \mathbb{R}^D$: spatial region for attention to choose
    - LSTM decoder with init states predicted by 2 separate small nets (init,c & init,h)
      $\Rightarrow$ memory state $c_0 = f_{\text{init,c}}(\bar{\mathbf{a}})$ & hidden state $h_0 = f_{\text{init,h}}(\bar{\mathbf{a}})$,
      where $\bar{\mathbf{a}} = \frac{1}{L}\sum \mathbf{a_i}$ (the avg pool of $\mathbf{a}$)
    - generate current spatial attention α_t from previous hidden state h_{t-1} & feature map $\mathbf{a}$
      $\Rightarrow (\mathbf{a}_i, h_{t-1}) \xrightarrow{f_{\text{att}}}$ score $e_{ti} \xrightarrow{\text{softmax}} \alpha_{ti}$
    - generate current context vector $\hat{\mathbf{z}}_t = \phi(\mathbf{a}, \alpha_t)$
      (different ϕ for hard-/soft- attention)
    - generate word prob with current context $\hat{\mathbf{z}}_t$, hidden state h_t & previous word y_{t-1}
      $\Rightarrow p(y_t \mid \mathbf{a}, y_1, ..., y_{t-1}) \propto \exp(\mathbf{L}_o\,(Ey_{t-1} + \mathbf{L}_h h_t + \mathbf{L}_z \hat{\mathbf{z}}_t))$,
      where $\mathbf{E}$ the word embedding matrix, $\mathbf{L}_o, \mathbf{L}_h, \mathbf{L}_z$ the weights matrices to be learned
  - Structure
    - CNN encoder: VGG net to the last conv ($14 \times 14 \times 512$ feature map)
    - LSTM decoder takes [previous generated word, context vector] as input
      $\Rightarrow$ context vector $\hat{\mathbf{z}}_t$ involves in both: LSTM input & projecting LSTM output
  - Training: Stochastic Hard-Attention
    - let one-hot variable $s_t \in \{1, ..., L\}$ indicate the focused location at time t
      $\Rightarrow \hat{\mathbf{z}}_t = \sum_i s_{ti}\mathbf{a}_i$
    - view attention α a multinoulli distribution over location $1, ..., L$
      (multinoulli: discrete distribution over k variables)
      $\Rightarrow p(s_{ti} = 1|\mathbf{a}, s_1, ..., s_{t-1}) = \alpha_{ti}$
    - loss: marginal log-likelihood of word sequence conditioned on image $\log p(\mathbf{y}|\mathbf{a})$
      $\Rightarrow$ maximize its variational lower bound with s introduced as latent var

      $$\begin{aligned} L_s &= \sum_s p(s|\mathbf{a}) \log p(\mathbf{y}|, s, \mathbf{a}) \\ &\le \log \sum_s p(s|\mathbf{a}) p(\mathbf{y}|s, \mathbf{a}) \\ &= \log p(\mathbf{y}|\mathbf{a}) \end{aligned}$$

    - $\dfrac{\partial L_s}{\partial W} = \sum_s p(s|\mathbf{a}) \left[\dfrac{\partial \log p(\mathbf{y}|s, \mathbf{a})}{\partial W} + \log p(\mathbf{y}|s, \mathbf{a}) \dfrac{\partial \log p(s|\mathbf{a})}{\partial W} \right]$
    - $\Rightarrow$ approximated by Monte Carlo:
      $\dfrac{\partial L_s}{\partial W} \approx \dfrac{1}{N} \sum_{n=1}^{N} \left[\dfrac{\partial \log p(\mathbf{y}|\tilde{s}^n, \mathbf{a})}{\partial W} + \log p(\mathbf{y}|\tilde{s}^n, \mathbf{a}) \dfrac{\partial \log p(\tilde{s}^n|\mathbf{a})}{\partial W} \right]$,
      where $\tilde{s}^n = (s_1^n, ..., s_t^n, ...)$ a sequence of attention locations,
      which is sampled from multinoulli distribution described by α

        - to reduce the variance in Monte-Carlo estimation:
            - moving average baseline for k-th batch: $b_k = 0.9b_{k-1} + 0.1 \log p(\mathbf{y}|\tilde{s}_k, \mathbf{a})$
            - include entropy $H[s]$ of the multinoulli distribution into the loss
            - with 0.5 prob to set $\tilde{s}$ to its expected value α (turn into soft-attention)
        - $\Rightarrow$ final gradient on loss
        $$\frac{\partial L_s}{\partial W} \approx \frac{1}{N}\sum_{n=1}^{N}\left[\frac{\partial \log p(\mathbf{y}|\tilde{s}^n, \mathbf{a})}{\partial W} + \lambda_r \left(\log p(\mathbf{y}|\tilde{s}^n, \mathbf{a}) - b\right)\frac{\partial \log p(\tilde{s}^n|\mathbf{a})}{\partial W} + \lambda_e \frac{\partial H[\tilde{s}^n]}{\partial W}\right]$$
    - Training: Soft Attention
        - use the expectation $\mathbb{E}_{p(s_t|\mathbf{a})}[\mathbf{z}_t] = \sum_{i=1}^{L} \alpha_{ti}\mathbf{a}_i$
        (equivalent to using α weighted context)
        $\Rightarrow$ approximately optimize marginal likelihood $p(\mathbf{y}|\mathbf{a})$ under random variable s_t
        - $\mathbb{E}[\hat{\mathbf{z}}_t]$ approximates the normalized weighted geometric mean of the softmax of k-th generated word,
        which further approximates $\mathbb{E}[p(y_t = k|\mathbf{a})]$ over all possible attention locations (induced by s_t)
        - additional gate for $\phi(\mathbf{a}, \alpha_t) = \beta_t \sum_{i=1}^{L} \alpha_i \mathbf{a}_i$, where gate $\beta_t = \sigma(f_\beta(h_{t-1}))$
        $\Rightarrow$ enable model to decide when to emphasize on language model or image context
        - temporal constrain: encourage $\sum_t \alpha_{ti} = \tau$
        (normalization for each spatial location across time)
        - $\Rightarrow$ end-to-end maximize negative log-likelihood:
        $L_d = -\log(p(\mathbf{y}|\mathbf{a})) + \lambda \sum_{i}^{L}(\tau - \sum_{t}^{T}\alpha_{ti})^2$, where $\tau = 1$ for simplicity
    - Understanding
        - spatial attention to capture low-level detail for more accurate description
        $\Rightarrow$ dynamically select useful features & avoid info loss
        (using last-conv feature map vs. using last-dense layer)
        - provide more interpretable result
        $\Rightarrow$ able to interpret relationships over image-language
        - enable model to attend to non-salient features by soft-attention
        (e.g. large area of background)

12.7.2 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 $[\text{height}, \text{wide}, \text{channel}]$)
    - 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 $\Rightarrow$ 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)

12.8 Special Learning

12.8.1 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
          $\Rightarrow$ 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

12.8.2 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 ⇒ 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

12.8.3 K-shot Learning

Problem Formulation

- Training Set
  -

12.9 Interpretable Machine Learning

12.10 Tactile Sensing

12.10.1 Tactile Sensor

- Overview
  - Single-point Contact Sensors (single tactile cells)
    - able to confirm object-sensor contact
    - force sensor to detect force at contact point

        - biomimetic whiskers (dynamic tactile sensors) to detect vibration at contact point
    - High Resolution Tactile Arrays (fingertips)
        - sensors array of tactile sensing elements (usually planar array)
        - sensing elements: fiber optics, embedded camera, barometers, etc.
    - Large-Area Tactile Sensors (body skins)
        - usually able to be curved for large robotic skin

12.10.2 Tactile Sensing and Perception

Problem Formulation

- Input Data
    - Tactile Data
        - the data after on-board processing on the raw tactile sensor
        - come with different form/meaning, depending on the sensor
- Goal
    - Detail Object Recognition
        - recognize the object shape, material properties, in-hand pose and etc.
        - action related: grasp control, slippage detection, etc.

Feature Extraction

- Acoustic Signal
    - Input
        - microphone to collect signal from tapping/sliding/scratching
    - Processing
        - filter/function (e.g. fast Fourier transform) to map to frequency domain
        - then analyzed/classified into texture properties
    - Understanding
        - fast, low-cost, light-burden
        - need interaction
- Vibration and Force Data
    - Input
        - robot finger sliding with various speed to detect vibration/friction
        - strain gauges sensor to detect stiffness, compliance, elasticity and etc.
    - Processing
        - transform into frequency domain, then analyze the vibration intensities
        - or, directly use kNN, SVM, etc on mechanical impedance data to classify
- Micro-structure Patterns
    - Input
        - tactile sensors array for pressed/imprinted surface
    - Processing

        - sensor data with spatial relation ⇒ a small image (usually 14×6)
          ⇒ image processing (with hand-crafted/learned features)
        - learning features: end-to-end learned for various task

- Spatial-Temporal Data
    - Input
        - a series acquired local tactile data, each with sensor-object contact location & time
    - Processing
        - viewed as a spatial points cloud, then recognize the object contour
        - incorporate with LSTM algorithms: e.g. landmarks, grid based, particle filter, etc.
        - incorporate sequential model: e.g. KF to sequentially refine object properties

Application in Robot Tasks

- Robot Manipulation
    - Input
        - image from vision sensor, data from tactile sensor
    - Goal
        - successfully perform task involving object manipulation
          e.g. open a door via door handle
    - Single-Point Contact
        - structure as spatial-temporal points ⇒ in-hand pose estimation (usually with LSTM methods)
        - challenge: sensor-object contact influence the environment
          (use lidar?)
    - Tactile Arrays
        - image processing & SLAM
    - Tactile & Vision
        - 3D vision (from lidar/camera) ⇒ sensor fusion for localization
        - vision for haptic&location estimation + tactile for refinement
        - vision & tactile both treated as image: potentially sharing weights (multi-tasking)

- Sensor Fusion for Environment Modeling
    - Contact Verification
        - verify robot-object contact
        - active searching ambiguous region in visual perception
        - jointly estimate likelihood of contact at each location in the environment
    - Object Properties Refinement
        - model the in-hand object as in robot manipulation (can be end-to-end)
          e.g. estimate the angle of door handler while gripping