

# Notes on Machine Learning & Deep Learning

Ara Patapoutian, 2017

## CONTENTS

<b>I</b>	<b>Introduction</b>	2
I-A	Probabilistic formulation . . . . .	3
I-B	Datasets . . . . .	3
<b>II</b>	<b>Fundamentals</b>	5
II-A	Cost functions . . . . .	5
II-B	Neuron models . . . . .	6
II-C	Update Rules . . . . .	9
<b>III</b>	<b>Generalization</b>	13
III-A	Regularization . . . . .	14
III-B	Validation . . . . .	16
<b>IV</b>	<b>Feedforward Networks with Supervision</b>	18
IV-A	Multilayer perceptrons . . . . .	18
IV-B	Hyper-parameters . . . . .	20
IV-C	Visualization . . . . .	20
IV-D	Deep learning . . . . .	22
IV-E	Convolutional, inception & residual networks . . . . .	23
IV-F	Transfer Learning . . . . .	25
<b>V</b>	<b>Recurrent Neural Networks with Supervision</b>	27
<b>VI</b>	<b>Unsupervised Learning</b>	30
<b>VII</b>	<b>Applications</b>	32
VII-A	Health care & Biomedicine . . . . .	32
<b>VIII</b>	<b>Visual Recognition</b>	33
VIII-A	Color . . . . .	34
VIII-B	Dataset . . . . .	34
VIII-C	Image Processing . . . . .	35
VIII-D	Vision with legacy machine learning . . . . .	37
VIII-E	Vision with neural networks . . . . .	38
<b>IX</b>	<b>Natural language processing</b>	42
<b>X</b>	<b>Tools</b>	48
<b>XI</b>	<b>VC Generalization Bound</b>	51
<b>XII</b>	<b>Bias &amp; Variance Method</b>	53
<b>XIII</b>	<b>SVM's &amp; Kernel Methods</b>	54
<b>XIV</b>	<b>Similarity Methods</b>	58
<b>XV</b>	<b>Miscellaneous Mathematical Entities</b>	60
	<b>References</b>	64

## I. INTRODUCTION

- Computer science, artificial intelligence, machine learning & neural networks:
  - *computer science*, or CS, is the study of the principles and use of computers
  - *artificial intelligence*, or AI, is a subfield of CS that studies tasks that normally require human intelligence
  - *machine learning*, or ML, is a subfield of AI that evolved from the study of pattern recognition and computational learning theory
  - *neural networks*, or NN, is a subfield ML that is inspired by biological neural networks

$$\text{CS} \longrightarrow \text{AI} \longrightarrow \text{ML} \longrightarrow \text{NN} \quad (1)$$

- Machine learning:
  - ML is the science of getting computers to act without being explicitly programmed
  - ML is applied on problems where
    - 1) data is available
    - 2) a pattern or correlations exist
    - 3) mathematical formulation is not practical or possible

- Learning:

- in nature, cause precedes effect

$$\text{cause} \rightarrow \text{effect} \quad (2)$$

in learning, observations lead to constructing cause

$$\text{effect} \rightarrow \text{cause} \quad (3)$$

- formalization of learning  $(\mathcal{H}, \mathcal{A})$ :

- \*  $\mathcal{H} = \{h_i\}$  is a hypothesis set,

- \* a learning algorithm,  $\mathcal{A}$ , picks the *final hypothesis*  $g$

$$\mathcal{H} = \{h_i\} \xrightarrow{\mathcal{A}} g \in \mathcal{H} \quad (4)$$

- learning versus memorization:

- \* learning allows for generalization, whereas memorization does not, see Section III

- Paradigms of learning:

- 1) supervised learning, Sections IV and V

- supervised learning implies data is available for training with the desired outcome
- in classification, the correct outcome is referred to as the *label*
- at present, supervised learning is the prevalent learning paradigm

- 2) unsupervised learning, Section VI

- no labels available during training

- 3) reinforcement learning,

- no supervision, only a reward system
- feedback is delayed, not instantaneous
- for applications where time matters, i.e. sequential, not i.i.d.
- e.g. backgammon

- Neural networks:

- a neural network is sometimes referred to as *artificial neural network* or ANN

- NN is hands-on, empirical field for now

- \* experimentation over deep thoughts

- a NN is composed of neurons, see Section II-B

- little is known about biological learning to provide guidance for learning of NNs

- NNs come in two flavors:

- \* feedforward, see Section IV, and

- \* recurrent, see Section V

- NNs have been successful in recent years and are the main focus of this write-up [1]

- Factors of variation:

- when analyzing a speech, the *factors of variation* include the speakers age, their sex, their accent and the words that they are speaking

- when analyzing an image of a car, the factors of variation include the position of the car, its color, and the angle and brightness of the sun
- unfortunately, many of the factors of variation influence every single piece of data we are able to observe
- it is usually needed to disentangle the factors of variation and discard the ones that we do not care about
- Features in a representation:
  - the performance of a machine learning algorithm depends on the *representation* of the data they are given
  - each piece of information included in the representation is known as a *feature*
  - feature extraction from input data is nonlinear
  - features reduce the input data dimensionality by trying to extract useful information
- Template matching:
  - template matching is a simple feature extractor, that may work if factors of variations are limited
- Representation learning:
  - externally identifying relevant features is difficult
  - current philosophy is to use NN to discover not only the mapping from representation to output but also the representation itself
  - this approach is known as *representation learning*

#### A. Probabilistic formulation

- Interpreting NN as generating a conditional probability: target distribution
  - consider a NN with supervised learning, where the data & label pairs  $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ , are available
    - \*  $\mathbf{x}_n$  is a  $d$  dimensional random vector, representing the input
    - \*  $\mathbf{y}_n$  is a  $d_o$  dimensional vector, representing the observed output,
  - a NN models the conditional probability of the map

$$\mathbf{x}_n \xrightarrow{P(\mathbf{y}|\mathbf{x})} \mathbf{y}_n \quad (5)$$

where  $P(\mathbf{y}|\mathbf{x})$  is a conditional probability function

- $P(\mathbf{x})$  &  $P(\mathbf{y}|\mathbf{x})$  are in general unknown
- $(\mathbf{x}_n, \mathbf{y}_n)$  can be generated from  $P(\mathbf{x}, \mathbf{y})$ , where

$$P(\mathbf{x}, \mathbf{y}) = P(\mathbf{x}) P(\mathbf{y}|\mathbf{x}) \quad (6)$$

- $P(\mathbf{y}|\mathbf{x})$ , is referred to as the *target distribution*
  - \* the target distribution is the quantity that needs to be learned

- Equivalent formulation: target function
  - the target distribution can be interpreted as a deterministic mapping, known as the *target function*

$$f(\mathbf{x}_n) \triangleq E_{\mathbf{y}}(\mathbf{y}|\mathbf{x}_n) \quad (7)$$

together with some *stochastic noise*

$$\mathbf{n}_n \triangleq \mathbf{y}_n - f(\mathbf{x}_n) \quad (8)$$

- stochastic noise is a characteristic of the environment and is independent of the NN
- under this formulation, the goal of a NN is to learn  $f(\mathbf{x})$  (7), from  $\{(\mathbf{x}_n, \mathbf{y}_n)\}$
- target function is assumed to be unknown

- Classification versus regression:
  - in a *classification* problem,  $\{\mathbf{y}_n\}$  is a finite set
  - in a *regression* problem,  $\{\mathbf{y}_n\}$  is a subset of the real number
    - \* regression means real-valued output

#### B. Datasets

- Obtaining large repository of data, along with talent, are the two scarce resources
  - this is a major bottleneck for startups
  - some products are launched with the purpose of collecting data rather than revenue
- Mean & variance:
  - consider sample data set  $\{\mathbf{x}_n\}$ , of *sample size*  $N$
  - arrange the data set  $\{\mathbf{x}_n\}$  into an  $(N \times d)$  *data matrix*  $X$ , whose  $n^{\text{th}}$  rows is  $\mathbf{x}_n$

- the mean  $\bar{\mathbf{x}}$ , associated with  $\{\mathbf{x}_n\}$ , is

$$\begin{aligned}\bar{\mathbf{x}} &= \frac{1}{N} \sum_{n \leq N} \mathbf{x}_n \\ &= \frac{1}{N} \mathbf{1}^T X\end{aligned}\tag{9}$$

where  $\mathbf{1}$  is  $(N \times 1)$  vector of 1s

- the covariance  $\bar{C}$ , associated with  $X$  is

$$\begin{aligned}\bar{C} &= \frac{1}{N} \sum_{n \leq N} (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \\ &= \frac{X^T X}{N} - \bar{\mathbf{x}} \bar{\mathbf{x}}^T\end{aligned}\tag{10}$$

- Dataset modifications:

- various modifications can be applied to a dataset, including
  - \* preprocessing
  - \* normalization
  - \* expansion, or augmentation
  - \* reuse

- Data preprocessing:

- dataset  $\{\mathbf{x}_n\}$  is usually modified before applying to a NN, by making it zero mean,

$$\hat{\mathbf{x}}_n \triangleq \mathbf{x}_n - \bar{\mathbf{x}}\tag{11}$$

- it is possible to reduce data dimensionality ( $d$ ), using *principle component analysis*, or PCA (234)
- a dataset  $\{\mathbf{x}\}$  can be *whitened* by normalizing the variance on each component by dividing each component by the square root of its eigenvalue
- for images, usually only mean centering is performed

- Batch normalization:

- in practice the data-set is partitioned to *mini-batches* and the NN is updated one mini-batch at a time
  - \* denote the mini-batch size by  $m$
- with *batch normalization*, every layer  $k$ , of a network is augmented with another normalizing layer
- each layer performs the following normalization:
  - \* determine the batch mean  $\mu_k$  (9) & variance  $\sigma_k^2$
  - \* given scale & shift parameters  $(\gamma^{(k)}, \beta^{(k)})$  compute

$$\mathbf{x}_k \longrightarrow \hat{\mathbf{x}}_k \triangleq \frac{\mathbf{x}_k - \mu_k}{\sigma_k} \longrightarrow \mathbf{y}_k \triangleq \gamma^{(k)} \hat{\mathbf{x}}_k + \beta^{(k)}\tag{12}$$

- Data expansion or augmentation:

- the training data  $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ , can be artificially expanded by applying, on the available data, application-dependent transformations
  - \* for image recognition, can use
    - translation
    - rotation
    - horizontal flip
    - random crops & scales and then rescale
    - deformation, distortion, stretch and sheer
    - color jitter such as contrast, brightness, saturation
  - \* for speech recognition, can introduce background interference, change the speed etc.

- Data snooping:

- if a data set has affected any step in the learning process, its ability to assess the outcome has been compromised
- this phenomenon is known as *data snooping*
- data reuse and expansion should be carried out carefully
- data snooping is a common trap for a practitioner

- Data partitioning:

- data is partitioned into training, testing and validation, see Section III

## II. FUNDAMENTALS

### A. Cost functions

- General remarks:
  - learning is accomplished by minimizing some cost function
    - \* or equivalently by reward maximization
  - cost functions are also referred to as *error metrics*, *lost functions*, or *objective functions*
- Pointwise cost function:
  - for a given input  $\mathbf{x}$ , the pointwise *cost function* between hypotheses  $h$  &  $f$  is

$$c_{h,y} \triangleq c(\mathbf{y}^{(h)}, \mathbf{y}^{(f)}) \quad (13)$$

where

- \*  $\mathbf{y}^{(h)} \triangleq h(\mathbf{x})$  is the output associated with hypothesis  $h$
- \* such a cost function can be applied for all learning paradigms
- \* when the label  $\mathbf{y}$  associated with data  $\mathbf{x}$  is available, the cost function associated with hypothesis  $h$  becomes

$$c_h \triangleq c(\mathbf{y}^{(h)}, \mathbf{y}) \quad (14)$$

- Cost function examples:
  - 1) The *Hamming distance*, defined as

$$c_{h,f} \triangleq [h(\mathbf{x}) \neq f(\mathbf{x})] \quad (15)$$

is often used when  $f(\mathbf{x}) \in \{0, 1\}$

- 2) The *generalized Euclidean distance* is defined as

$$c_{h,f} \triangleq (\mathbf{y}^{(h)} - \mathbf{y}^{(f)})^T Q (\mathbf{y}^{(h)} - \mathbf{y}^{(f)}) \quad (16)$$

where  $Q$  is an arbitrary positive semi-definite matrix  
two special classes include

- the *Euclidean distance*, when  $Q = I$
- the *Mahalanobis distance*, when  $Q$  is the inverse of the covariance matrix of the data
  - \* the Mahalanobis distance metric depends on the data set
  - \* the main advantage of the Mahalanobis distance over the standard Euclidean distance is that it takes into account correlations among the data dimensions and scale

- 3) The log-likelihood cost function:

- the log-likelihood cost function, from the ML criterion (239), is given by

$$\log_2 \frac{1}{P(y|\mathbf{x})} \quad (17)$$

- since a NN can model  $P(y|\mathbf{x})$  (5), using hypothesis  $h$  (4, 38), the log-likelihood cost function is expressed as

$$\begin{aligned} c_h &= \log_2 \frac{1}{h(\mathbf{x})} \\ &= -\log_2 h(\mathbf{x}) \end{aligned} \quad (18)$$

- the log-likelihood cost function is often used with softmax activation (47)

- 4) The *cross-entropy* cost function (247):

- the cross-entropy measure is a useful metric when the NN is estimating the probability distribution associated with a classification problem
- then the label  $\mathbf{y}$  represents a probability distribution
  - \*  $\mathbf{y} = p(\mathbf{x})$  is the desired probability distribution
  - \*  $\hat{\mathbf{y}} = h(\mathbf{x})$  is the estimated probability distribution
- from (247), the cross-entropy cost function associated with data  $(\mathbf{x}, \mathbf{y})$  and hypothesis  $h$  is

$$c_h = -\sum_i y_i \log_2 h_i(\mathbf{x}) \quad (19)$$

- with *disjoint classification* problems only a single outcome is valid, &  $\mathbf{y}$  becomes a one-hot vector
  - \* the cross-entropy simplifies to

$$c_h \triangleq -\log_2 h_i(\mathbf{x}) \quad (20)$$

\* the above cost coincides with the log-likelihood cost function (18)

5) Multi-class SVM loss function:

- consider a margin of  $\Delta$ , as described in Section XIII
- given  $(\mathbf{x}_i, y_i)$ , define the score of the  $j^{\text{th}}$  class as

$$s_j \triangleq f_j(\mathbf{x}_i, \mathbf{w}), \quad (21)$$

- the *multi-class SVM loss*, or *hinge loss*, for data  $(\mathbf{x}_i, y_i)$  is defined as,

$$c_i \triangleq \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \quad (22)$$

6) The *cosine similarity metric*,  $\in [1, 1]$ , is defined as

$$c_{h,f} \triangleq \frac{\mathbf{y}^{(h)} \cdot \mathbf{y}^{(f)}}{\|\mathbf{y}^{(h)}\| \|\mathbf{y}^{(f)}\|} \quad (23)$$

7) The *Jaccard coefficient* is often used when the objects represent sets,

$$c(S1, S2) \triangleq \frac{|S1 \cap S2|}{|S1 \cup S2|} \quad (24)$$

• In-sample versus out-of-sample cost functions:

- using (14), the *in-sample cost function*  $C_{\text{in}}(h)$ , over  $N$  samples, of using hypothesis  $h$  is defined as

$$C_{\text{in}}(h) \triangleq \frac{1}{N} \sum_{n=1}^N c(\mathbf{y}_n^{(h)}, \mathbf{y}_n) \quad (25)$$

- the *out-of-sample cost function*  $C_{\text{out}}(h)$  is defined as

$$C_{\text{out}}(h) \triangleq E_{\mathbf{x}, \mathbf{y}}[c(h(\mathbf{X}), \mathbf{Y})] \quad (26)$$

## B. Neuron models

• A neuron:

- a neuron is also referred to as a *unit* or a *node*
- a neuron takes a  $(d + 1)$  dimensional input vector  $\mathbf{x}$ , and outputs a scalar  $y$
- a neuron is some linear circuit followed by a non-linear function,

$$\mathbf{x} \xrightarrow{\mathbf{w}} s \triangleq \mathbf{w}^T \mathbf{x} \xrightarrow{\theta} y \triangleq \theta(s) \quad (27)$$

\* the linear portion is modeled with the parameter vector  $\mathbf{w}$ , called the *weight*

- *linearity in the weights* means the weights have linear dependency
- $s$  is a scalar called the *signal* or the *score*

\* the non-linearity is introduced through the *activation function*  $\theta$

- a neuron can also be defined in a more setting

$$y \triangleq \theta(\mathbf{x}, \mathbf{w}) \quad (28)$$

- most neurons considered here have a  $(d + 1)$  dimensional  $\mathbf{w}$ , with  $x^0 \triangleq 1 \Rightarrow w^0$  is the *bias*
- a hypothesis  $h$  is a function of weights  $\mathbf{w}$

• Non-linearities & Activation functions:

- deeper networks can generate more sophisticated non-linear transformations
- various activation functions can be applied on  $s$  (27)
- in general, activations need to be monotonically non-decreasing, and mostly differentiable

• Feature transform:

- in addition to activation functions, non-linearities could also be introduced by non-linear mapping of the input data
- a *feature transform*  $\Phi$ , is a non-linear transformation

$$\mathbf{x} = (x_0, \dots, x_d) \xrightarrow{\Phi} \mathbf{z} = (z_0, \dots, z_{\tilde{d}}) \quad (29)$$

where  $\mathbf{z} \in \mathbb{R}^{\tilde{d}+1}$  is in the *feature space*

- strictly speaking  $\Phi$  may not satisfy the constraints of a function
- the linear summation (27) generalizes to

$$\tilde{\mathbf{w}}^T \mathbf{z} = \tilde{\mathbf{w}}^T \Phi(\mathbf{x}) \quad (30)$$

- linearity in weights is retained since non-linearities are applied on the inputs, not the weights,
- if  $\tilde{d} > d$ , then we might have generalization concerns
- if  $\tilde{d} < d$ , then we are acting as a learning machine, and should account for that for generalization
- latest approaches rely on deeper NN rather than feature transformations to obtain sophisticated non-linearities

- Identity, linear classifiers & cybernetics:

- in certain networks, a fraction of the neurons do not have activation functions
- in this case,  $\theta$  collapses to the identity function, and the *linear regression* hypotheses is obtained, see Section XV

$$\theta(s) = s = \mathbf{w}^T \mathbf{x} \quad (31)$$

- if a single neuron is used, a hypothesis can then be determined by the  $(d+1)$  parameters  $\mathbf{w} = \{w_i\}$ , and the signal  $s$  (27) itself is used to classify inputs
- \* *cybernetics* implied learning  $\mathbf{w}$  of such a model

- List of activation functions:

- 1) Perceptron:

- the *McCulloch-Pitts Neuron*, 1943, is a binary classifier, obtained by setting the activation function to be the sign() of the input value

$$\theta_{MP}(s) \triangleq \text{sign}(s) = \text{sign}(\mathbf{w}^T \mathbf{x}) \quad (32)$$

- a *perceptron* [2] is a McCulloch-Pitts Neuron that can learn its weights
- since the derivative of this activation vanishes, it is not a useful activation for learning
- based on Section XI a complexity measure can be associated with the perceptron
- \* theorem:  $d_{VC} = d + 1$ , which coincides with the number of parameters  $w_i$

- 2) The *softsign* function is the soft version of the McCulloch-Pitts Neuron:

$$\theta_{SS}(s) \triangleq \frac{s}{1 + |s|}, \quad (33)$$

- 3) The logistic sigmoid function:

- the *logistic sigmoid* function:

$$\theta_L(s) \triangleq \frac{e^s}{1 + e^s} = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \quad (34)$$

- the derivative is

$$\dot{\theta}_L(s) = \theta_L(s) (1 - \theta_L(s)) \quad (35)$$

- $\theta_L(s) \geq 0 \Rightarrow x_i \geq 0$ , (i.e. not centered around zero) implies the changes of  $w_i$ , due to gradient, are all positive or negative, see sub-section II-C
- furthermore, saturated regions have no gradient information
- useful for gating functions

- 4) Stochastic neuron & logistic regression:

- a boolean variable  $y \in \{0, 1\}$  can be referred to as an *indicator variable*
- in *logistic regression*, the goal is to determine the conditional probability of an indicator variable,  $P(y = 1|\mathbf{x})$
- since the output of (34) is  $\in [0, 1]$ , this output can be reinterpreted as a conditional probability,  $P(y = 1|\mathbf{x})$
- a *stochastic neuron* has the same activation function as the the logistic sigmoid,  $\theta_L(s)$  that implements a logistic regression
- \* the goal is to estimate  $P(y = 1|\mathbf{x})$  rather than  $y$
- this is similar to obtaining soft data in data communication
- in general  $P(y = 1|\mathbf{x})$  is not a linear function with respect to  $\mathbf{x}$
- logistic regression converts the problem to a linear regression (with infinite range), by *assuming* the transformation

$$\begin{aligned} \log \frac{P(y = 1|\mathbf{x})}{1 - P(y = 1|\mathbf{x})} &= \mathbf{w}^T \mathbf{x} = s \Rightarrow \\ P(y = 1|\mathbf{x}) &= \frac{e^s}{1 + e^s}, \end{aligned} \quad (36)$$

- note that this coincides with (34), and that

$$\begin{aligned} \theta(-s) &= \frac{1}{1 + e^s} \\ &= 1 - \frac{e^s}{1 + e^s} \\ &= 1 - \theta(s) \\ &= P(y = 0|\mathbf{x}) \end{aligned} \quad (37)$$

- both  $P(y = 1|\mathbf{x})$  &  $P(y = 0|\mathbf{x})$  can be expressed by the unified form

$$P(y|\mathbf{x}) = \theta(y\mathbf{w}^T\mathbf{x}) \quad (38)$$

- 5) The *tangent hyperbolic* function:

$$\theta_T(s) \triangleq \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} \quad (39)$$

- popular in 1980s
- the range is  $\theta_T \in [-1, 1]$
- $\theta_T(s)$  can be obtained from  $\theta_L$  (34) by the transformation  $y = 2x - 1$

- 6) The *hard tanh* function:

$$\theta_{HT}(s) \triangleq \begin{cases} -1, & \text{if } s < -1 \\ s, & \text{if } -1 \leq s \leq 1 \\ 1, & \text{if } s > 1 \end{cases} \quad (40)$$

- 7) The *rectified linear unit*, or ReLU:

$$\begin{aligned} \theta_R(s) &\triangleq \max(0, s) \\ &= \max(0, \mathbf{w}^T\mathbf{x}) \end{aligned} \quad (41)$$

- currently a popular activation function
- typically learns faster in networks with many layers
- when the signal stays negative, we have a *dead ReLU*
  - \* positive bias initialization addresses the dead ReLU concern

- 8) The *parametric ReLU*, or PReLU:

$$\theta_{PR}(s) \triangleq \max(\alpha s, s) \quad (42)$$

- setting  $\alpha = 0.01$  gives the *leaky ReLU*:

$$\theta_{LR}(s) \triangleq \max(0.01s, s) \quad (43)$$

- 9) The *exponential linear unit*, or ELU:

$$\theta_{ELU}(s) \triangleq \begin{cases} s, & \text{if } s > 0 \\ \alpha(e^s - 1), & \text{if } s \leq 0 \end{cases} \quad (44)$$

- 10) The *softplus* function is smooth (soft) version of (41):

$$\theta_{SP}(s) \triangleq \log(1 + \exp(s)) \quad (45)$$

- 11) The *Maxout* activation:

$$\theta_{MO}(\mathbf{x}) \triangleq \max(\mathbf{w}_1^T\mathbf{x}, \mathbf{w}_2^T\mathbf{x}) \quad (46)$$

- 12) The *softmax* function, with  $1 \leq j \leq K$  output nodes, is

$$\theta_{SM}(j; s_1^K) \triangleq \frac{e^{s_j}}{\sum_{k=1}^K e^{s_k}} \quad (47)$$

where the  $j^{\text{th}}$  node generates the probability of the  $j^{\text{th}}$  event

- since  $\theta(j) \geq 0$  and  $\sum_j \theta(j) = 1$ , we can also interpret  $\{\theta(j)\}$  as probabilities
- softmax can be useful with classification problems involving disjoint classes
- the log-likelihood cost function is often used with softmax activation

- Log-likelihood criterion on stochastic neuron:

- substituting (38) into (18)

$$\begin{aligned} c_{\mathbf{w}} &= \log \frac{1}{\theta(y_{\mathbf{w}}^T\mathbf{x})} \\ &= \log(1 + e^{-y_{\mathbf{w}}^T\mathbf{x}}) \end{aligned} \quad (48)$$

- analytic solution to  $\nabla C_{\text{in}}(\mathbf{w}) = 0$  is not feasible, but gradient descent algorithms can be used
- with classification problems, this cost metric converges faster than the mean square error metric



### C. Update Rules

- Perceptron Learning Algorithm (PLA):

- pick any  $(\mathbf{x}, y)$  that is misclassified, and make the update

$$\begin{aligned} \mathbf{w}(t+1) &= \mathbf{w}(t) + y\mathbf{x} \Rightarrow \\ y\mathbf{w}^T(t+1)\mathbf{x} &= y\mathbf{w}^T(t)\mathbf{x} + y^2\|\mathbf{x}\|^2 \Rightarrow \\ y\mathbf{w}^T(t+1)\mathbf{x} &> y\mathbf{w}^T(t)\mathbf{x} \end{aligned} \quad (49)$$

- since its a misclassified pair

$$y\mathbf{w}^T(t)\mathbf{x} < 0 \quad (50)$$

- combining (49) with (50) implies  $\mathbf{w}$  is moving in the right direction
- theorem: if the input data is linearly separable then the PLA will converge with  $C_{\text{in}} = 0$

- Pocket algorithm:

- when input data-set is not linearly separable, then the PLA is not stable
  - \* the problem of choosing  $\mathbf{w}$  that minimizes  $C_{\text{in}}$ , i.e. error rate, is NP-hard
- the *pocket algorithm* is a variation of PLA, that keeps "in its pocket" the best  $\mathbf{w}$ 
  - \* in other words, pick the best  $\mathbf{w}$  rather than the last  $\mathbf{w}$
  - \*  $C_{\text{in}}$  needs to be computed in every step

- Gradient descent:

- gradient descent can be applied to a differentiable graph whether or not there is an analytic solution
- let  $\nabla C_{\text{in}}(\mathbf{w}) = \nabla_{\omega} C_{\text{in}}(\mathbf{w})$  be the gradient of the cost function  $C_{\text{in}}(\mathbf{w})$  with respect to  $\mathbf{w}$
- gradient descent is based on the observation that  $C_{\text{in}}(\mathbf{w})$  decreases fastest in the direction of  $-\nabla C_{\text{in}}(\mathbf{w})$
- define  $\hat{\mathbf{v}}$  to be a unit vector in the direction of *steepest descent*

$$\hat{\mathbf{v}} \triangleq -\frac{\nabla C_{\text{in}}(\mathbf{w})}{\|\nabla C_{\text{in}}(\mathbf{w})\|} \quad (51)$$

- *gradient descent* is an iterative solution, where at time  $(t+1)$ , the weights are adjusted as

$$\begin{aligned} \mathbf{w}(t+1) &= \mathbf{w}(t) + \delta\mathbf{w} \\ &= \mathbf{w}(t) + \eta_t \hat{\mathbf{v}} \end{aligned} \quad (52)$$

where  $\eta_t$  is a hyper-parameter called the *learning rate*

- to see why gradient descent proceeds in direction of  $-\nabla C_{\text{in}}(\mathbf{w})$ , apply Taylor expansion on  $C_{\text{in}}(\mathbf{w})$

$$\begin{aligned} \Delta C_{\text{in}} &\triangleq C_{\text{in}}(\mathbf{w}(t) + \delta\mathbf{w}) - C_{\text{in}}(\mathbf{w}(t)) \\ &\approx \nabla C_{\text{in}}(\mathbf{w}(t)) \cdot \delta\mathbf{w} \end{aligned} \quad (53)$$

- choosing  $\delta\mathbf{w} = \hat{\mathbf{v}}$  and  $\eta_t = 1$

$$\begin{aligned} \Delta C_{\text{in}} &= \nabla C_{\text{in}}(\mathbf{w}(t)) \cdot \hat{\mathbf{v}} \\ &= -\nabla C_{\text{in}}(\mathbf{w}(t)) \cdot \frac{\nabla C_{\text{in}}(\mathbf{w}(t))}{\|\nabla C_{\text{in}}(\mathbf{w}(t))\|} \\ &\geq -\|\nabla C_{\text{in}}(\mathbf{w}(t))\| \end{aligned} \quad (54)$$

- \* steepest descent is achieved when we have equality in (53), i.e. when

$$\delta\mathbf{w} = \hat{\mathbf{v}} = -\frac{\nabla C_{\text{in}}(\mathbf{w}(t))}{\|\nabla C_{\text{in}}(\mathbf{w}(t))\|} \quad (55)$$

- Local and global minima:

- if the cost function is *convex* then there is no *local minima* but only a single *global minimum*
- in practice, poor local minima are rarely a problem with large networks, or in higher dimensional space
- instead, the landscape is packed with a combinatorially large number of saddle points where the gradient is zero in many but not all dimensions
- practically, it does not much matter which of these saddle points the algorithm ends up being

- Batch gradient descent:

- it is desirable to get large steps when we are far from minimum, & small steps when close to the minimum
- can choose learning rate  $\eta_t$  to be proportional to the gradient

$$\eta_t = \eta \|\nabla C_{\text{in}}(\mathbf{w}(t))\| \quad (56)$$

- substituting (55, 56) into  $\delta \mathbf{w}(t)$

$$\begin{aligned}\delta \mathbf{w}(t) &= \eta_t \hat{\mathbf{v}} \\ &= -\eta \|\nabla C_{\text{in}}(\mathbf{w}(t))\| \frac{\nabla C_{\text{in}}(\mathbf{w}(t))}{\|\nabla C_{\text{in}}(\mathbf{w}(t))\|} \\ &= -\eta \nabla C_{\text{in}}(\mathbf{w}(t))\end{aligned}\tag{57}$$

- it follows that

$$\boxed{\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla C_{\text{in}}(\mathbf{w}(t))}\tag{58}$$

- this algorithm is sometimes referred to as the *batch gradient descent* or *fixed learning rate gradient descent algorithm*
- note that a single update is generated from the entire data set
  - \* batch gradient descent can be very slow since the gradients for the whole dataset are calculated to perform just one update
  - \* batch gradient descent are intractable for datasets that don't fit in memory
- Batch gradient descent for logistic regression:
  - the gradient of (48) is

$$\begin{aligned}\nabla C_{\text{in}}(\mathbf{w}(t)) &= -\frac{1}{N} \sum_{n=1}^N \frac{y_n \mathbf{x}_n e^{-y_n \mathbf{w}^T \mathbf{x}_n}}{(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n})} \\ &= -\frac{1}{N} \sum_{n=1}^N \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}} \\ &= -\frac{1}{N} \sum_{n=1}^N y_n \mathbf{x}_n \theta(-y \mathbf{w}^T \mathbf{x}_n)\end{aligned}\tag{59}$$

- substituting (59) into (57)

$$\delta \mathbf{w}(t) = \frac{1}{N} \sum_{n=1}^N y_n \mathbf{x}_n \theta(-y \mathbf{w}^T \mathbf{x}_n)\tag{60}$$

- Stochastic gradient descent (SGD):
  - *stochastic gradient descent* (SGD), is the sequential version of batch gradient descent, where an update is generated for each sample point  $(\mathbf{x}_n, y_n)$ 
    - \*  $(\mathbf{x}_n, y_n)$  is picked uniformly at random, hence the name 'stochastic'
  - compared to batch gradient descent, SGD is simpler, faster and, due to randomization, is less prone to getting trapped in a local minima
  - the rule of thumb is to use  $\delta \approx 0.1$
  - SGD complicates convergence to the exact minimum, as SGD will keep overshooting
- Mini-batch gradient descent:
  - *mini-batch gradient descent* implies a single gradient update is generated per mini-batch of data  $\{(\mathbf{x}_n, y_n)\}$
  - each element of the mini-batch is picked uniformly at random
  - mini-batch gradient descent takes the best of both SGD and batch gradient descent
    - \* compared to SGD, it reduces the variance of the parameter updates, which can lead to more stable convergence
  - common mini-batch sizes range between 50 and 256
  - the term SGD usually is employed also when mini-batches are used
- Learning rate with annealing:
  - with *step decay*, the learning rate is reduced by some factor as a function of training iterations
  - the *exponential decay* and the *1/t-decay* annealing functions are respectively

$$\begin{aligned}\eta_t &= \eta_o \exp(-kt) \\ \eta_t &= \frac{\eta_o}{1 + kt}\end{aligned}\tag{61}$$

where  $\eta_o, k$  are hyper-parameters &  $t$  is the iteration number

- Hessian technique:
  - applying Taylor's extension, as in (53), but including second order terms

$$\Delta C_{\text{in}}(\mathbf{w} + \delta \mathbf{w}) \approx \nabla C_{\text{in}} \cdot \delta \mathbf{w} + \frac{1}{2} \delta \mathbf{w}^T H \delta \mathbf{w}\tag{62}$$

where the *Hessian matrix*  $H$ , is defined as

$$H_{jk} \triangleq \frac{\partial^2 C_{\text{in}}}{\partial w_j \partial w_k} \quad (63)$$

- differentiating (62) with respect to  $\delta \mathbf{w}$ , and setting it to zero

$$\begin{aligned} 0 &= \nabla C_{\text{in}} + H \delta \mathbf{w} \Rightarrow \\ \delta \mathbf{w} &= -H^{-1} \nabla C \end{aligned} \quad (64)$$

- this approach to minimizing a cost function is known as the *Hessian technique* or the *Hessian optimization*
  - \* it is also referred to as the *Newton's method*
- Hessian methods converge in fewer steps than standard gradient descent
- furthermore, the Hessian approach could avoid many pathologies that can occur in gradient descent
- these benefits come at the expense of significant additional complexity
- two simplifications are BGFS and limited-memory BFGS (L-BFGS)

- Momentum based gradient descent:

- the momentum technique modifies gradient descent by introducing two new concepts:
  - 1) Introducing the notion of "velocity"  $\mathbf{v}_t$ , for the parameters we're trying to optimize
    - \* the gradient acts to change the velocity, not (directly) the "position"  $\mathbf{w}_t$ , in much the same way as physical forces change the velocity, that only indirectly affect position
  - 2) Introduces the notion of "friction" which tends to gradually reduce the velocity.
- the update rules are summarized as follows

$$\begin{aligned} \mathbf{v}_t &= \mu \mathbf{v}_{t-1} - \eta \nabla C \\ \mathbf{w}_t &= \mathbf{w}_{t-1} + \mathbf{v}_t \end{aligned} \quad (65)$$

where  $\mu$  is the *momentum coefficient*, that controls the amount of damping or friction in the system

\*  $\mu = 0$  corresponds to the gradient descent method

\*  $0.5 \leq \mu \leq 0.9$  is common

- the momentum technique is more practical than the Hessian technique
- the momentum technique often speeds up learning and is commonly used

- Nesterov accelerated gradient:

- *Nesterov accelerated gradient* (NAG), is a momentum based technique, where the gradient is evaluated at  $\mathbf{w} + \mu \mathbf{v}$  rather than at  $\mathbf{w}$

- Adagrad:

- *Adagrad* adapts the learning rate to the parameters
- so far, at time  $t$ , all the parameter  $w_i$  used the same learning rate
- define  $G_t$  to be a diagonal matrix, where  $G_{t,ii}$  is the sum of the squares of the gradients w.r.t.  $w_i$  up to time step  $t$
- the update rule for Adagrad is

$$w_{t,i+1} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla C(w_i) \quad (66)$$

where  $\epsilon \sim 10^{-8}$  is a smoothing term that avoids division by zero

- compared to SGD, the learning rate is adaptively scaled by  $\sqrt{G_{t,ii} + \epsilon}$
- without the square root operation, the algorithm performs much worse
- Adagrad eliminates the need to manually tune the learning rate
- most implementations use a default value of  $\eta = 0.01$
- Adagrad also implicitly implements momentum based updates (why?)
- Adagrad's main weakness is its accumulation of the squared gradients in the denominator which eventually makes infinitesimally small

- Adadelta:

- *Adadelta* is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate
- instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size  $L$

- RMSProp:

- *RMSProp* is an adaptive gradient technique where the learning rate is tailored for each parameter

- if the gradient at time  $t$  is  $\nabla_t C$ , then the learning rate, at time  $t$  is

$$\begin{aligned} s_t &= \alpha s_{t-1} + (1 - \alpha) \|\nabla_t C\|^2 \\ \mathbf{w}_t &= \mathbf{w}_{t-1} + \eta \frac{\nabla_t C}{\sqrt{s_t}} \end{aligned} \tag{67}$$

- Adam:

- in the momentum technique (65),  $\mathbf{v}_t$  is linear in  $\mathbf{g}_t$ , whereas in RMSProp (67),  $s_t$  is quadratic in  $\mathbf{g}_t$
- *Adam* combines features from both of these techniques

$$\begin{aligned} \mathbf{v}_t &= \mu \mathbf{v}_{t-1} + (1 - \mu) \nabla_t C \\ s_t &= \alpha s_{t-1} + (1 - \alpha) \|\nabla_t C\|^2 \\ \mathbf{w}_t &= \mathbf{w}_{t-1} + \eta \frac{\mathbf{v}_t}{\sqrt{s_t}} \end{aligned} \tag{68}$$

- its is recommended to use  $\mu = 0.9$ ,  $\alpha = 0.999$

### III. GENERALIZATION

- Generalization & feasibility of learning:
  - *generalization* is the capability of predicting a function for unobserved data
    - \* learning an arbitrary boolean function is not feasible by just observing a subset of the function
  - in machine learning, generalization is done based on probabilistic inference, Section I-A
  - learning is feasible under the assumption that the training & testing samples share the same distribution
- Two goals of learning: choose hypothesis  $g \in \{h_i\}$ :
  - 1) to *approximate*, or minimize  $C(h)_{\text{in}}$  (25), and
  - 2) to generalize, i.e. from (25, 26),

$$C(g)_{\text{out}} \approx C(g)_{\text{in}} \quad (69)$$

where  $\approx$  means  $C_{\text{in}}(h) = C_{\text{out}}(h)$  is *probably approximately correct*, or P.A.C.

- Testing:
  - *testing* is performed after a hypothesis  $g$  is chosen
  - testing is performed as a *final* evaluation of accuracy
  - testing is performed on a different data set than the training set
- Overfitting:
  - *overfitting* is the phenomenon where the in-sample data is fitted more than is warranted
  - small  $C_{\text{in}}$  is observed at the expense of  $C_{\text{out}}$
  - when overfitting, the network is learning about the peculiarities of the training set
  - when overfitting, the network is
    - \* more memorizing than learning
    - \* not learning to generalize from trend
  - a simple approach to contain overfitting is to *early termination*, where training stops if no improvements are observed
- Overfitting dependencies:
  - overfitting gets worst as
    - \* stochastic noise variance  $\sigma_n^2$  increases
    - \* target function complexity increases
      - for example if target function is modeled by a polynomial of higher degree
    - \* data size  $N$  decreases
      - In many situations the data size should decide the complexity of the hypothesis family chosen, not the assumed complexity of the target function
- Deterministic noise:
  - as defined in (191), deterministic noise is the part of the target function  $f$  that  $\mathcal{H}$  cannot capture
  - deterministic noise is defined as
 
$$f(\mathbf{x}) - g(\mathbf{x}) \quad (70)$$
  - deterministic noise occurs when
    - \*  $\mathcal{H}$  does not capture important aspects of  $f$ , i.e.  $\mathcal{H}$  is oversimplified, or when
    - \* target function complexity  $Q_f$  is large and  $N$  is not large enough
  - unlike stochastic noise, deterministic noise
    - \* depends on  $\mathcal{H}$ , and
    - \* is repeatable
  - given  $\mathcal{H}$ , both noise sources have similar impact on the learning process
- VC generalization bound:
  - VC bound provide a lower bound on  $C_{\text{out}}(g) - C_{\text{in}}(g)$  as a function of sample size  $N$
  - see Appendix XI, and (189)
  - in general, the VC bounds are very loose
- Bias & variance model:
  - an alternative method to decompose  $C_{\text{out}}(g)$  is the *bias & variance method* reviewed in Appendix XII
- Two common approaches to address overfitting:
  - 1) Regularization
  - 2) Validation

### A. Regularization

- General remarks:
  - *regularization* constrains the learning algorithm to improve out-of-sample error
    - \* constraining implies lowering  $d_{VC}$
  - regularization is as much an art as it is science
    - \* there is no systematic understanding, only heuristics
  - in general, regularization reduces noise more than it limits the signal

- The augmented metric:
  - the goal of regularization is to minimize an *augmented metric*

$$C_{\text{aug}}(h) = C_{\text{in}}(h) + \Omega(h) \quad (71)$$

rather than  $C_{\text{in}}(h)$  alone, see (188)

- $\Omega(h)$  is the *regularizer*, or the *regularization term*
  - \*  $\Omega(h)$  depends on  $h$  rather than  $\mathcal{H}$
  - \*  $\Omega(h)$  is a proxy for overfit penalty
- the augmented metric  $C_{\text{aug}}(h)$  is a proxy for  $C_{\text{out}}(h)$
- Regularization methods:
  - 1) L2 regularization, or weight decay
    - other quadratic forms include
      - \* L2 regularization with importance factors
      - \* Tikhonov regularization
  - 2) L1 regularization
  - 3) Soft weight elimination
  - 4) Soft targets to train
  - 5) Dropout
  - 6) Artificially expand the data set
- Hard & soft order constraint:
  - for definiteness, consider a linear regression problem on the  $\mathcal{Z}$  space (30), of polynomials
  - a polynomial hypothesis with lower degree (simpler) may result in better out-of-sample error
    - \* excluding all polynomials higher than a degree threshold is known as *hard order constraint*
  - an example of a *soft order constraint* is the constraint that polynomial coefficients satisfy

$$\sum_{q=0}^Q w_q^2 \leq C \Rightarrow \mathbf{w}^T \mathbf{w} \leq C \quad (72)$$

\* this is similar to FIR with normalized weight energy constraint

- L2 regularization:
  - the augmented error metric associated with (72) uses Lagrange multipliers to solve this constrained linear regression problem

$$C_{\text{aug}}(\mathbf{w}) = C_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N} \mathbf{w}^T \mathbf{w} \quad (73)$$

- with this augmented metric, the network prefers to learn small weights, all other things being equal
  - \* the smallness of the weights means that the behaviour of the network won't change too much if we change a few random inputs here and there
  - \* this implies less noisy behaviour and simpler models
- $\lambda$  is called the *regularization parameter*
- minimizing (73) is called *L2 regularization*, or *weight decay*, since taking the gradient of (73) gives the additional term that is proportional to  $\mathbf{w}$
- for the linear regression case, the resulting weight vector becomes (compare to (250))

$$\mathbf{w}_{\text{reg}} = (Z^T Z + \lambda I)^{-1} Z^T \mathbf{y} \quad (74)$$

- for gradient descent, the updated weights have the form

$$w \rightarrow \left(1 - \frac{\eta \lambda}{N}\right) w - \eta \frac{\partial e}{\partial w} \quad (75)$$

- Weight decay of bias terms:
  - having a large bias doesn't make a neuron sensitive to its inputs
  - no need to worry about large biases enabling our network to learn the noise in our training data
  - allowing large biases gives our networks more flexibility in behaviour
    - \* in particular, large biases make it easier for neurons to saturate, which is sometimes desirable
  - for these reasons, usually bias terms are not regularized

- Alternative weight constraints:

- the constraint (72) can be generalized to,

$$\sum_{q=0}^Q \gamma_q w_q^2 \leq C \quad (76)$$

where the *importance factor*  $\{\gamma_q\}$  are parameters that amplify/attenuate certain degrees

- in neural networks, different layers may get different importance factors
- the *Tikhonov regularizer*

$$\mathbf{w}^T \Gamma^T \Gamma \mathbf{w} \leq C \quad (77)$$

considers non-diagonal quadratic form, and captures relationships within  $w_i$  by using matrix  $\Gamma$

- L1 regularization:

- the augmented error metric of L2 regularization (73), is modified to

$$C_{\text{aug}}(\mathbf{w}) = C_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N} \sum_n |w_n| \quad (78)$$

- the weights are then updated as

$$w \rightarrow w - \frac{\eta \lambda}{N} \text{sign}(w) - \eta \frac{\partial e}{\partial w} \quad (79)$$

- since derivative is not defined at  $w = 0$ , apply the un-regularized rule for stochastic gradient, i.e. set  $\text{sign}(0) = 0$
- compared to (75), the decay in weight is constant, whereas in L2 regularization it is proportional  $w$
- L1 regularization tends to concentrate the weight of the network in a relatively small number of high-importance connections, while the other weights are driven toward zero

- Hard & soft weight elimination:

- with *hard weight elimination*, weights with values less than a threshold are eliminated
- with *soft weight elimination*, the regularizer could be defined as

$$\Omega(\mathbf{w}) \triangleq \sum_{i,j,l} \frac{(w_{ij}^{(l)})^2}{\beta^2 + (w_{ij}^{(l)})^2} \quad (80)$$

- \* for  $\|\mathbf{w}^{(l)}\| \ll \beta$ , numerator dominates and the regularizer reduces to weight decay (73)
- \* for  $\|\mathbf{w}^{(l)}\| \gg \beta$ ,  $\Omega(\mathbf{w})$  is a constant and no updates are generated from the regularizer
- \* in summary, the regularizer attenuates small weights but not the large weights

- Dropout & dropconnect:

- *dropout* modifies the network itself rather than modifying the cost metric
- more specifically, for each training example, forward propagation involves randomly deleting any hidden neuron with probability  $p$ 
  - \* the error is then backpropagated only through the remaining activations
  - \*  $p = 0.5$  is common
- we can think of dropout as a way of making sure that the model is robust to the loss of any individual piece of evidence
- dropout has been useful in training large, deep networks, where the problem of overfitting is often acute
- inverted dropout: after dropout the signal levels should be attenuated by  $p$ , (not  $\sqrt{p}$ ?) since more inputs are contributing
- *dropconnect* generalizes dropout by randomly dropping the weights rather than the activations

- Soft targets:

- *soft target* means that, the label is a probability distribution rather than a one-hot vector
- when using a small network, use some combination of soft target, obtained from a larger circuit, and hard target to train data,
- soft targets are good regularizers

- Practical rule for regularization:

- constrain learning towards smoother hypothesis, since noise is high frequency
- in general smaller weights result in smoother hypothesis

## B. Validation

- The concept:
  - partition available training data, that was originally designated to training, to two parts

$$\mathcal{D} = \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{train}} \quad (81)$$

where the validation sample size is  $|\mathcal{D}_{\text{val}}| = K$ , and the re-allocated *training* data size as  $|\mathcal{D}_{\text{train}}| = N - K$

- \* validation comes at the expense of training
- \* validation is used to optimize various hyper-parameters
- similar to the in-sample error (25), validation error is defined as

$$C_{\text{val}}(h) \triangleq \frac{1}{K} \sum_{n=1}^K e(h(\mathbf{x}_n), y_n) \quad (82)$$

- whereas regularization predicts  $C_{\text{out}}$  indirectly through  $\Omega(h)$ , *validation* directly measures  $C_{\text{out}}$
- A validation procedure:
  - generate a hypothesis  $g^-$  from  $N - K$  samples
  - measure  $C_{\text{out}}(g^-)$  from remaining  $K$  samples
  - generate a hypothesis  $g$  from all  $N$  samples
  - report  $g$  with measure  $C_{\text{out}}(g^-)$
  - as a rule of thumb, use  $K \approx \frac{N}{5}$
- Early stopping:
  - with *early stopping*, validation data is used to monitor overfitting, picking the hypothesis where  $C_{\text{out}}$  estimate is minimized
  - validation then becomes an integral part of training
- Training, testing and validating:
  - when training,  $C_{\text{in}}$  is optimistically (deceptively) biased
  - since testing does not alter the outcome of the learning process,  $C_{\text{test}}$  is unbiased
  - validating is similar to testing but can influence the outcome of the learning process because of feedback
    - \* then validation becomes optimistically biases as in training
    - \* i.e.  $C_{\text{val}}$  becomes a biased estimate of  $C_{\text{out}}$
  - we want  $C_{\text{val}}$  to be only slightly contaminated to be useful
- Dependence of  $g \in \mathcal{H}$  on data:
  - $g$  is a strong function of peculiarities of the training data
  - $g$  should be a weak function of the peculiarities of validation data
  - $g$  is independent of the peculiarities of the test data
- $M$  models:
  - many validation approaches can be interpreted as choosing one of  $M$  hypotheses, where each hypothesis was obtained from some model and training data
  - then validation coincides with training, and bounds similar to (180, 185) can be applied
- Cross-validation:
  - so far the goal was to achieve

$$C_{\text{out}}(g) \approx C_{\text{out}}(g^-) \approx C_{\text{val}}(g^-) \quad (83)$$

where for the first approximation to hold,  $K$  needed to be small, while for the second approximation  $K$  needed to be large

- in theory can run training  $N$  separate times, where at each attempt the  $n^{\text{th}}$  sample point is removed for validation purposes
- the *cross-validation error* is then defined as the average of the  $N$  validation errors

$$E_{\text{cv}} \triangleq \frac{1}{N} \sum_{n=1}^N e_n \quad (84)$$

- even though
  - \* each error term is from a different hypothesis, and
  - \* the error terms are correlated
- effectively, the error samples are almost uncorrelated



- in summary, effectively  $K = 1$  was used for first approximation of (83), and  $K = N$  was used for second
- $R$ -fold validation:
  - cross validation, is not practical as described above, since it introduces  $N$  times amplification in training time
  - *R-fold validation* implies partitioning sample data into  $R$  subsets rather than  $N$
  - training sessions reduce from  $N$  to  $R$
  - rule of thumb is to use 10-fold cross-validation

#### IV. FEEDFORWARD NETWORKS WITH SUPERVISION

- Historical overview:
  - *cybernetics* in the 1940s-1960s
    - \* biological learning
    - \* perceptron
  - artificial neural networks (ANN) or *connectionism*, in the 1980s-1990s
    - \* parallel distributed processing (Rumelhart et al., 1986 ; McClelland et al., 1995)
    - \* a large number of simple computational units can achieve intelligent behavior when networked together
    - \* back propagation (Rumelhart et al., 1986 )
  - *deep learning* starting in 2006, see Section IV-D
    - \* pioneered by Georey Hinton at University of Toronto, Yoshua Bengio at University of Montreal, and Yann LeCun at New York University
- Universality:
  - since a perceptron can model a NAND, a network of perceptrons can model any boolean function
  - consider an arbitrary continuous function  $f(\mathbf{x})$ , with range within the activation function
  - then for any  $\epsilon > 0$ , there exist a NN  $g(\mathbf{x})$ , with a single hidden layer, such that
 
$$|g(\mathbf{x}) - f(\mathbf{x})| < \epsilon \quad (85)$$

for all inputs  $\mathbf{x}$

    - the underlying concept is for the hidden layers to generate a basis function
    - the result holds for any activation function  $\theta(z)$ , where  $\theta(\pm\infty)$  are distinct and well-defined
    - the question is not whether any particular function is computable, but rather what's a good way to compute the function
- Feedforward networks:
  - feedforward multilayer networks are the most popular ANNs
  - such networks include an *input layer*, an *output layer*, and *hidden layers*
  - the role of the *hidden layers* can be interpreted as distorting the input in a non-linear way so that categories become linearly separable by the last layer

##### A. Multilayer perceptrons

- General remarks:
  - *multilayer perceptron*, or MLP, is a feedforward network where each layer is fully connected (FC) to the previous layer
  - proposed in the 1980s
  - the name is misnomer since the unit of MLP is not the perceptron
  - MLPs use the supervised backpropagation algorithm to learn as will be discussed shortly
  - MLPs do not learn well in the presence of many hidden layers
- Notation:
  - a network is  $L$  layers with  $l \in \{0, 1, \dots, L\}$ 
    - \* the input layer corresponds to  $l = 0$  and does not count as a layer
    - \* the output layer corresponds to  $l = L$  which determines the value of the function
  - layer  $l$  has *dimension*  $d^{(l)}$ , which means it has  $(d^{(l)} + 1)$  *nodes* (or *units*)
    - \* every layer has a *bias node*, labelled by 0
  - connections are between layers  $(l - 1)$  and  $l$ 
    - \*  $\mathbf{W}^{(l)} = \{w_{ij}^{(l)}\}$  is the weight from node  $i$  in layer  $(l - 1)$  to node  $j$  in layer  $l$
    - \* the signal and the output of node  $j$  in layer  $l$  are denoted respectively as  $s_j^{(l)} = \{s_j^{(l)}\}$  and  $\mathbf{x}^{(l)} = \{x_j^{(l)}\}$
  - hypothesis  $h(\mathbf{w}) \in \mathcal{H}$  corresponds to hypothesis

$$\mathbf{w} \triangleq \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(L)}\} \quad (86)$$

- Forward propagation:
  - load  $\mathbf{x}^{(0)} = \mathbf{x}$

- for  $l \in \{1, 2, \dots, L\}$

$$\begin{aligned} \mathbf{s}^{(l)} &= (\mathbf{W}^{(l)})^T \mathbf{x}^{(l-1)} \\ \mathbf{x}^{(l)} &= \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(l)}) \end{bmatrix} \end{aligned} \quad (87)$$

- the final output is  $h(\mathbf{x}) = \mathbf{x}^{(L)}$
- The sensitivity vector:
  - the *sensitivity vector*  $\boldsymbol{\delta}^{(l)}$  has components defined as

$$\boxed{\boldsymbol{\delta}_j^{(l)} \triangleq \frac{\partial C_{\text{in}}}{\partial \mathbf{s}_j^{(l)}}} \quad (88)$$

where the signal  $\mathbf{s}_j^{(l)}$  is given by (87), and the sample cost function  $C_{\text{in}}$  is defined in (25)

- interpretation
  - \*  $\boldsymbol{\delta}_j^{(l)}$  is the 'signal error' in the  $j^{\text{th}}$  unit
  - \*  $\|\boldsymbol{\delta}^{(l)}\|$  is a measure on how fast layer  $l$  is learning
- applying the chain rule

$$\begin{aligned} \boldsymbol{\delta}_j^{(l)} &= \frac{\partial C_{\text{in}}}{\partial \mathbf{x}_j^{(l)}} \cdot \frac{\partial \mathbf{x}_j^{(l)}}{\partial \mathbf{s}_j^{(l)}} \\ &= \theta'(\mathbf{s}_j^{(l)}) \cdot \frac{\partial C_{\text{in}}}{\partial \mathbf{x}_j^{(l)}} \end{aligned} \quad (89)$$

where the unit output  $\mathbf{x}_j^{(l)}$  is given by (87)

- Recursive computation of the sensitivity vector:
  - computing  $\boldsymbol{\delta}^{(l)}$ :
    - \* for last layer  $L$ , both terms in (89) are readily available
    - \* for the other layers, a recursion on the sensitivity vectors can be obtained by running, a slightly modified version of the neural network backwards
  - since any component of  $\mathbf{x}^{(l)}$  can affect  $\mathbf{s}_k^{(l+1)}$ ,  $\partial C_{\text{in}} / \partial \mathbf{x}_j^{(l)}$  in (89) is computed as follows

$$\begin{aligned} \frac{\partial C_{\text{in}}}{\partial \mathbf{x}_j^{(l)}} &= \sum_{k=1}^{d^{(l+1)}} \frac{\partial \mathbf{s}_k^{(l+1)}}{\partial \mathbf{x}_j^{(l)}} \frac{\partial C_{\text{in}}}{\partial \mathbf{s}_k^{(l+1)}} \\ &= \sum_{k=1}^{d^{(l+1)}} w_{jk}^{(l+1)} \boldsymbol{\delta}_k^{(l+1)} \end{aligned} \quad (90)$$

- substituting (90) into (89)

$$\boxed{\boldsymbol{\delta}_j^{(l)} = \theta'(\mathbf{s}_j^{(l)}) \sum_{k=1}^{d^{(l+1)}} w_{jk}^{(l+1)} \boldsymbol{\delta}_k^{(l+1)}} \quad (91)$$

- in vector form

$$\boldsymbol{\delta}^{(l)} = \theta'(\mathbf{s}^{(l)}) \otimes [\mathbf{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}]_1^{d^{(l)}} \quad (92)$$

where  $\otimes$  denotes component-wise multiplication, known as *Hadamard product* or *Schur product*

- The backpropagation algorithm:
  - for each batch  $\{(\mathbf{x}_n, \mathbf{y}_n)\}$ , and for all  $i, j$  and  $l$ , it is desired to efficiently compute

$$\nabla C_{\text{in}} : \frac{\partial C_{\text{in}}}{\partial w_{ij}^{(l)}} \quad (93)$$

- *backpropagation* is an algorithm that efficient computes (93)
  - \* backpropagation is usually applied using the batch gradient descent update rule

- to express (93) in terms of sensitivity vector  $\delta^{(l)}$ , first apply the chain rule, then substitute the components with (87, 88)

$$\begin{aligned}\frac{\partial C_{\text{in}}}{\partial w_{ij}^{(l)}} &= \frac{\partial C_{\text{in}}}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} \\ &= \delta_j^{(l)} \cdot x_i^{(l-1)} \Rightarrow \\ \frac{\partial C_{\text{in}}}{\partial \mathbf{W}^{(l)}} &= \mathbf{x}^{(l-1)} \cdot \boldsymbol{\delta}^{(l)}\end{aligned}\tag{94}$$

- \* recursively computation of  $\delta^{(l)}$  (91), enables the backpropagation algorithm
- in forward propagation, the nonlinearity was the activation  $\theta(\cdot)$ , whereas in backpropagation, it is multiplication by  $\theta'(s_i^{(l)})$
- the backpropagation algorithm described above is SGD based, where the weights change after processing each data sample  $(\mathbf{x}_i, y_i)$
- in practice, multiple  $\delta$ s are computed before making a weight update
- the size of the samples grouped together is called the *mini batch size*
- more specifically, if the mini-batch size is  $m$ , then from (94)

$$\mathbf{W}^{(l)} \rightarrow \mathbf{W}^{(l)} - \frac{\eta}{m} \sum_x \mathbf{x}^{(l-1)} (\boldsymbol{\delta}_x^{(l)})^T\tag{95}$$

### B. Hyper-parameters

- General remarks:
  - *hyper-parameters* are variables set before actually optimizing the weights of the NN
  - finding a pseudo-optimal set of such parameters is a major challenge
  - randomly generated parameters do in general better than grid search
- An *epoch* is a single forward and backward pass of the whole dataset
- List of *hyper-parameters* include:
  - 1) hidden layers, & nodes per layer
  - 2) cost function
  - 3) activation
  - 4) epochs & mini batch sizes
  - 5) learning rate
  - 6) generalization parameter
  - 7) weight initialization
- Weight initialization:
  - initialization is critical for deeper networks
  - if initial weights are large then many nodes will saturate
  - if initial weights are small then many node outputs stay close to zero, and weights do not get updated
  - with fan-in  $d$ , it is recommended to generate weights using

$$\mathcal{N}\left(0, \sqrt{\frac{\alpha}{d}}\right)\tag{96}$$

- where  $\alpha$  is a hyper-parameter that is usually set to  $\alpha = 1$  for tanh and  $\alpha = 2$  for ReLU neuron models
- bias is not sensitive to initialization, so can be set to 0, to Gaussian with standard deviation of one, or as in (96)

### C. Visualization

- General remarks:
  - NN have high dimensionality and are hard to visualize
  - in *dimensionality reduction* high-dimensional data is mapped into lower dimensional data
    - \* one such method is PCA, see Appendix XV
    - \* another approach is to find maps that are distance preserving
  - visualization addresses the problem of visualizing high dimensional data
  - the space traversed by  $\{\mathbf{x}_n\}$ , at the input of a NN, is a very small subset of  $\mathcal{R}^d$
- Multidimensional scaling

- *multidimensional scaling*, or MDS, finds a distance preserving map by minimizing the cost function

$$\sum_{i,j} (d_{i,j}^* - d_{i,j})^2 \quad (97)$$

where  $d_{i,j}^*$ , and  $d_{i,j}$  are the distances between  $\mathbf{x}_i$  and  $\mathbf{x}_j$  in the original and reduced spaces respectively

- Sammon's mapping:

- in Sammons mapping, more emphasis is given to preserving the distances between nearby points than between those which are far apart
- the associated cost function is a modification of (97)

$$\sum_{i,j} \frac{(d_{i,j}^* - d_{i,j})^2}{d_{i,j}^*} \quad (98)$$

- Graph based visualization:

- similar to Sammon's mapping, graph based visualization prioritizes local or nearby points
- in a *nearest neighbor graph*, input vectors are the nodes and each node is connected to the three points that are closest to it in the original space
- the associated cost function is

$$\sum_{i,j} \frac{1}{d_{i,j}} + \frac{1}{2} \sum_{(i,j) \in E} (d_{i,j} - d_{i,j}^*)^2 \quad (99)$$

- t-SNE:

- t-SNE stands for *t-distributed stochastic neighbor embedding*
- t-SNE finds an embedding in two dimensions, such that locally, pairwise distances are conserved
- t-SNE tries preserve the topology of the data
  - \* for every point, t-SNE constructs a notion of which other points are its neighbors, trying to make all points have the same number of neighbors
  - \* then it tries to embed them so that those points all have the same number of neighbors
  - \* t-SNE is similar to the graph based visualization, but instead of just having points be neighbors or not neighbors, t-SNE has a continuous spectrum of having points be neighbors to different extents

- Deconv approach to visualization:

- to visualize some neuron, forward propagate signals to that neuron
- set that neuron gradient to one and set all other neuron gradients in that layer to zero
- back propagate and reconstruct the input
- in (91), set

$$S_j^{(l)} \triangleq \sum_{k=1}^{d^{(l)}} w_{jk}^{(l)} \delta_k^{(l)} \quad (100)$$

- with *guided backpropagation*, backpropagation (91), is modified to

$$\delta_j^{(l)} = \theta'(s^{(l)}) (S_j^{(l+1)} > 0) S_j^{(l+1)} \quad (101)$$

i.e. only positive gradients are passed on to the previous layer, and so only neurons that positively help activation are incorporated

- with *backward deconvnet*

$$\delta_j^{(l)} = (S_j^{(l+1)} > 0) S_j^{(l+1)} \quad (102)$$

- Examples of visualization:

- at first, or lowest, layer can use weights for visualization
  - \* the filters look like Gabor filters, see Section XV
- at the top layers, check what set of stimuli does a neuron responds to
- in an *occlusion experiment* the classification decision is monitored as a squared section of the input is zeroed out and moved around

- Optimization over image approaches:

- the goal is to find an image that maximizes some score,
- start by feeding all zeros
- set the gradient of the scores vector to be  $[0, 0, \dots, 1, \dots, 0]$ , then back-propagate to image
- update image

- \* can blur it, or zero pixels with small norm
  - forward propagate
  - keep iterating
- Adversarial examples:
  - can take an arbitrary image, can make targeted but minor changes to the input so that it will be classified to any other false object
  - presently this is a concern

#### D. Deep learning

- Deep neural networks & deep learning:
  - the quintessential example of *deep neural networks* (DNN) is the feedforward ANN
    - \* DNN includes more hidden layers, and each hidden layer in general includes more neurons
  - in recent years, *deep learning* (DL) has become the most popular approach to developing AI
- Shallow versus deep NNs:
  - even though any function could be computed using a shallow network, it may not be a good choice
  - with DNN, it is possible to construct a solution through multiple layers of abstraction
  - for some functions very shallow circuits require exponentially more elements to compute a problem than do DNNs
- Unstable gradient as main challenge to DNNs:
  - in general, the gradient in DNNs is unstable, tending to either explode or vanish in earlier layers
    - \* this is due to the fact that the updated quantity is the product of terms from all the later layers
  - when the gradient tends to get smaller as we move backward through the hidden layers, then neurons in the earlier layers learn slower than neurons in later layers
    - \* this is known as the *vanishing gradient problem*
    - \* it occurs due to product of small quantities
  - in other instances, the gradient gets larger in earlier layers
    - \* this is known as *the exploding gradient problem*
    - \* it occurs if the weights in the product are of large quantities
- The traditional ML approach
  - to address unstable gradient problem, traditional ML approach was based on first developing hand-tuned feature extractors
  - kernel methods, Appendix XIII, can handle non-linearities but do not generalize well in practice
  - hand tuned feature extractors are not robust
  - it is difficult to design reliable feature extractors
- Deep belief networks:
  - *deep belief networks*, or DBNs, are a class of DNN that use RBMs (126), to train the initial layers
  - they were the first generation of DNNs to address unstable gradient problem
- Hierarchical approach in DL:
  - DL methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level
  - deep layers allow for hierarchical feature representations
  - with the composition of enough such transformations, very complex functions can be learned
  - each module in the stack transforms its input to increase both the selectivity and the invariance of the representation
  - for classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations
- Enablers of DNN:
  - 1) Diverse data availability
    - availability of digitized data due to internet, cloud etc.
    - techniques to artificially expand the training data, see Section I-B
  - 2) Improvements in learning algorithms
    - introduction of new activations such as ReLU (41)
    - regularization algorithms such as dropout
    - automatic feature extractors using convolutional & pooling layers

- 3) Hardware acceleration
  - Moore’s law
  - Graphic Processing Units (GPU), see Section X

#### E. Convolutional, inception & residual networks

- Convolutional neural network:
  - a *convolutional neural network*, or CNN, is a class within DNN
  - the *Neocognitron* (Fukushima, 1980) introduced a powerful model architecture for processing images, inspired by the structure of the mammalian visual system
  - Neocognitron is the basis for CNN network as proposed by LeCun et al., in 1998 used in vision tasks
- Network size concerns & remedies:
  - if size of data set is fixed, larger network usually means more parameters, making it prone to overfitting
  - larger networks increase computational resources
  - to remedy these two issues, additional constraints are introduced into the network such as regularization, sparsity, or structure
  - sparsity replaces the fully connected layers by sparser ones
    - \* today’s computing infrastructures are inefficient when it comes to numerical calculation on non-uniform sparse data structures
  - convolutional neural networks introduce some form of constraint to the network
- CNNs use three ideas:
  - 1) Local receptive fields
  - 2) Shared weights
  - 3) Pooling
- Local receptive fields & stride length:
  - usually input data is represented as a rectangular format, rather than linear
  - in ANN there is full connectivity between layers  $(l - 1)$  and  $l$
  - in a CNN the connections are in a small, localized regions of the input image
  - the region in the input image that is connected to a hidden neuron is called the *local receptive field* or a *patch*
  - different hidden neurons cover different local receptive fields
  - adjacent local receptive fields are separated by one or more pixels, known as the *stride length*
- Shared weights:
  - the set of hidden units that cover all the receptive fields, all share the same weights and biases
    - \* the *shared weights* and *bias* are often said to define a *kernel* or *filter bank*
  - every element of this set is extracting the same feature, but at a different position
    - \* the map from the input layer to the hidden layer is sometimes called a *feature map*
  - the weights and biases learned for a given output layer are shared across all patches in a given input layer
- Filter Depth:
  - in many visual applications, data at each layer is presented in 3-dimensions  $(H \times W \times F)$ 
    - \* the amount of filters  $F$ , in a convolutional layer is called the *filter depth*
    - \* for input layer, the feature set could be the three primary colors
    - \* for hidden layers, multiple feature maps are usually extracted using multiple filter banks
  - for example, consider the convolutional layer connection
 
$$(H \times W \times F_1) \xrightarrow{F_2 \times (3 \times 3 \times F_1)} (H \times W \times F_2) \quad (103)$$

where each filter has  $3 \times 3 \times F_1$  local receptive field, & for each location on the output layer there are  $F_2$  filters
  - $F_1$  &  $F_2$  refer to the feature sets at first and second layers
- Convolutional layer:
  - a hidden layer with shared filter bank is called a *convolutional* layer since the output is the convolution of a rectangular window and the input
  - it is common to zero pad the border if hidden layer sizes need to be preserved,
  - convolutional filters
    - \* provides translation invariance of images
    - \* dramatically reduces the number of parameters, compared to full FC layers

- On filter size trends:  $(1 \times 1)$  convolutions
  - recently, multiple layers of smaller sized filters are chosen over a single layer of larger sized filters
    - \*  $(1 \times 1 \times F)$  layers
    - \* such architectures, in general, use less memory, are more computationally efficient, & provide more non-linearities
  - a FC layer with  $n$  output elements is equivalent to a  $(1 \times 1 \times n)$  layer
  - can factor  $(n \times n)$  convolutions into  $(1 \times n)$  &  $(n \times 1)$
- Example on filter size tradeoffs: a bottleneck architecture
  - consider the connection architecture of (103) with  $F_1 = F_2 = F$
  - this connection architecture can be replaced by a *bottleneck architecture*, where

$$(H \times W \times F) \xrightarrow{\frac{F}{2} \times (1,1,F)} (H \times W \times \frac{F}{2}) \xrightarrow{F/2 \times (3,3,\frac{F}{2})} (H \times W \times \frac{F}{2}) \xrightarrow{F \times (1,1,\frac{F}{2})} (H \times W \times F) \quad (104)$$

- \* first, the layer size is reduced to  $(H \times W \times F/2)$  by using  $F/2$   $(1 \times 1 \times F)$  filters per position
- \* then another layer of same size is generated by using  $F/2$   $(3 \times 3 \times F/2)$  filters
- \* finally, original layer size  $(H \times W \times F)$  is restored using  $F$   $(1 \times 1 \times F/2)$  filters
- Transposed convolution layer: asd
  - *transposed convolution* helps in upsampling the previous layer to a higher resolution or dimension, see Section VIII-D
  - the term transpose means to transfer to a different place or context
    - \* use a transposed convolution to transfer patches of data onto a sparse matrix
    - \* then can fill the sparse area of the matrix based on the transferred information
- Super-parameters needed for convolutional layer:
  - 1) Features or filter banks
    - choose a power of 2, like 64
  - 2) Filter size, or local receptive fields
    - use smaller sizes at lower layers and larger sizes for higher layers
  - 3) The stride length
    - choose 1 or 2
  - 4) Zero padding
- Pooling layer:
  - a *pooling layer* is usually used after convolutional layers
  - a pooling layer simplifies, or condenses, the output information from the convolutional layer
    - \* the role of the pooling layer is to merge semantically similar features into one
    - \* each feature map is pooled separately
  - conceptually, the benefit of the max pooling operation is to reduce the size of the input, and allow the neural network to focus on only the most important elements
  - pooling reduces the dimension of the representation and creates an invariance to small shifts and distortions
    - \* pooling reduces the number of parameters needed in later layers
  - common pooling strategies do not introduce new parameters but do introduce new hyper-parameters, including *pooling size* and *pooling stride*
  - the pooling operation is applied individually for each depth slice
    - \* for a pooling layer the output depth is the same as the input depth
  - $(2 \times 2)$  pooling with stride of 2 is common
- Pooling methods:
  - with *max-pooling*, a unit simply outputs the maximum activation in an input region

$$y = \max_i(X_i) \quad (105)$$

- with *average pooling*, the average signal is picked

$$y = \text{mean}_i(X_i) \quad (106)$$

- with *L2 pooling*, the square root of the sum of the squares of the activations are computed

$$y = \sqrt{\sum_i X_i^2} \quad (107)$$



- Hierarchical approach to the softmax output layer:
  - when there are large number of classes, the softmax output layer becomes complex
  - a hierarchical approach introduces classes  $c_t$ , where multiple classes would belong to class  $c_t$
  - the conditional probability distribution can then be factored as

$$p(o_t|\text{history}) = p(c_t|\text{history}) p(o_t|c_t) \quad (108)$$

- Inception network:
  - *inception* is computationally efficient architecture
  - inception-v1, see [3]:
    - \* an inception module or layer, is constructed by the concatenation of various filters such as pooling operations & convolutional filters of different width
    - \* an inception layer also uses filter size reduction as discussed in (103)
    - \* an *inception network* is a network consisting of Inception modules, stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid
  - inception-v2 introduced batch normalization (12)
  - inception-v4 includes more diverse types of inception & reduction layers [4]
- Residual connections:
  - introduced by He et al. in 2015
  - residual connection utilize additive merging of signals from different layers
  - if  $x$  is the ReLU output from previous layer, then residual connection computes the next layer output as

$$y = \text{ReLU}[\text{Conv}(\text{Conv}(x)) + x] \quad (109)$$

- Trends:
  - smaller filters
  - deeper architectures
  - possibly dropping pooling
    - \* recent datasets are so big and complex we are more concerned about under-fitting
    - \* dropout is a better regularizer
    - \* pooling results in a loss of information
  - dropping some or all FC layers
  - special layers such as residual connections, inception layers or some combinations of the two

## F. Transfer Learning

- *Transfer learning* involves taking a pre-trained neural network and re-adapting it to a new, different data set
- Approaches to transfer learning:
  - 1) new data set is small, new data is similar to original training data
    - retrain only top FC layer
  - 2) new data set is small, new data is different from original training data
    - replace all FC layers and last convolutional layer with a single FC layer
    - this is known as *feature extraction*
  - 3) new data set is large, new data is similar to original training data
    - remove the last FC layer and replace with a layer matching the number of classes in the new data set
    - initialize the rest of the weights using the pre-trained weights
    - re-train the entire neural network
    - this is known as *finetuning*
  - 4) new data set is large, new data is different from original training data
    - remove the last FC layer and replace with a layer matching the number of classes in the new data set
    - retrain the network from scratch with randomly initialized weights
- With modest training data, can retrain both middle and upper layers
  - re-learn middle layers with 1% learning rate of original network
  - re-learn upper layers with 10% learning rate of original network
- Popular networks to start from:
  - 1) LeNet [5]
    - the first successful applications of CNN, in 1998

- best known for reading zip codes, digits, etc.

2) AlexNet [6]

- popularized CNN in computer vision by winning ImageNet 2012, by a significant margin
- compared to LeNet, its deeper, bigger, & featured convolutional layers stacked on top of each other
- used ReLU activation and dropout to avoid overfitting
- 8 layers, 5 convolution layers followed by 3 FC layers

3) ZFNet [7]

- winner of ImageNet 2013
- compared to AlexNet, the size of the middle convolutional layers were expanded and the stride and filter size on the first layer were reduced

4) GoogLeNet [3]

- winner of ImageNet 2014
- the *GoogLeNet* name refers to a particular incarnation of the inception architecture used in the submission for the ILSVRC 2014 competition
- it is known to be fast and has potential for real-time applications
- 22 layers with parameters, i.e. without pooling

5) VGGNet [8]

- from Visual Geometry Group at Oxford University
- the runner-up in ImageNet 2014
- it features a homogeneous architecture that only performs  $(3 \times 3)$  convolutions with  $(2 \times 2)$  pooling layers, followed by three layers of FC layers
- known to be a flexible network
- but it is expensive to evaluate and uses a lot more memory and parameters (140M)
- VGG16 is the 16 layer version, while VGG19 has 19 layers

6) ResNet [9]

- winner of ImageNet 2015
- *ResNet*, short for residual network
- ResNet is an ultra-deep network with 152 layers
- ResNet uses residual connections (109), batch normalization (12), and  $(1 \times 1)$  convolutional filters (103)
  - \* ResNet does not include FC fully layers at the top of the network
- ResNet is currently the state of the art of CNNs, and are the default choice for using CNNs in practice

## V. RECURRENT NEURAL NETWORKS WITH SUPERVISION

- Overview:

- *Recurrent Neural Networks* (RNN) entail feedback,
- RNN networks are effective for
  - \* visual attention systems work sequentially on an input,
  - \* systems that receive a sequence of inputs, such as sentiment analysis,
  - \* systems that produce a sequence of outputs, for example, captioning an image,
  - \* systems with both sequential inputs and outputs such as machine translation,
- RNNs can be viewed as programs, or state machine,
- the final state vector of an RNN's hidden units may represent the thought expressed by the sentence.

- RNN model:

- if  $\mathbf{x}$  is the input, and  $\mathbf{h}$  is output of the hidden layer, then

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}), \quad (110)$$

- if hidden layers form a single layer,

$$\mathbf{h}_t = \theta(W^{(x)}\mathbf{x}_t + W\mathbf{h}_{t-1}), \quad (111)$$

- alternatively, if  $\mathbf{h}$  is the signal level at the hidden layer, then the an RNN can be modeled as

$$\mathbf{h}_t = W^{(x)}\mathbf{x}_t + W\theta(\mathbf{h}_{t-1}), \quad (112)$$

- then the output is of the form

$$\hat{y}_t = W^{(s)}\theta(\mathbf{h}_t). \quad (113)$$

- Deep RNN:

- stack multiple RNN layers, where the hidden state of one RNN is the input of the next RNN,
- there are multiple hidden layers between inputs and outputs.

- Bidirectional RNN:

- causal is forward, anti-causal in backwards, like BCJR,
- the output is a function of the concatenation of the two hidden layer outputs.

- Deep bidirectional RNN:

- a deep and bidirectional extensions of an RNN can be combined to create a deep bidirectional RNN.

- Encoder and decoders:

- for some applications, such and language translation, can think of the system model of doing two tasks, encoding and decoding
- in the basic RNN model (112)
  - \* encoding occurs while there is an incoming stream of inputs  $\mathbf{x}_t$ ,
  - \* decoding occurs when there is an output  $\hat{y}_t$ ,
- assume decoder starts after encoder ends.

- Extensions to RNN for encoders/decoders:

- this basic model (112), can be extended in multiple ways:
  - \* use different weights  $W'$ , for decoding,

$$\mathbf{h}_{D,t} = W'\theta(\mathbf{h}_{t-1}), \quad (114)$$

- \* remember last state of the encoder  $\mathbf{h}_T$ ,

$$\mathbf{h}_{D,t} = W'\theta(\mathbf{h}_{t-1}) + W^T\theta(\mathbf{h}_T) \quad (115)$$

- \* include the previous output  $\hat{y}_{t-1}$ ,

$$\mathbf{h}_{D,t} = W'\theta(\mathbf{h}_{t-1}) + W^T\theta(\mathbf{h}_T) + W^y\hat{y}_{t-1}. \quad (116)$$

- Vanishing / exploding problem in RNN:

- denote the cost function at time  $t$  as  $E_t$ ,

- the derivative of  $E_t$  w.r.t.  $W$  can be expressed as,

$$\frac{\partial E_t}{\partial W} = \sum_{\tau=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} \frac{\partial \mathbf{h}_\tau}{\partial W} \quad (117)$$

- using (112), expand one of Jacobian terms in (117),

$$\begin{aligned} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_\tau} &= \prod_{j=\tau+1}^t \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \\ &= \prod_{j=\tau+1}^t W^T \text{diag}[\theta'(\mathbf{h}_{j-1})], \end{aligned} \quad (118)$$

- then the gradient becomes the product of all these Jacobian matrices which becomes very small or very large quickly,
- for exploding problem can saturate the gradient,
- for vanishing problem initialize  $W = I$  and use ReLU.

- Learning with long term memory:

- theoretical and empirical evidence shows that it is difficult to learn to store information for long sequences,
- three approaches include,
  - \* gating techniques,
  - \* *Neural Turing Machine*, where the network is augmented by a tape-like memory that the RNN can choose to read from or write to,
  - \* *memory networks*, where a regular network is augmented by a kind of associative memory.

- Gating:

- *gating* is a technique that helps the network to decide when to forget or remember an input,
- gating techniques use RNN where each hidden layer is more complex,
- two popular gating techniques are GRU and LSTM,

- GRU:

- the ideas of *Gated Recurrent Units* (GRU) are,
  - \* to keeps around memories to capture long distance dependencies,
  - \* allow error messages to flow at different strengths depending on inputs,
- the GRU first computes two gates, *update gate*  $z_t$ , and the *reset gate*  $r_t$ ,

$$\begin{aligned} z_t &\triangleq \sigma \left( U^{(z)} x_t + W^{(z)} h_{t-1} \right), \\ r_t &\triangleq \sigma \left( U^{(r)} x_t + W^{(r)} h_{t-1} \right), \end{aligned} \quad (119)$$

- \*  $\sigma$  is the sigmoid activation (34),
- \*  $h_t$  is the output of the activation,
- \* think of these two gates as a special hidden layers,
- intermediate memory content is

$$\tilde{h}_t \triangleq \tanh(W x_t + r_t \circ U h_{t-1}), \quad (120)$$

where  $\circ$  is the Hadamard multiplication,

- the hidden layer is updated as

$$h_t \triangleq z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t. \quad (121)$$

- LSTM:

- *Long short-term memory* (LSTM) [10], is a popular and powerful tool,
- LSTM is similar to GRU but three gates are used rather than two: input gate  $i_t$ , forget gate  $f_t$ , and output gate  $o_t$ ,

$$\begin{aligned} i_t &\triangleq \sigma \left( W^{(i)} x_t + U^{(i)} h_{t-1} \right), \\ f_t &\triangleq \sigma \left( W^{(f)} x_t + U^{(f)} h_{t-1} \right), \\ o_t &\triangleq \sigma \left( W^{(o)} x_t + U^{(o)} h_{t-1} \right), \end{aligned} \quad (122)$$

- the intermediate memory cell  $\tilde{c}_t$ , the final memory cell  $c_t$ , and the final hidden state  $h_t$ , are respectively,

$$\begin{aligned} \tilde{c}_t &\triangleq \tanh \left( W^{(c)} x_t + U^{(c)} h_{t-1} \right), \\ c_t &\triangleq f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \\ h_t &\triangleq o_t \circ \tanh(c_t), \end{aligned} \quad (123)$$

- $c_t$  introduces additive interaction, similar to a ResNet,
  - these additions also help backpropagation,
  - multiple LSTM's are usually stacked up to generate a deep LSTM.
- Recursive neural networks are covered in Section IX.

## VI. UNSUPERVISED LEARNING

- Unsupervised learning:
  - the goal of *unsupervised learning* is to learn some structure of the data, without labels or teacher,
  - still mostly a research topic,
  - examples include
    - \* clustering,
      - $k$ -means clustering is unsupervised clustering algorithm, not based in NN, see Appendix XV.
    - \* dimensionality reduction & feature learning,
      - see autoencoders below,
    - \* generative modeling,
      - see variational autoencoders below.
- Hebbian rule:
  - neurons that fire together, wire together,
  - more specifically, *Hebbian rule*, is a simple, unsupervised learning algorithm [11], where the weight  $w_{ij}$  connecting neurons  $x_i$  to  $x_j$  is updated proportional to

$$\frac{dw_{ij}}{dt} \propto \text{correlation}(x_i x_j). \quad (124)$$

- Energy-based models:
  - *energy-based models* associate a scalar energy to each configuration of the variables of interest,
  - learning corresponds to modifying the energy function so that its shape has desirable properties,
    - \* for example, we would like desirable configurations to have low energy,
  - Hopfield networks and Boltzmann machines are examples of energy-based models.
- Hopfield networks:
  - a *Hopfield network* is an undirected graph with only visible nodes,
  - a Hopfield model is an energy-based model, where the goal is to make the  $N$  samples  $\{x_i\}$ , the stable states of the network,
  - the neurons can be activated synchronously (simultaneously), or asynchronously (one at a time),
  - applications:
    - \* associative memories, or content addressable memories,
    - \* optimization problems.
- Hopfield network learning:
  - let  $x_i^{(n)} \in [-1, 1]$ , be the activation output,
  - the weights are set based on Hebbian learning (124),

$$w_{ij} \sim \sum_{n \leq N} x_i^{(n)} x_j^{(n)}, \quad (125)$$

- the resulting learning algorithm is unsupervised,
- the weights are symmetric, i.e.,  $w_{ij} = w_{ji}$ ,
- there are no self connections, i.e., set  $w_{ii} = 0$ .
- Boltzmann machines & RBM:
  - a *Boltzmann machine* differs from a Hopfield network in two ways,
    - \* the Boltzmann machine also allows hidden nodes,
    - \* the Boltzmann machine uses stochastic neuron, with probabilistic firing mechanism,
  - a *restricted Boltzmann machine* (RBM) is a Boltzmann machine with a bipartite structure

$$\mathbf{x} \leftrightarrow \mathbf{h} \quad (126)$$

where the hidden nodes constitute one layer and visible nodes the other,  
 \* there are no connections within a layer.

- Autoencoders:
  - an *autoencoder* is trained to encode the input  $\mathbf{x}$  into some representation  $c(\mathbf{x})$ , so that  $\mathbf{x}$  can be reconstructed from  $c(\mathbf{x})$ ,

$$\mathbf{x} \xrightarrow{\text{encoder}} c(\mathbf{x}) \xrightarrow{\text{decoder}} \mathbf{x} \quad (127)$$

\* autoencoders are symmetric in the sense that the encoder and a decoder could share weights,

- \* the encoder could be a deep network,
- \*  $c(\mathbf{x})$  is viewed as a lossy compression of  $\mathbf{x}$ ,
- \* can think of  $c(\mathbf{x})$  as features of  $\mathbf{x}$ ,
- the associated cost function is

$$-\log p(\mathbf{x}|c(\mathbf{x})), \quad (128)$$

- as an example, if there is one linear hidden layer and the mean squared error criterion is used to train the network, then the  $k$  hidden units learn to project the input in the span of the first  $k$  principal components of the data,
- an autoencoder is usually used as component of neural network, such as feature extraction, and dimensionality reduction, where only the encoder is used after training,
  - \* autoencoders are not very effective in training deep networks.
- Autoencoder versus RBM:
  - taking  $\mathbf{h} = c(\mathbf{x})$  in (126, 127), an autoencoder and RBM become equivalent when,
    - \* the encoder is one layer,
    - \* the encoder and decoder are symmetric and share weights,
    - \* an RBM uses the same activation as an autoencoder.
- Variation autoencoder:
  - using a Bayesian approach on an autoencoder, a *variational autoencoder* (VAE) can generate new data,
    - \* generating new data from available one is called generative modeling,
  - in VAE,  $\mathbf{z} \triangleq c(\mathbf{x})$  (see (127)) is called the *latent variables*,
    - \* think of  $\mathbf{z}_i$  as an object or a high level attribute of an object,
    - \*  $P(\mathbf{z})$  is assumed to be diagonal, unit Gaussian random vector,
  - assume  $P(\mathbf{x}|\mathbf{z})$  &  $P(\mathbf{z}|\mathbf{x})$  are also diagonal Gaussian, with mean and variances of  $(\mu, \Sigma)$ ,
  - the encoder & decoder are used to predict the corresponding means & variances,

$$\begin{aligned} \mathbf{z} \in P(\mathbf{z}) & \xrightarrow{\text{VAE decoder}} (\mu^x, \Sigma^x), \\ \mathbf{x} & \xrightarrow{\text{VAE encoder}} (\mu^z, \Sigma^z), \end{aligned} \quad (129)$$

- during training the process is as follows

$$\mathbf{x} \rightarrow (\mu^z, \Sigma^z) \rightarrow \mathbf{z} \rightarrow (\mu^x, \Sigma^x) \rightarrow \mathbf{x}', \quad (130)$$

- the loss function is the *reconstruction loss* which could be based on KL divergence rather than L2,
- to generate data,

$$\mathbf{z} \rightarrow (\mu^x, \Sigma^x) \rightarrow \mathbf{x}. \quad (131)$$

- Generative adversarial networks:
  - \* *generative adversarial networks* is a framework for estimating generative models via an adversarial process, with two components
    - a generative model (G), similar to VAE decoder, generates fake data,
    - a discriminator model (D), is a binary classifier that decides whether a data is real or generated by G.
  - \* generator and discriminator model are trained together,

## VII. APPLICATIONS

- Andrew Ng: AI is the new electricity
  - rule of thumb: what humans can do in less than a second
- Applications:
  - vision, Section VIII
  - automatic speech recognition & natural language processing, Section IX
  - health care & biomedicine, Section VII-A
  - web searches, content filtering & online advertising
    - \* what ad to provide to a given user?
  - personal assistant such as e-mail smart reply
  - recommendations on e-commerce web sites
    - \* Netflix
  - automotive & robotics
  - analyzing particle accelerator data
  - video games
  - finance
  - education
- Search engine:
  - given a query, a search engines can also search for other phrases that have word vectors close to the query, see Section IX,
  - given a query and a document, a neural network can return a score for the query, document pair.
- Some startups:
  - Catio, clarifai, calrify, Dato, emotient, enlitic, ersatz labs, EyeEm, herta security, IFLYTEK, Intelligent Voice, iQIY, Letv, megv11, MetaMind, NERVANA SYSTEMS, rbeus, SENSETIME, Sogou, Unisound, VIONVISION, zebra.

### A. Health care & Biomedicine

- Medical image understanding
  - radiology
- Massive amount of genomic data availability
- Bioinformatics:
  - can think of gene expression as object classification
  - drug discovery
- Molecular activity prediction



## VIII. VISUAL RECOGNITION

- Generalities:
  - vision recognition is also known as *image perception*, *machine vision*, *scene understanding* or *computer vision*
  - computer vision enables computers to see, identify and process images in the same way that human vision does, and then provide appropriate output
    - \* the goal of computer vision is to extract useful information from an image
  - a camera looks at a 3D object and transforms them to 2D image
    - \* this transformation is modeled by a *camera matrix*  $C$
    - \* computer vision can infer depth resolution, i.e. 3D
  - compared to other sensors
    - \* images are characterized as having high spatial resolution
    - \* cameras and image processing are cheap
- A list of vision tasks:
  - classification (of a single object)
    - \* *classification* implies single object classification
    - \* classification partitions images to  $C$  classes, or labels
    - \* e.g., face recognition, identity verification
  - localization (put box around an object), see below
  - classification + localization
    - \* generates both the label as well as the location of an object
  - detection (classification + localization of variable number of objects)
  - semantic segmentation (classification at pixel level)
  - instance segmentation
  - attention models
  - video & action recognition
  - 3D and depth interpretation
- Localization:
  - *localization* implies localizing a classified object
  - given an image, generate box around object
    - \* box means object coordinates at upper left corner  $(x, y)$ , width and height  $(w, h)$
    - \* sometimes referred to as *bounding-box regression*
  - can be approached as a regression problem, where the metric is the L2 distance between estimated box and ideal box coordinates
  - FC layers do not preserve spatial information, while convolutional layers do
- IoU:
  - intersection over union, or IoU, is a metric used to measure performance of localization and segmentation problems
  - IoU is a number between zero and one, with higher IoU's implying better performance
  - for localization, IoU is the intersection of the box and the object, divided by their union
  - for segmentation, IoU is the intersection of the reference and classified pixels in a given class, divided by their union
- Localization of  $K$  objects:
  - $K$  is the number of objects in an image
  - for example,  $K = 4$ 

$$\{\text{cat, cat's head, left ear, right ear}\} \quad (132)$$
  - in the above example, the output is  $4K$  values
- Image location to other regression tasks:
  - the localization regression technique can also be used to characterize other real valued properties of an object
  - for example, human pose estimation
- Detection & mAP:
  - *detection* is the classification and the localization of variable number of objects,
  - evaluation is through the metric *mean average precision*, or mAP
    - \* mAP computes average precision separately for each class, then averages over classes,
    - \*  $0 \leq \text{mAP} \leq 100$ , high is good

### A. Color

- Color space:
  - a *color space* is a specific organization of colors
  - a color space provides a way to categorize colors and represent them in digital images
  - the color space of a pixel is usually represented by a triplet while greyscale is a scalar
- Hue:
  - hue represents color independent of any change in brightness
  - *hue* is in degrees on the color wheel
    - \* hue is a cylindrical-coordinate representation
  - let  $V_{\max} \triangleq \max(R, G, B)$ ,  $V_{\min} \triangleq \min(R, G, B)$  and  $\Delta = V_{\max} - V_{\min}$ , then the hue  $H$  is

$$\begin{aligned} H &= \frac{30}{\Delta}(G - B), & \text{if } V_{\max} = R \\ H &= 60 + \frac{30}{\Delta}(B - R), & \text{if } V_{\max} = G \\ H &= 120 + \frac{30}{\Delta}(R - G), & \text{if } V_{\max} = B \end{aligned} \quad (133)$$

\* 0 (or 360/2) is red, 120/2 is green, 240/2 is blue

- Lightness or value:
  - *lightness* measures the relative lightness or darkness of a color
  - lightness is a percentage value
    - \* 0% is dark (black)
    - \* 100% is light (white)
  - if  $V_{\max} \triangleq \max(R, G, B)$ , and  $V_{\min} \triangleq \min(R, G, B)$ , then the lightness  $L$  is

$$L = \frac{V_{\max} + V_{\min}}{2} \quad (134)$$

- Saturation:
  - *saturation* is a measurement of colorfulness
    - \* as colors get lighter and closer to white, they have a lower saturation value, whereas colors that are the most intense, like a bright primary color have a high saturation value
  - saturation is a percentage value
    - \* 100% is the full colour
  - if  $V_{\max} \triangleq \max(R, G, B)$ , and  $V_{\min} \triangleq \min(R, G, B)$ , then the saturation  $S$  is

$$\begin{aligned} S &= \frac{V_{\max} - V_{\min}}{V_{\max} + V_{\min}} & \text{if } L < 0.5 \\ S &= \frac{V_{\max} - V_{\min}}{2 - (V_{\max} + V_{\min})} & \text{if } L \geq 0.5 \end{aligned} \quad (135)$$

- Color space representations:
  - RGB (Red, Green, Blue)
  - BGR (Blue, Green, Red)
  - HSV (Hue, Saturation, Value)
  - HLS (Hue, Lightness, Saturation)
  - LUV (Luminance, while  $(x, y) \rightarrow (u, v)$ )
  - Lab (Lightness, a and b for the color opponents green-red and blue-yellow)

### B. Dataset

- Data sets:
  - MNIST, see below
  - CIFAR, see below
  - ImageNet
    - \* *ImageNet Large Scale Visual Recognition Challenge*, or ILSVRC
    - \* a database of labeled images
  - vehicle image database, GTI, [http://www.gti.ssr.upm.es/data/Vehicle\\_database.html](http://www.gti.ssr.upm.es/data/Vehicle_database.html)

- KITTI, <http://www.cvlibs.net/datasets/kitti/>
- PASCAL VOC
- MS-COCO
- MNIST data set:
  - MNIST data set is a database of the 10 digits
  - <http://yann.lecun.com/exdb/mnist/>
  - NIST stands for the United States' *National Institute of Standards and Technology*
  - MNIST is a *modified* subset of two data sets collected by NIST
  - specifics:
    - \* the images are greyscale,  $(28 \times 28)$  pixels
    - \* training data contains 60,000 images
      - these images are scanned handwriting samples from 250 people, half of whom were US Census Bureau employees, and half of whom were high school students
    - \* test data contains 10,000 images
      - the test data was taken from a different set of 250 people than the original training data (a group split between Census Bureau employees and high school students)
- CIFAR:
  - <https://www.cs.toronto.edu/~kriz/cifar.html>
  - the CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset
  - CIFAR-10 dataset:
    - \* 60,000 images
      - 50,000 training images and 10,000 test images
    - \*  $32 \times 32$  colour images
    - \* 10 classes
      - with 6000 images per class
  - CIFAR-100 dataset:
    - \* like the CIFAR-10, except it has 100 classes containing 600 images each
    - \* there are 500 training images and 100 testing images per class
- Formats:
  - LMDB
  - HDF5

### C. Image Processing

- Edge detection:
  - an edge detection algorithm extracts edges from an image
  - an image is converted to a binary image where the ones correspond to the edges
  - approaches to detect edges
    - \* gradient based, for example by using Sobel operator or Canny Edge detection, where first gradient is taken and then passed through a threshold
    - \* threshold based, by looking into some of the color space attributes and then passing it through a threshold
- Image convolution:
  - an image  $f(x, y)$  is convolved with a *filter*, or *mask*  $h(x, y)$ 

$$g(x, y) = h(x, y) * f(x, y) \quad (136)$$
  - $h(x, y)$  is  $(n \times n)$ , when  $n$  is odd, e.g.  $n = 3$
  - applications include blurring, sharpening, edge detection, noise reduction etc.
    - \* Gaussian smoothing
    - \* Sobel operator
- Sobel operator:
  - the Sobel operators or filters are defined as

$$S_x \triangleq \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad S_y \triangleq \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad (137)$$

- the Sobel operators take the derivative of the image in the x (horizontal) or y (vertical) direction
- Canny Edge Detection:
  - developed by John F. Canny in 1986
  - the goal is to identify the boundaries in an image
  - the Sobel operator is at the heart of the Canny edge detection algorithm
  - the following steps are usually included:
    - \* convert image to greyscale
    - \* apply Gaussian smoothing function to suppress noise and spurious gradients
    - \* compute the gradient
    - \* have two thresholds, where all edges detected above the high threshold are retained while pixels between two threshold are retained if connected to strong edges
- The Hough Transform & lines:
  - developed by Paul Hough in 1962
  - *Hough transform* represents lines in parameter space, or in *Hough space*
    - \* the line  $y = mx + b$  (in *image space*) transforms to the point  $(m, b)$  in Hough space
$$(x, y) \rightarrow (m, b) \quad (138)$$
  - \* a point  $(x_o, y_o)$  transforms to the line  $(m, y_o - mx_o)$  over variable  $m$ 
    - a line is transformed to a point
    - a point is transformed to line
  - \* a line that passes through two points in image space is the intersection of the two lines, associated with the two points, in Hough space
  - instead of detecting lines, from multiple points, in image space, can detect intersection of lines in Hough space- Hough Transform in polar coordinates:
  - polar coordinates  $(\rho, \theta)$ , are used to address vertical line representations with  $m = \infty$ ,
  - draw a segment  $s$ , from origin that intersects perpendicular to  $y = mx + b$ 
    - \*  $\rho$  is the perpendicular distance of  $s$
    - \*  $\theta$  is the angle at origin from x-axis to  $s$
  - since the product of the slopes of perpendicular lines is  $-1$
$$-1 = m \cdot \tan \theta \Rightarrow m = -\frac{\cos \theta}{\sin \theta} \quad (139)$$
  - using simple geometry
$$\sin \theta = \frac{\rho}{b} \Rightarrow b = \frac{\rho}{\sin \theta} \quad (140)$$
  - substituting  $m$  and  $b$  into the line equation
$$\begin{aligned} y &= mx + b \\ &= -\frac{\cos \theta}{\sin \theta} x + \frac{\rho}{\sin \theta} \Rightarrow \\ x \cos \theta + y \sin \theta &= \rho \end{aligned} \quad (141)$$
  - a point in image space  $(x_o, y_o)$  corresponds to the sinusoidal equation in Hough space
$$x_o \cos \theta + y_o \sin \theta = \rho \quad (142)$$
  - detecting lines (from points) in image space becomes equivalent to detecting intersection of sinusoidal equations in Hough space- Distortion:
  - during 3D to 2D transformation of a camera capture, distortion changes the shape and size of 3D objects captured on 2D image
  - types of distortion:
    - \* radial distortion
    - \* tangential distortion
  - a regular pattern, like a chessboard, can be used to calibrate a camera
- Radial distortion:

- cameras use curved lenses to form an image, and light rays often bend a little too much or too little at the edges of these lenses
- this creates an effect that distorts the edges of images, where lines or objects appear more or less curved than they actually are
- this phenomenon is known as *radial distortion*, and its the most common type of distortion
- let  $(x, y)$  be a point on the distorted image
- let  $r$  be the distance from *distortion center* to  $(x, y)$
- to correct the appearance of radially distorted points in an image, a correction formula can be applied that uses three distortion coefficients  $k_1$ ,  $k_2$ ,  $k_3$ , and  $r$

$$\begin{aligned}x_c &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\y_c &= y(1 + k_1r^2 + k_2r^4 + k_3r^6)\end{aligned}\tag{143}$$

- Tangential distortion:

- tangential distortion occurs when a cameras lens is not aligned perfectly parallel to the imaging plane, where the camera film or sensor is
- this makes an image look tilted so that some objects appear farther away or closer than they actually are
- given the two distortion coefficients  $p_1$  &  $p_2$ , the correction formulae is

$$\begin{aligned}x_c &= x + 2p_1xy + p_2(r^2 + 2x^2) \\y_c &= y + 2p_2xy + p_1(r^2 + 2y^2)\end{aligned}\tag{144}$$

- Perspective transform:

- a perspective transform maps the points in a given image to different, desired, image points with a new perspective
- the birds-eye view transform that lets us view a lane from above is useful for calculating the lane curvature

#### D. Vision with legacy machine learning

- Features:

- raw pixel intensity
  - \* it can provide both color & shape characteristics about an object
  - \* for example, template matching
- histogram of pixel intensity
  - \* it provides color characteristics about an object
  - \* histograms lose shape characteristics but make detection less sensitive to orientation
  - \* for example, saturation histogram forms a feature vector
- gradient of pixel intensity
  - \* it provides shape characteristic of an object
  - \* it loses color characteristics & becomes color invariant
  - \* for example, the direction of gradient can be used to identify a shape, see HOG below
- recommended to use multiple features
  - \* need to normalize features if all features are represented as a single vector
- finally, the features are fed to a classifier to be trained on data

- HOG:

- *histogram of oriented gradients* (HOG) is a detection algorithm from 2005
- HOG assumes that an object can be described by the distribution (histogram) of intensity gradients (edge directions)
- an image is divided into small connected regions called *cells*, and for the pixels within each cell, a histogram of gradient directions is compiled
  - \* orientation is quantized to  $M$  bins
  - \* each pixel in a cell is assigned to one of the  $M$  orientations
  - \* each pixel influences on that bin based on its magnitude
  - \* the resulting histogram is the feature vector associated with each cell
- hyperparameters include cell size, cell overlap, quantized bins  $M$ ,
- afterwards a linear classifier is applied on these features
- there are similarities to CNNs

- DPM:

- *deformable parts model* (DPM), from 2010, is based on HOG but uses more more complex functions than linear classifier

- DPM was state of the art before CNNs
- Sliding window:
  - so far feature extraction and classification was performed on a single window on a larger image
  - to be able to do detection, run classification at multiple locations & scales on a high resolution image
  - at each location generate score
  - using heuristics, combine classifier & regressor predictions across all scales for final prediction
- Overfeat:
  - Overfeat [12] is an example of a sliding window architecture
  - Overfeat was the winner of ILSVRC 2013 classification + localization challenge

#### E. Vision with neural networks

- Classification and localization with CNN:
  - the network is the union of some CNN for classification, with a second FC *regression-head*, appended to the network, at the top
  - the resulting CNN will have two heads, a *classification-head* and a regression-head
  - the regression head gets its input either from the final convolutional layer, or after last FC layer
  - the regression head is trained separately
  - localization can be class agnostic, with 4 coordinate numbers, or class specific, with  $4C$  numbers
- Detection as classification:
  - detection can be reduced to classification / localization problem by using moving window techniques
  - need to test many positions and scales
- Region proposal and selective search:
  - *region proposal* algorithm looks only at a small subset of possible positions
  - it finds blobby image regions that are likely to contain objects
  - it performs class-agnostic object detector on a region
  - one method of doing region proposal is called *selective search* which is a bottom-up segmentation approach, merging regions at multiple scales
- R-CNN [13]:
  - *region based CNN*, or R-CNN is a region based algorithm proposal, implemented on a CNN
  - using selective search module (not NN), called *region proposal*, get around 2000 different sized regions of interest (RoI)
  - crop and warp *each* region to some fixed size
  - run CNN on each region of interest
  - the classification-head used was SVM with capability of classifying  $C = 21$  different objects
  - R-CNN is slow, with multistage training pipeline
- Fast R-CNN [14],
  - to minimize process time

$$\text{R-CNN} \rightarrow \text{Fast R-CNN} \quad (145)$$

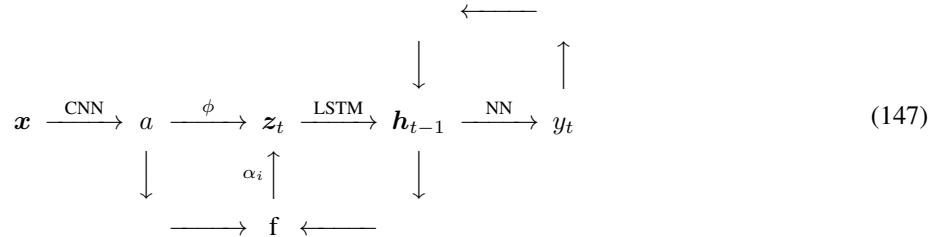
- in fast R-CNN, an image goes through CNN once, not per RoI
- the flow is as follows

$$\begin{array}{ccccc}
 x & \xrightarrow{\text{CNN}} & \text{feature map} & \xrightarrow{\text{region proposal}} & \text{RoIs} \\
 & & & & \downarrow \text{1layer} \\
 & & \text{classifier, loc.} & \xleftarrow{\text{FCs}} & \text{pooled RoI}
 \end{array} \quad (146)$$

- region proposal module is not NN
  - \* region proposal is the bottleneck in fast R-CNN
- coarse localization is inferred from the RoI, fine localization from regression head
- RPN:
  - RPN stands for *region proposal network*
  - RPN takes an image as input and outputs a set of rectangular object proposals, each with a score
  - RPN is modeled as a fully deep convolutional network
  - RPN is trained to produce region proposals directly; no need for external region proposals
  - for each position  $N$  rectangular sizes are considered called *anchor boxes*

- Faster R-CNN [15]:
  - with *faster R-CNN*, region proposal is done by RPN
  - the entire system is a single, unified network for object detection
  - two level of decisions:
    - \* whether there is a RoI
    - \* what the object is
  - results were further improved by using faster R-CNN with ResNet [16]
- YOLO:
  - You only look once (YOLO) is a state-of-the-art, real-time object detection system
- SSD:
  - Single Shot MultiBox Detector, or SSD, is a method for detecting objects in images using a single deep neural network
- Image segmentation:
  - bounding boxes are not practical for some objects such as curvy roads, a forest or a sky
    - \* bounding boxes achieve partial scene understanding
  - image segmentation aims to group perceptually similar pixels into regions and is a fundamental problem in computer vision
  - two segmentation types:
    - 1) semantic segmentation
    - 2) instance segmentation
  - image segmentation is part of scene understanding
- Semantic segmentation:
  - with *semantic segmentation*, every pixel in an image is classified to belong to some object
  - different instances of the same object are not differentiated
  - if different object types are labeled by different colors, then every pixel in an image is colored by a classification
- Semantic segmentation using NN:
  - semantic segmentation can be performed by extracting patches, performing patch classification, and by labeling the patch center-pixel by that object
    - \* this brute-force method is straight forward but tedious and expensive
  - similar to detection (R-CNN), can run CNN on the whole image once, and then pick patches from higher layers
  - *refinement* combines CNN and recurrence by using raw image and output of previous iteration as inputs for next iteration
- Upsampling:
  - *upsampling* addresses the issue of mapping CNN upper layers (with lower dimensions) to higher (pixel) dimensions
    - \* upsampling is the opposite of downsampling that occurs in generic CNNs
  - upsampling can be generated using a union of inner layers using skipping
  - another approach is to use convolution type technique, where a filter will take a single input and will copy it to all  $(n \times n)$  output values, each scaled by some filter weight  $w_{ij}$ 
    - \* a stride length of  $> 1$  is used, at the output rather than the input
    - \* overlapping outputs, from multiple filters, sum their values
  - sometimes referred to as *backward strided convolution*
- FCN:
  - a network constructed exclusively from convolutional layers is called *fully convolutional network*, or FCN
    - \* locality is preserved in such networks & is used in semantic segmentation
  - characteristics of FCNs:
    - 1) replace FC layers with  $(1 \times 1 \times F)$
    - 2) upsampling through the use of transposed convolutional layers
    - 3) uses skip connections for better segmentation
  - an FCN is comprised of two parts: encoder & decoder, see Section V
    - \* encoder:
      - the goal of encoder is to extract features from image
      - transfer learning is usually used, where the encoder can be one of the available CNN networks such as VGG16, see Section IV-F

- the last layer of encoder is  $(1 \times 1 \times F)$  layer
- \* decoder:
  - the decoder upscales the encoder output back to the original size of image
  - decoder uses transposed convolutional layers
- FCN-8:
  - an implementation of FCN from Berkeley
  - the encoder for FCN-8 is the VGG16 model pretrained on ImageNet for classification
  - the fully-connected layers are replaced by 1-by-1 convolutions
- Instance segmentation:
  - *instance segmentation* is similar to semantic segmentation, however different instances of the same object are differentiated
  - sometimes called *simultaneous detection and segmentation* or SDS
  - can use combination of
    - \* region proposal network (RPN)
    - \* capturing RoI and passing them through CNN to generate segmentation masks
    - \* mask our background and predict object class
- Attention:
  - visual attention is the ability to focus on a certain region of an image with high resolution, while perceiving the surrounding image in low resolution
  - in NLP, see Section IX, attention is the ability to focus on certain words in a sentence
    - \* examples include translation, question & answer
- Visual Attention with captioning, a case study [17]:
  - attention can be used in image captioning, where the network flow is as follows



- \*  $a$  is a set of  $L$  features, with  $a_i \in \mathbb{R}^d$
- \* the *context vector*  $z_t$  is a dynamic representation of the relevant part of the image input at time  $t$
- \*  $h_t$  is the hidden state of an LSTM network
- \* the caption is

$$y = \{y_1, \dots, y_C\}, \quad y_i \in \mathbb{R}^K \quad (148)$$

where  $K$  is vocabulary size and  $C$  is caption length

- \*  $f$  is the *attention model*, where ignoring normalization

$$\alpha_{ti} = f(\{a_i\}, h_{t-1}) \quad (149)$$

- \* given the weights  $\alpha_i$

$$z_t = \phi(\{a_i\}, \{\alpha_i\}). \quad (150)$$

- \* *soft attention* corresponds to interpreting  $\{\alpha_{ti}\}$  as probabilities of locations
- \* *hard attention* corresponds to picking a single location, or non-zero  $\alpha_{ti}$ 
  - hard attention gradient descent not effective
  - one proposal is to use reinforcement learning

- Spatial transformer module [18]:
  - the *spatial transformer module*, included in a standard neural network architecture transforms deformed inputs to non-deformed ones by providing spatial transformation capabilities
  - the action of the spatial transformer is conditioned on individual data samples, with the appropriate behaviour learnt during training
  - the spatial transformer module is a dynamic mechanism that can actively spatially transform an image by producing an appropriate transformation for each sample



- the transformation is then performed on the entire feature map (non-locally) and can include scaling, cropping, rotations, as well as non-rigid deformations
- the spatial transformer has three component:
  - \* the *localisation network* predicts a transformation to apply to the input image input sample, this is where attention comes is
  - \* the *grid generator* generates a set of points where the input map should be sampled to produce the transformed output
  - \* given the feature map and the sampling grid, the *sampler* produces the output map, sampled from the input at the grid points
- insert spatial transformer into a classification network
- Dense trajectories in action recognition:
  - dense trajectories is a legacy approach to action recognition
  - idea is to find key points & track them using tracklets
    - \* a tracklet is 15 set of  $(x, y)$  coordinates
  - three steps to do this:
    - \* detect feature points
    - \* track each key point feature using *optical flow*
      - optical fields provide motion field
    - \* extract HOG/HOF/MBH time-based features in the coordinate system of each tracklet
      - HOG is generalized to include time
      - HOF is *histogram of flow*
    - \* the output of histograms was processed by an SVM
- NN & action recognition:
  - one approach to include time, is to extend the convolutional filters, in a CNN, by adding a fourth dimension
    - \* these filters are referred to as 3D filters to emphasize their spatio-temporal nature,
    - \* 3D filters are useful for local, short-term events
    - \* the fusion of time into the network can be done at different levels: slow, early, late or single-frame fusion
  - another approach is to have two networks, one dedicated to images and one to temporal flow, and then fusing the two
  - if global motion need to be captured (i.e. longer than a fraction of a second), then use RNN or LSTM
  - can combine CNN (with or without 3D filters) with LSTMs
  - another proposal is to have each neuron in the CNN to have a local feedback term as well [19]
    - \* this is similar to using a first order IIR filter rather than a an FIR filer

## IX. NATURAL LANGUAGE PROCESSING

- Natural language processing:
  - *natural language processing*, or NLP, is the study of the computational treatment of natural language
    - \* natural means human
  - the goal of NLP is to be able to design algorithms to allow computers to "understand" natural language to perform certain tasks
- Related fields:
  - linguistics
  - theoretical computer science & AI
  - statistics
  - psychology
- Linguistic Definitions:
  - *syntax* refers to grammatical structure,
  - *semantics* refers to the meaning of the vocabulary symbols arranged with that structure
  - *morphology* is the study of word components, how they are formed, & their relationship to other words in the same language
  - *phonetics* is the study of sound
  - *linguistics* is the scientific study of language and its structure, including the study of morphology, syntax, phonetics, and semantics
  - *lexicon* is the set of words used in a language
  - a *corpus* is a collection of written texts
  - *n-gram* is a contiguous sequence of  $n$  items from a given sequence of text or speech
- Phonemes & triphones:
  - a *phoneme* is a single "unit" of sound that has meaning in any language
    - \* there are 44 phonemes in English
  - a *triphone* is a sequence of three phonemes
    - \* there are  $44^3 = 85,184$  triphones in English
  - because of interference between neighboring phonemes, the 44 phonemes are not sufficiently distinguishable
  - triphones address this interference
- Automatic speech recognition:
  - *automatic speech recognition*, or ASR, addresses the problem of converting spoken speech to text
  - ASR's output is a *transcription* which could be NLP's input
  - conventional ASR is based on GMM-HMM model, described shortly
- Challenges in ASR:
  - natural / conversational speech
  - low SNR
  - speaker variability- age, gender, accents
- Traditional acoustic model:
  - feature extraction, such as spectrogram
  - speaker adaptation
  - phoneme prediction, such as GMM
- Spectrogram:
  - *spectrogram* is a two dimensional, time-frequency representation of voice
  - through signal processing, voice is converted to a spectrogram
  - spectrograms are generated using short term Fourier transforms (SFT)
  - a spectrogram has similarities to an image, but its interpretation is not just a classification problem
  - through filtering, background noise and echoes could be removed from a spectrogram
- GMM:
  - the *Gaussian Mixture Model* (GMM), is a statistical acoustic model

$$\text{spectrogram, or observation vector, } x \xrightarrow{\text{GMM}} \text{feature } p(x|s) \quad (151)$$

where  $s$  is a triphone (class)

- $p(x|s)$  is assumed to be a mixture of  $M$  Gaussian distributions

$$p(x|s) = \sum_{m=1}^M c_{sm} N(x; \mu_{sm}, C_{sm}) \quad (152)$$

where for each component  $m$  & class  $s$ , there is an associated weight  $c_{sm}$ , mean  $\mu_{sm}$  and covariance  $C_{sm}$

- GMM training:

- the GMM parameters are trained iteratively using the expectation maximisation (EM) algorithm
- the expectation step estimates  $p_m(x|s)$  for all  $m$  & the data points  $x_i$  in class  $s$
- the maximization step updates the parameters of the model using the following formulas:

$$\begin{aligned} \mu_{sm} &= \frac{\sum_{i=1}^N p_m(x_i|s) x_i}{\sum_{i=1}^N p_m(x_i|s)} \\ C_{sm} &= \frac{\sum_{i=1}^N p_m(x_i|s) (x_i - \mu_{sm})(x_i - \mu_{sm})^T}{\sum_{i=1}^N p_m(x_i|s)} \\ c_{sm} &= \frac{1}{N} \sum_{i=1}^N p_m(x_i|s) \end{aligned} \quad (153)$$

where  $N$  is the number of training points that belong to the class  $j$

- HMM:

- hidden Markov-model (HMM) was a breakthrough technology in speech recognition during the 1970's
- an HMM considers triphones as its states  $s$ , and reconstruct words & sentences
- the trellis of HMM is constrained, i.e. not fully connected
- using merged/shared states, the triphone states are reduced from

$$85,184 \rightarrow 10,000 \quad (154)$$

- HMMs are the most complex part of an ASR system
- training the model involves estimating the transition probabilities, and the emission probability density functions
  - \* this is usually performed by an instance of the EM algorithm known as the Baum-Welch algorithm
  - \* the expectation part of Baum-Welch algorithm is the BCJR algorithm

- HMM-DNN:

- starting around 2011, DNNs replaced GMMs as acoustic models
  - \* DNN replaced the whole acoustic model
- a DNN generates posterior probabilities  $p(s|x)$  rather than  $p(x|s)$
- this hybrid system dominates current ASR

- HMM-free RNN recognition:

- an active research area is the replacement of HMMs with RNNs
  - \* RNNs seem to work well as a replacement for both DNN and HMM
- phonemes can be replaced by characters as the building block
- the acoustic model outputs  $p(a|s)$  where  $a$  is a character
- blanks and junk are represented as underscore
- using dynamic programming, character sequences are converted to words, multiple characters are collapsed, etc.

- Word vectors:

- the English vocabulary  $V$  is around  $|V| = 1$  million words
- if each word forms a vector basis we end up with a one million dimensional space
- since words are related, or correlated, the space could be shrunk to  $d \ll 10^6$  dimensions
  - \*  $d = 100 - 500$  is a good range
  - \* each dimension would encode some meaning that we transfer using speech
  - \* this is known as distributed representation of a word
- thus, we will treat words as a real valued vector

$$w \in \mathbb{R}^d \quad (155)$$

called *word vectors*

- correlation between vectors is a measure to how related these words are

- SVD based methods to generate word vectors:

- see Appendix XV, for SVD review,
- word  $w_j$  is *near* word  $w_i$  if its within some window  $t$  away from  $w_i$ ,
- given a corpus, construct a  $(|V| \times |V|)$  *co-occurrence matrix*  $X$ , where  $X_{ij}$  is the number of times word  $w_j$  was found near  $w_i$ ,

$$X_{ij} = X_{ji}, \quad (156)$$

- perform *singular value decomposition* (SVD) on  $X$  and pick  $d$  vectors based on largest  $d$  eigenvectors,
- this is a global approach and is not practical.

- Language models:

- a *language model* computes the joint probability associated with a sequence of words

$$p(w_1, \dots, w_n), \quad (157)$$

- \* consistent sentences are associated with high probabilities,
- the *unigram* model is very simple,,

$$p(w_1, \dots, w_n) = \prod_{i=1}^n p(w_i), \quad (158)$$

- the *bigram* model is a Markov chain with single memory,

$$p(w_1, \dots, w_n) = \prod_{i=2}^n p(w_i | w_{i-1}). \quad (159)$$

- more generally, the conditional probability is conditioned over a causal or anti-causal window  $m$ ,

$$p(w_t | w_{t-m}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+m}). \quad (160)$$

- Word vectors to conditional probabilities:

- given word vectors  $u$  and  $v$ , the conditional probability  $p(u|v)$  is assumed to be of the form,

$$p(u|v) = \frac{\exp(u^T v)}{\sum_{i=1}^{|V|} \exp(u_i^T v)}, \quad (161)$$

- in other words, the softmax activation (47) of the inner products of two word vectors, is the conditional probability.

- Continuous Bag of Words Model:

- both the Continuous Bag of Words Model and Skip-Gram model were proposed by *Mikolov* et al.,
- in *Continuous Bag of Words Model* (CBOW) model, the center word is predicted from the surrounding context,
- using ML criterion (239) and (160), minimize

$$\min[-\log p(w_t | w_{t-m}, \dots, w_{t+m})], \quad (162)$$

- define

$$\hat{v} \triangleq \frac{w_{t-m} + \dots + w_{t-1} + w_{t+1} + \dots + w_{t+m}}{2m}, \quad (163)$$

- as the name CBOW implies, the order of words in the window does not influence the projection  $\hat{v}$ ,
- relabeling  $u \triangleq w_t$ , the ML criterion (162) is approximated to

$$\min[-\log p(u|\hat{v})], \quad (164)$$

- substituting in (161), we want to minimize,

$$\begin{aligned} & - \frac{\exp(u^T \hat{v})}{\sum_{i=1}^{|V|} \exp(u_i^T \hat{v})} \Rightarrow \\ & - u^T \hat{v} + \log \sum_{i=1}^{|V|} \exp(u_i^T \hat{v}), \end{aligned} \quad (165)$$

- this can be implemented by a single hidden layer that holds  $\hat{v}$  and with an output layer with softmax activation (47).

- Skip-Gram model:

- the *Skip-Gram model*, the surrounding words are predicted from the center word,

- the objective function  $J$ , that we want to minimize becomes

$$\begin{aligned}
& -\log p(w_{c-m}, \dots, w_{c+m} | w_c) \\
& \approx -\log \prod_{j=0, j \neq m}^{2m} p(w_{c-m+j} | w_c) \\
& = -\log \prod_{j=0, j \neq m}^{2m} p(u_{c-m+j} | v_c) \\
& = -\log \prod_{j=0, j \neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{i=1}^{|V|} \exp(u_i^T v_c)} \\
& = -\sum_{j=-m, \neq 0}^m u_{c+j}^T v_c + 2m \log \sum_{i=1}^{|V|} \exp(u_i^T v_c).
\end{aligned} \tag{166}$$

- given the objective function (166), and a window, the gradient w.r.t each center vector  $v$  and outside vectors  $u$  can be computed,
- collectively denote the parameters that we can optimize as  $\sigma \in \mathbb{R}^{2d|V|}$ ,
  - \* each word is associated with two vectors, one center and one outside,
- then,

$$\theta^{\text{new}} = \theta^{\text{old}} - \eta \nabla_{\theta} J(\theta). \tag{167}$$

- Negative sampling:

- since  $|V| \gg 1$  in (166), the resulting algorithm is not practical,
- instead of the softmax function, can train a binary logistic regression for the center word, and a randomly picked  $k$  few word vectors to push down the probability,
- then a cost function that is simpler than (166), can be used,

$$-\log \theta(u_{c-m+j}^T v_c) - \sum_{j=1}^k E_{j \sim p(w)} \log \theta(-u_j^T v_c), \tag{168}$$

- note that  $\theta(-x) = 1 - \theta(x)$ ,
- pick  $p(w)$  to be the unigram distribution (158) to the power 3/4 to amplify rare occurrences.

- GloVe model:

- the GloVe model is an alternate approach of NLP modelling where global and iterative features are combined,
- the cost function is

$$J(\sigma) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(P_{ij})(u_i^T v_j - \log P_{ij})^2, \tag{169}$$

where  $u_i^T v_j$  is a proxy for  $P_{ij}$ ,

- can think of the  $f$  function as a linear function that saturates for very high frequent words.

- Neural network language model:

- with neural networks, both the weights and code vectors are being optimized and are parameters,
- as a result, overfitting is a serious concern,
- various architectures for *neural network language model* (NNLM),
- effective models include deep RNNs, and RNTN.

- Recursive networks:

- recurrent neural networks are based on a chain graph, while recursive networks form a parsing tree,
- unfortunately both systems are abbreviated by RNN,
  - \* when ambiguous, will refer to them as recurrent NN and recursive NN
- a sentence is better modeled using a parsed tree rather than a chain,
  - \* recursion is helpful in describing natural language,
- since chains are special case of a tree, recursive NN's are a superset of recurrent NN's.

- Principle of compositionality:

- phrases, like words, can be mapped to vectors,
- phrase vectors are generated using word vectors in that phrase, and combining them using some rule.

- Standard recursive neural network:

- given a parsed tree, a recursive neural network progresses from the leaves to the root of the tree,
  - \* recursive NN's require a parser to get tree structure,
  - \* when forming a parsing tree, the greedy algorithm can be used, using a score metric,
  - \* the *greedy algorithm* recursively makes locally optimal choices at each stage with the hope of finding a global optimum,
- if  $p_1$  and  $p_2$  are two vectors associated with two phrases, then the concatenated phrase, represented by vector  $q$ , is modeled by a single layer neural network,

$$\begin{aligned} q &= \tanh(W_1 p_1 + W_2 p_2 + b), \\ s &= U^T p, \end{aligned} \quad (170)$$

where  $W_1, W_2 \in \mathbb{R}^{d \times d}$ ,  $p_1, p_2, b, q \in \mathbb{R}^{d \times 1}$ ,

- the associated score is  $s$ ,
- recursive implies all nodes of the tree would use the same  $W_1, W_2$ ,
- the rule (170) can be expressed in a more compact form

$$q = \tanh(Wp + b), \quad (171)$$

where

$$W \triangleq (W_1, W_2), \quad p \triangleq \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}. \quad (172)$$

- Sentiment:

- one application of recursive neural network is to classify a word, phrase, or sentence to a sentiment or a rating,
- the  $n$ -way sentiment  $y^a$  associated with sentence  $a$ , can be extracted using the softmax function (47),

$$y^a = \text{softmax}(W_s a), \quad (173)$$

where  $W_s \in \mathbb{R}^{n \times d}$ , and  $d$  is the dimensionality of  $a$ .

- Determining tree structures using score:

- the score  $s(x, y)$  associated with tree  $y$ , and sentence  $x$ , is the sum of the parsing decision scores at each node (170),

$$s(x, y) = \sum_{n \in \text{nodes}(y)} s_n, \quad (174)$$

- the training set is  $\{x_i, y_i\}$ .

- Recursive RNN variations:

- Syntactically-Untied, or SU-RNN uses different weights  $W$  depending on verb, noun etc.,
- Matrix-Vector, or MV-RNN associates a word with both a vector and a matrix,
  - \* the matrix is used to pre-multiply the vector of another phrase vector.

- Recurrent neural tensor networks:

- *recurrent neural tensor networks*, RNTN were used for sentimental classification,
- the goal is to generalize (171) by incorporating multiplicative interactions in the update rule to allow a greater interactions between the input vectors,
- given tensor  $V \in \mathbb{R}^{2d \times 2d \times d}$ , the generalized update rule (171, 172) is

$$q = \tanh(p^T V p + Wp + b), \quad (175)$$

- when  $V = 0$  the standard recursive neural network is obtained.

- CNN for NLP:

- the idea is to compute all phrase combinations rather than using a parser,
- the convolution operator of a CNN is well suited for considering multiple parsing structures in parallel,
  - \* a convolutional layer computes all sequential  $h$  word-vectors,
  - \* a single convolution operation involves a filter  $w \in \mathbb{R}^{h \times d}$ , which is applied to a window of  $h$  words to produce a scalar feature  $c_i$ ,
  - \* if CNN can handle  $n$  words then a feature map  $c \in \mathbb{R}^{n-h+1}$ ,
  - \* multiple variations are possible,
- a max pooling layer picks the most important feature by the function,

$$\hat{c} = \max\{c\}, \quad (176)$$

- will zero-pad shorter phrases,
- a CNN would use multiple (100s of) different filters, with varying window sizes, resulting in multiple features,
- the pooled layer outputs are passed to a fully connected softmax layer whose output is the probability distribution over labels.
- Machine translation:
  - consider
    - \* English language  $e$  with model  $p(e)$ ,
    - \* French language  $f$ , with translation model  $p(f|e)$ ,
  - using Bayes' rule, French to English translation can be interpreted as maximizing
 
$$\max_e p(e|f) = \max_e p(f|e)p(e), \quad (177)$$
  - i.e., given  $f$ , using  $p(f|e)$  generate multiple candidates for  $e$ 's, then a decoder computes the most probable joint probability.

## X. TOOLS

- GPU:
  - GPU is an effective hardware that accelerates parallel computations,
  - a GPU is composed of multiple *streaming multiprocessors* or SMs,
    - \* each SM includes multiple simple processors and a memory,
  - GPUs need to be explicitly parallel-programmed,
  - compute capability is usually measured with *floating point operations per second*, or FLOPS.
- CPU:
  - Intel Advanced Vector Extensions (AVX) is a set of instructions for doing Single Instruction Multiple Data (SIMD) operations on a CPU,
    - \* AVX2 allows for 32 single precision FLOPS per second, per core
    - \* power consumption is a concern and AVX2 is usually operated at lower frequencies,
    - \* at 2500 MHz, AVX2 generates 80 GFLOPS,
  - FMA instruction set is an extension to SIMD instruction set to perform *fused multiply add* (FMA) operations,
  - AVX2 together with FMA could generate up to 500 GFLOPS of computation from a single CPU.
- GPU versus CPU:
  - GPUs optimize throughput whereas CPU optimizes latency,
  - GPUs use more, but simpler processors than CPUs,
  - FLOPS:
    - \* CPUs  $\sim$  500 GFLOPS,
    - \* GPUs  $\sim$  10 TFLOPS.
- GPU interface:
  - fastest GPU interface is PCIe3  $\times$  16,
    - \* with PCIe3 each channel toggles at 8 GT/s,
    - \* over 16 channels that's 16 GB/s,
    - \* compared to 10 TFLOPS 16 GB/s may not be sufficient,
  - can connect multiple GPUs using high-performance computer-networking communications standard such as *InfiniBand*(IB).
- NVIDIA Titan X:
  - GPU architecture is Pascal,
  - 3584 NVIDIA CUDA cores,
  - Graphics Card Power is 250 W,
  - GPU engine base clock is 1417 MHz,
  - memory 12 GB GDDR5X,
  - 11 TFLOPS.
- System software i.e. drivers for GPU,
  - CUDA:
    - \* NVIDIA's CUDA is general purpose parallel computing tool that allows the user to write a single C code that runs on both CPU and GPU,
    - \* <https://developer.nvidia.com/deep-learning/getting-started>,
    - \* higher-level APIs: cuBLAS, cuFFT, cuDNN, etc, see below.
  - OpenCL:
    - \* similar to CUDA, but runs on any GPU brand,
    - \* usually slower than CUDA.
- CPU  $\leftrightarrow$  GPU interactions in CUDA:
  - CPU is in charge,
  - each CPU and GPU have their dedicated memory,
  - CPU moves data back and forth the two memories using the command *cudaMemcpy*,
  - CPU allocates memory on GPU memory using command *cudaMalloc*,
  - GPU memory capacity may be limited,
  - the host launches kernels on the device.
- Parallel computing using CUDA
  - a *thread* is one path of execution through the code,
  - a CUDA command is of the form



\* name<<<blocks, threads>>>(d-out,d-in)

- number of threads per block is 512 on older GPUs, and 1024 on newer GPUs,
- GPU is responsible of allocating blocks to SMs,
- the blocks and threads are in general three-dimensional entities denoted by

$$\text{dim3}(x, y, z), \quad (178)$$

- a *map* operation with arguments elements and function, applies the function element-wise on each element.

- Data versus model parallelism:

- with *data parallelism*, different data  $x_i$ , are sent to different GPUs,
- with *model parallelism*, the ANN is partitioned over multiple GPUs.

- With distributed asynchronous gradient descent, different workers don't wait on other events to complete but continue processing data asynchronously.

- BLAS:

- the BLAS (Basic Linear Algebra Subprograms) are low-level routines that provide standard building blocks for performing basic vector and matrix operations,
- they are the de facto standard low-level routines for linear algebra libraries,
- BLAS implementations take advantage of special floating point hardware,
- many numerical software applications use BLAS-compatible libraries to do linear algebra computations, including cuBLAS, Mathematica, MATLAB, NumPy, and R.

- Software package trends:

- many of the programming frameworks have performance libraries that harness the HW acceleration,
- NVIDIA's cuDNN, cuBLAS are optimized for DNN,
  - \* cuDNN provides a common set of tools that higher level frameworks can use/reuse,
  - \* cuDNN identifies the 5% of code that (such as convolutions, pooling, activation) takes 80% of run time and delegates the execution of that code to GPU,

- List of software packages:

- the four major packages are Caffe, Torch, Theano, Tensorflow,
- Kaldi, is specialized to speech recognition toolkit, & is written in C++,
- NVIDIA's DIGITS, is an interactive system that provides a quick design capability and visual monitoring tools.

- Caffe:

- <http://caffe.berkeleyvision.org/>,
- developed by U.C. Berkeley,
- written in C++ / CUDA,
  - \* do not need to write code to train,
  - \* Python and Matlab interfaces optional,
  - \* need to write C++ / CUDA for new GPU layers,
- popular for CNN users,
- four main classes:
  - \* *blobs* are tensors that store data, weights & activations,
    - both data and gradients (diffs) are stored,
  - \* *layers* interact with bottom blobs & top blobs,
  - \* a *net* is a bunch of layers, or a graph,
  - \* a *solver* runs the net forward and backward,
- makes use of *protocol buffers*, (.proto) to define, for example, net & solver
- many pre-trained models available through *model zoo*,
  - \* good for fine-tuning existing networks.
- not great for RNNs,
- cumbersome for big networks (GoogLeNet, ResNet).

- Torch:

- developed at NYU,
- written in C & Lua,
  - \* Python and Matlab interfaces optional,
- used & maintained by Facebook & Twitter,,
- uses tensors that are very similar to NumPy,

- \* in addition to tensor, the nn module lets you easily build and train neural nets,
- unlike NumPy, GPU is just a data-type away,
  - \* very easy to run code that runs on GPU,
- uses *modules* instead of net/layers (Caffe),
- many pre-trained models available,
- not great for RNNs.
- Theano:
  - <http://deeplearning.net/software/theano/>,
  - developed at University of Montreal,
  - *Theano* is a Python library for symbolic math, with compiler
    - \* is built on top on NumPy, but also shares similarities with SymPy
    - \* its some combination of a programming language, a compiler and Python library,
    - \* embraces computational graphs,
    - \* automatically computes gradients, through symbolic differentiation,
  - Theano can run on either a CPU or a GPU/CPU,
    - \* C/C++ compiler on CPU
    - \* CUDA/OpenCL compiler for GPU,
    - \* using g++ compiler from TDM-GCC-64 for CPU, see <http://tdm-gcc.tdragon.net>.
  - a *shared variable* lives in computational graph & persist call to call,
    - \* shared variables do not need to be identified as inputs in a function,
  - *Keras* and *Lasagne* are higher level wrappers around Theano, or Tensorflow
    - \* raw Theano is somewhat low-level,
    - \* Lasagne sets up weights & writes update rules for you,
    - \* Keras is more high level than Lasagne,
  - works well for RNN,
  - large models can have long compile times,
  - fatter than Torch; more magic,
  - patchy support for pre-trained models.
- Tensorflow is discussed separately.

## XI. VC GENERALIZATION BOUND

- Hoeffding's inequality:

- Hoeffding's inequality addresses the conditions under which generalization (69) holds,
- Hoeffding's inequality is a form of large number theory that states

$$P[|C_{\text{in}}(h) - C_{\text{out}}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N} \quad (179)$$

- the right hand side of (179) does not depend on  $C_{\text{out}}(h)$ ,
- Hoeffding's inequality can be applied during testing but not during learning.

- Bounds with  $M$  hypotheses:

- when learning, all  $M$  hypotheses may be considered,
- we do not want any of the  $M$  hypothesis deviate from  $E(h)_{\text{out}}$ ,
- then whatever hypothesis  $g$  we chose we are OK,
- using the union bound, the probability that final hypothesis  $g$  is bad (i.e. not tracking) is upper bounded as

$$P[|C_{\text{in}}(g) - C_{\text{out}}(g)| > \epsilon] \leq 2Me^{-2\epsilon^2 N}, \quad (180)$$

- it follows that as  $M \rightarrow \infty$  the inequality gets looser.

- Dichotomies & growth functions:

- $N$ -tuple  $(h(x_1), \dots, h(x_N))$  for classifier  $h \in \mathcal{H}$ , applied on a finite samples  $x_1, \dots, x_N \in \mathcal{X}$ , is called a *dichotomy*,  
\* in other words, a dichotomy is an allowed hypothesis that assigns each vector to a one or a zero,
- the *growth function*  $m_{\mathcal{H}}(N)$ , is the maximum number of distinct dichotomies that can be generated with wisely chosen set of  $N$  vectors,

$$m_{\mathcal{H}}(N) \triangleq \max_{x_1, \dots, x_N \in \mathcal{X}} |\mathcal{H}|, \quad (181)$$

- the growth function is upper bounded by

$$m_{\mathcal{H}}(N) \leq 2^N, \quad (182)$$

- e.g., for the set of convex function hypotheses,

$$m_{\mathcal{H}}(N) = 2^N. \quad (183)$$

- Break points  $\Rightarrow$  polynomial growth functions:

- the smallest value of  $N$  for which  $m_{\mathcal{H}}(N) \neq 2^N$  is called the *break point*,
- theorem: if the break point is less than infinity then  $m_{\mathcal{H}}(N)$  is polynomial in  $N$ ,
- more specifically, *Sauer's Lemma* states that given a break point  $k$ ,

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{k-1} \binom{N}{i} \quad (184)$$

- note that there are no middle points;  $m_{\mathcal{H}}(N) = 2^N$ , or  $m_{\mathcal{H}}(N)$  is polynomial in  $N$  of order  $(k-1)$ .

- Vapnik-Chervonenkis (VC) Inequality:

- the *Vapnik-Chervonenkis Inequality*,

$$P[|C_{\text{in}}(g) - C_{\text{out}}(g)| > \epsilon] \leq 4m_{\mathcal{H}}(2N)e^{-\frac{1}{8}\epsilon^2 N} \quad (185)$$

is a tighter bound than (180),

- compared to (180), the growth function  $m_{\mathcal{H}}(N)$  is substituted for  $M$ ,
- VC inequality implies that as long as there is a break point, generalization from in sample to out of sample is possible.

- VC dimension:

- the *VC dimension* of a hypothesis set  $\mathcal{H}$ , denoted by  $d_{VC}(\mathcal{H})$ , is equivalent to break point, where

$$d_{VC}(\mathcal{H}) \triangleq k-1, \quad (186)$$

- $d_{VC}(\mathcal{H})$  is finite  $\Rightarrow g \in \mathcal{H}$  will generalize,  
\* independent of the learning algorithm,  
\* independent of the input distribution,  
\* independent of target function,
- VC dimension is the effective binary degrees of freedom of a hypothesis,  
\* it measures the effective number of parameters,

- observation: larger VC dimensions need larger example set  $N$  to learn,
- rule of thumb is to use  $N \approx 10 d_{VC}$ .
- VC generalization bound:
  - denote the right hand side of (185) to be  $\delta$ ,

$$\begin{aligned}
 \delta &\triangleq 4m_{\mathcal{H}}(2N)e^{-\frac{1}{8}\epsilon^2 N} \Rightarrow \\
 \ln \frac{\delta}{4m_{\mathcal{H}}(2N)} &= -\frac{1}{8}\epsilon^2 N \Rightarrow \\
 \epsilon &= \sqrt{\frac{8}{N} \ln \frac{4m_{\mathcal{H}}(2N)}{\delta}} \\
 &\triangleq \Omega,
 \end{aligned} \tag{187}$$

where  $\Omega$  is called the *generalization error*,

- it follows that with probability  $\geq 1 - \delta$ ,

$$\begin{aligned}
 |C_{\text{out}} - C_{\text{in}}| &\leq \Omega(N, \mathcal{H}, \delta) \Rightarrow \\
 C_{\text{out}} - C_{\text{in}} &\leq \Omega \Rightarrow \\
 C_{\text{out}} &\leq C_{\text{in}} + \Omega,
 \end{aligned} \tag{188}$$

- theorem: for any tolerance  $\delta > 0$ , with probability  $\geq 1 - \delta$ ,

$$\boxed{C_{\text{out}}(g) \leq C_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln \frac{4m_{\mathcal{H}}(2N)}{\delta}}} \tag{189}$$

- with a larger hypothesis set, we get a better approximation (smaller  $C_{\text{in}}$ ) but a worst generalization error (larger  $\Omega$ ).

- Classification versus estimation:

- VC generalization bound can be generalized from classification problems to also include estimation problems,
- bias and variance analysis is an alternative to VC analysis, that is suited when squared error measure is used as an error metric.

## XII. BIAS & VARIANCE METHOD

- Average hypothesis:

- let  $g^{(\mathcal{D})}(\mathbf{x})$  be the final hypothesis associated with data set  $\mathcal{D}$ ,
- the *average hypothesis*  $\bar{g}(\mathbf{x})$  is defined as

$$\bar{g}(\mathbf{x}) \triangleq E_{\mathcal{D}}[g^{(\mathcal{D})}(\mathbf{x})], \quad (190)$$

- in general  $\bar{g}(\mathbf{x})$  is not known,
- $\bar{g}(\mathbf{x})$  is a function of the specific learning algorithm,
- given  $N$ , it is assumed that  $\bar{g}(\mathbf{x})$  is the best hypothesis a family of hypothesis  $\mathcal{H}$  can generate.

- Bias or deterministic noise:

- define *bias* as

$$\text{bias} \triangleq E_{\mathbf{x}}[(\bar{g}(\mathbf{x}) - f(\mathbf{x}))^2], \quad (191)$$

- the bias term is the part of the target function  $f$  that  $\mathcal{H}$  can not capture,
- an alternative name is *deterministic noise*,
- bias is related to approximation, i.e. on how well the hypothesis fits the data.

- Bias & variance method:

- *bias and variance* analysis is an alternate decomposition of  $C_{\text{out}}$ ,

$$C_{\text{out}}(g^{(\mathcal{D})}) \triangleq E_{\mathbf{x}}[(g(\mathbf{x}) - f(\mathbf{x}) - n)^2] \quad (192)$$

- instead of  $C_{\text{in}}$ , the decomposition is w.r.t. the average hypothesis  $\bar{g}(\mathbf{x})$ ,
- \* as a result, the bias-variance analysis depends on the learning algorithm,
- the *variance* term,

$$\text{var} \triangleq E_{\mathbf{x}}[E_{\mathcal{D}}(g^{(\mathcal{D})}(\mathbf{x}) - \bar{g}(\mathbf{x}))^2], \quad (193)$$

is the variation in error due to different data sets and the corresponding hypotheses choices,

- let  $\sigma_n^2$  denote the variance of stochastic noise,
- by introducing  $\bar{g}(\mathbf{x})$  inside (192), by taking expectations w.r.t. all data sets  $\mathcal{D}$ , and by substituting (190, 191, 193) into (192), we obtain

$$E_{\mathcal{D}}[C_{\text{out}}(g^{(\mathcal{D})})] = \text{var} + \text{bias} + \sigma_n^2, \quad (194)$$

- the bias & variance analysis decomposes  $C_{\text{out}}$  to three components:
  - 1) the variance term quantifies the distance from a hypothesis  $g^{(\mathcal{D})}$  to optimal hypothesis  $\bar{g}$ ,
  - 2) the bias term quantifies the distance from optimal hypothesis  $\bar{g}$  to target function  $f$ ,
  - 3) the noise term quantifies the distance target function  $f$  to observed quantities  $y$ ,
- bias-variance analysis is a conceptual tool; in practice the terms can not be computed.

### XIII. SVM'S & KERNEL METHODS

- Support vector machine classifiers:
  - a *support vector machine*, or SVM, is a learning algorithm for the perceptron
  - SVM is a classifier based on supervised training
  - two other classifiers with supervised learning are covered in Section XV:
    - \* naive Bayes
    - \* decision tree
- Hard versus soft margin:
  - *hard margin* SVM implies training data is linearly separable
  - *soft margin* SVM implies training data is not linearly separable
  - will start with *hard margin*, where in general, an infinite number of linear solutions classify data correctly
- Distance:
  - relabel & shorten  $\mathbf{w}$  by pulling out the bias  $b = w_0$ , and dropping  $x_0 = 1$ 
    - \* the resulting vector  $\mathbf{w}$ , which represents the decision hyperplane, is orthogonal to the hyperplane itself
  - define the *distance*  $d(\mathbf{x}_n, \mathbf{x})$ , between any  $\mathbf{x}_n \in \mathbb{R}^d$  & an arbitrary  $\mathbf{x}$  on hyperplane, as the projection of  $(\mathbf{x}_n - \mathbf{x})$  orthogonal to the decision hyperplane

$$\begin{aligned}
 d(\mathbf{x}_n, \mathbf{x}) &\triangleq \left| \frac{\mathbf{w}^T}{\|\mathbf{w}\|} \cdot (\mathbf{x}_n - \mathbf{x}) \right| \\
 &= \frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T \mathbf{x}_n + b - \mathbf{w}^T \mathbf{x} - b| \\
 &= \frac{|\mathbf{w}^T \mathbf{x}_n + b|}{\|\mathbf{w}\|}
 \end{aligned} \tag{195}$$

- Margin:
  - an SVM determines the weight vector  $\mathbf{w}$  in order to maximize the distance from the decision hyperplane to the nearest point(s)
  - the minimum distance is called the *margin*
  - with SVM, the *margin* around a hyperplane is maximized
  - similar to minmax
- Support vectors:
  - the nearest vectors (points) that define the margin are called the *support vectors* (SV)
    - \* only SVs influence the weights, i.e. learning
    - \* all other points are ignored
- SVM & complexity  $d_{VC}$ :
  - imposing margin around hyperplane reduces the growth function
  - as a result,  $d_{VC}$  gets smaller which should help with generalization
- Normalization:
  - normalize  $(\mathbf{w}, b)$  such that for any  $\mathbf{x}_{SV}$

$$|\mathbf{w}^T \mathbf{x}_{SV} + b| = 1 \tag{196}$$

- since all  $\mathbf{x}_{SV}$  are at same distance from hyperplane, the above normalization applies to all support vectors
- when  $\mathbf{x}_n = \mathbf{x}_{SV}$ , substituting (196) into (195)

$$d(\mathbf{x}_{SV}, \mathbf{x}) = \frac{1}{\|\mathbf{w}\|} \tag{197}$$

- the constraint (196) can be re-interpreted as

$$\begin{aligned}
 \min_{\mathbf{x}_n \in \mathbf{x}_{SV}} |\mathbf{w}^T \mathbf{x}_n + b| &= 1 \Rightarrow \\
 \forall \mathbf{x}_n, y_n (\mathbf{w}^T \mathbf{x}_n + b) &\geq 1
 \end{aligned} \tag{198}$$

- Margin violation with soft-margin:
  - with soft-margin, data points  $(\mathbf{x}_n, y_n)$  are permitted to violate the margin by  $\xi_n > 0$
  - in other words, from (198), training data  $(\mathbf{x}_n, y_n)$  violates the margin by  $\xi_n > 0$ , if

$$y_n (\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n \tag{199}$$

- the *total violation* is

$$\sum_i \xi_i \quad (200)$$

- SVM optimization:

- with hard margin SVM, & under constraint (196), maximizing  $1/\|\mathbf{w}\|$  (197), is equivalent to minimizing

$$C = \frac{1}{2} \mathbf{w}^T \mathbf{w} \quad (201)$$

- with soft margin SVM, the total violation (200) is also included in the cost function

$$C = \min \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_i \xi_i \quad (202)$$

under the two sets of constraints,  $\forall n$

$$\begin{aligned} y_n(\mathbf{w}^T \mathbf{x}_n + b) &\geq 1 - \xi_n \\ \xi_n &\geq 0 \end{aligned} \quad (203)$$

- $C$  is usually determined through cross validation
  - \*  $C$  trades error penalty for stability
  - \* larger  $C$  may minimize error rate on training data but may also overfit
- the associated Lagrangian  $\mathcal{L}(\mathbf{w}, b, \alpha, \xi)$ , with the inequality constraints is known as KKT, and is given by

$$\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_n \xi_n - \sum_n \beta_n \xi_n - \sum_n \alpha_n [y_n(\mathbf{w}^T \mathbf{x}_n + b) - 1 + \xi_n] \quad (204)$$

- we want to minimize  $\mathcal{L}(\mathbf{w}, b, \alpha, \xi)$  w.r.t  $(\mathbf{w}, b)$  and  $\xi$ , but maximize it w.r.t.  $\alpha_n \geq 0$

- $\mathcal{L}(\mathbf{w}, b, \alpha, \xi) \rightarrow \mathcal{L}(\alpha)$ :

- differentiating w.r.t  $(\mathbf{w}, b)$  and  $\xi$

$$\begin{aligned} \nabla_{\omega} \mathcal{L} &= \mathbf{w} - \sum_n \alpha_n y_n \mathbf{x}_n = 0 \\ \frac{\partial \mathcal{L}}{\partial b} &= \sum_n \alpha_n y_n = 0 \\ \frac{\partial \mathcal{L}}{\partial \xi_n} &= C - \alpha_n - \beta_n = 0 \end{aligned} \quad (205)$$

- substituting (205) into (204), SVM optimization is reduced to a maximization over  $\alpha$

$$\mathcal{L}(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n,m=1}^N y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m^T \quad (206)$$

under the constraints

$$\begin{aligned} \forall n, \quad 0 &\leq \alpha_n \leq C \\ \sum_{n=1}^N \alpha_n y_n &= 0 \end{aligned} \quad (207)$$

- Quadratic programming,  $\mathcal{L}(\alpha) \rightarrow \alpha$ :

- the optimization in (206) can be expressed as

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - \mathbf{1}^T \alpha \quad (208)$$

where

$$Q_{ij} \triangleq y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (209)$$

- from (207), the constraints in vector form are

$$\begin{aligned} \mathbf{0} &\leq \alpha \leq C \\ \mathbf{y}^T \alpha &= 0 \end{aligned} \quad (210)$$

- there are tools available to solve such optimization problems, known as *quadratic programming*
- since  $Q$  is  $(N \times N)$ , such an optimization can become problematic for large  $N$

- Classes of  $\alpha$ :
  - $\alpha_n = 0 \Rightarrow \mathbf{x}_n$  is an interior point,
  - $0 < \alpha_n < C \Rightarrow \mathbf{x}_n$  is a *margin support vector*, with  $\xi_n = 0$
  - $\alpha_n = C \Rightarrow \mathbf{x}_n$  is *non-margin support vector*, with  $\xi_n > 0$

- $\alpha \rightarrow \mathbf{w}, b$ :
  - given  $\alpha$ ,  $\mathbf{w}$  is determined from (205)
  - the KKT condition implies  $\alpha_n = 0$  for all  $n$  that are not support vectors,
  - as a result  $\mathbf{w}$  can be expressed as

$$\mathbf{w} = \sum_{\alpha_n > 0} \alpha_n y_n \mathbf{x}_n \quad (211)$$

- in PLA (49), all points that mismatch are updated whereas in (211) only the SV are used to derive  $\mathbf{w}$
- the  $b$  parameter can be solved from any SV, see (204)

$$\begin{aligned} y_i(\mathbf{w}^T \mathbf{x}_i + b) &= 1 \Rightarrow \\ b &= \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i \\ &= \frac{1}{y_i} - \sum_{\alpha_n > 0} \alpha_n y_n \mathbf{x}_n^T \mathbf{x}_i \end{aligned} \quad (212)$$

- SVM in feature space & the kernel:
  - consider a nonlinear space mapping

$$\mathbf{x} \in \mathbb{R}^n \rightarrow \mathbf{z} \in \mathbb{R}^m \quad (213)$$

- \* in general  $m > n$
- \* the goal for this mapping is to make a classification problem linearly separable in the  $z$ -space, whereas it was not in the  $x$ -space
- optimization in  $z$ -space proceeds similar to (208, 210), but with the substitution  $\mathbf{x}_i \rightarrow \mathbf{z}_i$
- since sample size  $N$  does not change, the only difference is the change in the inner products

$$\mathbf{x}_i^T \mathbf{x}_j \rightarrow \mathbf{z}_i^T \mathbf{z}_j \quad (214)$$

inside the Lagrangian (209)

- \* only the inner product of  $\mathbf{z}$  is needed to do the computations, not  $\mathbf{z}$  itself
- \* the dimensionality of  $\mathbf{z}$  is not important
- \* this inner product  $\mathbf{z}_i^T \mathbf{z}_j$  is some function of  $(\mathbf{x}_i, \mathbf{x}_j)$ , called the *kernel*

$$\boxed{\mathbf{z}_i^T \mathbf{z}_j \triangleq K(\mathbf{x}_i, \mathbf{x}_j)} \quad (215)$$

- when used with non-linear transformations, SVM's generate sophisticated boundaries (or hypothesis  $h$ ) that are simply generated (i.e. simple  $\mathcal{H}$ )

- Bound on  $C_{\text{out}}$ :

- theorem: given the sample size  $N$  and number of SV's, the out-of-sample performance can be bounded

$$E[C_{\text{out}}] \leq \frac{E[\#SV's]}{N-1} \quad (216)$$

where  $\#SV's$  is an in-sample determined value that is applied on out-of-sample estimate

- key takeaway is that this bound is independent of dimensionality  $d$ 
  - \* we can transform space so that  $\tilde{d} \rightarrow \infty$ , without impacting  $C_{\text{out}}$
- this is the main theoretical result in support of SVM

- $\mathbf{z}_i^T \mathbf{z}_j$  is sufficient statistics:

- it was observed in (214) that individual  $\mathbf{z}_i$  vectors need not to be known to optimize the Lagrangian, just their inner product
- i.e., when using quadratic programming the  $Q$  matrix components (209), become

$$Q_{ij} \triangleq y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \quad (217)$$



- similarly, only  $z_i^T z_j$  is needed to compute  $g$ , since from (211, 212),  $g(\mathbf{x})$  is given by

$$\begin{aligned}
 & \text{sign}(\mathbf{w}^T \mathbf{z} + b) \\
 &= \text{sign} \left( \sum_{\mathbf{z}_n \text{ is SV}} \alpha_n y_n (\mathbf{z}_n^T \mathbf{z} - \mathbf{z}_n^T \mathbf{z}_i) + \frac{1}{y_i} \right) \\
 &= \text{sign} \left( \sum_{\alpha_n > 0} \alpha_n y_n (K(\mathbf{x}_n, \mathbf{x}) - K(\mathbf{x}_n, \mathbf{x}_i)) + \frac{1}{y_i} \right)
 \end{aligned} \tag{218}$$

- note that the signal goes through two nonlinearities,  $K$  and the  $\text{sign}()$
- as long as we have access to  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_i^T \mathbf{z}_j$ , we do not need to know about  $z$ -space

- Kernel requirements:

- two characteristics desired from a kernel:
  - \* the kernel should be valid, i.e., represent the inner product in some  $\mathcal{Z}$  space
  - \* the kernel should be a good match for the learning problem in hand
    - hand picking a kernel is a disadvantage compared to deep learning
- theorem: Mercer's condition, a kernel is valid iff for any  $\mathbf{x}_1, \dots, \mathbf{x}_N$ 
  - \*  $K(\mathbf{x}_i, \mathbf{x}_j)$  is symmetric, and
  - \* the  $(N \times N)$  matrix with coefficients  $K(\mathbf{x}_i, \mathbf{x}_j)$  is positive semi-definite

- Kernel examples:

- kernels of space  $\mathcal{Z}$ , that supports polynomial order  $Q$ , can be computed efficiently in the form

$$K(\mathbf{x}_i, \mathbf{x}_j) = (b + a \mathbf{x}_i^T \mathbf{x}_j)^Q \tag{219}$$

where  $a$  and  $b$  are scale factors

- the *radial basis function* (RBS) kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) \triangleq e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2} \tag{220}$$

is associated with an infinite dimensional  $\mathcal{Z}$  space

## XIV. SIMILARITY METHODS

- Nearest neighbor method:
  - decode any  $\mathbf{x}$  to its nearest data set of  $N$  elements  $(\mathbf{x}_i, y_i)$
  - in other words, the data output is the same output as its *nearest neighbor* (NN)
  - like ML decoding
  - complexity is proportional to  $N$
- $k$ -nearest neighbor method:
  - find the best  $k$  closest neighbors, and make a decision based on the  $k$  candidates  $y_i$
  - higher values of  $k$  have a smoothing effect that makes the classifier more resistant to outliers
- Bounded distance method:
  - this is similar to bounded distance decoding
  - can think of a cylinder around each training data point
- Radial basis function method:
  - after observing  $N$  data samples, the *radius basis function* hypothesis, or RBF, is

$$h(\mathbf{x}) = \sum_{n=1}^N w_n e^{-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2} \quad (221)$$

- the exponents form the basis functions
- since we have  $N$  equations and  $N$  unknowns, the exact interpolation solution ends up being

$$\mathbf{w} = \Phi^{-1} \mathbf{x} \quad (222)$$

where  $\Phi_{ij} = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$

- Bounded distance as an RBF:
  - when the basis function is chosen to be the cylinder function rather than Gaussian, RBF becomes bounded distance method
  - can think of RBF to be the softened version of bounded distance method
- Similarity methods:
  - the nearest neighbor, bounded distance and the RBF are examples of *similarity methods* since we are comparing how similar data is to the training set
- RBF with  $K$  centers:
  - use  $K \ll N$  centers instead of  $N$ , for better generalization,
  - then the model of (221) changes to

$$h(\mathbf{x}) = \sum_{n=1}^K w_n e^{-\gamma \|\mathbf{x} - \boldsymbol{\mu}_n\|^2}, \quad (223)$$

- the  $K$  centers  $\boldsymbol{\mu}_n$  are new parameters, each being a  $d$  dimensional vector.

- Weights for  $K$ -centers:

- from (223),

$$\Phi \mathbf{w} = \mathbf{y}, \quad (224)$$

where  $\Phi_{ij} = e^{-\gamma \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2}$ ,

- given the centers  $\boldsymbol{\mu}_k$ , or  $\Phi$ , there are  $N$  equations and  $K$  unknowns in (224),
- such a system, can be solved similar to linear regression,
- if  $\Phi^T \Phi$  is invertible,

$$\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}. \quad (225)$$

- Determining  $\gamma$  through EM:
  - $\gamma$  in (223) can be determined iteratively, using *expectation-maximization* (EM) algorithm,
    - \* given  $\gamma$ , solve for  $\mathbf{w}$ ,
    - \* given  $\mathbf{w}$ , minimize error w.r.t.  $\gamma$ .
- RBF versus SVM:
  - when the kernel is chosen as in (220), the SVM solution is in the form, see (218)

$$\text{sign} \left( \sum_{\alpha_n > 0} \alpha_n y_n e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2} + b \right), \quad (226)$$

- similarity with RBF can be noted by comparing (226) to (223).
- Smoothness quantified:
  - smooth function are functions with small derivatives,
  - smoothness can be measured by the metric

$$\sum_{k=0}^{\infty} a_k \int_{-\infty}^{\infty} \left( \frac{d^k h}{dx^k} \right) dx. \quad (227)$$

- RBF and regularization:
  - consider minimizing a function under smoothness (227) constraint,

$$\sum_{n=1}^N (h(x_n) - y_n)^2 + \lambda \sum_{k=0}^{\infty} a_k \int_{-\infty}^{\infty} \left( \frac{d^k h}{dx^k} \right) dx \quad (228)$$

- when this is solved, we get RBF with Gaussian functions.

## XV. MISCELLANEOUS MATHEMATICAL ENTITIES

- Singular value decomposition:

- $\forall (m \times n) \Omega, \exists$ 
  - \* unitary  $(m \times m) U$
  - \* unitary  $(n \times n) V$  and
  - \* non-negative real diagonal  $(m \times n) D$
- such that

$$\Omega = UDV^H \quad (229)$$

- this factorization is called *singular value decomposition* (SVD)

- SVD singular vectors & singular values:

- the columns of  $U$ ,  $\mathbf{u}_k$ , are the eigenvectors of  $AA^T$ 
  - \*  $\mathbf{u}_k$  are called the *left singular vectors*
- the columns of  $V$ , are the eigenvectors of  $A^T A$ 
  - \* the columns of  $V$ ,  $\mathbf{v}_k$ , are called the *right singular vectors*
- $D_{ii} > 0$  are called the *singular values* of  $\Omega$ 
  - \* by convention, order diagonal entries such that

$$D_{i,i} \geq D_{i-1,i-1} \quad (230)$$

- if there are  $r \leq \min(m, n)$  singular values, then the rank of  $\Omega$  in (229) is  $r$
- $D_{ii}$  are the square roots of the nonzero eigenvalues of both  $AA^T$  &  $A^T A$
- if all singular values of  $\Omega$  are unique and non-zero, then its SVD is unique
  - \* this uniqueness is up to the multiplication of a column of  $U$  by a unit-phase factor and simultaneous multiplication of the corresponding column of  $V$  by the same unit-phase factor

- SVD & vector compression:

- let  $D^{(s)}$  be the matrix derived from  $D$ , that keeps only the  $s < r$  largest singular values, replacing the rest with zeros
- from (229), define rank  $s < r$  matrix

$$\begin{aligned} \Omega^{(s)} &\triangleq U D^{(s)} V^H \\ &= \sum_{k=0}^{s-1} \mathbf{u}_k D_{kk} \mathbf{v}_k^T \end{aligned} \quad (231)$$

- each term  $\mathbf{u}_k D_{kk} \mathbf{v}_k^T$  is called *principle image*
- $\Omega^{(s)}$  is the closest rank- $s$  matrix to  $\Omega$ , where the term closest is in term of componentwise Euclidean norm

$$\sum_{i,j} (\Omega_{ij} - \Omega_{ij}^{(s)})^2 \quad (232)$$

- the property of SVD to provide the closest rank- $s$  approximation for a matrix  $\Omega$  can be used to reduce vector dimensionality
  - \* in other words, SVD can be used to compress  $\Omega$
- such a compression is not necessarily the best way to compress images

- Principle component analysis:

- *principle component analysis*, or PCA, reduces the data set dimensionality from  $d$  to  $\tilde{d} < d$
- given  $(N \times d)$  data matrix  $X$ , first the  $(d \times d)$  covariance matrix  $\bar{C}$ , is computed (10)
- $\bar{C}$  is symmetric and has a spectral factorization

$$\bar{C} = V \Lambda V^T \quad (233)$$

- the  $d$  dimensional columns of  $V$  are the eigenvectors of  $\bar{C}$  called *principle axes*
- the  $(d \times d)$  diagonal  $\Lambda$  contains the associated eigenvalues
- choose  $\tilde{d}$  eigenvalues corresponding to the largest  $\tilde{d}$  eigenvalues
- denote the resulting  $(\tilde{d} \times d)$  matrix by  $\tilde{V}$
- the projections of the data on the principal axes are called *principal components*
- any data vector  $\mathbf{x}$  can be compressed to  $\tilde{\mathbf{x}}$  by

$$\tilde{\mathbf{x}} = \tilde{V} \mathbf{x} \quad (234)$$

- Relating SVD to PCA:
  - when  $X = \Omega$  (229),

$$\begin{aligned} X^T X &= (UDV^T)^T (UDV^T) \\ &= VDU^T UDV^T \\ &= VD^2V^T \end{aligned} \quad (235)$$

- comparing (235) to (233),

$$D^2 = \Lambda. \quad (236)$$

- Relative entropy:

- the *relative entropy* of  $p(x)$  with respect to the *entropy measure*  $q(x)$  is defined as

$$D(p||q) \triangleq \sum_{x \in \mathfrak{X}} p(x) \log \frac{p(x)}{q(x)} = E_p \log \frac{p(X)}{q(X)}, \quad (237)$$

- $q(x)$  does not need to be a probability measure,
- when  $q(x) = 1$ , the entropy measure is known as the *uniform measure*,
  - \* with uniform measure, relative entropy reduces to *entropy*,

$$H(X) \triangleq D(X||1), \quad (238)$$

- \* in that sense relative entropy generalizes entropy,
- when  $q(x)$  is a probability measure, then relative entropy is also called *Kullback Leibler distance* between  $p(x)$  and  $q(x)$ ,
  - \*  $D(p||q)$  is a measure of the inefficiency of assuming the distribution is  $q$  when the true distribution is  $p$ ,
  - \*  $D(p||q) \geq 0$ ,
  - \*  $D(p||q) = 0 \Leftrightarrow p = q$ ,
  - \*  $D(p||q)$  is not a true distance since it is not *symmetric* and does not satisfy the *triangle inequality*,
  - \*  $D(p||q)$  is *convex* in the pair  $(p, q)$ .

- Bayesian criterion:

- consider the hypotheses set  $h \in \mathcal{H}$ , & a data set  $\mathcal{D}$ ,
  - \* practical hypotheses sets are infinite,
- in the *Bayesian approach*, the posterior  $P(h|\mathcal{D})$  is computed from prior  $P(h)$ , and the maximum  $P(h = g|\mathcal{D})$  is chosen,
- it assumes the prior distribution on hypothesis set  $P(h)$ , is known,
  - \* this approach is justified when prior is valid or irrelevant.

- Maximum-likelihood criterion:

- for independently generated data  $\{\mathbf{x}_n\}$ , the maximum-likelihood (ML) criterion maximizes the expression

$$\begin{aligned} \max_{\{\mathbf{y}_n\}} \prod_{n=1}^N P(\mathbf{y}_n|\mathbf{x}_n) &\Rightarrow \\ \min_{\{\mathbf{y}_n\}} -\log \left( \prod_{n=1}^N P(\mathbf{y}_n|\mathbf{x}_n) \right) &\Rightarrow \\ \min_{\{\mathbf{y}_n\}} \frac{1}{N} \sum_{n=1}^N \log \frac{1}{P(\mathbf{y}_n|\mathbf{x}_n)} & \end{aligned} \quad (239)$$

- Naive Bayes model:

- *Naive Bayes model* is a subclass of Bayesian Network, BN, that makes "naive assumptions"
- often used in classification problems, where given feature observations  $X_1, X_2, \dots, X_n$  per sample, the model *inferences* the class  $C$

- analytically:

$$\begin{aligned}
& \forall i, j \neq i, (X_i \perp X_j \mid C) \Rightarrow \\
& P(C, X_1, \dots, X_n) = P(C) \prod_{i=1}^n P(X_i \mid C) \Rightarrow \\
& P(c_i \mid x_1, \dots, x_n) \propto P(c_i) \prod_{j=1}^n P(x_j \mid c_i) \Rightarrow \\
& \hat{c} = \arg \max_{c_i} P(c_i) \prod_{j=1}^n P(x_j \mid c_i)
\end{aligned} \tag{240}$$

- $P(c_i)$  is determined using the relative frequency of  $c_i$  from the training data
- *Gaussian Naive Bayes* model assumes  $P(x_j \mid c_i)$  is Gaussian
  - \* use MAP estimation to find  $(\mu_i, \sigma_i)$

- Decision tree:

- the space is partitioned into sub-regions by sequentially & linearly splitting the space based on single features
- the resulting structure is a decision tree
- a decision is designed to maximize the *information gain*, where

$$\text{information gain} = \text{parent-entropy} - (\text{weighted average}) \text{ children-entropy} \tag{241}$$

- Cross-entropy:

- define *cross-entropy* between two probability distributions  $p$  &  $q$  over the same underlying set of events  $X$  to be

$$H(p, q) \triangleq -E_p \log q(X), \tag{242}$$

- expanding (237),

$$\begin{aligned}
D(p \parallel q) &= E_p \log \frac{p(X)}{q(X)} \\
&= E_p \log p(X) - E_p \log q(X) \Rightarrow \\
H(p, q) &= H(p) + D(p \parallel q),
\end{aligned} \tag{243}$$

- the cross entropy between two probability distributions measures the average number of bits needed to identify an event from a set of possibilities.

- ML criterion & cross-entropy:

- consider the ML criterion (239) with scalar  $x_n$ , with  $y_n \in \{0, 1\}$  and in the limit  $N \rightarrow \infty$

$$\begin{aligned}
& \min_{\{y_n\}} \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N \log \frac{1}{P(y_n \mid x_n)} \\
&= \min_{\{y_n\}} \lim_{N \rightarrow \infty} \frac{1}{N} \left( \sum_{n: y_n=1} \log \frac{1}{P(y_n=1 \mid x_n)} + \sum_{n: y_n=0} \log \frac{1}{P(y_n=0 \mid x_n)} \right)
\end{aligned} \tag{244}$$

- define two terms

$$\begin{aligned}
q(x_n) &\triangleq P(y_n = 1 \mid x_n) \\
p &\triangleq \lim_{N \rightarrow \infty} \frac{\sum_{n=1}^N y_n}{N}
\end{aligned} \tag{245}$$

- substitute  $q(x_n)$  into (244)

$$\begin{aligned}
&= \min_p \lim_{N \rightarrow \infty} \left( \frac{1}{N} \sum_{i=1}^N \log \frac{1}{q(x_i)} + \frac{1}{N} \sum_{i=1}^{(1-p)N} \log \frac{1}{1-q(x_i)} \right) \\
&= \min_p \left( p E_X \log \frac{1}{q(X)} + (1-p) E_X \log \frac{1}{1-q(X)} \right)
\end{aligned} \tag{246}$$

- compare this to minimizing the cross-entropy (242)

$$= \min_p E_p \frac{1}{\log q(X)} \tag{247}$$

- Linear regression:

- linear regression has a long history in statistics
- given dataset  $\{(\mathbf{x}_n, y_n)\}$ , with  $y_n \in \mathbb{R}$ , find best linear fit that minimizes mean-square error
- from (25), the in-sample error can be represented as

$$\begin{aligned} C_{\text{in}}(\mathbf{w}) &= \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 \\ &= \frac{1}{N} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \frac{1}{N} (\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y}) \end{aligned} \quad (248)$$

where

- \*  $\mathbf{X}$  is the  $N \times (d+1)$  data matrix whose rows are the inputs  $\mathbf{x}_n$
- \*  $\mathbf{y} = \{y_n\}$  is  $(N \times 1)$
- imposing  $\nabla C_{\text{in}}(\mathbf{w}) = 0$  on (248)

$$\begin{aligned} \nabla(\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w}) &= \nabla(2\mathbf{w}^T \mathbf{X}^T \mathbf{y}) \Rightarrow \\ \mathbf{X}^T \mathbf{X} \mathbf{w} &= \mathbf{X}^T \mathbf{y} \end{aligned} \quad (249)$$

- if  $(d+1) \times (d+1)$   $\mathbf{X}^T \mathbf{X}$  is invertible, a one-shot learning formulation is obtained

$$\begin{aligned} \mathbf{w} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{X}^\dagger \mathbf{y} \end{aligned} \quad (250)$$

where

$$\mathbf{X}^\dagger \triangleq (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \quad (251)$$

is the pseudo-inverse of  $\mathbf{X}$

- Gabor filter:

- in image processing, a *Gabor filter*, named after Dennis Gabor, is a linear filter used for edge detection,
- in the spatial domain, a 2D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave.

- $K$ -means clustering:

- $K$ -means clustering is a method of clustering data into  $K$  classes where each cluster is identified by its center  $\boldsymbol{\mu}_k$ ,
- each center  $\boldsymbol{\mu}_k$  is the representative of a group of data, called *cluster*  $S_k$ ,
- the goal is to minimize

$$\sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2, \quad (252)$$

- since no  $y_n$  is involved in the above minimization, this is an example of unsupervised learning,
  - \* unsupervised learning does not 'corrupt' the data,
- in general this is an NP hard problem.

- Lloyd's algorithm:

- *Lloyd's algorithm* is an iterative solution to the  $K$ -means clustering:
  - \* given  $S_k$ , update  $\boldsymbol{\mu}_k$ ,

$$\mathbf{u}_k = \frac{1}{|S_k|} \sum_{\mathbf{x}_n \in S_k} \mathbf{x}_n, \quad (253)$$

- \* given  $\boldsymbol{\mu}_k$ , update  $S_k$  so that each sample picks the nearest cluster.

## REFERENCES

- [1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.
- [2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, 1958.
- [3] C. Szegedy, W. Liu, *et al.*, "Going deeper with convolutions," 2015.
- [4] C. Szegedy, S. Ioffe, V. Vanhoucke, *et al.*, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016.
- [5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradientbased learning applied to document recognition," 1998.
- [6] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," 2012.
- [7] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013.
- [8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.
- [9] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016.
- [10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," 1997.
- [11] D. Hebb, *The Organization of Behavior*. New York: Wiley & Sons, 1949.
- [12] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," 2014.
- [13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation supplementary material," 2014.
- [14] R. Girshick, "Fast r-cnn," 2015.
- [15] S. Ren, K. He, R. Girshick, , and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.
- [16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.
- [17] K. Xu, J. L. Ba, R. Kiros, K. Cho, *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," 2015.
- [18] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," 2016.
- [19] N. Ballas, L. Yao, C. Pal, and A. Courville, "Delving deeper into convolution networks for learning video representation," 2016.