

CS5339 Machine Learning

Representation III

Lee Wee Sun
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

Semester 2, 2019/20

Outline

1 Ensemble Methods

2 Deep Neural Networks

3 Architecture Design

4 Appendix

Ensemble Methods

- Ensemble methods take a weighted average of outputs of learning algorithms.
- It is often quite effective for improving the performance of the algorithms, e.g. most Kaggle winners use ensemble of predictors.
- Ensembles play multiple roles:
 - It can reduce the variance of the resulting algorithm: averaging helps to eliminate the noise in the combined predictors.
 - By taking a weighted combination of predictors, the combined predictor is able to provide a better approximation of the target function, compared to the original base learning algorithms.

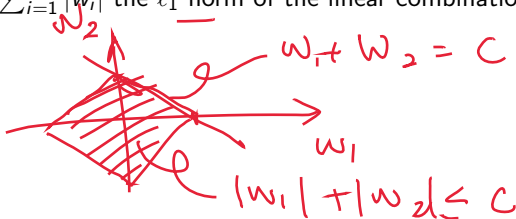
Example: Decision trees require large representation for DISTINCT.
Weighted average of decision trees can small

Ensemble Method in Practice: XGBoost [3]

- Favourite method in Kaggle competitions.
 - Among 29 winning solutions in 2015, 17 used XGBoost.
 - Deep neural networks second most popular, used in 11 solutions.
- Used by every winning team in top 10 of KDDCup 2015.
- Won the 2016 John Chambers Statistical Software Award.
- XGBoost is a tree ensemble method (weighted average of trees).
- More recent popular tree ensemble methods include LightGBM and CatBoost.

Ensemble Methods and Linear Combinations

- We will generalize ensembles into linear combinations with bounded ℓ_1 norm and look at approximation results of the larger class (the results also hold for ensembles).
- Consider $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$ where w_i > 0 and $\sum_{i=1}^{\infty} w_i = 1$. This describes an ensemble of functions.
- We can generalize it by allowing w_i to also be negative and $\sum_{i=1}^{\infty} |w_i| = C$ for some constant C .
 - We call $C = \sum_{i=1}^{\infty} |w_i|$ the ℓ_1 norm of the linear combination.



- Let $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$ be a linear combination with finite ℓ_1 norm
- We can transform it back into an ensemble as follows:
 - If w_i is negative, we can replace g_i with $-g_i$ in f : w_i now becomes positive.
 - By replacing each g_i by Cg_i , we can scale w_i to be w_i/C so that $\sum_{i=1}^{\infty} w_i = 1$

Example: $2g_1 - 3g_2 = 2g_1 + 3(-g_2)$

$$= \frac{2}{5}(5g_1) + \frac{3}{5}(-5g_2)$$

- Transforming the function into an ensemble is often useful for analysis: we can assume that w_i forms a probability distribution of functions – this is used in deriving the approximation results.

Approximation with Square Loss

Let $\|f - g\|_2 = \sqrt{\int_K (f(\mathbf{x}) - g(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x}}$. We will refer to this as the L_2 distance. Let $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$, $C = \sum_{i=1}^{\infty} |w_i|$ and G include all the functions g_i used in f . Assume $\|g_i\| \leq 1$.

Theorem: There is a k term linear combination f_k of functions from G such that $\|f - f_k\|_2^2 \leq C^2/k$.

$$= O(1/k)$$

Proof in Appendix.

- The theorem shows that the approximation error for $\|f - f_k\|$ decays at a rate $O(C/\sqrt{k})$.
- The constant C depends on the function being approximated and can be thought of as a measure of the complexity of f .
- To get an approximation error $\|f - f_k\| \leq \epsilon$, a relatively small number $k = O(C^2/\epsilon^2)$ of functions is required.
 - Useful, e.g. for fast prediction, or for compression.

Exercise 1:

Get a sense of what is good or bad approximation rates. Say something about how $O(1/\sqrt{k}) = O(1/k^{1/2})$ compares to $O(1/k)$ and $O(1/k^{1/d})$ where d is the input dimension. Which rate suffers from the curse of dimensionality?

- To halve the error, $\frac{1}{k}$ need double number of terms
- " " " " , $\frac{1}{\sqrt{k}}$ " 4 times
- " " " " , $\frac{1}{\sqrt[k]{k}}$ $\frac{1}{k^{1/d}}$ " 2^d times

Depends on C as well. C may grow badly with d for some fun classes.

Maximum Norm Approximation

- We have measured the approximation using the L_2 norm

$$\|f\|_2 = \sqrt{\int_K (f(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x}}.$$

- We will now look at approximation using the maximum norm
 $\|f\|_\infty = \max_K |f(\mathbf{x})|$ (also called supremum norm, Chebyshev norm or infinity norm).
- We will consider the case of a finite set K , e.g. a training set.

- We will use **Hoeffding's Inequality**:

Let Z_1, \dots, Z_m be a sequence of i.i.d. random variables and let $\bar{Z} = \frac{1}{m} \sum_{i=1}^m Z_i$. Assume that $E[\bar{Z}] = \mu$ and $\Pr[a \leq Z_i \leq b] = 1$ for every i . Then for any $\epsilon > 0$,

$$\Pr \left[\left| \frac{1}{m} \sum_{i=1}^m Z_i - \mu \right| > \epsilon \right] \leq 2 \exp(-2m\epsilon^2 / (b-a)^2).$$

Example: coin tosses, $Z_i = 1$ if head
 $Z_i = 0$ if tail

$\bar{Z} = \frac{1}{m} \sum Z_i$ is avg number of heads

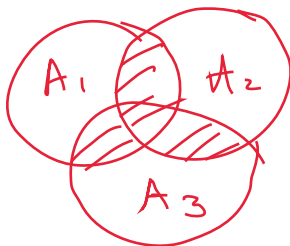
$\mu = \text{prob of head}$

$a=0, b=1$

$$\Pr \left[\left| \frac{1}{m} \sum Z_i - \mu \right| > \epsilon \right] < 2e^{-2m\epsilon^2}$$

- And the **union bound**:

$$\Pr[A_1 \cup A_2, \dots, A_n] \leq \Pr[A_1] + \dots + \Pr[A_n].$$



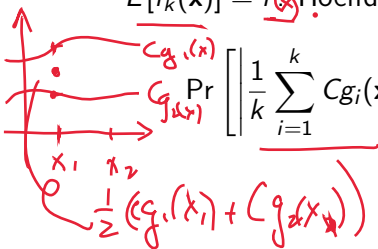
Let $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i \underline{g_i}(\mathbf{x})$ and $\underline{C} = \sum_{i=1}^{\infty} |w_i|$. Further assume $\underline{g_i}$ has been normalized so that $\sup_{\mathbf{x} \in K} |g_i(\mathbf{x})| = 1$, and \underline{G} contains all the functions g_i used in f .

Theorem: There is a \underline{k} term linear combination $\underline{f_k}$ of functions in \underline{G} such that $\|f - f_k\|_{\infty} \leq \frac{(C+1)\sqrt{\log 2|\mathcal{X}|}}{\sqrt{8}\sqrt{k}} = O\left(\frac{\underline{C}\sqrt{\log |\mathcal{X}|}}{\sqrt{k}}\right)$.

- Same $\frac{1}{\sqrt{k}}$ rate as in L_2
- But grows as $|\mathcal{X}|$ grows.

Proof:

- For each $w_i < 0$, multiply the corresponding $g_i(\mathbf{x})$ by -1 so that we can represent the function as $\bar{f}(\mathbf{x}) = \bar{C}(\sum_{i=1}^d \bar{\gamma}_i g_i(\mathbf{x}))$ where $\bar{\gamma}_i = \frac{|w_i|}{C}$. We have $\bar{\gamma}_i > 0$ and $\sum_{i=1}^d \bar{\gamma}_i = 1$, i.e. a distribution of functions.
- Sample k functions \bar{g}_i independently from the distribution and let $\bar{f}_k = \frac{1}{k} \sum_{i=1}^k \bar{C} \bar{g}_i$. For any point $\mathbf{x} \in \mathcal{X}$, we have $E[\bar{f}_k(\mathbf{x})] = \bar{f}(\mathbf{x})$. Hoeffding's inequality gives



$$\Pr \left[\left| \frac{1}{k} \sum_{i=1}^k \bar{C} \bar{g}_i(\mathbf{x}) - \bar{f}(\mathbf{x}) \right| > \epsilon \right] \leq 2 \exp(-8k\epsilon^2 / \bar{C}^2).$$

$\frac{1}{2} (g_1(x_1) + g_2(x_2))$

- Using the union bound, with probability no more than $2|\mathcal{X}| \exp(-8k\epsilon^2/C^2)$, some $\mathbf{x} \in \mathcal{X}$ has $\left| \sum_{i=1}^k Cg_i(\mathbf{x}) - f(\mathbf{x}) \right| > \epsilon$.
*Prob at least $1 - 2|\mathcal{X}| \exp(-8k\epsilon^2/C^2)$
all \mathbf{x} satisfies $\left| \frac{1}{k} \sum Cg_i(\mathbf{x}) - f(\mathbf{x}) \right| < \epsilon$*
- This shows that when $2|\mathcal{X}| \exp(-8k\epsilon^2/C^2) < 1$, there exists a set of g_i 's which gives the max norm approximation of ϵ .
- Solving for ϵ , we have a k term approximation for any

$$\epsilon > \frac{C\sqrt{\log 2|\mathcal{X}|}}{\sqrt{8}\sqrt{k}}.$$

□

Exercise 2:

Consider a function $f(x) = \sum_{i=1}^d w_i g_i(x)$ where $g_i(x) \geq 0$, $w_i > 0$, $\sum_{i=1}^d w_i = 1$, and d is very large.

Is it a good idea to construct an approximation

$h(x) = \frac{1}{k} \sum_{j=1}^k g'_j(x)$ by randomly sampling k function g'_j from the functions (g_1, \dots, g_d) according to the distribution (w_1, \dots, w_d) ?

Answer “Yes” or “No” and add a phrase to explain your reasoning.

Yes,

- In the proof, we look for $2|x| \exp(-\frac{1}{2} k \epsilon^2) < 1$
- Instead of 1, set prob of failure to δ
 $2|x| \exp(-\frac{1}{2} k \epsilon^2) = \delta$
 $\epsilon > \sqrt{\frac{\log 2|x| + \log 1/\delta}{k}}$
 $\sqrt{8/k}$

Using Ensemble for Classification

- We can use the real valued function as a classifier by composing it with a sign function.
- Assume that $f(x) = \sum_{i=1}^{\infty} w_i g_i(x)$ and $\sum_{i=1}^{\infty} |w_i| = 1$.
- Let $f(x)$ classify all instances in \mathcal{X} correctly. $\|w\|_1 = 1$
- We define $\gamma = \min_{\mathbf{x} \in \mathcal{X}} |f(\mathbf{x})|$ to be the ℓ_1 -margin for f ,
- Recall that we defined the margin to be $\gamma = \min_{\mathbf{x} \in \mathcal{X}} |f(\mathbf{x})|$ subject to $\|w\|_2 = 1$. We will call that the ℓ_2 -margin. When it is clear from the context, we will just use the term margin.
- We will also refer to $yf(\mathbf{x})$ as the margin at \mathbf{x} for $y \in \{-1, 1\}$.

Collorary:

Consider a function $f(\mathbf{x})$ of the form $\sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$. Assume $\max_{\mathbf{x} \in K} |g_i(\mathbf{x})| = 1$. Let $\sum_i |w_i| = 1$. Assume f correctly classifies all $\mathbf{x} \in \mathcal{X}$ with margin γ . Then it is possible to correctly classify all the instances in \mathcal{X} using a linear combination of k terms from $\{g_i | i = 1, \dots, \infty\}$ for $k > k_{\min} = \frac{\log(2|\mathcal{X}|)}{2\gamma^2}$.

Proof: The theorem implies that there exists a function g using k terms in the linear combination that can approximate $f(\mathbf{x})$ within γ . Since $|f(\mathbf{x})| \geq \gamma$ for all \mathbf{x} , the approximation classifies all instances in \mathcal{X} correctly when composed with the sign function. \square

- The corollary suggests that classifiers with large margins are easier to approximate.



Large Margin Classifier and Weak Learning

We would like to relate large margin with properties of the component function classes/algorithms.

Definition (Weak Learning): For $\gamma > 0$, an algorithm A is called a γ -weak learning algorithm for a Boolean function f if for any distribution \mathcal{D} over \mathcal{X} , the algorithm takes a set of examples from distribution \mathcal{D} as input and outputs a hypothesis $h \in \mathcal{H}$ with error at most $1/2 - \gamma$, i.e. $\Pr_{\mathcal{D}}(h(x) \neq f(x)) \leq 1/2 - \gamma$.

A weak learning algorithm is guaranteed to find a classifier that does slightly better than chance, regardless of the input distribution.

$$\text{Accuracy} \geq \frac{1}{2} + \gamma$$

Example:



(claim: Alg that use $H = \{h_1, h_2, h_3\}$ to min err on $X = \{x_1, x_2, x_3\}$ is a $\frac{1}{6}$ -weak learner for

$$\gamma = \frac{1}{6} \quad \text{Err} \leq \frac{1}{2} - \frac{1}{6} = \frac{1}{3}$$

Ex: set $p(x_1) = \frac{1}{3}$, $p(x_2) = \frac{1}{3}$, $p(x_3) = \frac{1}{3}$

Select h_3 , $\frac{1}{6}$ -weak learner because h_3 is correct $\frac{2}{3}$ of the time

$$\text{Err} \leq \frac{1}{3}$$

Assume that the component functions are $\{-1, 1\}$ -valued.

Theorem: There is a γ -weak learning algorithm using \mathcal{H} if and only if there is an ensemble of functions from \mathcal{H} with margin 2γ .

~~For~~ Prev example, construct

$$h = \frac{1}{3} h_1 + \frac{1}{3} h_2 + \frac{1}{3} h_3$$



approx
with
margin $1/3$

so $1/6$
weak learn
exist

Proof in the Appendix.

- It is possible to iteratively add weak learning hypotheses to construct the ensemble.
- This is the boosting procedure, to be described later in the course.
- To ensure that the component function can classify substantially better than chance, sometimes fairly powerful component functions are used.
 - Boosting works quite successfully with decision trees as the component functions/weak learners.

Exercise 3:

We will look at the Covertype dataset using boosted decision trees and boosted linear SVM. The dataset predicts forest cover type from GIS features:

<https://archive.ics.uci.edu/ml/datasets/Coverttype>.

The aim is to distinguish lodgepole pine from other forest types (spruce/fir, Ponderosa pine, cottonwood/willow, aspen, Douglas-fir). The features are:

- Elevation / quantitative / meters / Elevation in meters
- Aspect / quantitative / azimuth / Aspect in degrees azimuth
- Slope / quantitative / degrees / Slope in degrees
- Horizontal_Distance_To_Hydrology / quantitative / meters /
Horz Dist to nearest surface water features
- Vertical_Distance_To_Hydrology / quantitative / meters /
Vert Dist to nearest surface water features
- Horizontal_Distance_To_Roadways / quantitative / meters /
Horz Dist to nearest roadway

- Hillshade_9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
- Hillshade_Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer solstice
- Hillshade_3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice
- Horizontal_Distance_To_Fire_Points / quantitative / meters / Horiz Dist to nearest wildfire ignition points
- Wilderness_Area (4 binary columns) / qualitative / 0 (absence) or 1 (presence) / Wilderness area designation
- Soil_Type (40 binary columns) / qualitative / 0 (absence) or 1 (presence) / Soil Type designation
- Cover_Type (7 types) / integer / 1 to 7 / Forest Cover Type designation

Run the experiment. Then comment on the suitability of decision trees and SVM as weak learners for this dataset.

- SVM not that useful with boosting
 - change distribution on training data, fn does not change much
- DT usually ~~have~~ not that stable
 - ensemble often helps

Ensembles in Practice

- Ensembles of decision trees are very successful in practice.
 - Boosting methods explicitly optimize the weights and examples used to learn each component tree.
 - Bagging methods subsample the examples to get a diverse collection of trees and give equal weighting. It is targeted at variance reduction but may also result in improved approximation.
- Appears to improve performance of trees making it competitive with other methods and yet scale quite well to large data sets.
- Besides trees, can also be used with other methods.
- Combining the output of various methods using ensembles often helps in practice, possibly for both variance reduction and improved approximation.

Outline

1 Ensemble Methods

2 Deep Neural Networks

3 Architecture Design

4 Appendix

Deep Neural Networks

- Deep neural networks have achieved some very impressive results recently.
- What do we gain by going from a single hidden layer neural network to a deep network?
 - This is also related to the fundamental question of what can be efficiently computed using serial computation (deep network) versus parallel computation (wide network).

Feedforward Neural Network

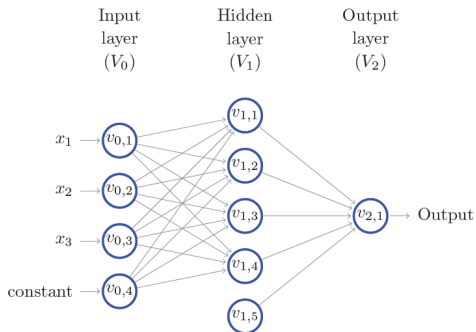


Figure: From SSBD: Single hidden layer neural network.

- We will mostly describe results for feedforward neural networks with linear threshold units. Some of the intuition carry over to other cases.

- A feedforward neural network can be described by a directed acyclic graph $G = (V, E)$.
- Nodes without predecessors in the graph correspond to the input variables x_1, \dots, x_d and let $\mathbf{x} = (x_1, \dots, x_d)$. We consider all inputs to be at level 0 of the graph.
- Nodes without successors in the graph are the outputs.
- Any node that is not an input or an output is a hidden node.
- Let v_1, \dots, v_k be the direct predecessors of node v of the graph. We associate a function $f_v(f_{v_1}(\dots), \dots, f_{v_k}(\dots))$ with the node v .
- We typically consider linear functions, linear threshold functions, linear functions composed with sigmoids, rectified linear functions, radial basis functions, etc. For Boolean functions, we also consider AND, OR and NOT functions.

- The depth of the network is the longest path from any input to any output.
- The level of a node is the longest path from any input to the node.
- The network are often structured such that all direct predecessors of a node at level ℓ is at level $\ell - 1$.
 - In such cases, depth ℓ networks are also called $\ell - 1$ hidden layer networks.
 - We can represent linear (threshold) functions of component functions that we considered in the previous lecture as single hidden layer networks.

VC-Dimension of Neural Networks

We bound the number of functions that can be represented by a neural network with linear threshold units that has $|E|$ weights, and obtain a bound on the VC-dimension.

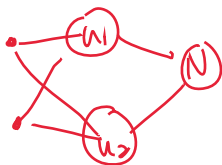
Theorem: The VC-dimension of a neural networks with linear threshold units is $O(|E| \log |E|)$ where $|E|$ is the number of weights in the network.

sigmoid VC Dim $\mathcal{N}(|E|^2)$ to $O(|E|^2 |V|^2)$

Proof:

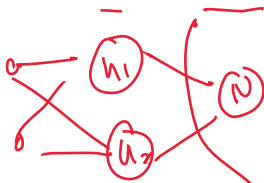
- Recall Sauer's lemma which states that $\tau_{\mathcal{H}}(\underline{m}) \leq (\underline{em}/\underline{d})^d$ where \mathcal{H} has VC-dimension d and the growth function $\tau_{\mathcal{H}}(m)$ counts the number of functions that can be represented by \mathcal{H} when restricted to m inputs.

- Recall that the VC-dimension of a linear threshold unit is the number of weights parameterizing the unit.
- We will bound the growth function for a vector output neural network where every hidden and output units are considered as the output of the network. The growth function can only be smaller when restricted to one output node.



Consider vector output
 $(h_1(x), h_2(x), \underline{N(x)})$
More distinct functions
when treated as
a vector valued fn.

- Let $\tau_G(m)$ denote the growth function of the vector output neural networks G .
- We will argue by induction that $\tau_G(m) \leq \prod_{i=1}^N (em/d_i)^{d_i}$ where N is the number of processing units and d_i is the VC-dimension of unit i .
 - The base case is true for a single processing unit by Sauer's lemma.
- Assume as inductive hypothesis that $\tau_{G'}(m) \leq \prod_{i=1}^k (em/d_i)^{d_i}$ for all networks G' with $k < N$. Consider a network G with N processing units.
 - Remove any unit that does not have a successor and relabel the processing units so that the removed unit is labeled N . We obtain a network G' with $N - 1$ units.



- Continued ...

- Select any m input points. For any fixed function on G' , the outputs of the function and the original inputs form a feature space for the m points. By Sauer's lemma, if we add unit N , we can construct at most $(em/d_N)^{d_N}$ functions on the new feature space.

Handwritten diagram illustrating the feature space construction. The matrix shows the outputs of functions $f_1(x), f_2(x), \dots, f_{d_N}(x)$ and units $N_1(x), N_2(x), \dots, N_2(x)$ for input points x_1, x_2, \dots, x_m . The entries are 0 and 1. To the right, the expression $(em/d_N)^{d_N}$ is written, with a '2' and a 'd' next to it.

- By the inductive hypothesis the number of fixed functions on these m points without unit N is no more than $\prod_{i=1}^{N-1} (em/d_i)^{d_i}$. Hence, adding unit N , the number of possible functions is no more than $\prod_{i=1}^N (em/d_i)^{d_i}$, as required.

- We have shown that

$$\tau_G(m) \leq \prod_{i=1}^N (em/d_i)^{d_i} \leq \prod_{i=1}^N (em)^{d_i} = (em)^{|E|}.$$

- To have m shattered points, we must have

$$2^m \leq (em)^{|E|} \Rightarrow m \leq |E| \log(em) / \log 2.$$

Num of fns
must be \geq
2^{VCdim}

- By Lemma A.2. in SSBD

$$m \leq |E| \log(em) / \log 2 \Rightarrow m = O(|E| \log |E|).$$

□

Using the bound on the VC-dimension, we can show that there are Boolean functions that requires an exponential sized neural network with linear threshold processing units to represent, regardless of the architecture (deep or shallow).

Theorem: (SSBD Theorem 20.2) For every d , let $s(d)$ be the minimal integer such that there is a graph (V, E) with $|V| = s(d)$ such that the hypothesis class \mathcal{H}_G with linear threshold processing units contains all functions from $\{0, 1\}^d$ to $\{0, 1\}$. Then $s(d)$ is exponential in d .

Proof:

- To have all functions from $\{0,1\}^d$ to $\{0,1\}$, \mathcal{H}_G must shatter 2^d input points.
- Hence the VC-dimension must be at least 2^d . But we have shown that the VC-dim is $O(|E| \log |E|) = O(|V|^3)$.
- To have $|V|^3 \geq C2^d$, we must have $|V| \geq C'2^{d/3}$ for some constant C' . □

0 0 0
 0 0 1
 ⋮
 1 1 1

$d=3$

num of fun = 2^d
 $= 8$

On the other hand, any function that is computable in polynomial time also has a representation that is polynomial sized.

Theorem: (SSBD Theorem 20.3) Let $T : \mathbb{N} \rightarrow \mathbb{N}$ and for every n , let \mathcal{F}_n be the class of functions that can be implemented using a Turing machine using a runtime of at most $T(n)$. Then, there exists constants $b, c \in \mathbb{R}_+$ such that for any n there is graph $G_n = (V_n, E_n)$ of size at most $cT(n)^2 + b$ such that \mathcal{H}_{G_n} with linear threshold processing units contains \mathcal{F}_n .

- The result follows from the same result using AND, OR, NOT gates and the fact that the linear threshold units can represent these gates.
- Good news for representation – anything which can be computed quickly can be represented with small network.
- But learning some of these functions may be difficult.
 - Learning the weights is a non-convex optimization problem.

Exercise 4: The result that VC-dim is $O(|E| \log |E|)$ holds for networks with linear threshold units as hidden units. This need not be the case when other types of hidden units are used, e.g. there exists functions of a single parameter that has infinite VC-dim.

However, we always represent parameters with finite precision, e.g. 64 bits, in a computer. Assume a network with arbitrary hidden units parameterized using $|E|$ parameters, where each parameter is discretized to take a finite number of values. What is the VC-dimension of this function class?

- A. VCdim = $O(|E|)$
- B. VCdim can be at least $C|E| \log |E|$
- C. VCdim can be infinite.

Num of fns
 $2^{64|E|}$
 $2^{\text{VCdim}} \leq 2^{64|E|}$
 $\text{VCdim} \leq 64|E|$

Deep Vs Shallow

The PARITY function is a Boolean function from $\{0, 1\}^d$ which returns 1 if the number of the number of occurrences of 1 in the input is odd and returns 0 otherwise.

Theorem: Every *constant depth* circuit with AND, OR, NOT gates for computing the PARITY function requires size exponential in d .

When logarithmic depth is allowed, PARITY can be represented in size polynomial in d .

→ For some functions, network depth can have a substantial impact on the size of representation.

So for logic circuits, there are functions that can be represented compactly if depth is unrestricted but requires exponential size if depth is restricted.

Convolutional Neural Networks

We have earlier seen that PARITY can be represented using a single hidden layer linear threshold network with $d + 1$ hidden units – no need depth.

So does the same phenomenon affect networks used practically?

- Commonly used convolutional neural networks with rectified linear activation function and max pooling is known to be depth efficient – shallow network requires exponential size to realize the same function as some polynomial sized deep networks [4].
- For convolutional arithmetic circuit, where linear activation is used with product pooling, there is complete depth efficiency – all (except for a set of measure zero) functions realizable by a deep network requires exponential size to be realized by a shallow network.

Summary

- There are some functions that requires exponential sized representation for linear threshold networks, even with arbitrary network configuration.
- But any function computable in polynomial time has polynomial sized representation when arbitrary network configuration allowed.
- Deep Vs Shallow
 - Depth 2 networks sufficient to represent all functions.
 - But restricting depth sometimes results in requiring exponentially larger representation – sometimes some sequential computation can be very helpful.

Deep Learning in Practice

- Often useful when dataset is large
 - To exploit the complex architectures, need to provide sufficient data
- When dataset is not very large, methods such as SVM or ensemble of decision trees may work better
 - Easier to train
 - Regularization methods fairly well developed
- Often advantageous when there is structure in the data that we want to exploit. Can design architectures to take advantage of structure in the data – see next section.

Outline

- 1 Ensemble Methods
- 2 Deep Neural Networks
- 3 Architecture Design**
- 4 Appendix

Architecture Design

Polynomial time algorithm can be encoded in a polynomial sized network: how do we use this?

Design heuristics

- Encode knowledge about the problem: represent features, subproblems, etc. that are useful for solving the problem.
- Approximate known computational structures and algorithms.
- Take into account amount of data required to learn.
- Take into account difficulty of optimization problem.

Convolutional Neural Networks









Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$		
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$		
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$		
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$		

Figure: (Image from Wikipedia [7]) Convolution is a useful operation in signal and image processing. Want to encode into network architecture of convolutional neural networks.

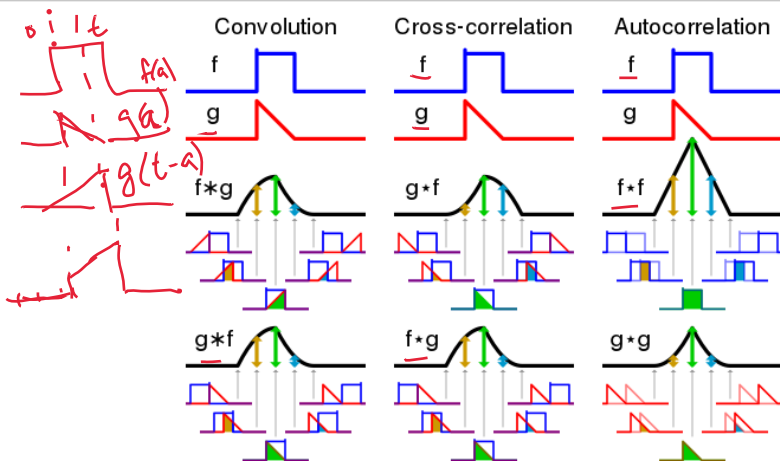


Figure: Image from Wikipedia.

For a signal x and convolution kernel w , the discrete convolution is:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

For a two dimensional image I and two dimensional kernel K , we have:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n).$$

Convolution is commutative, i.e.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n).$$

As the convolutional kernel is learned (as weights), neural network implementations often use cross correlation (without flipping the kernel), instead of convolution.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

Animation of 2D convolution: http://deeplearning.stanford.edu/wiki/images/6/6c/Convolution_schematic.gif

For simplicity of discussion, assume input is an image.

Applying convolution, we get as output another (filtered) image, highlighting certain features (e.g. think of edges).

Some properties of a convolution stage:

- Kernel is usually local and have small support, so quickly computed. *eg 3x3 vs 100x100 image*
- Parameter is shared, i.e. the same kernel is applied across entire image, rather than having a different kernel for each position (reduce the amount of memory to store model, the amount of data required to learn).
- Representation is equivariant with respect to translation, i.e. if the input image is translated, output image is translated by the same amount.

As a design, we have approximated the convolution operation, but left the kernel to be learned from data.

- The output of convolution $S(i, j)$ is composed with a nonlinear unit, such as a rectifier linear unit. This is often called the detector stage.
- Each convolution kernel will give an image as the output – often this is called a channel. With k kernels, each location (i, j) actually has a vector of k units.
- In the third stage, we often use a pooling function. Common pooling functions include
 - max pooling, which selects the maximum value within a rectangular neighbourhood.
 - average of a rectangular neighbourhood
 - weighted average based on distance from the center pixel
- Pooling makes the output approximately invariant to small translations of the input, e.g. the maximum in a small neighbourhood usually does not change when we translate the image slightly.

- Other invariances can also be approximated using pooling.
 - For example, by having other convolution kernels that detects slightly rotated version of the same feature and taking the max over the detector of those convolution kernels, we can achieve invariance to small rotations.
 - These invariances can be learned as the convolution kernels are learned from data.
- The pooling layer is often subsampling, i.e. one out of every k elements is retained.
 - This can improve computational efficiency, often without reducing accuracy as neighbouring pooling elements often have similar values.
 - May also reduce the number of parameters, e.g. if the next layer is fully connected.



- The combination of convolution stage, detection stage and pooling stage is often called a convolution layer.
- Convolution layers are often stacked together, sometimes interleaved with fully connected layers, to form a deep network.
- With a deep network, have hierarchical representation where deeper features are composed from shallower features.

Properties of Convolutional Networks

- Summary: Convolutional network is designed to have several properties.
 - Feature detectors (based on convolution) that detects the same feature in different locations.
 - Same kernel at different locations utilize parameter tying – reduce data required for learning. Local kernels for efficiency.
 - Approximate invariance to small translations and other transformations through pooling.
- Furthermore convolutional neural networks is known to be depth efficient [4].

Object Recognition



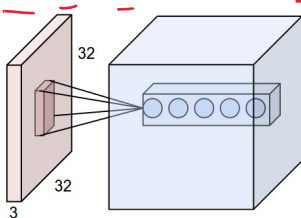
- In object recognition, we want a classifier that takes in an image and outputs the class of the object shown in the image.
- The classifier is often learned using supervised learning.
 - Deep convolutional neural networks has been very successful recently.
 - ImageNet competition: 1000 classes, more than 1 million training images
 - 2010 to 2015 error rates: 28.2, 25.8, 16.4, 11.7, 6.7, 3.6
 - Around human level performance on the dataset now.

Exercise 5: Consider doing text classification with convolutional neural networks. To use a convolutional neural network, we represent each word as an embedded vector of length ℓ . A convolutional kernel with support $k\ell$ can then be used as a feature detector detecting patterns of length k words. For text classification, the position of the pattern within the document is often not very important to the outcome of the classification. Describe how the output of the network can be made invariant to the position of the patterns within the file.

- Do max pooling over all positions
 - If the pattern is present, max pooling a large value will be output
- Use W kernels, can detect W patterns, provides W features

CNN in Computer Vision Practice

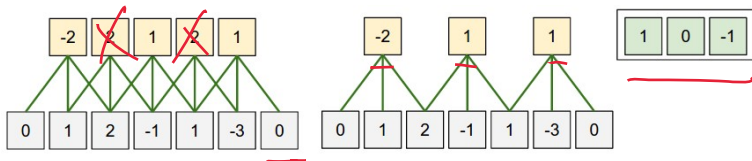
- The input images usually consist of three colour channels (e.g. R, G, B), so is a $W \times H \times 3$ layer, where W is the width, H is the height, and 3 is the *depth* of the layer.
- The convolutional kernel usually has 3D rather than 2D support, e.g. $w \times h \times d$, i.e. the d channels of $w \times h$ pixels. If $w = h = 3$ and $d = 4$, then there are 36 inputs to the filter.
- The output of the convolution stage with k filters/channels is a $W \times H \times k$ layer where k is the depth of the output.



Five filters at the same location.

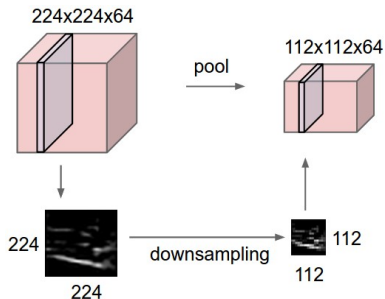
Image source: <http://cs231n.github.io/convolutional-networks/>.

- To get the same output size as the input size, the input is often padded (often with zero) so that the outputs at the boundaries are well defined.
- The output of the convolutional stage is sometimes not computed at each pixel location: the number of pixels moved for each computation is called the stride. This is done in addition to the down-sampling at the pooling stage.



Left part shows one pixel padding on both sides of signal.
Middle part shows stride of 2. The right part is the filter.

Image source: <http://cs231n.github.io/convolutional-networks/>.

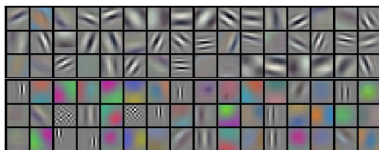


Units from the same channel are pooled and downsampled.

Image source: <http://cs231n.github.io/convolutional-networks/>.

- Pooling is usually done per channel, i.e. neighbouring pixels from the same filter are often pooled together. Subsampling by using stride > 1 is often done.
- A fully connected layer is often used as the last layer.
- Different combinations of convolution, detector and pooling stages are often used in practice, e.g. a few convolution stages may be done before any pooling is done.

- Many different architectures have been successfully used in practice: residual network, one of the recent successful ones, has skip connections between layers.



Features learned by AlexNet.

Image source: <http://cs231n.github.io/convolutional-networks/>.

- The features learned by the earlier convolutional layers have been found to transfer well to other vision tasks, to be used without further training, or as initialization for the new task [9].

Exercise 6: Do you think convolutional neural networks is useful for classifying sentences in natural language? If no, say why. If yes, comment further on which properties are likely to be useful.

- Filters can detect phrases
- Max pooling - invariant to position of phrases
- Deeper network combine info from earlier layers
 - (if subsampled, info at distance n apart can be combined at depth $\log n$).

Exercise 7: The earlier layers of deep neural networks can be considered as features. Deeper units represent more complex features composed from simpler features. These features are learned as part of supervised learning. In this exercise, we consider whether features that are learned for one task can be used for another task. This is sometimes called transfer learning.

A convolutional neural network with a single convolutional layer of 32 filters and a fully connected layer with 128 units is trained on the MNIST digit classification dataset for digits 0 to 4. We then freeze the convolutional layer with the trained weights, and train the rest of the network with digits 5 to 9. Call this the transfer scheme.

Compare the performance of the transfer scheme with training from scratch for 300, 1000, 3000, and 10,000 training examples for the digits 5 to 9.

As training takes some time, we give the results here.

- 300: non-transfer accuracy 0.723, transfer accuracy 0.913
- 1000: non-transfer accuracy 0.868, transfer accuracy 0.973
- 3000: non-transfer accuracy 0.931, transfer accuracy 0.987
- 10,000: non-transfer accuracy 0.986, transfer accuracy 0.991
- Whole training set: non-transfer accuracy 0.991, transfer accuracy 0.994

Comment on the results.

In this case, transfer works.

Recurrent Neural Networks

- Recurrent neural networks are often used for modelling sequences.
 - For example, we may want to learn a mapping from an input sequence $(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)})$ to an output sequence $(y^{(1)}, \dots, y^{(\tau)})$, e.g. input sequence of audio features mapped to output sequence of words in speech recognizer.
 - The sequences may be variable length, i.e. different τ for different sequence.
- At time t , it maintains a vector of hidden units $h^{(t)}$ that stores all the information from the before t that is relevant to the future computation.
 - The hidden units can store a fair amount of information, binary vectors of length k can store 2^k states. The hidden units are real valued (although discretized to a finite number of a computer).

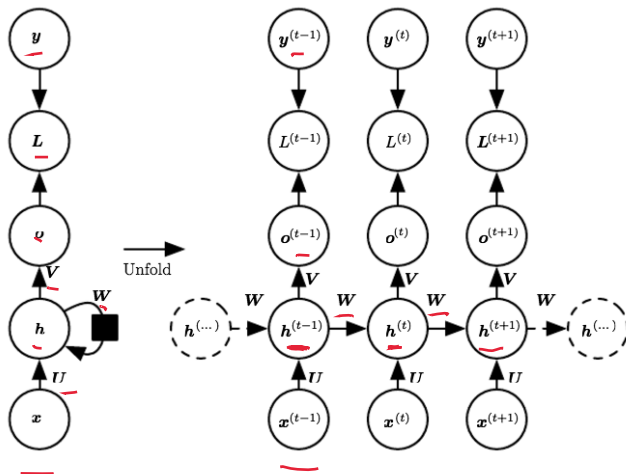


Figure: Image from [6]. Recurrent neural networks. The dependency graph on the left, and unfolded through time on the right.

- The left side of the figure shows the dependency graph of the computational elements. \mathbf{U} , \mathbf{V} , and \mathbf{W} are weight matrices, L specifies the loss function for the pair of output o and label y .
 - \mathbf{h} is a vector of hidden layer units obtained from applying the weights from \mathbf{W} to the value of \mathbf{h} at the previous time step and the weights \mathbf{U} to the value of input \mathbf{x} (typically also composed with activation function).
 - o is the output obtained from applying the weights from \mathbf{V} to the hidden units \mathbf{h} .

- MIN
- The execution at each time step depends on the values of \mathbf{h} at the previous time step. The computation can be unfolded through time as shown on the figure on the right.
 - When unrolled, we get a deep neural network. Size of the deep network depends on length of input.
 - Parameters are shared, i.e. same parameters used in each layer.
 - The hidden layer value (and also the output) at time t depends on the input at time t as well as all previous inputs through the hidden layer value at time $t - 1$

$$\begin{aligned}\underline{\mathbf{h}^{(t)}} &= f(\underline{\mathbf{h}^{(t-1)}}, \underline{\mathbf{x}^{(t)}}) \\ &= \underline{g^{(t)}(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})}.\end{aligned}$$

The hidden layer value $\mathbf{h}^{(t-1)}$ summarizes all the useful information from the past.

- Many variants are possible.
 - One natural variant is to replace the linear functions parameterized by \mathbf{U} , \mathbf{V} , and \mathbf{W} , with deep neural networks to obtain deep recurrent neural networks.
 - Another natural variant is to have chain with a hidden layer that propagates backwards in time from the end of the sequence in addition to the forward propagating hidden layer. The output then depends on the hidden layers from both chains.
 - This allows the output unit to access information from the past, as well as the future – for problems where the output is not required until the end of the input sequence is observed.

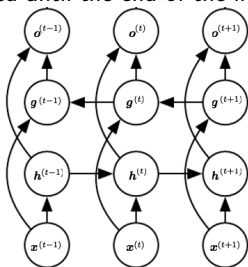


Figure: Image from [6]. Bidirectional recurrent neural networks.

Gated RNN: Long Short Term Memory

- Vanilla RNN tend to have difficulties learning long range dependencies.
 - Gradients tend to vanish over time, or occasionally explode.
 - For single hidden unit RNN, gradient along time roughly proportional to $\prod_i \underline{w_i} \underline{\sigma'(z_i)}$ where w_i and $\sigma'(z_i)$ are weight and gradient of activation fn at time i (more discussion on form of gradient when we discuss gradient descent later).
 - Vanishes if $\underline{w_i}$ or $\underline{\sigma'(z_i)}$ small for many i .
- Simple way to avoid: set $\underline{w_i} = 1$ (unchanged memory content) and $\underline{\sigma'(z_i)} = 1$ (linear or no activation).
- Use gated architectures to approximate this.
 - Gated models allow the hidden states to remain the same over long periods.
 - Gate units are explicitly used to change the states when required – these gates are also learned.
 - LSTM is a well-known gated unit.

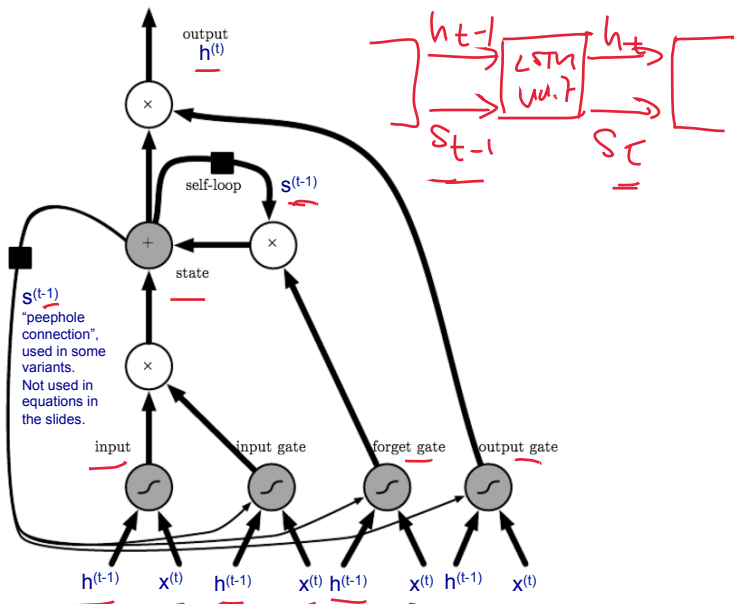


Figure: Image annotated from [6]. LSTM.

- There is a self-loop in the state of the memory \mathbf{s} which keeps the state in the loop unchanged unless changed by the forget gates.
 - The forget units $\mathbf{f}^{(t)}$ act as gating units; they are multiplied with the current state to get a signal that is used for updating the current state

$$\underline{f_i^{(t)}} = \sigma \left(\underline{b_i^f} + \sum_j U_{i,j}^f \underline{x_j^{(t)}} + \sum_j W_{i,j}^f \underline{h_j^{(t-1)}} \right),$$

where forget gates have sigmoid activations that depend on the inputs and state at the previous time step. The sigmoid units have range [0, 1] and will keep the values it multiply with unchanged if it takes the value 1.

- The inputs at each time step is also gated. The external input gates are

$$\underline{g_i^{(t)}} = \sigma \left(\underline{b_i^g} + \sum_j U_{i,j}^g \underline{x_j^{(t)}} + \sum_j W_{i,j}^g \underline{h_j^{(t-1)}} \right).$$

- The states are updated by adding the gated self loop with the gated input

$$\underline{s_i^{(t)}} = \underline{f_i^{(t)}} \underline{s_i^{(t-1)}} + \underline{g_i^{(t)}} \sigma \left(b_i + \sum_j \underline{U_{i,j} x_j^{(t)}} + \sum_j \underline{W_{i,j} h_j^{(t-1)}} \right).$$

- The output **h** of the LSTM unit, can also be shut off by output gates to prevent unit from influencing future (hence affected by future changes) unless necessary

$$\underline{h_i^{(t)}} = \underline{\tanh(s_i^{(t)})} \underline{q_i^{(t)}},$$

$$\underline{q_i^{(t)}} = \sigma \left(b_i^o + \sum_j \underline{U_{i,j}^o x_j^{(t)}} + \sum_j \underline{W_{i,j}^o h_j^{(t-1)}} \right).$$

Speech Recognition

- In July 2015, Google announced that they have cut the transcription error rate in Google Voice by 49% compared to their existing system [8, 12]. Their new system uses LSTM RNN, compared to hidden Markov models with Gaussian mixture models used in their old system.
- In October 2016, Microsoft Research announced that they have achieved human parity in word error rate for speech recognition [13].
 - Error rate of professional transcriber is 5.9% on topic specific conversation and 11.3% on open-ended conversation as measured by Microsoft.
 - The system gave slightly better performance.
 - The Microsoft system used a combination of various techniques including LSTM RNN and convolutional neural networks.

Exercise 8:

In the experiment, we will use a recurrent neural network to add two numbers represented as strings. For example the output of "535+61" should be "596". As the code takes a long time to run, you can run it in your own time.

The code suggests that inverting the input digit sequences may make the problem easier to learn. Suggest why that may be the case.

- Usual seq addition do summation from least significant bit with a carry bit - does not need to be
- If sum from most sig remembered log. bit, need to remember for a long time.

Sequence-to-Sequence and Machine Translation

- Sequence-to-sequence models have been tremendously successful: became standard method for machine translation within a few short years.
 - Consist of an encode network and a decoder network, e.g. encoder takes in an English sentence and decoder generates a French sentence.
 - LSTM models commonly used.

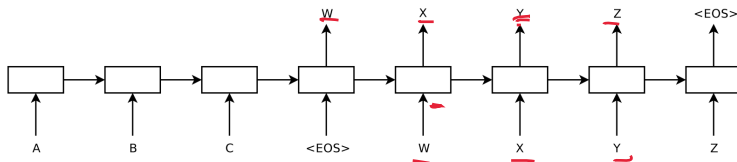


Figure: Image from [10]. Transforms 'ABC' into 'WXYZ'.

- Simple seq2seq models use the final state of the encoder as the starting state of the decoder.
 - Need to store all the information in final state of encoder.
Difficulties with long sentences.
 - More effective to distribute the information in all the hidden states h_1, \dots, h_T of encoder?
 - At each step of the decoder, figure out which of h_1, \dots, h_T are relevant: use attention mechanism.
 - Larger amount of memory for storing information, possibly easier learning (short-cut to long distance information through attention connections).

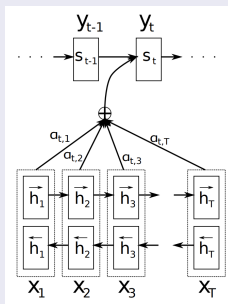


Figure:
Image
from [1].

- Compute next state in decoder using context c_i

$$\underline{s_i} = f(\underline{s_{i-1}}, \underline{y_{i-1}}, \underline{c_i})$$

where c_i approximates relevant portions of encoder states $\underline{h_1, \dots, h_T}$

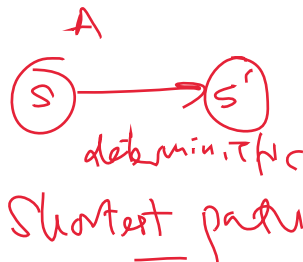
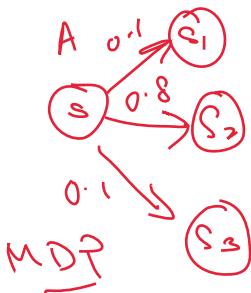
$$\underline{c_i} = \sum_{j=1}^T \underline{\alpha_{ij}} \underline{h_j}$$

$$\alpha_{ij} = \frac{\exp(\underline{e_{ij}})}{\sum_{k=1}^T \exp(\underline{e_{ik}})}, \quad \underline{e_{ij}} = \underline{a}(\underline{s_{i-1}}, \underline{h_j})$$

using some alignment function a learned by a subnetwork.

Value Iteration Network

- Value iteration network encodes the value iteration process for Markov decision processes.
 - Dynamic programming.
 - Essentially a generalization of Bellman-Ford shortest path algorithm (can also encode shortest path this way).
 - In MDP, the transition from one state to another is noisy, unlike in shortest path problems.



- A Markov decision process (MDP) is described by a tuple $\langle S, A, T, R \rangle$ where
 - S denotes the state space
 - E.g. all possible positions a robot could be at.
 - For shortest path problems, this would be all the vertices in the graph.
 - In MDP, the state is observed at every time step.
 - A denotes the action space
 - A robot may have 4 actions, move left, right, forward, and backwards.
 - For shortest path, the actions are to move to connected vertices

- T denotes the transition function $T : S \times A \times S \rightarrow \mathbb{R}$
 - E.g. a robot taking action move forward may have probability p_1 of moving to a state in the forward direction, p_2 of moving to a state on the left, p_3 of moving to a state on the right. The parameters p_1 , p_2 and p_3 can be parameters to be learned.
 - As a special case, the transition may be deterministic, e.g. for shortest path problems.
- R denotes the reward function $R : S \times A \rightarrow \mathbb{R}$
 - E.g. a robot may receive a reward r_G and transition to the terminal state if it takes an action at a goal state, and pays a cost c_1 for movement into open space and a high penalty c_2 for moving into obstacles.
 - In shortest path problems, the weight of the edge is the cost.

- A policy $\pi(s)$ prescribes the action to take in state s .
- The aim of the policy is to maximize the expected sum of (discounted) reward $V^\pi(s) = E^\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s]$ where γ is the discount factor.
 - A discount factor $\gamma < 1$ allows the sum to be finite when we sum to ∞ .
 - We may also be interested in the finite horizon case where we sum from $t = 0$ to K and set $\gamma = 1$. We will consider this case. E.g. in the case of deterministic shortest path, this would give the shortest path to reach the goal using K edges.
- Value iteration is a dynamic programming algorithm for computing the value/policy.

$$V_{t+1}(s) = \max_a Q_t(s, a), \text{ where}$$

$$Q_t(s, a) = \begin{cases} R(s, a) & \text{if } t = 0 \\ R(s, a) + \gamma \sum_{s'} T(s', a, s) V_t(s') & \text{otherwise.} \end{cases}$$

Bellman-Ford

We specialize the algorithm to run on the deterministic shortest path (Bellman Ford) in a graph to help you see how it works.

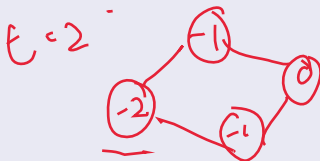
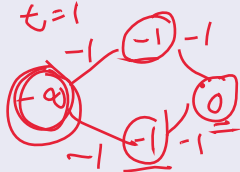
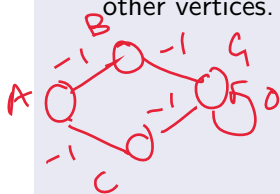
- In this case, the states are vertices in the graph.
- The action space correspond to moving to the vertices.
- The transition is deterministic: if the algorithm specifies to move to vertex v , it will be at vertex v at the next time step. The target vertex t will transition to itself regardless of action taken.
- We can use the reward function to specify the (negative) edge cost in the graph ($-\infty$ when trying to move to a vertex that is not connected to the current vertex). Actions in the target vertex have cost zero (and loops back to itself).
- The discount $\gamma = 1$.

- Consider the value iteration algorithm:

$$\underline{V_{t+1}(s)} = \max_a \underline{Q_t(s, a)}, \text{ where}$$

$$\underline{Q_t(s, a)} = \underline{R(s, a)} + \gamma \sum_{s'} T(s', a, s) \underline{V_t(s')}$$

with $\underline{V_0(s_g)} = 0$ of the goal state s_g and $\underline{V_0(s)} = -\infty$ for all other vertices.



- We can show that the algorithm works correctly using the following inductive hypothesis: after t step of value iteration (equivalent to Bellman-Ford in this case), the values at all vertices are the shortest path costs to reach the goal s_g within t steps.
 - After 1 step, all vertices v , such that (v, s_g) is in the graph (vertices connected to the target vertex) will have the shortest path as they take the cost of the direct connection to the target, whereas those not connected to the target will still have value $-\infty$.
 - Assume that the inductive hypothesis is true after t steps.
 - In the $(t + 1)$ -st step, all vertices will be updated to the shortest path cost to reach s_g within $(t + 1)$ steps by going through the edge that connects to the best vertex within t steps from s_g .

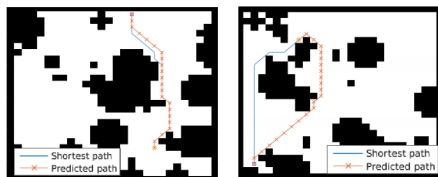
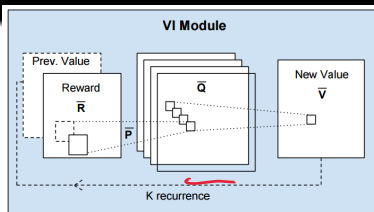


Figure: From [11]. Example application.

We can construct a network similar to a convolutional neural network (CNN) to represent the value iteration algorithm. To make it concrete, we consider the application shown in the figure.

- The states can be discrete grid positions on the map.



From [11]. VI module.

- The Q functions are similar to convolution layers in convolutional neural networks. There is one Q function for each action (usually called a channel in convolutional neural network).

$$Q_t(s, a) = R(s, a) + \gamma \sum_{s'} T(s', a, s) V_t(s')$$

- The Q function is a linear activation function which sums the reward function R with an inner product of the previous value (layer in the network) with the transition probability (weights).

- In the 2-D environment, the transition function only depends on neighbouring states – parameterized and learned, similar to convolution kernels in CNN which are usually local.
- The reward function is a mapping of the image to reward map on the grid positions, learned using some continuous function such as a CNN.
- The convolution layer is max-pooled to select among the actions to produce the value at the next layer of the value function.

$$\underline{V}_{t+1}(s) = \max_a Q_t(s, a).$$

- To run value iteration for k steps, k copies of the VI module are connected together – same parameter values for each copy, rolled out like in RNN.

For training, we can do supervised learning if there is label denoting the best action to take in each state (e.g. expert demonstration, often called imitation learning)

Can also do reinforcement learning to learn the expected sum of current and future rewards.

In either case, network can be trained discriminatively using gradient methods to obtain the advantages of discriminatively learned functions.

- Similarity to convolutional neural networks makes it easy to specify in neural network packages.
- Different pooling pattern compared to commonly used convolutional networks.

Network in this case is deep, to run k iterations of dynamic programming.

Summary

We have seen examples of architecture designs and explored some of the design considerations for the architectures.

- Convolutional neural networks.
- Recurrent neural networks
 - LSTM
 - Attention mechanism
- Value iteration networks

Given a task of interest, how should you design an architecture for it?

Readings

- SSBD Chapter 20 on neural networks
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville, “Deep Learning”, <http://www.deeplearningbook.org/>

References I

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).
- [2] Andrew R Barron. “Universal approximation bounds for superpositions of a sigmoidal function”. In: *IEEE Transactions on Information theory* 39.3 (1993), pp. 930–945.
- [3] Tianqi Chen and Carlos Guestrin. “XGboost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SigKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, pp. 785–794.

References II

- [4] Nadav Cohen and Amnon Shashua. “Convolutional rectifier networks as generalized tensor decompositions”. In: *International Conference on Machine Learning*. 2016, pp. 955–963.
- [5] Yoav Freund and Robert E Schapire. “Game theory, on-line prediction and boosting”. In: *Proceedings of the ninth annual conference on Computational learning theory*. ACM. 1996, pp. 325–332.
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [7] *Kernel (Image Processing)*. [Online: [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))], accessed July 2017].

References III

- [8] *Neon prescription... or rather, New transcription for Google Voice.* [Online:
<https://googleblog.blogspot.sg/2015/07/neon-prescription-or-rather-new.html>, accessed July 2017].
- [9] Ali Sharif Razavian et al. “CNN features off-the-shelf: an astounding baseline for recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.
- [10] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.
- [11] Aviv Tamar et al. “Value iteration networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2154–2162.

References IV

- [12] *The neural networks behind Google Voice transcription.*
[Online:
<https://research.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html>, accessed July 2017].
- [13] Wayne Xiong et al. "Achieving human parity in conversational speech recognition". In: *arXiv preprint arXiv:1610.05256* (2016).

Outline

- 1 Ensemble Methods
- 2 Deep Neural Networks
- 3 Architecture Design
- 4 Appendix**

Proofs (Ensembles)

Let $\|f - g\|_2 = \sqrt{\int_K (f(\mathbf{x}) - g(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x}}$. We will refer to this as the L_2 distance. Let $f(\mathbf{x}) = \sum_{i=1}^{\infty} w_i g_i(\mathbf{x})$, $C = \sum_{i=1}^{\infty} |w_i|$ and G include all the functions g_i used in f . Assume $\|g_i\| \leq 1$.

Theorem [2]: There is a k term linear combination f_k of functions from G such that $\|f - f_k\|_2^2 \leq C^2/k$.

Proof:

- For each $w_i < 0$, multiply the corresponding $g_i(\mathbf{x})$ by -1 so that we can represent the function as $f(\mathbf{x}) = C(\sum_{i=1}^d \gamma_i g_i(\mathbf{x}))$ where $\gamma_i = \frac{|w_i|}{C}$. We have $\gamma_i > 0$ and $\sum_{i=1}^d \gamma_i = 1$, i.e. a distribution of functions.
- Sample k functions g_i independently from the distribution and let $f_k = \frac{1}{k} \sum_{i=1}^k C g_i$. We have $E[f_k] = f$. This gives

$$E\|f_k - f\|^2 = E\left\|\frac{1}{k} \sum_{i=1}^k C g_i - \frac{1}{k} \sum_{i=1}^k f\right\|^2$$

$$\begin{aligned} &= \frac{1}{k^2} E \left\| \sum_{i=1}^k (Cg_i - f) \right\|^2 \\ &= \frac{1}{k^2} E \int_{\mathbf{x} \in K} \left(\sum_{i=1}^k (Cg_i(\mathbf{x}) - f(\mathbf{x})) \right)^2 p(\mathbf{x}) d\mathbf{x} \\ &= \frac{1}{k^2} E \int_{\mathbf{x} \in K} \left(\sum_{i=1}^k (Cg_i(\mathbf{x}) - f(\mathbf{x})) \right) \left(\sum_{j=1}^k (Cg_j(\mathbf{x}) - f(\mathbf{x})) \right) p(\mathbf{x}) d\mathbf{x} \\ &= \frac{1}{k^2} E \int_{\mathbf{x} \in K} \sum_{i=1}^k \sum_{j=1}^k (Cg_i(\mathbf{x}) - f(\mathbf{x})) (Cg_j(\mathbf{x}) - f(\mathbf{x})) p(\mathbf{x}) d\mathbf{x} \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{k^2} \sum_{i=1}^k \left(\int_{\mathbf{x} \in K} E(Cg_i(\mathbf{x}) - f(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} \right. \\
&\quad \left. + \sum_{j \neq i} \int_{\mathbf{x} \in K} E(Cg_i(\mathbf{x}) - f(\mathbf{x})) E(Cg_j(\mathbf{x}) - f(\mathbf{x})) p(\mathbf{x}) d\mathbf{x} \right) \\
&= \frac{1}{k^2} \sum_{i=1}^k \int_{\mathbf{x} \in K} E(C^2 g_i(\mathbf{x})^2) - 2f(\mathbf{x}) E(Cg_i(\mathbf{x})) + f(\mathbf{x})^2 p(\mathbf{x}) d\mathbf{x} \\
&= \frac{1}{k^2} \sum_{i=1}^k \int_{\mathbf{x} \in K} E(C^2 g_i(\mathbf{x})^2) - f(\mathbf{x})^2 p(\mathbf{x}) d\mathbf{x} \\
&= \frac{1}{k} (E\|Cg\|^2 - \|f\|^2) \leq \frac{C^2}{k}
\end{aligned}$$

- Since the expected value is less than or equal to $\frac{C^2}{k}$, there exists a function f_k such that $\|f - f_k\|^2 \leq C^2/k$.



Relating Weak Learning to Margin

We will need the following:

Definition (Two-player zero-sum game): A two-player zero-sum game is defined by a matrix \mathbf{M} . A row player selects an action (row) using a row distribution p and a column player selects an action (column) using column distribution q . Matrix entry $M_{i,j}$ denotes the *loss* of the row player when the row player chooses row i and the column player chooses column j .

Example: The loss matrix for “Rock, Paper, and Scissors”.

	R	P	S
R	0	1	-1
P	-1	0	1
S	1	-1	0

If the row player plays by sampling from distribution p and the column player plays by sampling from distribution q , the expected loss is $\sum_{i,j} p_i q_j \mathbf{M}_{ij} = p^T \mathbf{M} q$.

The row player wants to minimize expected loss, while the column player wants to maximize.

Assume sequential play, where the row player selects p first. The column player will then want to select q (achievable by deterministically selecting a column) such that it satisfies

$$\max_q p^T \mathbf{M} q.$$

Given this, the row player wants to select p such that will minimize its maximum loss, i.e. p satisfies

$$\min_p \max_q p^T \mathbf{M} q.$$

This is called the *minimax strategy*.

Similarly, if the column player selects q first, it will want to play the *maximin strategy*, i.e. select q to satisfy

$$\max_q \min_p p^T \mathbf{M} q.$$

We expect the second player to have an advantage, i.e.

$$\max_q \min_p p^T \mathbf{M} q \leq \min_p \max_q p^T \mathbf{M} q.$$

Surprisingly, the order of play does not matter when the players choose optimally (but they have to play a mixed strategy, i.e. sample from a distribution, instead of a deterministic strategy).

Theorem (von Neumann's Minimax Theorem):

$$\max_q \min_p p^T \mathbf{M} q = \min_p \max_q p^T \mathbf{M} q = v,$$

where v is the value of the game.

We relate the margin to the weak learning parameter [5].

Theorem: There is a γ -weak learning algorithm using \mathcal{H} if and only if there is a positive linear combination of functions from \mathcal{H} with margin 2γ .

Proof:

We will consider each row as an instance in \mathcal{X} and each column as a hypothesis $h \in \mathcal{H}$. If hypothesis j classifies instance i correctly, let $M_{ij} = 1$, otherwise $M_{ij} = -1$.

Let $\mathbb{1}_i(x)$ be the indicator vector for row i and $\mathbb{1}_j(h)$ be the indicator vector for column j . Using the minimax theorem, and noting that the second player can optimally play a deterministic strategy, we have

$$\begin{aligned}\min_p \max_j p^T \mathbf{M} \mathbb{1}_j(h) &= \min_p \max_q p^T \mathbf{M} q \\ &= \max_q \min_p p^T \mathbf{M} q \\ &= \max_q \min_i \mathbb{1}_i(x)^T \mathbf{M} q = v.\end{aligned}$$

We now consider the implications on the classifier margin. Note that

$$p^T \mathbf{M} \mathbb{1}_j(h) = \Pr_p(h(x) = f(x)) - \Pr_p(h(x) \neq f(x)).$$

We previously noted $\min_p \max_j p^T \mathbf{M} \mathbb{1}_j(h) = \nu$, showing that there exists p such that $\Pr_p(h(x) = f(x)) - \Pr_p(h(x) \neq f(x)) \leq \nu$ for all h , with equality achieved at some h . Substituting $\Pr_p(h(x) \neq f(x)) = 1 - \Pr_p(h(x) = f(x))$ and solving, we get

$$2 \Pr_p(h(x) = f(x)) - 1 \leq \nu$$

$$\Pr_p(h(x) = f(x)) \leq 1/2 + \nu/2$$

The definition of weak learner implies that there is a h such that $\Pr_p(h(x) = f(x)) \geq 1/2 + \gamma$. When that is true, we have $\nu \geq 2\gamma$.

We also know that there exists a q such that

$$\mathbb{1}_i(x)^T \mathbf{M}q = \Pr_q(h(x) = f(x)) - \Pr_q(h(x) \neq f(x)) \geq \nu$$

for all i .

The expression $\Pr_q(h(x) = f(x)) - \Pr_q(h(x) \neq f(x))$ is just the weighted sum of the output of the functions and is hence the margin.

As $\nu \geq 2\gamma$, the margin is at least 2γ , relating the margin to weak learning.

Conversely, we can argue that when the margin is 2γ , there is a γ -weak learning algorithm.

If the margin is 2γ , then $\mathbb{1}_i(x)^T \mathbf{M} q \geq 2\gamma$ for all i , implying that $p^T M q \geq 2\gamma$ for any p .

This implies that there exists a column j such that $p^T M \mathbb{1}_j(h) \geq 2\gamma$, i.e.

$$p^T \mathbf{M} \mathbb{1}_j(h) = \Pr_p(h(x) = f(x)) - \Pr_p(h(x) \neq f(x)) \geq 2\gamma.$$

Substituting $\Pr_p(h(x) \neq f(x)) = 1 - \Pr_p(h(x) = f(x))$ and solving, we get

$$2 \Pr_p(h(x) = f(x)) - 1 \geq 2\gamma$$

$$\Pr_p(h(x) = f(x)) \geq 1/2 + \gamma.$$