Computational and Statistical Learning Theory

TTIC 31120

Prof. Nati Srebro

Lecture 17:

Stochastic Optimization

Part II:

Neural Networks

Online vs Statistical Learning

- Online Regret of learning rule $A: \mathcal{Z}^* \to \overline{\mathcal{H}}$

$$\forall_{z_1,z_2,\ldots,z_m} \frac{1}{m}\sum_{t=1}^{m} \ell(A(z_1..z_{t-1}), z_t) \leq \inf_{h\in\mathcal{H}} \frac{1}{m}\sum_{t=1}^{m} \ell(h, z_t) + Reg(m)$$

- Statistical (PAC) "Regret" / excess error:

$$\forall_{\mathcal{D}(\mathcal{Z})} \forall^{\delta}_{S\sim\mathcal{D}^m} L_{\mathcal{D}}\big(A(S)\big) \leq \inf_{h\in\mathcal{H}} \frac{1}{m}\sum_{t=1}^{m} L_{\mathcal{D}}(h) + \epsilon(m,\delta)$$

Low Statistical Regret $\not\Rightarrow$ Low Online Regret

A has Low Online Regret $\Rightarrow$ $\overline{A}$ has Low Statistical Regret

Online ⇒ Batch

- For a convex learning problem (i.e. $\overline{\mathcal{H}}$ is convex and $\ell(h,z)$ is convex in h):

$$\overline{A}(z_1,\dots,z_m)=\frac{1}{m}\sum_{t=1}^{m}A(z_1,\dots,z_{t-1})$$

> $\overline{A}(S)$- Batch Conversion of online rule A
>
> Input: training set $S=\{z_1,z_2,\dots,z_m\}$
>
> 1. Run A on $z_1,\dots,z_m$ to obtain $h_1,h_2,\dots,h_{m+1}$
> 2. Return $\overline{h}_m=\frac{1}{m}\sum_{t=1}^{m}h_t$

- Theorem:

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[L_{\mathcal{D}}\left(\overline{A}(S)\right)\right]\leq\inf_{h\in\mathcal{H}}L_{\mathcal{D}}(h)+Reg(m)$$

- If $|\ell(h,z,)|\leq a$,

$$\forall^{\delta}_{S\sim\mathcal{D}^m}L_{\mathcal{D}}\left(\overline{A}(S)\right)\leq\inf_{h\in\mathcal{H}}L_{\mathcal{D}}(h)+Reg(m)+3a\sqrt{\frac{\log{}^{1}/_{\delta}}{m}}$$

Online ⇒ Statistical

	Statistical	Online
Finite cardinality	$\sqrt{\frac{\log\lvert\mathcal{H}\rvert}{m}}$	$\sqrt{\frac{\log\lvert\mathcal{H}\rvert}{m}}$
Finite VC-dimension	$\sqrt{\frac{VCdim}{m}}$	Not sufficient for diminishing regret
Linear in $\mathbb{R}^d$	$\sqrt{\frac{d}{m}}$	1
ℓ_2 margin γ	$\sqrt{\frac{1/\gamma^2}{m}}$	$\sqrt{\frac{1/\gamma^2}{m}}$
$\lVert w\rVert_2 \le B$, $\lVert\phi(x)\rVert_2 \le 1$, loss is 1-Lipschitz	$\sqrt{\frac{B^2G^2}{m}}$	$\sqrt{\frac{B^2G^2}{m}}$
$\lVert w\rVert_1 \le B$, $\lVert\phi(x)\rVert_\infty \le 1$, loss is 1-Lipschitz	$\sqrt{\frac{B^2G^2\log d}{m}}$	$\sqrt{\frac{B^2G^2\log d}{m}}$

Implications of Online-to-Batch

- Anything online learnable is also statistically learnable, with same regret/excess error
  - Converse not true!
  - Important: online learnable $\not\Rightarrow$ ULLN nor learnability with ERM.

- More computationally efficient learning methods
  - E.g. Perceptron(+online-to-batch) vs SVM

- Useful even as optimization approach!

Stochastic Optimization

$$\min_{w \in \mathcal{W}} F(w) = \mathbb{E}_{z \sim \mathcal{D}}[f(w, z)]$$

based only on stochastic information on F

- Only unbiased estimates of $F(w), \nabla F(w)$
- No direct access to F

E.g., fixed $f(w, z)$ but $\mathcal{D}$ unknown

- Optimize $F(w)$ based on iid sample $z_1, z_2, \ldots, z_m \sim \mathcal{D}$
- $g = \nabla f(w, z_t)$ is unbiased estimate of $\nabla F(w)$

- Traditional applications
  - Optimization under uncertainty
    - Uncertainty about network performance
    - Uncertainty about client demands
    - Uncertainty about system behavior in control problems
  - Complex systems where its easier to sample then integrate over z

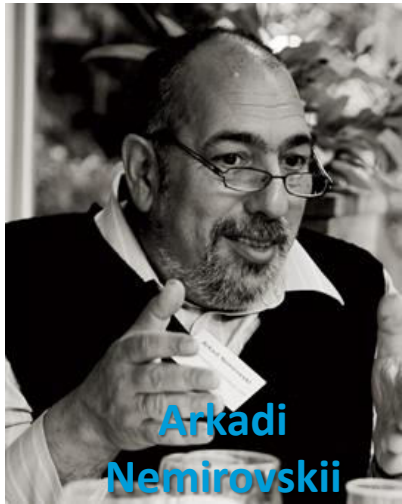
Machine Learning *is* Stochastic Optimization

$$\min_h L(h) = \mathbb{E}_{z\sim\mathcal{D}}[\ell(h,z)] = \mathbb{E}_{x,y\sim\mathcal{D}}[loss(h(x),y)]$$

- Optimization variable: predictor h
- Objective: generalization error $L(h)$
- Stochasticity over $z = (x, y)$

"General Learning" ≡ Stochastic Optimization:


Valdimir Vapnik


Arkadi Nemirovskii

Two Approaches to Stochastic Optimization / Learning

$$\min_{w \in \mathcal{W}} F(w) = \mathbb{E}_{z \sim \mathcal{D}}[f(w, z)]$$

- Empirical Risk Minimization (ERM) / Sample Average Approximation (SAA):
  - Collect sample $z_1, \ldots, z_m$
  - Minimize $F_S(w) = \frac{1}{m} \sum_i f(w, z_i)$
  - Analysis typically based on Uniform Convergence

- Stochastic Approximation (SA): [Robins Monro 1951]
  - Update w_t based on z_t
    - E.g., based on $g_t = \nabla f(w, z_t)$
  - E.g.: stochastic gradient descent
  - Online-to-batch conversion of online algorithm…

Stochastic (Sub)-Gradient Descent

Online Gradient Descent
[Zinkevich 03]

online2stochastic
[Cesa-Binachi et al 02]

Stochastic Gradient Descent
[Nemirovski Yudin 78]

Optimize $F(w) = \mathbb{E}_{z\sim\mathcal{D}}[f(w,z)]$ s.t. $w \in \mathcal{W}$

1. Initialize $w_1 = 0 \in \mathcal{W}$
2. At iteration $t = 1,2,3,\ldots$
   1. Sample $z_t \sim \mathcal{D}$ (Obtain g_t s.t. $\mathbb{E}[g_t] \in \partial F(w_t)$)
   2. $w_{t+1} = \Pi^{\mathcal{W}}(w_t - \eta_t \underbrace{\nabla f(w_t, z_t)}_{g_t})$
3. Return $\overline{w}_m = \frac{1}{m}\sum_{t=1}^{m} w_t$

If $\|\nabla f(w,z)\|_2 \leq G$ then with appropriate step size:

$$\mathbb{E}[F(\overline{w}_m)] \leq \inf_{w\in\mathcal{W},\|w\|_2\leq B} F(w) + O\left(\sqrt{\frac{B^2G^2}{m}}\right)$$

SGD for SVM

$$\min L_S(w) \;\; s.t. \;\; \|w\|_2 \leq B$$

Use $g_t = \nabla_w loss^{hinge}\left(\langle w_t, \phi_{i_t}(x)\rangle; y_{i_t}\right)$ for random i_t

Initialize $w^{(0)} = 0$

At iteration t:

- Pick $i \in 1 \ldots m$ at random
- If $y_i \langle w^{(t)}, \phi(x_i)\rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_i \phi(x_i)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$
- If $\left\|w^{(t+1)}\right\|_2 > B$, then $w^{(t+1)} \leftarrow B \frac{w^{(t+1)}}{\left\|w^{(t+1)}\right\|_2}$

Return $\overline{w}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$

$$\|\phi(x)\|_2 \leq G \rightarrow \|g_t\|_2 \leq G \rightarrow L_S\left(\overline{w}^{(T)}\right) \leq L_S(\hat{w}) + \sqrt{\frac{B^2 G^2}{T}}$$

(in expectation over randomness in algorithm)

Stochastic vs Batch

- Intuitive argument: if only taking simple gradient steps, better to be stochastic

- To get $L_S(w) \leq L_S(\hat{w}) + \epsilon_{opt}$:

	#iter	cost/iter	runtime
Batch GD	B^2G^2/ϵ_{opt}^2	md	$\boldsymbol{m}d\frac{B^2G^2}{\epsilon_{opt}^2}$
SGD	B^2G^2/ϵ_{opt}^2	d	$d\frac{B^2G^2}{\epsilon_{opt}^2}$

- What about $L(w)$, which is what we really care about?
- How small should ϵ_{opt} be?
- Comparison to methods with a $\log 1/\epsilon$ dependence that use the structure of $L_S(w)$ (not only local access)?

Overall Analysis of $L_{\mathcal{D}}(w)$

- Recall for ERM: $L_{\mathcal{D}}(\widehat{w}) \leq L_{\mathcal{D}}(w^*) + 2\sup_w |L_{\mathcal{D}}(w) + L_S(w)|$

$\widehat{w} = \arg\min_{\|w\| \leq B} L_S(w)$

$w^* = \arg\min_{\|w\| \leq B} L_{\mathcal{D}}(w)$

- For ϵ_{opt} suboptimal ERM $\bar{w}$:

$$L_{\mathcal{D}}(\bar{w}) \leq \underbrace{L_{\mathcal{D}}(w^*)}_{\epsilon_{aprox}} + \underbrace{2\sup_w |L_{\mathcal{D}}(w) - L_S(w)|}_{\epsilon_{est} \leq 2\sqrt{\frac{B^2G^2}{m}}} + \underbrace{(L_S(\bar{w}) - L_S(\widehat{w}))}_{\epsilon_{opt} \leq \sqrt{\frac{B^2G^2}{T}}}$$

- Take $\epsilon_{opt} \approx \epsilon_{est}$, i.e. $\#iter\ T \approx sample\ size\ m$
- To ensure $L_{\mathcal{D}}(w) \leq L_{\mathcal{D}}(w^*) + \epsilon$:

$$T, m = O\left(\frac{B^2R^2}{\epsilon^2}\right)$$

Direct Online-to-Batch: SGD on $L_{\mathcal{D}}(w)$

$$\min_w L_{\mathcal{D}}(w)$$

use $g_t = \nabla_w hinge(y\langle w, \phi(x)\rangle)$ for random $y, x \sim \mathcal{D}$

➔ $\mathbb{E}[g_t] = \nabla L_{\mathcal{D}}(w)$

Initialize $w^{(0)} = 0$

At iteration t:

- Draw $x_t, y_t \sim \mathcal{D}$
- If $y_t\langle w^{(t)}, \phi(x_t)\rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_t \phi(x_t)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$

Return $\overline{w}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$

$$L_{\mathcal{D}}\left(\overline{w}^{(T)}\right) \leq \inf_{\|w\|_2 \leq B} L_{\mathcal{D}}(w) + \sqrt{\frac{B^2 G^2}{T}}$$

➔ $m = T = O\left(\frac{B^2 G^2}{\epsilon^2}\right)$

SGD for Machine Learning

$$\min_w L(w)$$

Direct SA (online2batch) Approach:

Initialize $w^{(0)} = 0$

At iteration t:

- Draw $x_t, y_t \sim \mathcal{D}$
- If $y_t \langle w^{(t)}, \phi(x_t) \rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_t \phi(x_t)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$

Return $\bar{w}^{(T)} = \frac{1}{T} \sum_{t=1}^{T} w^{(t)}$

- Fresh sample at each iteration, $m = T$
- No need to project nor require $\|w\| \leq B$
- Implicit regularization via early stopping

SGD on ERM:

$$\min_{\|w\|_2 \leq B} L_S(w)$$

Draw $(x_1, y_1), \ldots, (x_m, y_m) \sim \mathcal{D}$

Initialize $w^{(0)} = 0$

At iteration t:

- Pick $i \in 1 \ldots m$ at random
- If $y_i \langle w^{(t)}, \phi(x_i) \rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_i \phi(x_i)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$
- $w^{(t+1)} \leftarrow proj\ w^{(t+1)}\ to\ \|w\| \leq B$

Return $\bar{w}^{(T)} = \frac{1}{T} \sum_{t=1}^{T} w^{(t)}$

- Can have $T > m$ iterations
- Need to project to $\|w\| \leq B$
- Explicit regularization via $\|w\|$

SGD for Machine Learning

$$\min_w L(w)$$

Direct SA (online2batch) Approach:

Initialize $w^{(0)} = 0$

At iteration t:

- Draw $x_t, y_t \sim \mathcal{D}$
- If $y_t \langle w^{(t)}, \phi(x_t) \rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_t \phi(x_t)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$

$\eta_t = \sqrt{B^2/G^2 t}$

Return $\bar{w}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$

$$L(\bar{w}^{(T)}) \leq L(w^*) + \sqrt{\frac{B^2 G^2}{T}}$$

SGD on ERM:

$$\min_{\|w\|_2 \leq B} L_S(w)$$

Draw $(x_1, y_1), \ldots, (x_m, y_m) \sim \mathcal{D}$

Initialize $w^{(0)} = 0$

At iteration t:

- Pick $i \in 1 \ldots m$ at random
- If $y_i \langle w^{(t)}, \phi(x_i) \rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_i \phi(x_i)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$
- $w^{(t+1)} \leftarrow proj\ w^{(t+1)}\ to\ \|w\| \leq B$

Return $\bar{w}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$

$$L(\bar{w}^{(T)}) \leq L(w^*) + 2\sqrt{\frac{B^2 G^2}{m}} + \sqrt{\frac{B^2 G^2}{T}}$$

$w^* = \arg\min_{\|w\| \leq B} L(w)$

SGD for Machine Learning

$$\min_w L(w)$$

Direct SA (online2batch) Approach:

Initialize $w^{(0)} = 0$

At iteration t:

- Draw $x_t, y_t \sim \mathcal{D}$
- If $y_t \langle w^{(t)}, \phi(x_t) \rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_t \phi(x_t)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$

$\eta_t = \sqrt{B^2/G^2 t}$

Return $\bar{w}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$

- Fresh sample at each iteration, $m = T$
- No need to project nor require $\|w\| \leq B$
- Implicit regularization via early stopping

SGD on RERM:

$$\min L_S(w) + \frac{\lambda}{2}\|w\|$$

Draw $(x_1, y_1), \ldots, (x_m, y_m) \sim \mathcal{D}$

Initialize $w^{(0)} = 0$

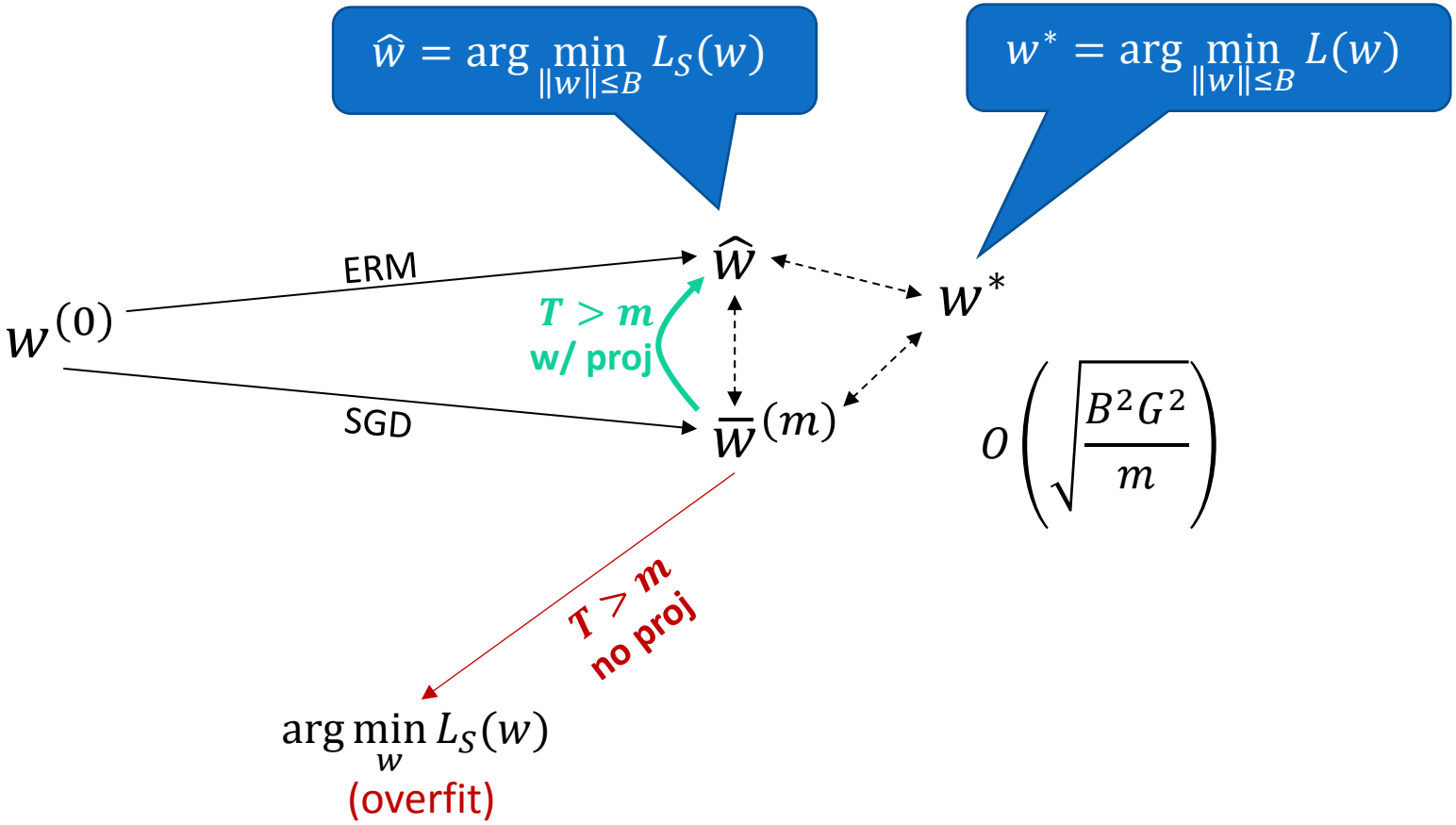
At iteration t:

- Pick $i \in 1 \ldots m$ at random
- If $y_i \langle w^{(t)}, \phi(x_i) \rangle < 1$,
  $w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_i \phi(x_i)$
  else: $w^{(t+1)} \leftarrow w^{(t)}$
- $w^{(t+1)} \leftarrow w^{(t+1)} - \lambda w^{(t)}$

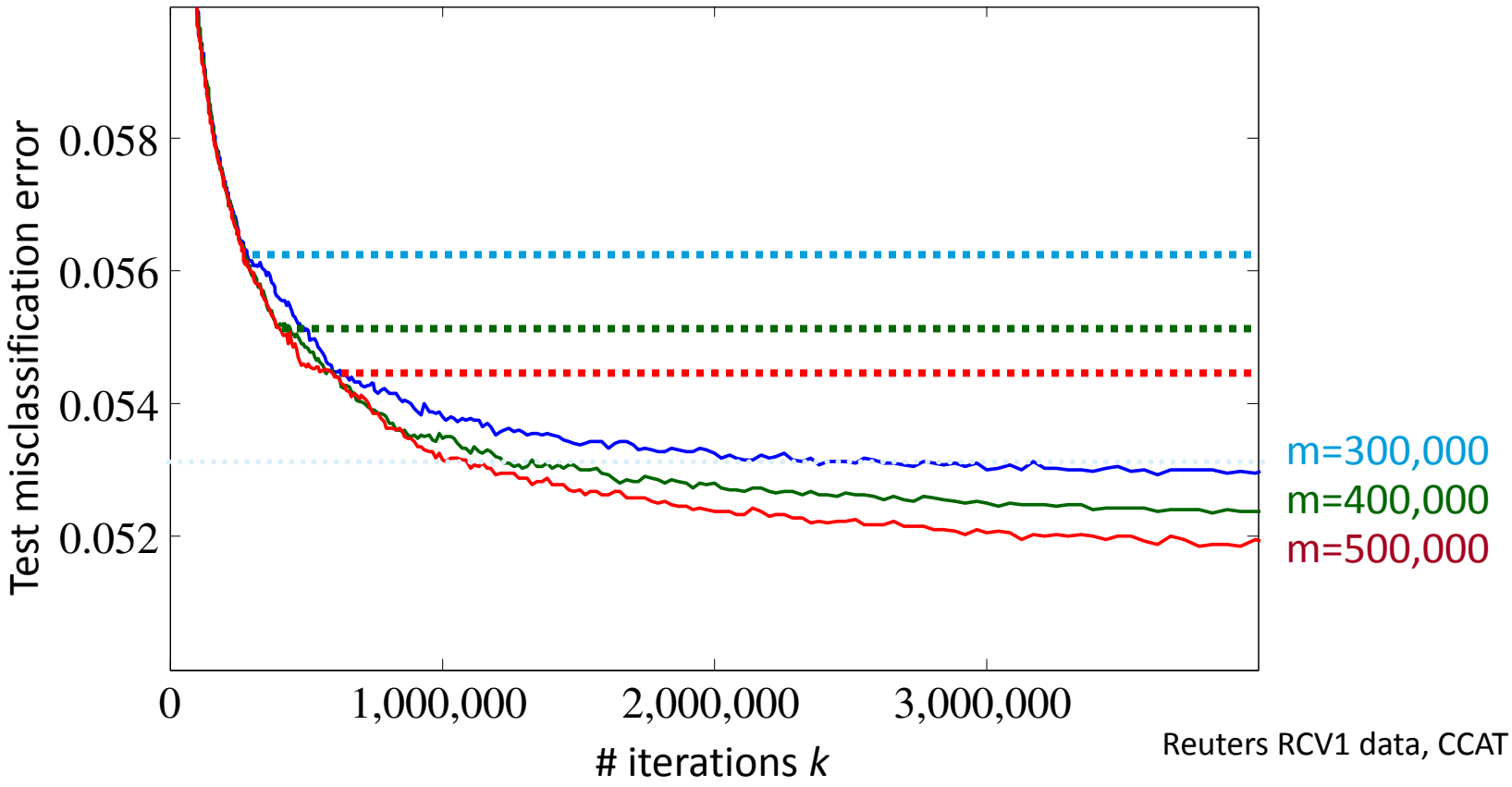
Return $\bar{w}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$

- Can have $T > m$ iterations
- Need to shrink w
- Explicit regularization via $\|w\|$

SGD vs ERM


$\widehat{w} = \arg\min_{\|w\| \le B} L_S(w)$
$w^* = \arg\min_{\|w\| \le B} L(w)$
$w^{(0)}$
ERM
SGD
$\widehat{w}$
$\overline{w}(m)$
w^*
$T > m$
w/ proj
$O\left(\sqrt{\frac{B^2 G^2}{m}}\right)$
$T > m$
no proj
$\arg\min_w L_S(w)$
(overfit)

Mixed Approach: SGD on ERM



Reuters RCV1 data, CCAT task

- The mixed approach (reusing examples) can make sense
- Still: fresh samples are better
  -) With a larger training set, can reduce generalization error faster
  -) *Larger* training set means *less* runtime to get target generalization error

Online Optimization vs Stochastic Approximation

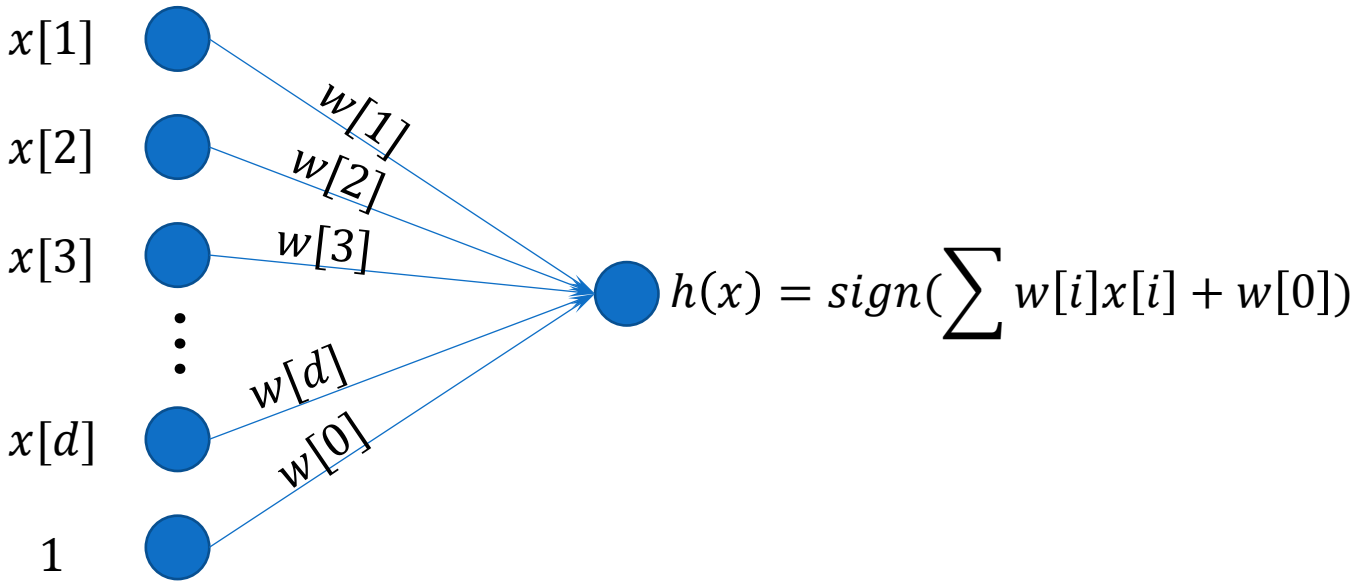
- In both Online Setting and Stochastic Approximation
  - Receive samples sequentially
  - Update **w** after each sample

- But, in Online Setting:
  - Objective is empirical regret, i.e. behavior on observed instances
  - z_t chosen adversarialy (no distribution involved)
- As opposed on Stochastic Approximation:
  - Objective is $\mathbb{E}[\ell(w, z)]$, i.e. behavior on "future" samples
  - i.i.d. samples z_t

- Stochastic Approximation is a computational approach, Online Learning is an analysis setup
  - E.g. "Follow the leader"

Lecture 17, part II:
Neural Networks

Linear Learning



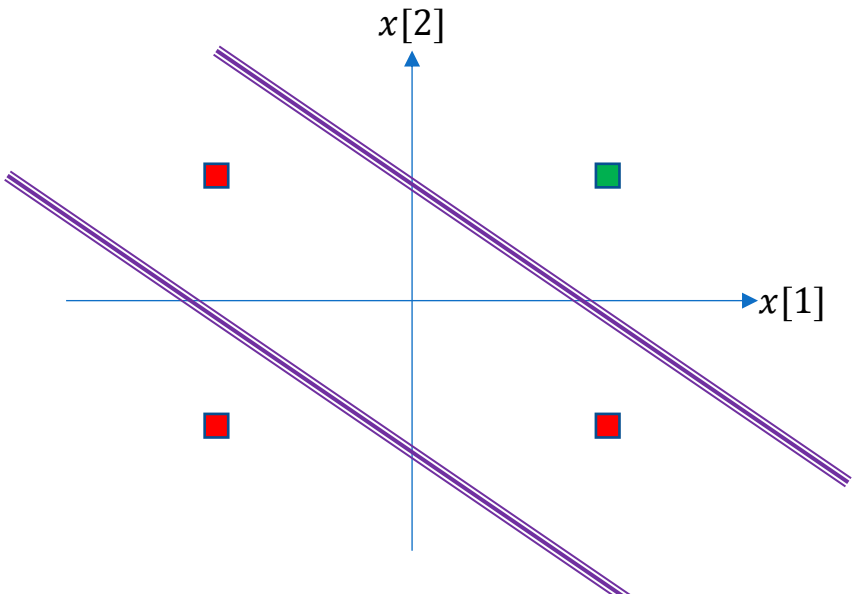
- Perceptron (gradient based) update:

$$w[i] \mathrel{+}= yx[i]$$

- Biological analogy: single neuron
  - Stimuli reinforce synaptic connections

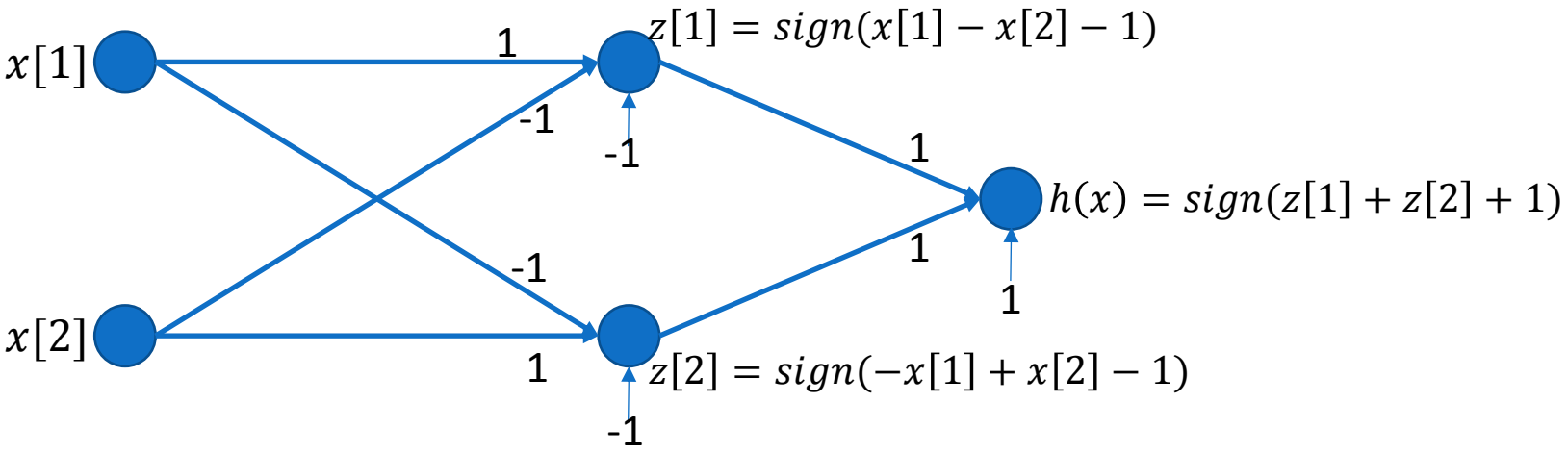
What can we represent with a single Linear Unit?

- AND (disjunctions):
  - $x[2] \wedge x[3] \wedge x[5] =$
  - $x[2] \wedge x[3] \wedge \overline{x[5]} = sign(x[1] + x[3] - x[5] - 2)$
  - $\overline{x[2] \wedge x[3] \wedge x[5]} = sign(-x[1] - x[3] - x[5] + 2)$
- OR (conjunctions):
  - $x[2] \vee x[3] \vee x[5] = sign(x[1] + x[3] + x[5] + 2)$
- XOR (parities):
  - $x[1] \oplus x[2] = ???$



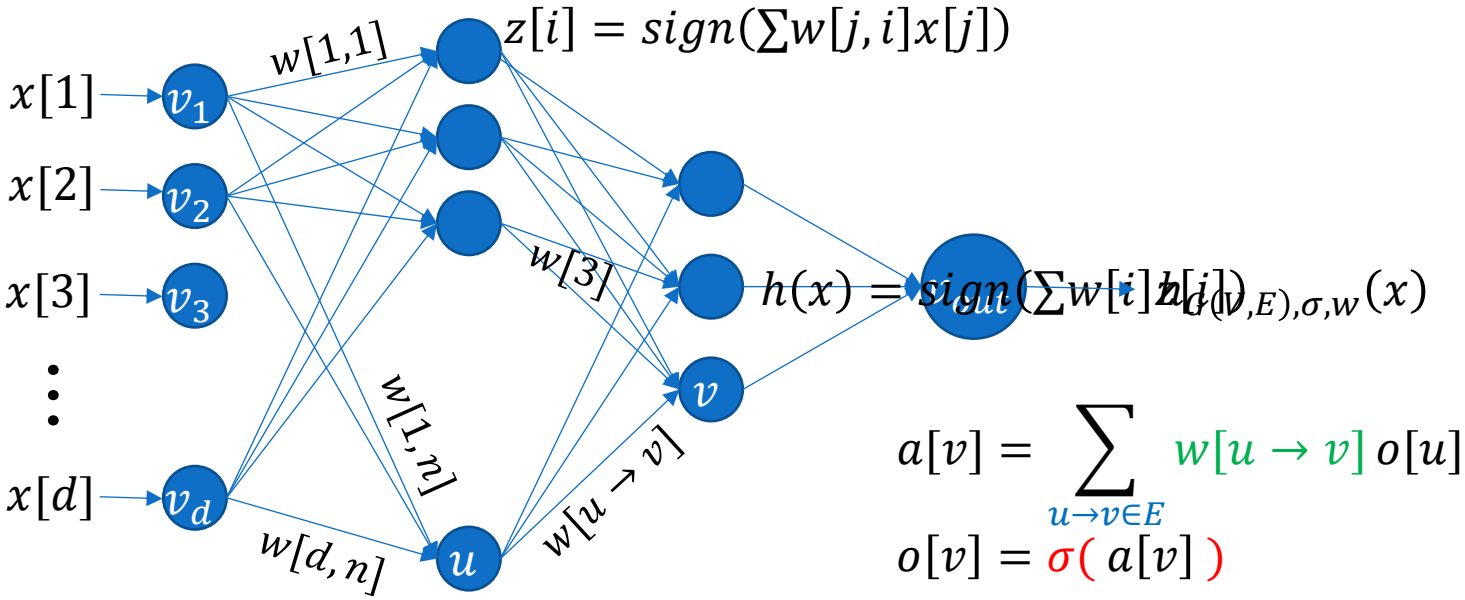
$y = x[1] \wedge x[2]$
$y = x[1] \vee x[2]$
$y = x[1] \oplus x[2]$

Combining Linear Units

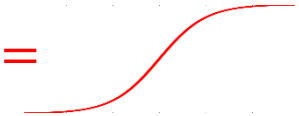


Claim: $h(x) = x[1] \oplus x[2]$

Feed-Forward Neural Networks (The Multilayer Perceptron)



Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \ldots v_d \in V$, with no incoming edges and $o[v_i] = x[i]$
  - "Output Unit" $v_{out} \in V$, $h_w(x) = o[v_{out}]$
- "Activation Function" $\sigma: \mathbb{R} \to \mathbb{R}$. E.g. $\sigma(z) = sign(z)$ or $\sigma(z) =$ 

Parameters:

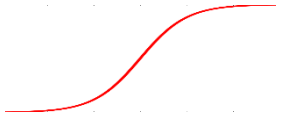
- Weight $w[u \to v]$ for each edge $u \to v \in E$

Feed-Forward Neural Networks as a Hypothesis Class

- Hypothesis class specified by:
  (ie we typically decide on this in advance)
  - Graph G(V,E)
    - V includes input, output and "hidden" nodes
  - Activation function σ

    e.g. $\text{sign}(z)$, $\tanh(z)$, $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$

Architecture



- Hypothesis specified by: (ie we need to learn)
  - Weights w, with weight $w[u \to v]$ for each edge $u \to v \in E$

$$\mathcal{H}_{G(V,E),\sigma} = \left\{ h_{G(V,E),\sigma,w} \mid w\colon E \to \mathbb{R} \right\}$$

- Issues:
  - Expressive power: What can we represent/approximate with $\mathcal{H}_{G(V,E),\sigma}$?

    ➔ **approximation error**
  - Statistical issues: Sample complexity of learning w

    ➔ **estimation error**
  - Computational issues: Can we learn efficiently and how?

    ➔ **optimization error**

Sample Complexity of NN

- #params = $|E|$ (number of weights we need to learn)
- More formally: $VCdim\left(\mathcal{H}_{G(V,E),sign}\right) = O(|E|\log|E|)$
- Other activation functions?
  - $VCdim\left(\mathcal{H}_{G(V,E),\sin}\right) = \infty$ even with single unit and single real-valued input
  - For $\sigma(z) = sigmoid(z) = \frac{1}{1+e^{-z}}$:

$$\Omega(|E|^2) \leq VCdim(\mathcal{H}_{G(V,E),\text{sigmoid}}) \leq O(|E|^4)$$

  - With finite precision:

$$VCdim\left(\mathcal{H}_{G(V,E),\sigma}\right) = O(|E|)$$

- Bottom line: **$|E|$ (number of weights) controls sample complexity**

What can Feed-Forward Networks Represent?

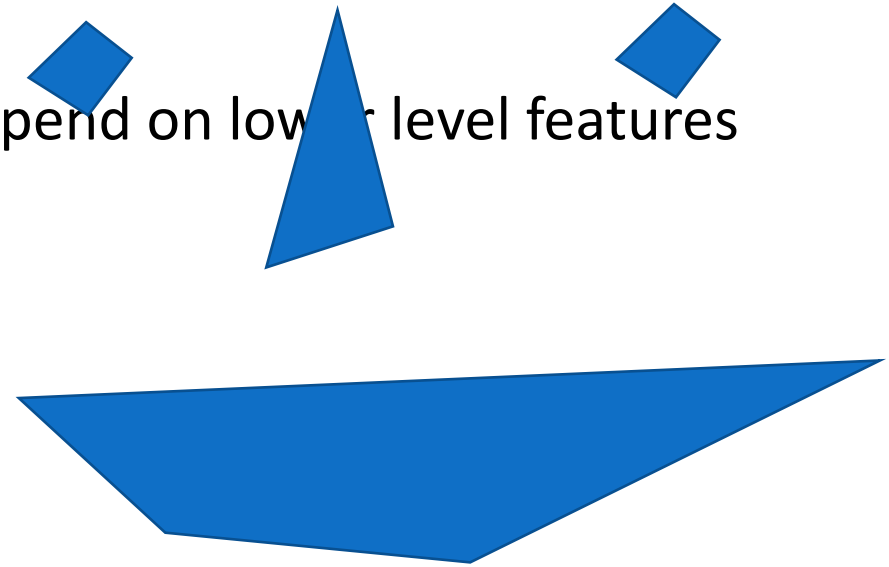
- Any function over $\mathcal{X} = \{\pm 1\}^d$
  - With a single hidden layer, using DNF (hidden layer does AND, output does OR)
  - $|V| = 2^d, |E| = d2^d$
  - Like representing the truth table directly…

- Universal Representation Theorem: Any continuous functions $f: [0,1]^d \to \mathbb{R}$ can be approximated to within any ϵ by a feed-forward network with sigmoidal (or almost any other) activation and a single hidden layer.
  - Size of layer exponential in d

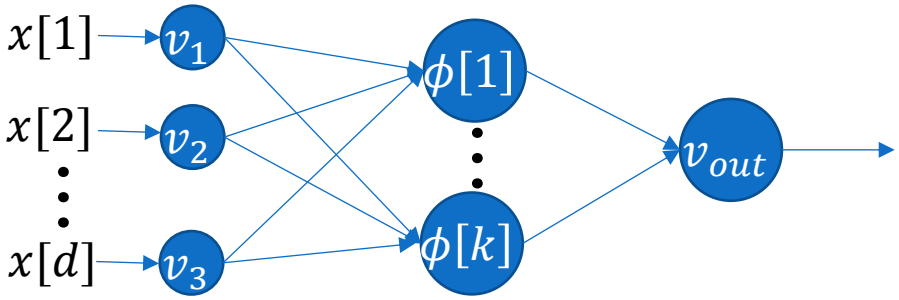
- Compare: With a large enough #params (large enough #features, small enough margin) even a *linear* model can approximate any continuous function arbitrary well (e.g. using Gaussian kernel)

What can SMALL Networks Represent?

- Intersection of halfspaces
  - Using single hidden layer
- Union of intersection of halfspaces (and also sorting, more fun stuff, …)
  - Using two hidden layers
- Functions that depend on lower level features



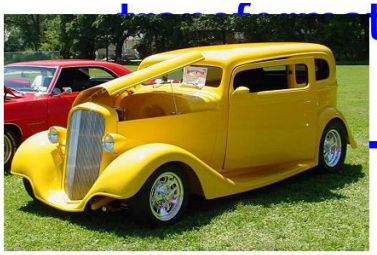
Neural Nets as Feature Learning



- Can think of hidden layer as "features" $\phi(x)$, then a linear predictor based on $\langle w, \phi \rangle$
- "Feature Engineering" approach: design $\phi(\cdot)$ based on domain knowledge
- "Deep Learning" approach: learn features from data

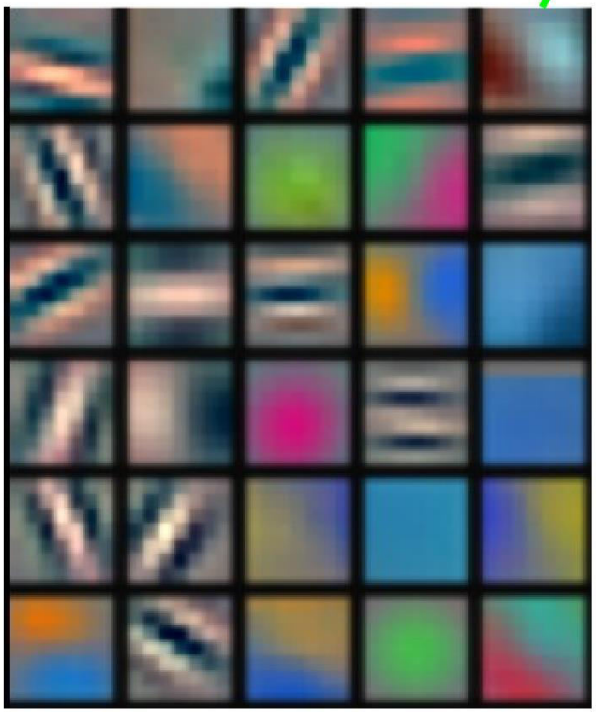
- Multilayer networks: more and more complex features

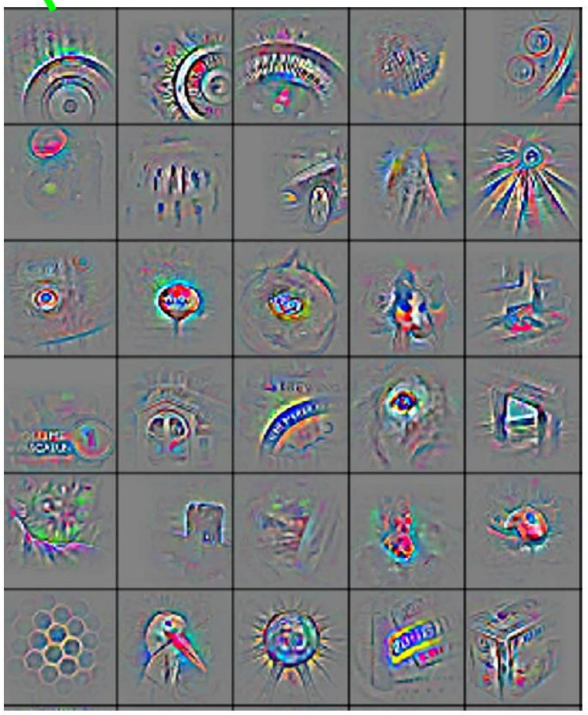
Multi-Layer Feature Learning



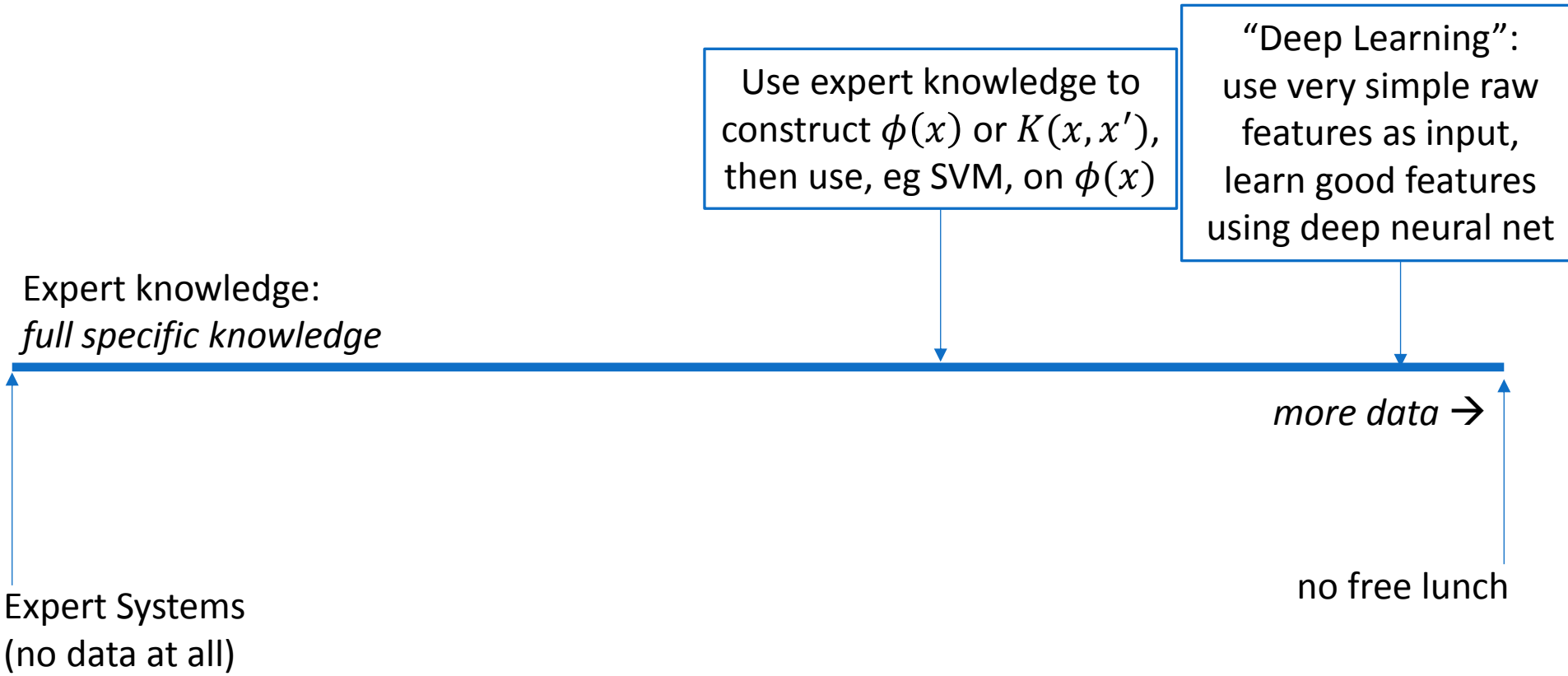
tion

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier





More knowledge or more learning



What can SMALL Networks Represent?

- Union of intersection of halfspaces (and also sorting, more fun stuff, …)
  - Using two hidden layers

- Functions that depend on lower level linear features

- Everything we want:

$$\{ f \mid f \text{ computable in time } T \} \subseteq \mathcal{H}_{G(V,E),\sigma}$$

with $|E| = O(T^2)$

  - Using a depth-T network, since anything computable in time T is also computable using a logical circuit of size $O(T^2)$

$\Rightarrow$ Universal Learning (learn anything computable in time T) with $poly(T)$ samples

- Compare: to get "universal approximation" with linear models / kernels, margin must shrink (and #features must grow) exponentially

Optimization

$$ERM(S) = \arg\min_{w} L_S(f_w)$$

- Highly non-convex problem, even if $loss$ and activation σ are convex
- NP-Hard even with single hidden layer and three hidden units

- Not surprising: otherwise, can learn hypothesis class of all poly-time functions
- Conclusion: Under crypto assumptions, no algorithm for properly PAC learning $\mathcal{H}_{G(V,E),\sigma}$ in time $poly(|E|)$

- In fact, we know it is hard to *improperly* learn intersection of half-spaces, and so (subject to crypto assmptions) no algorithm for **improperly** PAC learning $\mathcal{H}_{G(V,E),\sigma}$ in time $poly(|E|)$

Choose your universal learner:

Short Programs

- Universal
- Captures anything we want with reasonable sample complexity
- NP-hard to learn
- Hard to optimize in practice
  - No practical local search
  - Highly non-continuous, disconnected discrete space
  - Not much success

Deep Networks

- Universal
- Captures anything we want with reasonable sample complexity
- NP-hard to learn
- Often easy to optimize
  - Continuous
  - Amenable to local search, stochastic local search
  - **Lots of empirical success**

So how do we learn?

$$ERM(S) = \arg\min_{w} L_S\left(h_{G(V,E),\sigma,w}\right)$$

- Stochastic gradient descent:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta_t \nabla_w loss\left(h_{G(V,E),\sigma,w^{(t)}}(x), y\right)$$

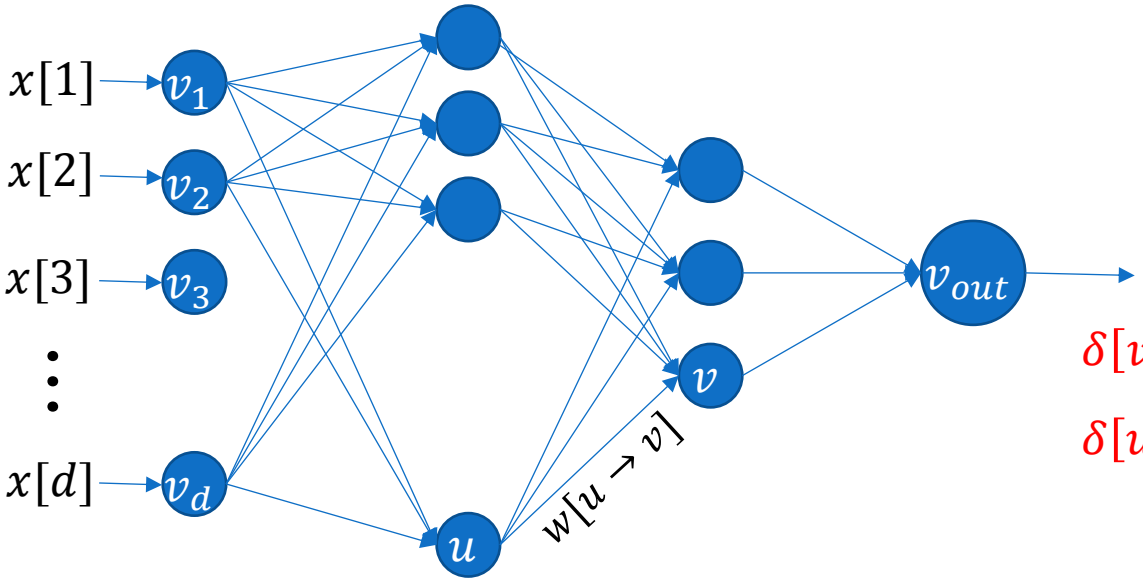
for random $(x, y) \in S$

(yes, even though its not convex)

- How do we efficiently calculate $\nabla_w loss(h_w(x), y)$?

Back-Propagation

- Efficient calculation of $\nabla_w loss(h_w(x), y)$ using chain rule



$$a[v] = \sum_{u \to v \in E} w[u \to v]\, o[u]$$

$$o[v] = \sigma(\, a[v]\,)$$

$$\delta[v_{out}] = loss'(o[v_{out}], y)$$

$$\delta[u] = \sum_{u \to v} w[u \to v]\delta[v]\sigma'(a[v])$$

- Forward propagation: calculate activations $a[v]$ and outputs $o[v]$
- Backward propagation: calculate $\delta_v \stackrel{\text{def}}{=} \frac{\partial loss(h_w(x), y)}{\partial o_v}$
- Output: $\frac{\partial loss}{\partial w[u \to v]} = \delta[v]\sigma'(a[v])o[u]$
- I.e. $w[u \to v] \mathrel{-}= \eta(\delta[v]\sigma'(a[v]))o[u]$

Theory of Neural Network Learning: Interim Summary

- Expressive Power
  - Universal, all poly-time functions

- Capacity Control (Sample Complexity)
  - $\propto$ number of weights
  - regularization

- Optimization

  ?????

Not: "what about reality is captured by my NN architecture"

Rather: "what about reality makes it easy to optimize my NN"

"its easy to optimize my NN ***on real data,***
because ***real data has such and such properties***"

History of Neural Networks

- 1940s-70s:
  - Inspired by learning in the brain, and as a model for the brain (Pitts, Hebb, and others)
  - Various models, directed and undirected, different activation and learning rules
  - Perceptron Rule (Rosenblatt), Problem of XOR, Multilayer perceptron (Minksy and Papert)
  - Backpropagation (Werbos 1975)
- 1980s-early 1990s:
  - Practical Back-prop (Rumelhart, Hinton et al 1986) and SGD (Bottou)
  - Relationship to distributed computing; “Connectionism”
  - Initial empirical success
- 1990s-2000s:
  - Lost favor to implicit linear methods: SVM, Boosting
- 2010s:
  - Computational advances allow training HUGE networks
  - …and also a few new tricks
  - Empirical success and renewed interest (Ng, LeCun, Hinton)

Neural Networks: Current Trends

- Very large architectures:
  #weights ≈ #samples ≈ $10^7 \sim 10^9$
  - SGD training on GPUs
  - Optimization technology: momentum, quasi-2nd order (beyond scope of course)
  - What stayed constant since the 50s: training runtime is about 10-14 days
- Use different activation functions:
  - Hinge-like activation ("rectified linear units")
  - Max (instead of summation) in some layers
- "Drop-out" regularization
  - Each SGD iteration, ignore random subset of edges (pretend they are not in the model))
  - Implicit regularization, not yet fully understood
- Convolutional Networks