Lecture 17:
Stochastic Optimization
Part II:
Neural Networks
Online vs Statistical Learning

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

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

• Statistical (PAC) “Regret” / excess error:

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

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. $\mathcal{H}$ is convex and $\ell(h, z)$ is convex in $h$):

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

$\bar{A}(S)$- Batch Conversion of online rule $A$

Input: training set $S = \{z_1, z_2, ..., z_m\}$
1. Run $A$ on $z_1, ..., z_m$ to obtain $h_1, h_2, ..., h_{m+1}$
2. Return $\bar{h}_m = \frac{1}{m} \sum_{t=1}^{m} h_t$

• Theorem:

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

• If $|\ell(h, z, )| \leq a$,

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

<table>
<thead>
<tr>
<th></th>
<th>Statistical</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite cardinality</td>
<td>$\sqrt{\frac{\log</td>
<td>\mathcal{H}</td>
</tr>
<tr>
<td>Finite VC-dimension</td>
<td>$\sqrt{\frac{\text{VCdim}}{m}}$</td>
<td>Not sufficient for diminishing regret</td>
</tr>
<tr>
<td>Linear in $\mathbb{R}^d$</td>
<td>$\sqrt{\frac{d}{m}}$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\ell_2$ margin $\gamma$</td>
<td>$\sqrt{\frac{1/\gamma^2}{m}}$</td>
<td>$\sqrt{\frac{1/\gamma^2}{m}}$</td>
</tr>
<tr>
<td>$|w|_2 \leq B$, $|\phi(x)|_2 \leq 1$, loss is 1-Lipschitz</td>
<td>$\sqrt{\frac{B^2 G^2}{m}}$</td>
<td>$\sqrt{\frac{B^2 G^2}{m}}$</td>
</tr>
<tr>
<td>$|w|<em>1 \leq B$, $|\phi(x)|</em>\infty \leq 1$, loss is 1-Lipschitz</td>
<td>$\sqrt{\frac{B^2 G^2 \log d}{m}}$</td>
<td>$\sqrt{\frac{B^2 G^2 \log d}{m}}$</td>
</tr>
</tbody>
</table>
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 \( z_1, z_2, ..., z_m \sim \mathcal{D} \)
- \( g = \nabla f(w, z_t) \) 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 it's easier to sample than integrate over \( z \)
Machine Learning is Stochastic Optimization

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

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

“General Learning” $\equiv$ Stochastic Optimization:
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

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 $t = 1, 2, 3, ...$
   1. Sample $z_t \sim \mathcal{D}$ (Obtain $g_t$ s.t. $\mathbb{E}[g_t] \in \partial F(w_t)$)
   2. $w_{t+1} = \prod_{w}(w_t - \eta_t \nabla f(w_t, z_t))$
3. Return $\overline{w}_m = \frac{1}{m} \sum_{t=1}^{m} w_t g_t$

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

\[ \min L_S(w) \quad s.t. \quad \|w\|_2 \leq B \]

Use \( g_t = \nabla_w \text{loss}^{\text{hinge}}(\langle w_t, \phi_i(x) \rangle; y_i) \) 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 \( \|w^{(t+1)}\|_2 > B \), then \( w^{(t+1)} \leftarrow B \frac{w^{(t+1)}}{\|w^{(t+1)}\|_2} \)

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

\[ \|\phi(x)\|_2 \leq G \Rightarrow \|g_t\|_2 \leq G \Rightarrow L_S(\bar{w}^{(T)}) \leq L_S(\hat{w}) + \sqrt{\frac{B^2G^2}{T}} \]

\( \text{(in expectation over randomness in algorithm)} \)
Stochastic vs Batch

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

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

<table>
<thead>
<tr>
<th></th>
<th>#iter</th>
<th>cost/iter</th>
<th>runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch GD</td>
<td>$B^2 G^2 / \epsilon_{opt}^2$</td>
<td>$md$</td>
<td>$md \frac{B^2 G^2}{\epsilon_{opt}^2}$</td>
</tr>
<tr>
<td>SGD</td>
<td>$B^2 G^2 / \epsilon_{opt}^2$</td>
<td>$d$</td>
<td>$d \frac{B^2 G^2}{\epsilon_{opt}^2}$</td>
</tr>
</tbody>
</table>

• What about $L(w)$, which is what we really care about?
• How small should $\epsilon_{opt}$ 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 $L_D(w)$

• Recall for ERM: $L_D(\hat{w}) \leq L_D(w^*) + 2\sup_{w} |L_D(w) + L_S(w)|$

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

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

• For $\epsilon_{opt}$ suboptimal ERM $\bar{w}$:

$L_D(\bar{w}) \leq L_D(w^*) + 2\sup_{w} |L_D(w) - L_S(w)| + (L_S(\bar{w}) - L_S(\hat{w}))$

$\epsilon_{est} \leq 2 \sqrt{\frac{B^2G^2}{m}}$

$\epsilon_{opt} \leq \sqrt{\frac{B^2G^2}{T}}$

• Take $\epsilon_{opt} \approx \epsilon_{est}$, i.e. $\#iter T \approx sample size m$

• To ensure $L_D(w) \leq L_D(w^*) + \epsilon$:

$T, m = O\left(\frac{B^2R^2}{\epsilon^2}\right)$
Direct Online-to-Batch: SGD on $L_D(w)$

$$\min_w L_D(w)$$

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

$$\Rightarrow \mathbb{E}[g_t] = \nabla L_D(w)$$

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

At iteration $t$:

- Draw $x_t, y_t \sim D$

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

$$L_D(\overline{w}^{(T)}) \leq \inf_{\|w\|_2 \leq B} L_D(w) + \sqrt{\frac{B^2 G^2}{T}}$$

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

\[
\min_{\|w\| \leq B} L_S(w)
\]

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: $w^{(t+1)} \leftarrow w^{(t)}$
- $w^{(t+1)} \leftarrow \text{proj } w^{(t+1)} \text{ to } \|w\| \leq B$

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 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 \(y_i (w^{(t)}, \phi(x_i)) < 1\),
  \(w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_i \phi(x_i)\)
  else: \(w^{(t+1)} \leftarrow w^{(t)}\)
\(w^{(t+1)} \leftarrow \text{proj } w^{(t+1)} \text{ to } \|w\| \leq B\)
Return \(\bar{w}^{(T)} = \frac{1}{T} \sum_{t=1}^{T} w^{(t)}\)

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

Direct SA (online2batch) Approach: SGD on ERM:
\[
\min_{w} L(w)
\]
Initialize \(w^{(0)} = 0\)
At iteration \(t\):
• Draw \(x_t, y_t \sim \mathcal{D}\)
• If \(y_t (w^{(t)}, \phi(x_t)) < 1\),
  \(w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_t \phi(x_t)\)
  else: \(w^{(t+1)} \leftarrow w^{(t)}\)
Return \(\bar{w}^{(T)} = \frac{1}{T} \sum_{t=1}^{T} w^{(t)}\)

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

\(w^* = \arg \min_{\|w\| \leq B} L(w)\)
SGD for Machine Learning

\[
\min_w L(w)
\]

SGD on RERM:

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

Direct SA (online2batch) Approach:

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

At iteration \( t \):

- Draw \( x_t, y_t \sim \mathcal{D} \)
- If \( y_t w(t), \phi(x_t) < 1 \),
  \[
  w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_t \phi(x_t)
  \]
  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

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 \( y_i w(t), \phi(x_i) < 1 \),
  \[
  w^{(t+1)} \leftarrow w^{(t)} + \eta_t y_i \phi(x_i)
  \]
  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

\[ \hat{w} = \arg \min_{\|w\| \leq B} L_S(w) \]

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

\[ \arg \min_w L_S(w) \text{ (overfit)} \]

\[ O \left( \sqrt{\frac{B^2 G^2}{m}} \right) \]
The mixed approach (reusing examples) can make sense.
Still: fresh samples are better.

With a larger training set, can reduce generalization error faster.

\textit{Larger} training set means \textit{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 adversarially (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:
  \[ w[i] \leftarrow w[i] + yx[i] \]

• Biological analogy: single neuron
  • Stimuli reinforce synaptic connections
What can we represent with a single Linear Unit?

• AND (disjunctions):

• OR (conjunctions):

• XOR (parities):
  • $x[1] \oplus x[2] = ???$
  • $y = x[1] \land x[2]$
  • $y = x[1] \lor x[2]$
  • $y = x[1] \oplus x[2]$
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” $\sigma: \mathbb{R} \rightarrow \mathbb{R}$. E.g. $\sigma(z) = \text{sign}(z)$ or $\sigma(z) = \sigma(z)$

Parameters:
• Weight $w[u \rightarrow v]$ for each edge $u \rightarrow v \in E$
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 $\sigma$
    e.g. $\text{sign}(z)$, $\text{tanh}(z)$, $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$

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

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

• Issues:
  • Expressive power: What can we represent/approximate with $\mathcal{H}_{G(V,E),\sigma}$?
    $\Rightarrow$ approximation error
  • Statistical issues: Sample complexity of learning $w$
    $\Rightarrow$ estimation error
  • Computational issues: Can we learn efficiently and how?
    $\Rightarrow$ optimization error
Sample Complexity of NN

• #params = |E| (number of weights we need to learn)

• More formally: \( VCDim(\mathcal{H}_{G(V,E),\text{sign}}) = O(|E| \log |E|) \)

• Other activation functions?
  • \( VCDim(\mathcal{H}_{G(V,E),\text{sin}}) = \infty \) even with single unit and single real-valued input
  • For \( \sigma(z) = \text{sigmoid}(z) = \frac{1}{1+e^{-z}} \):
    \[ \Omega(|E|^2) \leq VCDim(\mathcal{H}_{G(V,E),\text{sigmoid}}) \leq O(|E|^4) \]
  • With finite precision:
    \[ VCDim(\mathcal{H}_{G(V,E),\sigma}) = O(|E|) \]

• 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 $f: [0,1]^d \rightarrow \mathbb{R}$ can be approximated to within any $\varepsilon$ 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
More knowledge or more learning

Expert knowledge: full specific knowledge

Expert Systems (no data at all)

Use expert knowledge to construct $\phi(x)$ or $K(x, x')$, then use, eg SVM, on $\phi(x)$

“Deep Learning”: use very simple raw features as input, learn good features using deep neural net

more data →

no free lunch
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 \( O(T^2) \)

\[ \Rightarrow \text{Universal Learning (learn anything computable in time } T) \]
with \( \text{poly}(T) \) 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 \( \sigma \) 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 \( \text{poly}(|E|) \)
- In fact, we know it is hard to improperly learn intersection of half-spaces, and so (subject to crypto assumptions) no algorithm for improperly PAC learning \( \mathcal{H}_{G(V,E),\sigma} \) in time \( \text{poly}(|E|) \)
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?

\[ ERM(S) = \arg \min_w L_S(h_{G(V,E),\sigma,w}) \]

- Stochastic gradient descent:
  \[ w^{(t+1)} \leftarrow w^{(t)} - \eta_t \nabla_w \text{loss} \left( h_{G(V,E),\sigma,w^{(t)}}(x), y \right) \]
  for random \((x, y) \in S\)

  (yes, even though its not convex)

- How do we efficiently calculate \( \nabla_w \text{loss}(h_w(x), y) \)?
Back-Propagation

• Efficient calculation of $\nabla_w \text{loss}(h_w(x), y)$ using chain rule

Forward propagation: calculate activations $a[v]$ and outputs $o[v]$

Backward propagation: calculate $\delta_v \overset{\text{def}}{=} \frac{\partial \text{loss}(h_w(x), y)}{\partial o_v}$

Output: $\frac{\partial \text{loss}}{\partial w[u \rightarrow v]} = \delta[v] \sigma'(a[v]) o[u]$

I.e. $w[u \rightarrow v] = \eta(\delta[v] \sigma'(a[v])) o[u]$
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 (Minsky 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:
  \[ \#\text{weights} \approx \#\text{samples} \approx 10^7 \sim 10^9 \]
  • SGD training on GPUs
  • Optimization technology: momentum, quasi-\(2^{\text{nd}}\) 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