Linear Learning

- Perceptron (gradient based) update:
  \[ w_i \leftarrow yx_i \]

- Biological analogy: single neuron
  - Stimuli reinforce synaptic connections

**Graphical Representation**

\[ h_w(x) = \text{sign}\left( \sum w[i]x[i] + w[0] \right) \]

\[ x[0] = 1 \]
What can we represent with a single Linear Unit?

• AND (conjunctions):

• OR (disjunctions):

• XOR (parities):
  • $x[1] \oplus x[2] = ???$
What can we represent with a single Linear Unit?

• **AND** (conjunctions):

• **OR** (disjunctions):

• **XOR** (parities):
  - \( x[1] \oplus x[2] = ??? \)
What can we represent with a single Linear Unit?

- **AND (conjunctions):**

- **OR (disjunctions):**

- **XOR (parities):**
  - \(x[1] \oplus x[2] = ???\)

![Diagram](image)
What can we represent with a single Linear Unit?

• AND (conjunctions):

• OR (disjunctions):

• XOR (parities):
  • \( x[1] \oplus x[2] = ??? \)
Combining Linear Units


$h(x) = \text{sign}(z[1] + z[2] + 1)$

Claim: $h(x) = x[1] \oplus x[2]$
Feed-Forward Neural Networks (The Multilayer Perceptron)

\[ z[i] = \text{sign}(\sum w[j, i]x[j]) \]

\[ h(x) = \text{sign}(\sum w[i]z[i]) \]
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. $\sigma(z) = sign(z)$ or $\sigma(z) = \frac{1}{1 + e^{-z}}$

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

- 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: $\text{VCdim}(\mathcal{H}_{G(V,E),\text{sign}}) = O(|E| \log |E|)$
• Other activation functions?
  • $\text{VCdim}(\mathcal{H}_{G(V,E),\text{sin}}) = \infty$ even with single unit and single real-valued input
  • For $\sigma(z) = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$:
    $$\Omega(|E|^2) \leq \text{VCdim}(\mathcal{H}_{G(V,E),\text{sigmoid}}) \leq O(|E|^4)$$
  • For piecewise linear, e.g. $\text{ramp}(z) = \text{clip}_{[-1,1]}(z)$ or $\text{ReLU}(z) = \max(0, z)$:
    $$\Omega(|E|^2) \leq \text{VCdim}(\mathcal{H}_{G(V,E),\sigma}) \leq O(|E|^2)$$
• With finite precision:
  $$\text{VCdim}(\mathcal{H}_{G(V,E),\sigma}) \leq \log |\mathcal{H}_{G(V,E),\sigma}| \leq |E| \cdot \text{#bits}$$
• 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 $\epsilon$ 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
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 representable by a small logical circuit
  • Implement AND using single unit, negation by reversing weight

• 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

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

Expert Systems
(no data at all)

more data →

no free lunch
What can SMALL Networks Represent?

• Union of intersection of halfspaces (and also sorting, more fun stuff, ...)
• Functions that depend on lower level linear features
• Functions representable by small logical circuits

• Everything we want:
  computable in time $T$
  $\Rightarrow$ computable using depth $O(T)$ circuit with $O(T)$ gates per layer
  $\Rightarrow$ computable with depth $O(T)$ net with $O(T)$ nodes per layer and $O(1)$ fan-in

$$\{ f \mid f \text{ computable in time } T \} \subseteq \mathcal{H}_{G(V,E),\sigma} \quad \text{with } |E| = O(T^2)$$

Universal Learning: learn any $f$ 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
\[ ERM(S) = \arg \min_w L_S(f_w) \]

- NP-Hard even with single hidden layer, three hidden units, inputs in \( \{\pm 1\}^n \), and \( \sigma(z) = \text{sign}(z) \)

\[ V_n = \{\text{input units } v_1, \ldots, v_n\} \cup \{\text{hidden units } u_1, u_2, u_3\} \cup \{\text{output unit } v_{\text{out}}\} \]
\[ E_n = \{v_i \rightarrow u_j | i = 1..n, j = 1..3\} \cup \{u_j \rightarrow v_{\text{out}} | j = 1..3\} \]

\[ \arg \min_w L_S^{01}(f_{G_n(V_n,E_n),\text{sign},w}) \text{ NP-Hard} \]

\( \Rightarrow \) If \( NP \neq RP \), not efficiently agnostically properly PAC learnable

(not surprising: not efficiently agnostically PAC learnable even with no hidden units)

- NP-Hard, even if realizable, i.e.:

NP-Hard to decide whether \( \exists_w L_S^{01}(f_{G_n(V_n,E_n),\text{sign},w}) = 0 \)

\( \Rightarrow \) If \( NP \neq RP \), not efficiently properly PAC learnable

\( \Rightarrow \arg \min_w L_S^{\text{hinge}}(f_{G_n(V_n,E_n),\text{sign},w}) \text{ NP-Hard} \)

\( \Rightarrow \) If \( NP \neq RP \), not eff. agnostically properly PAC learnable w.r.t. hinge loss
Complexity of Learning Neural Nets

- Highly non-convex problem
- Hard to properly PAC learn, even with $d=1$ hidden layer and $k=3$ hidden units
- Not surprising: if we could learn poly-size neural networks, we could learn hypothesis class of all poly-time functions

$$G_{n,d,k} = \text{graph with } n \text{ inputs, } d \text{ hidden layers, } k \text{ units in each layer}$$

$$\text{TIME}(T(n)) \subseteq \mathcal{H}_{G_{n,d=O(T(n)),k=O(T(n))},\sigma}$$

- Conclusion: If crypto possible, no algorithm for PAC learning $\mathcal{H}_{G(V,E),\sigma}$ in time $\text{poly}(|E|)$
- Discrete cube root hard $\Rightarrow$ no efficient PAC learning of $\mathcal{H}_{G_{n,d=O(\log(n)),k=O(n)}}$
- RSAT $\Rightarrow$ no efficient PAC learning of $\mathcal{H}_{G_{n,d=1,k=\omega(1)}}$
- Still open: efficiently improperly learn network with constant number of hidden units.
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?

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

- 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. $w[u \rightarrow v] :-= \eta(\delta[v] \sigma'(a[v])) o[u]$

\[
\begin{align*}
a[v] &= \sum_{u \rightarrow v \in E} w[u \rightarrow v] o[u] \\
o[v] &= \sigma(a[v]) \\
\delta[v_{\text{out}}] &= \text{loss}'(o[v_{\text{out}}], y) \\
\delta[u] &= \sum_{u \rightarrow v} w[u \rightarrow v] \delta[v] \sigma'(a[v])
\end{align*}
\]
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: 
  \#weights \approx \#samples \approx 10^7 \sim 10^9
  • SGD training on GPUs
  • Optimization technology: momentum, quasi-2\textsuperscript{nd} order
  • What stayed constant since the 50s: training runtime is about 10-14 days

• Use different activation functions:
  • Hinge-like activation: \textit{ReLU}(z) = \max(z, 0)
  • 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 and Recurrent Networks
  • Many edges share the same weight, \#param << \#edges
Theory of Neural Network Learning

• 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”
Computational and Statistical Learning Theory

• Main Goals:
  • **Strengthen understanding of learning.** What effects how well we can learn, and the required resources (data, computation) for learning? How should we think of and evaluate different learning approaches?
  • **Obtain formal and quantitative understanding of learning:** Formalize learning, “no free lunch”, universal learning, computational limits on learning; Understand learning guarantees and bounds and what they tell us.
  • **Understand relationship between learning and optimization**, and explore modern optimization techniques in the context of learning.

• Secondary Goal:
  • Learn techniques and develop skills for analyzing learning and proving learning guarantees.
“Machine Learning”: Use data and examples, instead of expert knowledge, to automatically create systems that perform complex tasks.

- Does smoking contribute to lung cancer?
  • Yes, with p-value = $10^{-72}$

- How long ago did cats and dogs diverge?
  • About 55 MY, with 95% confidence interval [51,60]

- 99% of faces have two eyes
- People with beards buy less nail polish
- ...

\[(\text{Rotation time})^2 \propto (\text{avg radius})^3\]

Vapnik
Generic Learning

Learning Algorithm

- Sample emails
- Examples of bicycles
- Examples of faces
- Spanish and English texts
- Recorded transliterated audio
- Protein sequences and folds

- Spam detector
- Bike detectors
- Face recognizer
- Translation system
- Speech recognizer
- Protein fold predictor
The ability to learn grammars is hard-wired into the brain. It is not possible to “learn” linguistic ability—rather, we are born with a brain apparatus specific to language representation.

There exists some “universal” learning algorithm that can learn anything: language, vision, speech, etc. The brain is based on it, and we’re working on uncovering it. (Hint: the brain uses neural networks)

There is no “free lunch”: no learning is possible without some prior assumption about the structure of the problem (prior knowledge)
Inductive Bias

• “No Free Lunch”: no rule can learn anything, need some inductive bias
  • In terms of learning rule: Some bias toward some hypothesis over others
  • In terms of guarantee: Some assumption

• Encode as--
  • Hypothesis class $\mathcal{H}$: “reality is reasonably well captured by some hypothesis from $\mathcal{H}$”
  • Measure $p(h)$ (or continuous measure $P$, or measure over hypothesis classes):
    • “reality is reasonably well captured by $h$ with high $p(h)$”
    • Complexity measure ($\equiv$ length of description in chosen description language) $- \log p(h)$

  **Master learning rule:** $\arg\min L_S(h), -\log p(h)$

• Our analysis focused on learning a “flat” hypothesis class $\mathcal{H}$
  $\rightarrow$ Usually actually part of some hierarchy $\mathcal{H}(B)$, which we can think of as level sets $\mathcal{H} = \{ h \mid -\log p(h) \leq B \}$
Uniform and Non-Uniform Learnability

• **Definition:** A hypothesis class $\mathcal{H}$ is **agnostically PAC-Learnable** if there exists a learning rule $A$ such that $\forall \epsilon, \delta > 0, \exists m(\epsilon, \delta), \forall \mathcal{D}, \forall h, \forall S \sim \mathcal{D} m(\epsilon, \delta),$ 
  $$L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon$$

• **Definition:** A hypothesis class $\mathcal{H}$ is **non-uniformly learnable** if there exists a learning rule $A$ such that $\forall \epsilon, \delta > 0, \forall h, \exists m(\epsilon, \delta, h), \forall \mathcal{D}, \forall S \sim \mathcal{D} m(\epsilon, \delta, h),$ 
  $$L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon$$

• **Definition:** A hypothesis class $\mathcal{H}$ is **“consistently learnable”** if there exists a learning rule $A$ such that $\forall \epsilon, \delta > 0, \forall h \forall \mathcal{D}, \exists m(\epsilon, \delta, h, \mathcal{D}), \forall S \sim \mathcal{D} m(\epsilon, \delta, h, \mathcal{D}),$ 
  $$L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon$$

- Scale, domain and emphasis of ML:
  - Machine Learning as an engineering paradigm: performing a task vs obtaining understanding
  - Emphasis on computational issues: without computational constraints, ML is trivial
  - No need for sophisticated confidence measures: use validation
  - Big Data?
Learning Theory vs “Classical” Statistical Analysis: An Overly Simplistic View

• Typical statistical analysis:
  • Assume model $P((x, y)|\theta)$
  • Estimate $\hat{\theta}$ from data, study how $\hat{\theta} \to \theta$
  • Analysis often focused on asymptotic regime as $m \to \infty$ and $\epsilon = |\hat{\theta} - \theta| \to 0$

• Typical statistical learning theory analysis (this course):
  • Choose hypothesis class to compete with, don’t assume it
  • Learn predictor from data, study how it competes with $\mathcal{H}$
    • Since we don’t assume model, no true $\theta$ to approach
    • Might even be improper: estimate doesn’t even correspond to any $\theta$
    • Also, might not be identifiable
  • Finite sample analysis: Since model wrong anyway, relevant regime often $\epsilon \approx$ model error $> 0$
Example: Linear Regression

- Classical statistical view of linear regression:
  \[ y_i = \langle w^*, x_i \rangle + v_i \]

- Goal:
  - Parameter error: \( \| \hat{w} - w^* \|_2 \)
  - “De-noising”:
    \[
    \frac{1}{m} \sum_{i=1}^{m} (\langle \hat{w}, x_i \rangle - \langle w^*, x_i \rangle)^2 = (\hat{w} - w^*)^T \hat{\Sigma}_X (\hat{w} - w^*)
    \]
  - Prediction:
    \[
    \mathbb{E}[(\langle \hat{w}, x \rangle - y)^2] = \mathbb{E}[(\langle \hat{w}, x \rangle - \langle w^*, x \rangle)^2] + \sigma^2
    \]
    \[
    = (\hat{w} - w^*)^T \Sigma_X (\hat{w} - w^*) + \sigma^2 \to \frac{d \sigma^2}{m} + \sigma^2
    \]

- ML / This course:
  - Only assume there exists some \( w^0 \) s.t. \( \mathbb{E}[(\langle w^0, x \rangle - y)^2] \leq \sigma^2 \)
  - Equivalently: \( y_i = \langle w^0, x_i \rangle + v_i \) where \( \mathbb{E}[v_i^2] \leq \sigma^2 \), but not \( x \perp v \)

\[
\mathbb{E}[(\langle \hat{w}, x \rangle - y)^2] \leq \sigma^2 + O \left( \frac{d}{m} + \sqrt{\frac{\sigma^2 d}{m}} \right)
\]

\( |v|, |y| \) bounded with high probability

\( v_i \sim \mathcal{N}(0, \sigma^2) \) independent of \( x_i \)

Can avoid this term
A Different Approach

• Could take more probabilistic approach to machine learning, focusing on probabilistic models for $p(y|x; \theta)$ instead of hypothesis classes

• Likelihood $- \log p(y_1 \ldots y_m | x_1 \ldots x_m; \theta) = \sum_i (- \log p(y_i | h_\theta(x_i)))$

• Bayesian: also assuming probabilistic prior $p(\theta)$

• Posterior: $- \log p(\theta|S) \propto \sum_i (- \log p(y_i | h_\theta(x_i))) + (- \log p(\theta))$

• Many advantages: easier to discuss uncertainty in predictions, better interpretation of regularization parameter, often easier to combine models

• Analyze under probabilistic model: “Well specified”/Bayesian analysis typically tighter than our agnostic analysis, but assumptions much stronger

• Topic of a different class (that can also be called “Statistical Learning Theory”....)
Computational and Statistical Learning Theory

• Main Goals:
  • **Strengthen understanding of learning.** What effects how well we can learn, and the required resources (data, computation) for learning? How should we think of and evaluate different learning approaches?
  • **Obtain formal and quantitative understanding of learning:** Formalize learning, “no free lunch”, universal learning, computational limits on learning; Understand learning guarantees and bounds and what they tell us.
  • **Understand relationship between learning and optimization,** and explore modern optimization techniques in the context of learning.

• Secondary Goal:
  • Learn techniques and develop skills for analyzing learning and proving learning guarantees.