Day 6: Neural networks, backpropagation

Introduction to Machine Learning Summer School
June 18, 2018 - June 29, 2018, Chicago

Instructor: Suriya Gunasekar, TTI Chicago

25 June 2018
Schedule

• 9:00am-10:25am – Lecture 6.a: Review of week 1, introduction to neural networks
• 10:30am-11:30am – Invited Talk - Greg Durett (also the TTIC colloquium talk)
• 11:30am-12:30pm – Lunch
• 12:30pm-2:00pm – Lecture 6.b: Backpropagation
• 2:00pm-5:00pm – Programming
Review of week 1
Supervised learning – key questions

- **Data**: what kind of data can we get? how much data can we get?
- **Model**: what is the correct model for my data? – want to minimize the effort put into this question!
- **Training**: what resources - computation/memory - does the algorithm need to estimate the model \( \hat{f} \)?
- **Testing**: how well will \( \hat{f} \) perform when deployed? what is the computational/memory requirement during deployment?
Linear regression

• Input $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$, output $y \in \mathbb{R}$, want to learn $f: \mathcal{X} \rightarrow \mathbb{R}$

• Training data $S = \{(\mathbf{x}^{(i)}, y^{(i)}): i = 1,2,\ldots,N\}$

• Parameterize candidate $f: \mathcal{X} \rightarrow \mathbb{R}$ by linear functions,
  $$\mathcal{H} = \{\mathbf{x} \rightarrow \mathbf{w}.\mathbf{x}: \mathbf{w} \in \mathbb{R}^d\}$$

• Estimate $\mathbf{w}$ by minimizing loss on training data

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J^S_{LS}(\mathbf{w}): = \sum_{i=1}^{N} (\mathbf{w}.\mathbf{x}^{(i)} - y^{(i)})^2$$

  o $J^S_{LS}(\mathbf{w})$ is convex in $\mathbf{w} \rightarrow$ minimize $J^S_{LS}(\mathbf{w})$ by setting gradient to 0
  
  o $\nabla_{\mathbf{w}} J^S_{LS}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}.\mathbf{x}^{(i)} - y^{(i)})\mathbf{x}^{(i)}$
  
  o Closed form solution $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\mathbf{y}$

• Can get non-linear functions by mapping $\mathbf{x} \rightarrow \phi(\mathbf{x})$ and doing linear regression on $\phi(\mathbf{x})$
Overfitting

• For same amount of data, more complex models (e.g., higher degree polynomials) overfit more
• or need more data to fit more complex models
• complexity $\approx$ number of parameters

Model selection

• $m$ model classes $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m\}$
• $S \equiv S_{\text{train}} \cup S_{\text{val}} \cup S_{\text{test}}$
• Train on $S_{\text{train}}$ to pick best $\hat{f}_r \in \mathcal{H}_r$
• Pick $\hat{f}^*$ based on validation loss on $S_{\text{val}}$
• Evaluate test loss $L_{\text{test}}(\hat{f}^*)$
Regularization

• Complexity of model class can also be controlled by norm of parameters – smaller range of values allowed

• Regularization for linear regression

  $$\arg\min_w J_{LS}^S(w) + \lambda \|w\|_2^2$$

  $$\arg\min_w J_{LS}^S(w) + \lambda \|w\|_1$$

• Again do model selection to pick $\lambda$ – using $S_{val}$ or cross-validation
Classification

• Output $y \in \mathcal{Y}$ takes discrete set of values, e.g., $\mathcal{Y} = \{0,1\}$ or $\mathcal{Y} = \{-1,1\}$ or $\mathcal{Y} = \{\text{spam, nospam}\}$
  o Unlike regression, label-values do not have meaning

• Classifiers divide the space of input $\mathcal{X}$ (often $\mathbb{R}^d$) to “regions” where each region is assigned a label

• Non-parametric models
  o k-nearest neighbors – regions defined based on nearest neighbors
  o decision trees – structured rectangular regions

• Linear models – classifier regions are halfspaces
Classification — logistic regression

Logistic loss
\[ \ell(f(x), y) = \log(1 + \exp(-f(x)y)) \]

- \( \mathcal{X} = \mathbb{R}^d, \mathcal{Y} = \{-1, 1\}, S = \{(x^{(i)}, y^{(i)}): i = 1, 2, \ldots, N\} \)
- Linear model \( f(x) = f_w(x) = w \cdot x \)
- Output classifier \( \hat{y}(x) = \text{sign}(w \cdot x) \)
- Empirical risk minimization
\[
\hat{w} = \arg\min_w \sum_i \log \left( 1 + \exp(-w \cdot x^{(i)}y^{(i)}) \right)
\]
- Probabilistic formulation: \( \Pr(y = 1|x) = \frac{1}{1 + \exp(-w \cdot x)} \)
- Multi-class generalization: \( \mathcal{Y} = \{1, 2, \ldots, m\} \)
\[
\Pr(y|x) = \frac{\exp(-w_y \cdot x)}{\sum_{y'} \exp(-w_{y'} \cdot x)}
\]
- Can again get non-linear decision boundaries by mapping \( x \rightarrow \phi(x) \)
Classification – maximum margin classifier

Separable data
• Original formulation
\[ \hat{w} = \arg\max_{w \in \mathbb{R}^d} \min_i \frac{y^{(i)} w \cdot x^{(i)}}{||w||} \]
• Fixing \( ||w|| = 1 \)
\[ \hat{w} = \arg\max_w \min_i y^{(i)} (w \cdot x^{(i)}) \quad \text{s.t.} \quad ||w|| = 1 \]
• Fixing \( \min_i y^{(i)} w \cdot x^{(i)} = 1 \)
\[ \hat{w} = \arg\min_w ||w||^2 \quad \text{s.t.} \quad \forall i, y^{(i)} (w \cdot x^{(i)}) \geq 1 \]

Slack variables for non-separable data
\[ \hat{w} = \arg\min_w ||w||^2 + \lambda \sum_i \xi_i \quad \text{s.t.} \quad \forall i, y^{(i)} (w \cdot x^{(i)}) \geq 1 - \xi_i \]
\[ = \arg\min_w ||w||^2 + \lambda \sum_i \max(0,1 - y^{(i)} (w \cdot x^{(i)})) \]
Kernel trick

- Using representor theorem \( \mathbf{w} = \sum_{i=1}^{N} \beta_i \mathbf{x}^{(i)} \)

\[
\min_{\mathbf{w}} \|\mathbf{w}\|^2 + \lambda \sum_i \max(0, 1 - y^{(i)} \mathbf{w} \cdot \mathbf{x}^{(i)})
\]

\[\equiv \min_{\beta \in \mathbb{R}^N} \beta^T G \beta + \lambda \sum_i \max(0, 1 - y^{(i)}(G \beta)_i)\]

\( G \in \mathbb{R}^{N \times N} \) with \( G_{ij} = \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \) is called the gram matrix

- Optimization depends on \( \mathbf{x}^{(i)} \) only through \( G_{ij} = \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \)

- For prediction \( \hat{\mathbf{w}} \cdot \mathbf{x} = \sum_i \beta_i \mathbf{x}^{(i)} \cdot \mathbf{x} \), we again only need \( \mathbf{x}^{(i)} \cdot \mathbf{x} \)

- Function \( K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}' \) is called the Kernel

- When learning non-linear classifiers using feature transformations \( \mathbf{x} \rightarrow \phi(\mathbf{x}) \) and \( f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) \)
  - Classifier fully specified in terms of \( K_{\phi}(\mathbf{x}, \mathbf{x}') = K(\phi(\mathbf{x}), \phi(\mathbf{x}')) \)
  - \( \phi(\mathbf{x}) \) itself can be very very high dimensional (maybe even infinite dimensional)
Optimization

• ERM+regularization optimization problem

$$\hat{w} = \text{argmin}_{w} J^\lambda_S(w): = \sum_{i=1}^{N} \ell(w \cdot \phi(x^{(i)}), y^{(i)}) + \lambda \|w\|$$

• If $J^\lambda_S(w)$ is convex in $w$, then $\hat{w}$ is optimum if and only if gradient at $\hat{w}$ is 0, i.e., $\nabla J^\lambda_S(\hat{w}) = 0$

• Gradient descent: start with initialization $w^0$ and iteratively update

  o $w^{t+1} = w^t - \eta^t \nabla J^\lambda_S(w^t)$
  o where $\nabla J^\lambda_S(w^t) = \sum_i \nabla \ell(w^t \cdot \phi(x^{(i)}), y^{(i)}) + \lambda \nabla \|w^t\|$

• Stochastic gradient descent

  o use gradients from only one example

  o $w^{t+1} = w^t - \eta^t \nabla^{(i)} J^\lambda_S(w^t)$
  o where $\nabla^{(i)} J^\lambda_S(w^t) = \nabla \ell(w^t \cdot \phi(x^{(i)}), y^{(i)}) + \lambda \nabla \|w^t\|$ for a random sample $(x^{(i)}, y^{(i)})$
Other classification models

• Optimal unrestricted predictor
  o Regression + squared loss $\Rightarrow f^{**}(x) = E[y|x]$
  o Classification + 0-1 loss $\Rightarrow \hat{y}^{**}(x) = \arg\max_c \Pr(y = c|x)$

• Discriminative models: directly model $\Pr(y|x)$, e.g., logistic regression

• Generative models: model full joint distribution $\Pr(y, x) = \Pr(x|y) \Pr(y)$

• Why generative models?
  o One conditional might be simpler to model with prior knowledge, e.g., compare specifying $\Pr(\text{image}|\text{digit} = 1)$ vs $\Pr(\text{digit} = 1|\text{image})$
  o Naturally handles missing data

• Two examples of generative models
  o Naïve Bayes classifier
  o Hidden Markov model
Other classifiers

• Naïve Bayes classifier: with d features \( x = [x_1, x_2, ..., x_d] \) where each \( x_1, x_2, ..., x_d \) can take one of K values \( \rightarrow C K^d \) parameters
  
  o NB assumption: features are independent given class \( y \) \( \rightarrow C K d \) params.
  
  \[
  \Pr(x_1, x_2, ..., x_d | y) = \Pr(x_1 | y) \Pr(x_2 | y) ... \Pr(x_d | y) = \prod_{k=1}^{d} \Pr(x_k | y)
  \]

  o Training amounts to averaging samples across classes

• Hidden Markov model: variable length input/observations \( \{x_1, x_2, ..., x_m\} \) (e.g., words) and variable length output/state \( \{y_1, y_2, ..., y_m\} \) (e.g., tags)
  
  o HMM assumption: a) current state conditioned on immediate previous state is conditionally independent of all other variables, and (b) current observation conditioned on current state is conditionally independent of all other variables.
  
  \[
  \Pr(x_1, x_2, ..., x_m, y_1, y_2, ..., y_m) = \Pr(y_1) \Pr(x_1 | y_1) \prod_{k=2}^{m} \Pr(y_k | y_{k-1}) \Pr(y_k | x_k)
  \]

  o Parameters estimated using MLE dynamic programming
Today

Introduction to neural networks

Backpropagation
Graph notation

General variables
- can be input variables like $x_1, x_2, \ldots x_d$
- prediction $\hat{y}$
- or any intermediate computation (we will see examples soon)

\[ z_3 = \sigma(w_1z_1 + w_2z_2) \]

for some “activation” function $\sigma$ (specified apriori)
Linear classifier

- Biological analogy: single neuron – stimuli reinforce synaptic connections

\[ f(x) = 1(w \cdot x + w_0 \geq 0) \]

McCulloch and Pitts 1943 — introduced the linear threshold “neuron”.

Slide credits: Nati Srebro, David McAllester
Shallow learning

• We already saw how to use linear models to get non-linear decision boundaries

• Feature transform: map \( x \in \mathbb{R}^d \) to \( \phi(x) \in \mathbb{R}^{d'} \) and use

\[
f_w(x) = w \cdot \phi(x)
\]

• Shallow learning: hand-crafted and non-hierarchical \( \phi \)
  
  o Polynomial regression with squared or logistic loss, \( \phi(x)_p = x^p \)
  
  o Kernel SVM: \( K(x, x') = \phi(x). \phi(x') \)

Slide credit: Nati Srebro
Combining Linear Units

\[ z_L = \mathbf{1}(x_L - x_I > 0) \]

\[ f(x) = z_L + z_I > 0 \]

\[ z_I = \mathbf{1}(x_I - x_L > 0) \]

- The network represents the function
  \[ f(x) = (x_1 \text{ and not}(x_2)) \text{ or } (x_2 \text{ and not}(x_1)) \]
- Not a linear function of \( x \)

Slide credit: Nati Srebro
Combining Linear Units

\[ z_L = 1(w_1 \cdot x > 0) \]

\[ z_I = 1(w_2 \cdot x > 0) \]

\[ f(x) = 1(\hat{w}_1 z_1 + \hat{w}_2 z_2 \geq 0) \]
Feed-Forward Neural Networks

\[ z[i] = \sigma(\sum_j W^{(1)}[j, i] x_j) \]

\[ f(x) = \sigma(\sum_j W^{(2)}[j] z[j]) \]

Figure credit: Nati Srebro
Feed-Forward Neural Networks

\[ z[1] = \sigma \left( \sum_j W^{(1)}_{j, 1} x_j \right) \]

\[ f(x) = \sigma \left( \sum_j W^{(2)}_{j} z[j] \right) \]
Feed-Forward Neural Networks

\[ z[2] = \sigma\left(\sum_j W^{(1)}[j, 2] x_j\right) \]

\[ f(x) = \sigma\left(\sum_j W^{(2)}[j] z[j]\right) \]
Feed-Forward Neural Networks

\[ f(x) = \sigma(\sum_j W^{(2)}[j]z[j]) \]

\[ z[3] = \sigma(\sum_j W^{(1)}[j, 3]x_j) \]
Feed-Forward Neural Networks

\[ f(x) = \sigma(\sum_j W^{(2)}[j]z[j]) \]

\[ z[d_1] = \sigma(\sum_j W^{(1)}[j, d_1]x_j) \]
Feed-Forward Neural Networks

\[ z[i] = \sigma \left( \sum_j W^{(1)}[j, i] x_j \right) \]

\[ f(x) = \sigma \left( \sum_j W^{(2)}[j] z[j] \right) \]
Feed-Forward Neural Networks

Architecture:
- Directed Acyclic Graph $G(V, E)$. Units (neurons) indexed by vertices in $V$. 

Slide credit: Nati Srebro
Feed-Forward Neural Networks

Architecture:

- Directed Acyclic Graph $G(V,E)$. Units (neurons) indexed by vertices in $V$.
  - “Input Units” $v_1 \ldots v_d \in V$ : no incoming edges have value $o[v_i] = x_i$
Feed-Forward Neural Networks

Architecture:
- Directed Acyclic Graph $G(V,E)$. Units (neurons) indexed by vertices in $V$.
  - “Input Units” $v_1 \ldots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \rightarrow v$ has weight $W[u \rightarrow v]$
    - Pre-activation $a[v] = \sum_{u \rightarrow v \in E} W[u \rightarrow v] o[u]$
Feed-Forward Neural Networks

**Architecture:**
- Directed Acyclic Graph $G(V,E)$. Units (neurons) indexed by vertices in $V$.
  - “Input Units” $v_1 \ldots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \rightarrow v$ has weight $W[u \rightarrow v]$
    - Pre-activation $a[v] = \sum_{u \rightarrow v \in E} W[u \rightarrow v] o[u]$
    - Output value $o[v] = \sigma(a[v])$
Feed-Forward Neural Networks

Architecture:

- Directed Acyclic Graph $G(V,E)$. Units (neurons) indexed by vertices in $V$.
  - “Input Units” $v_1 \ldots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \rightarrow v$ has weight $W[u \rightarrow v]$
    - Pre-activation $a[v] = \sum_{u \rightarrow v \in E} W[u \rightarrow v] o[u]$
    - Output value $o[v] = \sigma(a[v])$
  - “Output Unit” $v_{out} \in V, f_W(x) = a[v_{out}]$
Feed-Forward Neural Networks

Architecture:

- Directed Acyclic Graph $G(V,E)$. Units (neurons) are indexed by vertices in $V$.
  - “Input Units” $v_1 \ldots v_d \in V$: no incoming edges, have value $o[v_i] = x_i$
  - Each edge $u \rightarrow v$ has weight $W[u \rightarrow v]$
- Pre-activation $a[v] = \sum_{u \rightarrow v \in E} W[u \rightarrow v] o[u]$
- Output value $o[v] = \sigma(a[v])$
- “Output Unit” $v_{out} \in V$, $f_W(x) = a[v_{out}]$

Some textbooks/convention don’t make the distinction between pre-activation and output value and simply compute $o[v] = \sigma(\sum_{u \rightarrow v \in E} W[u \rightarrow v] o[u])$.
Feed-Forward Neural Networks

Parameters:

• Each edge $u \to v$ has weight $W[u \to v]$

Activations:

• $\sigma: \mathbb{R} \to \mathbb{R}$, for example
  • $\sigma(z) = \text{sign}(z)$ or $\sigma(z) = \frac{1}{1+\exp(-z)}$
  • $\sigma(z) = \text{ReLU}(z) = \max(0, z)$
Feed-Forward Neural Networks

Deep learning
Generalize to hierarchy of transformations of the input, learned end-to-end jointly with the predictor.

\[ f_W(x) = f_L \left( f_{L-1} \left( f_{L-2} \left( \ldots f_1(x) \ldots \right) \right) \right) \]
Neural Nets as Feature Learning

• Can think of hidden layer as “features” $\phi(x)$, then a linear predictor based on $w \cdot \phi(x)$

• “Feature Engineering” approach: design $\phi(\cdot)$ based on domain knowledge

• “Deep Learning” approach: learn features from data

• Multilayer networks with non-linear activations
  o more and more complex features
Multi-Layer Feature Learning

Slide credit: Nati Srebro
More knowledge or more learning

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 knowledge: *full specific knowledge*

Expert Systems (no data at all)

more data →

no free lunch

Slide credit: Nati Srebro
Neural networks as hypothesis class

• Hypothesis class specified by:
  o Graph $G(V,E)$
  o Activation function $\sigma$
  o Weights $W$, with weight $W[u \rightarrow v]$ for each edge $u \rightarrow v \in E$

$$\mathcal{H} = \{ f_{G(V,E),\sigma,W} \mid W : E \rightarrow \mathbb{R} \}$$

• Expressive power:

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

• Computation: empirical risk minimization

$$\hat{W} = \arg\min_W \sum_{i=1}^N \ell(f_{G(V,E),\sigma,W}(x^{(i)}), y^{(i)})$$

  o Highly non-convex problem, even if loss $\ell$ is convex
  o Hard to minimize over even tiny neural networks are hard
So how do we learn?

\[
\hat{\mathbf{W}} = \arg \min_{\mathbf{W}} \sum_{i=1}^{N} \ell\left(f_{G(V,E),\sigma,\mathbf{W}}(\mathbf{x}^{(i)}), y^{(i)}\right)
\]

- Stochastic gradient descent: for random \((\mathbf{x}^{(i)}, y^{(i)}) \in S\)
  
  \[\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta^{(t)} \nabla \ell \left(f_{G(V,E),\sigma,\mathbf{W}^{(t)}}(\mathbf{x}^{(i)}), y^{(i)}\right)\]
  
  (Even though its not convex)

- How do we efficiently calculate
  
  \[\nabla \ell \left(f_{G(V,E),\sigma,\mathbf{W}^{(t)}}(\mathbf{x}^{(i)}), y^{(i)}\right)\]?

  - Karl will tell you!

- Now a brief detour into history and resurrection of NNs
Imagenet challenge – object classification

1000 kinds of objects.

Revolution of Depth

152 layers

ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

ImageNet Classification top-5 error (%)


(slid from Kaiming He’s recent presentation)
### PASCAL VOC Object Detection

<table>
<thead>
<tr>
<th>Year</th>
<th>bicycle</th>
<th>bus</th>
<th>car</th>
<th>motorbike</th>
<th>person</th>
<th>20 class average</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>36.9</td>
<td>23.2</td>
<td>34.6</td>
<td>27.6</td>
<td>21.3</td>
<td>17.1</td>
</tr>
<tr>
<td>2008</td>
<td>42.0</td>
<td>23.2</td>
<td>32.0</td>
<td>38.6</td>
<td>42.0</td>
<td>22.9</td>
</tr>
<tr>
<td>2009</td>
<td>46.8</td>
<td>43.8</td>
<td>37.2</td>
<td>42.0</td>
<td>41.5</td>
<td>27.9</td>
</tr>
<tr>
<td>2010</td>
<td>54.3</td>
<td>54.2</td>
<td>49.1</td>
<td>51.6</td>
<td>47.5</td>
<td>36.8</td>
</tr>
<tr>
<td>2011</td>
<td>58.1</td>
<td>57.6</td>
<td>54.4</td>
<td>58.3</td>
<td>51.6</td>
<td>40.9</td>
</tr>
<tr>
<td>2012</td>
<td>54.5</td>
<td>57.1</td>
<td>49.3</td>
<td>59.4</td>
<td>46.1</td>
<td>41.1</td>
</tr>
<tr>
<td>2013 DNN</td>
<td>56.3</td>
<td>51.4</td>
<td>48.7</td>
<td>59.8</td>
<td>44.4</td>
<td>43.2</td>
</tr>
<tr>
<td>2014 DNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>63.8</td>
</tr>
<tr>
<td>2015 ResNet</td>
<td>88.4</td>
<td>86.3</td>
<td>87.8</td>
<td>89.6</td>
<td>90.9</td>
<td>83.8</td>
</tr>
<tr>
<td>2016 ResNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
</tr>
</tbody>
</table>
History of Neural Networks

• 1940s-70s:
  o Inspired by learning in the brain, and as a model for the brain (Pitts, Hebb, and others)
  o Various models, directed and undirected, different activation and learning rules
  o Perceptron Rule (Rosenblatt), Problem of XOR, Multilayer perceptron (Minsky and Papert)
  o Backpropagation (Werbos 1975)

• 1980s-early 1990s:
  o Practical Backprop (Rumelhart, Hinton et al 1986) and SGD (Bottou)
  o Relationship to distributed computing; “Connectionism”
  o Initial empirical success

• 1990s-2000s:
  o Lost favor to implicit linear methods: SVM, Boosting

• 2000-2010s:
  o revival of interest (CIFAR groups)
  o ca. 2005: layer-wise pretraining of deepish nets
  o progress in speech and vision with deep neural nets

• 2010s:
  o Computational advances allow training HUGE networks
  o ...and also a few new tricks
  o Krizhevsky et al. win ImageNet
  o Empirical success and renewed interest
Deep learning - today

State of the art performance in several tasks and are actively deployed in real systems

- Computer vision
- Speech recognition
- Machine translation
- Dialog systems
- Computer games
- Information retrieval