CPSC 440: Advanced Machine Learning
Deep Structured Models

Mark Schmidt
University of British Columbia

Winter 2021
Backpropagation as Message-Passing

- Computing the gradient in neural networks is called **backpropagation**.  
  - Derived from the chain rule and memoization of repeated quantities.

- We’re going to view **backpropagation as a message-passing algorithm**.

- Key advantages of this view:
  - It’s easy to handle different graph structures.
  - It’s easy to handle different non-linear transformations.
  - It’s easy to handle multiple outputs (as in structured prediction).
  - It’s easy to add non-deterministic parts and combine with other graphical models.
Consider computing the output of a neural network for an example $i$,

$$y^i = v^T h(W^3 h(W^2 h(W^1 x^i)))$$

$$= \sum_{c=1}^{k} v_c h \left( \sum_{c'=1}^{k} W_{c'c}^3 h \left( \sum_{c''=1}^{k} W_{c''c'}^2 h \left( \sum_{j=1}^{d} W_{c''j}^1 x_j^i \right) \right) \right).$$

where we’ve assume that all hidden layers have $k$ values.

- In the second line, the $h$ functions are single-input single-output.

- The nested sum structure is similar to our message-passing structures.

- However, it’s easier because it’s deterministic: no random variables to sum over.
  - The messages will be scalars rather than functions.
Neural Networks and Message Passing

**Backpropagation Forward Pass**

- Forward propagation through neural network as message passing:

\[ y^i = \sum_{c=1}^{k} v_c h \left( \sum_{c'=1}^{k} W_{c'c}^3 h \left( \sum_{c''=1}^{k} W_{c''c'}^2 h \left( \sum_{j=1}^{d} W_{c''j}^1 x_j^i \right) \right) \right) \]

\[ = \sum_{c=1}^{k} v_c h \left( \sum_{c'=1}^{k} W_{c'c}^3 h \left( \sum_{c''=1}^{k} W_{c''c'}^2 h(M_{c''}) \right) \right) \]

\[ = \sum_{c=1}^{k} v_c h(M_{c'}) \]

\[ = \sum_{c=1}^{k} v_c h(M_y) \]

\[ = M_y, \]

where intermediate messages are the \( z \) values.
Backpropagation Backward Pass

- The backpropagation backward pass computes the partial derivatives.
  - For a loss $f$, the partial derivatives in the last layer have the form
    \[
    \frac{\partial f}{\partial v_c} = z^{i3}_c f'(v^T h(W^3 h(W^2 h(W^1 x^i)))) ,
    \]
    where
    \[
    z^{i3}_c = h \left( \sum_{c' = 1}^{k} W^3_{c'c} h \left( \sum_{c'' = 1}^{k} W^2_{c''c'} h \left( \sum_{j=1}^{d} W^1_{c''j} x^i_j \right) \right) \right).
    \]
  - Written in terms of messages it simplifies to
    \[
    \frac{\partial f}{\partial v_c} = h(M_c) f'(M_y).
    \]
In terms of forward messages, the partial derivatives have the forms:

\[
\frac{\partial f}{\partial v_c} = h(M_c) f'(M_y),
\]

\[
\frac{\partial f}{\partial W^3_{c'c}} = h(M_c') h'(M_c) w_c f'(M_y),
\]

\[
\frac{\partial f}{\partial W^2_{c''c'}} = h(M_{c''}) h'(M_{c'}) \sum_{c=1}^{k} W^3_{c'c} h'(M_c) w_c f'(M_y),
\]

\[
\frac{\partial f}{\partial W^1_{j_{c''}}} = h(M_j) h'(M_{c''}) \sum_{c'=1}^{k} W^2_{c''c'} h'(M_{c'}) \sum_{c=1}^{k} W^3_{c'c} h'(M_c) w_c f'(M_y),
\]

which are ugly but notice all the repeated calculations.
Neural Networks and Message Passing

Backpropagation Backward Pass

- It’s again simpler using appropriate messages

\[
\frac{\partial f}{\partial v_c} = h(M_c)f'(M_y),
\]

\[
\frac{\partial f}{\partial W^3_{c'c}} = h(M_{c'})h'(M_c)w_cV_y,
\]

\[
\frac{\partial f}{\partial W^2_{c''c'}} = h(M_{c''})h'(M_{c'}) \sum_{c=1}^{k} W^3_{c'c} V_c,
\]

\[
\frac{\partial f}{\partial W^1_{j_{c''c'}}} = h(M_j)h'(M_{c''}) \sum_{c'=1}^{k} W^2_{c''c'} V_{c'},
\]

where \( M_j = x_j \).
Backpropagation as Message-Passing

- The general forward message for child $c$ with parents $p$ and weights $W$ is
  \[ M_c = \sum_p W_{cp} h(M_p), \]
  which computes weighted combination of non-linearly transformed parents.
  - In the first layer we don’t apply $h$ to $x$.
- The general backward message from child $c$ to all its parents is
  \[ V_c = h'(M_c) \sum_{c'} W_{cc'} V_{c'}, \]
  which weights the “grandchildren’s gradients”.
  - In the last layer we use $f$ instead of $h$.
- The gradient of $W_{cp}$ is $h(M_c)V_p$, which works for general graphs.
Automatic Differentiation

- **Automatic differentiation:**
  - Input is code that computes a function value.
  - Output is code computing is one or more derivatives of the function.

- **Forward-mode** automatic differentiation:
  - Computes a directional derivative for cost of evaluating function.
    - So computing gradient would be \( d \)-times more expensive than function.
  - Low memory requirements.
  - Most useful for evaluating Hessian-vector products, \( \nabla^2 f(w)d \).

- **Reverse-mode** automatic differentiation:
  - Computes gradient for cost of evaluating function.
  - But high memory requirements: need to store intermediate calculations.
    - Backpropagation is (essentially) a special case.
  - Reverse-mode is replacing “gradient by hand” (less time-consuming/bug-prone).
Combining Neural Networks and CRFs

- Previously we saw conditional random fields like

  \[ p(y \mid x) \propto \exp \left( \sum_{c=1}^{k} y_c v^T x_c + \sum_{(c,c') \in E} y_c y_{c'} w \right), \]

  which can use logistic regression at each location \( c \) and Ising dependence on \( y_c \).

- Instead of logistic regression, you could put a neural network in there:

  \[ p(y \mid x) \propto \exp \left( \sum_{c=1}^{k} y_c v^T h(W^3 h(W^2(W^1 x_c))) + \sum_{(c,c') \in E} y_c y_{c'} w \right). \]

  Sometimes called a conditional neural field or deep structured model.

- Backprop generalizes:
  1. Forward pass through neural network to get \( \hat{y}_c \) predictions.
  2. Belief propagation to get marginals of \( y_c \) (or Gibbs sampling if high treewidth).
  3. Backwards pass through neural network to get all gradients.
Consider multi-label classification:

- Flickr dataset: each image can have multiple labels (out of 38 possibilities).
- Use neural networks to generate “factors” in an undirected model.
  - Decoding undirected model makes predictions accounting for label correlations.

Neural Networks and Message Passing

**Multi-Label Classification**

- **Learned correlation matrix:**

<table>
<thead>
<tr>
<th>female</th>
<th>people</th>
<th>indoor</th>
<th>baby</th>
<th>sea</th>
<th>portrait</th>
<th>transport</th>
<th>flower</th>
<th>sky</th>
<th>lake</th>
<th>structures</th>
<th>bird</th>
<th>plant life</th>
<th>food</th>
<th>male</th>
<th>clouds</th>
<th>water</th>
<th>animals</th>
<th>car</th>
<th>tree</th>
<th>dog</th>
<th>sunset</th>
<th>night</th>
<th>river</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.68</td>
<td>0.56</td>
<td>0.55</td>
<td>0.32</td>
<td>0.24</td>
<td>0.25</td>
<td>0.25</td>
<td>-0.07</td>
<td>-0.03</td>
<td>-0.01</td>
<td>-0.02</td>
<td>-0.04</td>
<td>-0.03</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>-0.12</td>
<td>-0.12</td>
<td>-0.12</td>
<td>-0.12</td>
<td>-0.12</td>
<td>0.12</td>
<td>0.12</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Automatic Differentiation (AD) vs. Inference

- If you use exact inference methods, automatic differentiation will give gradient.
  - You write message-passing code to compute $Z$.
  - AD modifies your code to compute expectations in gradient.

- With approximate inference, AD may or may not work:
  - AD will work for iterative variational inference methods (which we’ll cover later).
  - AD will not tend to work for Monte Carlo methods.
    - Can't AD through sampling (but there exist tricks like “common random numbers”).

- Recent trend: run iterative variational method for a fixed number of iterations.
  - AD can give gradient of result after this fixed number of iterations.
  - “Train the inference you will use at test time”.
Deep Learning for Structured Prediction (Big Picture)

- How is deep learning being used for structured prediction applications?
  - Discriminative approaches are most popular.

- Typically you will send $x$ through a neural network to get representation $z$, then:
  1. Perform inference on $p(y | z)$ (backpropagate using exact/approximate marginals).
     - Neural network learns features, CRF “on top” models dependencies in $y_c$.
  2. Run $m$ approximate inference steps on $p(y | z)$, backpropagate through these steps.
     - “Learn to use the inference you will be using” (usually with variational inference).
  3. Just model each $p(y_c | z)$ (treat labels as independent given representation).
     - Assume that structure is already captured in neural network goo (no inference).

- Current trend: less dependence on inference and more on learning representation.
  - “Just use an RNN rather than thinking about stochastic grammars.”
  - We’re improving a lot at learning features, less so for inference.
  - This trend may or may not reverse in the future...
Neural Networks with Latent-Dynamics

- Instead of modeling $y$ dependencies, could random $z$ values.
  - Like an HMM with neural networks defining the hidden dynamics.

- Combines deep learning, mixture models, and graphical models.
  - “Latent-dynamics model”.
  - Previously achieved among state of the art in several applications.
Summary

- **Implicit regularization:**
  - Some optimization methods may converge to regularized solutions.

- **Double descent curves:**
  - Weird phenomenon from increasing regularization as you increase complexity.

- **Backpropagation** can be viewed as a **message passing** algorithm.

- **Combining CRFs with deep learning.**
  - You can learn the features and the label dependency at the same time.

- **Reducing the reliance on inference** is a current trend in the field.
  - Rely on neural network to learn clusters and dependencies.

- Next time: “end-to-end” learning.