CPSC 540: Machine Learning
Deep Structured Models

Mark Schmidt

University of British Columbia

Winter 2019
Feedforward Neural Networks

- In 340 we discussed feedforward neural networks for supervised learning.
- With 1 hidden layer the classic model has this structure:

Motivation:
- For some problems it’s hard to find good features.
- This learns features $z$ that are good for particular supervised learning problem.
Neural Networks as DAG Models

- It’s a **DAG** model but there is an important difference with our previous models:
  - The latent variables $z_c$ are deterministic functions of the $x_j$.

- Makes inference given $x$ trivial: if you observe all $x_j$ you also observe all $z_c$.
  - In this case $y$ is the only random variable.
Neural Network Notation

- We’ll continue using our supervised learning notation:

\[
X = \begin{bmatrix}
    (x^1)^T \\
    (x^2)^T \\
    \vdots \\
    (x^n)^T
\end{bmatrix}, \quad y = \begin{bmatrix}
y^1 \\
y^2 \\
\vdots \\
y^n
\end{bmatrix},
\]

- For the latent features and one hidden layer we’ll use

\[
Z = \begin{bmatrix}
    (z^1)^T \\
    (z^2)^T \\
    \vdots \\
    (z^n)^T
\end{bmatrix}, \quad v = \begin{bmatrix}
v_1 \\
v_2 \\
\vdots \\
v_k
\end{bmatrix}, \quad W = \begin{bmatrix}
w_1 \\
w_2 \\
\vdots \\
w_k
\end{bmatrix},
\]

where \(Z\) is \(n\) by \(k\) and \(W\) is \(k\) by \(d\).
Introducing Non-Linearity

- We discussed how the “linear-linear” model,
  \[ z^i = W x^i, \quad \hat{y}^i = v^T z^i, \]
  is degenerate since it’s still a linear model.

- The classic solution is to introduce a non-linearity,
  \[ z^i = h(W x^i), \quad \hat{y}^i = v^T z^i, \]
  where a common-choice is applying sigmoid element-wise,
  \[ z_c^i = \frac{1}{1 + \exp(-w_c^T x^i)}, \]
  which is said to be the “activation” of neuron \( c \) on example \( i \).

  - A universal approximator with \( k \) a function of \( n \) (also true for \( \tanh \), ReLU, etc.)
Deep Neural Networks

- In deep neural networks we add multiple hidden layers,

Mathematically, with 3 hidden layers the classic model uses

\[ \hat{y}^i = v^T h(W^3 h(W^2 h(W^1 x^i))) \]

\[ z^{i1} \]

\[ z^{i2} \]

\[ z^{i3} \]
Biological Motivation

- Deep learning is motivated by theories of deep hierarchies in the brain.

But most research is about making models work better, not be more brain-like.
Deep Neural Network History

- Popularity of deep learning has come in waves over the years.
  - Currently, it is one of the hottest topics in science.

- Recent popularity is due to unprecedented performance on some difficult tasks:
  - Speech recognition.
  - Computer vision.
  - Machine translation.

- These are mainly due to big datasets, deep models, and tons of computation.
  - Plus tweaks to classic models and focus on structured networks (CNNs, LSTMs).

- For a NY Times article discussing some of the history/successes/issues, see:
  https://mobile.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html
Training Deep Neural Networks

- If we’re training a 3-layer network with squared error, our objective is
  \[ f(v, W^1, W^2, W^3) = \frac{1}{2} \sum_{i=1}^{n} (v^T h(W^3 h(W^2 h(W^1 x^i))) - \hat{y}^i)^2. \]

- Usual training procedure is **stochastic gradient**.
  - But we’re discovering sets of **tricks to make things easier to tune**.

- **Highly non-convex and notoriously difficult to tune.**

- Recent empirical/theoretical work indicates non-convexity may not be an issue:
  - All local minima may be good for “large enough” networks.
Training Deep Neural Networks

Some common data/optimization tricks we discussed in 340:

- **Data transformations.**
  - For images, translate/rotate/scale/crop each $x^i$ to make more data.
- **Data standardization:** centering and whitening.
- **Adding bias variables.**
- **Parameter initialization:** “small but different”, standardizing within layers.
- **Step-size selection:** “babysitting”, Bottou trick, Adam.
- **Momentum:** heavy-ball and Nesterov-style modifications.
- **Batch normalization:** adaptive standardizing within layers.
- **ReLU:** replacing sigmoid with $\max\{0, w^T_c x^i\}$.
  - Avoids gradients extremely-close to zero.
Common forms tricks to fight overfitting:

- Standard **L2-regularization** or **L1-regularization** “weight decay”.
  - Sometimes with different $\lambda$ for each layer.
  - Recent work shows this *introduces bad local optima*.

- **Early stopping** of the optimization based on validation accuracy.

- **Dropout** randomly zeroes $z$ values to discourage dependence.

- **Implicit regularization** from using SGD.

- **Hyper-parameter optimization** to choose various tuning parameters.

- **Special architectures** like **convolutional neural networks**:  
  - Yields $W^m$ that are *very sparse* and have many *tied parameters*.  


“Residual” Networks (ResNets)

- Suppose we fit a deep neural network to a *linearly-separable* dataset.
  - Original features $x$ are sufficient to perfectly classify training data.
  - For a deep neural network to work, each layer needs to preserve information in $x$.
    - You might be “wasting” parameters just re-representing data from previous layers.

- Consider **residual networks**:

![ResNet Diagram](https://en.wikipedia.org/wiki/Residual_neural_network)

- Take a previous (non-transformed) layer as input to current layer.
  - Also called “skip connections” or “highway networks”.

“Residual” Networks (ResNets)

- ResNets seemingly make learning easier:
  - You can “default” to just copying the previous layer.
  - The non-linear transform is only learning how to modify the input.
    - “Fitting the residual”.

- This was a key idea behind first methods that used 100+ layers.
  - Easy for information about $x$ to reach $y$ through huge number of layers.
  - Won all tasks in ImageNet 2015 competition.
  - Evidence that biological networks have skip connections like this.

- **Dense networks** (DenseNets): connect to many previous layers.
  - Basically gets rid of vanishing gradient issue.
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

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.
Outline

1 Neural Networks Review

2 Neural Networks and Message Passing
Backpropagation Forward Pass

- Consider computing the output of a neural network for an example $i$,

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

$$= \sum_{c=1}^{k} v_c h \left( \sum_{c'=1}^{k} W^3_{c,c'} h \left( \sum_{c''=1}^{k} W^2_{c'',c'} h \left( \sum_{j=1}^{d} W^1_{c'',j} x^i \right) \right) \right).$$

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.
Backpropagation Forward Pass

- Forward propagation through neural network as message passing:

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

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

$$\frac{\partial f}{\partial v_c} = z_{c}^{i3} f'(v^T h(W^3 h(W^2 h(W^1 x^i)))),$$

where

$$z_{c}^{i3} = h \left( \sum_{c' = 1}^{k} W_{c'c}^3 h \left( \sum_{c'' = 1}^{k} W_{c''c'}^2 h \left( \sum_{j=1}^{d} W_{c''j}^1 x_j^i \right) \right) \right).$$

- Written in terms of messages it simplifies to

$$\frac{\partial f}{\partial v_c} = h(M_c)f'(M_y).$$
Backpropagation Backward Pass

In terms of forward messages, the partial derivatives have the forms:

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

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

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

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

which are ugly but notice all the repeated calculations.
Backpropagation Backward Pass

- It’s again simpler using appropriate messages

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

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

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

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

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 a function.
  - Output 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

- Last time 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.
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 late).
  - 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”. 
Motivation: Gesture Recognition

- Want to recognize gestures from video:

- A gesture is composed of a sequence of parts:
  - And some parts appear in different gestures.

http://groups.csail.mit.edu/vision/vip/papers/wang06cvpr.pdf
Motivation: Gesture Recognition

- We may not know the set of “parts” that make up gestures.

- We can consider learn the “parts” and their latent dynamics (transitions).

http://groups.csail.mit.edu/vision/vip/papers/wang06cvpr.pdf
Motivation: Gesture Recognition

- We’re given a labeled video sequence, but don’t observe “parts”:

Our videos are labeled with “gesture” and “background” frames,
- But we don’t know the parts (G1, G2, G3, B1, B2, B3) that define the labels.
Latent-Dynamic Conditional Random Field

- Here we could use a **latent-dynamic conditional random field**

- Observed variable $x_j$ is the image at time $j$ (in this case $x_j$ is a video frame).
- The gesture $y$ is defined by sequence of parts $z_j$.
  - We’re learning what the parts should be.
  - We’re learning “latent dynamics”: how the hidden parts change over time.
- Notice in the above case that the conditional UGM is a tree.
Neural Networks with Latent-Dynamics

- Neural networks with **latent dynamics**: 

![Diagram of neural networks with latent dynamics](image)

- Combines deep learning, mixture models, and graphical models.
  - Achieved among state of the art in several applications.
Convolutional Neural Networks

In 340 we discussed **convolutional neural networks** (CNNs):

- **Convolutional layers** where $W$ acts like a convolution (sparse with tied parameters).
- **Pooling layers** that usually take maximum among a small spatial neighbourhood.
- **Fully-connected layers** that use an unrestricted $W$.

http://blog.csdn.net/strint/article/details/44163869
Motivation: Beyond Classification

- **Convolutional** structure simplifies the learning task:
  - **Parameter tying** means we have more data to estimate each parameter.
  - **Sparsity** drastically reduces the number of parameters.

We discussed CNNs for **image classification**: “is this an image of a cat?”. But many vision tasks are **not image classification** tasks.
Object Localization

- **Object localization** is task of finding locations of objects:
  - Need to find *where* in the image the object is.
  - May need to recognize *more than one* object.

Region Convolutional Neural Networks: “Pipeline” Approach

- Early approach (region CNN):
  1. Propose a bunch of potential boxes.
  2. Compute features of box using a CNN.
  3. Classify each box based on an SVM.
  4. Refine each box using linear regression.


- Improved on state of the art, but not very elegant with its 4 steps.
Summary

- **Neural networks** learn features for supervised learning.
  - For structured prediction, may reduce the need to rely on inference.

- **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.

- Next time: “end-to-end” learning