Admin

• **Assignment 5:**
  – Due Monday, 1 late day for Wednesday, 2 for next Friday.

• **Final:**
  – Details and previous exams posted on Piazza.

• **Extra office hours:**
  – 3:00 next Thursday (with me).
  – Monday/Tuesday (with TAs).
Last Time: Deep Learning

Deep neural networks:
\[ y_i = w^T h(W^{(2)} h(W^{(1)} x_i)) \]

- Unprecedented performance on difficult problems.
- Each layer combines "parts" from previous layer.

"Non-linear \( h \) makes it a universal approximator for large \( K \)."

"Learn \( W \) and \( w \) together."
"Learn features for supervised learning."
Deep Learning Practicalities

• This lecture focus on deep learning practical issues:
  – Backpropagation to compute gradients.
  – Stochastic gradient training.
  – Regularization to avoid overfitting.

• Next lecture:
  – Special ‘W’ restrictions to further avoid overfitting.
But first: Adding Bias Variables

• Recall fitting line regression with a bias:

\[ y_i = \sum_{j=1}^{d} w_j x_{ij} + \beta \]

– We avoided this by adding a column of ones to \( X \).

• In neural networks we often want a bias on the output:

\[ y_i = \sum_{c=1}^{k} w_c h(w_c x_i) + \beta \]

• But we also often also include biases on each \( z_{ic} \):

\[ y_i = \sum_{c=1}^{k} w_c h(w_c x_i + \beta_c) + \beta \]

– A bias towards this \( h(z_{ic}) \) being either 0 or 1.

• Equivalent to adding to vector \( h(z_i) \) an extra value that is always 1.
  – For sigmoids, you could equivalently make one row \( W_c \) be equal to 0.
But first: Adding Bias Variables

Linear model with bias:
Artificial Neural Networks

- With squared loss, our objective function is:
  \[ f(w, W) = \frac{1}{2} \sum_{i=1}^{n} (w^T h(Wx_i) - y_i)^2 \]

- Usual training procedure: **stochastic gradient**.
  - Compute gradient of random example ‘i’, update both ‘w’ and ‘W’.
  - Highly non-convex and can be difficult to tune.

- Computing the gradient is known as “**backpropagation**”.
  - Video giving motivation [here](#).
Backpropagation

• Overview of how we compute neural network gradient:
  
  – Forward propagation:
    • Compute $z_i^{(1)}$ from $x_i$.
    • Compute $z_i^{(2)}$ from $z_i^{(1)}$.
    • …
    • Compute $y_{	ext{hat}}_i$ from $z_i^{(m)}$, and use this to compute error.
  
  – Backpropagation:
    • Compute gradient with respect to regression weights ‘$w$’.
    • Compute gradient with respect to $z_i^{(m)}$ weights $W^{(m)}$.
    • Compute gradient with respect to $z_i^{(m-1)}$ weights $W^{(m-1)}$.
    • …
    • Compute gradient with respect to $z_i^{(1)}$ weights $W^{(1)}$.

• “Backpropagation” is the chain rule plus some bookkeeping for speed.
Let’s illustrate backpropagation in a simple setting:

– 1 training example, 3 hidden layers, 1 hidden “unit” in layer.

\[
f(W^{(1)}, W^{(2)}, W^{(3)}, w) = \frac{1}{2} \left( \hat{y}_i - y_i \right)^2 \text{ where } \hat{y}_i = w h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i)))
\]

\[
\begin{align*}
\frac{2f}{2w} &= \nabla h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) = \nabla h(z_i^{(3)}) \\
\frac{2f}{2w^{(3)}} &= \nabla w h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) h(W^{(2)} h(W^{(1)} x_i)) = \nabla w h(z_i^{(3)}) h(z_i^{(2)})
\end{align*}
\]
Backpropagation

- Let’s illustrate backpropagation in a simple setting:
  - 1 training example, 3 hidden layers, 1 hidden “unit” in layer.

\[
f(W^{(1)}, W^{(2)}, W^{(3)}, w) = \frac{1}{z} \left( \hat{y}_i - y_i \right)^2 \quad \text{where} \quad \hat{y}_i = w_0 h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i)))
\]

\[
\frac{2f}{2w} = \nabla h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) = \nabla h(z_i^{(3)})
\]

\[
\frac{2f}{2w^{(3)}} = \nabla h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) h(W^{(2)} h(W^{(1)} x_i)) = \nabla h(z_i^{(3)}) h(z_i^{(2)})
\]

\[
\frac{2f}{2w^{(2)}} = -\nabla h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) h(W^{(1)} x_i) = \nabla h(z_i^{(3)}) h(z_i^{(2)}) h(z_i^{(1)})
\]

\[
\frac{2f}{2w^{(1)}} = \nabla h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) W^{(2)} h(W^{(1)} x_i) x_i = \nabla h(z_i^{(3)}) h(z_i^{(2)}) x_i
\]
Backpropagation

Let’s illustrate backpropagation in a simple setting:

- 1 training example, 3 hidden layers, 1 hidden “unit” in layer.

$$\frac{2f}{2w} = \frac{1}{r} h(z_i^{(3)})$$

$$\frac{2f}{2w^{(3)}} = \frac{1}{r} w h'(z_i^{(3)}) h(z_i^{(2)})$$

$$\frac{2f}{2w^{(2)}} = r^{(3)} W^{(3)} h'(z_i^{(2)}) h(z_i^{(1)})$$

$$\frac{2f}{2w^{(1)}} = r^{(2)} W^{(2)} h'(z_i^{(1)}) x_i$$

- Only the first ‘r’ changes if you use a different loss.
- With multiple hidden units, you get extra sums.

- Efficient if you store the sums rather than computing from scratch.
Last Time: Backpropagation with 3 Hidden Layers

• I’ve marked those backprop math slides as bonus.
• Do you need to know how to do this?
  – Exact details are probably not vital (there are many implementations), but understanding basic idea helps you know what can go wrong.
  – See discussion here by a neural network expert.

• You should know cost of backpropagation:
  – Forward pass dominated by matrix multiplications by $W^{(1)}$, $W^{(2)}$, $W^{(3)}$, and ‘w’.
    • If have ‘m’ layers and all $z_i$ have ‘k’ elements, cost would be $O(dk + mk^2)$.
  – Backward pass has same cost as forward pass.
• For multi-class or multi-label classification, you replace ‘w’ by a matrix:
  – Softmax loss is often called “cross entropy” in neural network papers.
(pause)
Last Time: ImageNet Challenge

• ImageNet challenge:
  – Use millions of images to recognize 1000 objects.

• ImageNet organizer visited UBC summer 2015.

• “Besides huge dataset/model/cluster, what is the most important?”
  1. Image transformations (translation, rotation, scaling, lighting, etc.).
  2. Optimization.

• Why would optimization be so important?
  – Neural network objectives are highly non-convex (and worse with depth).
  – Optimization has huge influence on quality of model.
Stochastic Gradient Training

• Standard training method is **stochastic gradient (SG):**
  – Choose a random example ‘i’.
  – Use backpropagation to get gradient with respect to all parameters.
  – Take a small step in the negative gradient direction.

• **Challenging to make SG work:**
  – Often doesn’t work as a “black box” learning algorithm.
  – But people have developed a lot of tricks/modifications to make it work.

• **Highly non-convex, so are the problem local mimina?**
  – Some empirical/theoretical evidence that local minima are not the problem.
  – If the network is “deep” and “wide” enough, we think all local minima are good.
  – But it can be hard to get SG to even find a local minimum.
Parameter Initialization

- **Parameter initialization** is crucial:
  - Can’t initialize weights in same layer to same value, or they will stay same.
  - Can’t initialize weights too large, it will take too long to learn.

- **A traditional random initialization**:
  - Initialize bias variables to 0.
  - Sample from standard normal, divided by $10^5$ ($0.00001 \times \text{randn}$).
    - $w = 0.00001 \times \text{randn}(k,1)$
  - Performing multiple initializations does not seem to be important.

- **Popular approach from 10 years ago**:
  - Initialize with deep unsupervised model (like “autoencoders” – see bonus).
Parameter Initialization

- **Parameter initialization** is crucial:
  - Can’t initialize weights in same layer to same value, or they will stay same.
  - Can’t initialize weights too large, it will take too long to learn.

- Also common to **standardize data**:
  - Subtract mean, divide by standard deviation, “whiten”, standardize $y_i$.

- More recent initializations try to **standardize initial $z_i$**:
  - Use different initialization in each layer.
  - Try to make variance of $z_i$ the same across layers.
  - Use samples from standard normal distribution, divide by $\sqrt{2*\text{nInputs}}$.
  - Use samples from uniform distribution on $[-b,b]$, where $b = \frac{\sqrt{6}}{\sqrt{k(m)+k(m'-1)}}$. 
Setting the Step-Size

• Stochastic gradient is very sensitive to the step size in deep models.
• Common approach: manual “babysitting” of the step-size.
  – Run SG for a while with a fixed step-size.
  – Occasionally measure error and plot progress:

  ![Graph showing error over time](image)

  – If error is not decreasing, decrease step-size.
Setting the Step-Size

- Stochastic gradient is very sensitive to the step size in deep models.
- Bias step-size multiplier: use bigger step-size for the bias variables.
- Momentum:
  - Add term that moves in previous direction:
    \[ w^{t+1} = w^t - \alpha^t \nabla f_i(w^t) + \beta^t(w^t - w^{t-1}) \]
    - Usually $\beta^t = 0.9$. 
Setting the Step-Size

• Automatic method to set step size is Bottou trick:
  1. Grab a small set of training examples (maybe 5% of total).
  2. Do a binary search for a step size that works well on them.
  3. Use this step size for a long time (or slowly decrease it from there).

• Several recent methods using a step size for each variable:
  – AdaGrad, RMSprop, Adam (often work better “out of the box”).
  – Seem to be losing popularity to stochastic gradient (often with momentum).
    • Often yields lower test error but this requires more tuning of step-size.

• Batch size (number of random examples) also influences results.
  – Bigger batch sizes often give faster convergence but to worse solutions.

• Another recent trick is batch normalization:
  – Try to “standardize” the hidden units within the random samples as we go.
Vanishing Gradient Problem

- Consider the sigmoid function:

- Away from the origin, the gradient is nearly zero.
- The problem gets worse when you take the sigmoid of a sigmoid:

- In deep networks, many gradients can be nearly zero everywhere.
Rectified Linear Units (ReLU)

• Replace sigmoid with hinge-like loss (ReLU):
  \[
  \max \{ 0, W_c x_i \} \\
  \frac{1}{\text{tanh}(-W_c x_i)}
  \]

• Just sets negative values \( z_{ic} \) to zero.
  - Fixes vanishing gradient problem.
  - Gives sparser of activations.
  - Not really simulating binary signal, but could be simulating rate coding.
Deep Learning and the Fundamental Trade-Off

• Neural networks are subject to the fundamental trade-off:
  – As we increase the depth, training error decreases.
  – As we increase the depth, training error no longer approximates test error.

• We want deep networks to model highly non-linear data.
  – But increasing the depth leads to overfitting.

• How could GoogLeNet use 22 layers?
  – Many forms of regularization and keeping model complexity under control.
Standard Regularization

- We typically add our usual \textbf{L2-regularizers}:

\[
\hat{f}(w_1, W^{(3)}, W^{(2)}, W^{(1)}) = \frac{1}{2} \sum_{i=1}^{n} \left( w^T h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) - y_i \right)^2 + \frac{\lambda_1}{2} \|w\|^2 + \frac{\lambda_2}{2} \|W^{(3)}\|^2 + \frac{\lambda_3}{2} \|W^{(2)}\|^2 + \frac{\lambda_4}{2} \|W^{(1)}\|^2
\]

- L2-regularization is called “weight decay” in neural network papers.
  - Could also use L1-regularization.

- “Hyper-parameter” optimization:
  - Try to optimize validation error in terms of $\lambda_1, \lambda_2, \lambda_3, \lambda_4$.

- Unlike linear models, typically use \textbf{multiple types of regularization}. 

Early Stopping

• Second common type of regularization is “early stopping”:
  – Monitor the validation error as we run stochastic gradient.
  – Stop the algorithm if validation error starts increasing.

Dropout

- **Dropout** is a more recent form of regularization:
  - On each iteration, randomly set some $x_i$ and $z_i$ to zero (often use 50%).
  - Encourages distributed representation rather than using specific $z_i$.
  - Like ensembling a lot of models but without the high computational cost.
  - After a lot of success, dropout may already be going out of fashion.

[Diagram: (a) Standard Neural Net vs. (b) After applying dropout.]

Summary

• **Backpropagation** computes neural network gradient via chain rule.
• **Parameter initialization** is crucial to neural net performance.
• **Optimization and step size** are crucial to neural net performance.
• **Regularization** is crucial to neural net performance:
  – L2-regularization, early stopping, dropout.

• Next time:
  – The other crucial piece to get these working for vision problems.
Autoencoders

- **Autoencoders** are an **unsupervised deep learning model**:
  - Use the **inputs as the output** of the neural network.
  - Middle layer could be latent features in **non-linear latent-factor model**.
  - Can do outlier detection, data compression, visualization, etc.
  - A non-linear generalization of PCA.
Autoencoders

PCA

Autoencoder
Denoising Autoencoder

- Denoising autoencoders add noise to the input:

  - Learns a model that can remove the noise.