Announcements

- No homework 3, Kaggle and Lab 4 were due on Sunday

- Tests are almost graded
  - There will be a big curve so don’t worry too much about your grades

- There will be an option in a few weeks for a take home test for a bit of extra credit.
  - If you are above the median and have turned in labs and homeworks please don’t do this—you don’t need the credit.

Regularization in Neural Networks

Continuation from last lecture
Model Complexity

- Models range in their flexibility to fit arbitrary data

Simple model vs. Complex model:
- Simple model: high bias, low variance
- Complex model: low bias, high variance

Constrained vs. Unconstrained:
- Constrained: small capacity, may prevent it from representing all structure in data
- Unconstrained: large capacity, may allow it to memorize data and fail to capture regularities

Overfitting

- Neural networks can easily overfit data
  - There's a lot of parameters (each weight)
- How can we avoid overfitting?
  - Weight decay
  - Weight regularization
  - Early stopping
  - Model averaging
  - Dropout

Regularization Techniques

- Instead of starting with smallest net possible, use a larger network and apply various tricks to avoid using the full network capacity

  - Weight decay
  - Weight regularization
  - Early stopping
  - Model averaging
  - Dropout

Regularization Techniques

1. Early stopping
   - Rather than training network until error converges, stop training early
     - Rumelhart
       - Hidden units all go after the same source of error initially -> redundancy
     - Hinton
       - Weights start small and grow over training
         - When weights are small, model is mostly operating in linear regime
   - Dangerous: Very dependent on training algorithm
     - E.g., what would happen with random weight search?
   - While probably not the best technique for controlling model complexity, it does suggest that you shouldn’t obsess over finding a minimum error solution.

7 ideas to follow…
When To Stop Training (‘regularization’ 1)

- A. Train $n$ epochs; lower learning rate; train $m$ epochs
  - bad idea: can’t assume one-size-fits-all approach
- B. Error-change criterion
  - stop when error isn’t dropping
  - My recommendation: criterion based on % drop over a window of, say, 10 epochs
    - 1 epoch is too noisy
    - absolute error criterion is too problem dependent

When To Stop Training (‘regularization’ 1)

- C. Weight-change criterion
  - Compare weights at epochs t-10 and t and test:
    - Don’t base on length of overall weight change vector
    - Possibly express as a percentage of the weight
    - Be cautious: small weight changes at critical points can result in rapid drop in error

Regularization Techniques

2. Weight penalty terms

- L2 weight decay
  \[ E = \frac{1}{2} \sum_j (t_j - y_j)^2 + \frac{\lambda}{2} \sum_{i,j} w_{ij}^2 \]
  \[ \Delta w_{ij} = \epsilon \delta_j x_i - \epsilon \lambda w_{ij} \]
- L1 weight decay
  \[ E = \frac{1}{2} \sum_j (t_j - y_j)^2 + \frac{\lambda}{2} \sum_{i,j} |w_{ij}| \]
  \[ \Delta w_{ij} = \epsilon \delta_j x_i - \epsilon \lambda \text{sign}(w_{ij}) \]
- weight elimination
  \[ E = \frac{1}{2} \sum_j (t_j - y_j)^2 + \frac{1}{2} \sum_{i,j} \frac{w_{ij}^2}{1 + w_{ij}^2 / w_0^2} \]

Regularization Techniques

3. Hard constraint on weights
  - Ensure that $\sum w_{ij} < \Phi$ for every unit
  - If constraint is violated, rescale all weights: $w_{ij} \leftarrow w_{ij} \frac{\phi}{\sum_i w_{ij}}$
    - I’m not clear why L$_2$ normalization and not L$_1$

4. Injecting noise
  - Not covered here…
**Regularization Techniques**

6. Model averaging
   - Ensemble methods
   - Bayesian methods

7. Drop out

**More On Dropout**

- With $H$ hidden units, each of which can be dropped, we have $2^H$ possible models
- Each of the $2^{H-1}$ models that include hidden unit $h$ must share the same weights for the units
  - serves as a form of regularization
  - makes the models cooperate
- Including all hidden units at test with a scaling of 0.5 is equivalent to computing the geometric mean of all $2^H$ models
  - exact equivalence with one hidden layer
  - "pretty good approximation" according to Geoff with multiple hidden layers

**Graphical representations of models**

A quick aside to explain figures in book and various research papers
Graphical Models

- Nodes and edges
  - Nodes represent random variables
  - Edges represent statistical dependence between random variables
    - Conditional dependence

Graphical Models

- Advantages:
  - Structure brings statistical and computational efficiencies
  - Less data needed
  - Less time to perform inference

Examples: Markov chains

- First order Markov chain
  \[ p(x_t, x_{t-1}) = p(x_t) \prod_{t=2}^{T} p(x_t | x_{t-1}) \]
  \( x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \cdots \)

- Second order Markov chain
  \[ p(x_t, x_{t-1}, x_{t-2}) = p(x_t | x_{t-1}, x_{t-2}) \prod_{t=3}^{T} p(x_t | x_{t-1}, x_{t-2}) \]
  \( x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow \cdots \)

Examples: Markov chains

- Given that today was a bull market, what is the probability that tomorrow will be a bear market?
- Can look forward and backward in time.
- Further from current state, more uncertainty
Examples: Markov chains

• Given that today was a bull market, what is the probability that tomorrow will be a bear market?
• Can look forward and backward in time.
• Further from current state, more uncertainty

\[
P = \begin{bmatrix}
0.9 & 0.075 & 0.025 \\
0.15 & 0.8 & 0.05 \\
0.25 & 0.25 & 0.5 \\
\end{bmatrix}
\]

Examples: Bayes Nets

• Given that today was a bull market, what is the probability that tomorrow will be a bear market?
• Can look forward and backward in time.
• Further from current state, more uncertainty

\[
p_{10} = \begin{bmatrix}
0.9 & 0.075 & 0.025 \\
0.15 & 0.8 & 0.05 \\
0.25 & 0.25 & 0.5 \\
\end{bmatrix}
\]

Example: Academic Life

What is the expected lifetime income of an academic?
Examples: Bayes Nets

- If the alarm is going off what's the probability of an earthquake?
- We can think of these arrows as causal.
- Allows for probabilistic reasoning about causes and effects.

![Bayes Net Diagram]

What does this buy us?

- All the advantages of Bayesian models.
  - Prior knowledge is embedded in the graphical structure
  - Little data can still allow for meaningful predictions
  - Inference techniques
- Limited parameters but still lots of power
- We can abstract further, getting rid of even more parameters
  - We’ll see this by introducing latent variables

![Bayes Net Diagram]
Why do we need this?

- For one, lots of algorithms are designed from this type of thinking.
  - Decision trees
  - Naïve Bayes classifiers
  - Hidden Markov Models
- Unsupervised learning techniques
  - Inference over latent (unobserved) states.
  - Exploration of causality.
  - Formalization of a class of models.

Some standard algorithms

- Hidden Markov models

![Hidden Markov model diagram]

- Naïve Bayes Classifier

![Naïve Bayes Classifier diagram]

Gaussian Mixture Models
Latent Variable Models

- We often see dependence between variables in our data
- We can assume that the dependence can be captured by imposing structure in our model (e.g. NB, LDA etc.)
- Many of the basic (graphical) models assume that the observed variables are correlated with each other.
- What if instead we assume that there’s a “hidden” common cause?
- Then we’re working with latent (hidden) variable models

Latent Variable Models (LVMs)

- Just like most topics in this class, there are entire courses on this
- Here we introduce just one class of LVMs
- Mixture models
  - Interesting case to highlight a very powerful model GMM
  - Also useful to illustrate an important algorithm EM

Mixture models

- A simple form of LVM
- Assume there is a latent state indicating a particular class
- Then the probability of a given observation is just a combination of how likely a particular class is and then how likely that data would be under that class.
- E.g. We have computer science students and English students. We observe their scores on math exams. A particular set of scores is a weighted sum of how likely a particular score would be for a typical engineering student and (separately) for a particular English student
Mixture models

- A simple form of LVM
- Assume there is a latent state indicating a particular class
- Then the probability of a given observation is just a combination of how likely a particular class is and then how likely that data would be under that class.

\[ p(x_i | \theta) = \sum_{k=1}^{K} \pi_k p_k(x_i | \theta) \]

Mixture of Gaussians

- Assume that each of the k classes is distributed as a MVN
- Our model is then redefined as

\[ p(x_i | \theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k) \]

The power of Gaussians

- We know that a Gaussian distribution is a maximum entropy distribution
  - This means it’s very expressive
- In fact, given a sufficiently large number of mixture components, a GMM can be used to approximate any density defined on \( \mathbb{R}^P \)

Using mixture models

- First use case: Use them to model \( p(x_i) \) directly
  - This is unsupervised and can be used for
    - Data compression
    - Outlier detection
    - Generative classifiers
  - Book says this isn’t common but I think it is very widely used nowadays
- Second use case: clustering
  - Supervised
    - Fit a model, compute the probability of a class for new data given the learned parameters
    - Can compute the “responsibility” (think contribution) of each cluster on a particular observation
Using mixture models

- Model $p(x_i)$ directly
  - We don’t know the true class assignments
- Clustering
  - We see the class labels which we assume to be the latent class assignment
    \[
    p(z_i = k|x_i, \theta) = \frac{p(z_i = k|\theta) p(x_i|z_i = k, \theta)}{\sum_{k'}^{K} p(z_i = k'|\theta) p(x_i|z_i = k', \theta)}
    \]
  - The only difference in these two cases is during training
    - Whether we know the labels or not
    - Or if we have to iteratively guess the “true” labels and update the parameters