Large Scale Data Analysis Using Deep Learning

Regularization for Deep Learning - 2

U Kang
Seoul National University

In This Lecture

- Regularization techniques
  - Parameter tying and parameter sharing
  - Sparse representations
  - Bagging
  - Dropout
  - Adversarial training
  - Tangent propagation

Parameter Tying and Parameter Sharing

- L^2 regularization is a way to express our prior knowledge that we penalize model parameters that deviate significantly from the fixed value of 0
- Sometimes, we might not know precisely what values the parameter should take, but we know, from the knowledge of the domain and model architecture, that there should be some dependencies between model parameters

Parameter Tying and Parameter Sharing

- Common dependencies
  - Certain parameters should be close to one another
    - $\Omega\left(w^{(A)}, w^{(B)}\right) = ||w^{(A)} - w^{(B)}||_2^2$ where $w^{(A)}$ and $w^{(B)}$ are parameters from different model
  - Force set of parameters to be equal = parameter sharing
    - Convolutional Neural Network (CNN)
      - Share parameters over multiple image locations
      - Parameter sharing in CNN makes it translation invariant: i.e., we can find a cat with the same cat detector whether the cat appears at column i or column i+1 in the image
      - Parameter sharing in CNN also dramatically lowers the model parameters, and significantly increases network sizes without requiring a corresponding increase in training data

Sparse Representations

- Representational regularization: place a penalty on the activations of the units in a neural network
- Representational regularization is accomplished in a similar way to the parameter regularization
  - $\tilde{J}(\Theta; X, y) = J(\Theta; X, y) + \alpha\Omega(h)$ where $\Omega(h) = ||h||_1 = \sum_i |h_i|$

$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad$ $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ $\quad$ $\boldsymbol{h} \in \mathbb{R}^n$

Bagging

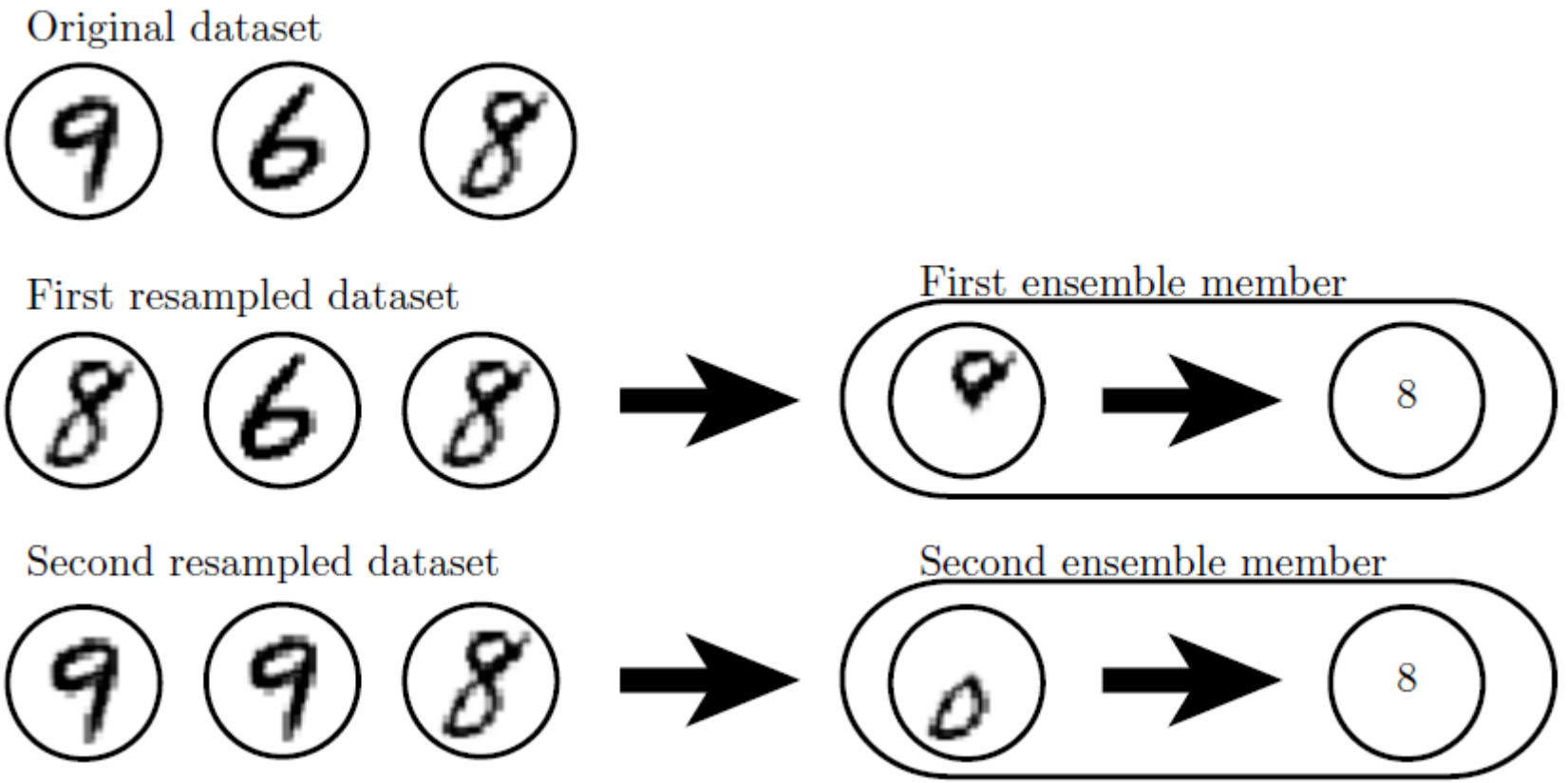
- Bagging (bootstrap aggregating) is a technique for reducing generalization error by combining several models
  - Train several different models separately, then have all of the models vote on the output for test examples
  - An example of model averaging (also called ensemble methods)
- Why does model averaging work?
  - Different models will usually not make all the same errors on the test set
  - Errors will cancel out if the members make independent errors
- Example: k regression models where i th model makes an error ϵ_i drawn from a zero-mean multivariate normal distribution with $E(\epsilon_i^2) = v$ and covariances $E(\epsilon_i \epsilon_j) = c$
  - The expected squared error of the ensemble predictor is

    $$E\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2} E\left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k} v + \frac{k-1}{k} c$$

  - If errors are perfectly correlated and c = v, the expected squared error is v (no benefit)
  - If errors are uncorrelated (i.e., c = 0), then the expected squared error is only $\frac{1}{k} v$

Bagging

- Bagging construct k different datasets; each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the original dataset
- Each dataset is missing some examples, and also contains duplicate ones



More on Model Averaging

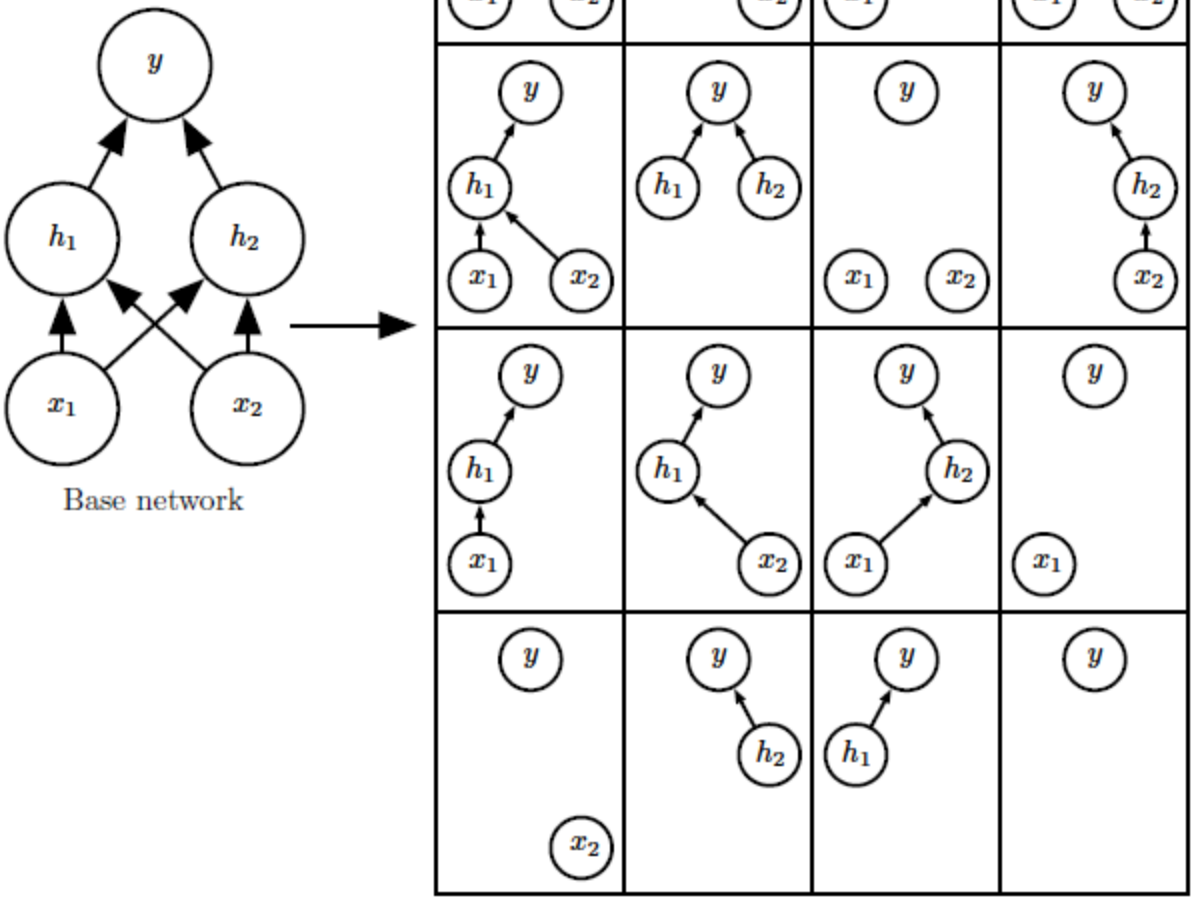
- Different ensemble methods construct the ensemble of models in different ways.
  - E.g., each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.

- Neural networks reach a wide enough variety of solution points that they can often benefit from model averaging even if all of the models are trained on the same dataset. The following models often make partially independent errors
  - Differences in random initialization
  - Random selection of minibatches,
  - Differences in hyperparameters
  - Different outcomes of non-deterministic implementations

Dropout

- Computationally inexpensive but powerful method of regularizing a broad family of models
- A method of making bagging practical for ensembles of many large neural networks
- Trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network
- In standard bagging, we train k different models from k different datasets by sampling from the training set with replacement; dropout aims to approximate this process, but with an exponentially large number of neural networks

Dropout



Dropout Training

- Procedure of training with dropout
  - Use a minibatch-based learning algorithm
  - For each minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network
  - The mask for each unit is sampled independently
  - The probability of sampling a mask value of one is a hyperparameter; typically, an input unit is included w/ prob 0.8, and a hidden unit is included w/ prob 0.5
  - We then run forward propagation, back-propagation, and the learning update as usual

Dropout Training

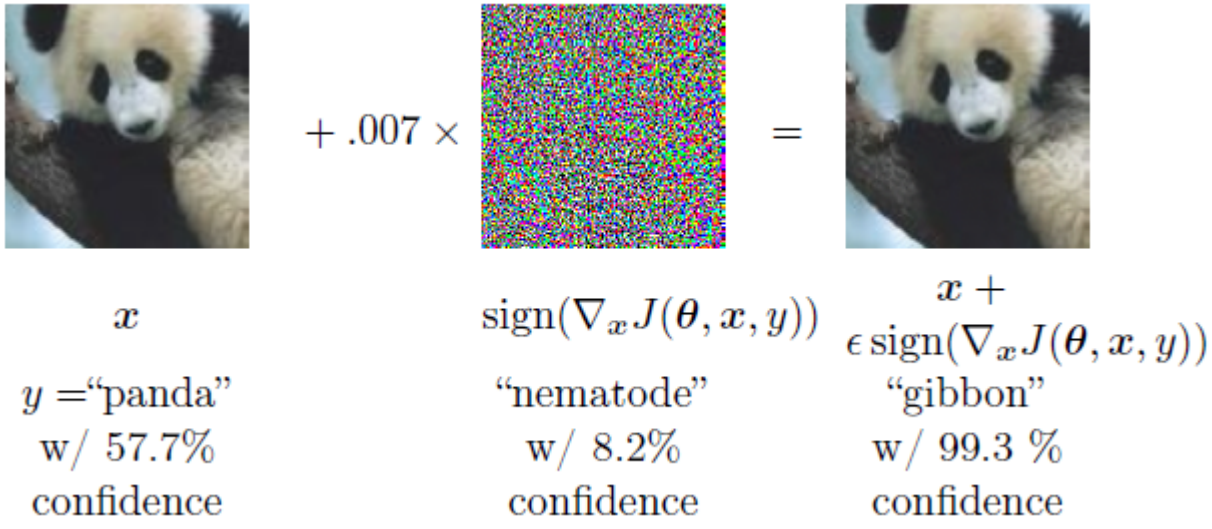
- Analysis
  - Suppose a mask vector μ specifies which units to include, and $J(\theta, \mu)$ is the cost of the model defined by θ and μ
  - Then, dropout training consists in minimizing $E_\mu J(\theta, \mu)$
  - The expectation contains exponentially many terms but we obtain an unbiased estimate of its gradient by sampling values of μ
- Model independence
  - In the case of bagging, the models are all independent
  - In the case of dropout, models share parameters
    - This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory

Inference After Dropout

- Assume that each sub-model defined by mask vector μ defines a probability distribution $p(y|x,\mu)$
- Then, the mean over all masks is given by
  - Arithmetic mean: $\sum_{\mu} p(\mu)\, p(y|x,\mu)$
  - Geometric mean: $\sqrt[2^d]{\prod_{\mu} p(y|x,\mu)}$
- Computing the mean is intractable because it includes an exponential number of terms
- However, we can approximate it by evaluating p(y|x) in one model with all units, but with the weights going out of unit *i* multiplied by the probability of including unit *i*
- This is called the *weight scaling inference rule*
  - Intuition: assume unit i is included with probability 0.5 in dropout training. In the inference stage with the one model with all units, we would output 2 times larger values if we do not adjust weights, and thus change the model trained with the dropout

Adversarial Examples

- Adversarial example
  - Assume a neural network that performs at human level accuracy
  - Given a data point x, it is possible to build x' (an adversarial example) around x such that the neural network makes nearly 100% error
  - In many cases, x' is so similar to x that a human observer cannot tell the difference between x' and x



Adversarial Training

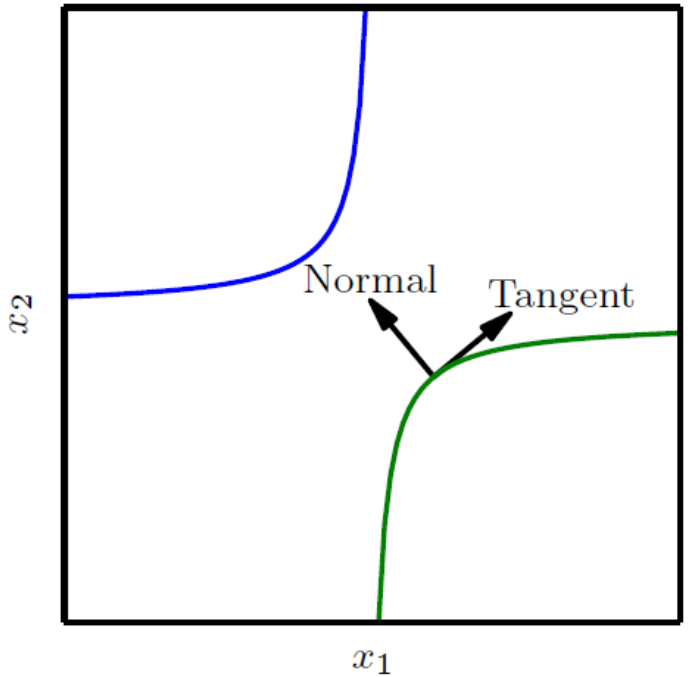
- A regularization technique is *adversarial training*: training on adversarially perturbed examples from the training set
- Neural networks are built out of linear building blocks; unfortunately, the value of a linear function can change very rapidly if it has numerous inputs
  - E.g., if each input changes by ϵ, then a linear function with weights w would change by as much as $\epsilon||w||_1$
- Adversarial training discourages this highly sensitive locally linear behavior by encouraging the network to be locally constant in the neighborhood of the training data
- This can be seen as a way of explicitly introducing a local constancy prior into supervised neural nets

Tangent Propagation

- Many machine learning algorithms aim to overcome the curse of dimensionality by assuming that the data lies near a low-dimensional manifold
- Tangent distance
  - An approach to use manifold hypothesis
  - A non-parametric nearest-neighbor algorithm in which the metric used is not the generic Euclidean distance but one derived from knowledge of the manifolds
  - Nearest-neighbor distance between points x_1 and x_2 is given by the distance between the manifolds M_1 and M_2 to which they respectively belong to
  - However, finding the nearest pair of points on M_1 and M_2 is difficult
  - A cheap alternative is to approximate M_i by its tangent plane at x_i, and measure the distance between the two tangents, or between a tangent plane and a point

Tangent Propagation

- Trains a neural net classifier with an extra penalty to make each output $f(x)$ of the neural net locally invariant to known factors of variation
- Local invariance is achieved by requiring $\nabla_x f(x)$ to be orthogonal to the known manifold tangent vectors $v^{(i)}$ at x, or equivalently that the directional derivative of f at x in the directions $v^{(i)}$ be small by adding a regularization penalty Ω
  - $\Omega(f) = \sum_i ((\nabla_x f(x))^T v^{(i)})^2$



What you need to know

- Regularization techniques
  - Parameter tying and parameter sharing
  - Sparse representations
  - Bagging
  - Dropout
  - Adversarial training
  - Tangent propagation

Questions?