Neural Network Course

John Platt

Microsoft Research

1

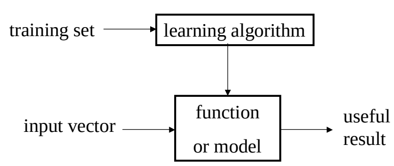
Suggested Reading

- Neural Networks for Pattern Recognition
  - by Christopher Bishop
  - This talk will refer to sections in Chris' book
- Neural Networks: Tricks of the Trade, by Orr & Müller
- http://research.microsoft.com/~jplatt/hands.ps

2

Machine Learning

- Create statistical models/functions from training data
  - These models & functions should perform well on data that is *not* in the training set (generalization)



3

Why Machine Learning?

- Learned functions > hand-designed functions
  - Accuracy
  - Speed
  - Memory

4

Typical Machine Learning Problems

- Clustering
  - Training data contains unlabeled examples
  - Map new input vector into cluster membership
- Classification
  - Training data contains examples + category labels
  - Map new input vector into category
- Regression
  - Training data contains examples + real values
  - Map new input vector into real valued number

5

Examples of Classification

- Text Categorization
  - Map e-mail message into spam or not spam
- Handwriting Recognition
  - Map handwritten glyph into character code (esp. Chinese!)
- Speech Recognition
  - Map sequence of sounds into words
- Optical Character Recognition
  - Map pixels into character code
- Etc.

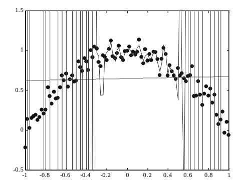
6

Why Machine Learning is Hard

- Overfitting
  - Try and infer general functions from limited data sets
  - You can never put too much data into a learning algorithm
- Curse of Dimensionality
  - Input vector spaces tend to be high dimensional
  - Fitting functions on high-dimensional spaces can take exponential number of parameters
    - neural networks avoid this (in certain circumstances)

Overfitting

- Learning algorithms can find structure that isn't really there: finite data size effects
- Example: regression (fitting a function)



Two-Class Classification

- Make a decision
  - which class? $c = 0$ or 1
  - based on continuous input vector = $\boldsymbol{x}$
  - cost of decision: pay $1 if wrong, nothing if correct
- Decision function:
  - $f(\boldsymbol{x}) = 0$ or 1
  - expected cost = $P(c = 1 | \boldsymbol{x})(1-f) + P(c = 0 | \boldsymbol{x}) f$
- Optimal decision
  - $f(\boldsymbol{x}) = P(c = 1 | \boldsymbol{x}) > P(c = 0 | \boldsymbol{x}) = P(c = 1 | \boldsymbol{x}) > 0.5$

Multi-Class Classification

$f(\boldsymbol{x}) = j$ if $P(c = j | \boldsymbol{x}) > P(c = k | \boldsymbol{x})$ for $k \neq j$

- Other loss functions also possible
  - See Bishop, section 1.10

Three Types of Classification Learning

1. Learn discrete function f from training set
   - Discriminant function
     Nearest Neighbor Classifier, Perceptron, Support Vector Machine
2. Learn $P(c = j | \boldsymbol{x})$ from training set
   - Ranking alternatives
   - Further post-processing (e.g., word models)
3. Infer $P(c = j | \boldsymbol{x})$ using Bayes' rule
   - Create *density model* of each class
   - Infer missing data
   - Perform temporal inference

Probability Density

- A probability density over a continuous variable

  $p(x)$

  - differential probability over a infintesimal region

$$P(x \in [a,b]) = \int_a^b p(x)dx$$

Bayes' Rule

- Two ways of expressing joint probability

$$p(\boldsymbol{x},c) = P(c \mid \boldsymbol{x})\, p(\boldsymbol{x})$$
$$p(\boldsymbol{x},c) = p(\boldsymbol{x} \mid c)\, P(c)$$

class density model

prior

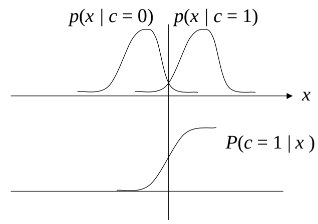
$$P(c \mid x) = \frac{p(x \mid c)P(c)}{p(x)}$$

posterior

13

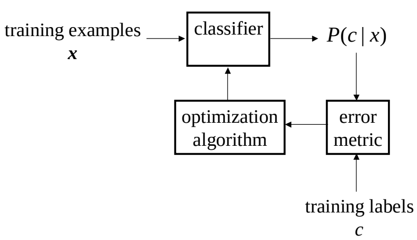
Fit a Density Model

- Fit a conditional density model to each class
  - derive posterior using weighted ratio of densities



14

Fit Posterior Directly



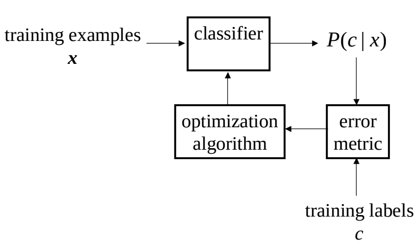
15

Posterior Fit vs. Bayes Rule

- Fitting posterior often produces more accurate results
  - More robust to incorrect models
  - Maximize metric related to classification performance, rather than density model fit
- Using Bayes rule has extra benefits
  - Infer missing data
  - Perform temporal inference
  - Easy incorporation of prior knowledge (Bayes nets)

16

Defining an Error Function



17

Some Notation

Classifier has vector of parameters $\boldsymbol{\theta}$

nth training example is $\{\boldsymbol{x}_n, c_n\}$

Output of classifier is real value y (y_n for nth example)

Probability density of nth training example according to model

$$p(\boldsymbol{x}_n, c_n \mid \boldsymbol{\theta})$$

18

Maximum Likelihood Learning

- Assume training set is drawn independently from same distribution
- Find θ so that training set is most likely
  - training set is "true", so it should be likely under model!

$$\max_\theta L(\theta) = \max_\theta \prod_n p(x_n, c_n \mid \theta)$$
$$= \max_\theta \prod_n P(c_n \mid x_n, \theta)\, p(x_n)$$

Minimize Negative Log Likelihood

Maximum of L occurs in same place as max log(L)

- monotonicity of log

max log(L) = min –log(L)

- Best parameter value occurs at

$$\min_\theta \quad -\sum_n \log P(c_n \mid x_n, \theta)$$

Labels Generated by Bernoulli Distribution

- Consider two-class classification (c = 0 or 1)
- Model: output of classifier controls a coin that flips to determine class

$$P(c_n \mid x_n, \theta) = y_n^{c_n}(1 - y_n)^{1-c_n}$$
$$P(0 \mid x_n, \theta) = 1 - y_n$$
$$P(1 \mid x_n, \theta) = y_n$$

Cross-Entropy Error Function

- Our error metric will be

$$\min_\theta E = \min_\theta -\sum_n \log P(c_n \mid x_n, \theta)$$
$$= \min_\theta -\sum_n c_n \log y_n + (1 - c_n)\log(1 - y_n)$$

Outputs will be Probabilities

- Cross-entropy is a *proper* error score
  - For infinite training set
  - If output of classifier can represent true probability
  - Minimum of score occurs when

$$y(x) = P(c = 1 \mid x)$$

- See Bishop, section 6.7
  - requires Calculus of Variation

Multi-Class Classification

- N possible disjoint classes
  - Example: digit recognition --- a digit is one class from 0-9
- Extra notation

  t_{in} := (c_n = i) is the ith target for the nth training example

  y_{in} is the ith classifier output for the nth training example
- Multinomial model
  - Classifier produces N probabilities (which sum to 1)
  - Model: roll an N-way die with those probabilities

Multinomial model

$$P(c_n \mid x_n, \theta) = \prod_{i=1}^{N} (y_{in})^{t_{in}}$$

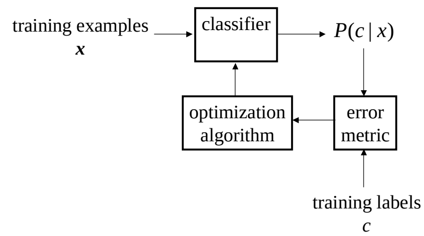
- Maximum likelihood solution (error metric)

$$\min_\theta E = \min_\theta -\sum_n \sum_{i=1}^{N} t_{in} \log y_{in}$$

also a proper scoring rule

25

Defining an Optimization Algorithm



26

Stochastic Gradient Descent (SGD)

- Simplest possible optimization algorithm
- Loop over each training example
  - adjust parameters to improve error metric a little bit
- Only evaluate error on one training example

$$E_n = -\sum_{i=1}^{N} t_{in} \log y_{in}$$

- Gradient $\frac{\partial E_n}{\partial \theta_i}$ points towards worst possible step
- Negative gradient $-\frac{\partial E_n}{\partial \theta_i}$ points to best possible step

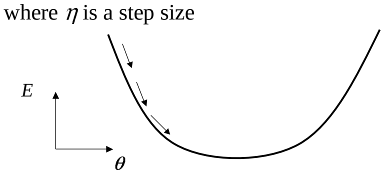
27

More about SGD

- At every step:

$$\Delta\theta_i = -\eta \frac{\partial E_n}{\partial \theta_i}$$

where η is a step size



28

Convergence of SGD

- Except E_n is only an approximation to E

$$\langle E_n \rangle = \frac{1}{\ell} E$$

- Stochastic approximation
  - or Robbins-Munro procedure
- Converges if step size η decreases with time

$$\sum_{t=0}^{\infty} \eta(t) = \infty \qquad \sum_{t=0}^{\infty} \eta^2(t) < \infty$$

- In practice, people use fixed step size

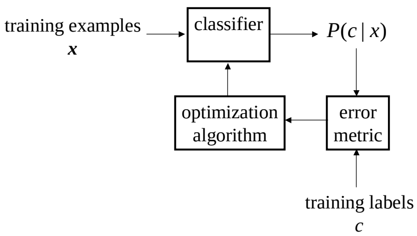
29

On-line versus Batch Learning

- What about sophisticated optimization algorithms?
  - Conjugate gradient
  - Quasi-Newton algorithm
- These algorithms require full gradient E for step
  - SGD often converges in small number of “epochs”
  - SGD much faster for neural nets & large data sets
- Noisy gradient in SGD avoids local minima
  - Non-convex optimization (in neural nets)
  - Noise acts as “annealing” to avoid poor local minima

30

Defining a Classifier



Logistic Model

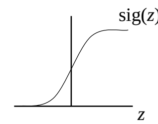
- Two-class classification
- Model: log-odds of $P(c \mid \boldsymbol{x})$ depends linearly on $\boldsymbol{x}$

$$\log\left(\frac{P(c=0 \mid x)}{P(c=1 \mid x)}\right) = \sum_{i=1}^{d} w_i x_i + w_0 \qquad \theta$$

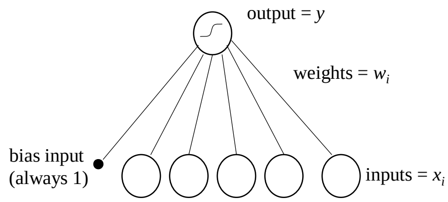
  - log odds a natural way of expressing probability

$$y = \mathrm{sig}\left(\sum_i w_i x_i + w_0\right)$$

$$\mathrm{sig}(z) = \frac{1}{1+e^{-z}}$$



Network Diagram for Logistic Model

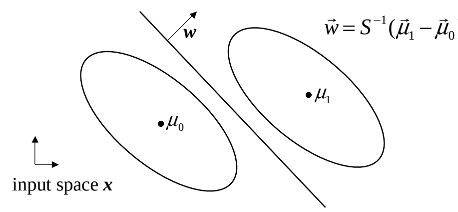


- Single-layer neural network
  - single layer of trainable weights

Relationship to Gaussian Densities

- If each class density model is Gaussian
  - with same covariance matrix S
- Then, Bayes' rule will yield a logistic classifier
  - linear discriminant analysis

$$\vec{w} = S^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$$



SGD applied to Logistic Model

Recall

$$E_n = -c_n \log y - (1-c_n)\log(1-y)$$

$$y = \frac{1}{1+e^{-z}}$$

$$z = \sum_i x_i w_i + b$$

We need to compute $\frac{\partial E_n}{\partial w_i}$ and $\frac{\partial E_n}{\partial b}$

Chain Rule

$$\frac{\partial E_n}{\partial w_i} = \frac{\partial E_n}{\partial z}\frac{\partial z}{\partial w_i} \qquad \frac{\partial E_n}{\partial z} = \frac{\partial E_n}{\partial y}\frac{dy}{dz}$$

$$\frac{\partial E_n}{\partial y} = -\frac{c_n}{y} + \frac{1-c_n}{1-y} \qquad \frac{dy}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = y(1-y)$$

$$\frac{\partial E_n}{\partial z} = y - c_n$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial w_0} = 1$$

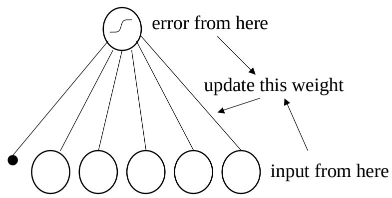
SGD + Cross-Entropy = Simple

- SGD uses very simple rules:
  - First, evaluate the network on a training example to get *y*
  - Then, update using

$$\Delta w_i = \eta(c_n - y)x_i$$
$$\Delta w_0 = \eta(c_n - y)$$

37

Local Learning Rule



38

Single-Layer Softmax

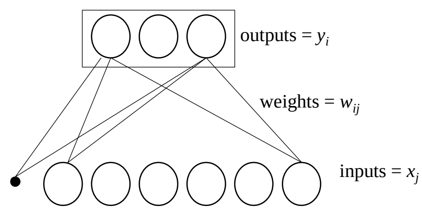
- For multi-class classification
- Need an output non-linearity
  - produces values
  - values should sum to 1
- *Softmax* is analogous to sigmoid

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \qquad z_i = \sum_j w_{ij} x_j + w_{i0}$$

- Can arise from a density model (Bishop, section 6.9)

39

Network Diagram for Single-Layer Softmax



40

SGD applied to Single-Layer Softmax

- Use multinomial error

$$E_n = -\sum_{i=1}^{N} t_{in} \log y_i$$

- SGD still has simple local form!

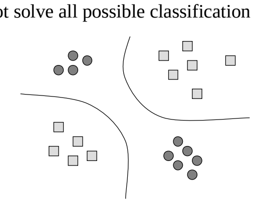
$$\Delta w_{ij} = \eta(t_{in} - y_i)x_j \qquad \Delta w_{i0} = \eta(t_{in} - y_i)$$

  - for derivation, see Bishop, section 6.9

41

Limitations of One-layer Models

- Discriminant computed by logistic classifier is linear
  - cannot solve all possible classification problems



42

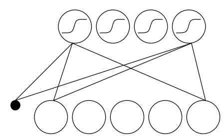
Pre-Process the Problem Away?

- Compute features, then compute linear function of features
  $$z = \sum_j w_j \varphi_j(\vec{x})$$
  – ϕ are called basis functions
- If you choose ϕ without looking at data (or problem)
  – either require an exponential number of ϕ (in input dim)
  – or cannot solve all interesting problems
- Bishop, section 3.5.4

43

Neural Network First Layer

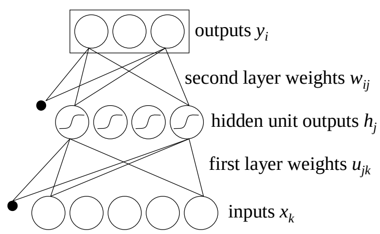
- Compute basis functions adaptively
  – reduce curse of dimensionality
- First, compute a set of non-disjoint binary features
  – probability of each binary feature is basis function



44

Neural Network Model

- Then, combine features in second layer



45

Evaluation of NN Model

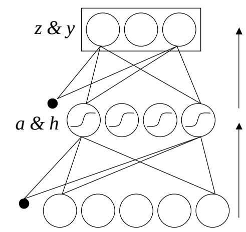
- Propagate values forward
  – load ***x*** with input vector

$$a_j = \sum_{k=1} u_{jk} x_k + u_{j0}$$

$$h_j = \frac{1}{1+e^{-a_j}}$$

$$z_i = \sum_{j=1} w_{ij} h_j + w_{i0}$$

$$y_i = \frac{e^{z_i}}{\sum_i e^{z_i}}$$



46

SGD applied to Neural Network

- Step for Δw_{ij} and Δw_{i0} the same as one-layer case

$$\Delta u_{jk} = -\eta \frac{\partial E_n}{\partial u_{jk}} \qquad \frac{\partial E_n}{\partial u_{jk}} = \frac{\partial E_n}{\partial h_j} \frac{\partial h_j}{\partial u_{jk}}$$

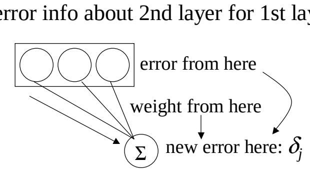
$$\frac{\partial E_n}{\partial h_j} = \sum_i \frac{\partial E_n}{\partial z_i} \frac{\partial z_i}{\partial h_j} \qquad \frac{\partial z_i}{\partial h_j} = w_{ij}$$

$$\frac{\partial E_n}{\partial h_j} = \sum_i (y_i - t_{in}) w_{ij} = \delta_j$$

47

Back-Propagation

δ_j contains all error info about 2nd layer for 1st layer



- Gradient information flows backwards through network
- All information is local

48

First Layer SGD Step

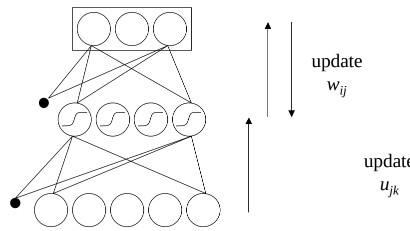
$$\frac{\partial h_j}{\partial u_{jk}} = \frac{dh_j}{da_j}\frac{\partial a_j}{\partial u_{jk}} = h_j(1-h_j)x_k$$

input *k*

$$\Delta u_{jk} = -\eta\frac{\partial E_n}{\partial u_{jk}} = -\eta\delta_j \underbrace{h_j(1-h_j)}_{\text{error at hidden unit } j} x_k$$

49

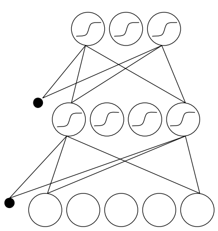
Overall Computational Flow



50

Variation I

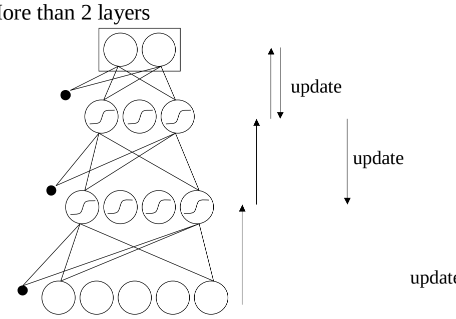
- Non-disjoint outputs



51

Variation II

- More than 2 layers



52

Summary So Far

- Classifiers take input vector, estimate posteriors
- Posteriors can fit directly from training data
- Cross-entropy is a sensible error metric
- Stochastic gradient descent is a good algorithm
- Two-layer neural networks are sensible
  - trained via on-line back-propagation

53

Why NN Model?

- Posterior corresponding to Bayes' rule applied to a density model
  - see Bishop, section 6.7.1
- Can approximate any function
- Reduces curse of dimensionality
- Works well on many problems

54

Universality of NN Function

- Two-layer NN can approximate any function
  - linear output units (no sigmoid or softmax)
  - sufficiently large number of hidden units
- NN can approximate any non-linear discriminant
  - sigmoid output units
  - sufficiently large number of hidden units
  - arbitrary accuracy

55

NN and Curse of Dimensionality

- For regression: neural net function *y* tries to match target function *f*
- Measure error via integrated squared error
- If $C = \int |\vec{\omega}| |\tilde{y}(\vec{\omega})| d\vec{\omega} < \infty$ N = number of hidden units

$$\int | y - f |^2 \, d\vec{x} = \frac{C}{N}$$

this term *independent* of input dimensionality

- Fixed basis functions have denominator $N^{d/2}$

56

Tricks of the Trade

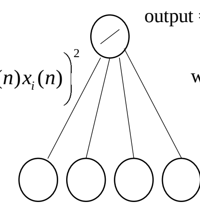
- Pre-processing of inputs
- Choice of non-linearity
- Weight initialization
- Prevention of Overfitting

57

Single Layer Linear Regression

$$y = \sum_i w_i x_i$$

$$E(n) = \frac{1}{2}\left(t(n) - \sum_i w_i(n) x_i(n) \right)^2$$



$$w_i(n+1) = w_i(n) + \eta \left(t(n) - \sum_j w_j(n) x_j(n) \right) x_i(n)$$

58

Stability Analysis

- Assume training data is drawn independently

$$\langle w_i(n+1) \rangle = \langle w_i(n) \rangle + \eta \left(\gamma_i - \sum_j R_{ij} \langle w_j(n) \rangle \right)$$

$$\gamma_i = \langle x_i t \rangle$$

$$R_{ij} = \langle x_i x_j \rangle$$

$$\langle \vec{w}(n+1) \rangle = (\mathbf{I} - \eta \mathbf{R}) \langle \vec{w}(n) \rangle + \eta \vec{\gamma}$$

$$\vec{w}_{\text{opt}} = \mathbf{R}^{-1} \vec{\gamma}$$

59

Decay of the Error Eigenvectors

$$\langle \vec{w}(n+1) - \vec{w}_{\text{opt}} \rangle = (\mathbf{I} - \eta \mathbf{R}) \langle \vec{w}(n) \rangle - \vec{w}_{\text{opt}} + \eta \vec{\gamma}$$

$$= (\mathbf{I} - \eta \mathbf{R}) \langle \vec{w}(n) \rangle - (\mathbf{I} - \eta \mathbf{R}) \vec{w}_{\text{opt}}$$

$$\langle \vec{\varepsilon}(n+1) \rangle = (\mathbf{I} - \eta \mathbf{R}) \langle \vec{\varepsilon}(n) \rangle$$

where $\langle \vec{\varepsilon}(n) \rangle$ is the expected error at example *n*

Decompose **R** into eigenvectors: $\mathbf{R} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$

$$\vec{V}_k^T \langle \vec{\varepsilon}(n+1) \rangle = \left(\vec{V}_k^T - \eta \lambda_k \vec{V}_k^T \right) \langle \vec{\varepsilon}(n) \rangle$$

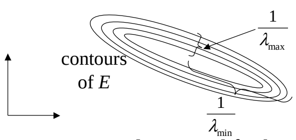
$$= (1 - \eta \lambda_k) \vec{V}_k^T \langle \vec{\varepsilon}(n) \rangle$$

60

Maximum Step Size

SGD converges if $|1-\eta\lambda_i|<1 \quad \Rightarrow \quad \eta < \frac{2}{\lambda_{\max}}$

where λ_i= eigenvalue of **R**.



- minimize eigenvalue spread for best performance
- speed of convergence depends on $\lambda_{\min} / \lambda_{\max}$

61

Zero Mean Inputs are Good

- Decompose input to net into zero mean x_i and mean μ

$$\mathbf{R} = \left\langle (x_i+\mu)(x_j+\mu) \right\rangle = \left\langle x_i x_j \right\rangle + \mu^2 \mathbf{1}$$

- Eigenvector of all ones has eigenvalue $\mu^2 N$
  - which is HUGE
- Make sure inputs have close to zero mean
  - subtract off mean input before applying to network

62

Sparse Inputs are Good

- Represent categorical inputs to net as *1* of *N* code
- Example: one input attains discete letter values A-G
  - Convert into 7 different inputs

```
0   0   0   0   1   0   0
A   B   C   D   E   F   G
```

- Good eigenvalue spread
- Sparse computation of $\sum_i w_i x_i$ is fast
- Can also quantize continuous variables

63

Unit Variance Inputs are Good

- Make sure inputs to net have roughly same scale (1)
  - improves eigenvalue spread

64

Uncorrelated Inputs are Good

- Principal Component Analysis before input to net
  - perform Singular Value Decomposition on training data
    - see Numerical Recipes, chapter 2
  - Rotate and scale inputs so that **R** is the identity matrix
  - Eigenvalue spread: 0!
  - Does not affect computational ability

65

Reduce Number of Inputs through PCA

- PCA computes eigenvalues and eigenvectors of **R**
- Remember: $\vec{w}_{\text{opt}} = \mathbf{R}^{-1}\vec{\gamma}$
- γ is estimated from data
- Small eigenvalues in **R** lead to big eigenvalues in $\mathbf{R}^{-1}$
  - weight vector will blow up: overfitting!
- Discard PCA input dimensions with small eigenvalue
  - Reduce size of network: reduces overfitting

66

Tanh Hidden Units are Good

- What about correlations between hidden units?
  - Want zero mean for hidden units
- For value of z close to zero: $\langle h \rangle = \left\langle \frac{1}{1+e^{-z}} \right\rangle = 0.5$
- Use tanh(z) rather than sig(z):

$$\tanh(z) = \frac{2}{1+e^{-2z}} - 1 = 2\,\mathrm{sig}(2z) - 1$$

- No change in computational ability
  - just scale in weights and shift in bias

67

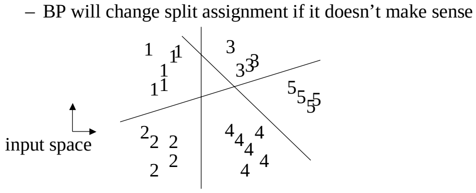
Weight Initialization

- If you start all weights = 0, then gradient = 0 ☹
  - If h = 0 and w = 0, then Δw = 0 and δ = 0
- Need to initialize some weights to Gaussian random non-zero values
  - sensitive to initial conditions ☹
- Initialize all but first-layer weights to non-zero
  - Causes δ to start out with slight non-zero values
- Weight should be small enough to keep network in linear regime

68

Effect of Weight Initialization

- Find splits of data into randomly assigned halves
  - BP will change split assignment if it doesn't make sense

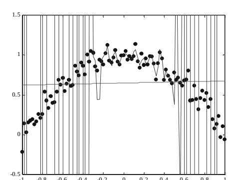


- First layer weights will not span null space of **R**

69

Overfitting

- Neural network can fit an overly complicated function to the training data



70

MAP Learning

- Find parameters θ that are most likely given data D
  - different from parameters that make data most likely

$$\max_\theta P(\theta \mid D) = \max_\theta \frac{P(D \mid \theta) \overbrace{P(\theta)}^{\text{prior over parameters}}}{P(D)}$$

$$= \max_\theta \quad \log P(D \mid \theta) + \log P(\theta)$$

$$= \min_\theta \quad \underbrace{-\log P(D \mid \theta)}_{\text{usual data likelihood term}} \underbrace{-\log P(\theta)}_{\text{regularizer}}$$

71

Regularization

- Regularizer penalizes models that
  - have unusual parameter settings
  - are too wiggly
  - are too complex

72

Weight Decay

- Prior: weights should not be insanely large
  - weights are derivatives inside neural network

$$P(\theta) = \exp\left(-\frac{\alpha}{2}\sum_i \theta_i^2\right)$$

$$-\log P(\theta) = \frac{\alpha}{2}\sum_i \theta_i^2 = g(\theta)$$

penalize large weights quadratically

$$\Delta\theta_i = -\eta\left(\frac{\partial E_n}{\partial \theta_i} + \frac{\partial g}{\partial \theta_i}\right)$$

$$= -\eta\frac{\partial E_n}{\partial \theta_i} - \eta\alpha\theta_i$$

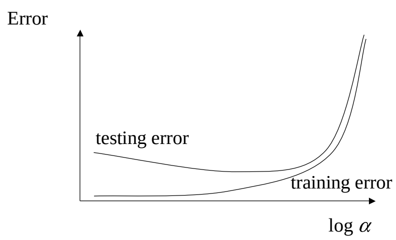
73

Hold Out Set

- How do you set α?
- Simplest method: hold out set
- Split training set into train (70%) and hold out (30%)
- For a series of different α:
  - Train NN on training subset
  - Test NN on hold out set
- Choose α which maximizes performance on hold out
- Retrain NN on entire training set with optimal α

74

Performance Curve



75

Better Weight Decay

- Want to fit a scaled or shifted function if you scale or shift inputs or outputs of net

$$\alpha_1 \sum_{j,k} u_{jk}^2 + \alpha_2 \sum_{i,j} w_{ij}^2$$

  - and do not put prior on biases
- See Bishop, section 9.2.2 for derivation

76

Analysis of Weight Decay

SGD applied to linear regression:

$$w_i(n+1) = w_i(n) + \eta\left(t(n) - \sum_j w_j(n)x_j(n)\right)x_i(n) - \eta\alpha w_i(n)$$

$$\langle w_i(n+1)\rangle = (1-\eta\alpha)\langle w_i(n)\rangle + \eta\left(\gamma_i - \sum_j R_{ij}\langle w_j(n)\rangle\right)$$

$$\vec{w}_{MAP} = (\mathbf{R} + \alpha\mathbf{I})^{-1}\vec{\gamma}$$

77

Adding Noise to Inputs

- At every step of SGD, add random noise to inputs:

$$w_i(n+1) = w_i(n) + \eta\left(t - \sum_j w_j(n)(x_j + \xi_j)\right)(x_i + \xi_i)$$

$$\langle w_i(n+1)\rangle = (1-\eta\alpha)\langle w_i(n)\rangle + \eta\left(\gamma_i - \sum_j (R_{ij} + \sigma^2\delta_{ij})\langle w_j(n)\rangle\right)$$

$$\vec{w}_{\text{noise}} = (\mathbf{R} + \sigma^2\mathbf{I})^{-1}\vec{\gamma}$$

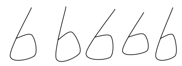
Applicable to any SGD, not just linear regression

- noise variance is much more intuitive than α

78

Domain-Specific Noise is Good

- At SGD step, add domain-specific noise to input:

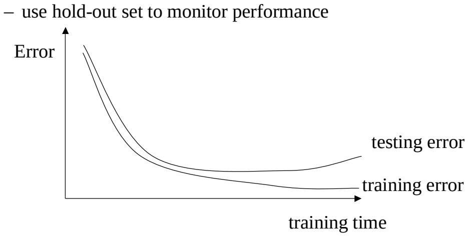


- Makes network robust to likely distortions of input
- This helps *a lot*
- Can be done analytically: tangent-prop by Simard
  - Bishop, section 8.7.1

79

Early Stopping is Good

- When do you stop back-propagation?
  - use hold-out set to monitor performance



80

Early Stopping Specifics

- Save net which yields best hold out performance
- If hold out performance does not improve in N epochs
  - stop

- Note: only need to perform learning once, not multiple times!

81

Why Early Stopping Works

- Early stopping is similar to weight decay
  - see Bishop, exercise 9.1

$$E(\vec{w}) = E_0 + \vec{b}^T\vec{w} + \frac{1}{2}\vec{w}^T\mathbf{H}\vec{w} + \frac{1}{2}\alpha\vec{w}^T\vec{w}\ldots$$

$$\vec{w}_{MAP} = (\mathbf{H} + \alpha\mathbf{I})^{-1}\vec{b}^T$$

- Weight decay softly erases small eigenvalues in **H**
- So does early stopping
  - large eigenvalues decay quickly
  - small eigenvalues decay slowly

82

Convolutional Neural Networks

- Using raw signals as inputs yields huge networks
  - one second of sound = 16000 inputs
  - one video frame = 640*480 inputs
- Also, you want to “spot” objects in signals
  - translation invariance
- Solution: convolutional neural networks

83

Use Convolutions

- Instead of dot product followed by sigmoid (or tanh):

$$y = \frac{1}{1+e^{-z}}$$

$$z = \sum_{k=1} w_k x_k + w_0$$

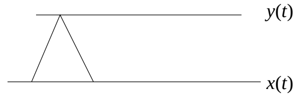
- Use convolution to get shift-invariant output:

$$y(t) = \frac{1}{1+e^{-z(t)}}$$

$$z(t) = \sum_{k=0} w_k x(t-k) + b$$

84

Network Diagram for Convolution

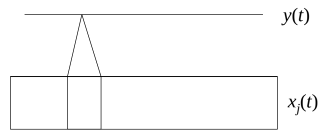


85

Multiple Inputs

- Convolutional Layer can take multiple inputs

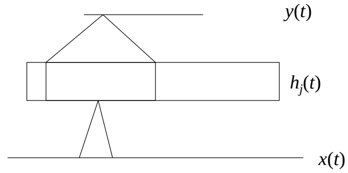
$$z(t) = \sum_{j=1}^{M} \sum_{k=0}^{N} w_{jk} x_j (t-k) + w_0$$



86

Time-Delay Neural Network (TDNN)

- Stack multiple convolutional layers to perform non-linear filtering



87

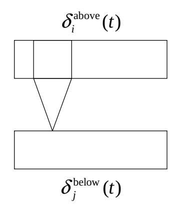
Backprop for Convolutional Layer

- Analogous to standard case, with more indices

$$z_i(t) = \sum_{j,k} w_{ijk} x_k (t-j)$$

$$\Delta w_{ijk} = -\eta \sum_t \delta_i^{\text{above}}(t) x_j (t-k)$$

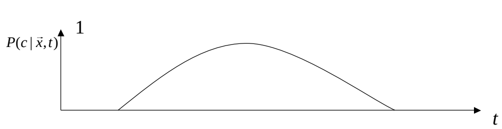
$$\delta_j^{\text{below}}(t) = \sum_{i,k} \delta_i^{\text{above}}(t+k) w_{ijk}$$



88

Targets for CNNs

- Your belief that an object in at that time/position

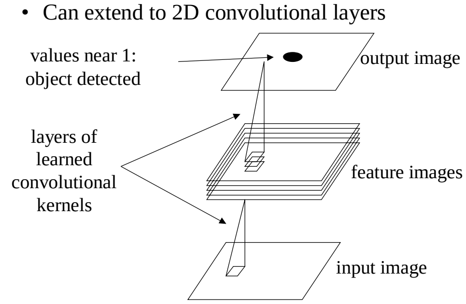


- More sophisticated target generation:
  - “Soft-OR” or “Integrated Segmentation and Recognition”
  - Bootstrap generation of targets
  - Keeler and Rumelhart, NIPS 3

89

2D Convolutional Layers

- Can extend to 2D convolutional layers



90

Examples

- Find a hand in an image using CNNs
  - http://research.microsoft.com/~jplatt/hands.ps
- Recognize handwritten optical digits
  - http://www.research.att.com/~yann/publis/psgz/lecun-bengio-94.ps.gz
- Recognize handwritten characters from tablet
  - Chap 13 in "Neural Networks: Tricks of the Trade"
- Combine HMMs and Neural Networks
  - ftp://ftp.dcs.shef.ac.uk/share/spandh/pubs/renals/icassp92.ps.gz

91

Summary of Second Half

- Analysis of SGD linear regression yields many tricks
- Regularization is important
  - early stopping
  - domain-specific noise
- Convolutional Neural Networks
  - useful for recognizing signals

92