CPSC 440/540: Machine Learning

Generative and Discriminative Classifiers, Neural Nets

Winter 2023

admin

Admin

- A1 due Friday
  - I'll post notice of updates as followups in the pinned Piazza post: was one earlier today (typo in softmax gradient for Q7)
- First tutorial today, 5-6pm, DMP 201
  - Also Wednesday, 5-6pm, DMP 101
  - Also Friday, 4-5pm, DMP 101 (after A1 deadline)
  - All three will *usually* be same content + same TA (Justin)
  - Sometimes we'll do "bonus" tutorials that might differ, will announce those
- Office hours schedule to be announced tonight
- Auditors: I think I've signed all forms I've gotten
  - Email me again, or bring me a paper form after class Wednesday

admin

CPSC 320 Prereq

- To grad students who haven't taken CPSC 320, what do you *actually* need to know from it?
  - Dynamic programming
  - Working with graphs (mathematically + as data structures)
  - Being very comfortable with big-O notation

- If you haven't it or a course like it, strongly consider taking it
  - Mark says people who he bugged to take it for background for this course "later thanked me for bugging them to take it"

admin

Very briefly

- Obviously not a Canadian holiday,
  but want to acknowledge Martin Luther King, Jr day
- Letter from a Birmingham Jail (and other writings/speeches)
  still extremely relevant today, in Canada and around the world

Last Time: Product of Bernoullis

- We discussed multivariate binary density estimation:
  - Input: 'n' IID samples of binary vectors $x^1, x^2, x^3, \ldots, x^n$ from population.
  - Output: model giving probability for any assignment of values $x_1, x_2, \ldots, x_d$.

X =

Inter 1	Inter 2	Inter 3	Inter 4	Inter 5	Inter 6	Inter 7	Inter 8	Inter 9
0	1	0	1	1	1	0	0	1
0	1	0	1	1	1	0	0	1
0	0	1	1	0	0	0	0	0
0	1	0	1	1	1	0	0	0

$\Rightarrow$ $\Pr(X_1 = 0, X_2 = 1, X_3 = 0, X_4 = 1, X_5 = 1, X_6 = 1, X_7 = 0, X_8 = 0, X_9 = 1) = 0.11$

(Estimates probability for all 2^9 values)

- We discussed the product of Bernoullis model:
  - Assumes x_j are mutually independent (strong assumption, easy computation).

$$p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2)\cdots p(x_d) = \theta_1^{x_1}(1-\theta_1)^{1-x_1}\theta_2^{x_2}(1-\theta_2)^{1-x_2}\cdots\theta_d^{x_d}(1-\theta_d)^{1-x_d}$$

- We started discussing generative classifiers:
  - Supervised learning methods that model $p(x_1, x_2, \ldots, x_d, y)$.
    - Compute $p(y \mid x_1, x_2, \ldots, x_d)$ to make predictions.

Naïve Bayes Generative Classifier

- Naïve Bayes: generative classifier, used for spam detection in 90s.
- Naïve Bayes Assumes features x_j are mutually independent given y:
  - $p(x_1, x_2, \ldots, x_d \mid y) = p(x_1 \mid y)\, p(x_2 \mid y) \cdots p(x_d \mid y)$.
    - Unlike product of Bernoullis where we all variables are mutually independent.
  - "We assume the features are independent within each class."
    - Another view: we use a <u>different</u> product of Bernoullis for each class.
- How it this used within a generative classifier?

$$p(x_1, x_2, \ldots, x_d, y) = p(x_1, x_2, \ldots, x_d \mid y)\, p(y) \quad \text{(product rule)}$$

$$= p(x_1 \mid y)\, p(x_2 \mid y) \cdots p(x_d \mid y)\, p(y) \quad \text{(under naive Bayes assumption)}$$

conditional univariate density estimation

figuring this out is a <u>univariate</u> density estimation problem.

Naïve Bayes Generative Classifier

- Naïve Bayes inference:
  - We have that $p(x_1, x_2, \ldots, x_d, y) = p(x_1 \mid y)\, p(x_2 \mid y) \cdots p(x_d \mid y)\, p(y)$.
  - Use $p(y \mid x_1, x_2, \ldots, x_d) \propto p(x_1, x_2, \ldots, x_d, y)$ (definition of conditional prob), to determine if $p(y = 1 \mid x_1, x_2, \ldots, x_d) > p(y = 0 \mid x_1, x_2, \ldots, x_d)$.
  - You could also do other inference tasks:
    - Normalization:
      - Sum up $p(x_1, x_2, \ldots, x_d, y)$ for $y=1$ and $y=0$ to get $p(x_1, x_2, \ldots, x_d)$ by the marginalzation rule.
    - Conditional mode decoding:
      - Find "most spammy" features possible: $\text{argmax}_{x1,\ldots,xd}\, p(x_1, \ldots, x_d \mid y = 1)$.
    - Find fewest words to add to your spam message that make it appear as non-spam.

Conditional Binary Density Estimation

- To train naïve Bayes, we want to build a model of $p(x_j \mid y)$.
  - "Probability of this x_j, given the class label y".
- For binary x_j and y, can parameterize as conditionally Bernoulli:

$$p(x_j = 1 \mid y = 1) = \theta_{j1}$$
$$p(x_j = 1 \mid y = 0) = \theta_{j0}$$

You can get the other probabilities from "sum to 1": $p(x_j = 0 \mid y = 1) = 1 - \theta_{j1}$

  - This has two parameters for each feature *j*:
    - θ_{jk}: probability of X_j being 1 when in class *k*.
  - Given the *y* value, this is a Bernoulli distribution.
    - Value of *y* causes you to "pick" between the two Bernoulli distributions.
    - With a fixed *y*, inference will work as it did for Bernoullis.
  - MLE is given by (exercise):

$$\hat{\theta}_{j1} = \frac{n_{j=1,1}}{n_1}$$

← number of times $x_j = 1$ and $y = 1$

← number of times $y = 1$

$$\hat{\theta}_{j0} = \frac{n_{j=1,0}}{n_0}$$

← number of times $x_j = 1$ and $y = 0$

← number of times $y = 0$

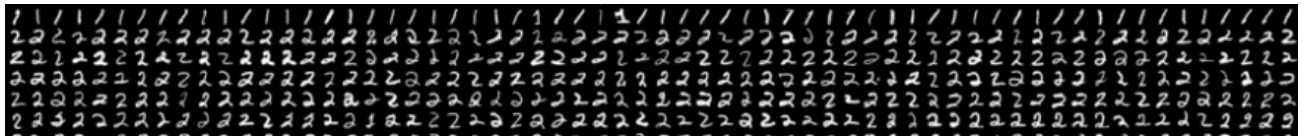
Generative Classifier: Implementation

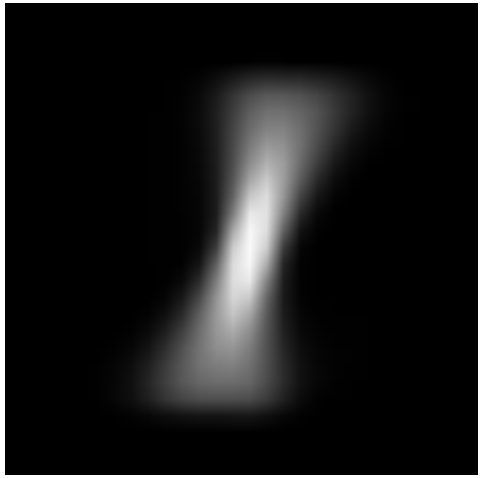
- Training phase for a generative classifier:
  1. Fit parameters of p(y).
     - For binary *y*, use Bernoulli and do MLE/MAP.
  2. For each class *k*:
     - Fit parameters of $p(x_1, x_2, \ldots, x_d \mid y = k)$ using examples in class *k*.
       - For naïve Bayes, fit $p(x_1 \mid y = k)$, then fit $p(x_2 \mid y = k)$,…, and finally fit $p(x_d \mid y = k)$.
         - » Can view as fitting a product of Bernoullis model for each class.
- Cost for naïve Bayes is O(nd):
  - O(n) to fit p(y), O(n) to fit each of the *d* parameters of p(x | y = k).
  - Can be reduced to O(z) if **X** only has *z* non-zeroes.

```
O(n)  for i in 1:n
        if y[i]==1
          p-y += 1
        end
      end
O(nd) for j in 1:d
        for i in 1:n
          if y[i]==1 & X[i,j]==1
            p-xy[j,1] += 1
          elseif y[i]==0 & X[i,j]==1
            p-xy[j,0] += 1
          end
        end
      end
O(1)  p-xy[:,1] ./= p-y
      p-xy[:,2] ./= n-p-y
      p-y ./= n
```

- Inference phase for generative classifier:
  - Use $p(y \mid x) \propto p(x, y)$ to get probabilities for different classes.

Naïve Bayes on MNIST

- Consider fitting naïve Bayes on MNIST digits to distinguish "1" vs. "2".
  - Binary supervised learning problem. 
- There are 6742 "1" examples and 5958 "2" examples.
  - So with MLE we have: $p(y=1) = 6742/(6742+5958)$, or $p(y=1) \approx 0.53$.
- Visualizing the $p(x_j \mid y)$ parameters for each class:

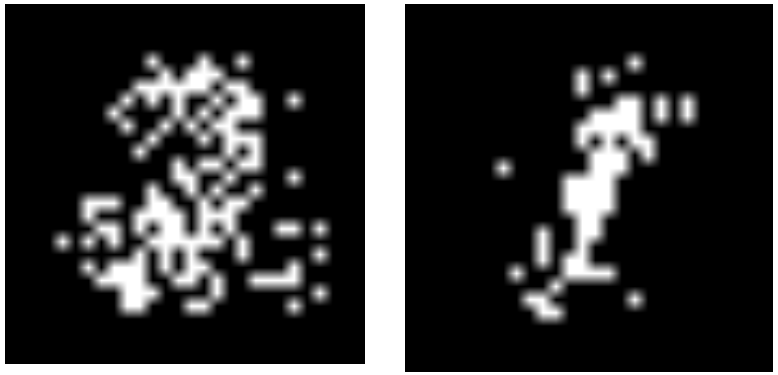


  - These are the product of Bernoullis models for each class.

https://www.kaggle.com/tarunkr/digit-recognition-tutorial-cnn-99-67-accuracy

Naïve Bayes on MNIST

- To sample from naïve Bayes model:
  - Sample a value $\tilde{y}$ from p(y), then independently sample each x_j from $p(x_j \mid \tilde{y})$.
    - "First sample whether the number will be a 1 or 2, then sample each pixel independently."
    - This is "ancestral sampling" – we'll talk in detail about why this works later.
- Two samples from a naïve Bayes model:



- Still a bad model, but they at least now look a bit like digits.
  - For naïve Bayes to classify well, we *don't need a perfect density estimator*.
    - It might have learned enough to say that images of 2s are more likely to be 2s than 1s, even though it does not have a perfect model of either class.
    - This is why naïve Bayes could accurately classify e-mail spam, even though the product of Bernoullis model is one of the worst density estimators.

Generative Classifiers - Discussion

- At the moment, generative classifiers aren't very popular.
  - Historically, you need to make a strong assumption like in naïve Bayes.
    - For "real" images, independence assumption makes the model basically useless.

- Instead of modeling $p(x_1, x_2,\ldots,x_d, y)$ ("generative model"), we usually directly model $p(y \mid x_1, x_2,\ldots,x_d)$ ("discriminative model", next).
  - And usually use a neural network to learn a non-linear mapping (next next).

- But this might change in the future:
  - May be able to learn effective classifiers with less data.
    - Discriminative: "find a way to combine the pixels to explain why this is a dog."
    - Generative: "this is an image of a dog, explain every pixel in the image".
  - Modern density estimation methods work much better than classic methods.

Next Topic: Discriminative Classifiers

Discriminative Classifiers

- Discriminative classifiers directly model $p(y \mid x_1, x_2, \ldots, x_d)$.
  - Might be easier than modeling $p(x_1, x_2, \ldots, x_d, y)$ as done in generative classifiers.

- Key advantage:
  - Only need to figure out how features affect the label.
    - Do not need to model the features, which themselves could be complicated.
    - Do not model p(y) either, we only focus on the mapping from x to y.

- Simple example: a dataset with a binary label and one binary feature.
    - For example, predict "hospitalization" based on "vaccinated".
      - We only focus on predicting "hospitalization" with a known value of "vaccinated", and ignore p("vaccinated").
  - Conditional binary parameterization (like we did with naïve Bayes):
    - $p(y = 1 \mid x = 1) = \theta_1$.
    - $p(y = 1 \mid x = 0) = \theta_0$.
    - Feature 'x' "switches" between 2 Bernoulli distributions for y.
  - Fit with MLE/MAP, compute $p(y \mid x)$ for new examples directly from relevant Bernoulli.
    - But can't do inference about x, since we don't model x at all.

Tabular Parameterization of Conditionals

- Now consider a dataset with binary label and 2 binary features.
  - For example, predict "hospitalization" based on "vaccinated" and "Paxlovid".
  - The tabular parameterization of the conditional probability:
    - $p(y = 1 \mid x_1 = 0, x_2 = 0) = \theta_{00}$.
    - $p(y = 1 \mid x_1 = 0, x_2 = 1) = \theta_{01}$.
    - $p(y = 1 \mid x_1 = 1, x_2 = 0) = \theta_{10}$.
    - $p(y = 1 \mid x_1 = 1, x_2 = 1) = \theta_{11}$.
    - Makes a different Bernoulli for each combination of x values.
  - Basic probability question: why do we need 4 parameters here and not only 3?
- Advantage of tabular representation:
  - Can represent any binary conditional (no restriction on distribution).
- Disadvantage of tabular representation:
  - With d features we need 2^d parameters.

Linear Parameterization of Conditionals

- Tabular parameterization will overfit when you have many features.
  - You might not see some of the 2^d combinations of features in training data.

- Common solution: use a "parsimonious" parameterization.
  - "Parsimonious": has fewer parameters.
  - Hope to need less data by giving up the ability to model any conditional.
- Standard choice parameterizes a linear combination of features:

$$p(y=1 \mid x_1, x_2, \dots, x_d, w) = f(w_1 x_1 + w_2 x_2 + \dots + w_d x_d) = f(w^T x)$$

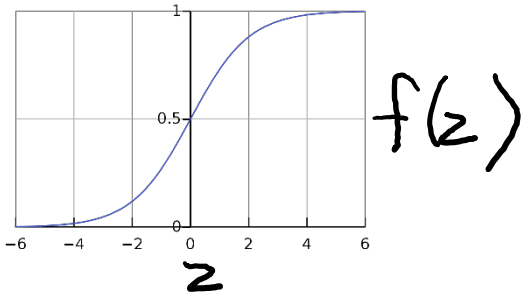
Function 'f' maps from reals $\mathbb{R}$ to $[0,1]$

parameter w_1 is the "weight" on w_1.

Sigmoid Function and Logistic Regression

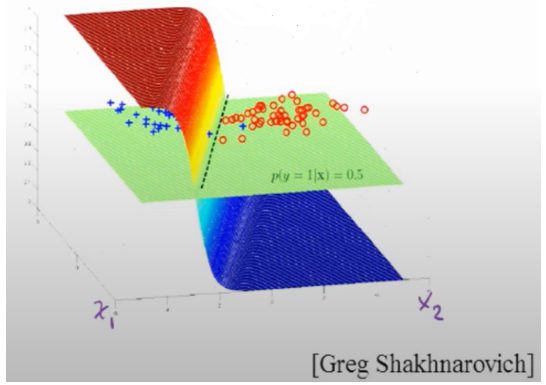
- Sigmoid function is a common choice for mapping (-∞,∞) to [0,1]:

$$f(z) = \frac{1}{1+\exp(-z)}$$



- Using sigmoid to model conditional based on linear combination:

$$p(y=1 \mid x, w) = f(w^T x) = \frac{1}{1+\exp(-w^T x)}$$



[Greg Shakhnarovich]

- This model is called logistic regression.
  - Usually fit with MLE or MAP.
  - Works well in many applications (usually beats naïve Bayes).

https://en.wikipedia.org/wiki/Sigmoid_function
https://www.youtube.com/watch?v=Zc7ouSD0DEQ

Inference in Logistic Regression

- For fixed *w* and *x*, logistic gives binary distribution over y^i values:

$$p(y=1 \mid x, w) = \underbrace{\frac{1}{1+\exp(-w^T x)}}_{\text{"}\theta\text{"}}$$

  - Cost for one example is O(d), due to the inner product $w^T x$.
- You can treat this value as the parameter "θ" in a Bernoulli.
  - If $w^T x > 0$ then $\theta > 0.5$, and if $w^T x < 0$ then $\theta < 0.5$.

```
θ = 1/(1+exp(-X[i,:]*w))
```

  - Usually we just take the mode of this distribution to predict most likely *y*.
  - But you could then do inference conditioned on the values of the features *x*.
    - Sample values of *y* given this value of *x*.
    - Compute probability of seeing 5 examples with *y*=1 among 10 examples for this *x*.
    - Compute the number of samples with these features before expect to get one with *y*=1.
    - Use "decision theory" to make predictions that maximize utility.
    - And so on.

Maximum Likelihood or Conditional Likelihood?

- MLE in generative compared to discriminative models:
  - In generative models, MLE maximizes $p(\mathbf{X}, y \mid w)$.
  - In discriminative models, MLE maximizes $p(y \mid \mathbf{X}, w)$.
    - We maximize the conditional likelihood of *y* (conditioning on features).
      - And we treat the features **X** as fixed.

- Logistic regression can use binary or continuous features in *x*.
  - Even though it only uses binary probabilities.
- This is different than we saw with naïve Bayes:
  - Naïve Bayes needed independence assumption even for binary features.
    - Naïve Bayes would need to model continuous probabilities for continuous features.

review

Review: Logistic "Negative Log-Likelihood"

- With *n* training examples, logistic regression NLL is:

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^i w^T x^i))$$

  - For binary linear classifiers, usually convenient to assume $y^i \in \{-1, +1\}$, instead of {0,1}.
  - NLL equivalent to what some people call "binary cross entropy".
  - Cost is O(nd); bottleneck is computing the *n* $w^T x^i$ values for O(d) each.
    - Code to compute *f* and its gradient *g*:
      - The $w^T x^i$ values are computed via matrix multiplication "X*w".

```
function logisticObj(w,X,y)
    yXw = y.*(X*w)
    f = sum(log.(1 .+ exp.(-yXw)))
    g = -X'*(y./(1 .+ exp.(yXw)))
    return (f,g)
end
```

- This is a convex function, so if $\nabla f(w) = 0$ then w is a global minimum.
- Setting $\nabla f(w) = 0$ does not lead to closed-form solution for *w* (in general).
- But since *f* is differentiable and convex,
  we can converge to a *w* with $\nabla f(w) = 0$ with gradient descent.
  - Or stochastic gradient descent, or other optimization algorithms…
  - Best choice depends on *n*, desired accuracy, computational setup, ….

Binary Naïve Bayes is a Linear Model

$$p(y=1 \mid x) = \frac{p(x \mid y=1)\, p(y=1)}{p(x \mid y=1)\, p(y=1) + p(x \mid y=0)\, p(y=0)}$$

$$= \frac{1}{1 + \frac{p(x \mid y=0)\, p(y=0)}{p(x \mid y=1)\, p(y=1)}} = \frac{1}{1 + \exp\left(-\log \frac{p(x \mid y=1)\, p(y=1)}{p(x \mid y=0)\, p(y=0)}\right)}$$

bonus!

$$= \sigma\left(\sum_{j=1}^{d} \log \frac{p(x_j \mid y=1)}{p(x_j \mid y=0)} + \log \frac{p(y=1)}{p(y=0)}\right)$$

$$= \sigma\left(\sum_{j=1}^{d} \log \frac{\theta_{j1}^{x_j} (1-\theta_{j1})^{1-x_j}}{\theta_{j0}^{x_j} (1-\theta_{j0})^{1-x_j}} + \log \frac{p(y=1)}{p(y=0)}\right)$$

$$= \sigma\left(\sum_{j=1}^{d} x_j \log \frac{\theta_{j1}}{\theta_{j0}} + (1-x_j) \log \frac{1-\theta_{j1}}{1-\theta_{j0}} + \log \frac{p(y=1)}{p(y=0)}\right)$$

$$= \sigma\left(\sum_{j=1}^{d} x_j \underbrace{\log \frac{\theta_{j1}(1-\theta_{j0})}{(1-\theta_{j1})\,\theta_{j0}}}_{w_j} + \underbrace{\sum_j \log \frac{1-\theta_{j1}}{1-\theta_{j0}} + \log \frac{p(y=1)}{p(y=0)}}_{b}\right)$$

$$= \sigma(w^T x + b)$$

bonus!

Non-probabilistic predictors

- There are also *non-probabilistic* discriminative models that directly learn a map from *x* to *y*
  - Support vector machines (SVMS), usual decision trees, …

	$p(x,y)$	$p(y \mid x)$	$f: x \mapsto y$	
Generative	✓	✓	✓	(naive Bayes)
Discriminative – probabilistic	✗	✓	✓	(logistic regression)
Discriminative – non-probabilistic	✗	✗	✓	(SVM)

- Accuracy is often (not always) higher as you model fewer steps
  - Vladimir Vapnik: "When solving a problem of interest, do not solve a more general problem as an intermediate step."
  - But number of inference tasks you can do gets more limited.
    - Discriminative models can't answer questions involving p(x, y).
    - "Pure classifiers" can't answer questions involving p(y | x).

review

Review: Regularization and MAP

- Common to add a regularizer, such as L2-regularization, to the NLL:

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^i w^T x^i)) + \frac{\lambda}{2}\|w\|^2$$

  - Typically gives better test error with appropriate hyper-parameter $\lambda > 0$.
  - L2-regularization corresponds to MAP estimation with a Gaussian prior.
    - We'll cover Gaussians later.

- In both generative/discriminative cases, MAP maximizes posterior:

$$\hat{w} \in \underset{w}{\operatorname{argmax}}\{p(w \mid X, y)\}$$

generative

$$\equiv \underset{w}{\operatorname{argmax}}\{p(y, X \mid w)p(w)\}$$

discriminative

$$\equiv \underset{w}{\operatorname{argmax}}\{p(y \mid X, w)p(w)\}$$

(assumes X is independent of w)

Recap: Tabular Conditional vs. Logistic Regression

- Our two discriminative models for binary classification:
  - Tabular parameterization:
    - Has 2^d parameters.
    - Can model any binary conditional probability.
    - Tends to overfit unless d is tiny.
  - Logistic regression:
    - Has d parameters (or d+1 if you add a "bias" variable).
    - Can only model a limited class of binary conditional probabilities.
    - Tends to underfit unless d is large.
- Classical "learning theory" results explore how factors like "number of parameters" and "model class limits" affect test error.

review

Review: Fundamental Trade-Off

- Tabular and logistic are on different parts of fundamental trade-off:
  1. E_{train}: how small you can make the training error.

     vs.

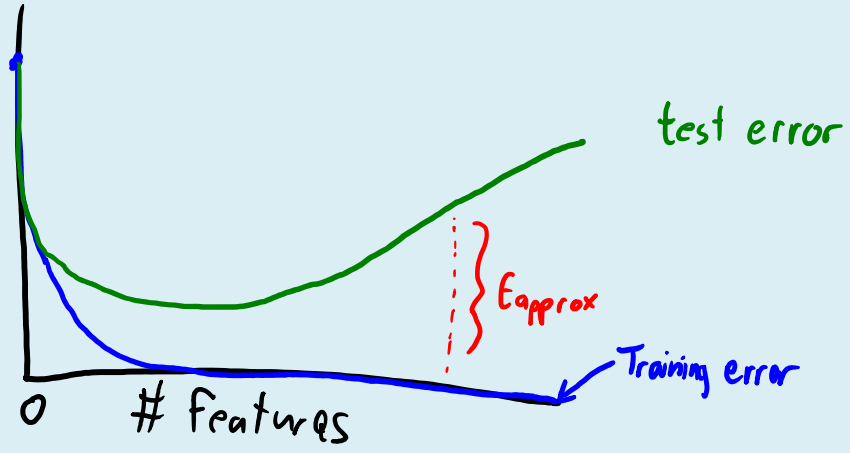
  2. $E_{generalization}$: how well training error approximates the test error (overfitting).

- Simple models (like logistic regression with few features):
  - E_{approx} is low (not very sensitive to training set).
  - But E_{train} might be high (cannot fit data very well).
- Complex models (like tabular conditionals with many features):
  - E_{train} can be low (can fit data very well).
  - But E_{approx} might be high (very sensitive to training set).

review

Review: Non-Linear Feature Transformations

- We can explore models between tabular and logistic:
  - For example, apply logistic regression with non-linear feature transforms:
    1. Transform each feature vector x^i into a new feature vector z^i.
    2. Train regression weights v using the features z^i as the data.
    3. At test time, do the same transformation for the test features.
  - Examples:
    - Polynomials, radial basis functions (RBFs), interaction terms, periodic functions.
- Effect on fundamental trade-off:
  - Adding features makes training error decrease.
  - But generalization gap might increase.
- Regularized logistic regression with linear or Gaussian RBF features, and using a validation set to choose λ (and σ), is often hard to beat.



Next Topic: Neural Networks

Neural Networks: Motivation

- Many domains require non-linear transforms of the features.
  - But, it may be obvious which transform to use.

- Neural network models try to learn good transformations.
  - Optimize the "parameters of the features".
    - And choose a class of features that have the ability to represent many functions.

- We'll start with a special case: "one hidden layer".
  - Then we'll move onto "deep learning," with uses multiple layers.

Neural Network History

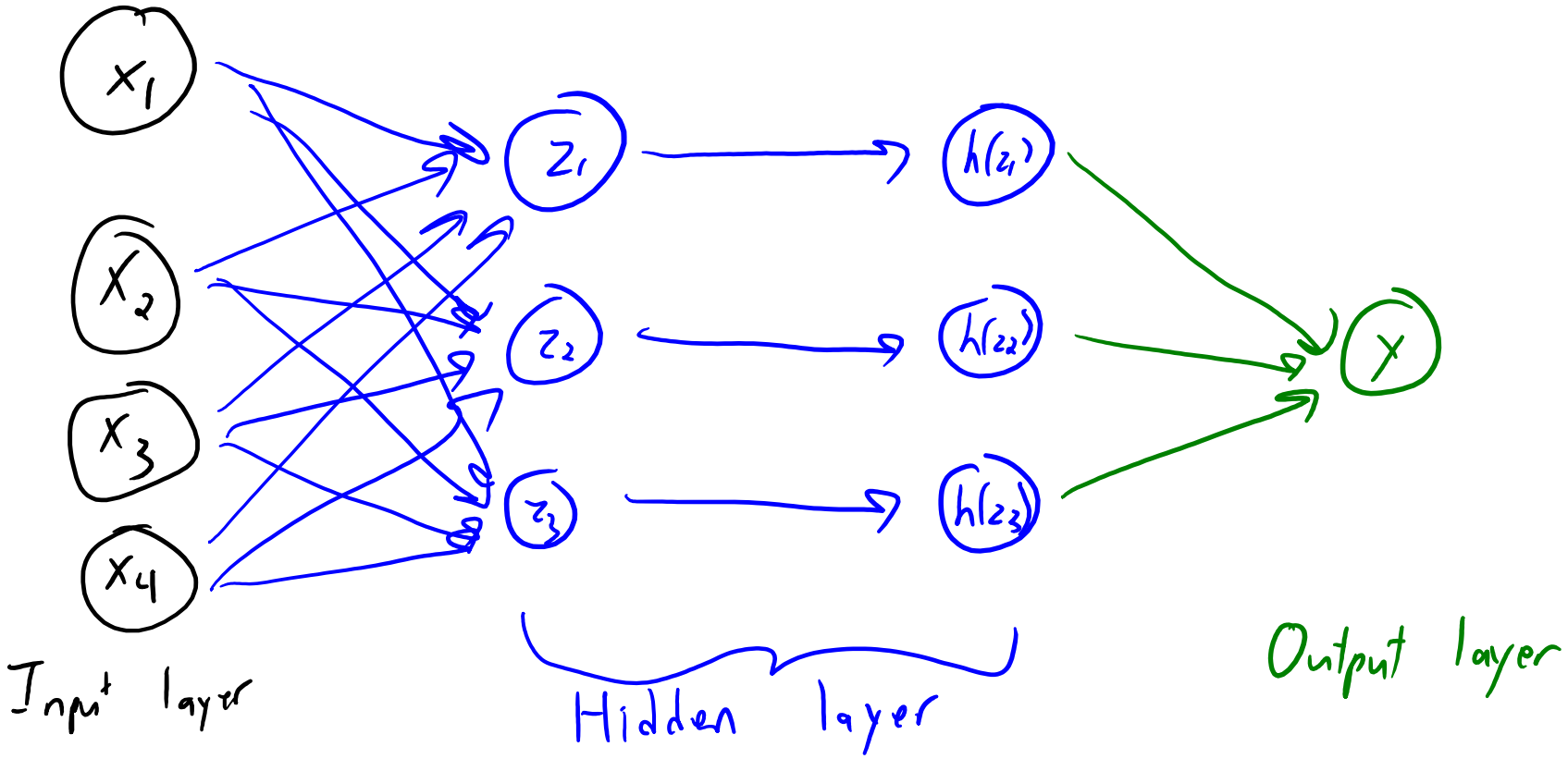
- Popularity of neural networks has come in waves over the years.
  - Currently, it is one of the hottest topics in science.

- Recent popularity due to unprecedented performance on some difficult tasks.
  - Speech recognition.
  - Computer vision.
  - Machine translation.
  - Natural language modeling.
- There are mainly due to big datasets, deep models, and tons of computation.
  - Plus tweaks to classic models and focus on structures 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

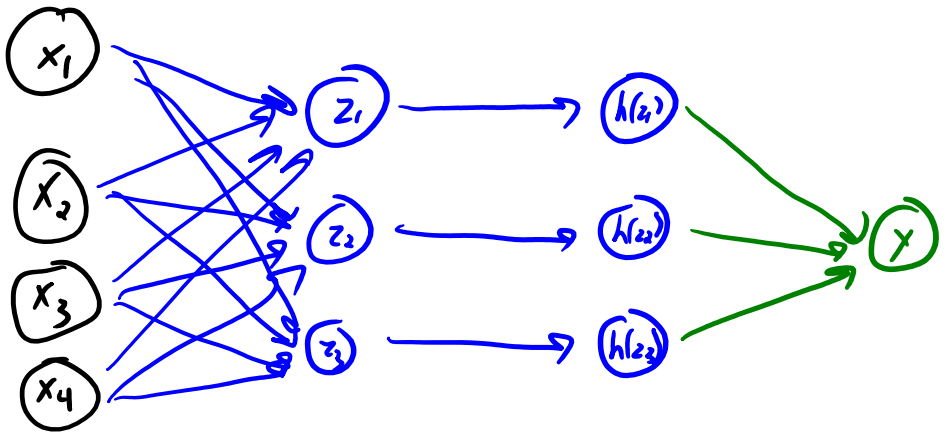
Neural Network with One Hidden Layer

- Classic neural network structure with one hidden layer:



Neural Network with One Hidden Layer

- As a picture:



- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}})$$

Linear combination of "activations"

Non-linear transformation of each z_j, called the "activations"

"z": linear combination of input

Neural Network with One Hidden Layer

- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}})$$

Linear combination of "activations"

Non-linear transformation of each z_j, called the "activations"

"z": linear combination of input

- Parameters: the "k times d" matrix W, and length-k vector v.
  - Using k as "number of activations".

$$W = \begin{bmatrix} \text{---} & w_1^T & \text{---} \\ \text{---} & w_2^T & \text{---} \\ & \vdots & \\ \text{---} & w_k^T & \text{---} \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

$k \times d$ $\qquad$ $k \times 1$

Neural Network with One Hidden Layer

- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}})$$

Linear combination of "activations"

Non-linear transformation of each z_j, called the "activations"

"z": linear combination of input

- Linear transformation $z=Wx$: can think of like doing PCA.
  - Mixes together the features in a way that we learn.
- Non-linear transform h might be sigmoid (or others), applied element-wise.
  - Without a non-linear transformation it degenerates to a linear model:
    - $v^T(Wx) = (v^TW)x = w^Tx$, for $w=W^Tv$.

Neural Network with One Hidden Layer

- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}})$$

Linear combination of "activations"

Non-linear transformation of each z_j, called the "activations"

"z": linear combination of input

- Second linear transformation $v^T h(z)$ gives final value.
  - This is like using a linear model with non-linear feature transformations.
    - But in this case we learned the features.
- Cost of computing $\hat{y}$ is $O(kd)$.
  - $O(kd)$ to compute Wx, $O(k)$ to apply h, then $O(k)$ to multiply by v.

Neural Network with One Hiden Layer

- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}})$$

Linear combination of "activations"

Non-linear transformation of each z_j, called the "activations"

"z": linear combination of input

- You then use $\hat{y}$ for inference.
  - For binary classification, you could use the sigmoid function:

$$p(y \mid x, W, v) = \frac{1}{1 + \exp(-y \underbrace{v^T h(Wx)}_{\hat{y}})}$$

  - This is like logistic regression with optimized features.

Adding Bias Variables

- Recall fitting linear models with a bias variable (so $\hat{y} \neq 0$ when x=0).

$$\hat{y} = \sum_{j=1}^{d} w_j x_j + \beta$$

  – We often implement this by adding a column of ones to X.
- In neural networks we often include biases on each z_c:

$$\hat{y} = \sum_{c=1}^{K} v_c h(w_c^T x + b_c)$$

  – As before, we could implement this by adding a column of ones to X.
- We also probably want a bias on the output:

$$\hat{y} = \sum_{c=1}^{K} v_c h(w_c^T x + b_c) + \beta$$

  – For sigmoids, you could equivalently fix one row of w_c to be equal to 0.
    - This gives $v_c h(w_c^T x) = v_c h(0) = v_c/2$, so the value $2v_c$ will give the bias β.

Universal Approximation with One Hidden Layer

- Classic choice of "activation" function is the sigmoid function.
- With enough hidden "units", this is a "universal approximator."
  - Any continuous function can be approximated arbitrarily well (on bounded domain).

- But this result is for a non-parametric setting of the parameters:
  - The number of hidden "units" must be a function of n.
  - A fixed-size network is not a universal approximator.

- Other universal approximators (always non-parametric):
  - K-nearest neighbours.
    - Need to have k depending on n (but this model is always non-parametric anyway).
  - Linear models on polynomial feature transformations.
    - Need degree of the polynomial to grow with n.
  - Linear models with Gaussian RBFs as non-linear features.
    - With on basis function centered on each x^i.

Is Training Neural Networks Scary?

- Learning:
  - For binary classification, the NLL under the sigmoid loss is:

$$f(W,v) = \sum_{i=1}^{n} \underbrace{\log(1+\exp(-y^i v^T h(Wx^i)))}_{f_i}$$

loss function on example i

    - With W fixed this is convex, but with W and v as variables it is non-convex.
    - And finding the global optimum is NP-hard in general.
  - Nearly always trained with variations on stochastic gradient descent (SGD).

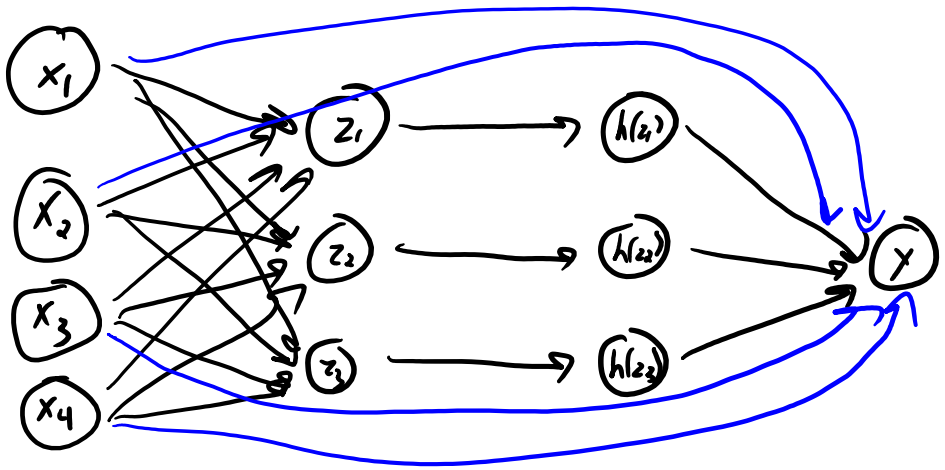
$$W^{k+1} = W^k - \alpha^k \nabla_W f_{i_k}(W^k, v^k)$$
$$v^{k+1} = v^k - \alpha^k \nabla_v f_{i_k}(W^k, v^k)$$

i_k is a training example chosen uniformly at random

    - Many variations exist (adding "momentum", AdaGrad, Adam, and so on).
    - SGD is not guaranteed to reach a global minimum for non-convex problems.
- Is non-convexity a big drawback compared to logistic regression?
  - And if k is large, is this likely to overfit?

Neural Networks ≥ Logistic Regression

- Consider a neural network with one hidden layer and connections from input to output layer.
  - The extra connections are called "skip" connections.



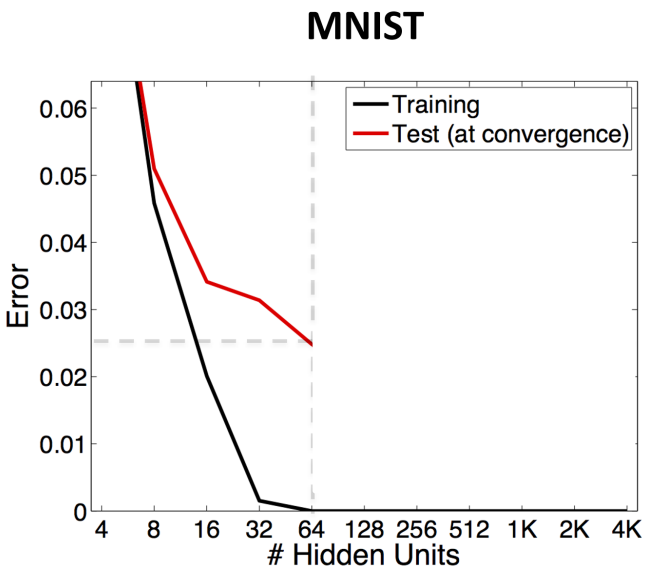
$$\hat{y} = \underbrace{w^T x}_{\text{linear model}} + \underbrace{v^T h(Wx)}_{\text{neural network}}$$

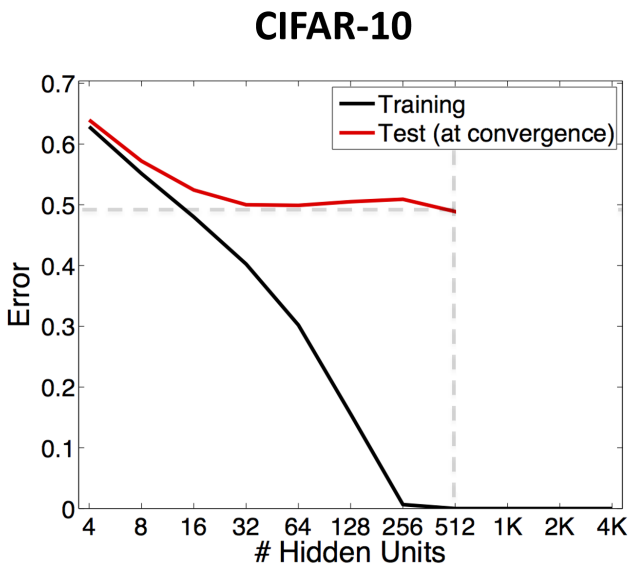
- You could first set v=0, then optimize w using logistic regression.
  - This is a convex optimization problem that gives you the logistic regression model.
- You could then set W and v to small random values, and start SGD from the logistic regression model.
  - Even though this is non-convex, the neural network can only improve on logistic regression (improves "residual" error).
- And if you are worried about overfitting, you could stop SGD by checking performance on validation set.
  - This is called regularization by "early stopping".
- In practice, we typically optimize everything at once (which usually works better than the above).

Next Topic: Implicit Regularization

"Hidden" Regularization in Neural Networks

- Fitting single-layer neural network with SGD and no regularization:



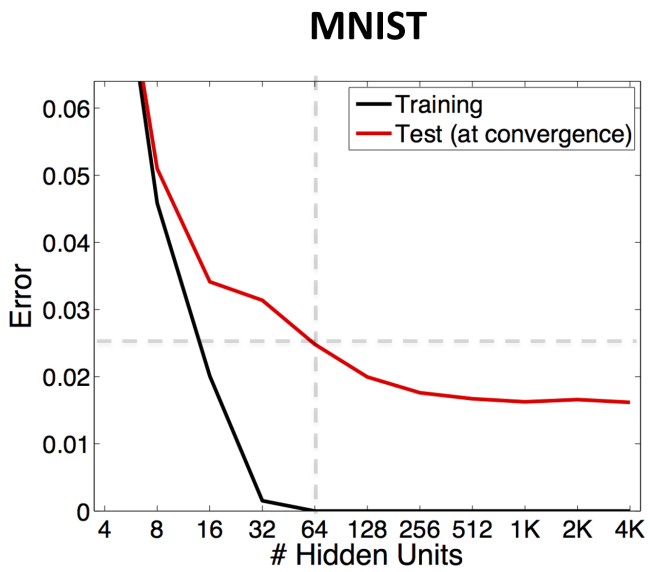


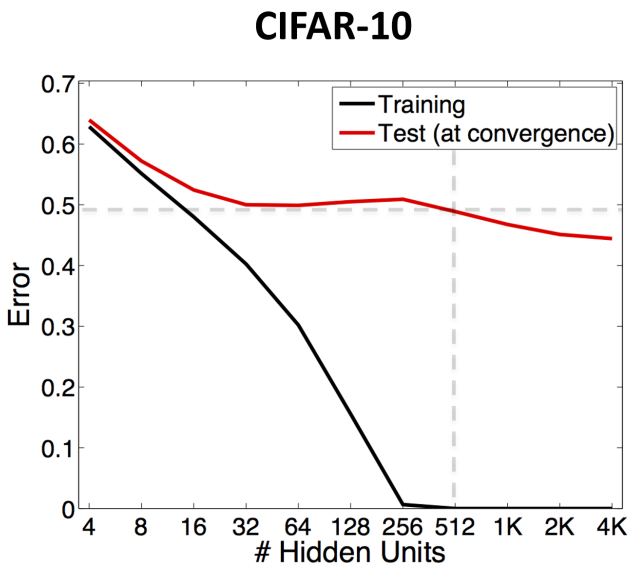
- On each step of the x-axis, the network is re-trained from scratch.
- Training goes to 0 with enough units: we're finding a global min.
- What should happen to training and test error for larger #hidden?

https://www.neyshabur.net/papers/inductive_bias_poster.pdf

“Hidden” Regularization in Neural Networks

- Fitting single-layer neural network with SGD and no regularization:





- Test error continues to go down!?! Where is fundamental trade-off??
  - Is it is still fundamental, but FTO focuses on the “worst” global minimum.
- There do exist global mins with large #hidden units have test error = 1.
  - But among the global minima, SGD is somehow converging to “good” ones.

https://www.neyshabur.net/papers/inductive_bias_poster.pdf

Summary

- Naïve Bayes:
  - Generative classifier, p(x|y) a product of Bernoullis
- Discriminative Classifiers:
  - Directly model p(y | x) rather than p(x ,y).
  - Most of modern machine learning is based on discriminative classifiers.
- Tabular parameterization:
  - Fit a parameter for p(y=1 |x) for each possible value of 'x'.
  - Can model any conditional, but overfits unless 'd' is small.
- Logistic regression:
  - Write p(y | x) using the sigmoid function.
  - MLE is a convex optimization problem.
  - Trained using variations on gradient descent.
  - Cannot model any conditional, but tends not to overfit (especially with regularization).
- Fundamental Trade-Off:
  - Simple models can underfit (high train error);
  - complex models usually overfit (high gen. gap).
- Neural networks with one layer:
  - Simultaneous learn a linear model and its features.
  - Universal approximator if size of layer grows with number of examples 'n'.
  - Training is a non-convex optimization problem.
- Empirical "good news" for training neural networks with SGD:
  - With enough hidden units, SGD often finds a global minimum.
- Next time: we descend deeper (twice).

bonus!

Logistic Regression Training Code

- Gradient descent for logistic regression:

$$w^{k+1} = w^k - \alpha^k \underbrace{\nabla f(w^k)}_{X^T r} \quad \text{where } r^i = \frac{-y^i}{1+\exp(y^i w^T x^i)}$$

- Simple method for setting the step size:
  - If $f(w^{k+1}) > f(w^k)$, divide α in half and see if that decreases 'f'.
    - There are much-more clever ways to set the step size (for example, Barzilai-Borwein method in assignment code).
    - There are also better "directions" than using the gradient, such as quasi-Newton and Hessian-free Newton.
    - For stochastic gradient descent, you need a decreasing set of step sizes to guarantee convergence.
- Deciding when to stop:
  - Check if $||\nabla f(w)|| \leq \epsilon$ for some small ϵ.
  - Or check for progress in function/iteration values, and "give up" if you no longer are making progress.
- Cost is O(nd) per iteration.
  - Computing each of 'n' inner-product $w^T x^i$ costs O(d), giving O(nd).
  - Computing $X^T r$ in the gradient costs O(nd).
  - Updating w given the gradient costs O(d) so does not increase cost.
- If the matrix 'X' only has 'z' non-zero values, can be implemented in O(z).
- Cost is only O(d) for stochastic gradient descent, but you will spend a lot of time tuning step sizes.