CPSC 440: Machine Learning

Neural Networks

Winter 2022

Last Time: Discriminative Classifiers

- Discriminative classifiers model p(y | x) for supervised learning.
  - Unlike generative classifiers that model (x, y).
  - Allows us to use complicated features, without modeling them.
- We discussed using tabular conditional probabilities:

$$p(y=1 \mid x_1=1, x_2=0, x_3=1) = \theta_{101}$$

Separate Bernoulli for each set of values $x_1, x_2, \ldots, x_d$

- We discussed using logistic regression:

$$p(y=1 \mid x_1=1, x_2=0, x_3=1) = \frac{1}{1+\exp(-(w_1x_1 + w_2x_2 + w_3x_3))}$$

One "regression weight" for each feature

  - Sigmoid function transforms from (-∞,∞) to (0,1).

Review: 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: Fundamental Trade-Off

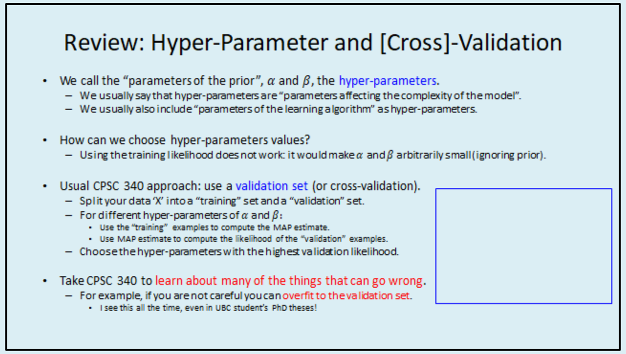
- Tabular and logistic are on different parts of fundamental trade-off:
  1. E_{train}: how small you can make the training error.

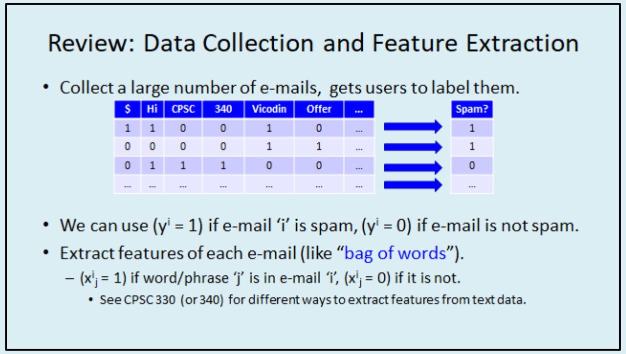
     vs.
  2. E_{approx}: how well training error approximates the test error (overfitting).

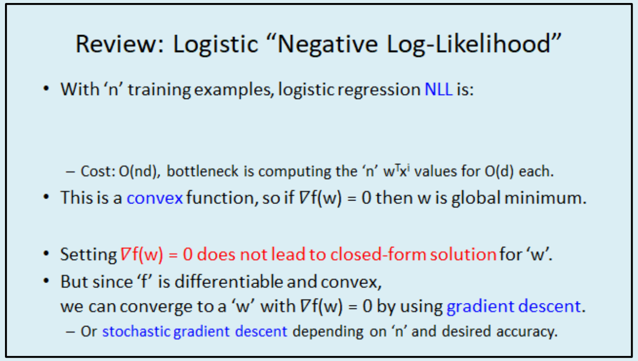
- Simple models (like logistic regression with few features):
  – E_{approx} is low (not very sensitive to training set).
  – But E_{train} might be high (cannot fit data very well).
- Complex models (like tabular conditionals with many features):
  – E_{train} can be low (can fit data very well).
 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

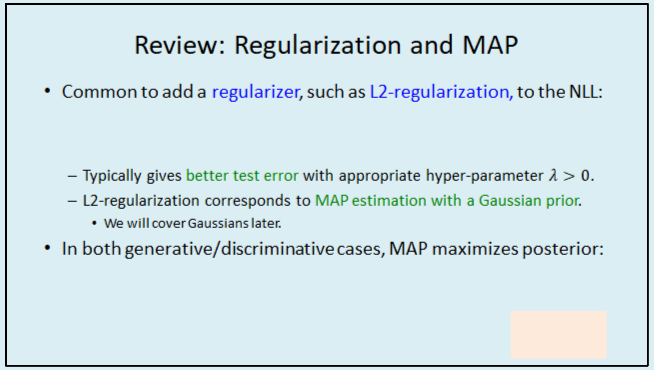
Review: “Review Slides”

- I have coloured some slides in blue, and used “Review:…” as their title.
  - These are topics that are covered in detail in CPSC 340.
  - I expect you to understand these topics to follow the course.
  - But we will not cover these topics in detail in this course.





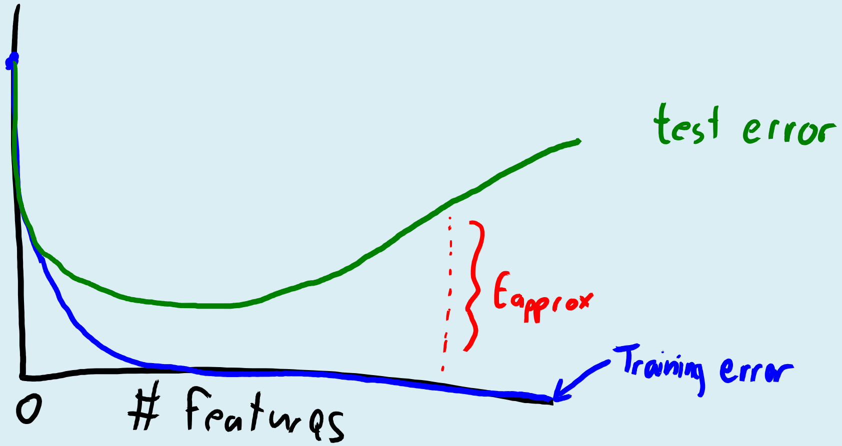




- I added this colour to some slides from the previous lectures.
  - L4: Hyperparameters, [cross]-validation, bag of words, feature extraction.
  - L5: NLLs, convexity, gradient descent, SGD, regularization, “why regularize?”.

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 approximation error 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 will first discuss the special case of “one hidden layer”.
  - Then we will move onto “deep learning” with uses multiple layers.

Neural Network History

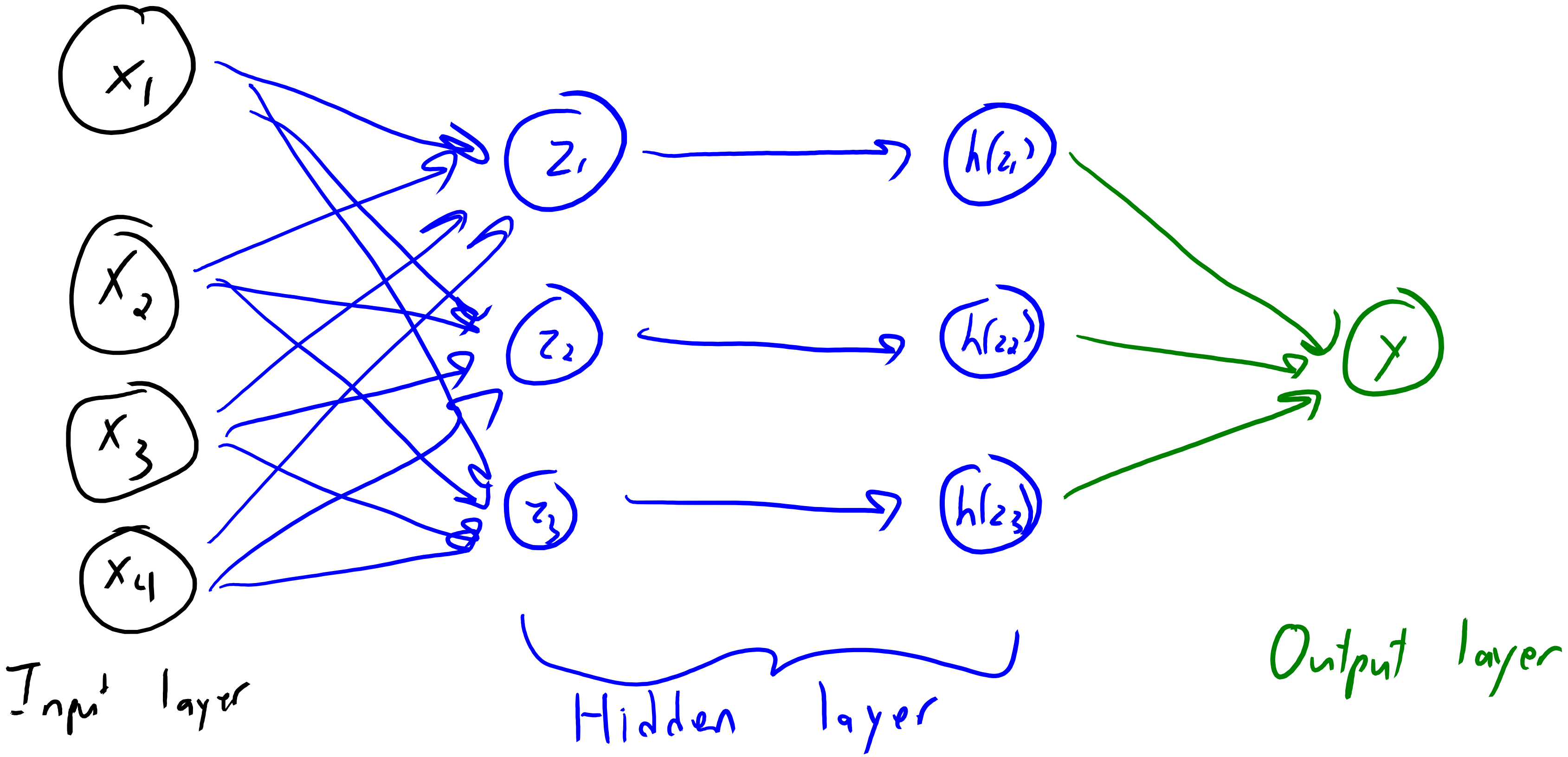
- Popularity of neural networks has come in waves over the years.
  - Currently, it is one of the hottest topics in science.

- Recent popularity due to unprecedented performance on some difficult tasks.
  - Speech recognition.
  - Computer vision.
  - Machine translation.
- 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

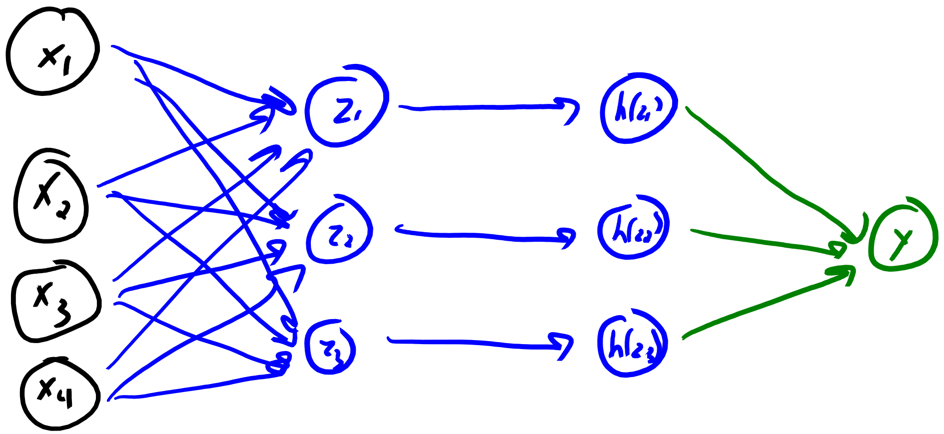
Neural Network with One Hiden Layer

- Classic neural network structure with one hidden layer:



Neural Network with One Hiden Layer

- As a picture:



- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}}_{\text{"z": linear combination of input}})$$

Linear combination of "activations"

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

Neural Network with One Hiden Layer

- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}}_{\text{"z": linear combination of input}})$$

Linear combination of "activations"

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

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

$$W = \underbrace{\begin{bmatrix} \text{---} & w_1^T & \text{---} \\ \text{---} & w_2^T & \text{---} \\ & \vdots & \\ \text{---} & w_k^T & \text{---} \end{bmatrix}}_{k \times d} \qquad v = \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}}_{k \times 1}$$

Neural Network with One Hiden Layer

- As a function:

$$\hat{y} = v^T h(\underbrace{W \underbrace{x}_{\text{input}}}_{\text{"z": linear combination of input}})$$

Linear combination of "activations"

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

- Linear transformation z=Wx is like doing PCA.
  - Mixes together the features in a way that we learn.
- Non-linear transform 'h' is often sigmoid, 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 Hiden Layer

- As a function:

$$\hat{y} = \underbrace{v}_{\text{Linear combination of "activations"}}^T \underbrace{h}_{\text{Non-linear transformation of each } z_j\text{, called the "activations"}}(\underbrace{W \underbrace{x}_{\text{input}}}_{\text{"z": linear combination of input}})$$

- 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:

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

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 often also 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'.
  - Linear models with polynomial non-linear features transformations.
    - Degree of polynomial depends on 'n'.
  - Linear models with Gaussian RBFs as non-linear features transformations.
    - 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).

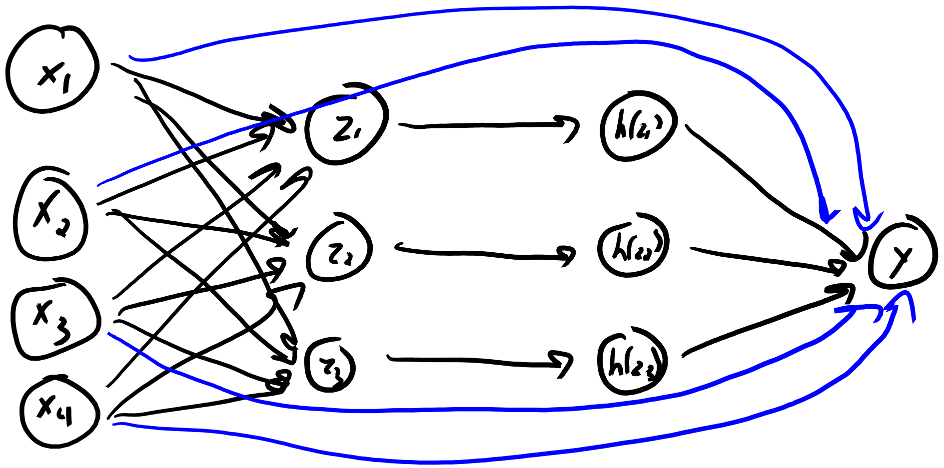
$$W^{k+1} = W^k - \alpha^k \nabla_W f_{i_k}(W^k, v^k)$$
$$v^{k+1} = v^k - \alpha^k \nabla_v f_{i_k}(W^k, v^k)$$

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.



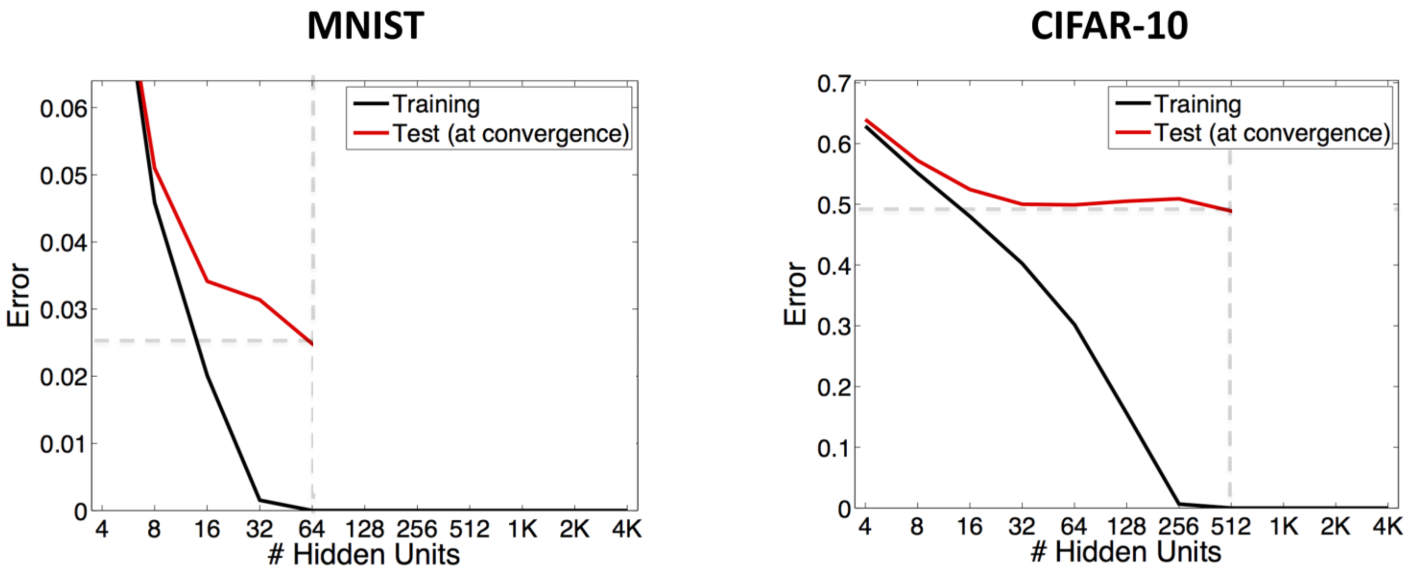
$$\hat{y} = \underbrace{w^T x}_{\text{linear model}} + \underbrace{v^T h(Wx)}_{\text{neural network}}$$

- 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:



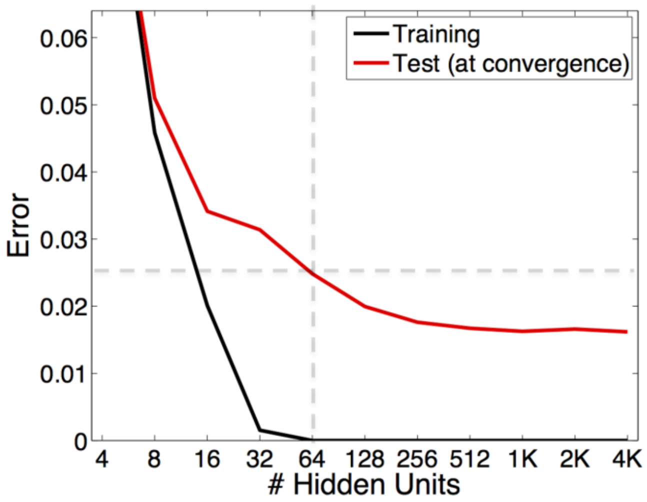
- On each step of the x-axis, the network is re-trained from scratch.
- Training goes to 0 with enough units: we’re finding a global min.
- What should happen to training and test error for larger #hidden?

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

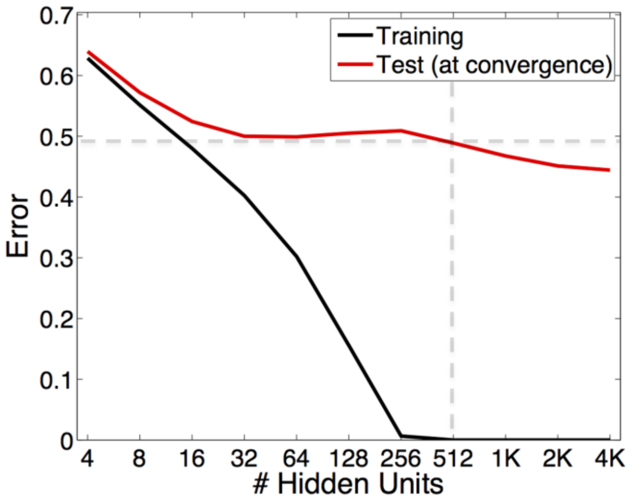
“Hidden” Regularization in Neural Networks

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

MNIST



CIFAR-10



- 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

- Fundamental Trade-Off:
  - Learning theory says that simple models do not overfit but may underfit.
  - Learning theory says that complicated models do not unferfit but may overfit.
- 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 start going “deep”.