Deep Learning Theory and Applications

Regularization

CPSC/AMTH 663

Yale

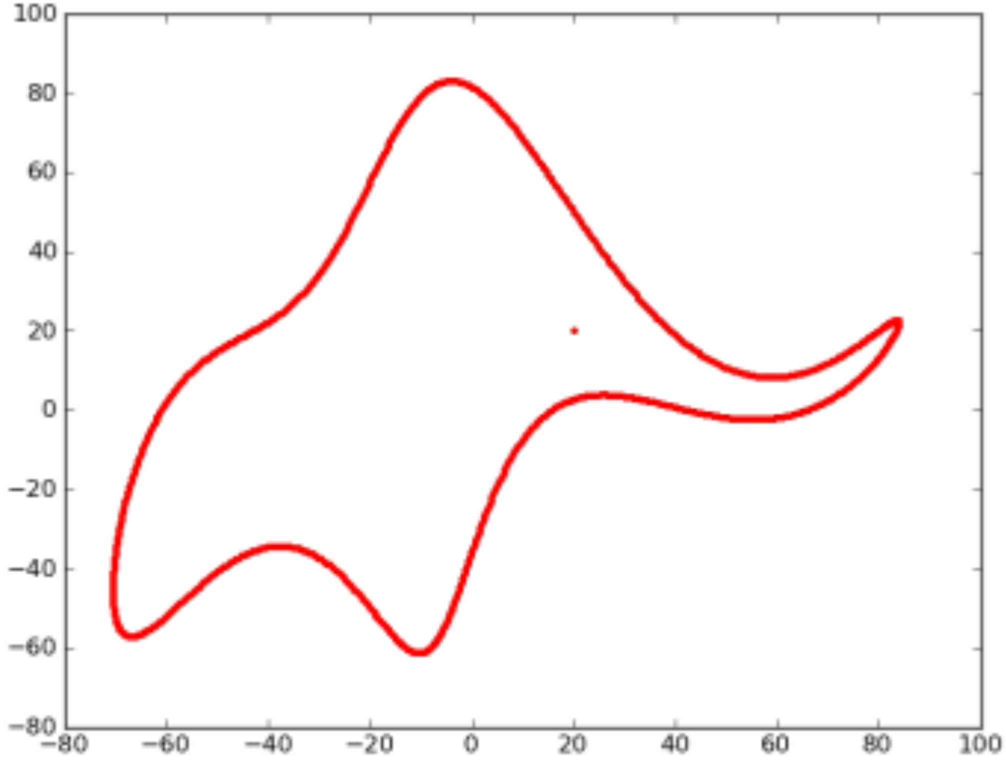
Outline

1. Overfitting
2. L2 Regularization
3. Other regularization techniques
    1. L1 Regularization
    2. Dropout
    3. Augmenting the training data
    4. Noise robustness
    5. Tangent Propogation
    6. Entropy Regularization

Free parameters

- Nobel prizewinning physicist Enrico Fermi was once asked about a mathematical model proposed as a solution to an important physics problem
- Fermi asked how many free parameters could be set in the model
  - Answer was “four”
- Fermi responded, “I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk.”

Free parameters



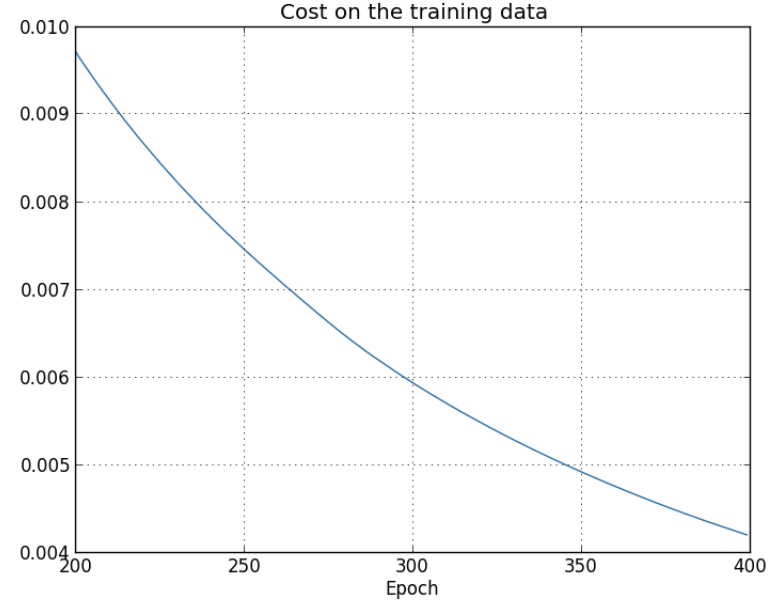
https://www.johndcook.com/blog/2011/06/21/how-to-fit-an-elephant/

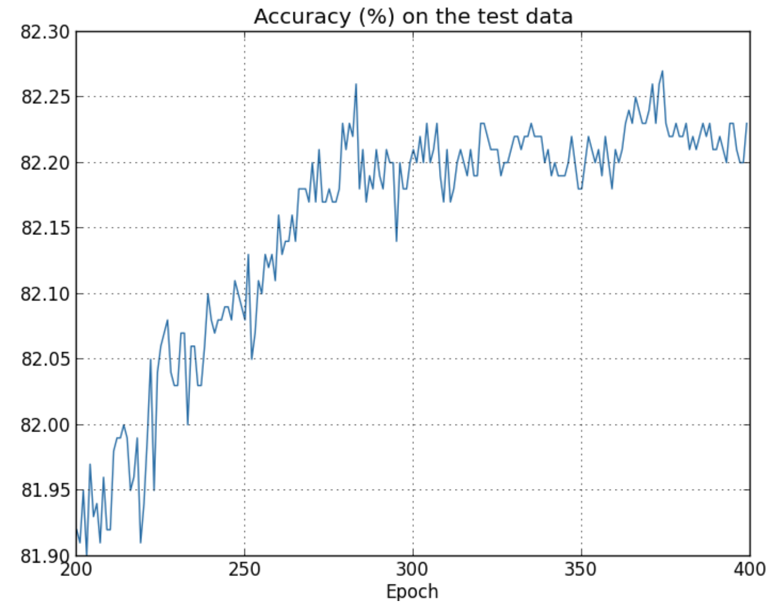
Free parameters

- Models with large # of free parameters can describe a wide range of phenomena
- Agreement with the data doesn’t guarantee a good model
  - Model may just be able to describe data set of a given size
  - I.e., model may work well for existing data but fail to generalize
- True test is can the model make accurate predictions on new data?
- How many parameters do our neural networks for handwritten digit recognition have?
  - 30 node hidden layer model: nearly 24,000 parameters
  - 100 node hidden layer model: nearly 80,000 parameters
- Can we trust their results?

MNIST recognition revisited

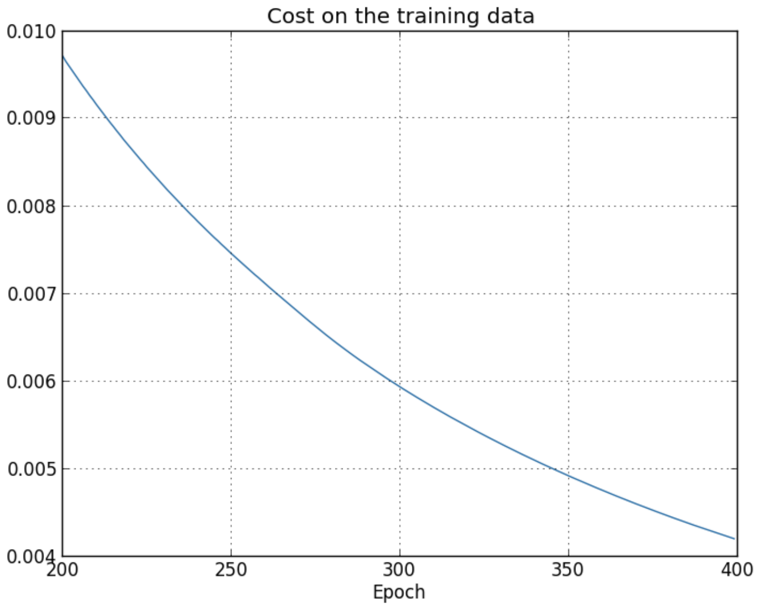
- 30 hidden neurons
- Train with first 1,000 training images
  - Cross-entropy cost
  - Learning rate $\eta = 0.5$
  - Mini-batch size 10
  - 400 epochs
- Training cost decreases with each epoch
- But accuracy flatlines ≈ epoch 280
  - The learned network is not generalizing well
  - The network is ***overfitting*** or ***overtraining***

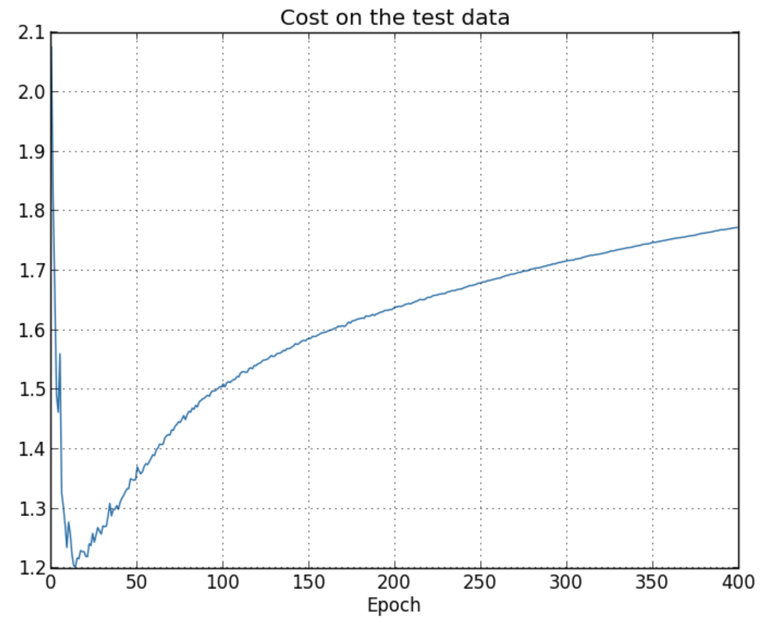




Cost vs classification accuracy

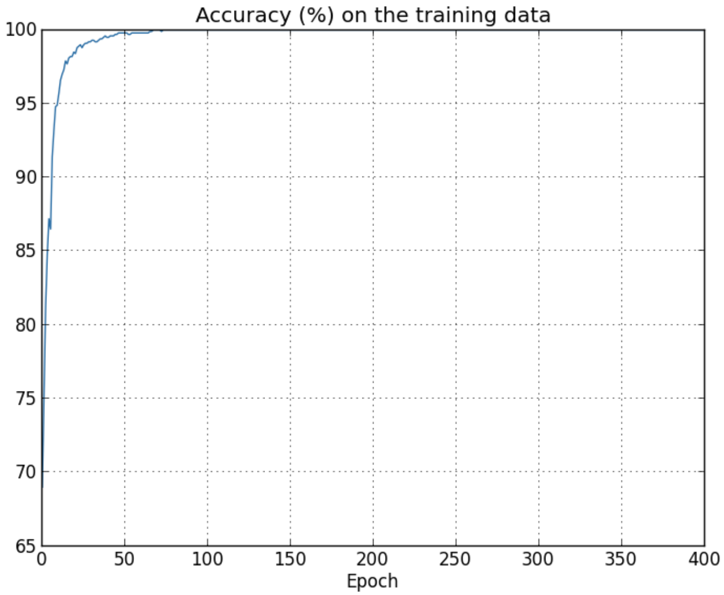
- We were comparing the ***cost*** on the training data to the ***classification accuracy*** on the test data
  - Is this an apples to oranges comparison?
  - Should we compare the cost in both cases or the accuracy in both cases?

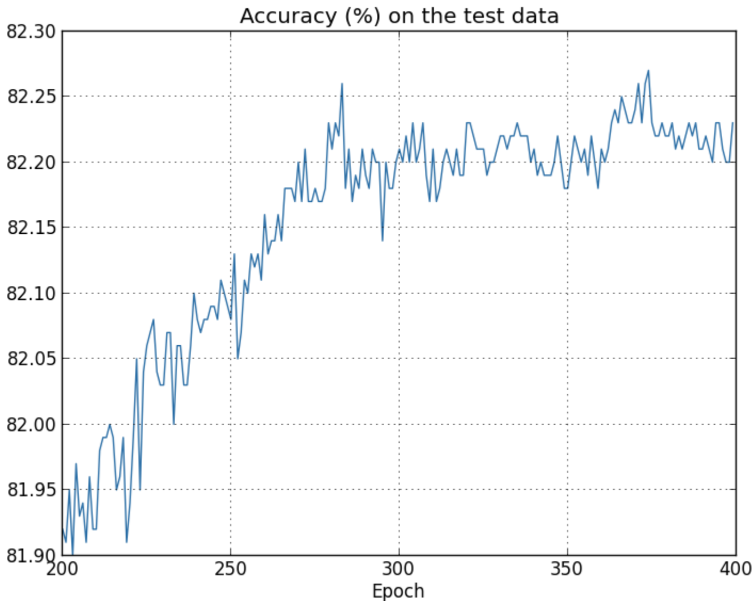




Cost vs classification accuracy

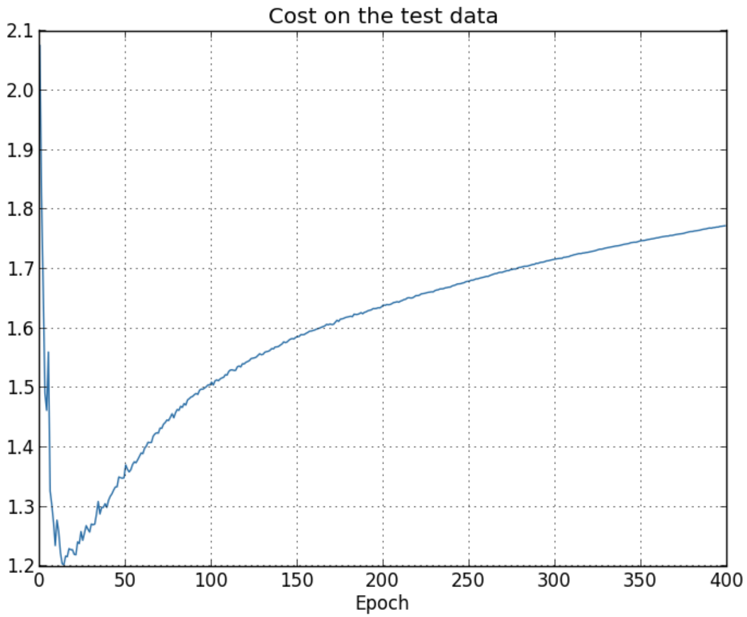
- Comparing the accuracy also suggests overfitting
  - The network is learning the peculiarities of the training set

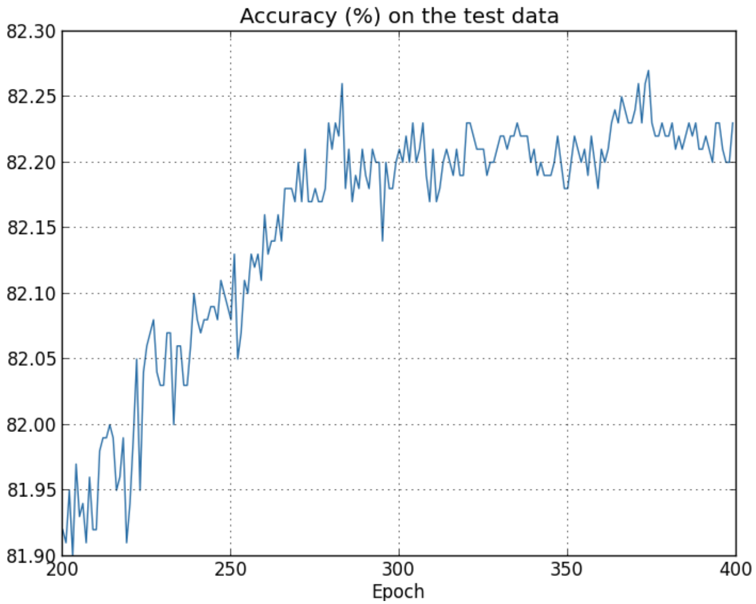




When does overfitting occur?

- Epoch 15 or epoch 280?





- The cost is a proxy for what we really care about: accuracy
- So epoch 280 makes the most sense

How can we prevent overfitting?

- One approach: track the test accuracy during training
  - Stop training if the test accuracy no longer improves
  - Caveat: this isn’t necessarily a sign of overfitting but stopping when this occurs will prevent overfitting
- A variation: track the accuracy on a ***validation*** *set*
  - Stop training when we’re confident the validation accuracy has saturated
    - Requires some judgment as networks sometimes plateau before improving
  - Referred to as ***early stopping***

How can we prevent overfitting?

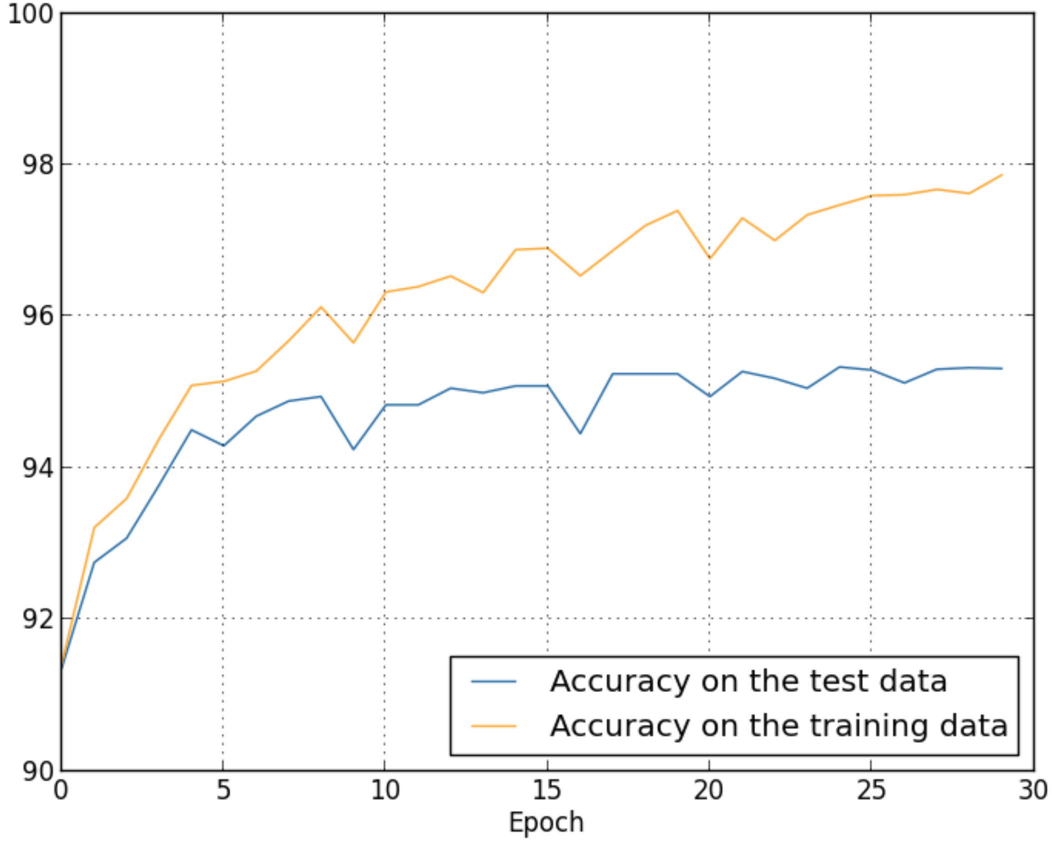
- Why should we use a validation set instead of the test set to determine the number of epochs (and other hyper-parameters)?
  - We’re likely to try many different choices for the hyper-parameters
  - If the hyper-parameters are set based on the test data, we may overfit to the test data
- After setting the hyper-parameters with the validation set, we evaluate the final performance on the test set
  - This gives us confidence that the test set results are a true measure of how well our learned network generalizes

How can we prevent overfitting?

- What if we don't like the test results?
  - We could try another approach including a different architecture
  - Isn't there a danger of overfitting the test data?
  - Yes!
- What can we do?
  - Good question…
  - Cross-validation can help some
  - Could consider different architectures during the validation stage

Handwritten digit recognition revisited

- Were we overfitting when we used all 50,000 training images?



- Some, but not nearly as much as before

A great way to prevent overfitting

Get more training data!

Another way to prevent overfitting

- Reduce the complexity of our model
  - I.e., reduce the size of the network
- However, large networks can be much more powerful than small networks
- What can we do instead?

Regularization

Weight decay/L2 regularization

- Add a ***regularization term*** to the cost function:

$$C = -\frac{1}{n}\sum_{xj}\left[y_j \ln a_j^L + \left(1 - y_j\right)\ln\left(1 - a_j^L\right)\right] + \frac{\lambda}{2n}\sum_{w} w^2$$

- First term is the cross-entropy cost function
- Second term is the regularization term
  - Scaled by factor $\frac{\lambda}{2n}$, λ the ***regularization parameter***
- Can write regularized cost function as

$$C = C_0 + \frac{\lambda}{2n}\sum_{w} w^2$$

  - C_0 is the unregularized cost

Weight decay/L2 regularization

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- Regularization forces the weights to be small
  - Large weights allowed only if they considerably improve C_0
- I.e., regularization is a compromise between finding small weights and minimizing the cost function C_0
  - λ controls this compromise
  - Small $\lambda \Rightarrow$ we prefer to minimize C_0
  - Large $\lambda \Rightarrow$ we prefer small weights
- How do we apply regularization in gradient descent?

Weight decay/L2 regularization

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- How do we apply regularization in gradient descent?

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}$$

  - Easily computed with backpropagation

Weight decay/L2 regularization

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

- Weight update rule:

$$w \rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} w$$

$$= \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$$

- Same as without regularization except the weight is rescaled by a factor $1 - \frac{\eta \lambda}{n}$
    - Forces the weights to become smaller (weight decay)
    - Other term may cause weights to increase

Weight decay/L2 regularization

- SGD weight update rule

$$w \rightarrow \left(1 - \frac{\eta\lambda}{n}\right) w - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial w}$$

  - C_x is the unregularized cost for training example x
- SGD bias update rule (unchanged)

$$b \rightarrow b - \frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial b}$$

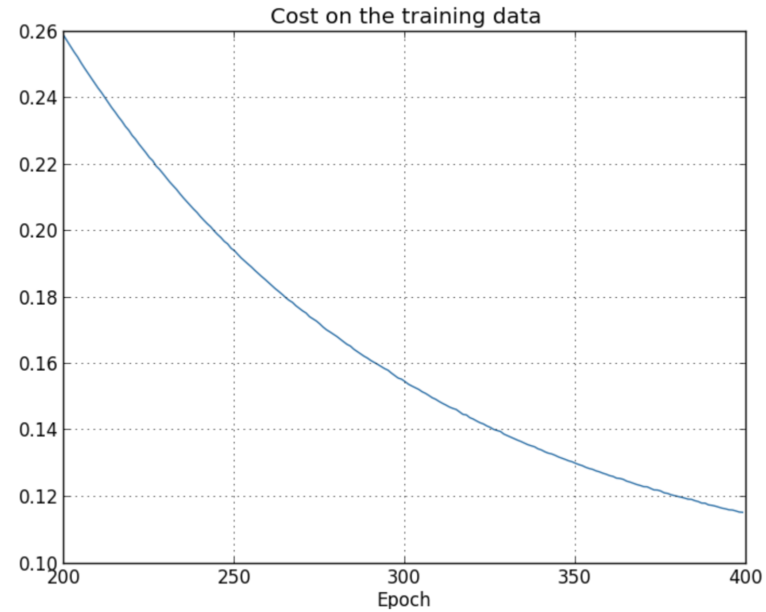
Weight decay/L2 regularization

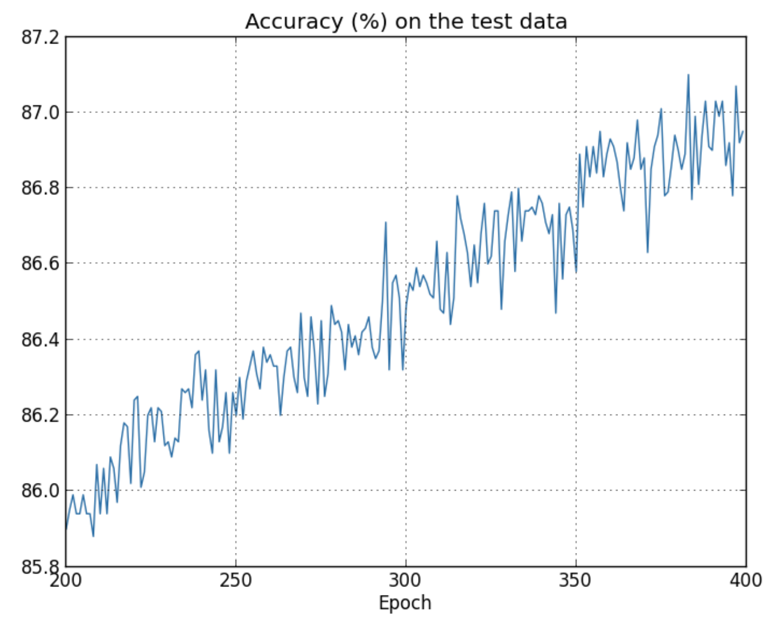
Why not regularize the biases?

1. Empirically, regularizing the biases doesn’t seem to have much positive effect
2. Large biases do not make neurons sensitive to inputs in the same way as large weights
3. Allowing large biases gives the network some flexibility
   - Makes it easier for neurons to saturate which may be desirable in some cases

L2 regularization applied to MNIST

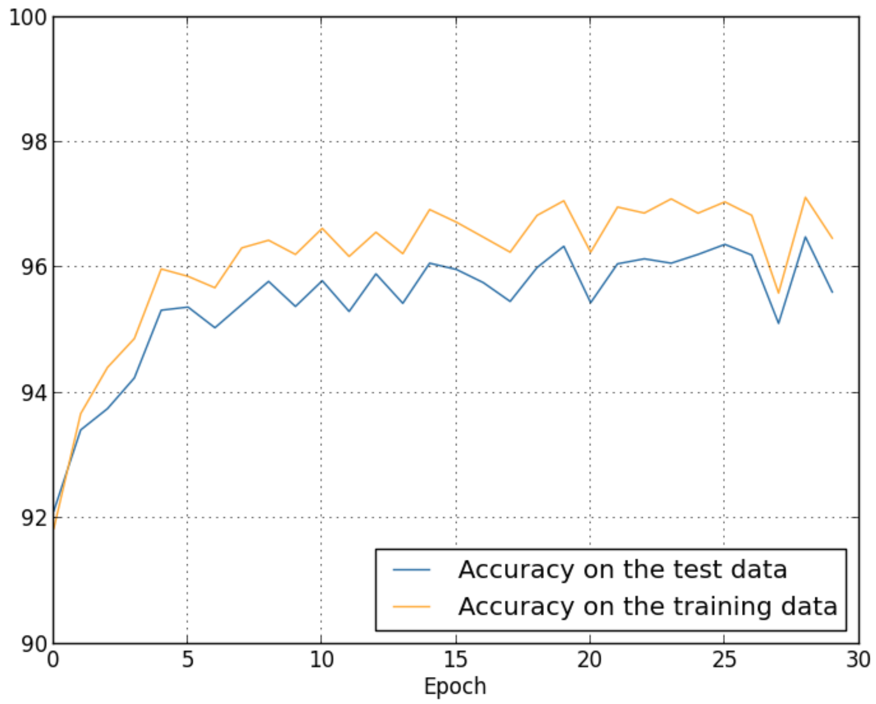
- 30 hidden neurons
- Train with first 1,000 training images
  - Cross-entropy cost
  - Learning rate $\eta = 0.5$
  - Mini-batch size 10
  - $\lambda = 0.1$
- Training cost decreases with each epoch
- Test accuracy continues to increase
  - More epochs would likely improve results further
  - Regularization improves generalization in this case





L2 regularization with more data

- Does regularization help with the 50,000 images?
  - Use same hyper-parameters: 30 epochs, $\eta = 0.5$, mini-batch size of 10
  - Increasing n affects the weight decay factor $1 - \frac{\eta\lambda}{n}$
  - Compensate by changing to $\lambda = 5.0$



Observations

- Test accuracy w/o regularization was **95.49%**
- Test accuracy w/ regularization is **96.49%**
- Gap between test and training accuracy is narrower

L2 regularization with more data

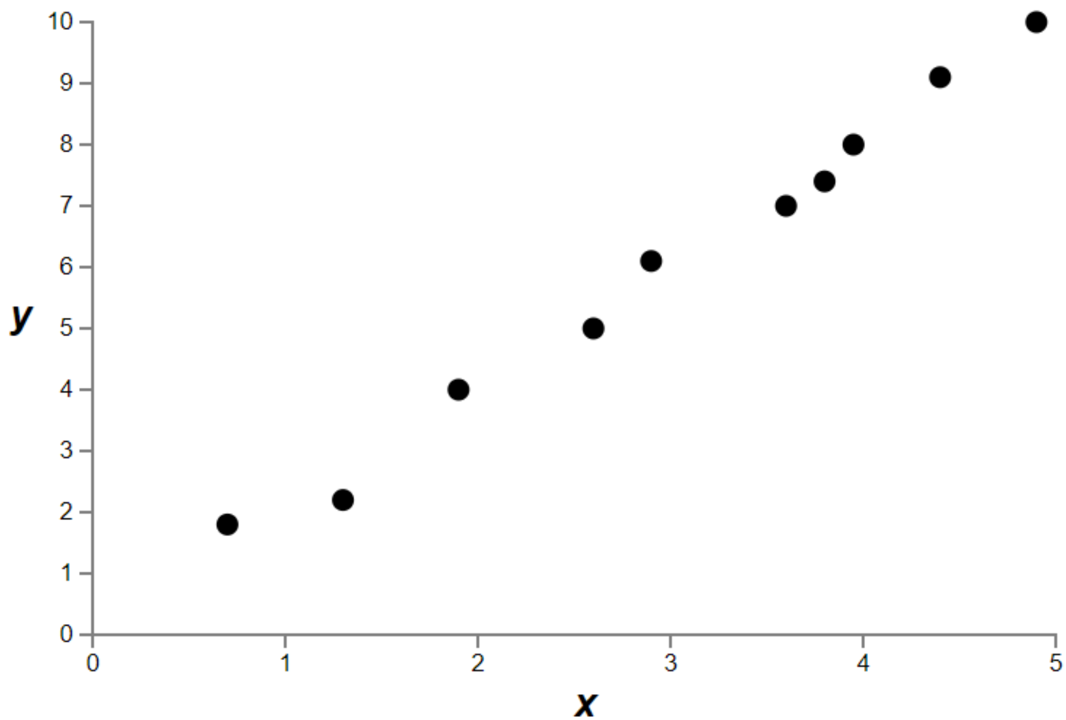
- Increase # of hidden neurons to 100, set $\lambda = 5.0$, $\eta = 0.5$, use cross-entropy cost function, train for 30 epochs
- Resulting accuracy is 97.92%
    - Better than 96.49% with 30 neurons
- Training for 60 epochs and setting $\eta = 0.1$ gives accuracy of 98.04%

Another benefit of regularization

- On this data, regularized runs are much more easily replicated
  - Unregularized runs will sometimes get stuck in local minima under different initializations
- Why is this?
- Possible heuristic: without regularization, the length of the weight vector may grow very large
  - The weight vector is stuck pointing in the same direction (gradient descent only makes tiny changes to direction when length is long)
  - This may make it hard for SGD to properly explore the weight space

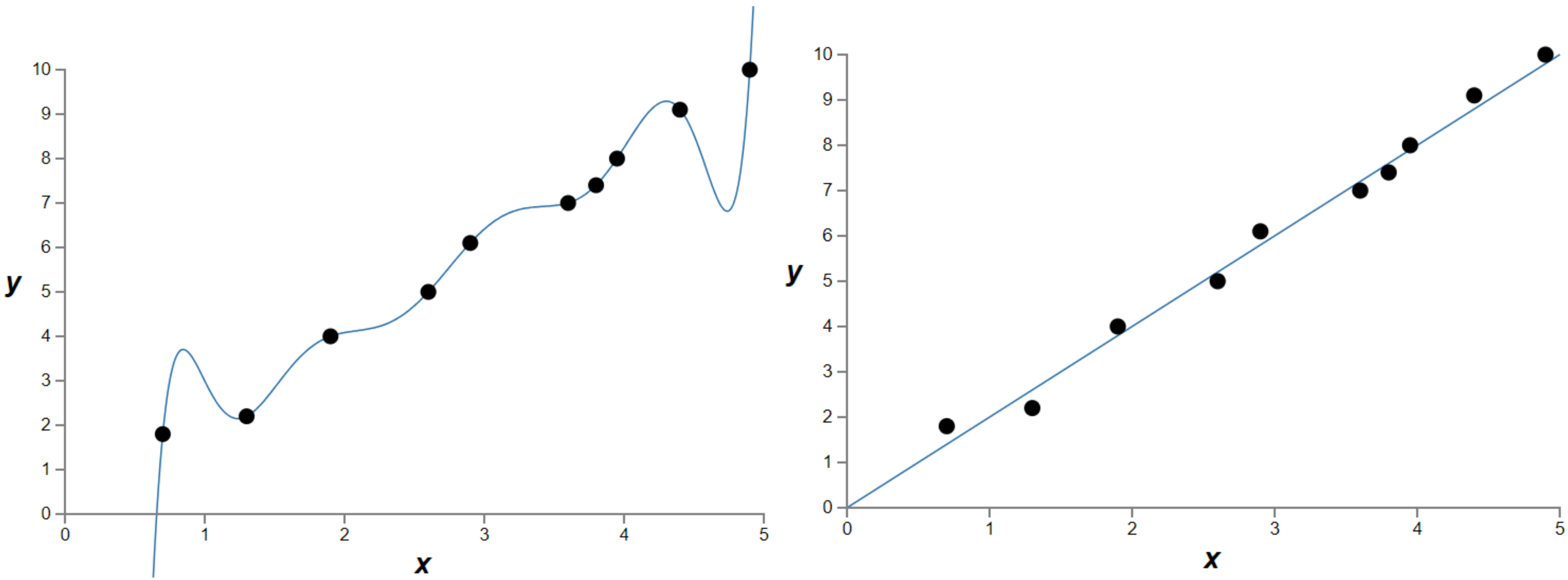
Why does regularization help reduce overfitting?

- Standard story: smaller weights result in lower complexity
- Polynomial regression revisited:



Why does regularization help reduce overfitting?

- Standard story: smaller weights result in lower complexity
- Polynomial regression revisited:



- Which is the better model?

Why does regularization help reduce overfitting?

- Which is the better model?
- Consider two scenarios:
  1. 9^{th} order polynomial best describes the real-world phenomenon
  2. The linear model is correct with some additional noise
- We cannot tell which possibility is correct (or if another possibility is correct)
  - The predictions from each model will be vastly different for a large value of x

One point of view

- Go with the simpler explanation/model (i.e. Occam's Razor)
  - It seems unlikely that a simple explanation occurs by chance
  - From this point of view, the 9th order polynomial is learning the effects of noise
- What does this mean for neural networks?
  - A regularized network has small weights ⇒ the behavior of the network won't change too much for a few random inputs
    - Difficult for regularized network to learn the effects of local noise
    - Regularized network responds to patterns seen often across the training set
  - A network with large weights may change its behavior drastically in response to small changes in the input
- The hope is that regularization forces networks to do real learning and generalize better

Is this point of view correct?

- Occam’s Razor is not a scientific principle
  - There is no *a priori* logical reason to prefer simple explanations over more complex explanations
- Example: Gravity
- In 1859, Urbain Le Verrier discovered that Mercury’s orbit doesn’t exactly match the prediction from Newton’s theory of gravitation
  - Many explanations at the time made small alterations to Newton’s theory
  - In 1916, Einstein showed that general relativity, a much more complex theory, explained the deviations
  - Today, Einstein’s theory is accepted as correct, largely because it explains and predicts phenomena not explained or predicted by Newton’s theory

3 Morals

1. It can be difficult to decide which explanation is "simpler"
2. Even if we can decide, simplicity may not be the best guide
3. The true test of a model is its ability to predict new phenomena

- Despite this caveat, empirically, regularized networks usually generalize better than unregularized networks
- Yet the story about gravity illustrates why there isn't a completely convincing theoretical explanation for why regularization works

How do we generalize?

- Humans generalize very well, despite using a system (the brain) with a huge number of free parameters
  - A child can learn to recognize an elephant quite well from only a few images
  - In some sense, our brains regularize well
- How do we do it?
  - We don’t know
  - Developing techniques that generalize well from small data sets is an active area of research

Generalization of neural networks

- Our unregularized neural networks actually generalize quite well
  - Network with 100 hidden neurons has 80,000 parameters
  - Training on 50,000 images is like fitting a 80,000 degree polynomial to 50,000 data points
- Why doesn’t our network overfit terribly?
  - One conjecture is that “the dynamics of gradient descent learning in multilayer nets has a ‘self-regularization’ effect” (LeCun et al., 1998)
  - This is fortunate, but kind of troubling that we don’t understand why
- Meanwhile, regularization is highly recommended

L1 regularization

- Add the sum of absolute values

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|$$

- L1 and L2 names come from the respective norms:

$$\|w\|_1 = \sum_w |w|$$

$$\|w\|_2^2 = \sum_w w^2$$

- L1 regularization also prefers small weights
- How does it differ from L2 regularization?

L1 regularization

- Partial derivative of the cost function:
  - is the sign of w:

$$\mathrm{sgn}(w) = \begin{cases} +1, & w > 0 \\ -1, & w < 0 \\ 0 & w = 0 \end{cases}$$

- Update rule for L1 regularization

$$w \rightarrow w - \frac{\lambda \eta}{n} \mathrm{sgn}(w) - \eta \frac{\partial C_0}{\partial w}$$

- Compare to update rule for L2 regularization

$$w \rightarrow \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$$

L1 regularization

- L1 regularization

$$w \rightarrow w - \frac{\lambda\eta}{n}\operatorname{sgn}(w) - \eta\frac{\partial C_0}{\partial w}$$

- L2 regularization

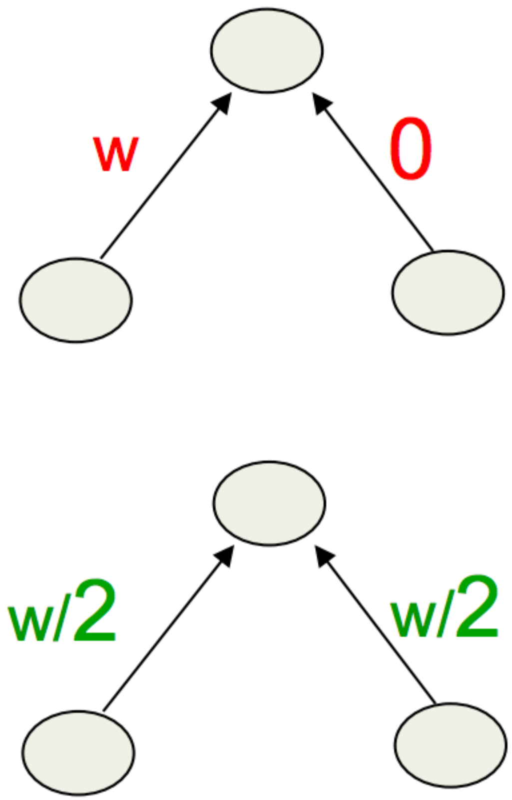
$$w \rightarrow \left(1 - \frac{\eta\lambda}{n}\right)w - \eta\frac{\partial C_0}{\partial w}$$

- Both shrink the weights
  - L1 shrinks the weights by a constant amount
  - L2 shrinks the weights by amount proportional to w
- If $|w|$ is large, L1 shrinks the weight much less than L2
- If $|w|$ is small, L2 shrinks the weight much less than L1

L1 regularization

- If $|w|$ is large, L1 shrinks the weight much less than L2
- If $|w|$ is small, L2 shrinks the weight much less than L1
- Net result: L1 concentrates the weights in a relatively small number of connections
  - Can result in a ***sparse*** number of connections if λ is big enough
- Sparsity can be very desirable
  - Improved computational speed
  - Improved interpretation

L2 Smoother Model



- Prefers to share smaller weights
- Makes model smoother
- More Convex

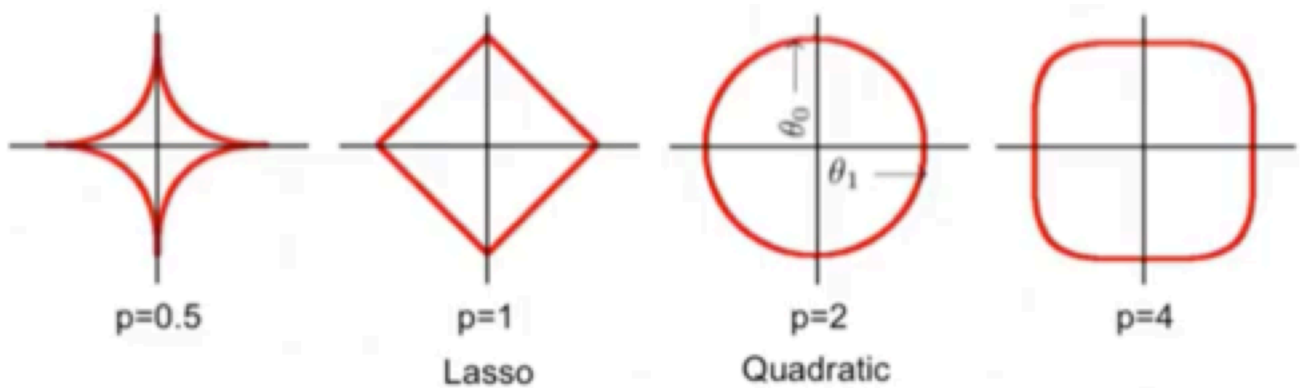
L1 vs L2

L2 regularization	L1 regularization
Computational efficient due to having analytical solutions	Computational inefficient on non-sparse cases
Non-sparse outputs	Sparse outputs
No feature selection	Built-in feature selection

Lp Regularization

$$L_p = (\sum_i w_i^p)^{1/p}$$

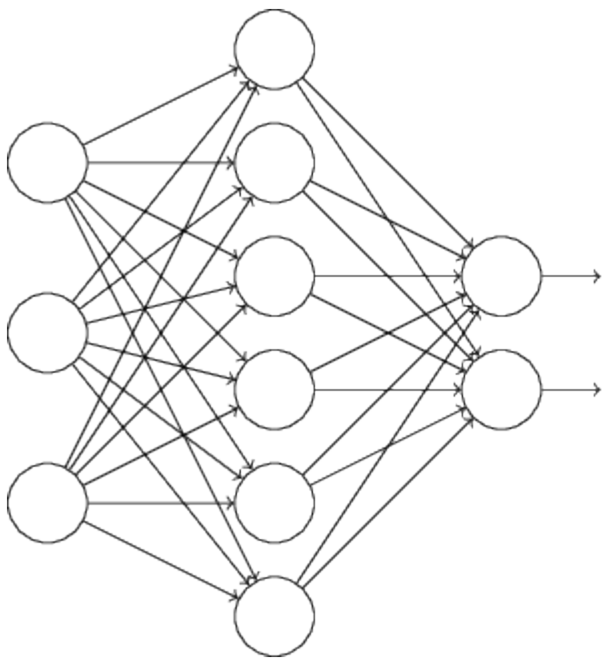
Isosurfaces



As p→0 you get a counting penalty

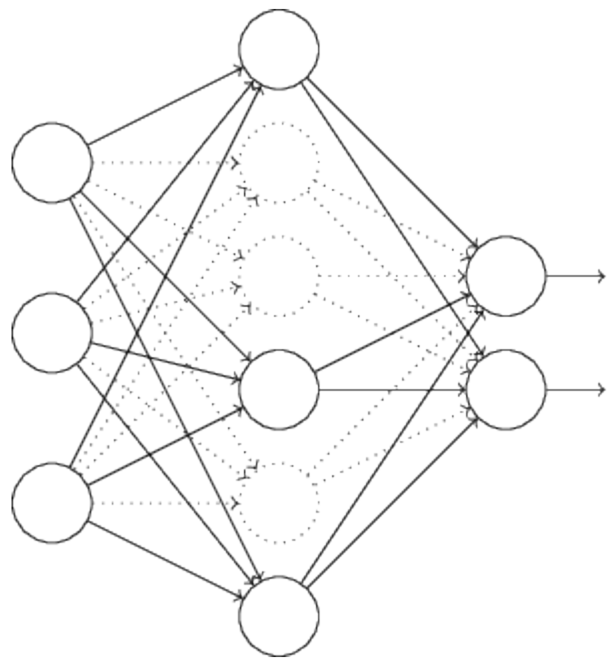
Dropout

- L1 and L2 regularization directly modify the cost function
- With dropout, we modify the network instead
- Standard network training on input x and desired output y
    - Forward propagate x and then backpropagate to get gradient



Dropout

- With dropout, start by randomly and temporarily deleting half the hidden neurons
  - Forward propagate x and backpropagate to get gradient
  - Update weights and biases over a mini-batch
- Repeat by restoring the dropout neurons and removing a different subset



Dropout

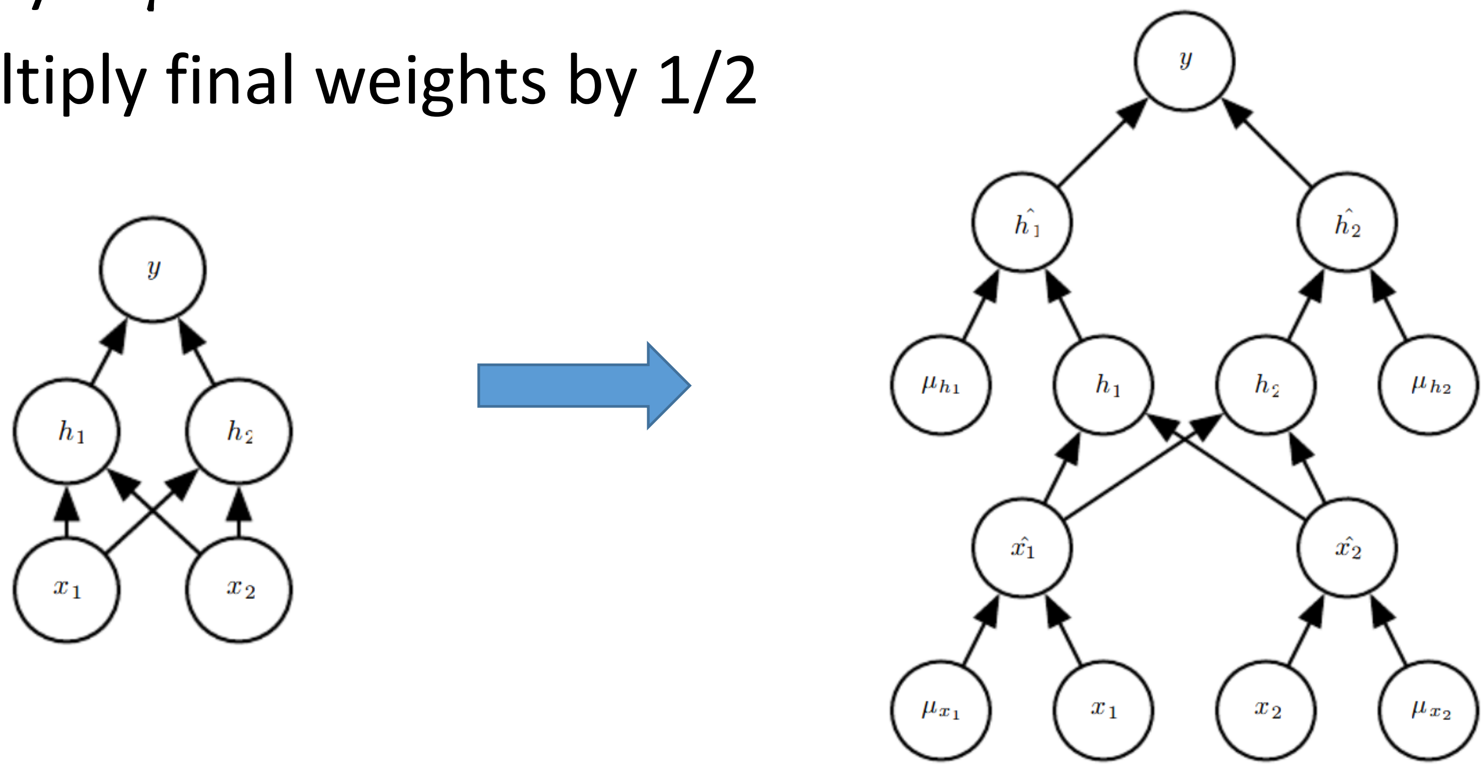
- Repeating this process over and over gives a set of learned weights and biases
- How does this help with regularization?
- Imagine we train several different neural networks using the same training data under different initializations
    - Different networks will overfit in different ways
    - We can average the results or do a majority vote
    - E.g., if 3/5 networks say a digit is a “3”, the other two networks are probably mistaken
- Averaging over multiple networks can be a powerful (and expensive) way to reduce overfitting
- Dropout is a lot like training different neural networks
    - Net effect of dropout generally reduces overfitting

Dropout

- Another explanation: “This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.” (Krizhevsky et al., 2012)
- I.e., dropout is forcing the prediction model to be robust to the loss of an individual node
  - Somewhat similar to L1 and L2 regularization which reduce weights (making the network more robust to losing an individual connection)
- Dropout also works empirically, especially when training large, deep networks

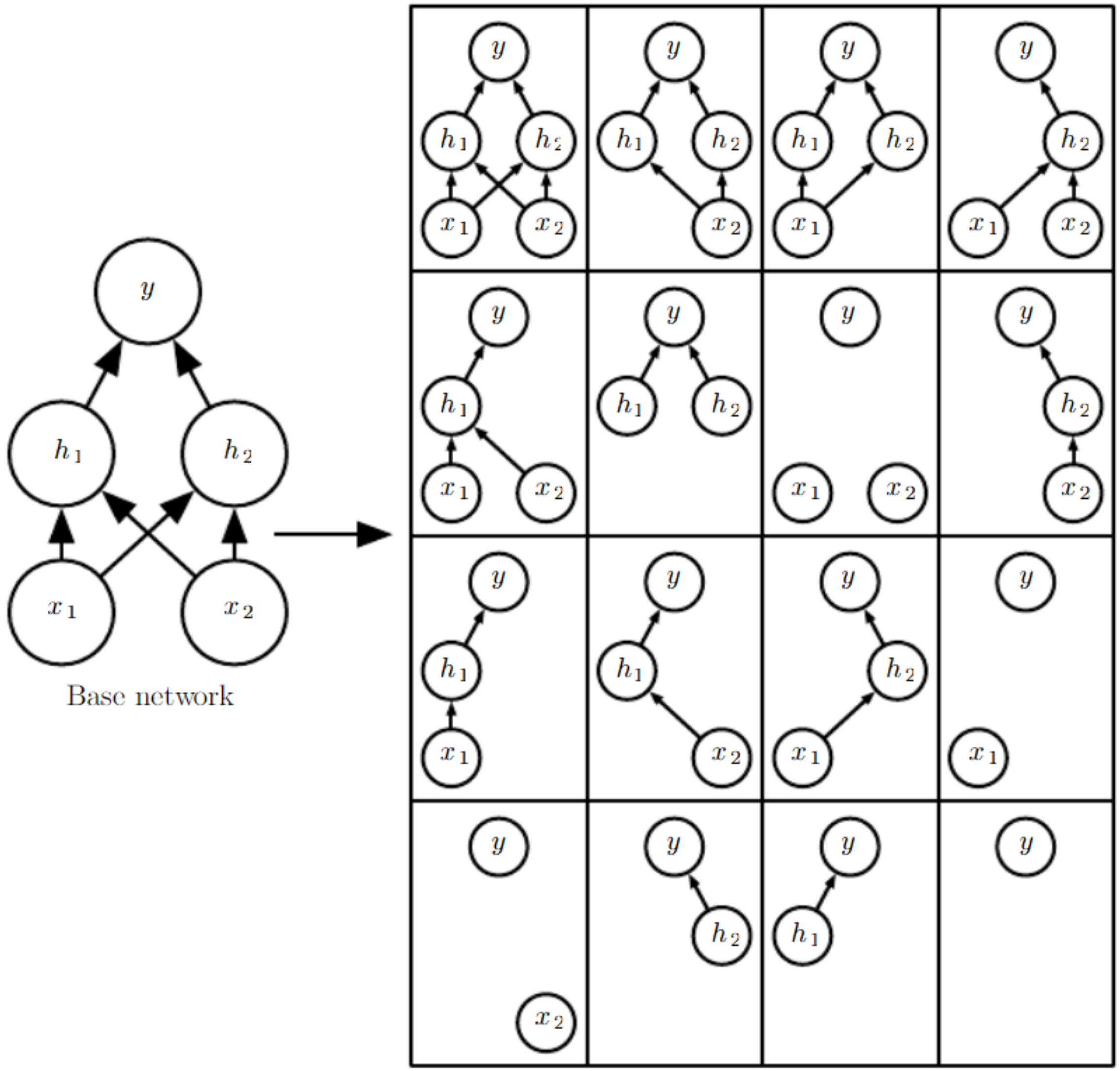
Implementing dropout

- Generate a binary random vector μ
  - Probability of each entry being 1 is a hyperparameter
- Multiply the output of each node with the corresponding entry in μ
- Multiply final weights by 1/2



Implementing dropout

- Equivalent to randomly selecting one of the following sub-networks



Base network

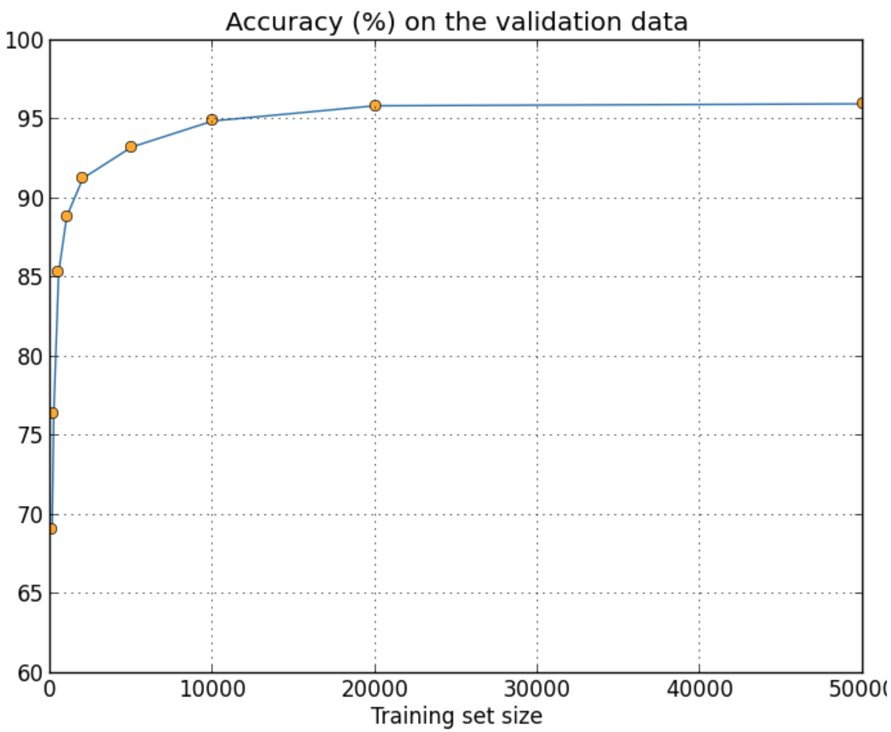
Ensemble of subnetworks

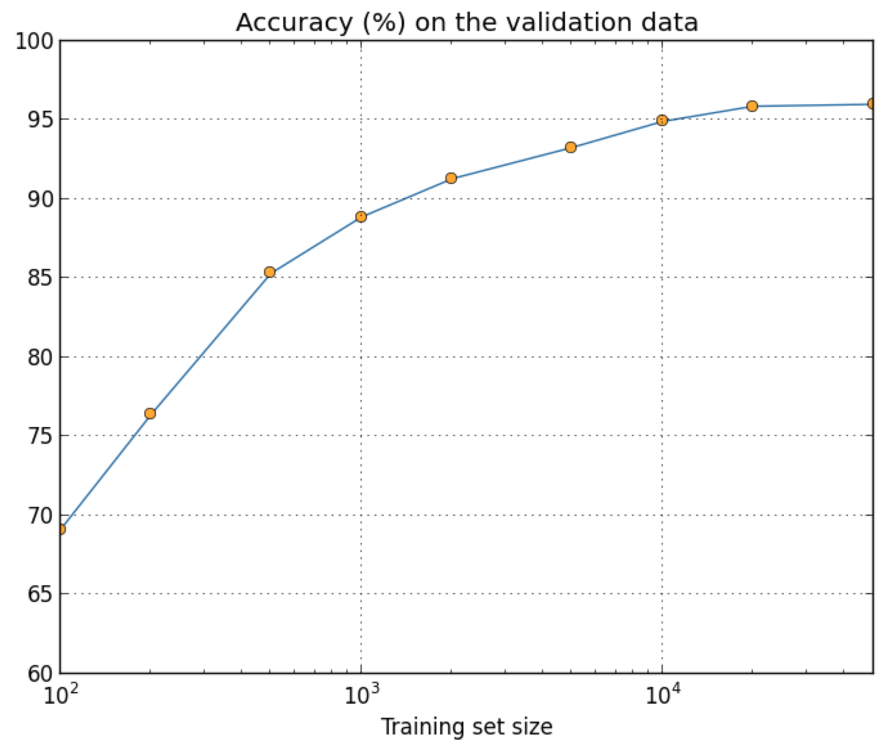
When should dropout be used?

- Dropout can be applied to nearly all models
  - Feedforward networks, probabilistic models, RNNs, etc.
  - Other regularization techniques may not be applicable in these cases
- For very large datasets, dropout (and regularization in general) doesn’t help much
- Dropout is less effective when using very small sample sizes
- See section 7.12 in the Goodfellow et al. book for more information

Artificially increasing the training data

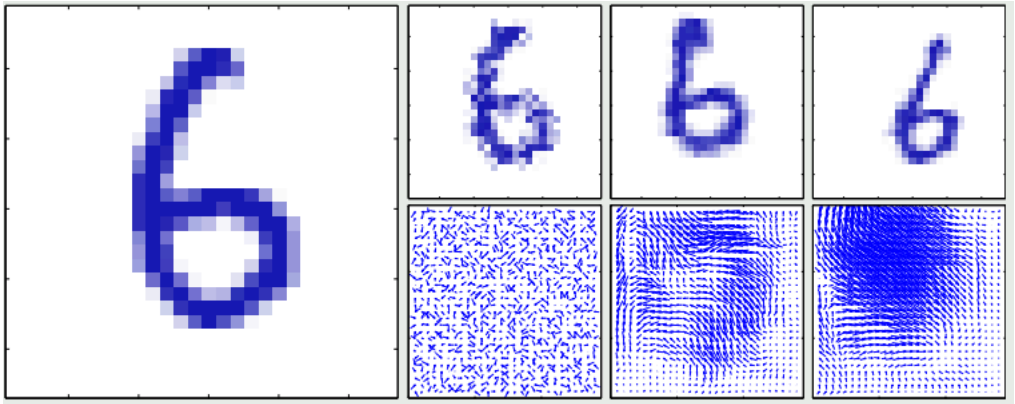
- Saw earlier that our MNIST classification accuracy decreased dramatically with only 1,000 training images
- How does accuracy improve as a function of sample size?





Input Augmentation

- Artificially transform inputs in ways that you don't want the neural network to care about
  - Does not change the label for classification



Artificially increasing the training data

- MNIST results from Simard et al. (2003)
  - Feedforward network with 800 hidden neurons and cross-entropy cost function
- Accuracy on standard dataset: 98.4%
- Applied rotations and translations: 98.9%
- Applied “elastic distortions” as well
  - Image distortion intended to emulate random oscillations in the hand muscles
  - Accuracy: 99.3%

Artificially increasing the training data

- General principle: expand the training data by applying operations that reflect real-world variation
- Example: speech recognition
  - Add background noise
  - Speed it up
  - Slow it down
- Alternatively, could do preprocessing to remove these effects
  - May be more efficient in some cases

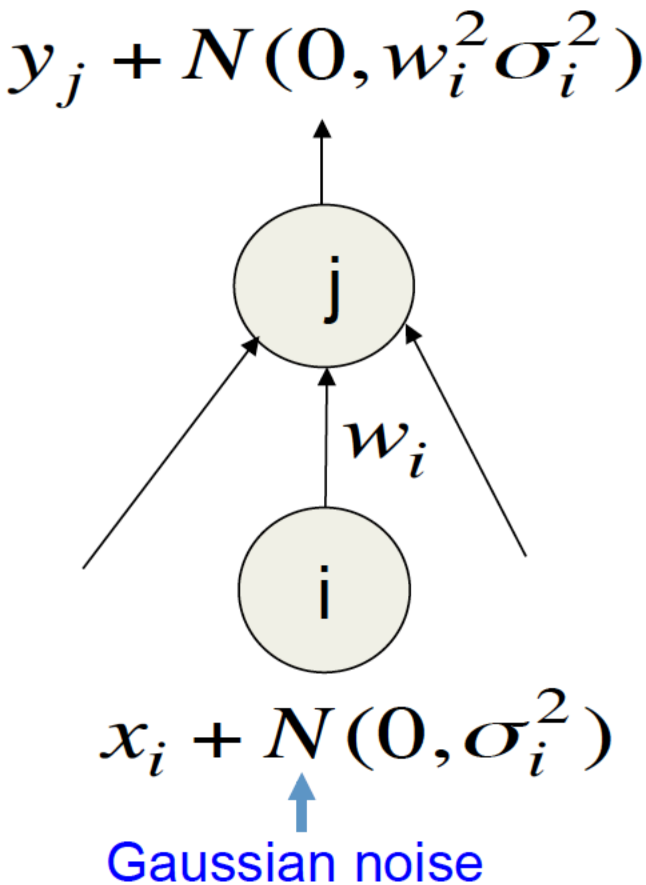
Noise robustness

Adding noise is a form of regularization

1. Add noise to the inputs
   - Can be viewed as increasing the training data
2. Add noise to the hidden layers
3. Add noise to the weights
4. Adding noise at the output layer
   - Can reflect noise or mistakes in the labels
   - Can be modeled explicitly in the cost function

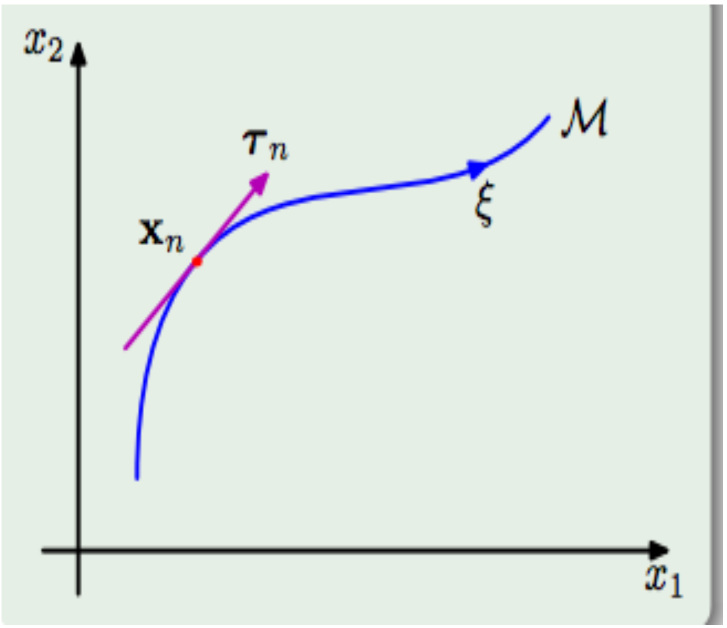
- Dropout is a form of multiplicative noise

Input Gaussian Noise



Minimizing squared error minimizes square weights

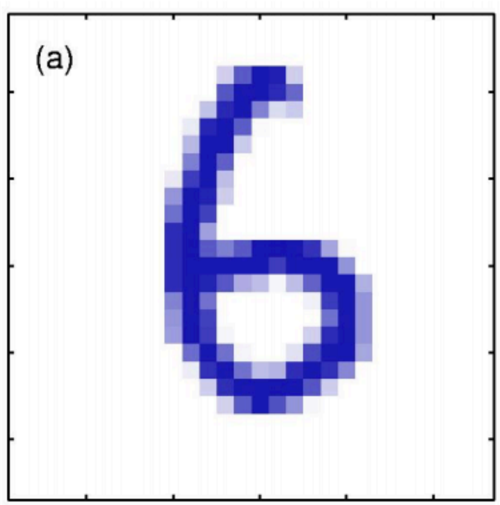
Tangent Propagation



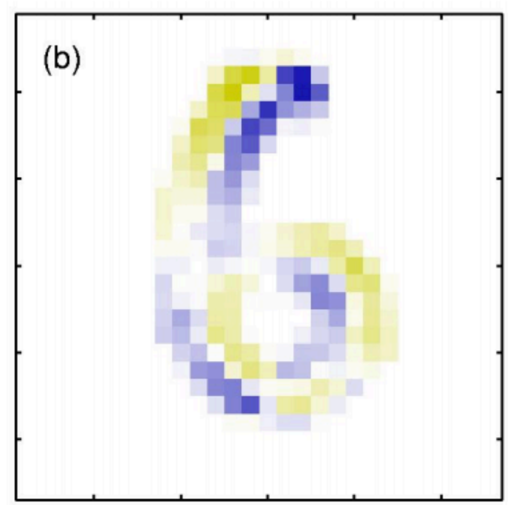
- Invariance to data transformations
- Penalize using the jacobian

$$\left.\frac{\partial y_k}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} \frac{\partial y_k}{\partial x_i} \left.\frac{\partial x_i}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} J_{ki} \boldsymbol{\tau}_i$$

Original Image $\mathbf{x}$



Tangent vector τ corresponding to small rotation



Penalizing Entropy

REGULARIZING NEURAL NETWORKS BY PENALIZING CONFIDENT OUTPUT DISTRIBUTIONS

Gabriel Pereyra [*†]
Google Brain
pereyra@google.com

George Tucker [*†]
Google Brain
gjt@google.com

Jan Chorowski
Google Brain
chorowski@google.com

Łukasz Kaiser
Google Brain
lukaszkaiser@google.com

Geoffrey Hinton
University of Toronto & Google Brain
geoffhinton@google.com

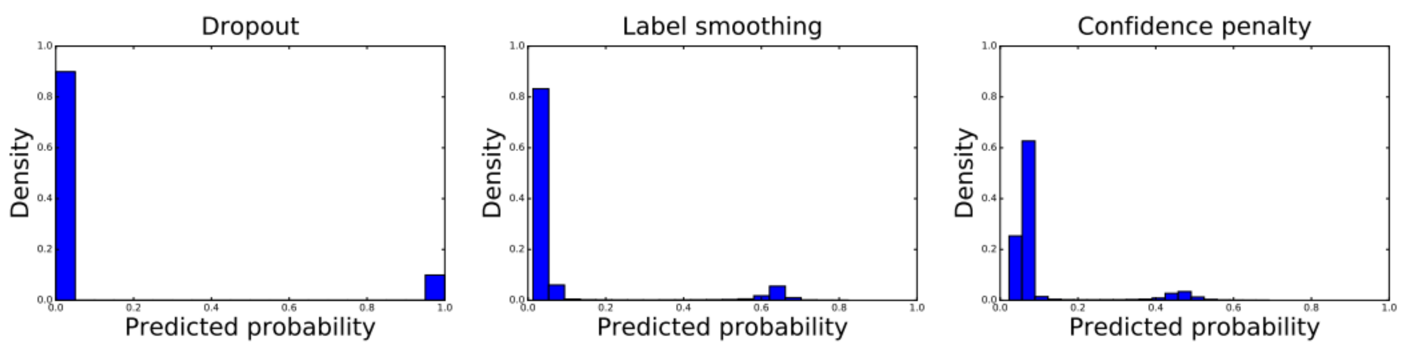
Maximum Entropy Principle

- E.T. Jaynes 1957
- The probability distribution that best represents the state of knowledge is the one with largest entropy
- Consider all probability distributions that fit prior data, choose the one with maximum entropy

Penalizing Output distribution

$$H(p_\theta(\boldsymbol{y}|\boldsymbol{x})) = -\sum_i p_\theta(\boldsymbol{y}_i|\boldsymbol{x}) \log(p_\theta(\boldsymbol{y}_i|\boldsymbol{x})).$$

Prevent it from putting all labels to one class



Summary

- Overfitting is a major problem in neural networks, especially large networks
- Regularization is a powerful technique for reducing overfitting
- Regularization is an active area of research
- Many modern architectures are based on some novel form of regularization
- We will see regularization again

Further reading

- Nielsen book, chapter 3
- Goodfellow et al., chapter 7