Deep Learning Theory and Applications

Regularization

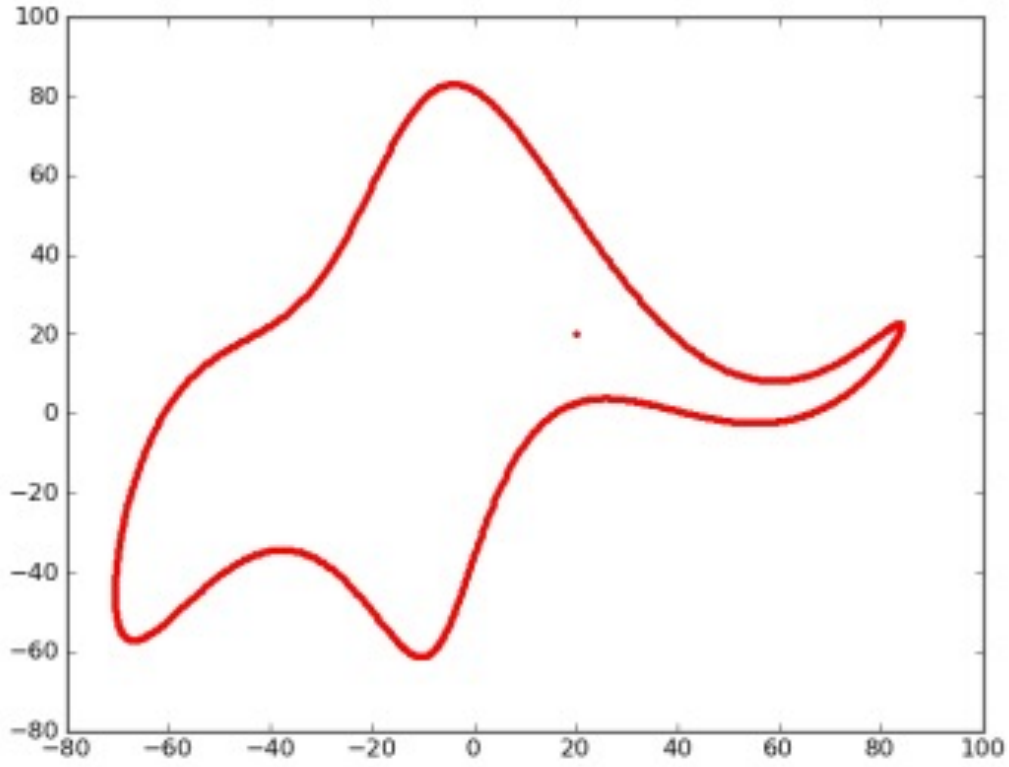
Yale

CPSC/AMTH 663

Outline

1. Overfitting
2. Stability
3. Regularization
4. Regularization techniques
    1. L1/L2 Regularization
    2. Dropout
    3. Augmenting the training data
    4. Noise robustness
    5. Tangent Propogation

Free parameters



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

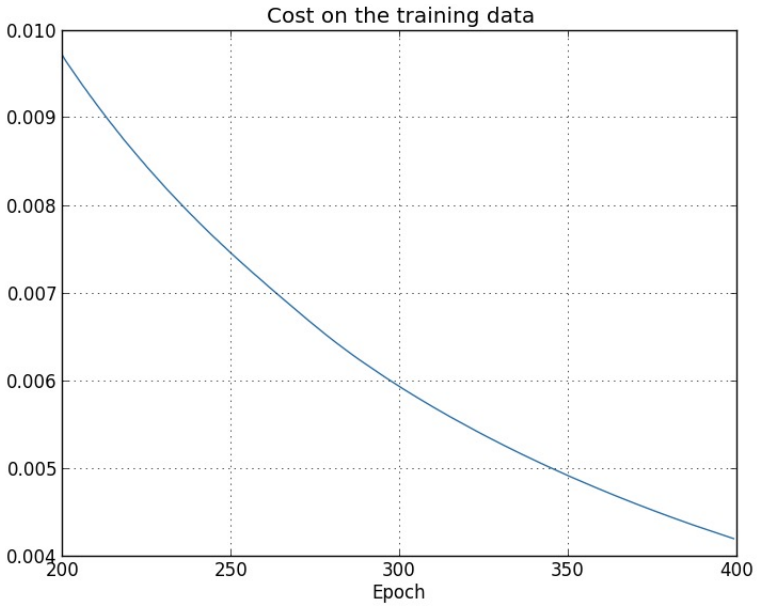
With four parameters I can fit an elephant, and with five I can make him wiggle his trunk. -- von Neumann.

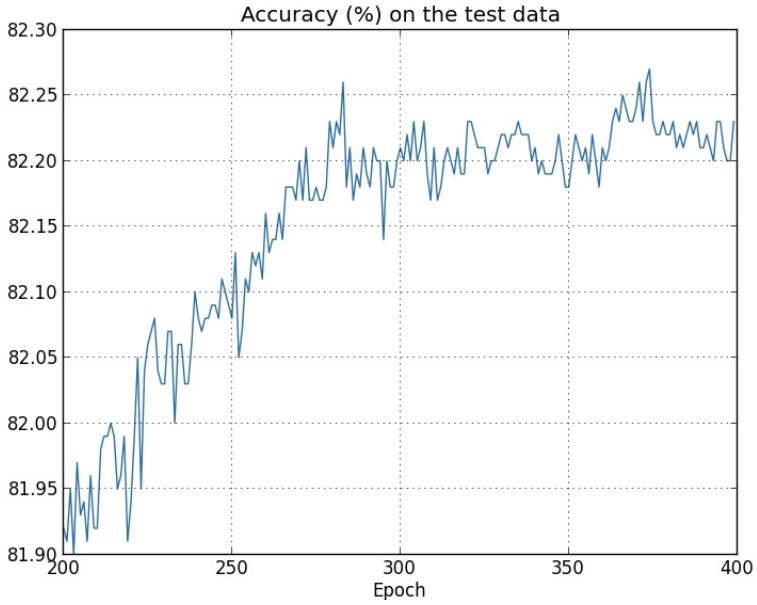
Free parameters

- 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
- The true test: can the model make accurate predictions on new data?

MNIST recognition revisited

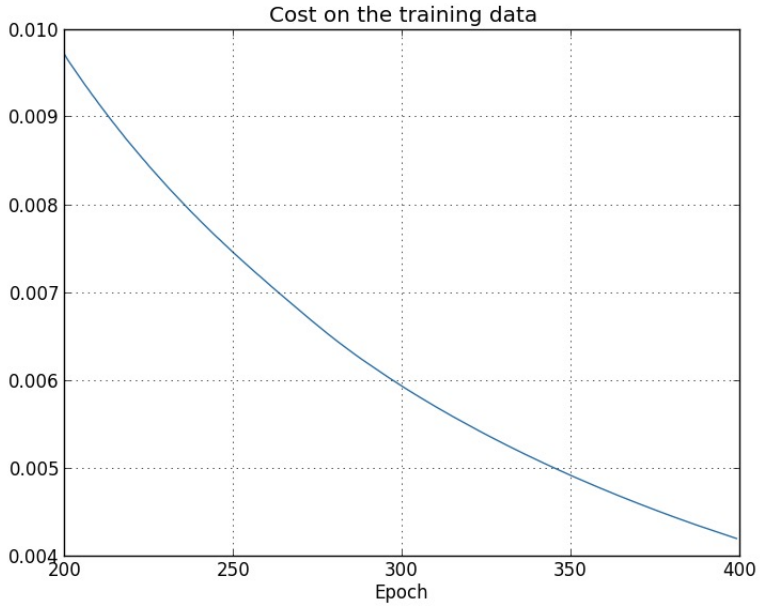
- 30 hidden neurons
- Train with first 1,000 training images
  - Cross-entropy cost
  - Learning rate $\eta = 0.5$
  - Mini-batch size 10
  - 400 epochs
- Training cost decreases with each epoch
- But accuracy flatlines $\approx$ epoch 280
  - The learned network is not generalizing well
  - The network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

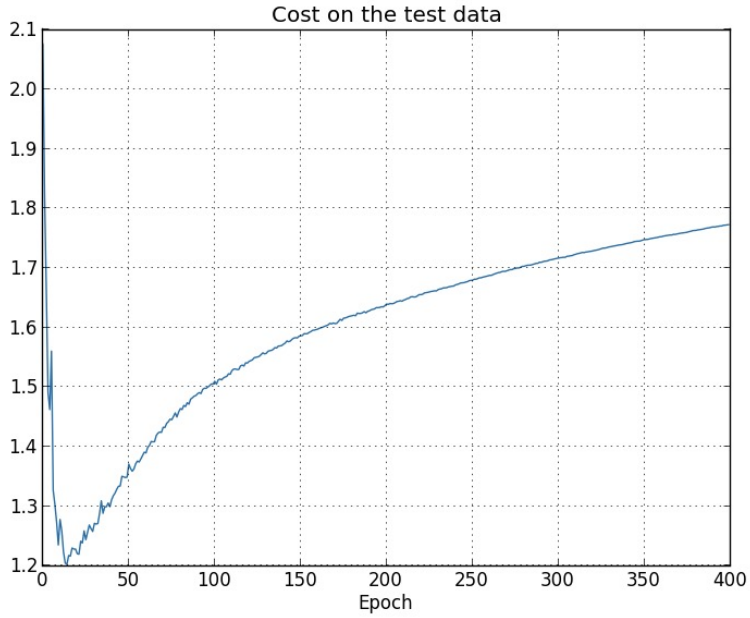




Cost vs classification accuracy

- 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?

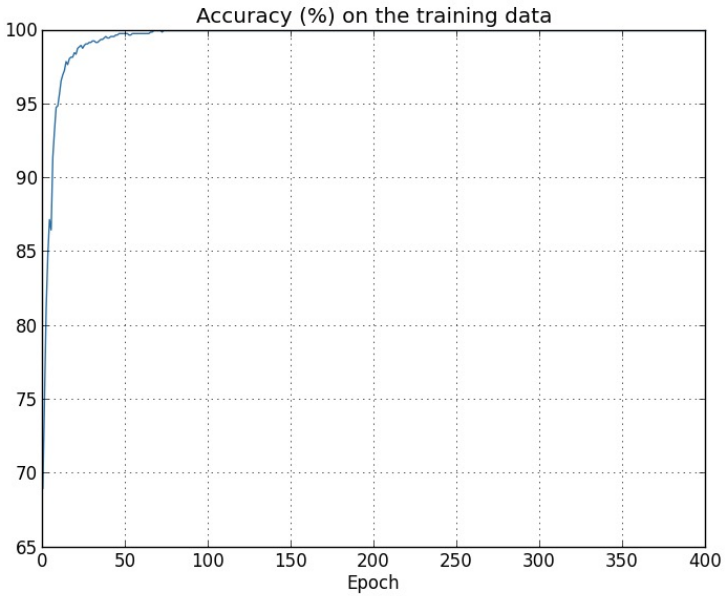


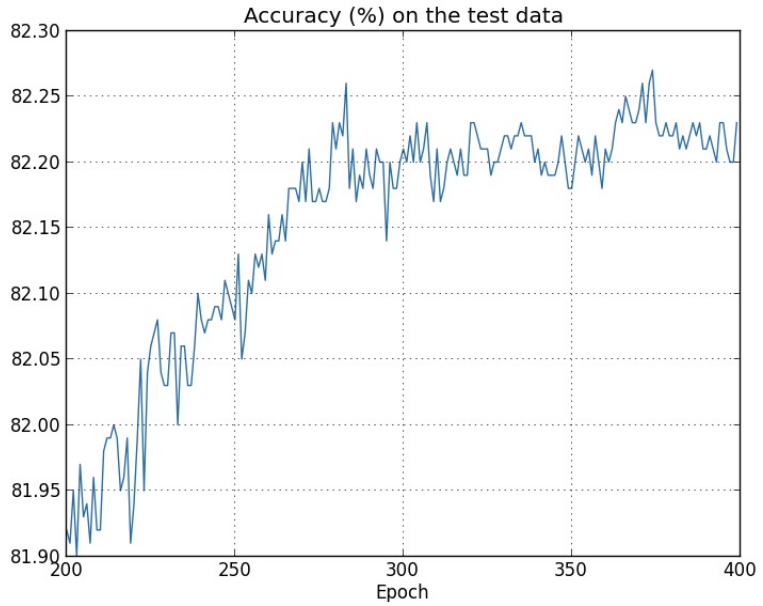


Cost vs classification accuracy

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

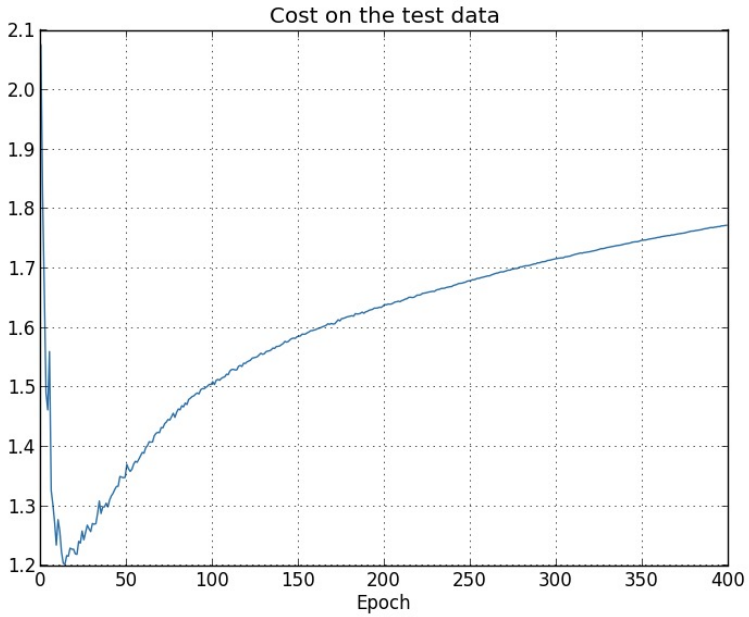
but we want generalization

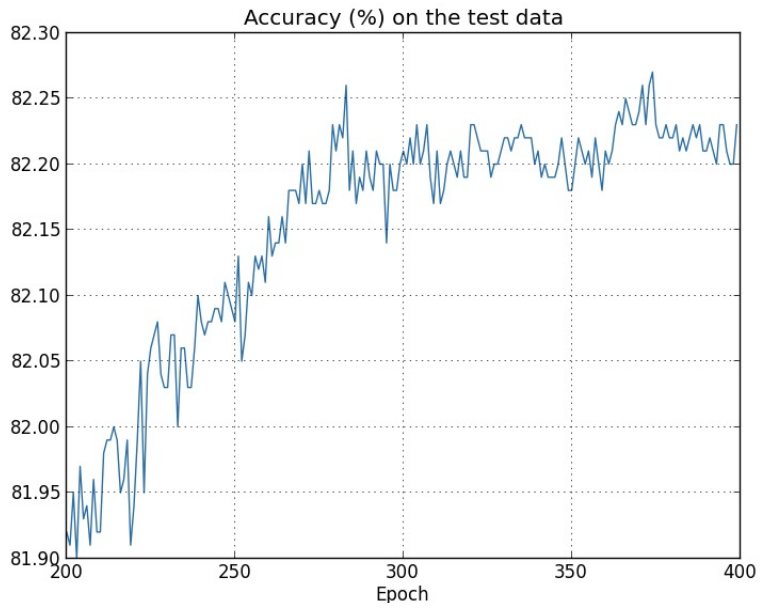




When does overfitting occur?

- Epoch 15 or epoch 280?





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

can think of as a type of regularization

Stability improves overfitting

- Adding or taking off a few inputs should not change the weights much
- Solution must be stable to small inputs and input perturbations
- SGD is naturally stable because it operates at a batch level, but does better with more training data
- To increase stability with SGD, get more data so that no one input pulls weights too far
- Another way is to stop before convergence so that the model doesn't spend too much time working on the last few examples

Early stopping

- 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***

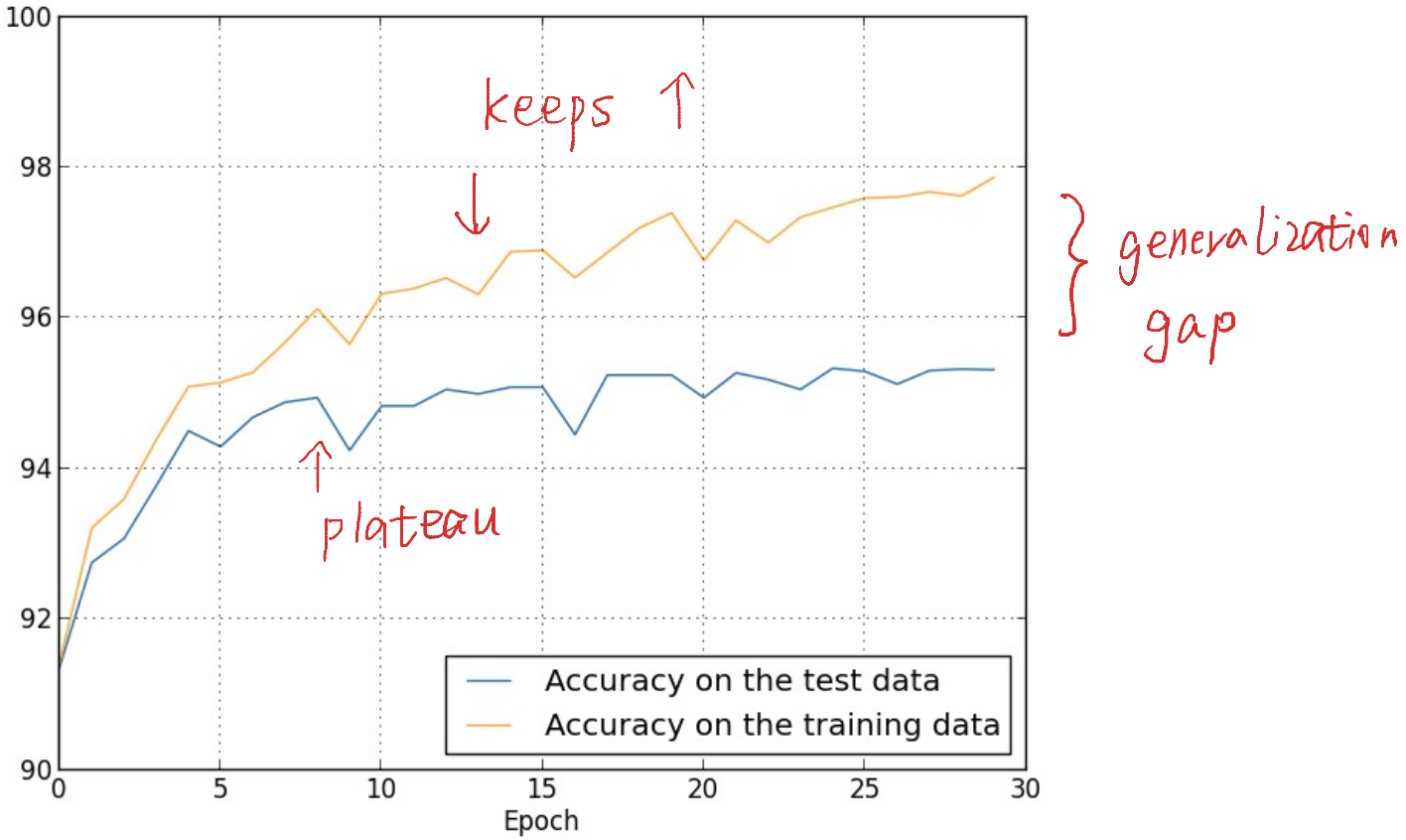
Implementing early stopping

- 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

Handwritten digit recognition revisited

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

Yes



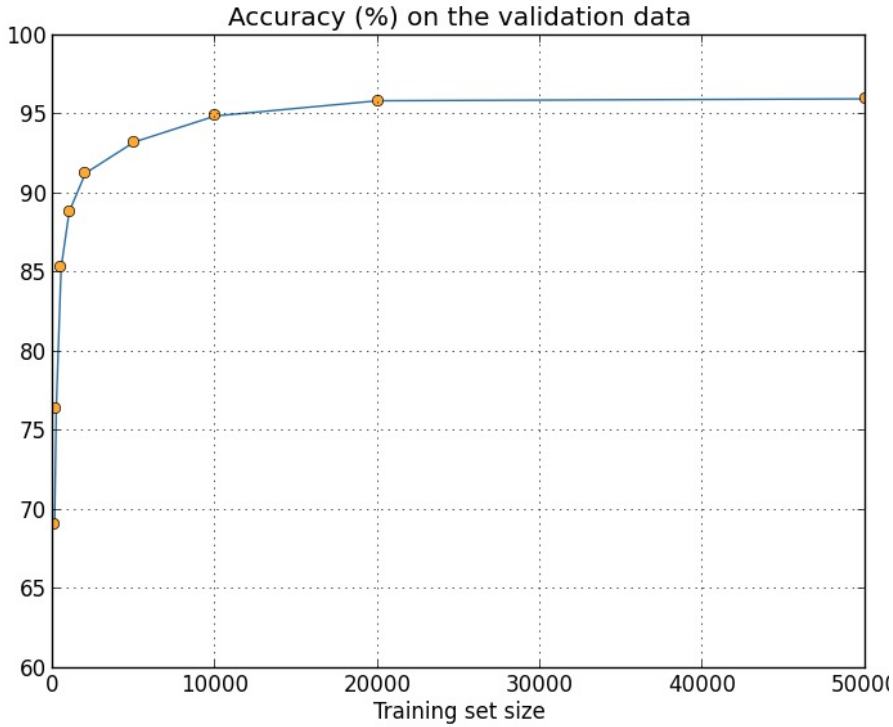
- Some, but not nearly as much as before

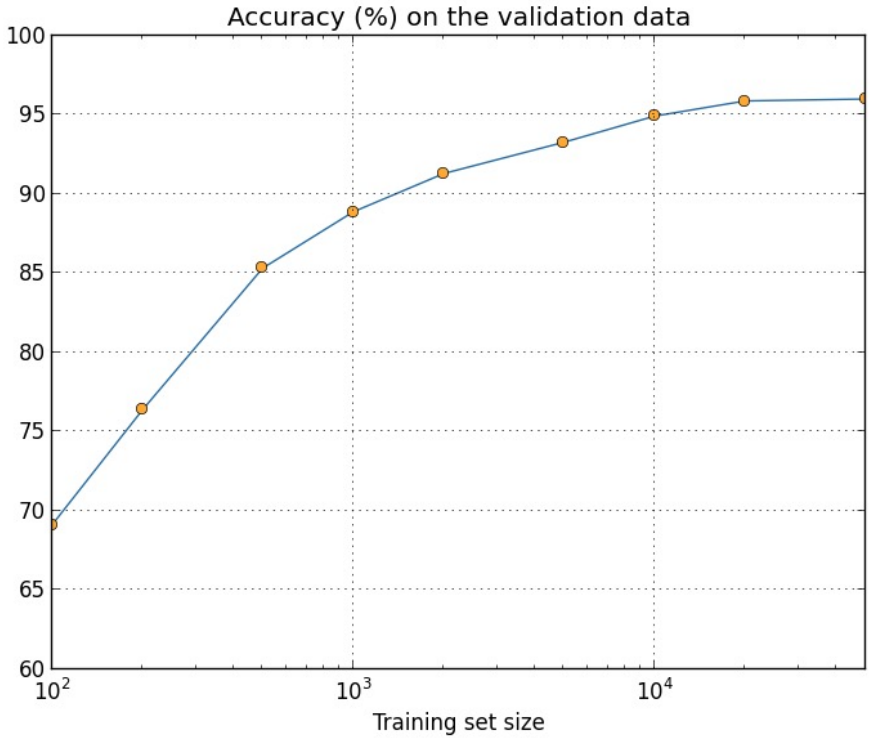
A great way to prevent overfitting

Get more training data!

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?



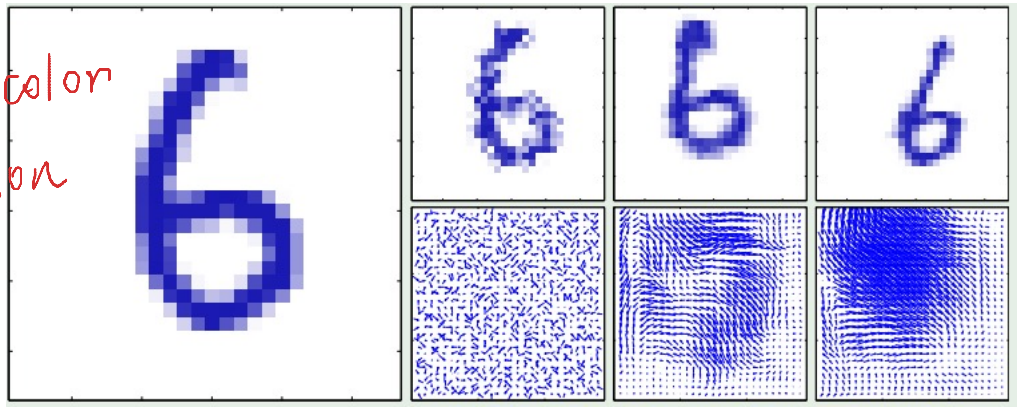


Input Augmentation

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

eg. 6

add noise
rotate
smear
change color
translation
stretch
flip



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

• change emotion 声调 tone

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

If we don't trust label

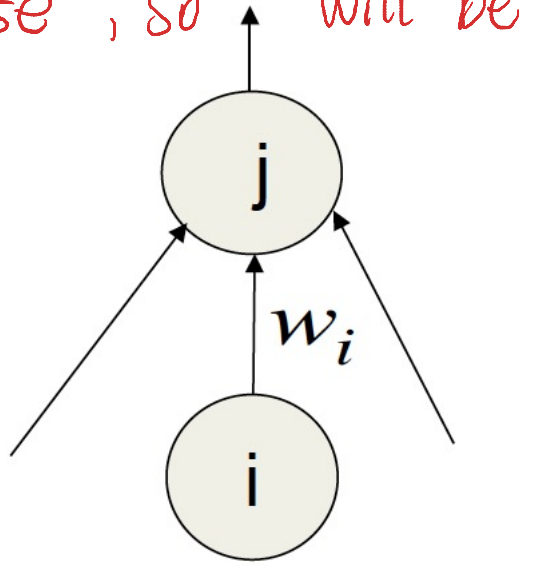
- Dropout is a form of multiplicative noise

Input Gaussian Noise

effect of regularization similar to L2 regularization

reason: large weight of certain variable will be amplified as multiply by noise, so will be shrink

$$y_j + N(0, w_i^2 \sigma_i^2)$$



$$x_i + N(0, \sigma_i^2)$$

add Gaussian noise

Minimizing squared error minimizes square weights

Regularization

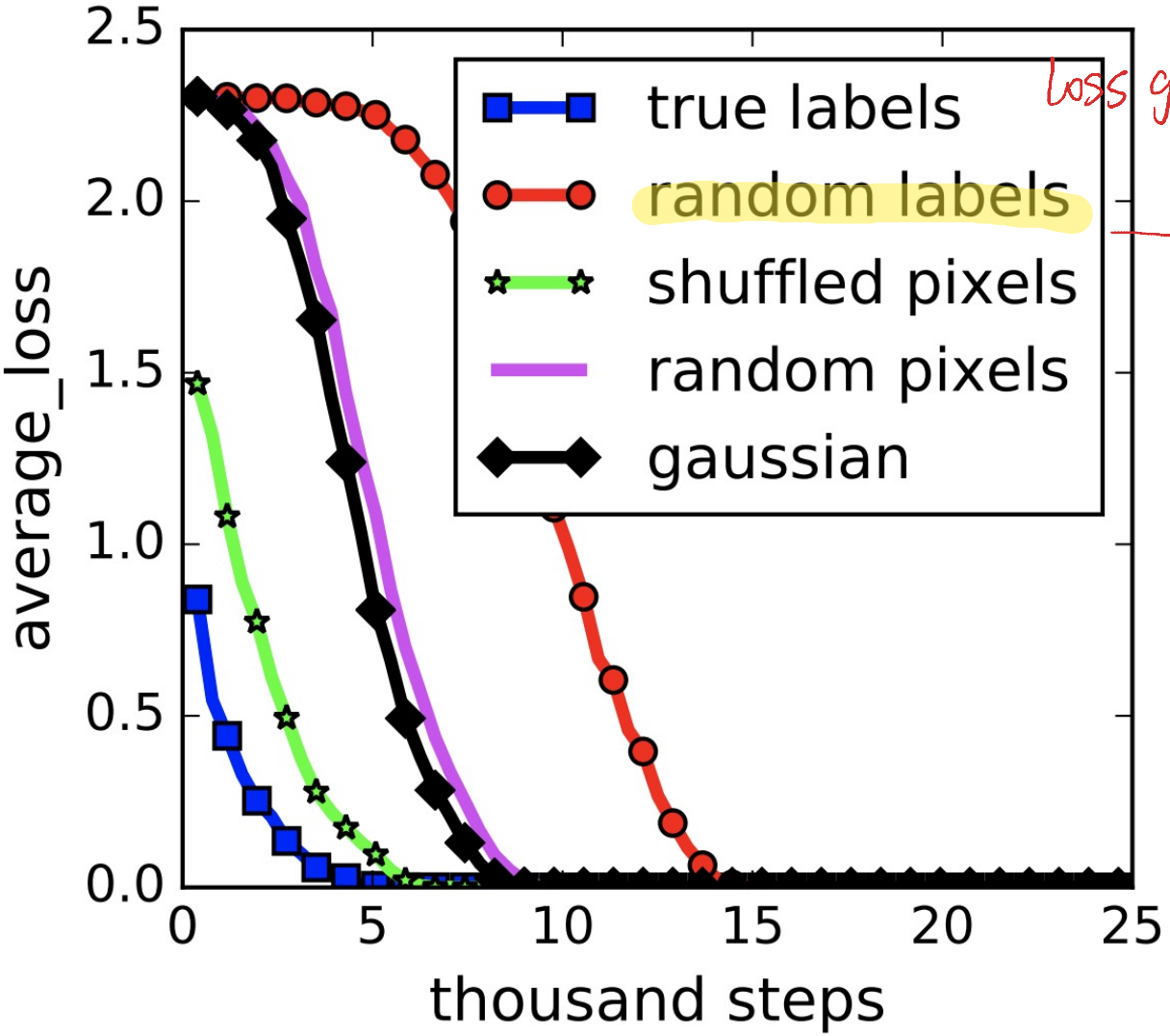
complexity / expressness ↑

- Traditionally regularizations decrease capacity of the model
- But neural networks have super high capacity!

because ML model has much less params than sample size

thus shrink some weights don't influence entire richness of params, but give model robustness to noise, outliers and inductive bias

Memorization = fitting to random data

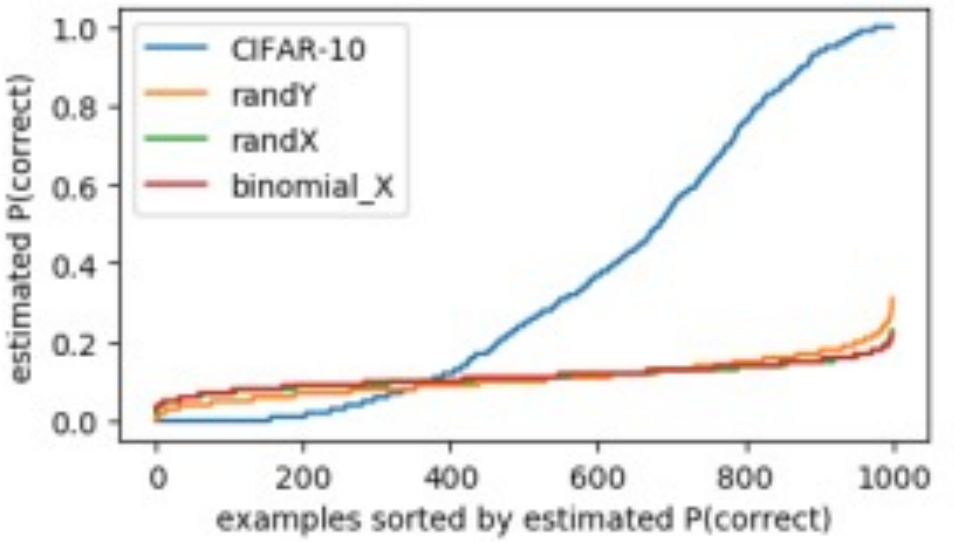


loss go down most quickly

still fit, means NN has capacity to learn no-pattern (memorize) from data

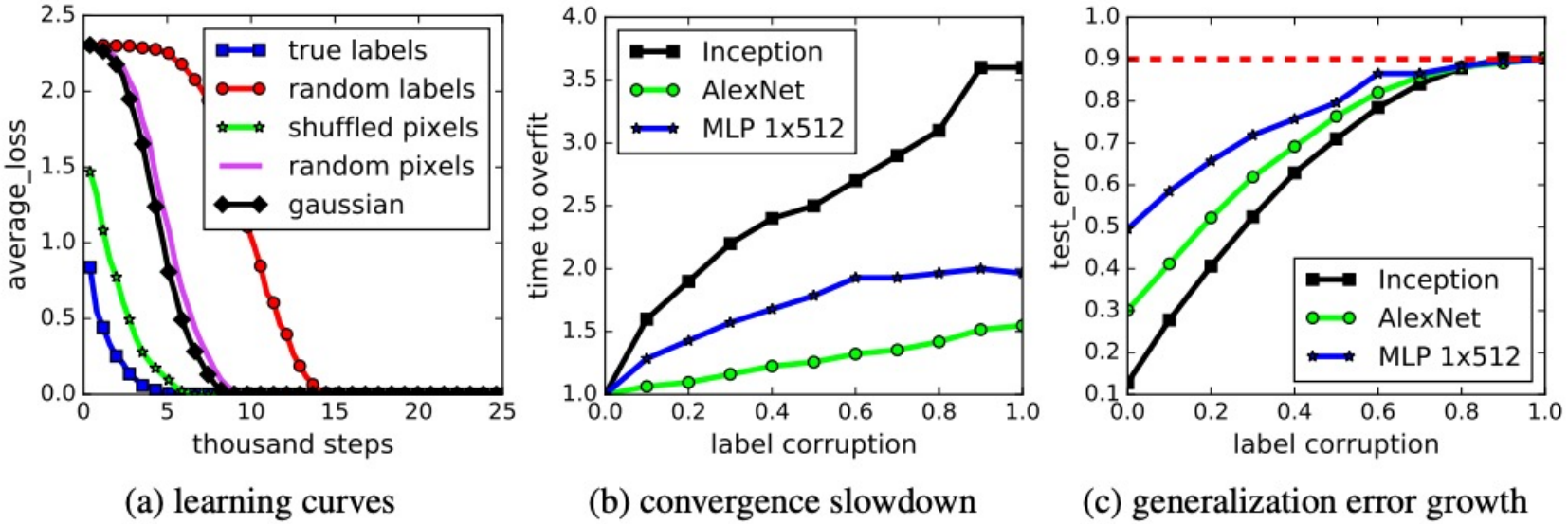
[Zhang et al. ICLR 2017]

Real data generalizes better



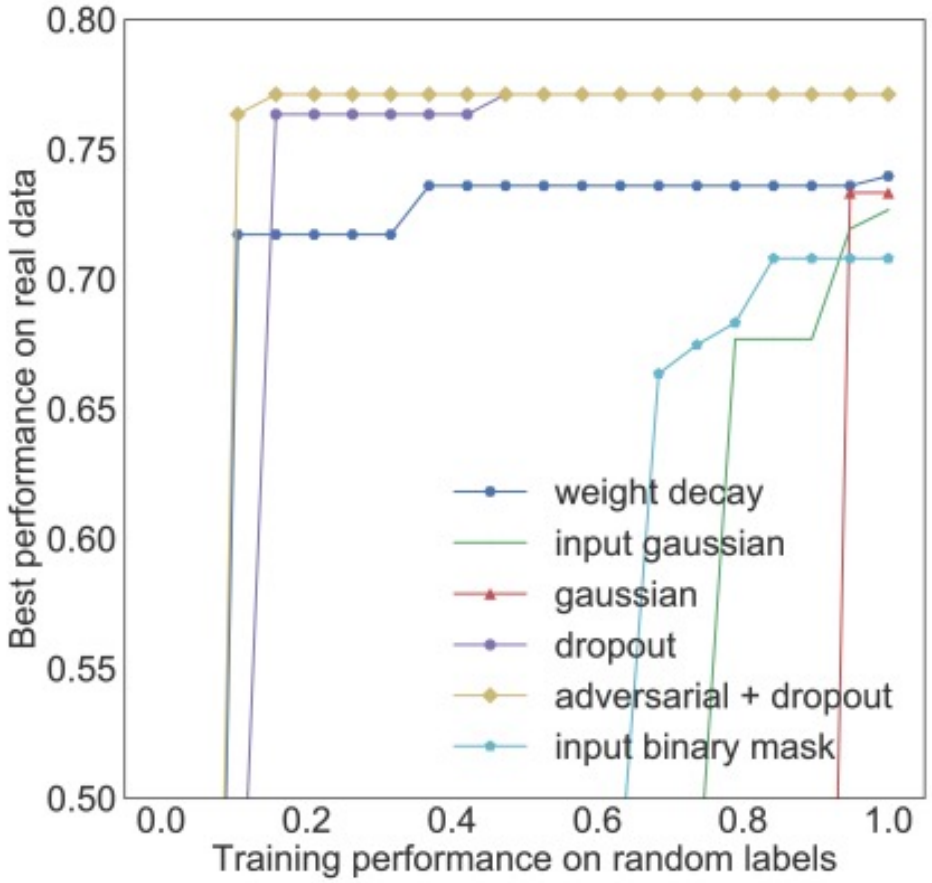
[Arpit et al 2017]

Overfitting does not generalize



(a) learning curves (b) convergence slowdown (c) generalization error growth

Regularizations do not have an effect on capacity !

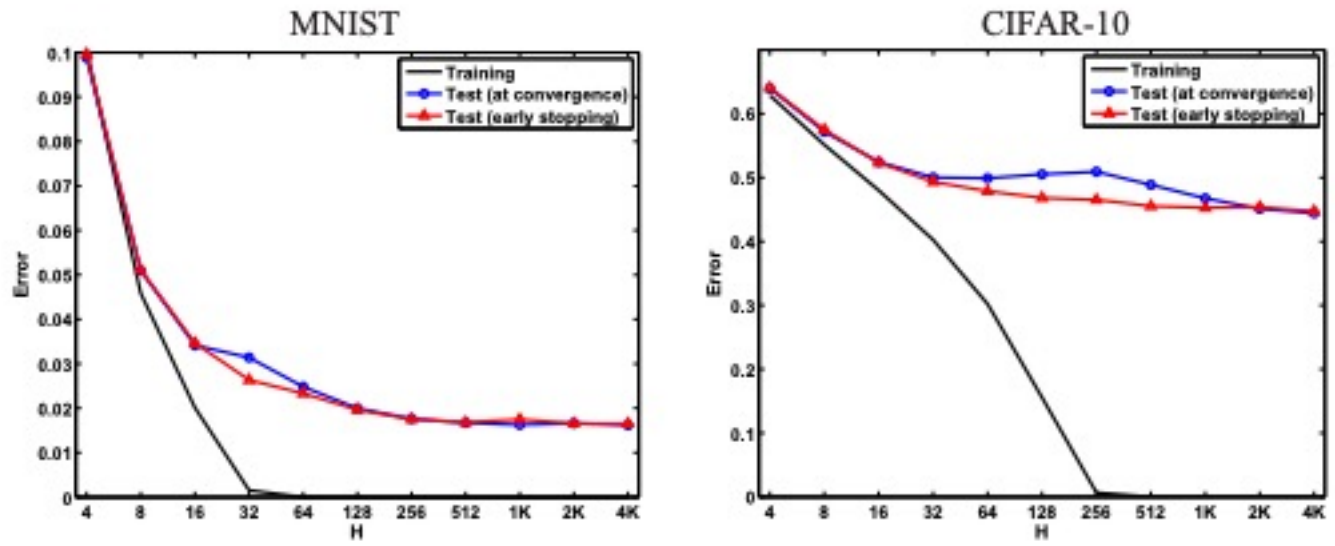


[Arpit et al. 2017[

They affect the ability to generalize but not on the ability model capacity

Inductive bias

regularization add inductive bias



[Neyshabur et al 2015]

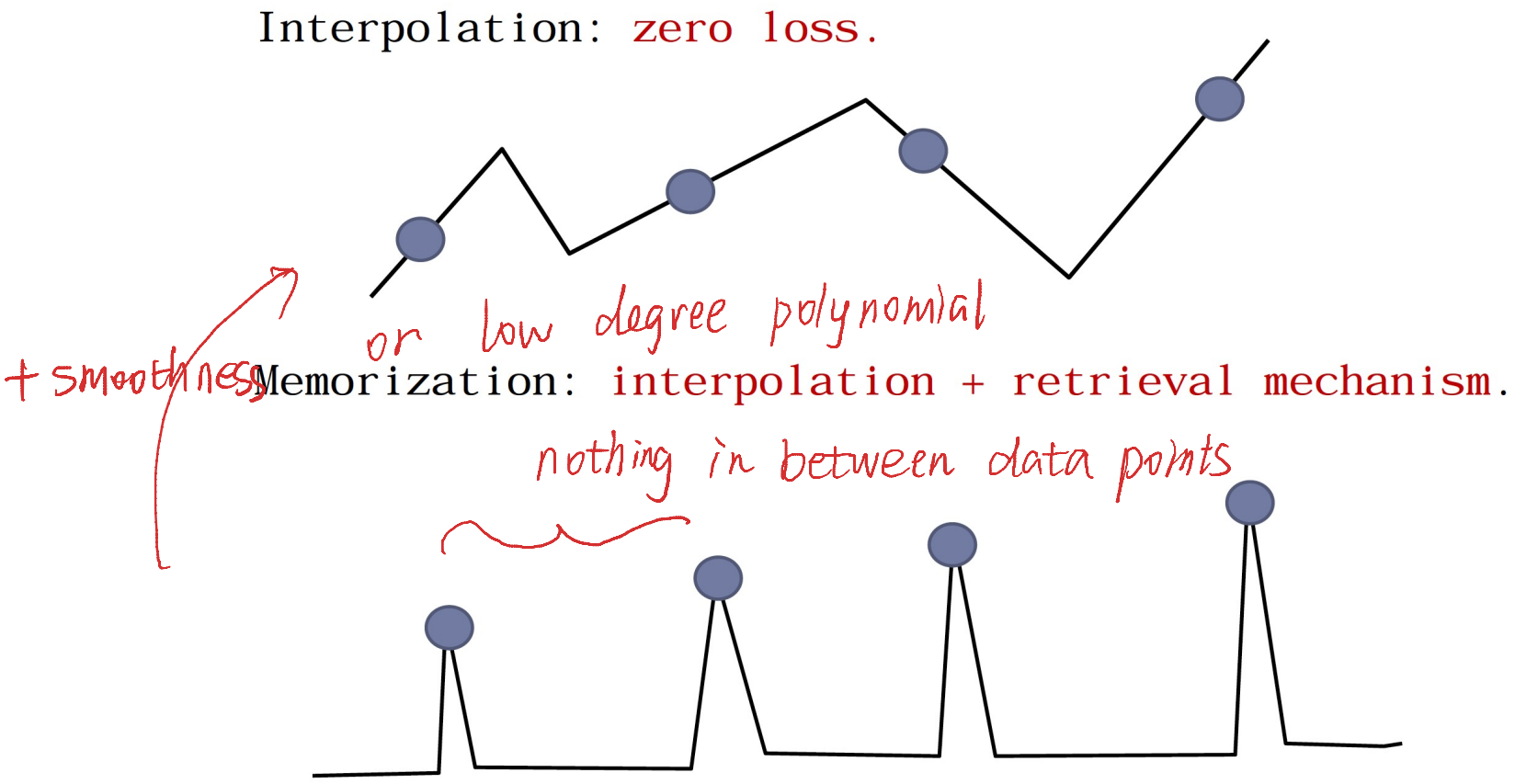
An inductive bias restricts or encourages predictors to be "simple" in some way, which in turn allows for generalization

The success of learning then depends on how well the inductive bias captures reality

The inductive bias is not the size of the neural network, what could it be?

model is still richness of params

Smoothness



- Difference between memorization and interpolation has to do with smoothness !

Ways to Increase Smoothness

minimize norm of weight

- Minimal norm solutions (L1, L2, frobenius norm)
- Averaging: (dropout)
- Other methods are boosting (Residual networks), bagging

Weight decay/L2 regularization

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

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

- 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

small cost

small weight

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

none of neurons carry too much info

- 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

regularization term must be differentiable as cost function

Weight decay/L2 regularization

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

- Weight update rule:

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

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

$\lambda \uparrow \quad w \downarrow$

- 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

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

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

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

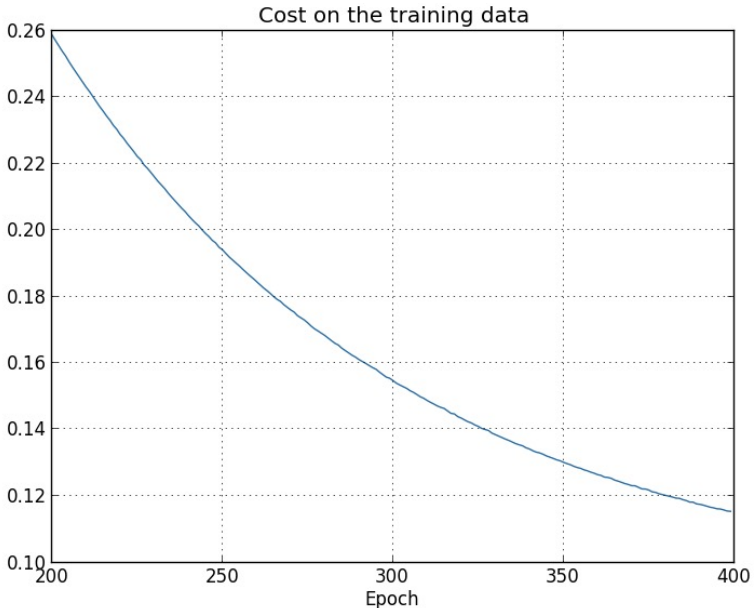
Weight decay/L2 regularization

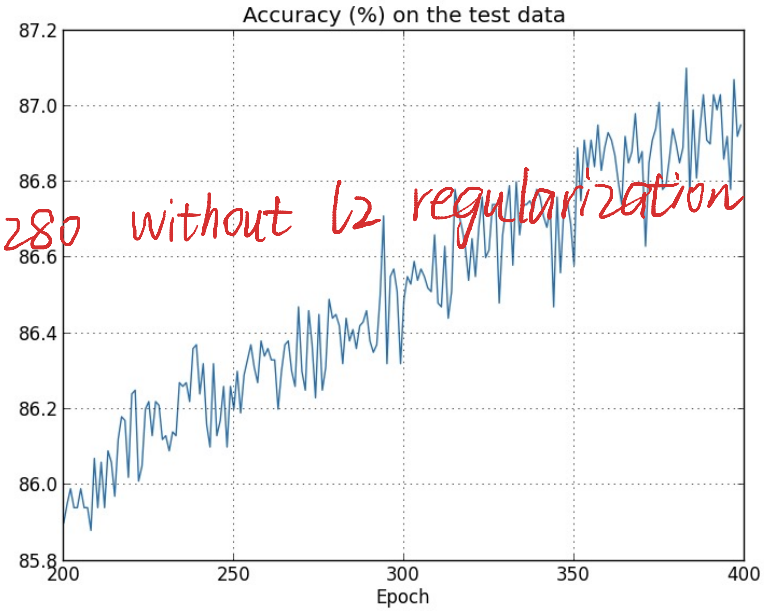
Why not regularize the biases?

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



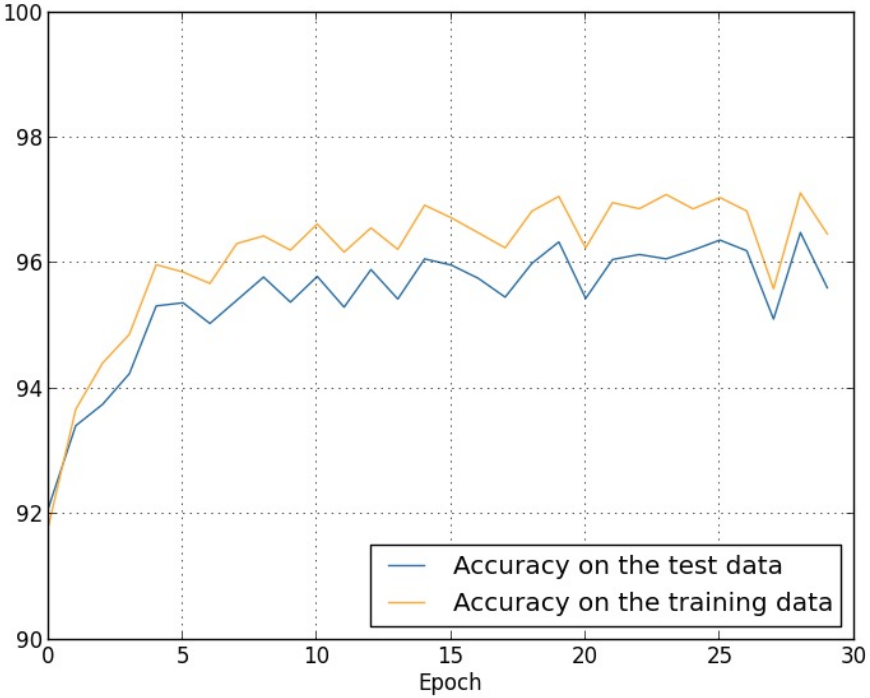


vs early stop at epoch 280 without L2 regularization

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 $\lambda = 5.0$

$n\uparrow$ $\lambda\uparrow$



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%

Space exploration

loss landscape

- On this data, regularized runs are much more easily replicated

  loss landscape is smoother with regularization

  - 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

Stability of regularization

prefer small weights

- 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

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:

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

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

$$\|w\|_p = \left(\sum_w w^p\right)^{\frac{1}{p}}$$

- 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:

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

- Update rule for L1 regularization

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

- Compare to update rule for L2 regularization

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

L1 regularization

- L1 regularization

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

- L2 regularization

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

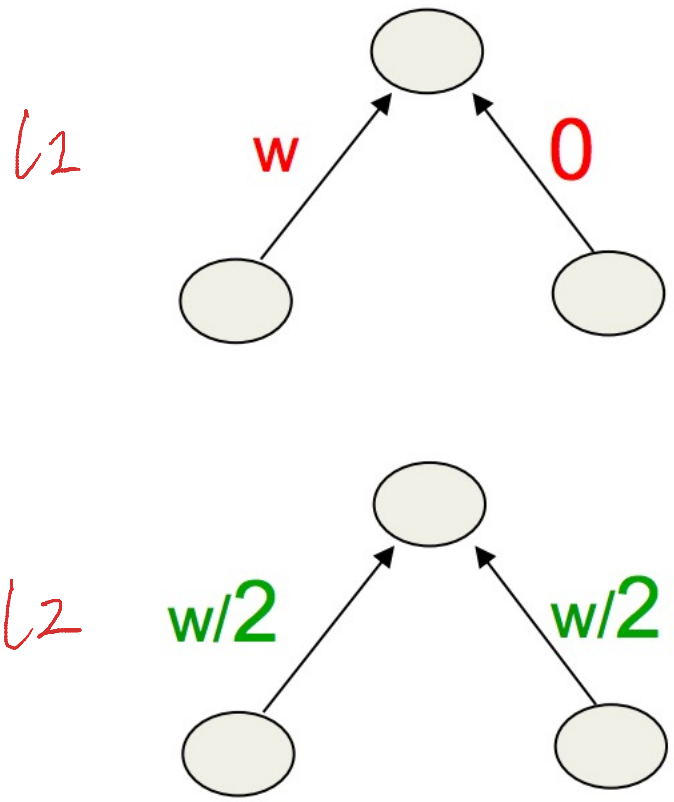
- Both shrink the weights
  - L1 shrinks the weights by a constant amount $\pm\frac{\lambda\eta}{n}$
  - L2 shrinks the weights by amount proportional to w $\frac{\eta\lambda}{n}w$
- If $|w|$ is large, L1 shrinks the weight much less than L2
- If $|w|$ is small, L2 shrinks the weight much less than L1

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

form of simplicity { sparsity
                   { loner form of solution

L2 Smoother Model



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

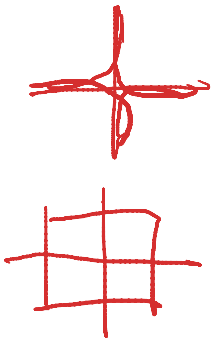
L1 vs L2

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

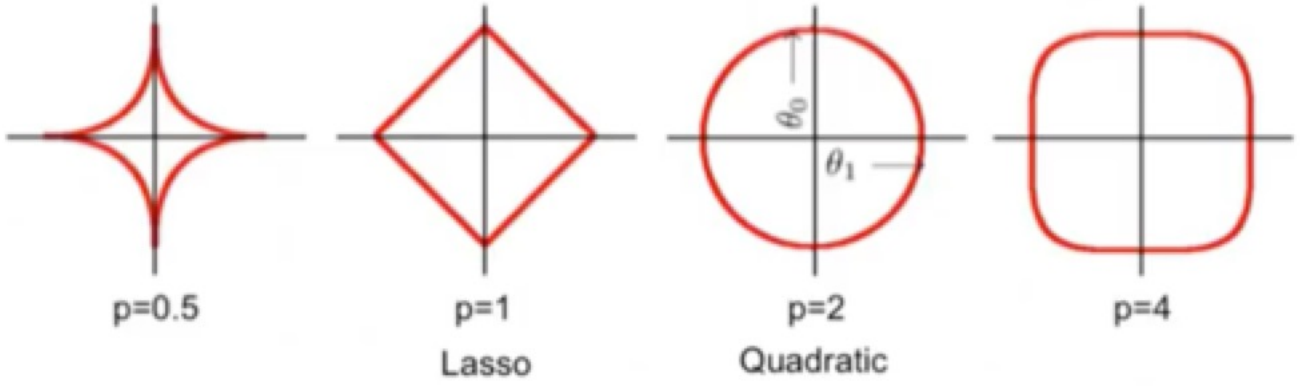
Lp Regularization

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

$\begin{cases} p \to 0 \\ p \to \infty \end{cases}$



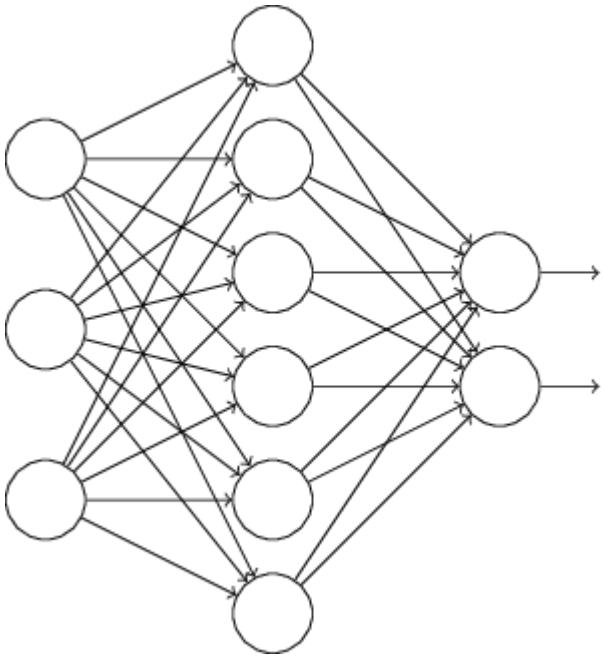
Isosurfaces



As p→0 you get a counting penalty

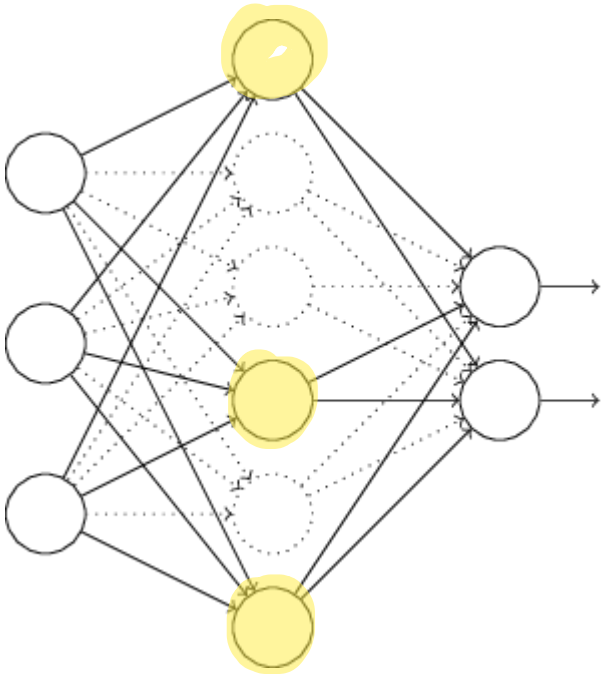
Dropout

- 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 every mini-batch
  - 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



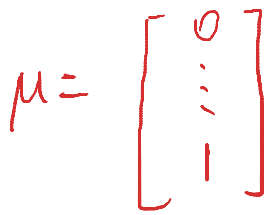
Dropout

- 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 → ensemble learning
- Dropout is a lot like training different neural networks
  - Net effect of dropout generally reduces overfitting

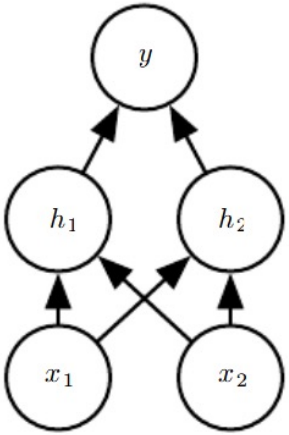
Dropout

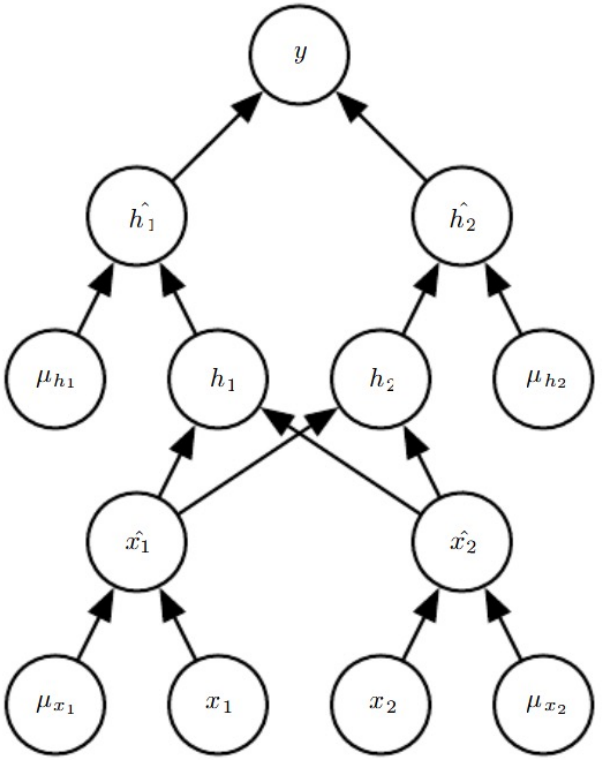
- Another explanation: “This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.” (Krizhevsky 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



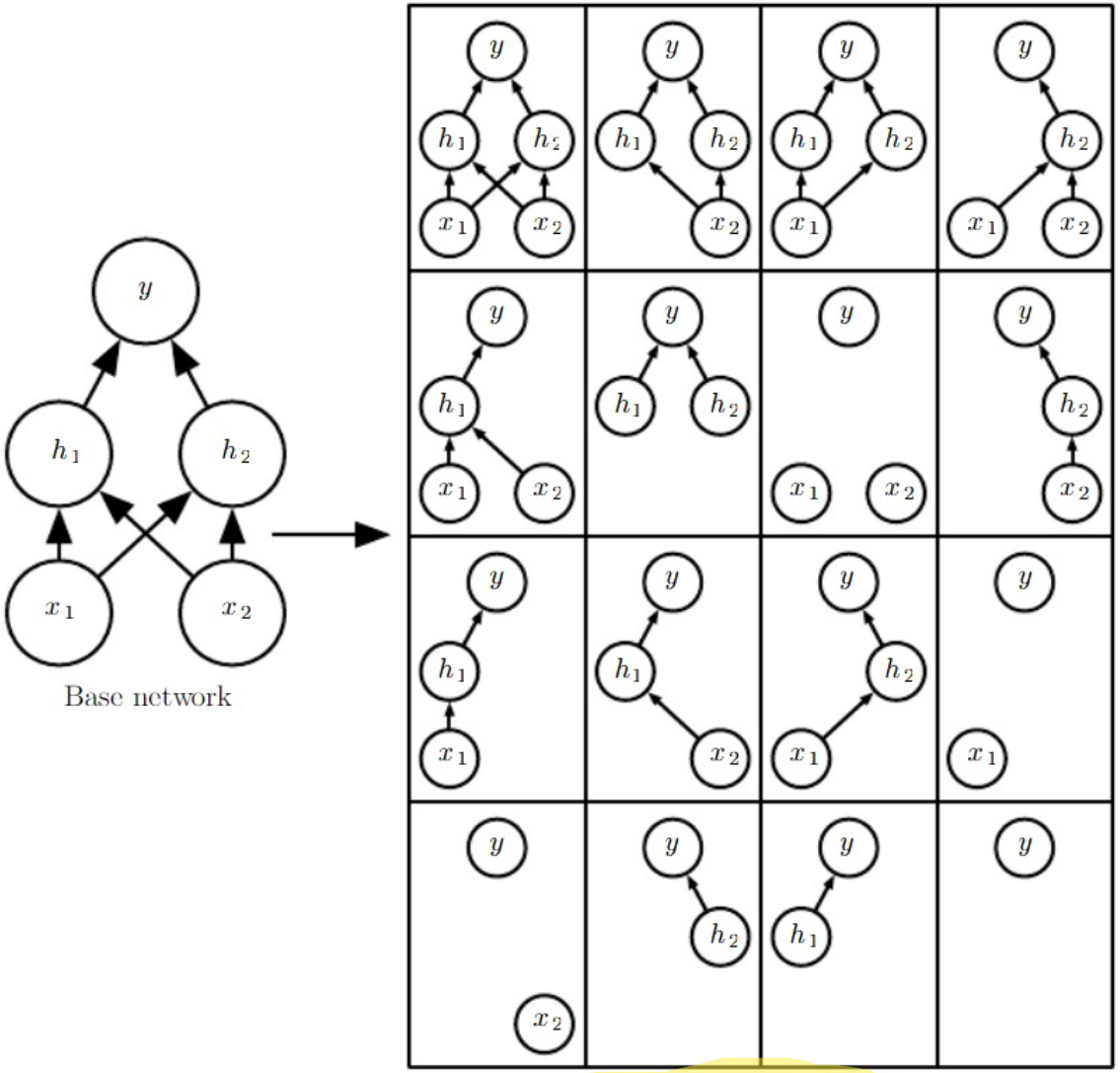
- Generate a binary random vector μ
  - Probability of each entry being 1 is a hyperparameter
- Multiply the output of each node with the corresponding entry in μ
- Multiply final weights by 1/2





Implementing dropout

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



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

Dropout is useful for dataset with medium size

Multitask training

- The same way that many datapoints can move parameters towards more generally good settings, many tasks can allow for generally good representations

- Earlier works used unsupervised pretraining before supervised training

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
- A closer look at memorization in deep networks [Arpit et al 2017] https://arxiv.org/pdf/1706.05394.pdf
- In search of the real inductive bias: on the role of implicit regularization in deep learning [Neyshabur et al, 2015] https://arxiv.org/pdf/1412.6614.pdf

- On the implicit Bias of Dropout https://arxiv.org/abs/1806.09777