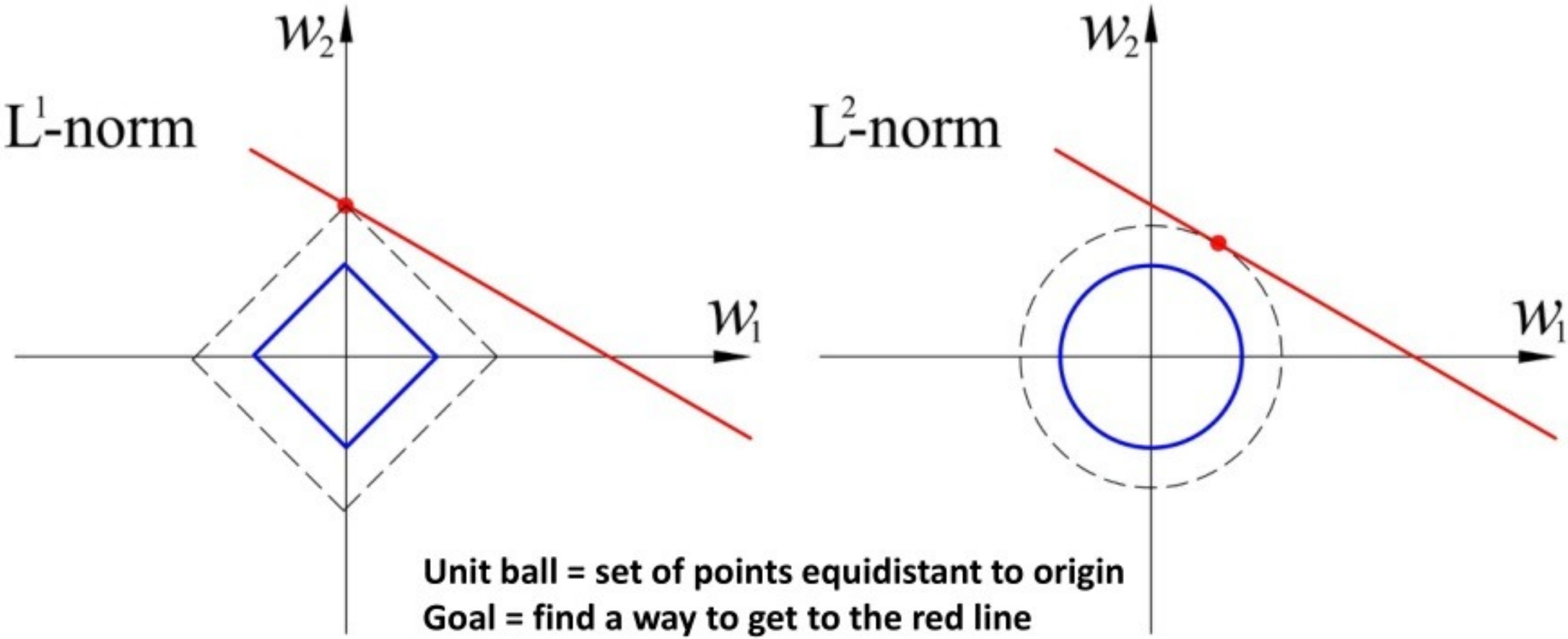
L1 and L2 Unit Balls



Unit ball = set of points equidistant to origin
Goal = find a way to get to the red line

Orthogonal matching pursuit (OMP)

- Another approach for representational sparsity
  - with a hard constraint on activation values
- Orthogonal matching pursuit (OMP) encodes $\boldsymbol{x}$ with h that satisfies

$$\arg\min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

  - where $||\boldsymbol{h}||_0$ is the number of zero entries of $\boldsymbol{h}$
- OMP-k, where k is the no. of zero entries
  - OMP-1 is very effective for deep architectures

1. Eliminating dead weights

- A reason to use explicit constraints and reprojection rather than enforcing constraints with penalties:
  - Penalties can cause nonconvex optimization procedures to get stuck in local minima corresponding to small θ
    - This manifests as training with *dead units*
  - Explicit constraints implemented by reprojection can work much better because they do not encourage weights to approach the origin

Dead unit = neuron will never activate on any datapoint again

Limiting Model Capacity

- Regularization has been used for decades prior to advent of deep learning
- Linear- and logistic-regression allow simple, straightforward and effective regularization strategies
  - Adding a parameter norm penalty $\Omega(\theta)$ to the objective function J :

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

    - where $\alpha\varepsilon[0,\theta)$ is a hyperparameter that weight the relative contribution of the norm penalty term Ω
      - Setting α to 0 results in no regularization. Larger values correspond to more regularization

Why does bagging work?

- Different models will not make the same error on the training set
- We can train k regression models separately, each with squared error ε_i
  - with variances $E[\varepsilon_i^2] = v$, covariances $E[\varepsilon_i \varepsilon_j] = c$,
  - and show that the average error, assuming independence, decreases linearly with ensemble size

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{j\neq i} \epsilon_i \epsilon_j\right)\right]$$
$$= \frac{1}{k}v + \frac{k-1}{k}c.$$

Prediction: Bagging vs. Dropout

- Bagging:
  - Ensemble accumulates votes of members
  - Process is referred to as inference
    - Assume model needs to output a probability distribution
    - In bagging, model i produces $p^{(i)}(y|\boldsymbol{x})$
    - Prediction of ensemble is the mean $\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y \mid \boldsymbol{x})$
- Dropout:
  - Submodel defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p(y|\boldsymbol{x},\boldsymbol{\mu})$
  - Arithmetic mean over all masks is $\sum_{\mu} p(y \mid \boldsymbol{x},\mu)$
    - Where $p(\boldsymbol{\mu})$ is the distribution used to sample $\boldsymbol{\mu}$ at training time

How α influences weights

- If Ω is L^2 norm
  - weights are then constrained to lie in an L^2 ball
- If Ω is the L^1 norm
  - Weights are constrained to lie in a region of limited L^1 norm
- Usually we do not know size of constraint region that we impose by using weight decay with coefficient α^* because the value of α^* does not directly tell us the value of k
  - Larger α will result in smaller constraint region
  - Smaller α will result in larger constraint region

Regularization for Deep Learning

C. Lee Giles

Thanks to Alexander Ororbia and Sargur Srihari

Topics in Early Stopping

1. Learning Curves
2. Early Stopping vs L^2 Regularization

Parameter Tying

- We want to express that certain parameters should be close to one another

Topics in Parameter Norm Penalties

1. Overview (limiting model capacity)
2. L^2 parameter regularization
3. L^1 regularization

What is Regularization?

- Central problem of ML is to design algorithms that will perform well not just on training data but on new inputs as well
- Regularization is:
  - "any modification we make to a learning algorithm to reduce its generalization error but not its training error"
  - Reduce test error even at the expense of increasing training error

Topics in Parameter Tying/Sharing

1. Other methods for prior knowledge of parameters
2. Parameter Tying
3. Parameter Sharing
4. Parameter sharing in CNNs

Topics in Dropout

- What is dropout?
- Dropout as an ensemble method
- Mask for dropout training
- Bagging vs Dropout
- Prediction intractability

Regularize Your Estimator!

- In Deep Learning, regularization means regularizing estimators
- Involves increased bias for reduced variance
  - Good **regularizer** reduces variance significantly while not overly increasing bias

A scenario of parameter tying

- Two models performing the same classification task (with same set of classes) but with somewhat different input distributions
- Model A with parameters $\boldsymbol{w}^{(A)}$
- Model B with parameters $\boldsymbol{w}^{(B)}$
- The two models map the input to two different but related outputs

$$\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$$
$$\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$$

L^1 Regularization

- While L^2 weight decay is the most common form of weight decay there are other ways to penalize the size of model parameters
- L^1 regularization is defined as

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

  - which is the sum of the absolute values of the individual parameters

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Direct versus Representational Sparsity

– A sparse linear regression model

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{A} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{x} \in \mathbb{R}^n$

– Linear regression with a sparse representation $\boldsymbol{h}$ of data $\boldsymbol{x}$

$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{B} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{h} \in \mathbb{R}^n$

Another expression for parameter prior

- L^2 regularization (or weight decay) penalizes model parameters for deviating from fixed value of zero
- Sometimes we need other ways to express prior knowledge of parameters
- We may know from domain and model architecture that there should be some dependencies between model parameters

Lagrange Formulation

- If we wanted to constrain $\Omega(\theta)$ to be less than some constant k, we could construct a generalized Lagrange function

$$L(\theta, \alpha; X, y) = J(\theta; X, y) + \alpha\left(\Omega(\theta) - k\right)$$

- The solution to the constrained problem is given by

$$\theta^* = \arg\min_{\theta} \max_{\alpha, \alpha \geq 0} L\left(\theta, \alpha\right)$$

- Solving this problem requires modifying both θ and α
  - Many different procedures are possible

Regularization Strategies

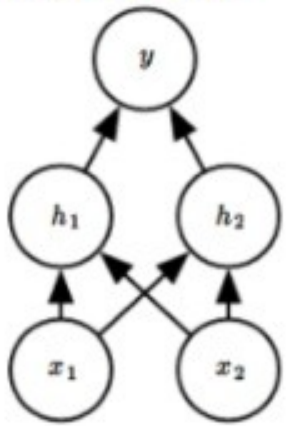
1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Finding the Best Model

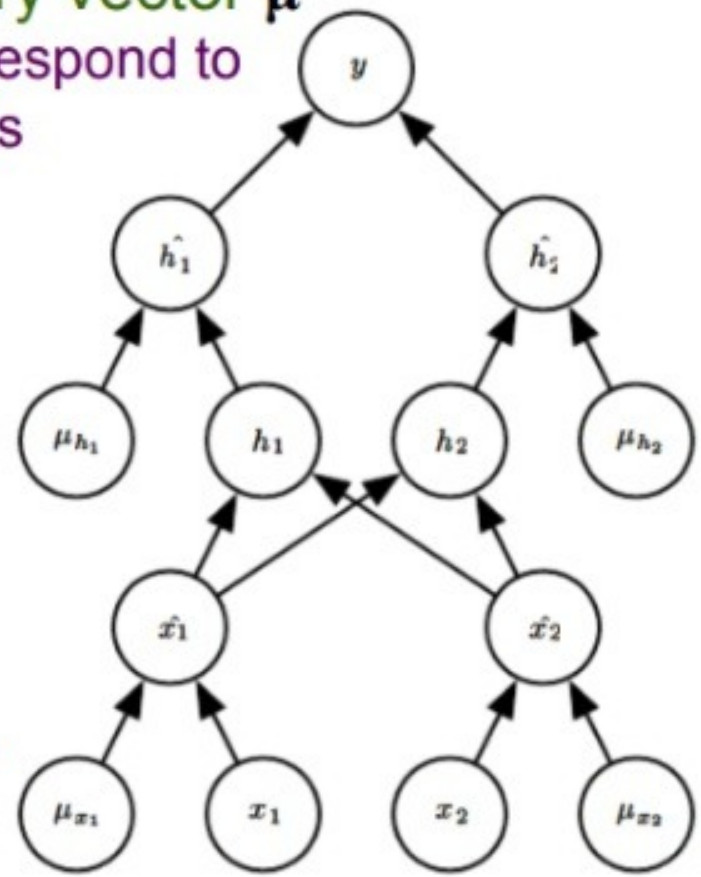
- Best fitting model obtained not by finding the right number of parameters
- Instead, best fitting model is a large model that has been regularized appropriately
- We review several strategies for how to create such a large, deep regularized model

Forward Propagation with dropout

Feed-forward network



- Network with binary vector $\boldsymbol{\mu}$ whose elements correspond to input and hidden units
- Elements of $\boldsymbol{\mu}$
- With probability of 1 being a hyperparameter
  - 0.5 for hidden
  - 0.8 for input
- Each unit is
  - Multiplied by corresponding mask



- Forward prop as usual
- Equivalent to randomly selecting one of the subnetworks of previous slide

Topics in Data Augmentation

1. More data is better
2. Augmentation for classification
3. Caution in data augmentation
4. Injecting noise
5. Benchmarking using augmentation
6. Ex: Heart disease diagnosis using deep learning

Regularization & Model Types

- Three types of model families
  1. Excludes the true data generating process
     - Implies underfitting and inducing high bias
  2. Matches the true data generating process
  3. Overfits
     - Includes true data generating process but also many other processes
- Goal of regularization is to take model from third regime to second

Adding Noise to Weights

- This technique primarily used with RNNs
- This can be interpreted as a stochastic implementation of Bayesian inference over the weights
  - Bayesian treatment of learning would consider the model weights to be uncertain and representable via a probability distribution that reflects that uncertainty
  - Adding noise to weights is a practical, stochastic way to reflect this uncertainty

$$g_t \leftarrow g_t + N(0, \sigma_t^2) \qquad \sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

More data is better

- Best way to make a ML model to generalize better is to train it on more data
- In practice amount of data is limited
- Get around the problem by creating synthesized data
- For some ML tasks it is straightforward to synthesize data

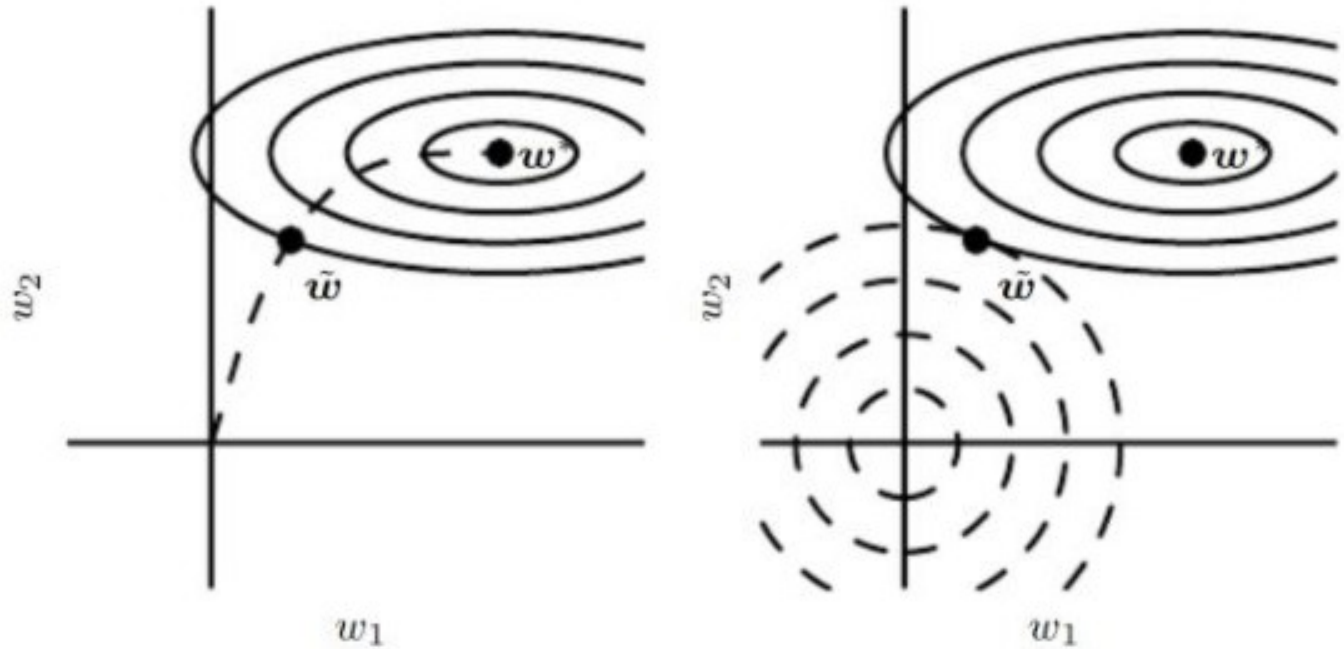
More robust to adversarial examples

Augment the data!

Bagging Method

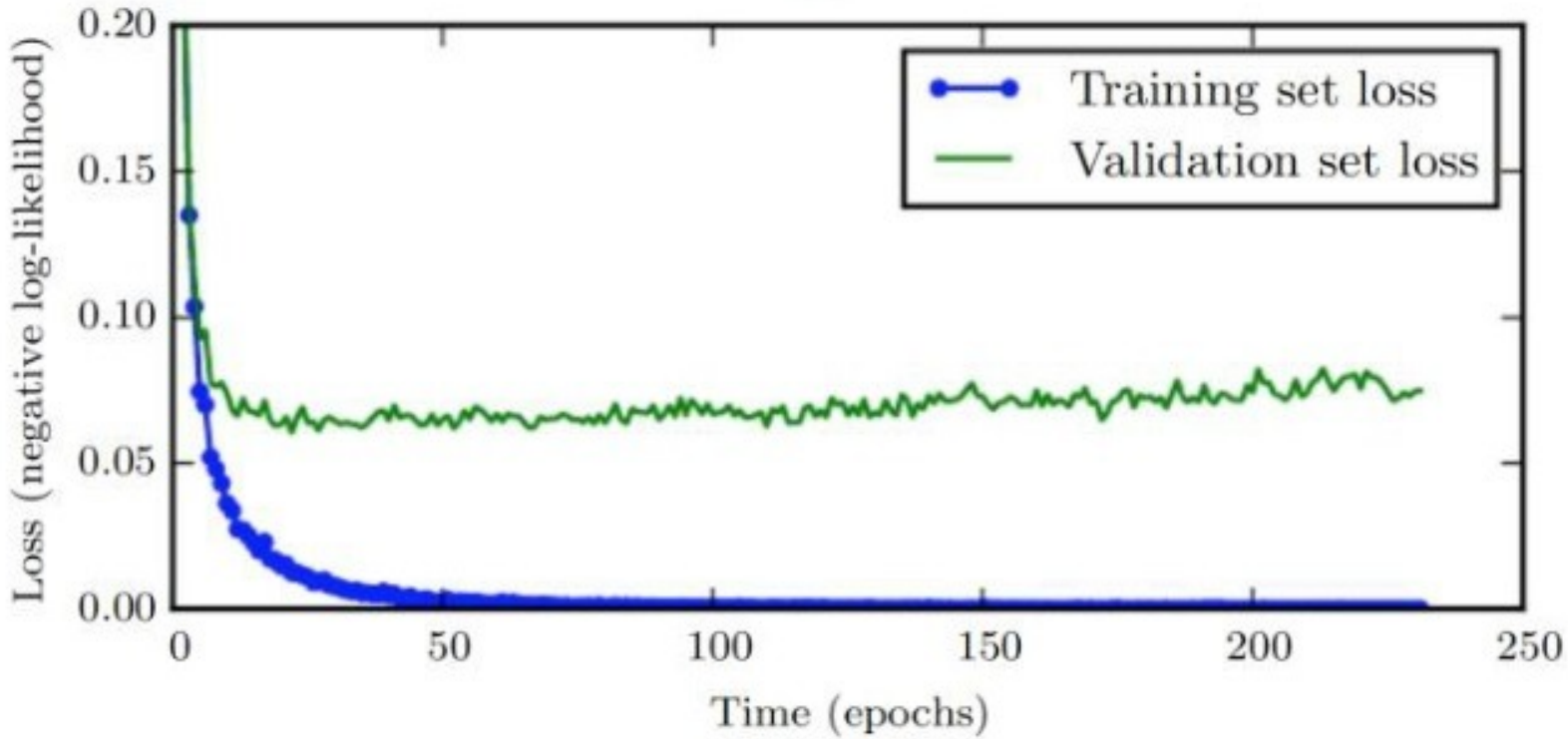
- Drawing n' < n samples from D with replacement
- Each bootstrap set is used to train a different component classifier
- Final classification decision based on vote of each component classifier
- Component classifiers are all of the same form
  - all HMMs, all neural networks, all decision trees
  - Different parameter values due to different training sets

Early Stopping vs L^2 regularization



- Two weights, Solid contour lines: contours of negative log-likelihood
- Left: dashed lines indicates trajectory of SGD. Rather than stopping at point w^* that minimizes cost, early stopping results in an earlier point in trajectory
- Right: dashed circles indicate contours of L^2 penalty which causes the minimum of the total cost to lie nearer the origin than the minimum of the the unregularized cost

Learning Curves



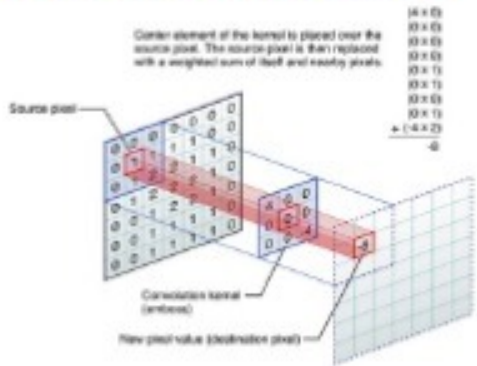
Shows how negative log-likelihood loss changes over time (indicated as no. of Training iterations over the data set, or epochs).

In this example, we train a maxout network on MNIST.

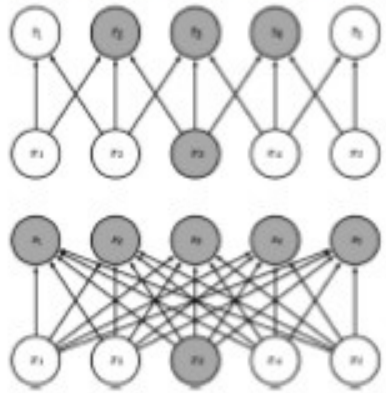
Training objectve decreases consistently over time, but validation set average Loss eventually begins to increase again for ming an asymmetric U shape

Simple description of CNN

Convolution operation

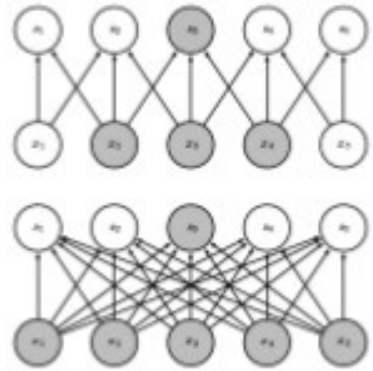


Sparsity viewed from below



- Highlight one input x_3 and output units s affected by it
- *Top*: when s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3
- *Bottom:* when s is formed by matrix multiplication connectivity is no longer sparse
  - So all outputs are affected by x_3

Sparsity viewed from above



- Highlight one output s_3 and
- inputs x that affect this unit
  - These units are known as the receptive field of s_3

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Formal description of dropout

- Suppose that mask vector $\boldsymbol{\mu}$ specifies which units to include
- Cost of the model is specified by $J(\boldsymbol{\theta},\boldsymbol{\mu})$
- Drop training consists of minimizing $E_{\boldsymbol{\mu}}(J(\boldsymbol{\theta},\boldsymbol{\mu}))$
- Expected value contains exponential no. of terms
- We can get an unbiased estimate of its gradient by sampling values of $\boldsymbol{\mu}$

Dropout as bagging

- In bagging we define k different models, construct k different data sets by sampling from the dataset with replacement, and train model i on dataset i
- Dropout aims to approximate this process, but with an exponentially large no. of neural networks

L^2 penalty for parameter tying

- If the tasks are similar enough (perhaps with similar input and output distributions) then we believe that the model parameters should be close to each other:

$$\forall i,\ w_i^{(A)} \approx w_i^{(B)}$$

- We can leverage this information via regularization
- Use a parameter norm penalty

$$\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = ||\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}||_2^2$$

Dropout method

- Dropout trains an ensemble of all subnetworks
  - Subnetworks formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero
  - For networks based on performing a series of affine transformations or on-linearities
  - Needs some modification for radial basis functions based on difference between unit state and a reference value

Parameter Sharing

- Parameter sharing forces sets of parameters to be equal
- Because we interpret various models or model components as sharing a unique set of parameters
- Only a subset of the parameters needs to be stored in memory
  - In a CNN significant reduction in the memory footprint of the model

What is dropout?

- An inexpensive but powerful method of regularizing a broad family of models
  - A method of bagging applied to neural networks
- Impractical to train many neural networks since it is expensive in time and memory
  - Dropout makes it practical to apply bagging to very many large neural networks

Implementation

- Implemented in Keras
  - Code available in Github
    - https://github.com/chuckyee/cardiac-segmentation
- Baseline is fully convolutional network (FCN)
- Endocardium and epicardium performance

Method	Train	Val	Test	Params
Human	–	–	0.90 (0.10)	–
FCN (Tran 2017)	–	–	0.86 (0.20)	~11M
U-net	0.93 (0.07)	0.86 (0.17)	0.77 (0.30)	1.9M
Dilated u-net	0.94 (0.05)	0.90 (0.14)	**0.88 (0.18)**	3.7M
Dilated densenet	0.94 (0.04)	0.89 (0.15)	0.85 (0.20)	**0.19M**

Method	Train	Val	Test	Params
Human	–	–	0.90 (0.10)	–
FCN (Tran 2017)	–	–	**0.84 (0.21)**	~11M
U-net	0.91 (0.06)	0.82 (0.23)	0.79 (0.28)	1.9M
Dilated u-net	0.92 (0.08)	0.85 (0.19)	**0.84 (0.21)**	3.7M
Dilated densenet	0.91 (0.10)	0.87 (0.15)	0.83 (0.22)	**0.19M**

L^2 parameter Regularization

- Simplest and most common kind
- Called *Weight decay*
- Drives weights closer to the origin
  - by adding a regularization term to the objectve function

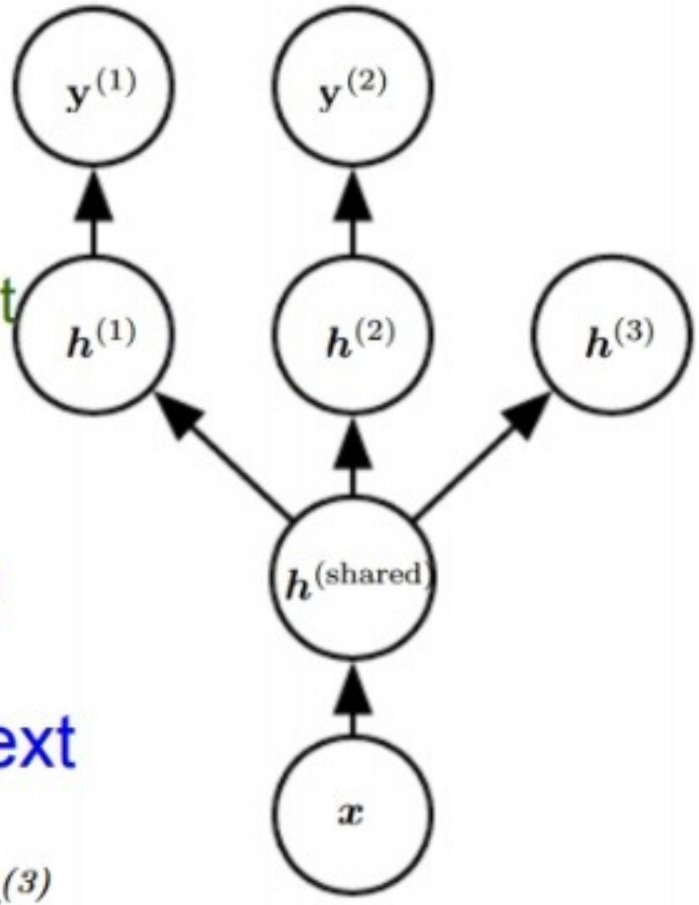
$$\Omega(\theta) = \frac{1}{2} || w ||_2^2$$

- In other communities also known as *ridge regression* or *Tikhonov regularization*

Common multi-task situation

- Common input but different target random variables.
  - Lower layers (whether feedforward or includes a generative component with downward arrows) can be shared across such tasks.
    - Task-specific parameters $h^{(1)}$, $h^{(2)}$ can be learned on top of those yielding a shared representation $h^{(shared)}$
- In the unsupervised learning context
  - some of the top level factors are associated with none of the output tasks $h^{(3)}$

  These are factors that explain some of the input variations but not relevant for predicting $h^{(1)}$, $h^{(2)}$



Increase in validation set error

- When training large models with sufficient representational capacity to overfit the task, training error decreases steadily over time, but validation set error begins to rise again
- An example of this behavior is shown next

The Value of Regularization

Model family

- Overly complex family does not necessarily include target function, true data generating process, or even an approximation
- Most deep learning applications are where true data generating process is outside family
  - Complex domains of images, audio sequences and text true generation process may involve entire universe
    - Fitting square hole (data generating process) to round hole (model family)

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Need for Data augmentation

- Dataset: 243 physician-segmented images of 16 patients.
  - 3697 additional unlabeled images, useful for unsupervised or semi-supervised techniques
    - Generalization to unseen images would be hopeless!
  - Typical situation in medical settings where labeled data is expensive.

Mask for dropout training

- To train with dropout we use minibatch based learning algorithm that takes small steps such as SGD
- At each step randomly sample a binary mask
  - Probability of including a unit is a hyperparameter
    - 0.5 for hidden units and 0.8 for input units
- We run forward & backward propagation as usual

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
6. Parameter tying and parameter sharing
7. Sparse representations
8. Bagging and other ensemble methods
9. Dropout
10. Adversarial training
11. Tangent methods

Different or Same αs for layers?

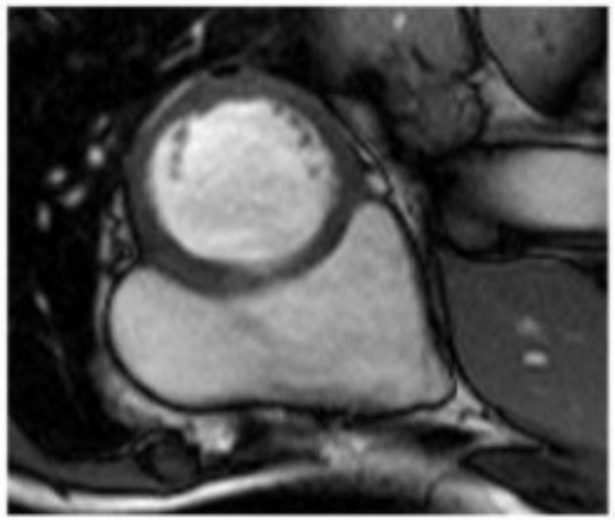
- Sometimes it is desirable to use a separate penalty with a different α for each layer

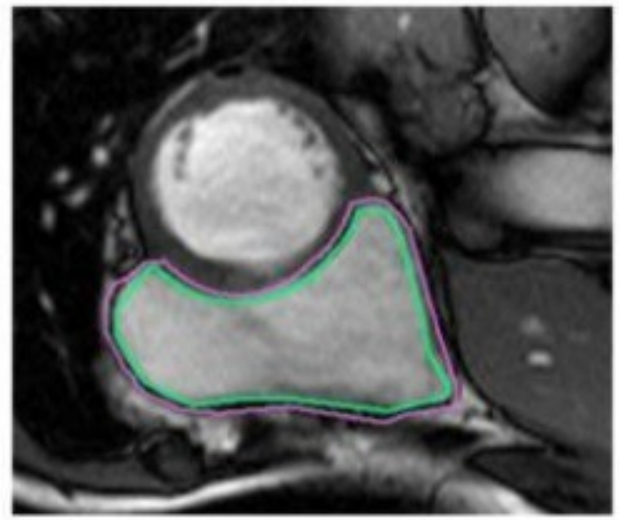
$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

- Because it can be expensive to search for the correct value of multiple hyperparameters, it is still reasonable to use same weight decay at all layers to reduce search space

Problem Description

- Develop system to segment RV in cardiac MRI
  - Currently handled by classical image processing
- RV has irregularly shaped thin walls: inner and outer walls (endocardium and epicardium)
  - Manually drawn contours shown:





Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
6. Parameter tying and parameter sharing
7. Sparse representations
8. Bagging and other ensemble methods
9. Dropout
10. Adversarial training
11. Tangent methods

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Insight into effect of constraint

- We can fix α^* and view the problem as just a function of θ:

$$\theta^* = \arg\min_{\theta} L\left(\theta, \alpha^*\right) = \arg\min_{\theta} J(\theta; X, y) + \alpha^* \Omega(\theta)$$

- This is exactly the same as the regularized training problem of minimizing $\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$
- We can thus think of the parameter norm penalty as imposing a constraint on the weights

Regularization Strategies

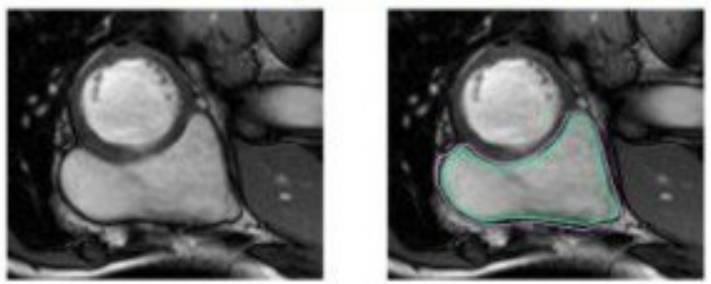
1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
6. Parameter tying and parameter sharing
7. Sparse representations
8. Bagging and other ensemble methods
9. Dropout
10. Adversarial training
11. Tangent methods

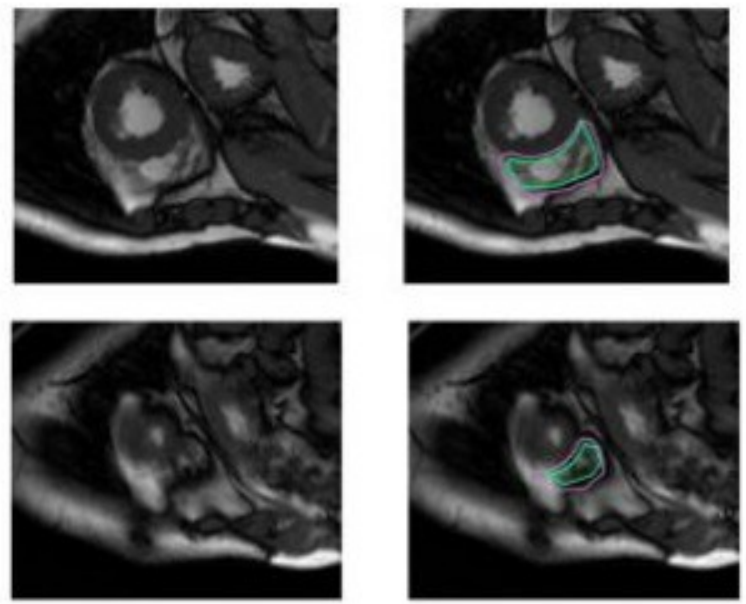
What is bagging?

- Bagging is short form of “bootstrap aggregrating”
- It is a technique for reducing generalization error by combining several models
  - Different models are trained separately
  - All the models vote on the output for test examples
- This strategy for machine learning is referred to as model averaging
  - Techniques employing this strategy are known as ensemble methods

RV segmentation is difficult

- Left ventricle segmentation is easier
  - LV is a thick-walled circle
    - Kaggle 2016 competition
- Right ventricle segmentation is harder
  - Complex crescent shape
  - Easy and hard cases





Task: determine whether each pixel is part of RV or not

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Goals of Regularization

- Many forms of regularization available
  - Major efforts are to develop better regularization
- Put extra constraints on objective function
  - They are equivalent to a soft constraint on parameter values
    - Result in improved performance on test set
- Some goals of regularization
  1. Encode prior knowledge
  2. Express preference for simpler model
  3. Needed to make underdetermined problem determined

Norm Penalty

- When our training algorithm minimizes the regularized objective function

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

  - it will decrease both the the original objective J on the training data and some measure of the size of the parameters θ
- Different choices of the parameter norm Ω can result in different solutions preferred
  - We discuss effects of various norms

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
6. Parameter tying and parameter sharing
7. Sparse representations
8. Bagging and other ensemble methods
9. Dropout
10. Adversarial training
11. Tangent methods

Caution in Data Augmentation

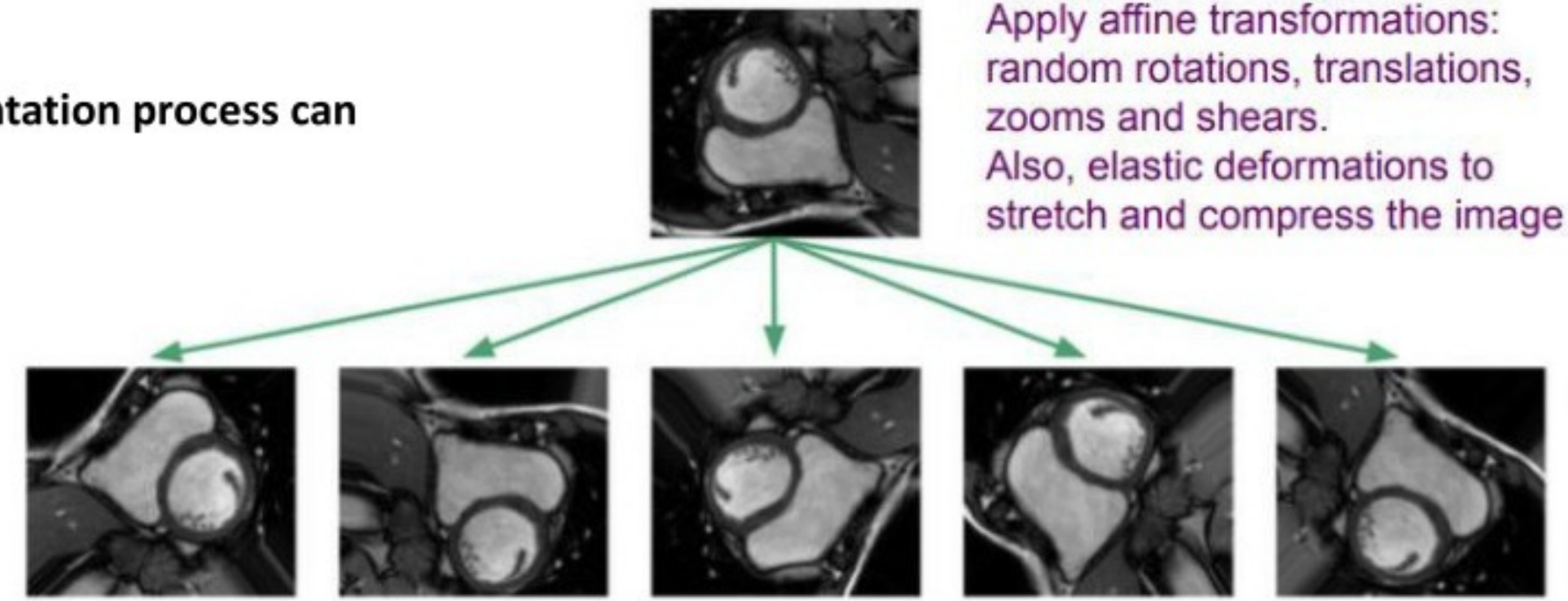
- Not apply transformation that would change the class
- OCR example: ‘b’ vs ‘d’ and ‘6’ vs ‘9’
  - Horizontal flips and 180 degree rotations are not appropriate ways
- Some transformations are not easy to perform
  - Out of plane rotation cannot be implemented as a simple geometric operation on pixels

Ensemble Methods and Bagging

- Each methods could be constructed by a completely different model, using different algorithm or objective function
- Bagging allows same model, training algorithm, objective function to be reused several times

Transformed data

Data augmentation process can be dynamic!



Apply affine transformations:
random rotations, translations, zooms and shears.
Also, elastic deformations to stretch and compress the image

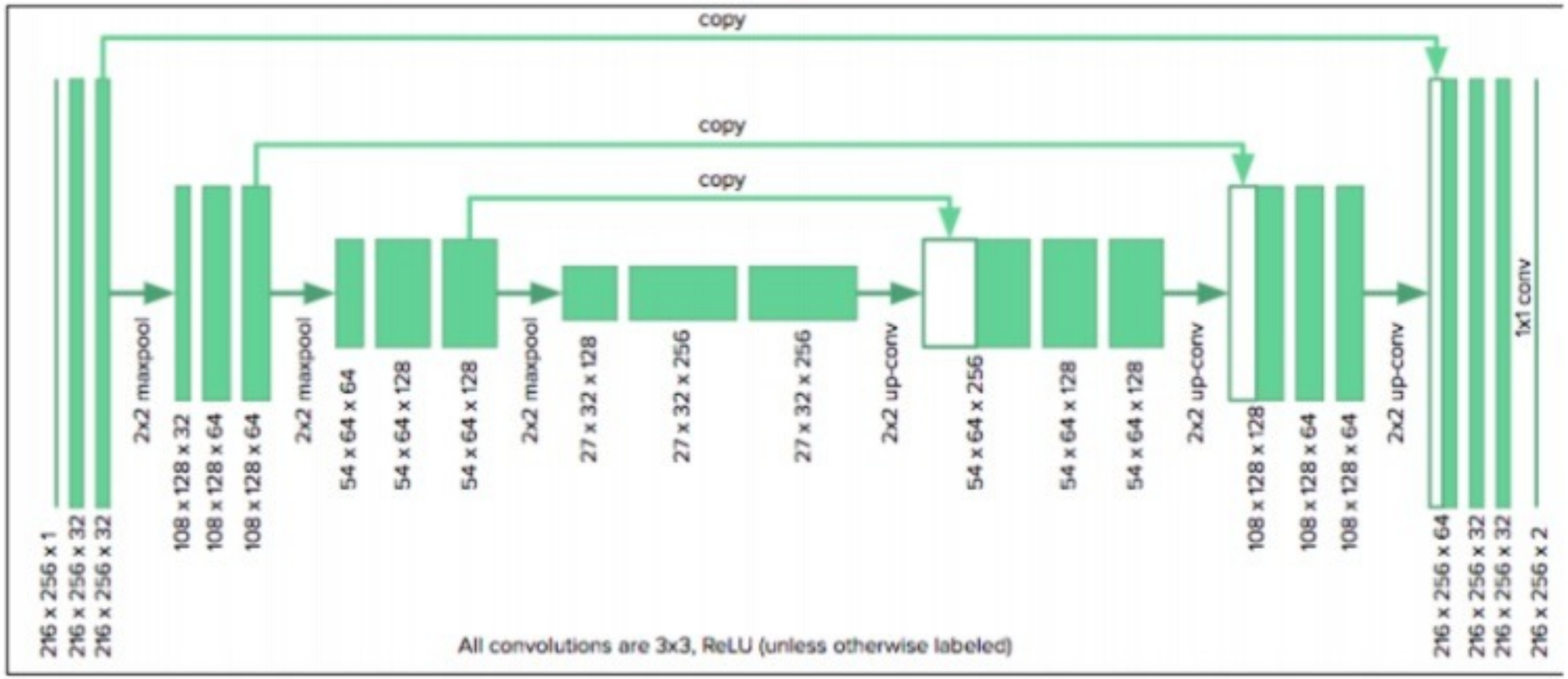
Goal: prevent network from memorizing just the training examples, and force it to learn that the RV is a solid, crescent-shaped object in a variety of orientations.

Apply transformations on the fly so the network sees new random transformations during each epoch.

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Deep Learning Architecture



U-net architecture

- Train network with only 30 images using augmentation and pixel-wise reweighting

- It consists of a contracting path, which collapse image into high level features,
- Uses the feature information to construct a pixel-wise segmentation mask.
- Copy and concatenate connections pass information from early feature maps to later portions of the network tasked with constructing the segmentation mask.

Adversarial training and Capacity

- It illustrates the power of using a large function family in combination with aggressive regularization
- Purely linear models are unable to resist adversarial examples
- Neural networks can capture linear trends as well as learning to resist local perturbation

Task of Semi-supervised Learning

- Both unlabeled examples from $P(\boldsymbol{x})$ and labeled examples from $P(\boldsymbol{x}, y)$ are used to estimate $P(y|\boldsymbol{x})$ or predict y from $\boldsymbol{x}$
- In the context of deep learning it refers to learning a representation $\boldsymbol{h} = f(\boldsymbol{x})$
- The goal is to learn a representation so that examples from the same class have similar representations

Representational Regularization

- Accomplished using same sort of mechanisms used in parameter regularization
- Norm penalty regularization of representation
  - Performed by adding to the loss function J, a norm penalty on the representation.
    - The regularized loss function is

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

      where $\alpha \in [0,\infty)$

    - An L^1 penalty term induces sparsity: $\Omega(\boldsymbol{h}) = ||\boldsymbol{h}||_1 = \sum_i |h_i|$

Reprojection

- Sometimes we may wish to use explicit constraints rather than penalties
  - We can modify SGD to take a step downhill on $J(\theta)$ and then project θ back to the nearest point that satisfies $\Omega(\theta) < k$
  - This useful when we have an idea of what values of k is appropriate and we do not want to spend time searching for the value of α that corresponds to this k
- Rationale for explicit constraints/Reprojection
  1. Dead weights
  2. Stability

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Gradient of Regularized Objective

- Objective function (with no bias parameter)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

- Corresponding parameter gradient

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- To perform single gradient step, perform update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \varepsilon\left(\alpha \boldsymbol{w} + \nabla_w J\left(\boldsymbol{w}; X, \boldsymbol{y}\right)\right)$$

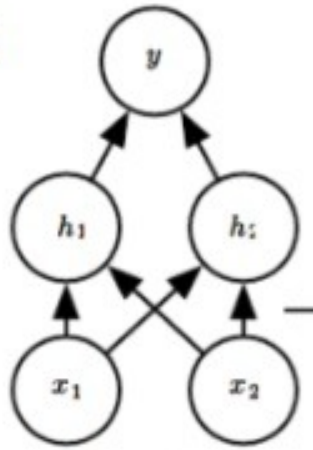
- Written another way, the update is

$$\boldsymbol{w} \leftarrow (1 - \varepsilon\alpha)\boldsymbol{w} - \varepsilon \nabla_w J\left(\boldsymbol{w}; X, \boldsymbol{y}\right)$$

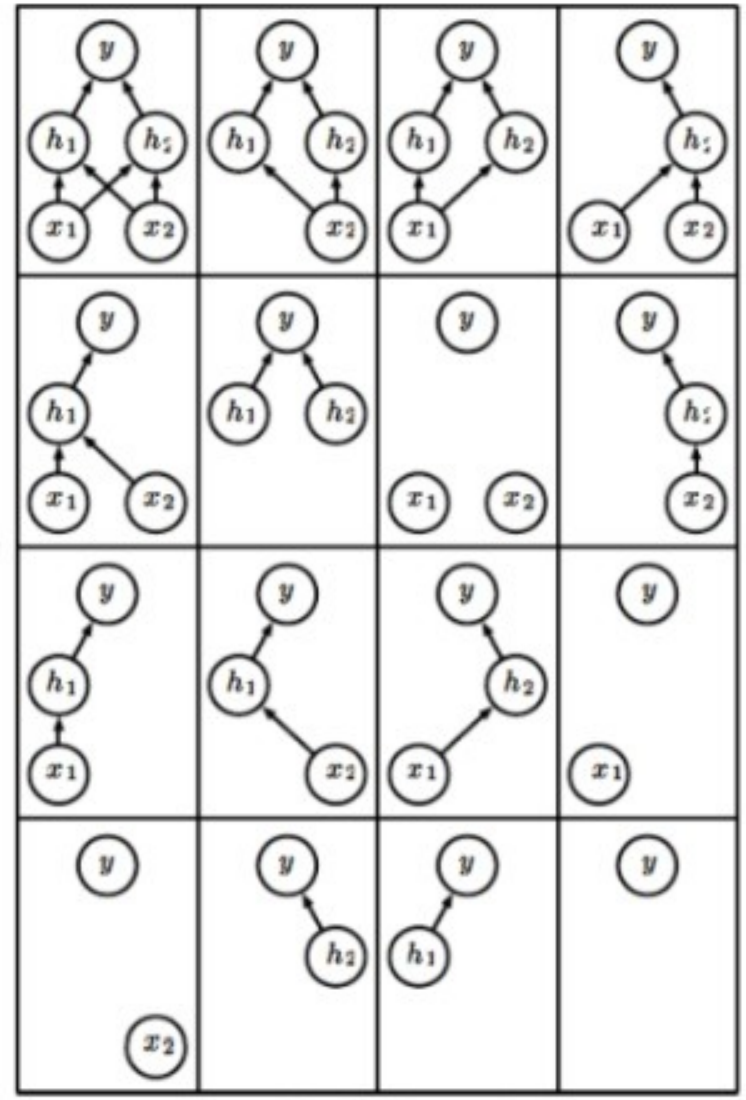
  - We have modified learning rule to shrink $\boldsymbol{w}$ by constant factor $1\text{-}\varepsilon\alpha$ at each step

Dropout: An ensemble method

- Remove non-output units from base network.
- Remaining 4 units yield 16 networks



Base network



Ensemble of subnetworks

- Here many networks have no path from input to output
- Problem insignificant with large networks

Intractability of prediction

- Dropout prediction is $\sum_{\mu} p(y \mid \boldsymbol{x}, \mu)$
- It is intractable to evaluate due to an exponential no. of terms
- We can approximate inference using sampling
  - By averaging together the output from many masks
    - 10-20 masks are sufficient for good performance
- Even better approach, at the cost of a single forward propagation:
  - use geometric mean rather than arithmetic mean of the ensemble member's predicted distributions

Boosting versus Bagging

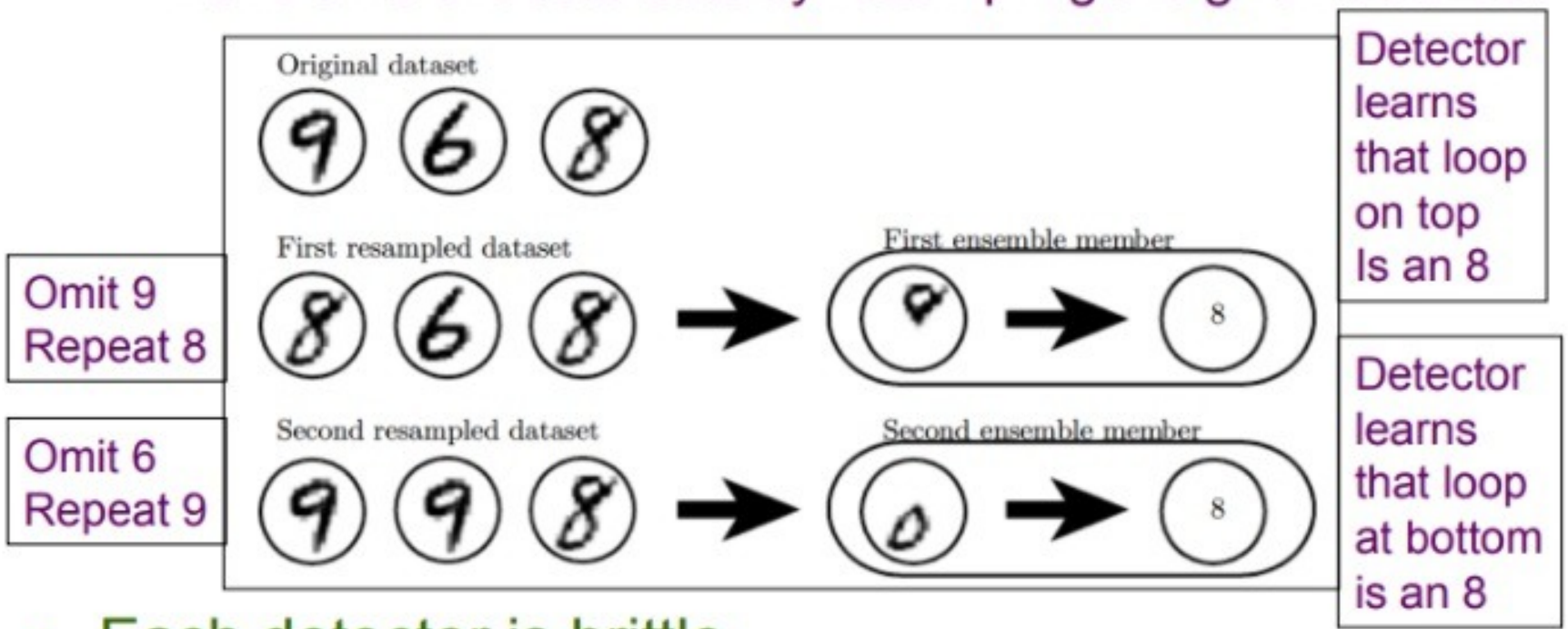
- Base classifiers are trained in sequence
- Each base classifier trained using a weighted form of the dataset
  - Weighting coefficient depends on the performance of the previous classifiers
  - Points misclassified by one of the classifiers is given more weight when used to train next classifier
- Decisions are combined using a weighted majority weighting scheme

Use of parameter tying

- Approach was used for regularizing the parameters of one model, trained as a supervised classifier, to be close to the parameters of another model, trained in an unsupervised paradigm (to capture the distribution of the input data)
  - Ex. of unsupervised learning: k-means clustering
    - Input $\boldsymbol{x}$ is mapped to a one-hot vector $\boldsymbol{h}$. If $\boldsymbol{x}$ belongs to cluster i then $h_i=1$ and rest are zero corresponding to its cluster
      - It could trained using an autoencoder with k hidden units

Example of Bagging Principle

- Task of training an 8 detector
- Bagging training procedure
  - make different data sets by resampling the given data set



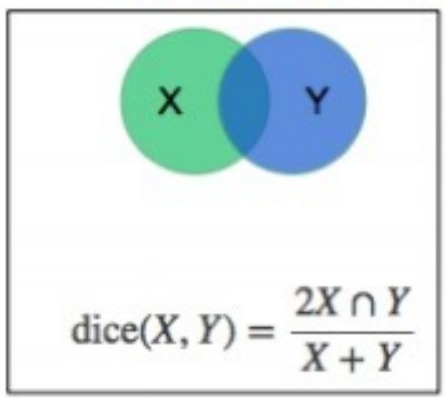
- Each detector is brittle
- Their average is robust achieving maximum confidence when both loops are present

Adversarial examples

- An optimization procedure is used to search for an input x' near data point x such that the model output is very different at x'
- x can be so similar to x' that a human observer cannot tell the difference between the original example and the adversarial example
- But the network makes a highly different prediction

Peformance Evaluation

- Training: 20% of images as validation set
  - RV challenge: separate test set of another 514 MRI images derived from a separate set of 32 patients
- Performance metric
  - The model will output a mask X delineating what it thinks is the RV, and the dice coefficient compares it to the mask Y produced by a physician via:



Metric is (twice) the ratio of the intersection over the sum of areas.
It is 0 for disjoint areas, and 1 for perfect agreement.

E.g., model performance is written as $0.82\ (0.23)$, where the parentheses contain the standard deviation.

Summary of Dropout

- Dropout is a very good and fast regularization method.
- Dropout is a bit slow to train (2-3 times slower than without dropout).
- If the amount of data is average-large – dropout excels. When data is big enough, dropout does not help much.
- Dropout achieves better results than former used regularization methods (Weight Decay).
- Dropout can be used on nearly all NN models.

Boosting

- Powerful technique for combining multiple "base" classifiers to form a committee whose performance can be significantly better than any of the base classifiers
- *AdaBoost* (adaptive boosting) can give good results even if base classifier performance is only slightly better than random
- Hence base classifiers are known as *weak learners*
- Designed for classification, can be used for regression as well

Augmentation for classification

- Data augmentation is easiest for classification
  - Classifier takes high-dimensional input $\boldsymbol{x}$ and summarizes it with a single category identity y
  - Main task of classifier is to be invariant to a wide variety of transformations
- Generate new samples $(\boldsymbol{x}, y)$ just by transforming inputs
- Approach not easily generalized to other problems
  - For density estimation problem
    - it is not possible generate new data without solving density estimation

Popularity of model averaging

- They are an extremely powerful method of reducing generalization error
- Discouraged for benchmarking different algorithms for scientific papers
  - because any learning method can benefit substantially from model averaging at the expense of increased cost and memory
  - Contests are usually won by ensemble methods

Saving Parameters

- We can thus obtain a model with better validation set error (and thus better test error) by returning to the parameter setting at the point of time with the lowest validation set error.
- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather than the latest set

Boosting

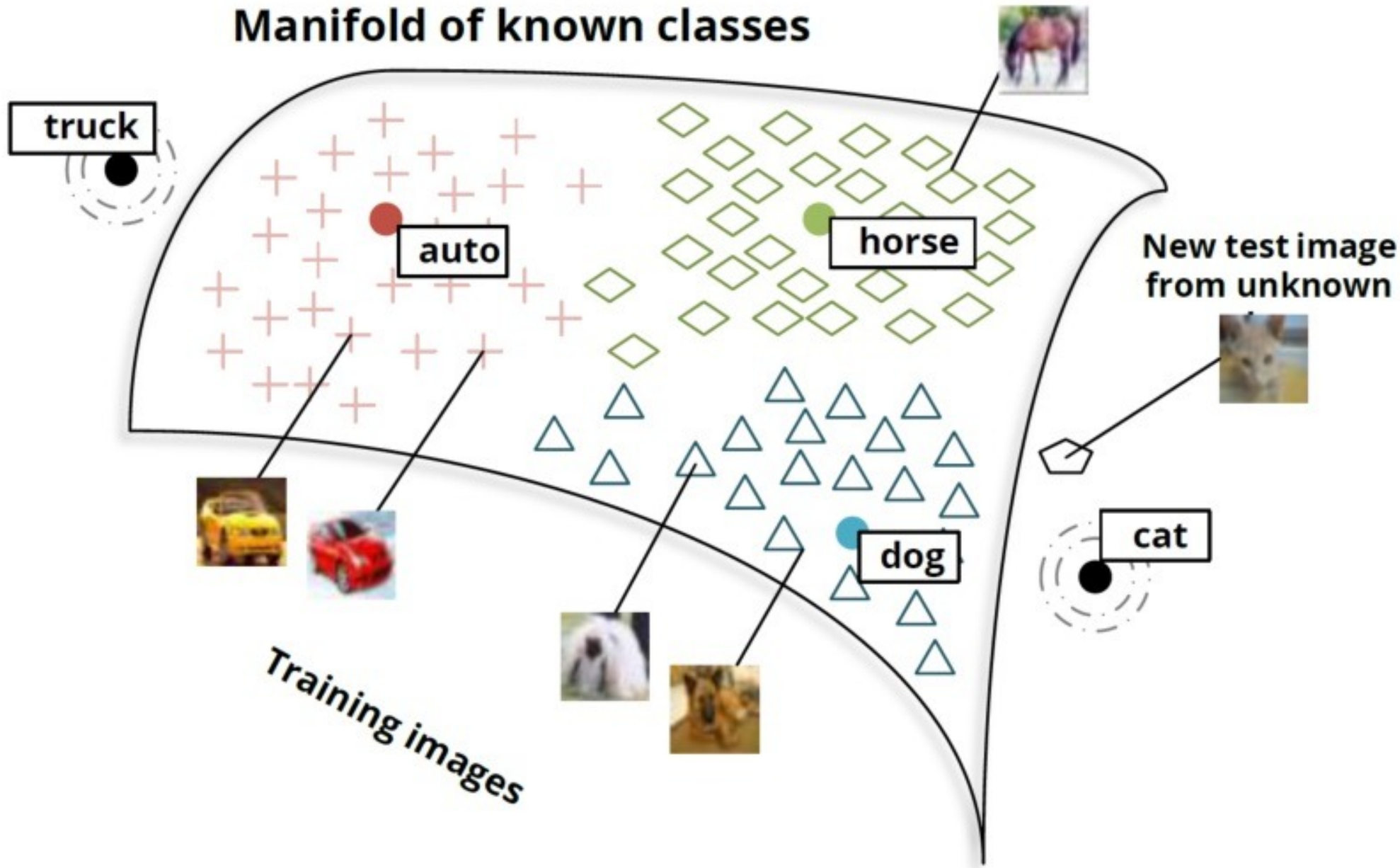
- Not all techniques for constructing ensembles are designed to make ensembles more regularized than individual models
- Boosting constructs an ensemble with higher capacity than individual models
- Boosting has been applied to ensembles of neural networks
  - By incrementally adding neural networks to the ensemble
  - By interpreting an individual neural network as an ensemble, incrementally adding hidden units

Human-level understanding?

- In some tasks, neural networks have begun to reach human level performance when evaluated on i.i.d. test set
- Have they reached human level understanding
- To probe the level of understanding we can probe examples misclassified
- Even neural networks that perform at human level accuracy have a 100% error rate on examples intentionally constructed!

Injecting noise

- Injecting noise into the input of a neural network can be seen as data augmentation
- Neural networks are not robust to noise
- To improve robustness, train them with random noise applied to their inputs
  - Part of some unsupervised learning, such as denoising autoencoder
- Noise can also be applied to hidden units
- Dropout, a powerful regularization strategy, can be viewed as constructing new inputs by multiplying by noise


Manifold of known classes
truck
auto
horse
New test image
from unknown
dog
cat
Training images

Sharing parameters over tasks

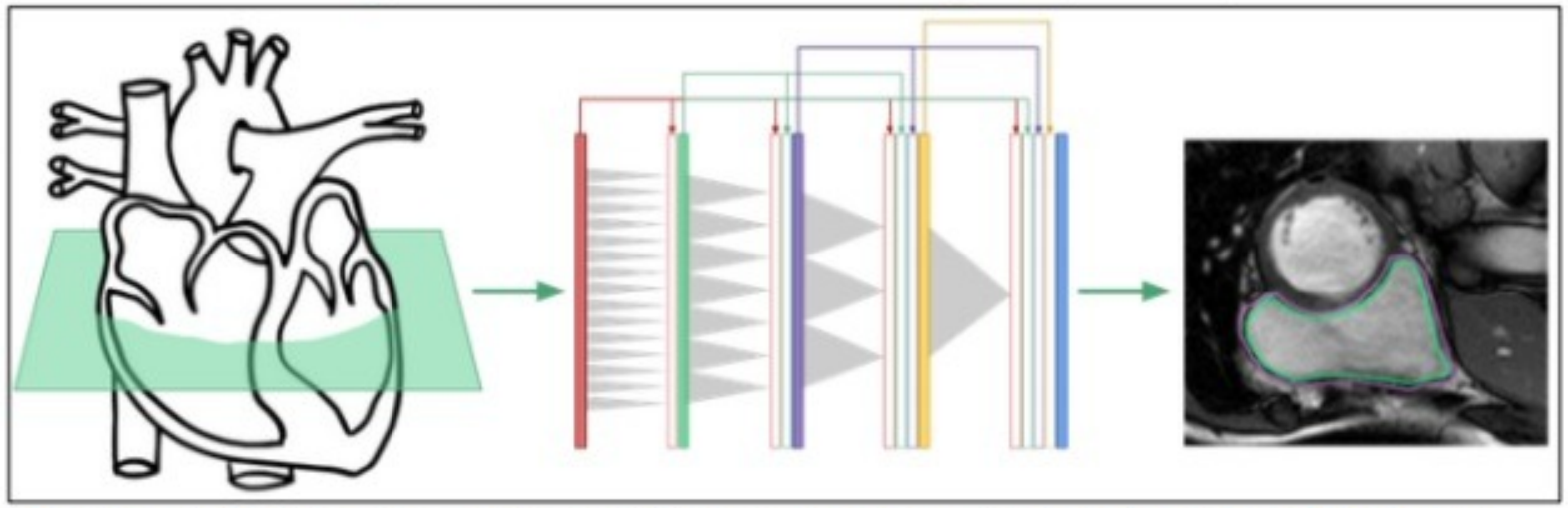
- Multi-task learning is a way to improve generalization by pooling the examples out of several tasks
  - Examples can be seen as providing soft constraints on the parameters
- In the same way that additional training examples put more pressure on the parameters of the model towards values that generalize well

Cause of adversarial examples

- Excessive linearity is the cause
- Neural networks are built primarily out of linear building blocks
- The overall function often proves to be linear
- Linear functions are easy to optimize
- But the value of a linear function can change rapidly with numerous inputs
  - If we change input by ε then a linear functions with weights w can change by $\varepsilon ||w||$ which can be large

Ex: Image segmentation for heart disease

- To determine *ejection fraction*: which measures of how well a heart is functioning
  - After relaxing to its *diastole* so as to fully fill with blood, what percentage is pumped out upon contracting to its *systole?*
    - This metric relies on segmenting right ventricles (RVs) in cardiac magnetic resonance images (MRIs)



https://blog.insightdatascience.com/heart-disease-diagnosis-with-deep-learning-c2d92c27e730

Benchmarking using augmentation

- Hand-designed data set augmentation can dramatically improve performance
- When comparing ML algorithms A and B, same data set augmentation should be used for both
  - If A performs poorly with no dataset augmentation and B performs well with synthetic transformations of the input, reason may be the data set rather than algorithm
- Adding Gaussian noise is considered part of ML while cropping input images is not

Use of parameter sharing in CNNs

- Most extensive use of parameter sharing is in convolutional neural networks (CNNs)
- Natural images have many statistical properties that are invariant to translation
  - Ex: photo of a cat remains a photo of a cat if it is translated one pixel to the right
  - CNNs take this property into account by sharing parameters across multiple image locations
  - Thus we can find a cat with the same cat detector whether the cat appears at column i or column $i+1$ in the image

Neural networks and bagging

- Neural nets can benefit from model averaging
- They reach a wide variety of solutions for a given training set, depending on differences in random initialization, random minibatch selection, hyperparameter selection
- Differences are enough to cause different members of the ensemble to make partially independent errors

No penalty for biases

- Norm penalty Ω penalizes only weights at each layer and leaves biases unregularized
  - Biases require less data to fit than weights
  - Each weight specifies how variables interact
    - Fitting weights requires observing both variables in a variety of conditions
- Each bias controls only a single variable
  - We do not induce too much variance by leaving biases unregularized
- $\boldsymbol{w}$ indicates all weights affected by norm penalty
- $\boldsymbol{\theta}$ denotes both $\boldsymbol{w}$ and biases

The bagging algorithm

- Bagging involves constructing k different data sets
- Each data set has the same number of samples
- Each data set is constructed by sampling with replacement from the original data set
  - With high probability each data set is missing some samples and contains several duplicates
  - On average 2/3 of the samples in the original dataset are found in the resulting training set

Effective for Object Recognition

- Data set augmentation very effective for the classification problem of object recognition
- Images are high-dimensional and include a variety of variations, may easily simulated
- Translating the images a few pixels can greatly improve performance
  - Even when designed to be invariant using convolution and pooling
- Rotating and scaling are also effective

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of subnetworks are trained for a single step
    - Parameter sharing allows good parameter settings

Noise injection is powerful

- Noise applied to inputs is a data augmentation strategy
  - For some models addition of noise with infinitesimal variance at the input is equivalent to imposing a penalty on the norm of the weights
- Noise injection can be much more powerful than simply shrinking the parameters
  - Especially when noise is applied to hidden units
- Noise applied to hidden units is so important that it merits its own separate discussion
  - Dropout is the main development of this approach

So What's Going on Here?

- Fraction of adversarial samples generalize **across** different models (i.e., convolutional neural networks)
  - Not a property of overfitting
- "Fooling subspaces"—could be thought of as statistical learning "optical illusions"
  - Not just limited to image data (e.g. binary data, as in malware classification)
  - Note: discrete variables (as in text data) make it difficult to directly generate adversarial noise
- Problem: the linearity might be to blame (i.e., those linear rectifier activation functions might be a problem)

How unsupervised learning helps

- Unsupervised learning can provide useful clues for how to group examples in representational space
- Examples that cluster tightly in the input space should be mapped to similar representations
- A linear classifier in the new space may achieve better generalization
- A variant is the application of PCA as a preprocessing step before applying a classifier to the projected data

Imposing Penalties Indirectly

- Weight decay penalizes parameters directly
- Another strategy is to place penalty on the activations of the units in the neural network
  - Encouraging their activations to be sparse
  - This indirectly imposes a complicated penalty on the model parameters
- We have seen how L^1 penalization induces sparse parameterization
- Representational sparsity describes a representation where many of the elements of the representation are close to zero

2. Stability of Optimization

- Explicit constraints with reprojection can be useful because thse impose some stability on the optimization procedure
- When using high learning rates, it is possible to encounter a positive feedback learning loop in which large weights induce large gradients, which then induce a large update of the weights
  - Can lead to numerical overflow
- Explicit constraints with reprojection prevent this feedback loop from continuing to increase magnitudes of weights without bound

Constrained Optimization

- Consider the cost function regularized by a norm penalty $\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$
- Recall we can minimize a function subject to constraints by constructing a generalized Lagrange function, consisting of the original objective function plus a set of penalties
  - Each penalty is a product between a coefficient called a Karush-Kuhn-Tucker (KKT) multiplier and a function representing whether constraint is satisfied

Sharing Parameters

- Instead of separate unsupervised and supervised components in the model, construct models in which generative models of either $P(\boldsymbol{x})$ or $P(\boldsymbol{x}, y)$ shares parameters with a discriminative model of $P(y|\boldsymbol{x})$
- One can then trade-off the supervised criterion $-\log P(y|\boldsymbol{x})$ with the unsupervised or generative one (such as $-\log P(\boldsymbol{x})$ or $-\log P(\boldsymbol{x}, y)$
  - The generative criterion then expresses a prior belief about the solution to the supervised problem
    - viz., structure of $P(\boldsymbol{x})$ is connected to structure of $P(y|\boldsymbol{x})$ in a way that is captured by shared parameterization

Summary

Training Error	Valid Error	Cause	Solution
High	High	High bias	- Increase model complexity - Train for more epochs
Low	High	High variance	- Add more training data (e.g., **dataset augmentation**) - Use **regularization** - User **early stopping** (train less)
Low	Low	Perfect tradeoff	- You are done!

Summary

Training Error	Valid Error	Cause	Solution
High	High	High bias	- Increase model complexity - Train for more epochs
Low	High	High variance	- Add more training data (e.g., **dataset augmentation**) - Use **regularization** - User **early stopping** (train less)
Low	Low	Perfect tradeoff	- You are done!

Illustration of L^2 regularization

Effect on value of optimal $\boldsymbol{w}$

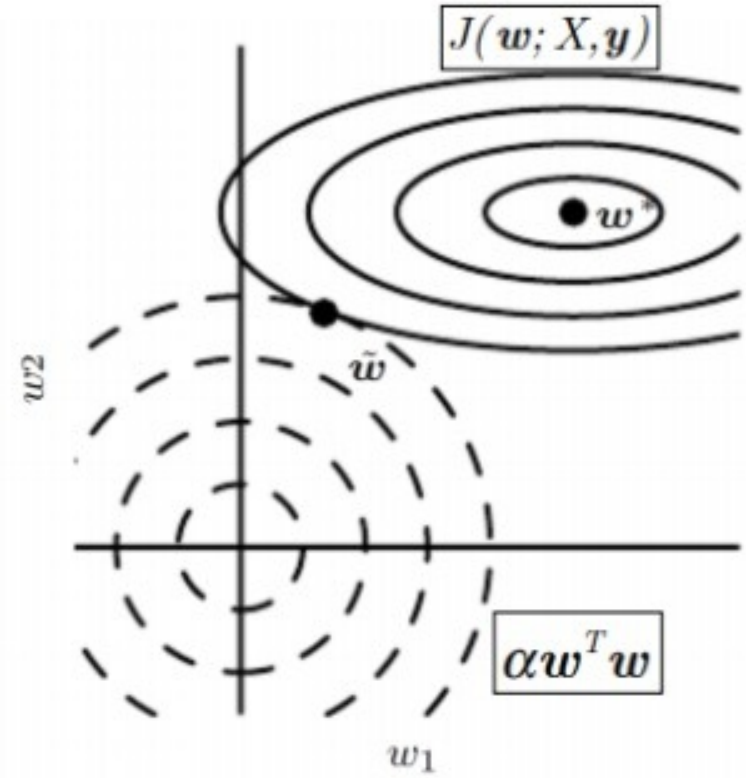
Solid ellipses:

contours of equal value of unregularized objective J

Dotted circles:

contours of equal value of L^2 regularizer

At point $\boldsymbol{w}$ competing objectives reach equilibrium



Along w_1, eigen value of Hessian of J is small. J does not increase much when moving horizontally away from $\boldsymbol{w}^*$. Because J does not have a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w_1 close to 0.

Along w_2, J is very sensitive to movements away from $\boldsymbol{w}^*$. The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little

Uses of adversarial training

- They are useful in computer security
- Using adversarially perturbed samples is interesting in regularization
  - To reduce error rate on test set
- Adversarial examples are hard to defend against

Motivations: The "Blind Spot" Problem

- Adversarial samples = data points that can "trick" the model into making incorrect predictions w/ high confidence
  - No model is safe, including simple linear classifiers
- Created by often imperceptible/tiny noise patterns
  - adversary could circumvent filters (i.e., content filters) by fool server-side model
  - representation of input?

We add to $\boldsymbol{x}$ an imperceptibly small vector
Its elements are equal to the sign of the elements of the gradient of the cost function wrt the input. It changes Googlenet's classification of the image

$\boldsymbol{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

http://karpathy.github.io/2015/03/30/breaking-convnets/

Motivations: The "Blind Spot" Problem

- Adversarial samples = data points that can "trick" the model into making incorrect predictions w/ high confidence
  - No model is safe, including simple linear classifiers
- Created by often imperceptible/tiny noise patterns
  - adversary could circumvent filters (i.e., content filters) by fool server-side model
  - representation of input?

We add to $\boldsymbol{x}$ an imperceptibly small vector
Its elements are equal to the sign of the elements of the gradient of the cost function wrt the input. It changes Googlenet's classification of the image

$\boldsymbol{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

http://karpathy.github.io/2015/03/30/breaking-convnets/

Tangent Propagation as Regularization

- Under a transformation of input vector
  - The network output vector will change
  - Derivative of output k wrt ξ is given by

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

  - where J_{ki} is the (k,i) element of the Jacobian Matrix J
- Result is used to modify the standard error function
  - So as to encourage local invariance in neighborhood of data point

$$\tilde{E} = E + \lambda \Omega$$

where λ is a regularization coefficient and

$$\Omega = \frac{1}{2} \sum_n \sum_k \left(\left.\frac{\partial y_{nk}}{\partial \xi}\right|_{\xi=0} \right)^2 = \frac{1}{2} \sum_n \sum_k \left(\sum_{i=1}^{D} J_{nki} \tau_{ni} \right)^2$$

Bootstrapping

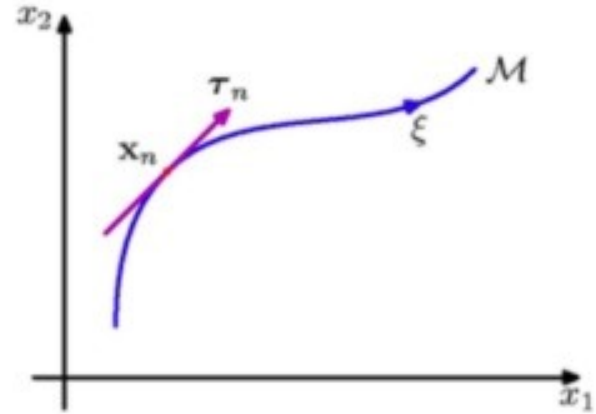
- Bootstrap is to draw repeated samples (of the same size) from the population of interest, a large number of times. The idea behind bootstrap is to use the data of a sample study at hand as a “surrogate population”, for the purpose of approximating the sampling distribution of a statistic; i.e. to resample (with replacement) from the sample data at hand and create a large number of “phantom samples” known as bootstrap samples.
- In other words, We randomly sample with replacement from the n known observations. We then call this a bootstrap sample. Since we allow for replacement, this bootstrap sample is most likely not identical to our initial sample. Some data points may be duplicated, and others data points from the initial may be omitted in a bootstrap sample.

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Tangent Methods

- Regularization can be used to encourage models to be invariant to transformations
  - by techniques of tangent propagation
- Consider effect of a transformation on input vector $\mathbf{x}_n$
  - A one-dimensional continuous transformation parameterized by ξ applied to $\mathbf{x}_n$ sweeps a manifold $\mathcal{M}$ in D-dimensional input space



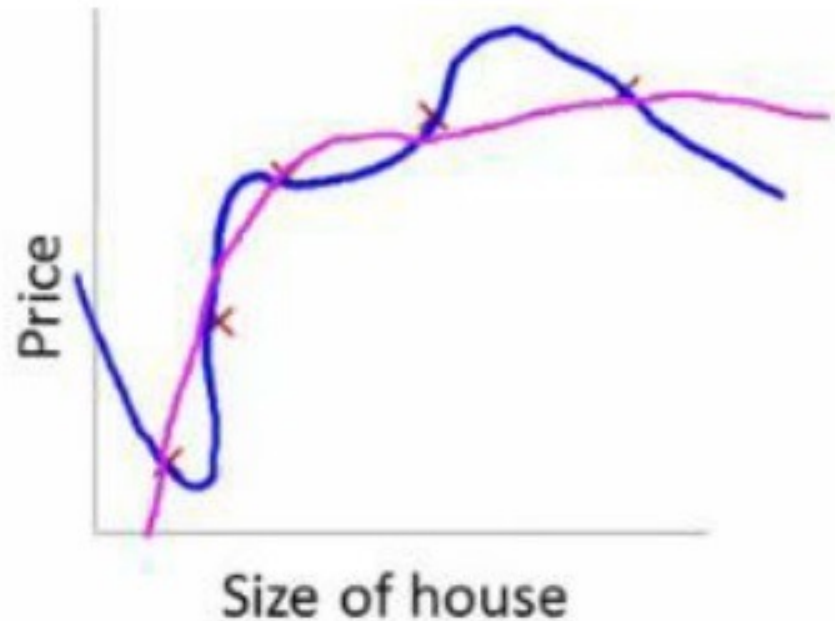
Two-dimensional input space showing effect of continuous transformation with single parameter ξ

Let the vector resulting from acting on $\mathbf{x}_n$ by this transformation be denoted by $s(\mathbf{x}_n,\xi)$ defined so that $s(\mathbf{x},0)=\mathbf{x}$.

Then the tangent to the curve $\mathcal{M}$ is given by the directional derivative $\tau = \delta s/\delta\xi$ and the tangent vector at point $\mathbf{x}_n$ is given by

$$\tau_n = \left.\frac{\partial s(x_n,\xi)}{\partial \xi}\right|_{\xi=0}$$

- Many strategies for regularization
  - Parameter norm penalties / priors & norm penalties as constraints
  - Dataset augmentation
  - Noise robustness
  - Semi-supervised and multi-task learning
  - Early stopping
  - Parameter tying/sharing
  - Encouraging sparsity
  - Bagging/ensemblling & Dropout
  - Adversarial training



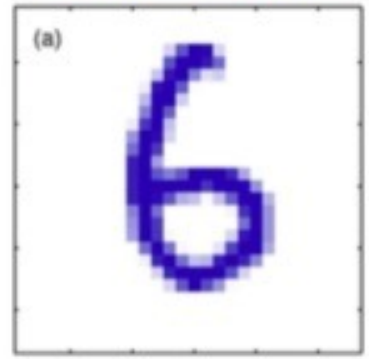
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Tangent vector from finite differences

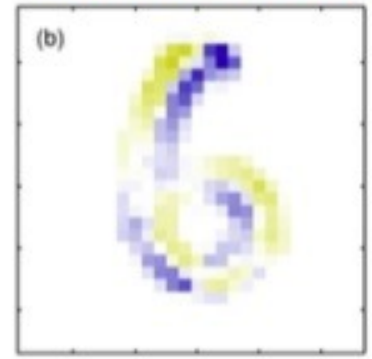
- In practical implementation tangent vector τ_n is approximated
  using finite differences
  by subtracting original vector $\mathbf{x}_n$
  from the corresponding vector
  after transformations using
  a small value of ξ and then
  dividing by ξ

- A related technique
  Tangent Distance is used
  to build invariance properties
  into distance-based
  methods such as
  nearest-neighbor classifiers

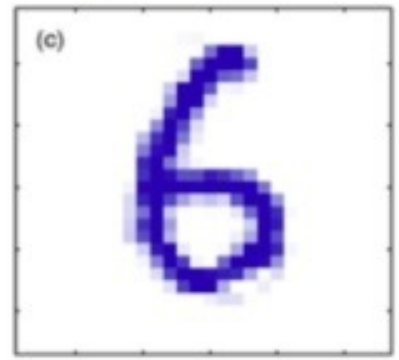
Original Image $\mathbf{x}$



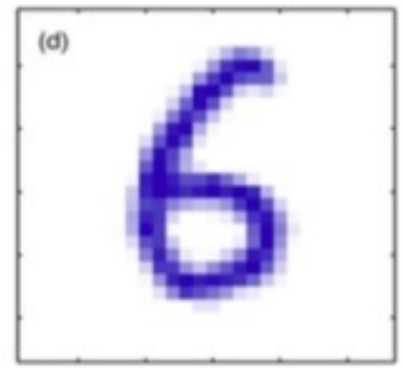
Tangent vector τ
corresponding to
small clockwise rotation

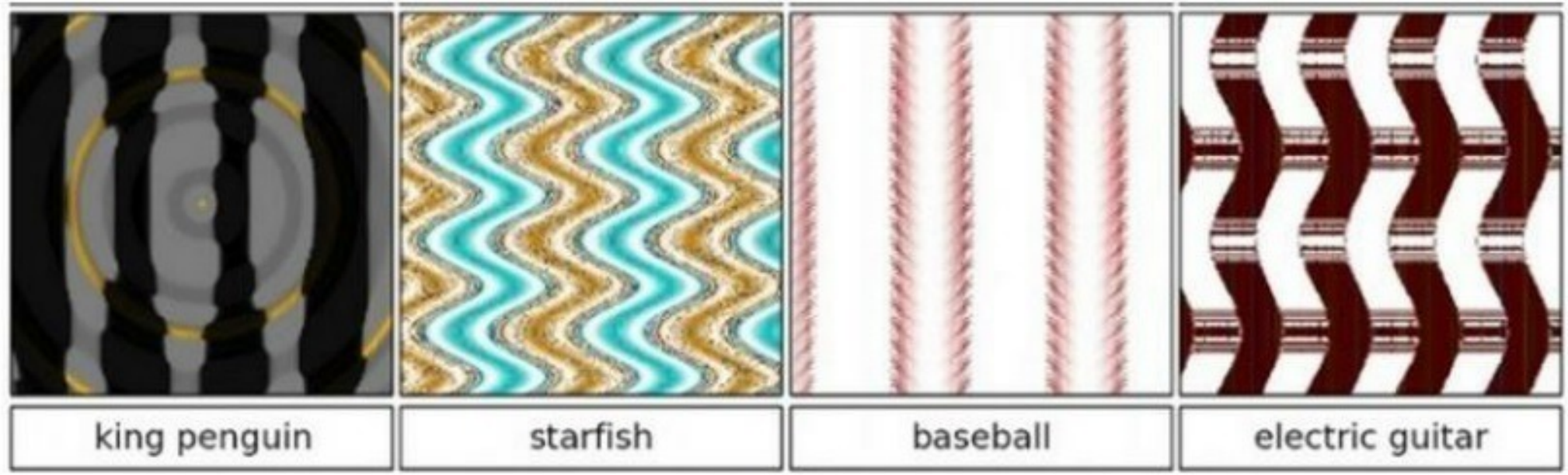


Adding small contribution
from tangent vector
to image $\mathbf{x}+\varepsilon\tau$

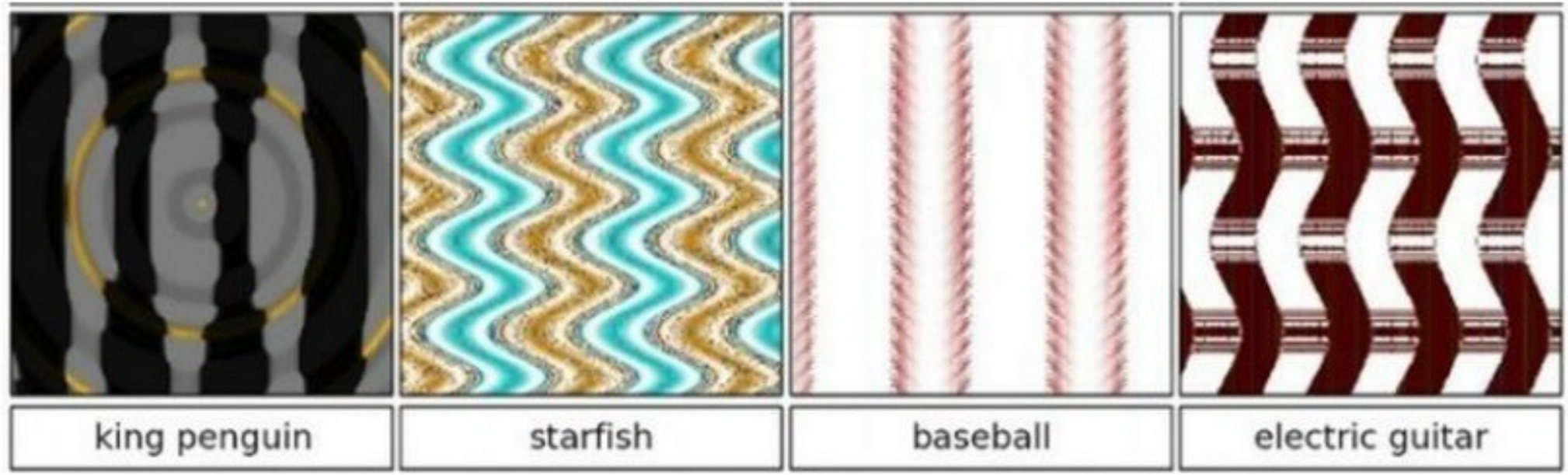


True image rotated
for comparison





http://karpathy.github.io/2015/03/30/breaking-convnets/



http://karpathy.github.io/2015/03/30/breaking-convnets/

Approaches to Invariance

1. Training set augmented by transforming training patterns according to desired invariances

   E.g., shift each image into different positions

2. Add regularization term to error function that penalizes changes in model output when input is transformed.

   Leads to tangent propagation.

3. Invariance built into pre-processing by extracting features invariant to required transformations

4. Build invariance property into structure of neural network (convolutional networks)

   Local receptive fields and shared weights

Why Adversarial Samples?

Explaining and Harnessing Adversaria Examples (Goodfellow et al., 2014)

- Neural models still work well in practice but…
  - Competence is relatively limited to a small region around data manifold that contains natural-looking images and distributions
  - Once we artificially push images away from this manifold by computing noise patterns with backpropagation, stumble into parts of space where:
    - Linear functions in network induce large subspaces of fooling inputs.
    - All bets are off!
- Tension between models that are easy to train (e.g. use linear functions) and models that resist adversarial perturbations
- Make many infinitesimal changes to input ↘ add up to one large change to the output
  - Linear model is forced to attend exclusively to signal that aligns most closely with its weights,
  - (Even if multiple signals are present and other signals have much greater amplitude)

- **Regularization** = reduce generalization error (but usually increase training error)
- Goal: combat overfitting (make overfit model match data generating process)
  - Constraints on cost fun, encode prior knowledge
  - Find simpler models (Occam's Razor)
  - Convert underdetermined problem to determined one
  - Reduce variance of estimator w/o increasing bias too much