

Regularization for Deep Learning

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 made to a learning algorithm to reduce generalization error but not training error”
 - Reduce test error even at expense of higher training error

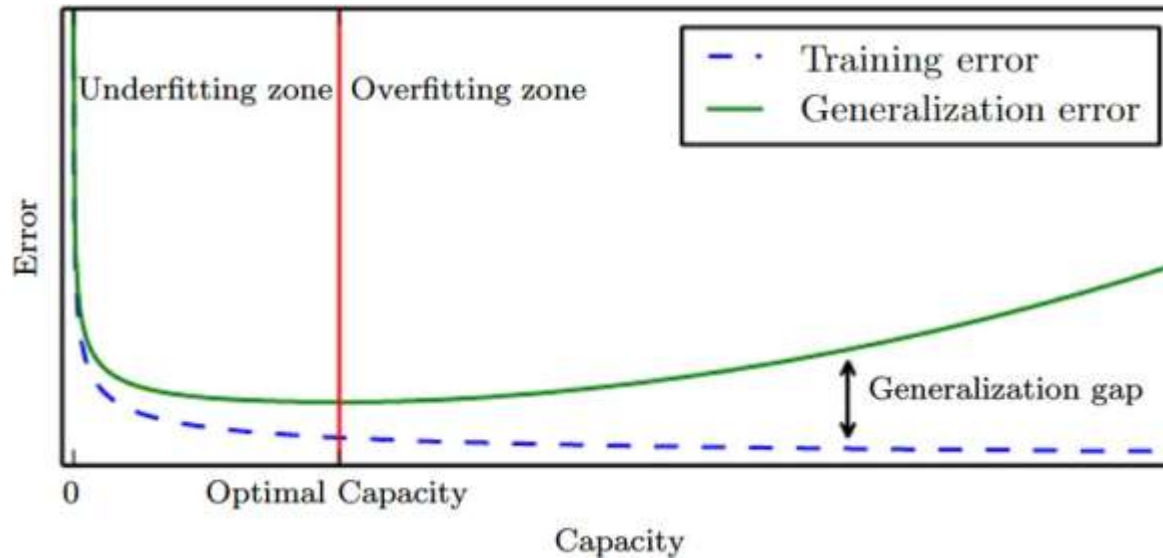
Philosophical view

- Regularization is a recurrent issue in ML
- Hinton borrowed the concept in his neural networked view:
 - used a shocking term "unlearning" to refer to it.
- To achieve a greater effectiveness, one must not learn the idiosyncrasies of the data
- One must remain a little ignorant in order to discover the true behavior of the data

Some 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

Regularization using a simpler model



Regularizing Estimators

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

Model Types and Regularization

- 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

Importance of Regularization

- 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)

What is 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

Regularization Strategies

1. Parameter Norm Penalties
 - (L^2 - and L^1 - regularization)
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness

More Regularization Strategies

- 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

Parameter Norm Penalties

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

Topics in Parameter Norm Penalties

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

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 :

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

- where $\alpha \in [0, \theta)$ 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

Norm Penalty

- When our training algorithm minimizes the regularized objective function

$$J^{\lambda}(\theta; X, y) = J(\theta; X, y) + \lambda \Omega(\theta)$$

- it will decrease both 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

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
- \mathbf{w} indicates all weights affected by norm penalty
- $\mathbf{\theta}$ denotes both \mathbf{w} and biases

Different or Same α s for layers?

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

$$J^{\mathbf{T}}(\mathbf{W}; X, y) = J(\mathbf{W}; X, y) + \sum \alpha_i \|\mathbf{W}_i\|^2$$

- Invariance under linear transformation \mathbf{T} is one case

– i.e., we want neural net to perform the same when the inputs are transformed

- But this creates too many hyperparameters
 - Search space reduced by using same hyperparameters

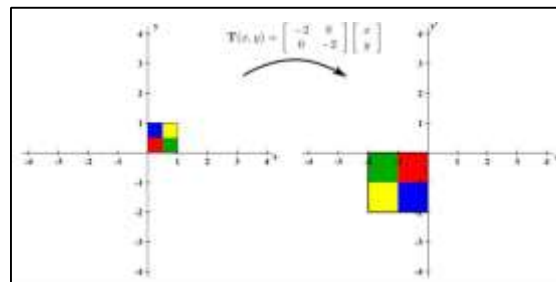
Linear Transformation T

- Consider a simple linear transformation of the input

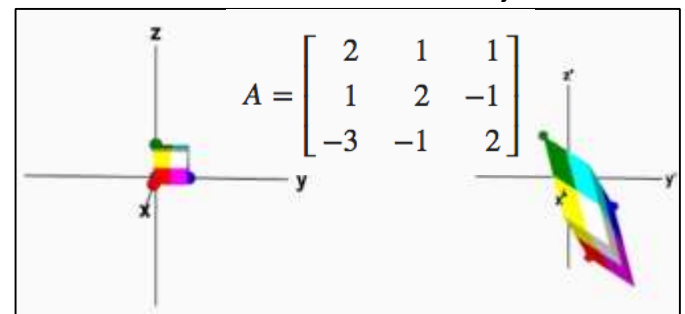
Two-variables
 x and y

$$T(x, y) = (ax + by, cx + dy) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$T(\mathbf{x}) = A\mathbf{x}$$



Three variables x, y and z



Weight decay and invariance

- Suppose we train two 2-layer networks
 - *First network*: trained using original data: $\mathbf{x}=\{x_i\}$, $\mathbf{y}=\{y_k\}$
 - *Second network*: input and/or target variables are transformed by one of the linear transformations

$$x_i \rightarrow \hat{x}_i = ax_i + b$$

$$y_k \rightarrow \hat{y}_k = cy_k + d$$

- Consistency requires that we should obtain equivalent networks that differ only by linear transformation of the weights

For first layer:

$$\begin{array}{c} w_{ji} \quad a_{ji} \\ \rightarrow \mathbf{1}_w \end{array}$$

And/or or second layer:

and/or

$$w_{kj} \rightarrow cw_{kj}$$

Simple weight decay fails invariance

- Simple weight

$$E^d(\mathbf{w}) = E(\mathbf{w}) + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w}$$

- Treats all weights and biases on equal footing

- While resulting w_{ji} and w_{kj} should be treated differently
- Consequently networks will have different weights and violate invariance

- We therefore look for a regularizer invariant under the linear transformations

– Such a regularizer is

$$\frac{\alpha_1}{2} \sum_{w \in W_1} w^2 + \frac{\alpha_2}{2} \sum_{w \in W_2} w^2$$

- where W_1 are weights of first layer and
- W_2 are the set of weights in the second layer
 - This regularizer remains unchanged under the weight transformations provided the parameters are rescaled using

$$\lambda_1 \rightarrow a^{1/2} \lambda_1 \quad \text{and} \quad \lambda_2 \rightarrow c^{-1/2} \lambda_2$$

Weight decay used in practice

- 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

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 objective function

$$\hat{J}(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2$$

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

Gradient of Regularized Objective

- Objective function (with no bias parameter)

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

- Corresponding parameter gradient

$$\frac{\partial}{\partial w} J(w; X, y) = \lambda w + \frac{\partial}{\partial w} J(w; X, y)$$

- To perform single gradient step, perform update:

$$w \leftarrow w - \epsilon \left(\lambda w + \frac{\partial}{\partial w} J(w; X, y) \right)$$

- Written another way, the update is

– We have modified learning rule to shrink w by constant factor $1 - \epsilon \lambda$ at each step

To study effect on entire training

- Make quadratic approximation to the objective function in the neighborhood of minimal unregularized cost $\mathbf{w}^* = \arg \min_{\mathbf{w}} J(\mathbf{w})$
- The approximation is given by $J(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T H(\mathbf{w} - \mathbf{w}^*)$
- Where H is the Hessian matrix of J wrt \mathbf{w} evaluated at \mathbf{w}^*

Effect of L^2 regularization on optimal w

Objective function:

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

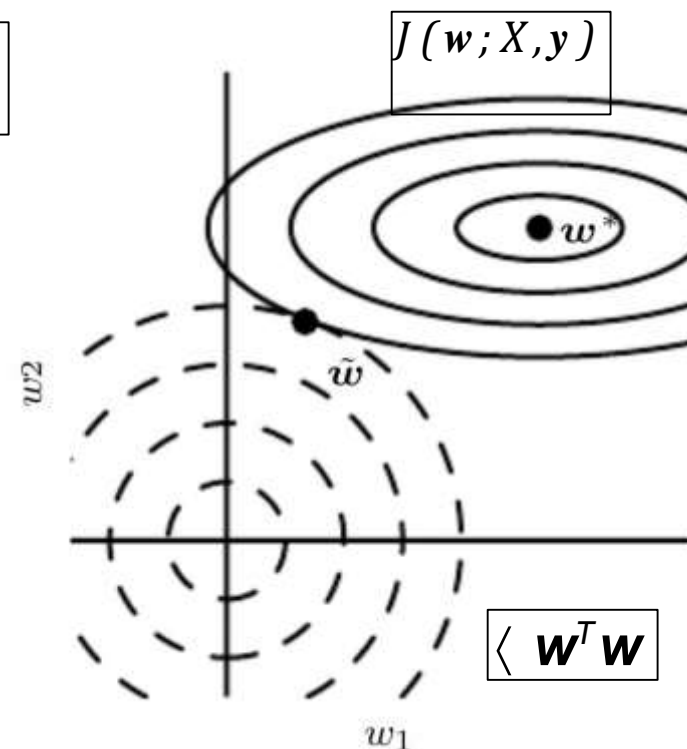
Solid ellipses:

contours of equal value of
unregularized objective $J(w; X, y)$

Dotted circles:

contours of equal value of L^2
regularizer $\langle w^T w \rangle$

At point \tilde{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 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.

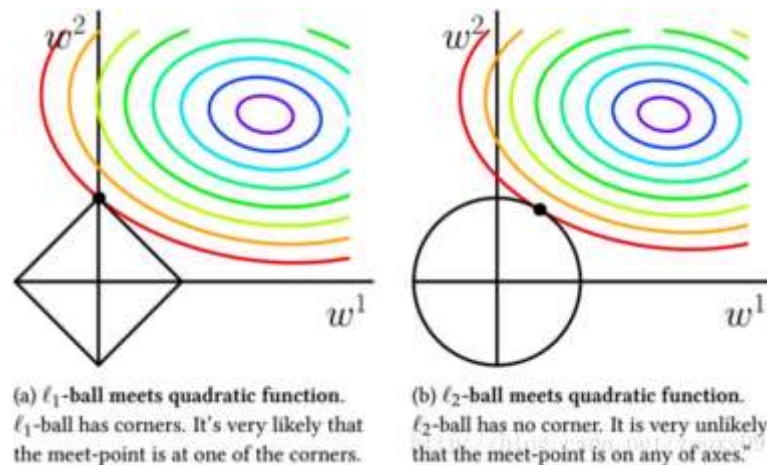
Along w_2 , J is very sensitive to movements away from w^* . The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little

L^1 Regularization

- L^2 weight decay is common weight decay
- Other ways to penalize model parameter size
- L^1 regularization is defined as

$$\Delta(w) = \sum_i |w_i| = \sum_i w_i^1$$

 – which sums the absolute values of parameters



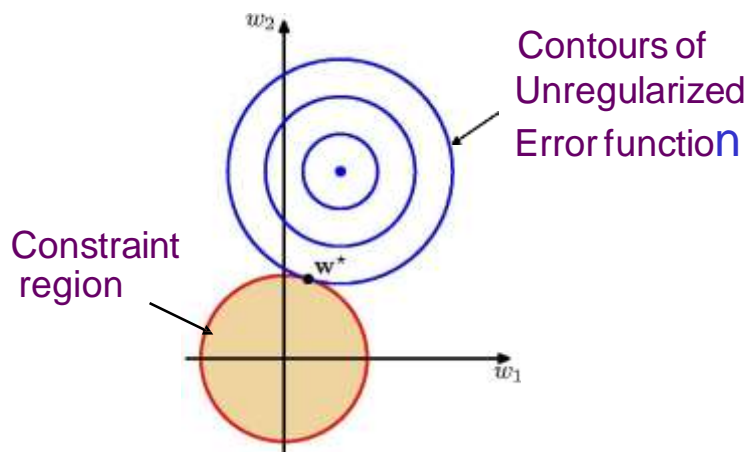
Sparsity and Feature Selection

- The sparsity property induced by L^1 regularization has been used extensively as a feature selection mechanism
 - Feature selection simplifies an ML problem by choosing subset of available features
- LASSO (Least Absolute Shrinkage and Selection Operator) integrates an L^1 penalty with a linear model and least squares cost function
 - The L^1 penalty causes a subset of the weights to become zero, suggesting that those features can be discarded

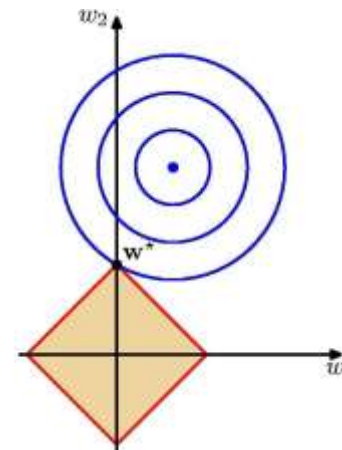
Sparsity with Lasso constraint

- With $q=1$ and λ is sufficiently large, some of the coefficients w_j are driven to zero
- Leads to a sparse model
 - where corresponding basis functions play no role
- Origin of sparsity is illustrated here:

Quadratic solution where w_1^* and w_0^* are nonzero



Minimization with Lasso Regularizer
A sparse solution with $w_1^*=0$



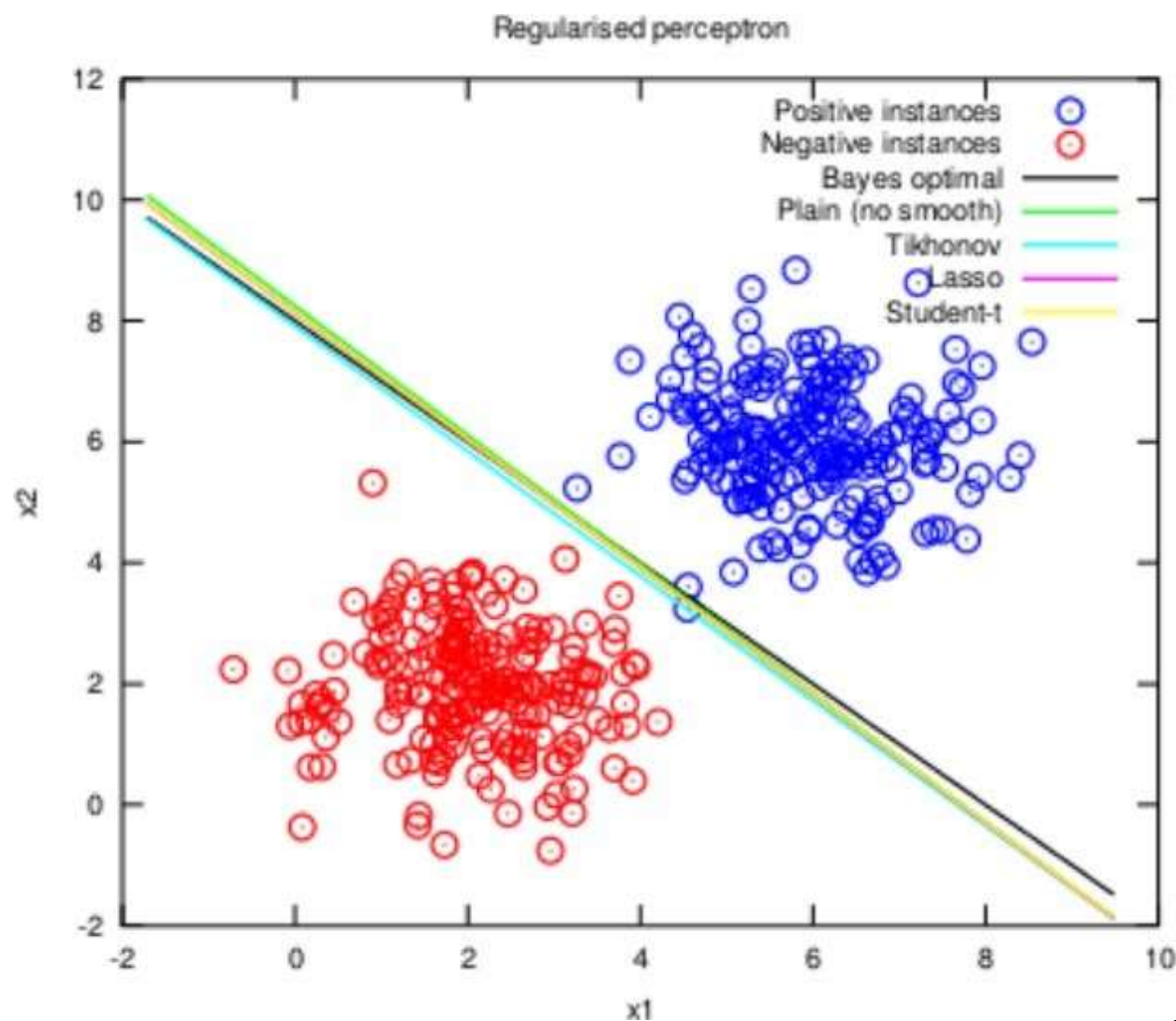
Regularization with linear models

Norm Regularization:

Tikhonov: $\lambda \sum_{\forall i} w_i^2$

Lasso: $\lambda \sum_{\forall i} |w_i|$

Student-t: $\lambda \sum_{\forall i} \log(1+w_i^2)$



Norm Penalties as Constrained Optimization

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

Topics in Norm Penalty Optimization

1. Lagrangian formulation
2. KKT multiplier
3. Equivalence to norm penalty
4. Explicit constraints and Reprojection

Constrained Optimization

- Consider the cost function regularized by norm penalty $J(\theta; X, y) = J(\theta; X, y) + \langle \lambda, \phi(\theta) \rangle$
- 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

Lagrange Formulation

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

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

- The solution to the constrained problem is given by

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

- Solving this problem requires modifying both θ

and λ . Many different procedures are possible

Insight into effect of constraint

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

$$\theta^* = \arg \min_{\theta} L(\theta, \alpha^*) = \arg \min_{\theta} J(\theta; X, y) + \langle \alpha^* \wedge (\theta) \rangle$$

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

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

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

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

2. Stability of Optimization

- Explicit constraints with reprojection can be useful because these 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 magnify the scale of weights without being to increase

Data Set Augmentation

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

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

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









Augmentation for classification

- Data augmentation is easiest for classification
 - Classifier takes high-dimensional input \mathbf{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 (\mathbf{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

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

Main data augmentation methods

Original	Flip	Rotation	Random crop
			
<ul style="list-style-type: none"> • Image without any modification 	<ul style="list-style-type: none"> • Flipped with respect to an axis for which the meaning of the image is preserved 	<ul style="list-style-type: none"> • Rotation with a slight angle • Simulates incorrect horizon calibration 	<ul style="list-style-type: none"> • Random focus on one part of the image • Several random crops can be done in a row
Color shift	Noise addition	Information loss	Contrast change
			
<ul style="list-style-type: none"> • Nuances of RGB is slightly changed • Captures noise that can occur with light exposure 	<ul style="list-style-type: none"> • Addition of noise • More tolerance to quality variation of inputs 	<ul style="list-style-type: none"> • Parts of image ignored • Mimics potential loss of parts of image 	<ul style="list-style-type: none"> • Luminosity changes • Controls difference in exposition due to time of day

Remark: data is usually augmented on the fly during training.

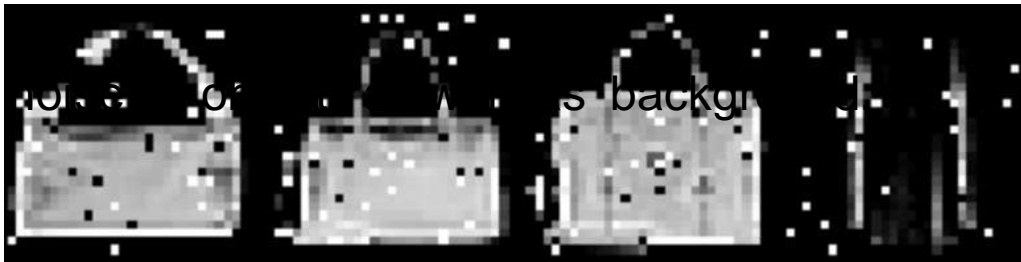
Fashion MNIST Noisy Images

Gaussian Noise



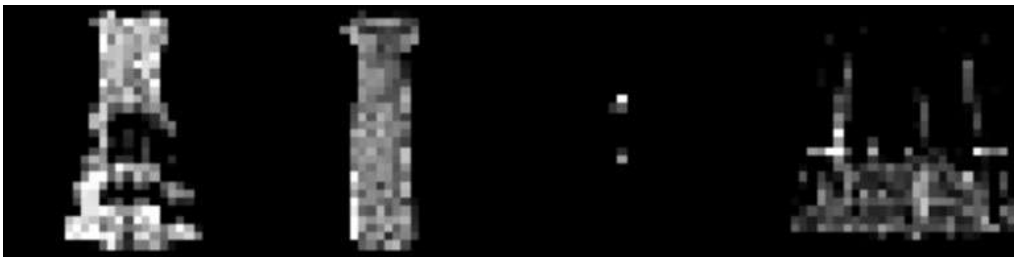
Noise on the objects only and not in the background.

Salt and Pepper Noise



Mixture of black and white

Speckle Noise



https://debuggercafe.com/adding-noise-to-image-data-for-deep-learning-data-augmentation/?fbclid=IwAR2KWCN5HExb8EmpW5Z6vsM_7y6n8le6-rcHvEwia6pze3DLM9hZEKU1arc

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

Injecting noise

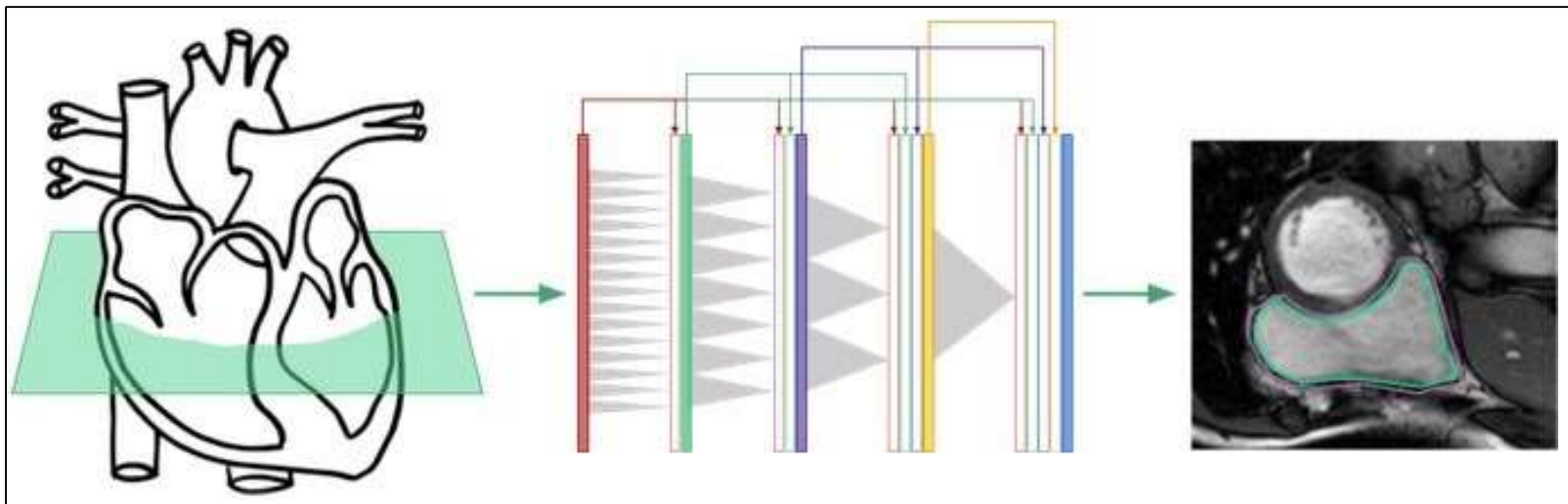
- Injecting noise into the input of a neural network can be seen as data augmentation
- Neural networks are not robust to noise
- To improve robustness, train them with random noise applied to their inputs
 - Part of some unsupervised learning, such as denoising autoencoder
- Noise can also be applied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

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

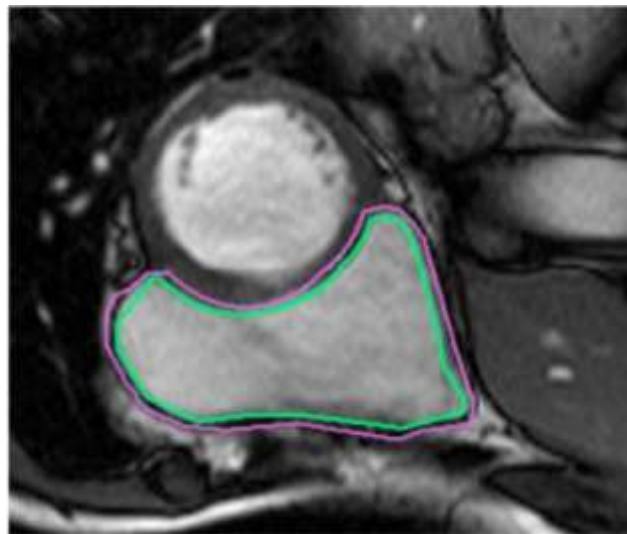
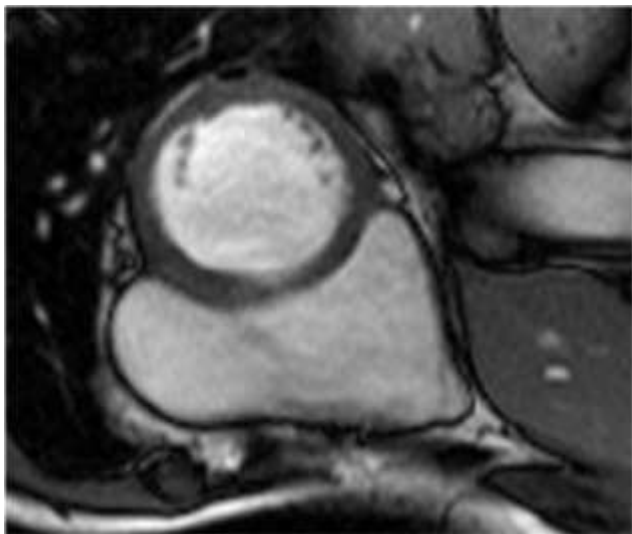
Ex: Image segmentation for heart disease

- 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)



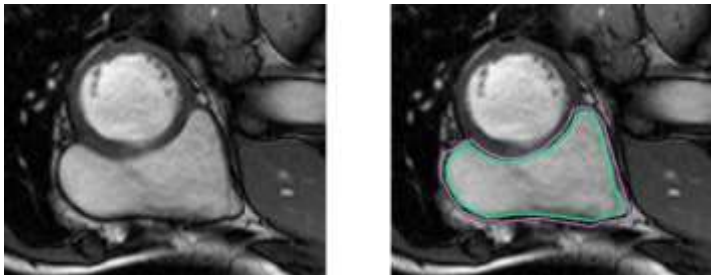
Problem Description

- 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)

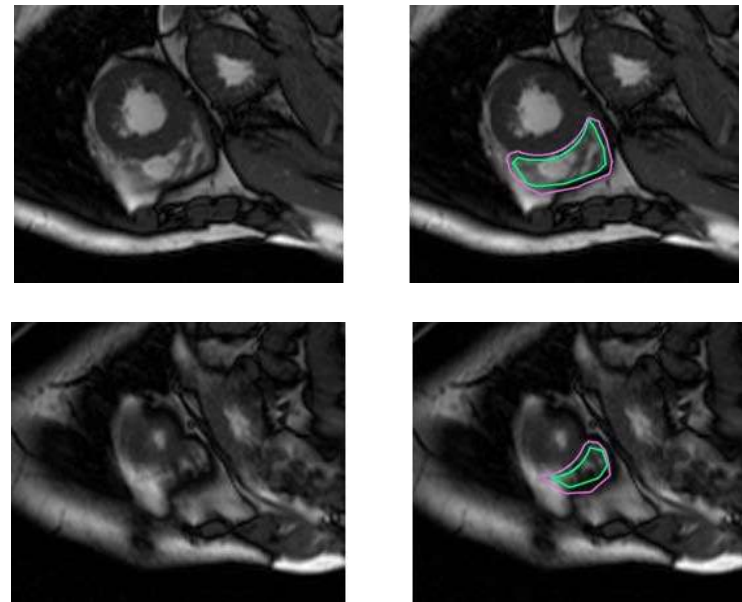


RV segmentation is difficult

- 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



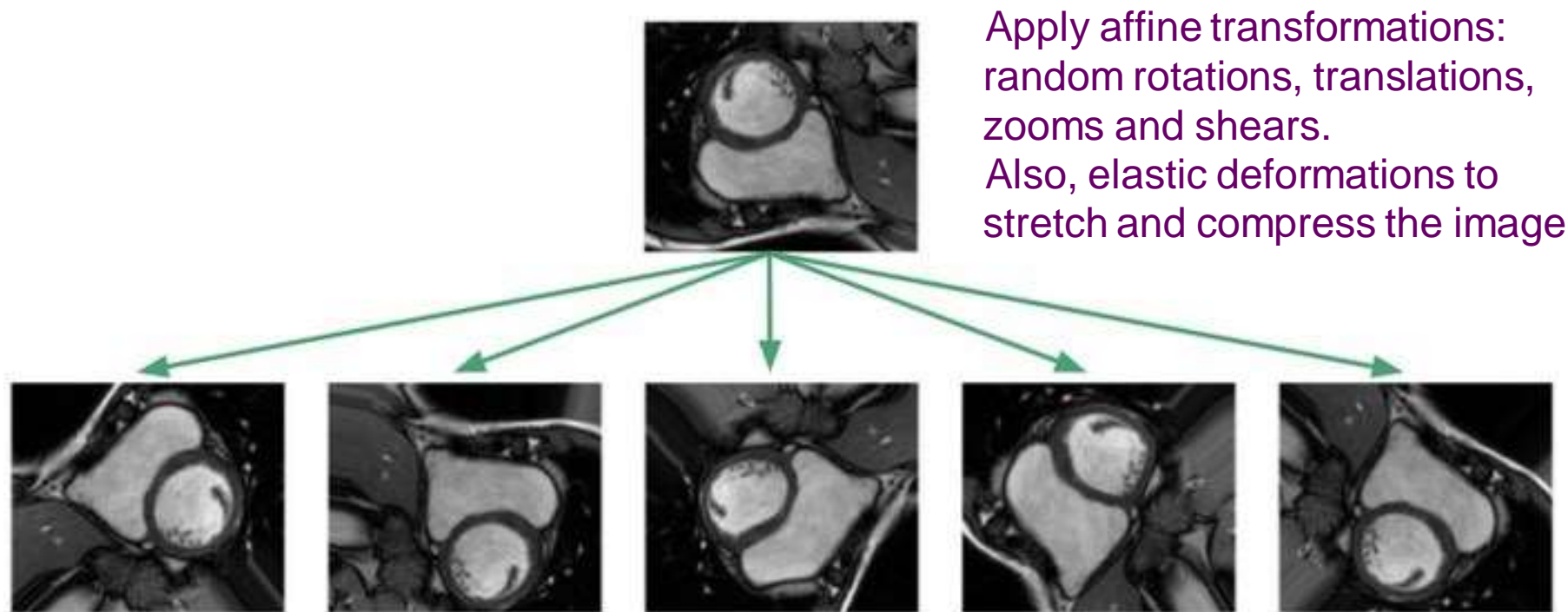
Task: determine whether each pixel is part of RV or not



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.

Transformed data

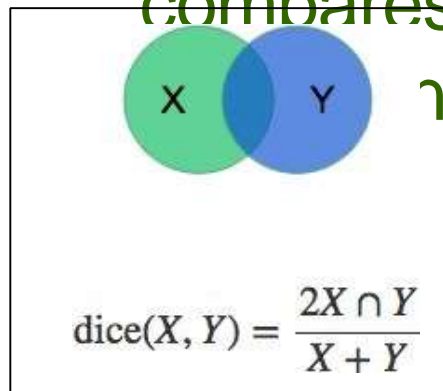


Goal: prevent network from memorizing just the training examples, and force it to learn that the RV is a solid, crescent-shaped object in a variety of orientations.

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

Performance 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

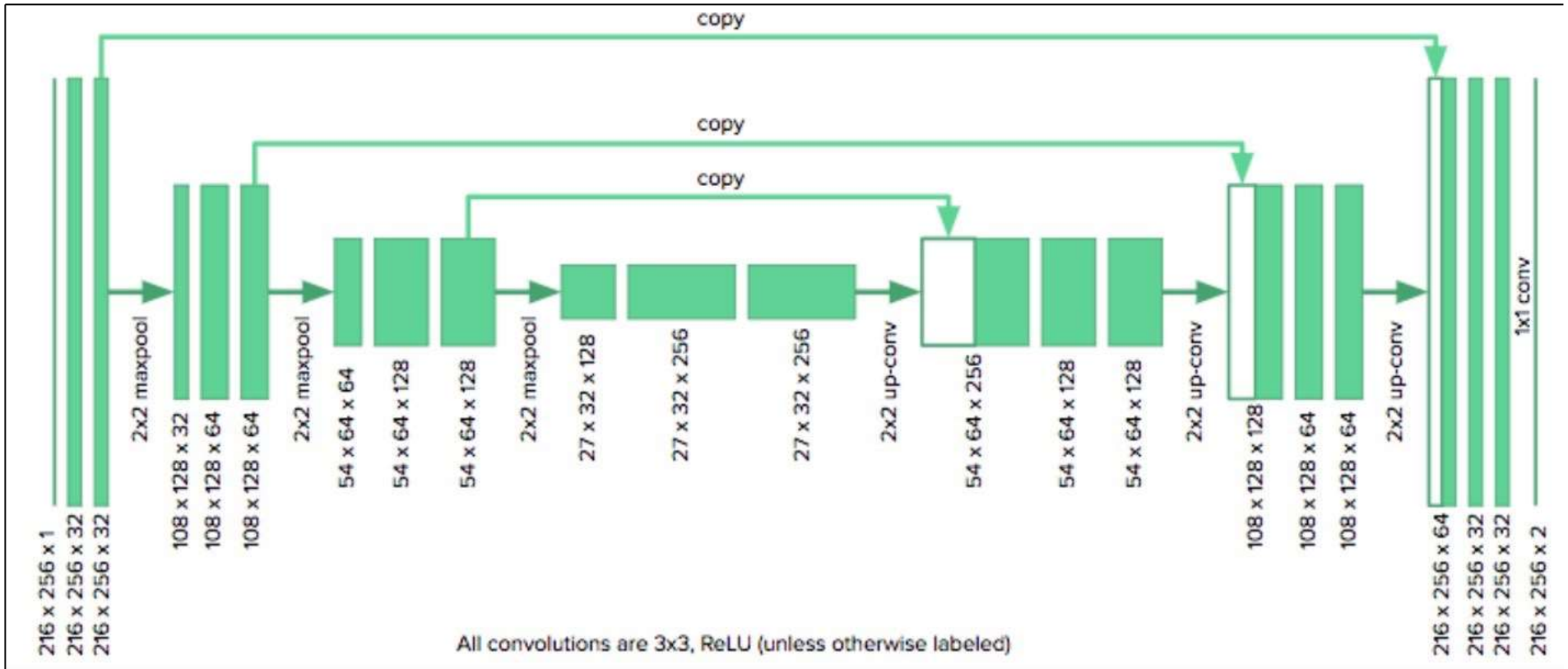


Metric is (twice) the ratio of the intersection over the sum of areas.

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.

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.

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

Acknowledgements

1. Goodfellow, I., Bengio, Y., and Courville, A., Deep Learning, MIT Press 2016
2. Yee, C-H., “Heart Disease Diagnosis with Deep Learning: State-of-the-art results with 60x fewer parameters”
<https://blog.insightdatascience.com/heart-disease-diagnosis-with-deep-learning-c2d92c27e730>

Noise Robustness

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

Topics in Noise Robustness

- Noise applied at inputs
- Noise applied to weights
- Injecting noise at the output targets

Noise injection is powerful

- Noise applied to inputs is a data augmentation
 - For some models addition of noise with infinitesimal variance at the input is equivalent to imposing a penalty on the norm of the weights, e.g., $\lambda \mathbf{w}^T \mathbf{w}$
- Noise applied to hidden units
 - Noise injection can be much more powerful than simply shrinking the parameters
 - Noise applied to hidden units is so important that it merits its own separate discussion
 - Dropout is the main development of this approach

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 learning considers model weights to be uncertain and representable via a probability distribution $p(\mathbf{w})$ that reflects that uncertainty,
 - Adding noise to weights is a practical, stochastic way to reflect this uncertainty

Noise at weights encourages stability

- Noise applied to weights is equivalent to traditional regularization, encouraging stability
- This can be seen in a regression setting
 - Train $\hat{y}(\mathbf{x})$ to map \mathbf{x} to a scalar using least squares between model prediction $\hat{y}(\mathbf{x})$ and true values y :

$$J = E_{p(\mathbf{x}, y)} \left[\left(\hat{y}(\mathbf{x}) - y \right)^2 \right]$$

- Training set: m labeled samples $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$
 - We perturb each input with $\varepsilon_W \sim N(\varepsilon; 0, \eta I)$
 - For small η , this is equivalent to a regularization

term $E_{p(\mathbf{x}, y)} \left[\left\| \frac{\partial \hat{y}(\mathbf{x})}{\partial \mathbf{w}} \right\|^2 \right]$

- It encourages parameters to regions where small perturbations of weights have small influence on output

Injecting Noise at Output

- **Targets** Most datasets have some mistakes in y labels
 - Harmful to maximize $\log p(y|\mathbf{x})$ when y is a mistake
- To prevent it we explicitly model noise on labels
 - Ex: we assume training set label y is correct with probability $1-\epsilon$, and otherwise any of the other labels may be correct
 - This can be incorporated into the cost function
 - Ex: *Local Smoothing* regularizes a model based on a softmax with k output values by replacing the hard 0 and 1 classification targets with targets of $\epsilon/(k-1)$ and $1-\epsilon$ respectively

Bagging and Other Ensemble 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

What is bagging?

- It is short for *Bootstrap Aggregating*
- It is a technique for reducing generalization error by combining several models
 - Idea is to train several models separately, then have all the models vote on the output for test examples
- This strategy is called *model averaging*
- Techniques employing this strategy are known as *ensemble methods*
- Model averaging works because different models will not make the same mistake

Ex: Ensemble error rate

- Consider set of k regression models
 - Each model makes error ε_i on each example, $i=1, \dots, N$
 - Errors drawn from a zero-mean multivariate with variance $E[\varepsilon_i^2]=v$ and covariance $E[\varepsilon_i \varepsilon_j]=c$
 - Error of average prediction of all ensemble models:
- Expected squared error of ensemble prediction is

$$\frac{1}{k} \sum_i \varepsilon_i$$

$$E \left[\left(\frac{1}{k} \sum_i \varepsilon_i \right)^2 \right] = \frac{1}{k^2} E \left[\sum_i \varepsilon_i^2 + 2 \sum_{i < j} \varepsilon_i \varepsilon_j \right] = \frac{1}{k^2} \left(k v + 2 \sum_{i < j} c \right) = \frac{1}{k} v + \frac{2}{k(k-1)} c$$

- If errors are perfectly correlated, $c=v$, and mean squared error reduces to v , so model averaging does not help
- If errors are perfectly uncorrelated and $c=0$, expected squared error of ensemble is only v/k

Ensemble vs Bagging

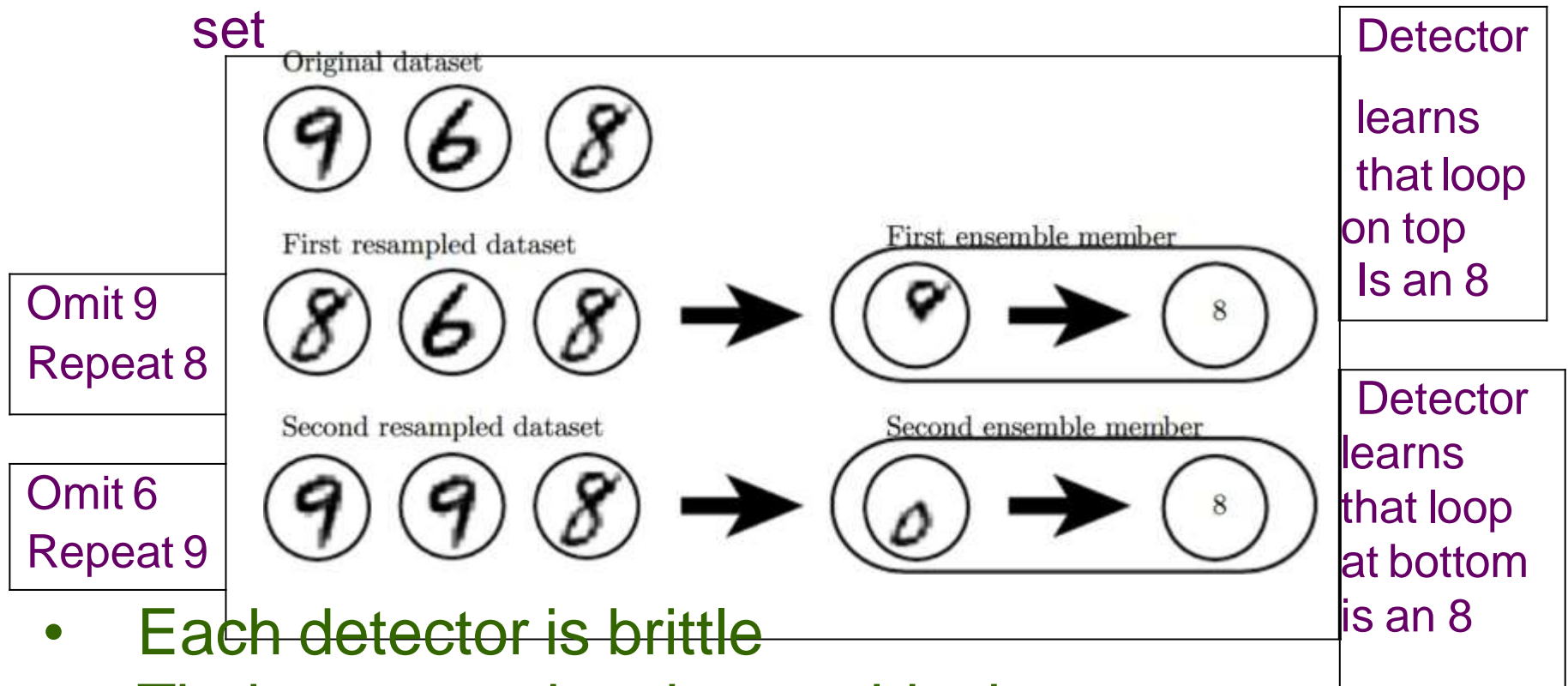
- Different ensemble methods construct the ensemble of models in different ways
 - Ex: each member of ensemble could be formed by training a completely different kind of model using a different algorithm or objective function
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times

The Bagging Technique

- Given training set D of size N , generate k data sets of same no of examples as original by sampling with replacement
 - Some observations may be repeated in each D_i the rest being duplicates. This is known as a bootstrap sample
 - The differences in examples will result in differences between trained models
 - The k models are combined by averaging the output (for regression) or voting (for classification)
- An example is given next

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 confidence when both loops are present

Neural nets and bagging

- Neural nets reach a wide variety of solution points
 - Thus they benefit from model averaging when trained on the same dataset
 - Differences in:
 - random initializations
 - random selection of minibatches, in hyperparameters,
 - cause different members of the ensemble to make partially independent errors

Model averaging is powerful

- Model averaging is a reliable method for reducing generalization error
 - Machine learning contests are usually won by model averaging over dozens of models
 - Ex: Netflix grand prize
- Since model averaging performance comes at the expense of increased computation and memory, benchmark comparisons are made using a single model

Boosting

- Incrementally adding models to the ensemble
 - After a weak learner is added, the data are reweighted:
 - examples that are misclassified gain weight and examples that are classified correctly lose weight
- Has been applied to ensembles of neural networks, by incrementally adding neural networks to the ensemble
- Also interpreting a neural network as an ensemble, incrementally adding hidden units to the network

Adaboost (informal)

- Consider 2-class problem
 - Data set $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$
 - Binary target variables $t_1, \dots, t_N, t_n \in \{-1, 1\}$
 - Each data point has weight w_n , initially set to $1/N$
 - We have a procedure available for training a base classifier using weighted data to give $y(\mathbf{x}) \in \{-1, 1\}$
- At each stage, train a new classifier
 - using a data set in which the weights are adjusted giving higher weight to those misclassified
- When desired no of base classifiers are trained they are combined to form a committee
 - With different weights to base classifiers

Adaboost Algorithm

1. Initialize $w^{(1)}_n = 1/N$ for $n=1, \dots, N$

2. For $m=1, \dots, M$

a) Fit a classifier $y_m(\mathbf{x})$ by minimizing weighted error

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(x_n) \neq t_n)$$

where $I(y_m(x_n) \neq t_n)$ is the indicator function and equals 1 when $I(y_m(x_n) \neq t_n)$ and 0 otherwise

b) Evaluate the quantities $\epsilon_m = \frac{J_m}{\sum_{n=1}^N w_n^{(m)}}$

and then use these to evaluate $\alpha_m = \ln \left\{ \frac{1-\epsilon_m}{\epsilon_m} \right\}$

c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(x_n) \neq t_n)\}$$

3. Make predictions using final model

$$Y_M(x) = \text{sign}(\sum_{m=1}^M \alpha_m y_m(x))$$

ϵ_m : weighted error rate of classifier

α_m : weighting coeffs give greater weight to more accurate classifiers

$w_n^{(m+1)}$: increase weight of misclassified data with exponential error