JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

EN.601.482/682 Deep Learning

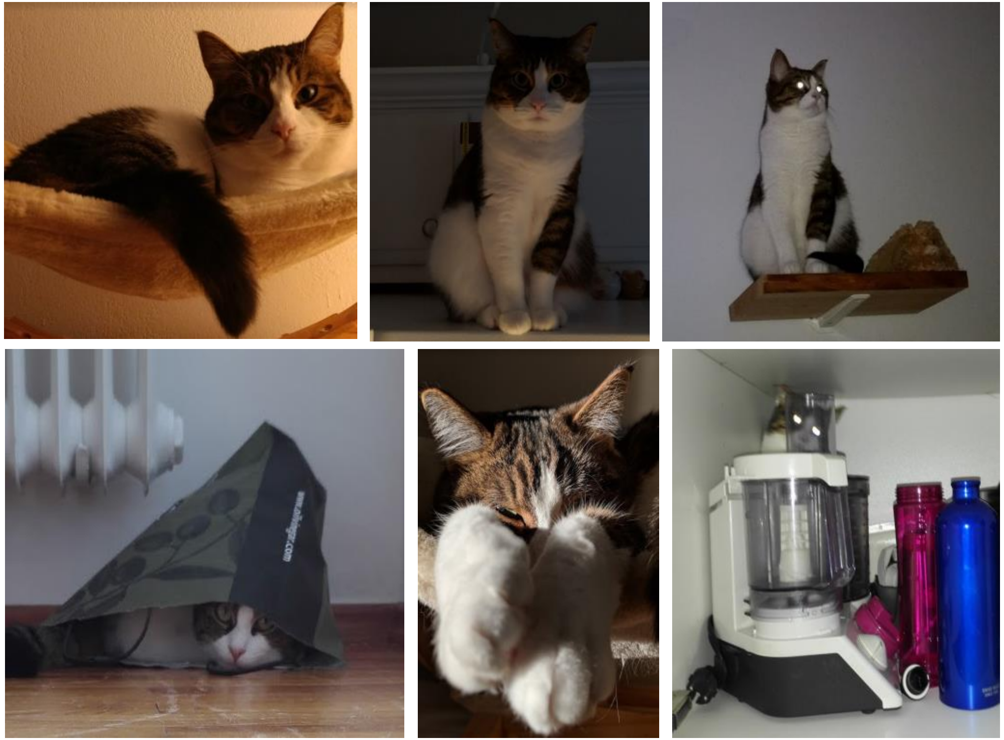
Basics Part II:
Regularization and Optimization

Mathias Unberath, PhD
Assistant Professor
Dept of Computer Science
Johns Hopkins University

Reminder

- Sign up on **Piazza** (access code dlF23)
- **Homework assignment 1 is due Wednesday**
- **Homework assignment 2 will be released Wednesday** (2 weeks)
- I will be traveling MW 10/12→ Friday session lectures or TA lectures?

Reminder



- Challenges for image recognition: Viewpoint, lighting, deformation, occlusion, background, …

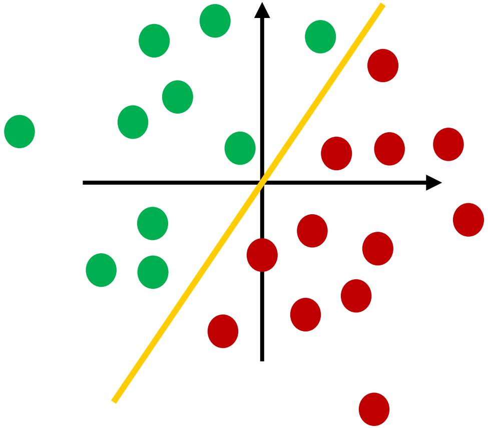
- Linear model yields scores:

$$s = f(x_i, W) = Wx_i + b$$

- Rows of W: “Templates”
  High response if input matches template
  → Strong assumption on image features

Reminder

- Extracting image features can yield a higher-level representation of an image



- Quantify unhappiness with current model parameters:
  - SVM loss
  - Softmax function and MLE, e.g. Kullback-Leibler
- Two questions for today:
  - Are good parameters unique?
  - How to get parameters that make us happy?

Today’s Lecture

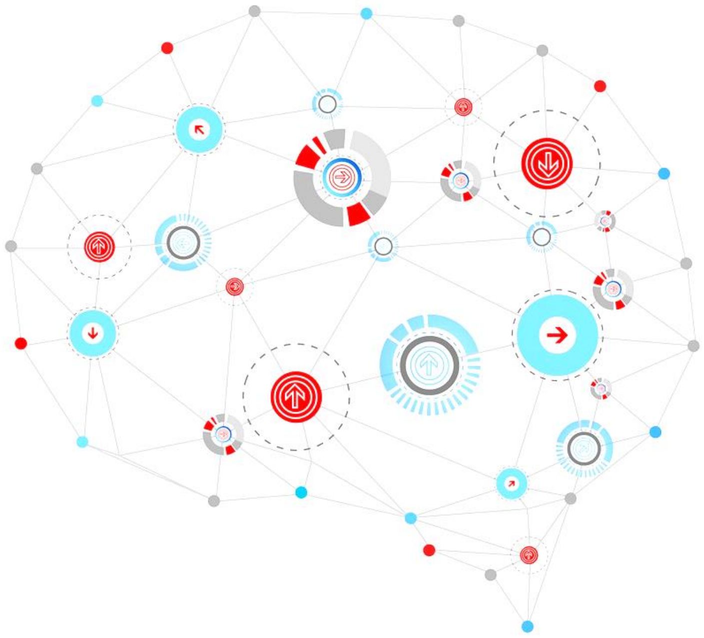
Regularization

- Bias-Variance-Tradeoff
- Common Regularizers

Optimization

- Search
- Gradient Descent
- Convergence

Regularization in Medical Problems

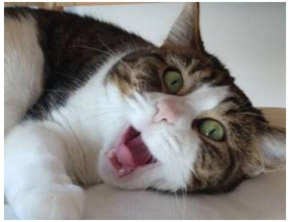


Regularization and Optimization

Regularization

Reminder: The SVM Loss

- 3 training examples and 3 classes: cat, car, bird
- W has been determined, the scores are:



Cat	**3.2**	1.3	2.2
Car	5.1	**4.9**	2.5
Bird	-1.7	2.0	**-3.1**
Loss	**2.9**	**0**	**12.9**

The Support Vector Machine (SVM) loss:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

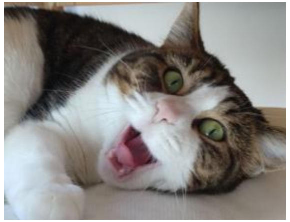
Overall loss:

$$L = \frac{1}{N} \sum_i L_i \left(f(x_i, W), y_i\right)$$

L = 1/3 * (2.9 + 0 + 12.9) = **5.27**

Reminder: The SVM Loss

- 3 training examples and 3 classes: cat, car, bird
- W has been determined, the scores are:



Cat	**3.2**		
Car		**4.9**	
Bird			**3.1**
Loss	**0**	**0**	**0**

The Support Vector Machine (SVM) loss:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Overall loss:

$$L = \frac{1}{N} \sum_i L_i \left(f(x_i, W), y_i \right)$$

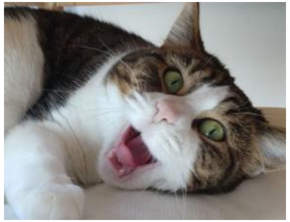
L = 1/3 * (0 + 0 + 0) = 0

We have found W such that the L = 0.

Is the W unique?

Reminder: The SVM Loss

- 3 training examples and 3 classes: cat, car, bird
- W has been determined, the scores are:



Cat	**3.2**		
Car		**4.9**	
Bird			**3.1**
Loss	**0**	**0**	**0**

The Support Vector Machine (SVM) loss:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Overall loss:

$$L = \frac{1}{N} \sum_i L_i \left(f(x_i, W), y_i\right)$$

L = 1/3 * (0 + 0 + 0) = 0

We have found W such that the L = 0.

Is the W unique?

→ No, 2W also has L = 0. How to chose?

Regularization: Expressing Preference

$$L(W) = \underbrace{\frac{1}{N} \sum_i L_i \left(f(x_i, W), y_i\right)}$$

Data term: Predictions must match annotations

Regularization: Expressing Preference

$$L(W) = \underbrace{\frac{1}{N} \sum_i L_i \left(f(x_i, W), y_i\right)}_{} + \underbrace{\lambda R(W)}_{}$$

Data term: Predictions must match annotations

Regularization
λ strength (hyperparameter)

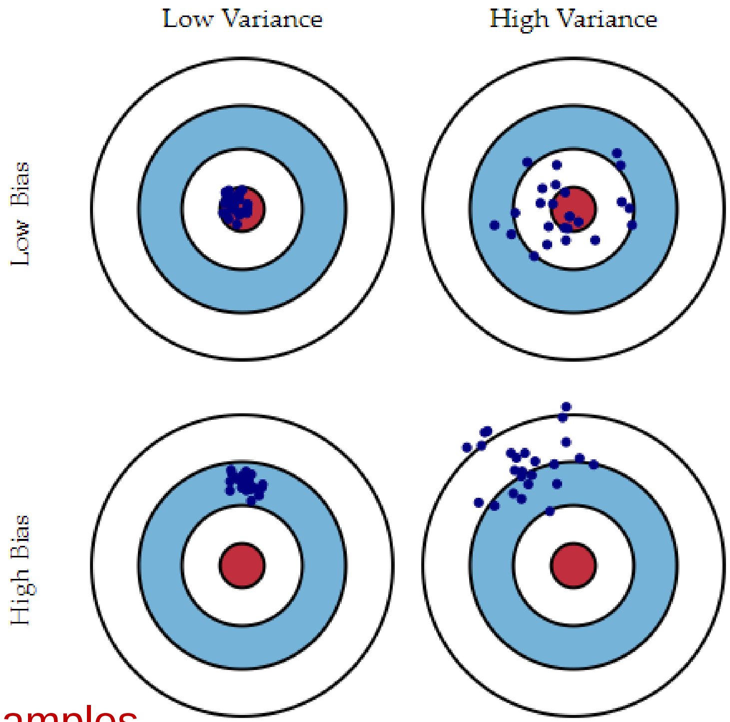
Avoiding Overfitting: The Bias Variance Tradeoff

- Estimator $f(x_i, W) = \hat{y}$

- Mean squared error
$$L(W) = E[(\hat{y} - y)^2]$$

- Decomposition into bias and variance
$$L(W) = \underbrace{(E[\hat{y}] - y)^2}_{\text{Bias}^2} + \underbrace{E[(\hat{y} - E[\hat{y}])^2]}_{\text{Variance}} + \sigma$$

Irreducible error

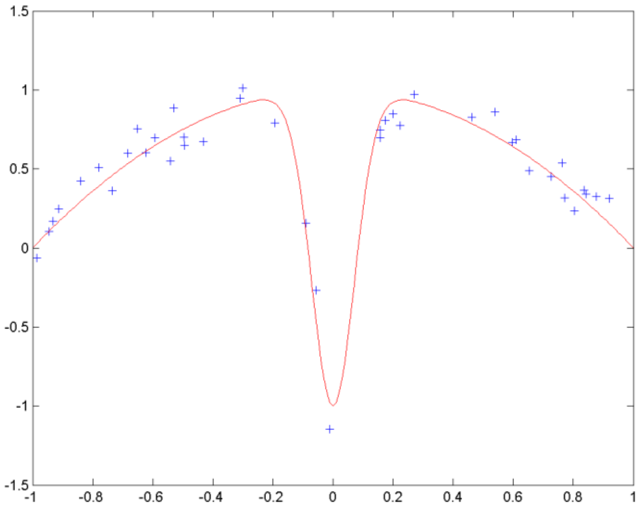


Expectations are computed over the currently observed samples.

Hastie, T., Tibshirani, R, and Friedman, J. (2017) The Elements of Statistical Learning

Why is this important? Example

- Every **blue curve** represents radial basis functions **fitted to a set of points** determined by the **red curve plus random noise**.

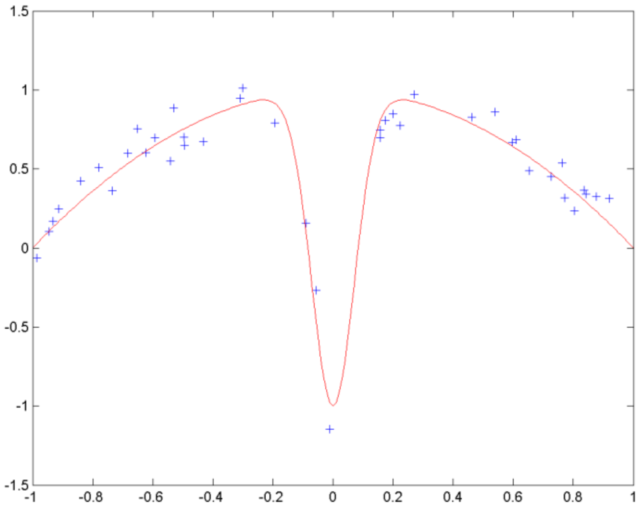


Function and noisy samples

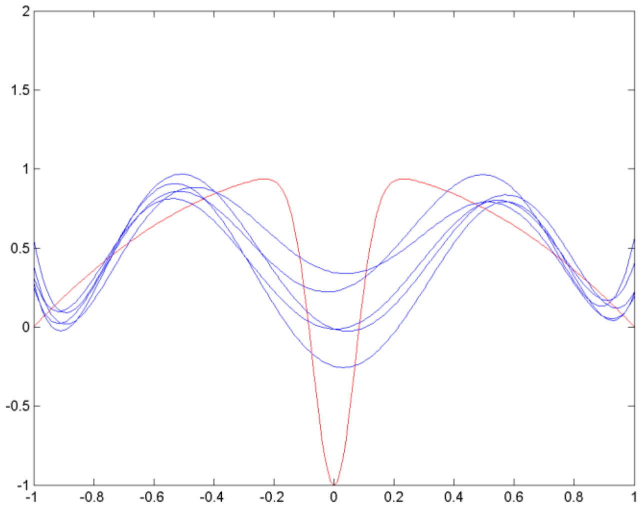
Images from Wikimedia commons, created by Anders Sandberg.

Why is this important? Example

- Every **blue curve** represents radial basis functions **fitted to a set of points** determined by the **red curve plus random noise**.



Function and noisy samples

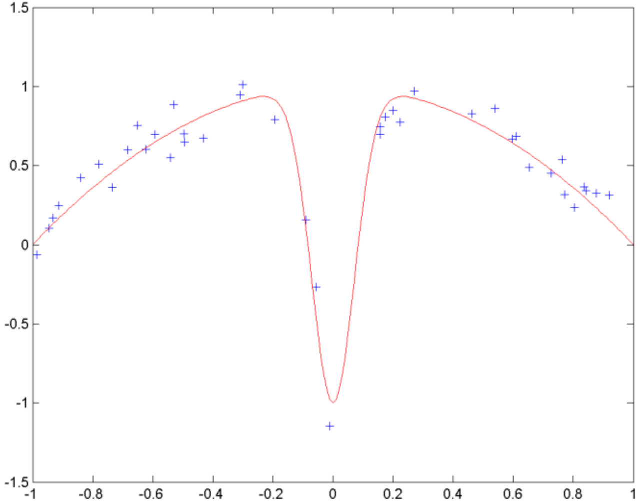


RBF spread: 5
→ High bias

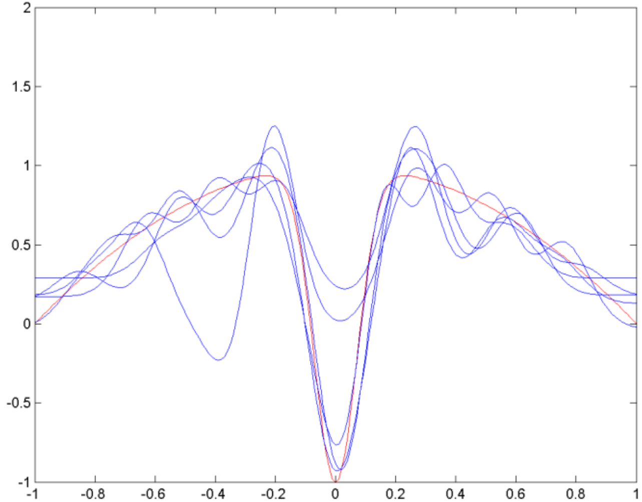
Images from Wikimedia commons, created by Anders Sandberg.

Why is this important? Example

- Every **blue curve** represents radial basis functions **fitted to a set of points** determined by the **red curve plus random noise**.



Function and noisy samples



RBF spread: 0.1
→ High variance

Images from Wikimedia commons, created by Anders Sandberg.

Regularization: Expressing Preference

$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)}_{\text{\textbf{Data term}: Predictions must match annotations}} + \lambda R(W)$$

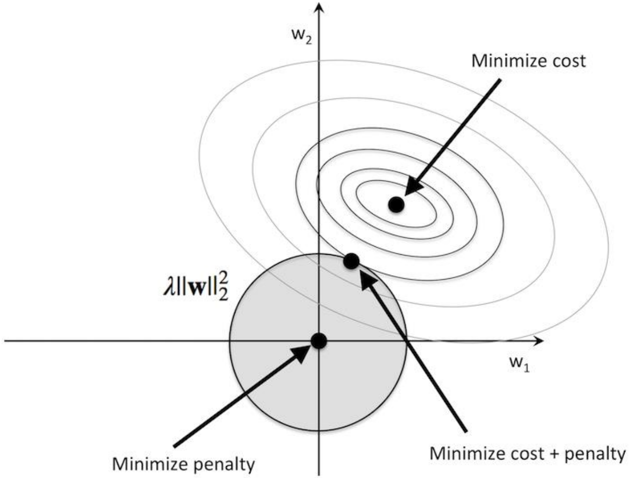
→ Loss function L(W) must accurately reflect our desired behavior of f(x,W)!

- Data loss
  - This is straight forward
- Regularization
  - Introduce preferences on weights; e.g. sparse or of small magnitude
  - Counteract overfitting by enforcing simpler models
  - Aid optimization (see later) by shaping the loss function (adding curvature)

Regularization: Examples

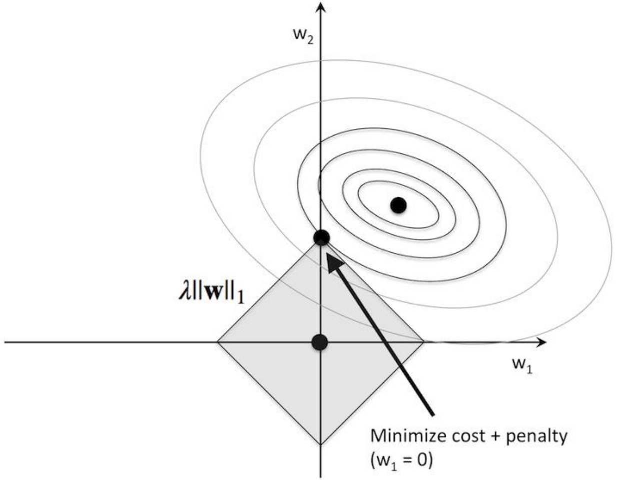
$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)}_{} + \lambda R(W)$$

Data term: Predictions must match annotations



Simple regularizers

- L2 (magnitude): $R(W) = \sum_{k,l} W_{k,l}^2$
- L1 (sparsity): $R(W) = \sum_{k,l} |W_{k,l}|$
- Versions, e.g. Elastic net, Huber,…



Images from rasbt.github.io.

Regularization: Examples

$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)} + \lambda R(W)$$

Data term: Predictions must match annotations

More complex regularizers

- Dropout [1]
- Batch normalization [2]
- Stochastic depth [3], … and many others

→ We will later learn about some

[1] Srivastava, N., et al (2014). Dropout: a simple way to prevent neural networks from overfitting. JMLR
[2] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv
[3] Huang, G., et al. (2016) Deep networks with stochastic depth. ECCV

Reminder

- Solutions are not unique
  → But we can incorporate prior knowledge or preference

- Bias variance tradeoff
  → Regularization to avoid overfitting

- Data terms
  - SVM loss
  - Softmax function and Kullback-Leibler divergence or cross-entropy

- Regularization
  - L-norm
  - Batchnorm, Dropout, …

Regularization and Optimization

Optimization

Optimization

$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)}_{\textbf{Data fidelity}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

A target loss function in its “full” glory, defining our preferred solution.

How do we actually retrieve this solution?

Optimization

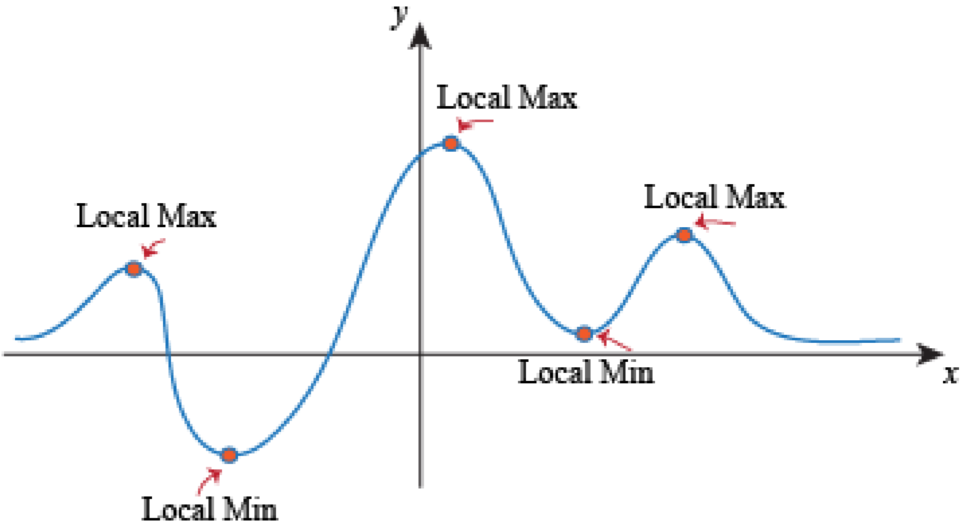
$$L(W) = \underbrace{\frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right)}_{\textbf{Data fidelity}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

The analytic approach:

- Express derivative
- Set to zero, and find solutions

→ Not possible in most cases!

→ Many questions!



Optimization

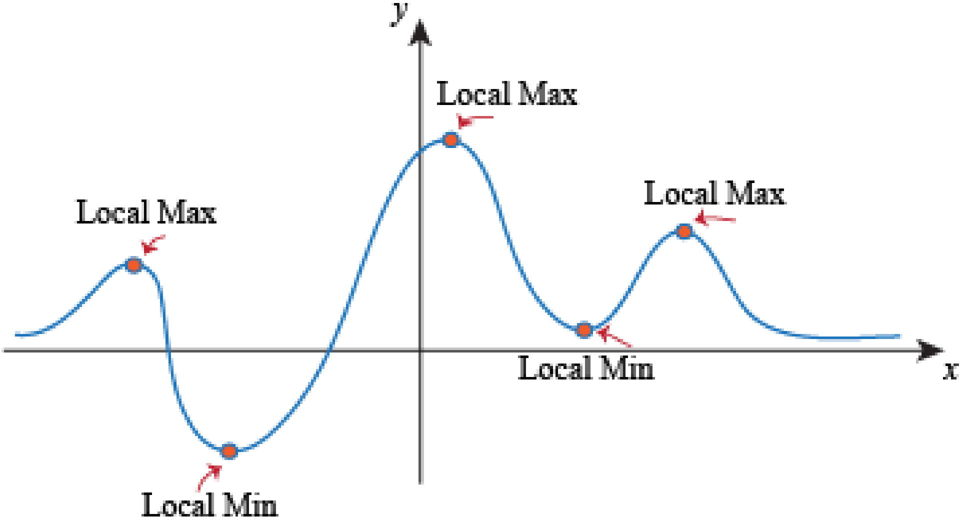
What exactly is f()?

What exactly is R()?

$$L(W) = \underbrace{\frac{1}{N} \sum_i L_i \left(f(x_i, W), y_i\right)}_{\text{Data fidelity}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

What are these pairs (xi, yi)?
Are they representative?

What is the loss?



An Analogy

Finding the lowest point: $W' = \arg\min_W L(W)$

Idea 1: Brute Force Search

The idea: List and evaluate all possible candidates for W

Guaranteed to find the global minimum, but it becomes prohibitively computationally expensive very fast.

→ For most problems, this is a fairly bad idea.

Idea 1*: Random Search

The idea: Randomly sample and evaluate several candidates for W

Less computationally expensive, but no more guarantee to find global optimum. Also, how many random samples are enough?

Stanford example for CIFAR 10:

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15.5% accuracy! not bad!
(SOTA is ~95%)

Idea 1*: Random Search

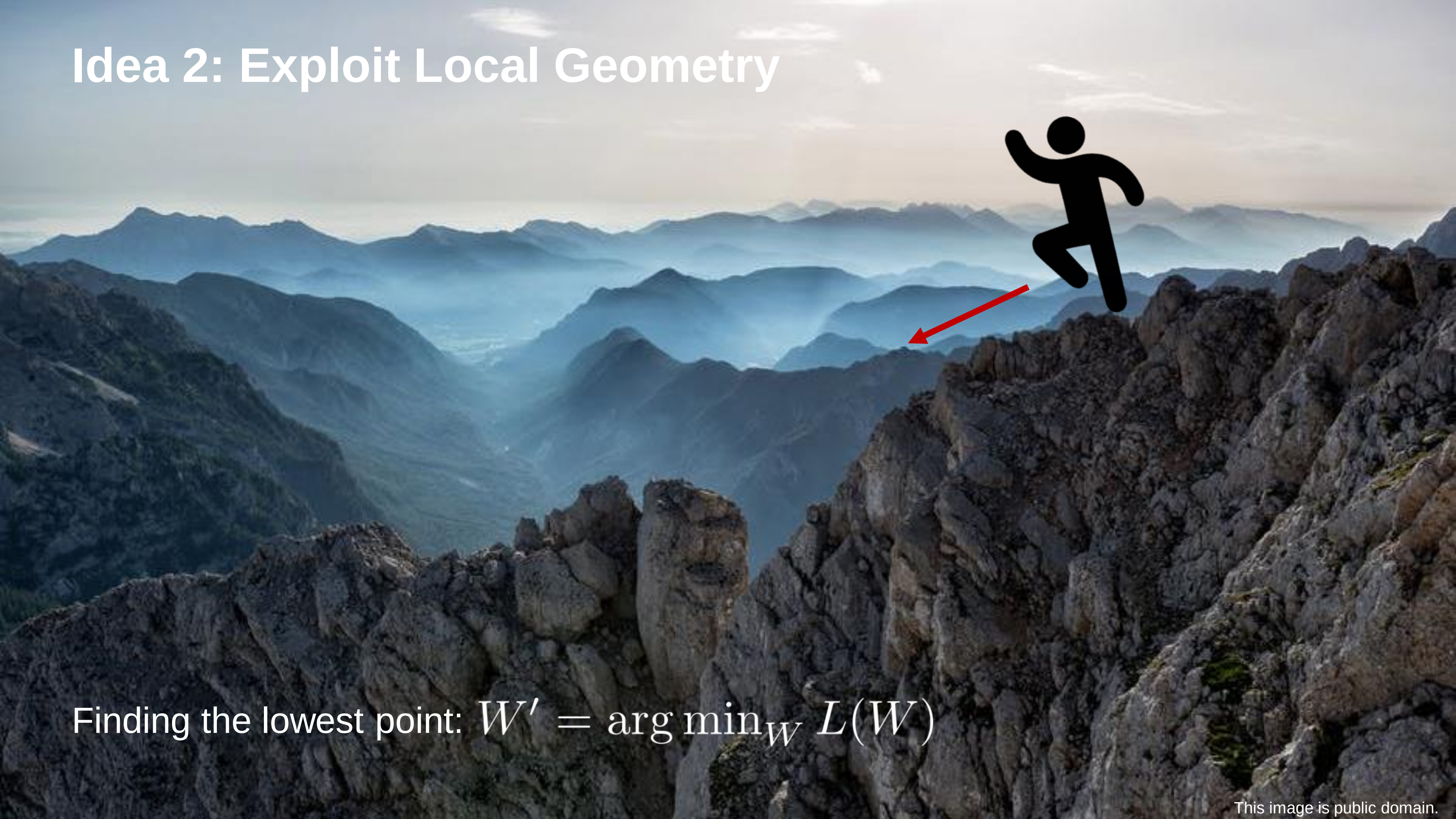
The idea: Randomly sample and evaluate several candidates for W

Less computationally expensive, but no more guarantee to find global optimum.
Also, how many random samples are enough?

→ Probably, an even worse idea.

To put things in perspective: ResNet-152 has ~60 Mio. parameters

Idea 2: Exploit Local Geometry



Finding the lowest point: $W' = \arg\min_W L(W)$

This image is public domain.

Idea 2: Exploit Local Geometry

The idea: At current position, find steepest slope and follow for a bit

Depending on L(W), there can be guarantees on finding global optimum. This is an iterative strategy.

→ Sounds simple, but works very well in practice.

Following the Slope: Gradients

The derivative of a 1D function: $f(x) : x \in \mathbb{R} \mapsto \mathbb{R}$

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

This concept extends to multi-dimensional functions: $f(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n \mapsto \mathbb{R}$

We obtain a gradient vector with partial derivatives along each dimension.

$$(\nabla f(\mathbf{x}))_i = \frac{\partial f(\mathbf{x})}{\partial x_i}, \quad \nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

Direction of steepest descent at a current point $\mathbf{x}_0$ is the negative gradient $-\nabla f(\mathbf{x}_0)$

Numeric Gradient Computation

f(W)	Current W		W + h		Gradient dW
	0.34		0.34 **+ 0.0001**		3.0
	-1.11		-1.11		
	0.78	→	0.78	→	
	0.12		0.12		
	⋮		⋮		⋮
f(W)	4.35423		4.35453		

(4.35453 - 4.35423) / 0.0001= 3.0

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$$

Numeric Gradient Computation

	Current W		W + h		Gradient dW
	0.34		0.34		3.0
	-1.11		-1.11 **+ 0.0001**		-1.3
	0.78	→	0.78	→	
	0.12		0.12		
	⋮		⋮		⋮
f(W)	4.35423		4.35410		

(4.35410 - 4.35423) / 0.0001= -1.3

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$$

Numeric Gradient Computation

	Current W	W + h	Gradient dW
	0.34	0.34	3.0
	-1.11	-1.11 **+ 0.0001**	-1.3
	0.78	0.78	
	0.12	0.12	
	⋮	⋮	⋮
f(W)	4.35423	4.35410	

(4.35410 - 4.35423) / 0.0001= -1.3

Slow and approximate.

Numeric Gradient Computation

	Current W			Gradient dW
	0.34			3.0
	-1.11			-1.3
	0.78	→	**Calculus!** →	
	0.12			
	⋮			⋮
f(W)	4.35423			

For suitable functions, calculus provides the analytic gradient!

Gradient Descent

- Numerical gradient: Approximate and slow (remember ResNet-152?) but easy
- Analytic gradient: Exact and fast but error-prone

A gradient descent algorithm

```
while not_converged:
    gradient = eval_gradient(loss, data, weights)
    weights += - step_size * gradient
```

Gradient Descent

- Numerical gradient: Approximate and slow (remember ResNet-152?) but easy
- Analytic gradient: Exact and fast but error-prone

A gradient descent algorithm

```
while not_converged:
    gradient = eval_gradient(loss, data, weights)
    weights += - step_size * gradient
```

Q: Any problems with this part?

Gradient Descent

- Numerical gradient: Approximate and slow (remember ResNet-152?) but easy
- Analytic gradient: Exact and fast but error-prone

A gradient descent algorithm

while not_converged:

    gradient = eval_gradient(loss, data, weights)

    weights += - **step_size** * gradient

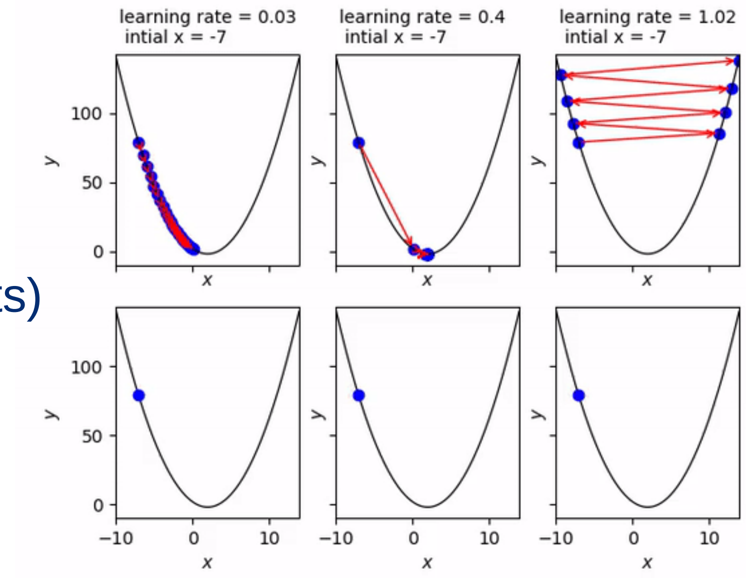


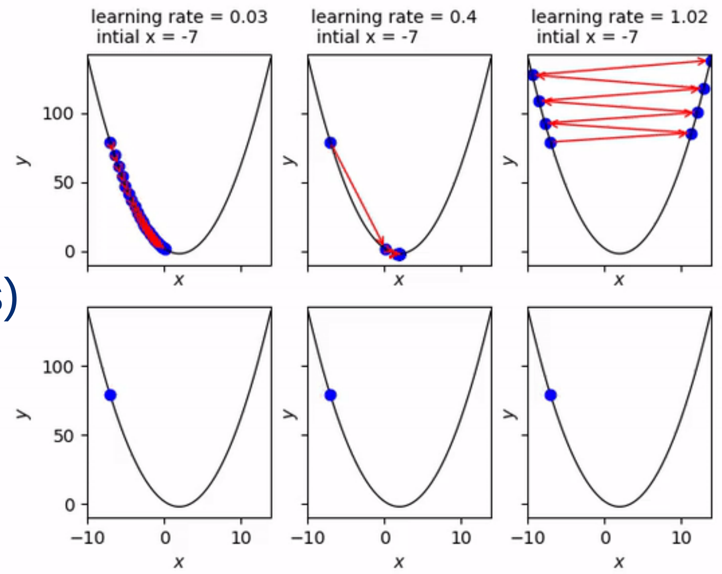
Image from jed-ai.github.io. Too large step sizes can cause divergence!

Gradient Descent

- Numerical gradient: Approximate and slow (remember ResNet-152?) but easy
- Analytic gradient: Exact and fast but error-prone

A gradient descent algorithm

while not_converged:
    gradient = eval_gradient(loss, **data**, weights)
    weights += - step_size * gradient



[Image from jed-ai.github.io.](https://jed-ai.github.io) Too large step sizes can cause divergence!

Q: Any problems with this part?

Stochastic Gradient Descent

- Number of samples can be large! $L(W) = \frac{1}{N}\sum_i L_i\left(f(x_i, W), y_i\right) + \lambda R(W)$
- Approximate sum over all samples by a sum over a much smaller **minibatch**

A stochastic gradient descent algorithm

```
while not_converged:
    data_batch = sample_training_data(data, batch_size)
    gradient = eval_gradient(loss, data_batch, weights)
    weights += - step_size * gradient
```

Online demo for gradient descent

Refresher: Jacobian

Vector-valued multi-dimensional functions: $\mathbf{f}(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n \mapsto \mathbb{R}^m$

$$J_{\mathbf{f}}(\mathbf{x}) \in \mathbb{R}^{m \times n}$$

$$J_{\mathbf{f}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla(\mathbf{f}(\mathbf{x}))_1 \\ \vdots \\ \nabla(\mathbf{f}(\mathbf{x}))_m \end{pmatrix}$$

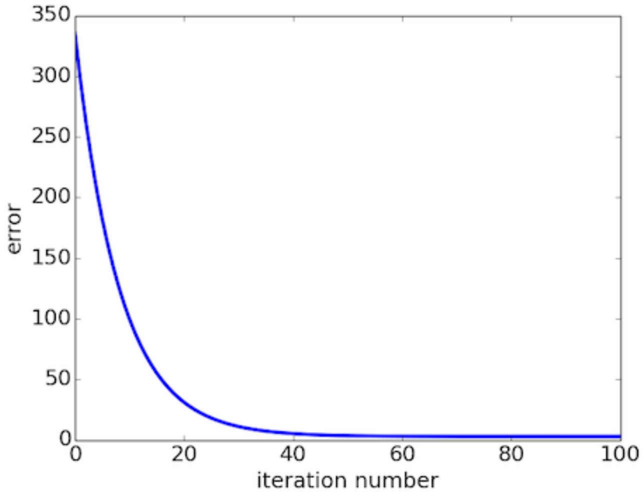
$$(J_{\mathbf{f}}(\mathbf{x}))_{i,j} = \frac{\partial f_i}{\partial x_j} \qquad J_{\mathbf{f}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Convergence

- Large annotated dataset and we want to find a good score function, e.g. for predicting the cat-i-ness of an input image

- Remember the **bias variance tradeoff**
  - First step: Define loss function and introduce regularization
  - Second step: Apply favorite gradient descent algorithm to find optimal W

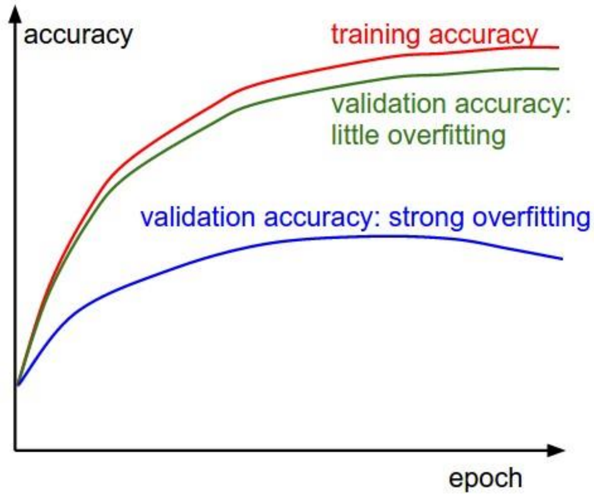
- Very nice convergence → Error is stable!



Q: Is this the error we are truly interested in?

Convergence

- Large annotated dataset and we want to find a good score function, e.g. for predicting the cat-i-ness of an input image
- Remember the **bias variance tradeoff**
  - First step: Define loss function and introduce regularization
  - Second step: Apply favorite gradient descent algorithm to find optimal W
- Very nice convergence → Error is stable!



A: Not in general, since we want to predict on instances not contained in our training set. → Early stopping.

Dataset Design

Approach 1: Use all data for training; test on, hum, the dataset?

Dataset

Dataset Design

Approach 1: Use all data for training; test on, hum, nothing? → Bad!

Dataset

Approach 2: Split data into train and test; hyperparameters chosen to be best on test data.

Train	Test

Dataset Design

Approach 1: Use all data for training; test on, hum, nothing? → Bad!

Dataset

Approach 2: Split data into train and test; hyperparameters chosen to be best on test data. → Bad, no way to know how this will generalize to new data.

Train	Test

Dataset Design

Approach 1: Use all data for training; test on, hum, nothing? → Bad!

Dataset

Approach 2: Split data into train and test; hyperparameters chosen to be best on test data. → Bad, no way to know how this will generalize to new data.

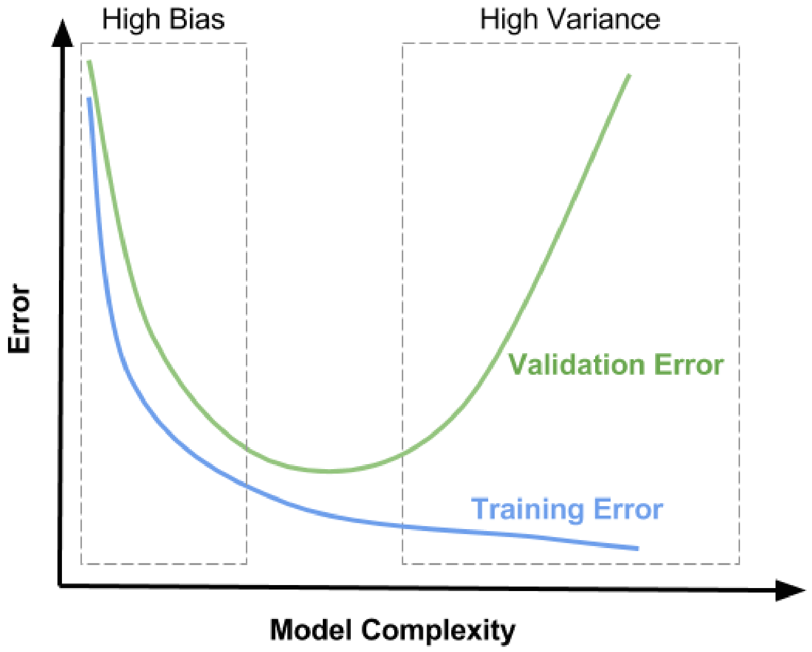
Train | Test

Approach 3: Split data into train, validation, and test; hyperparameters chosen on validation, then evaluated on test. → Better.

Train | Validation | Test

Dataset Design and Convergence

Remember the **bias variance tradeoff** when optimizing for parameters!

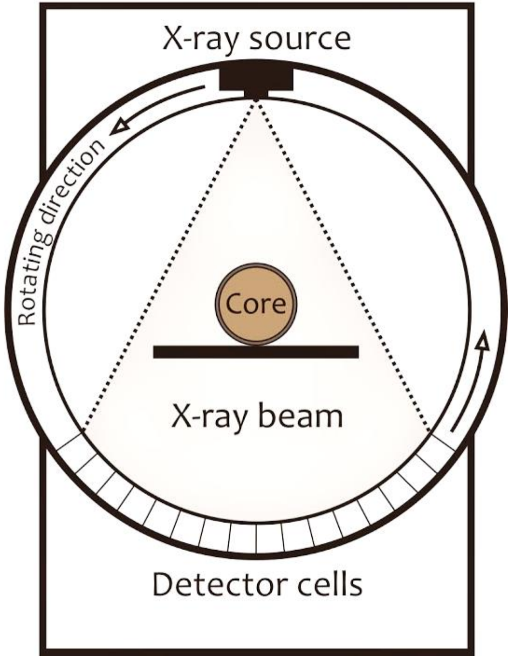


Regularization and Optimization

Regularization in Medical Problems

Total Variation Regularization in Computed Tomography

- X-ray source rotates around the object
- Acquires X-ray images on the trajectory
- If several conditions are met (e.g. Tuy's condition)
  → 3D reconstruction of slices is possible
  → Known as tomography
- X-radiation is ionizing (put concisely: bad for you)
  → Tradeoff: **Radiation dose vs. image quality**

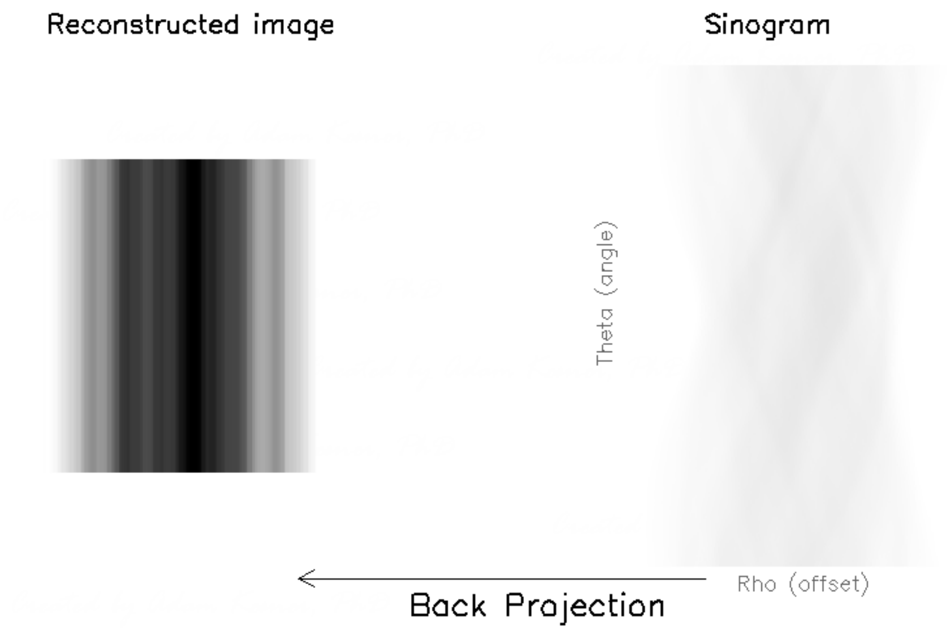


Further reading:

Sidky, E. Y., & Pan, X. (2008). Image reconstruction in circular cone-beam computed tomography by constrained, total-variation minimization. Physics in Medicine & Biology, 53(17), 4777.
Condat, L. (2014). A generic proximal algorithm for convex optimization—application to total variation minimization. IEEE Signal Processing Letters, 21(8), 985-989.

CT: Analytic Reconstruction

→ “One-shot” inversion of image formation model (filtered back-projection)



Very fast, but

- Inversion is approximate
- Assumes artifact free image formation

Further reading:

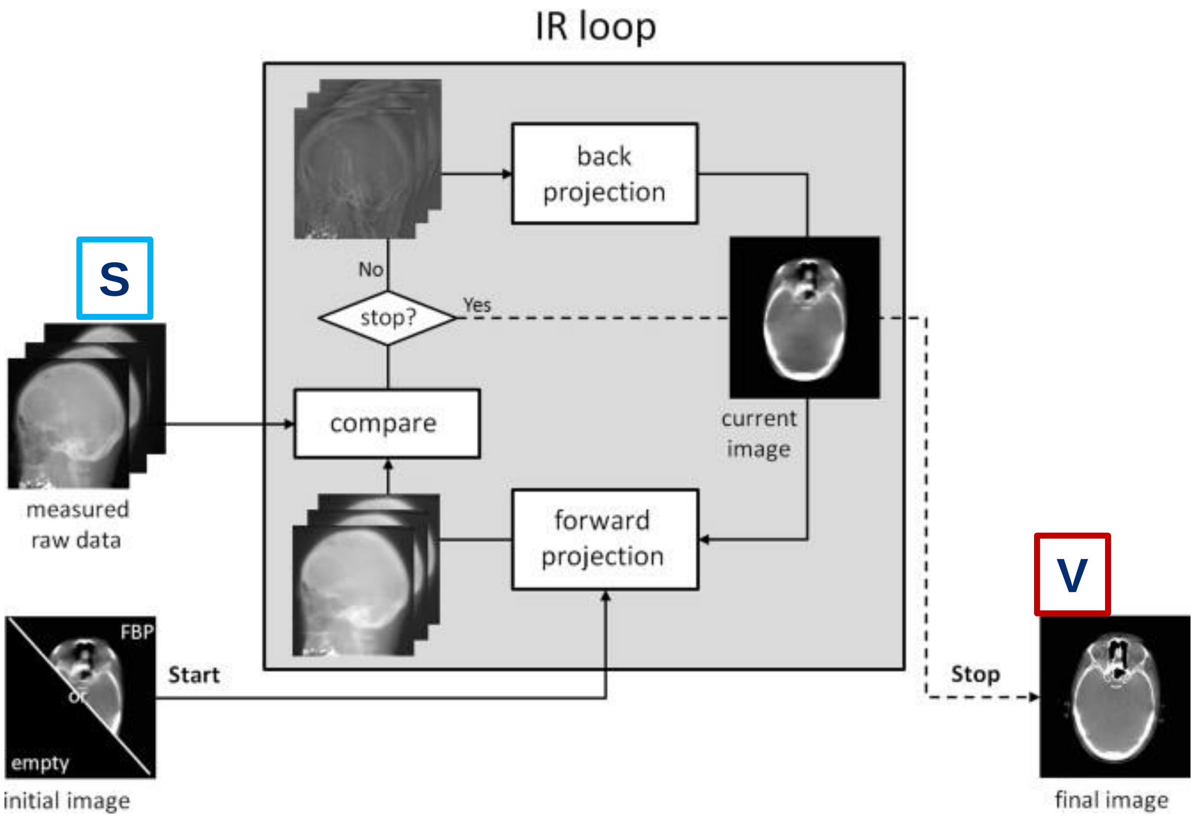
Sidky, E. Y., & Pan, X. (2008). Image reconstruction in circular cone-beam computed tomography by constrained, total-variation minimization. Physics in Medicine & Biology, 53(17), 4777.
Condat, L. (2014). A generic proximal algorithm for convex optimization—application to total variation minimization. IEEE Signal Processing Letters, 21(8), 985-989.

CT: Iterative Reconstruction

- Optimization rather than direct inversion!

$$\arg\min_V L_2(\mathbf{A}\boxed{V} - \boxed{S})$$

- **A**: System matrix
  encodes projection geom.

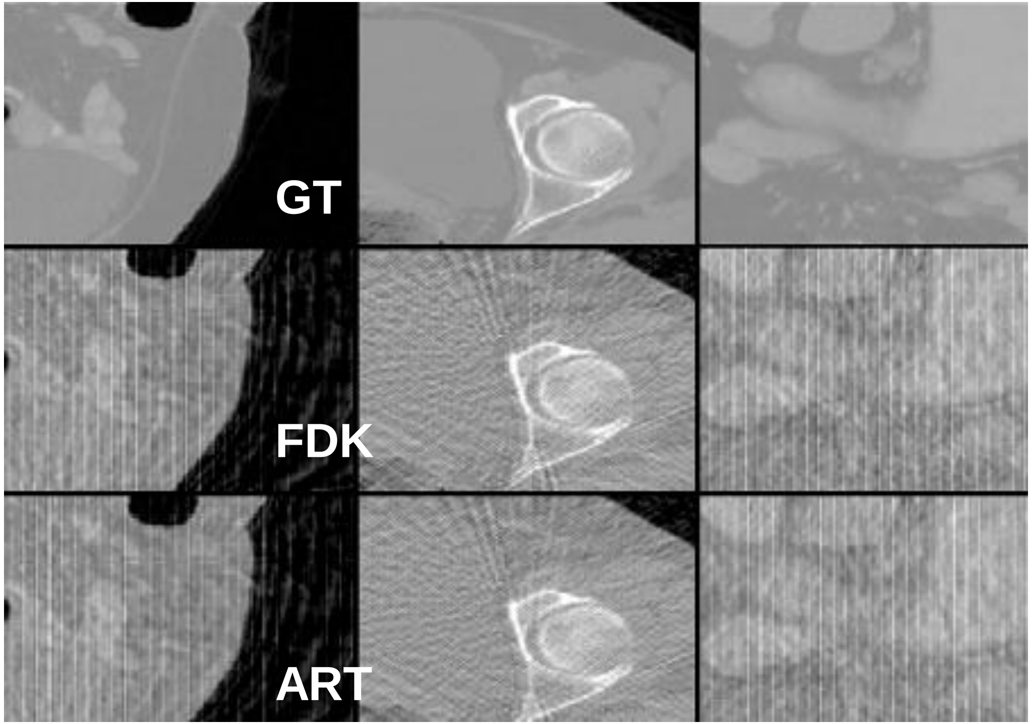


CT: Iterative Reconstruction

- Optimization rather than direct inversion!

$$\arg\min_V L_2(\mathbf{A}\boxed{V} - \boxed{S})$$

- **A**: System matrix encodes projection geom.



Optimization alone does not yet bring the desired benefit!

CT: Iterative Reconstruction

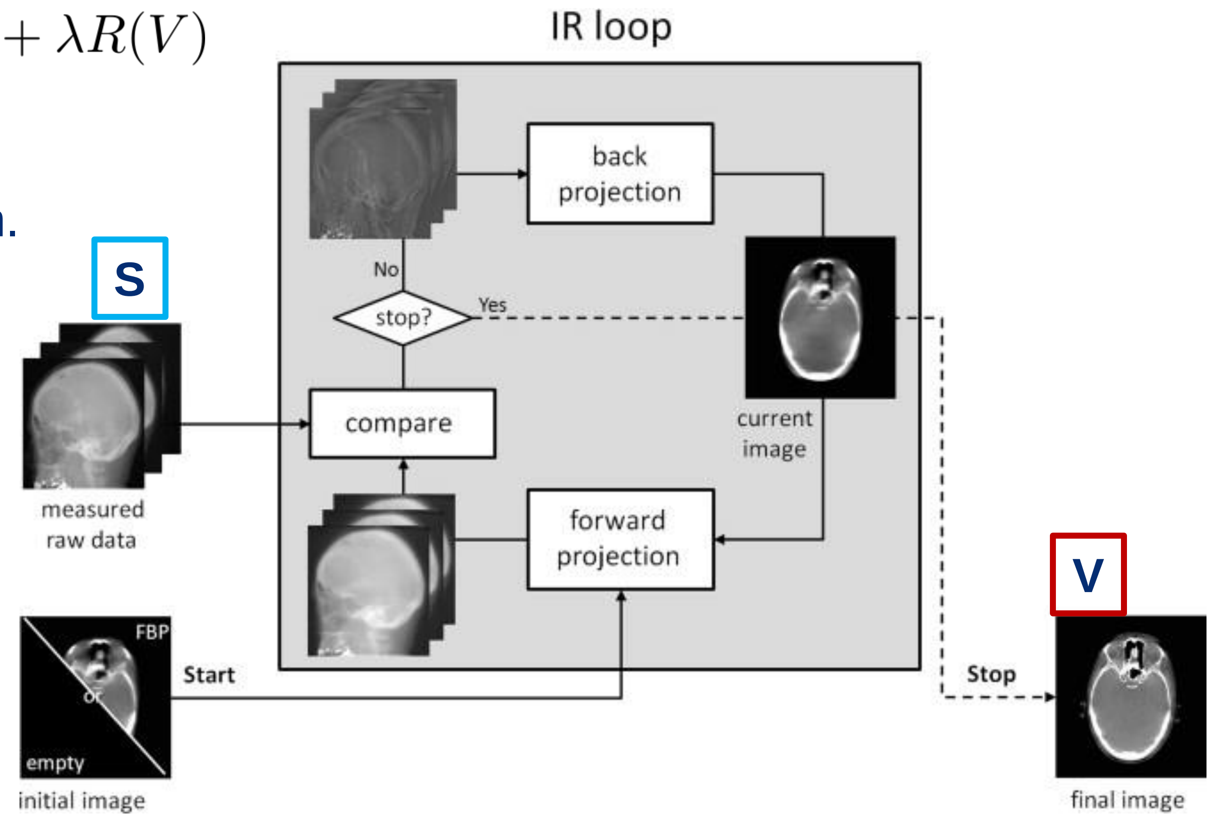
- Optimization rather than direct inversion!

$$\arg\min_V L_2(\mathbf{A}\boxed{V} - \boxed{S}) + \lambda R(V)$$

- **A**: System matrix
  encodes projection geom.

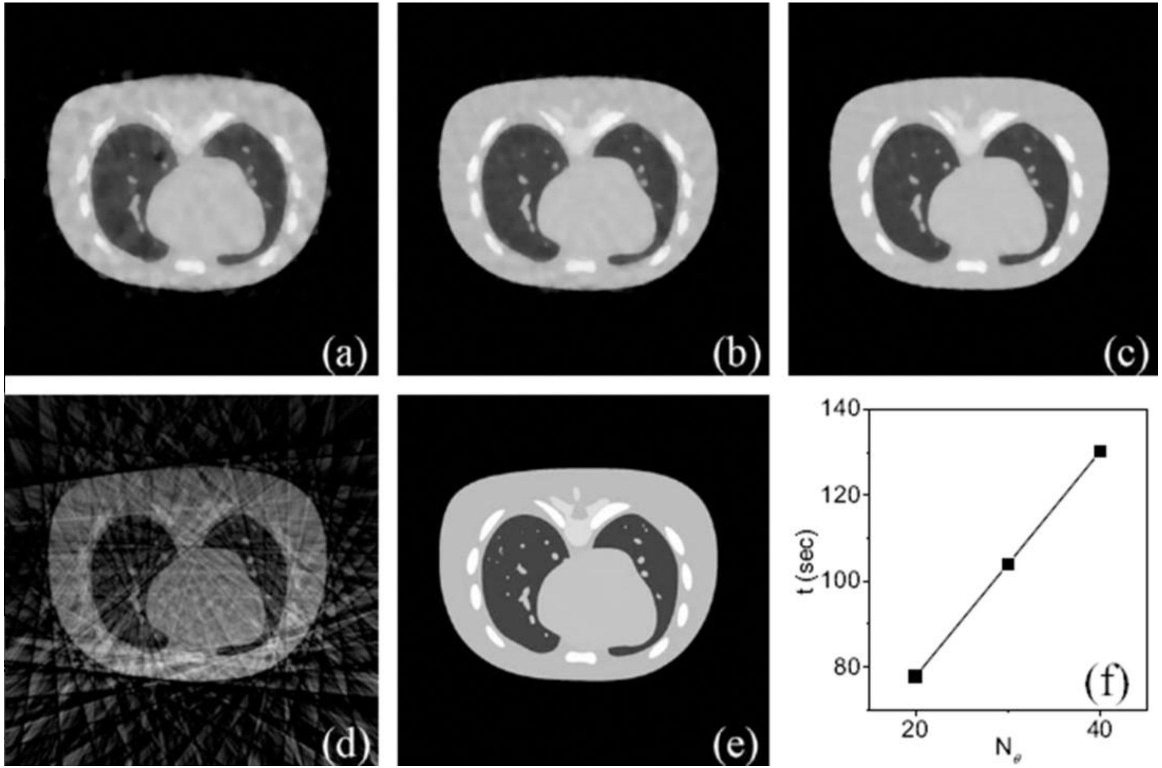
- Regularization:
  Total variation

$$R(V) = \sum_V L_1(\nabla V)$$



Examples of TV-Regularized Reconstructions

Reconstructions from (a) 20, (b) 30, (c) 40 views using ART-TV. (d) 40-view FDK



CT: Iterative Reconstruction

- Optimization rather than direct inversion!

$$\arg\min_V L_2(\mathbf{A}\boxed{V} - \boxed{S}) + \lambda R(V)$$

…but not differentiable

- **A**: System matrix
  encodes projection geom.

This optimization problem (and TV in general) is convex. However, optimization is **not straight-forward**!

→ Nested TV optimization

→ Primal-dual splitting (see Condat paper)

- Regularization:
  Total variation

$$R(V) = \sum_V L_1(\nabla V)$$

Regularization and Optimization

Questions?