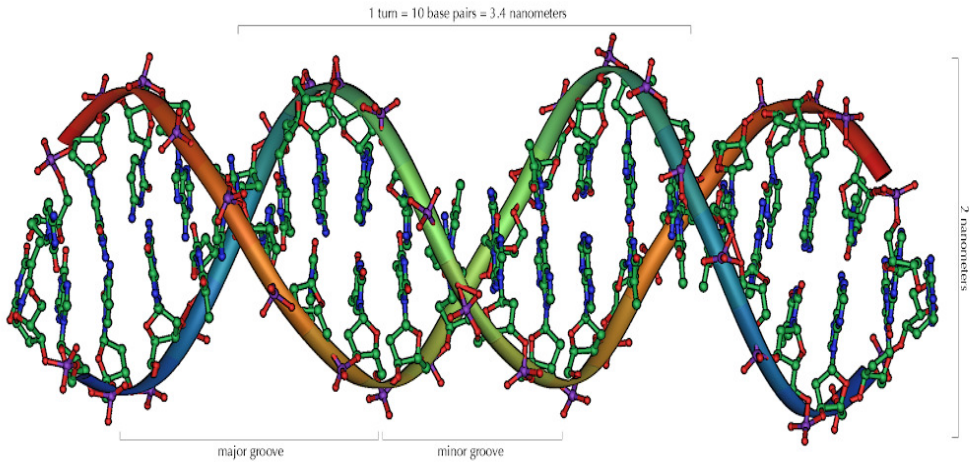
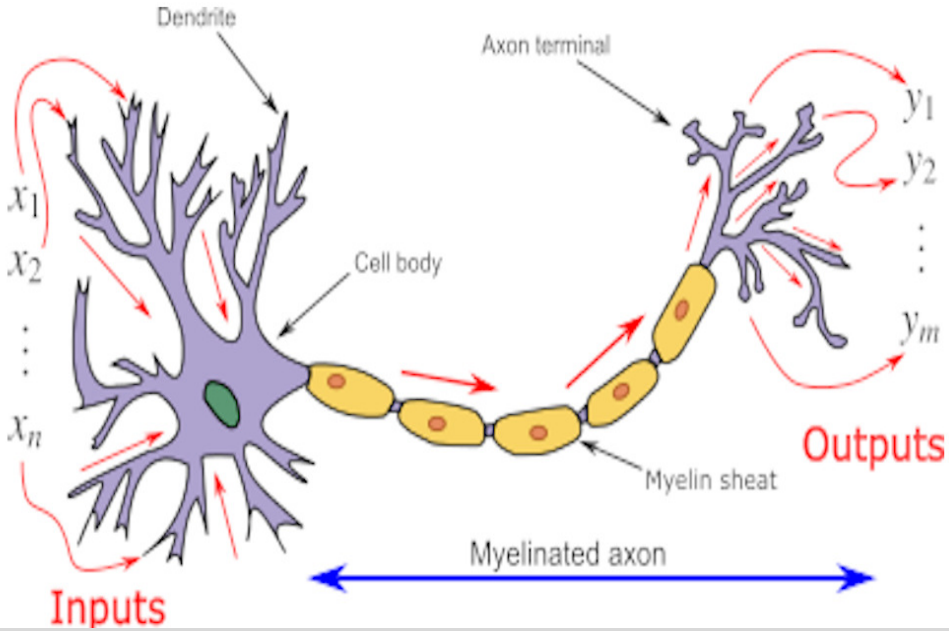
Karlsruher Institut für Technologie

Foundations of Neural Networks: Regularization

Andreas Geyer-Schulz | 22. April 2025

INFORMATION SERVICE AND ELECTRONIC MARKETS ANDREAS.GEYER-SCHULZ@KIT.EDU





Status:

V1.0

Feedback welome.

Learning Goals

You should

- know the difference between optimization and learning.
- know what overfitting and the bias-variance tradeoff are.
- know the phases of training, validating, and testing and the construction of proper training, validation, and test data sets or environments.
- know what regularization means.
- have an overview about regularization methods.
- know and be able to apply the various regularization methods.

Content

Optimization and Generalization

- The great challenge for machine learning algorithms is that they must perform well on new, previously unseen inputs and not only on those on which they were trained.

  The ability to perform well on previously unobserved inputs is called **generalization**.
- In a typical machine learning model, we have acces to a training data set: The machine learning model computes a **training error** and minimizes the training error.

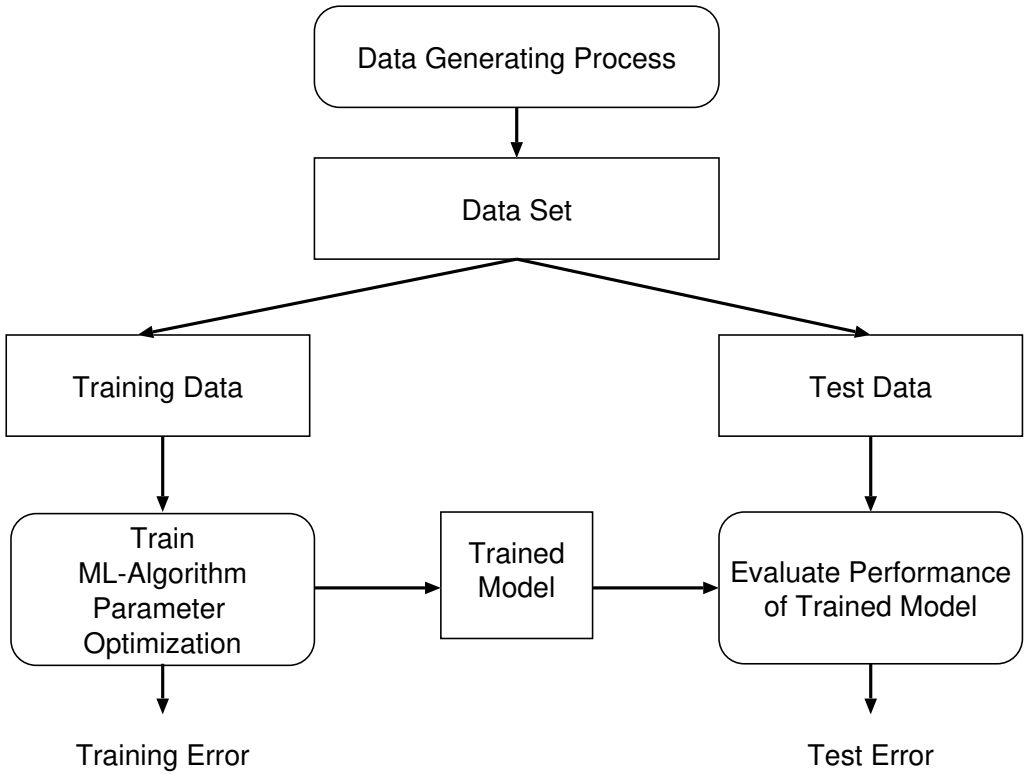
  This is an optimization problem.
- However, in addition, in machine learning, we want the model to have a low **test error** on an unseen test data set too.

  The **test error** or **generalization error** is defined as the expected value of the error on a new input.

  The expectation (average) is taken across the different possible inputs drawn from a distribution of inputs the system encounters in practice.

  The **test error** is usually computed by measuring the performance of a trained machine learning model on an independent test data set.

The Machine Learning Process



Goal: Minimize training **and test error**. But how?

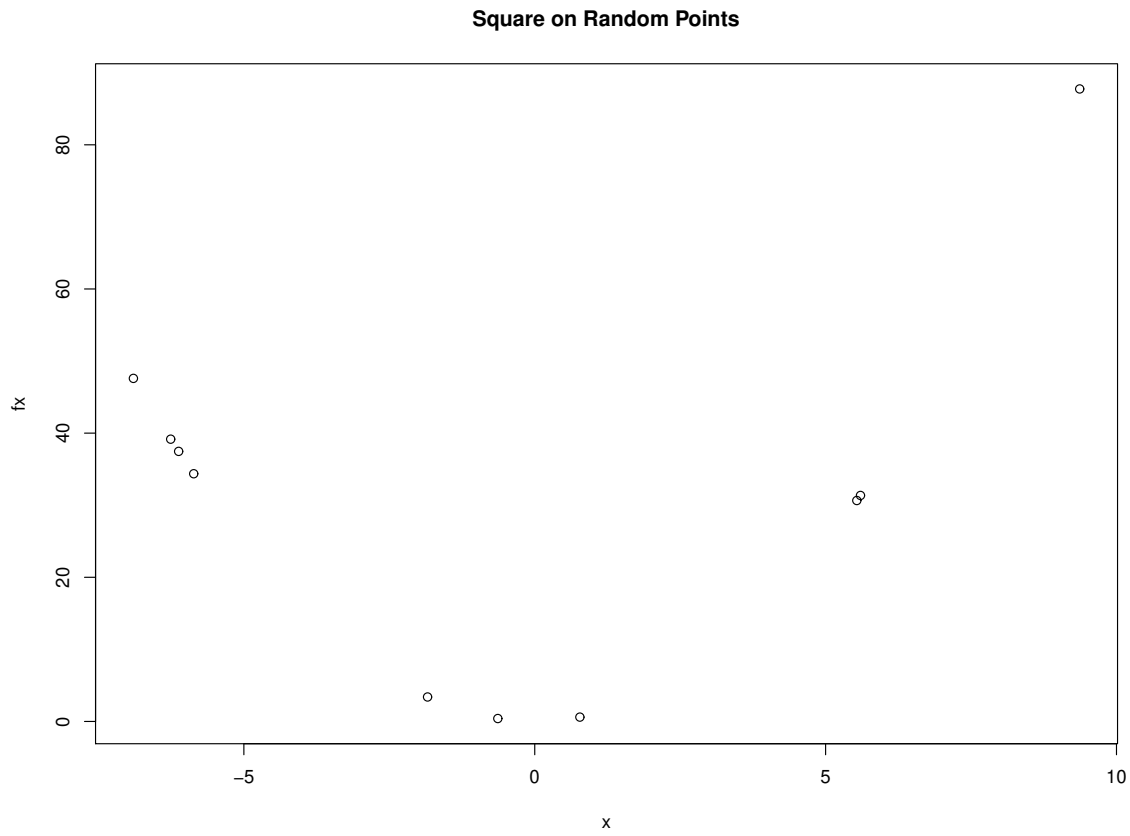
Model Capacity, Overfitting and Underfitting

1. Make the training error small!
2. Make the gap between training and test error small!

1. Reduce underfitting!
2. Reduce overfitting!

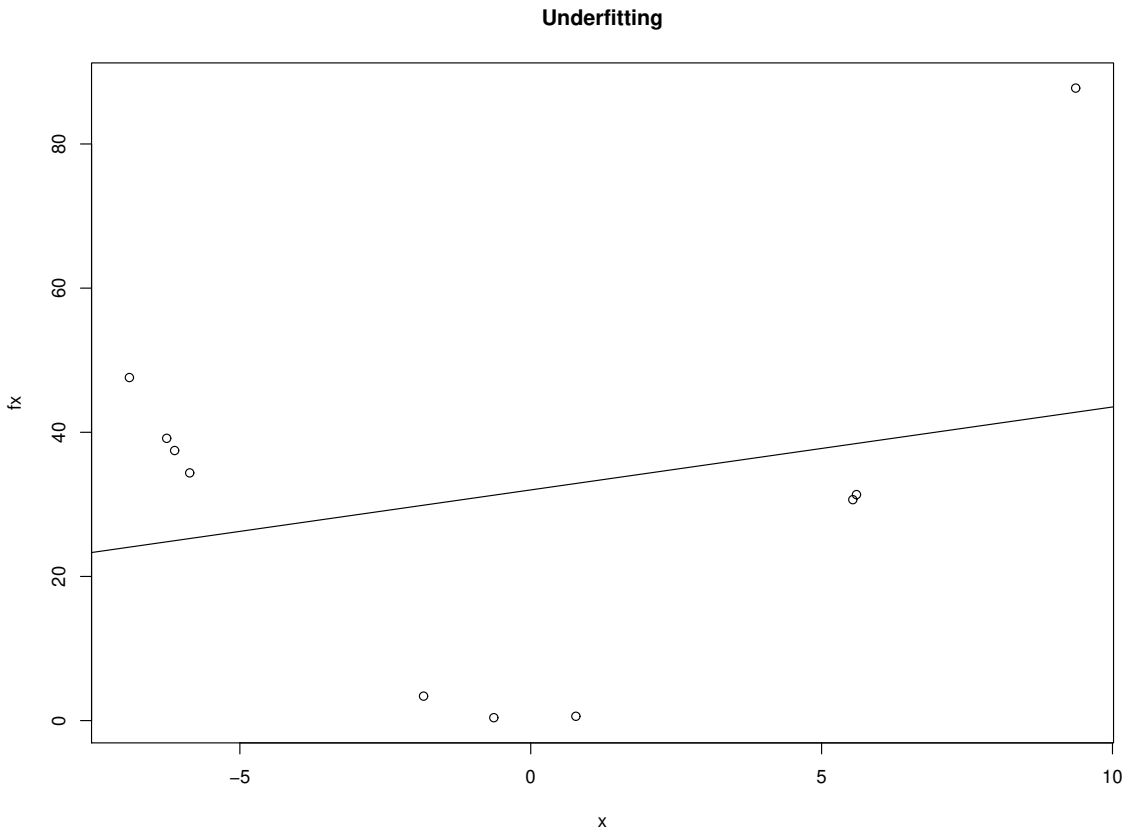
We can control whether a model tends to overfitting or underfitting by adapting the **model capacity**.
Informally, the model's capacity is its ability to fit a wide variety of functions.
Example: In linear regression, we influence the model capacity by the specification of a proper regression model, that is by exactly the variables on which the dependent variable depends.
This process is called **model selection** in econometrics.

Fitting a Square Function.



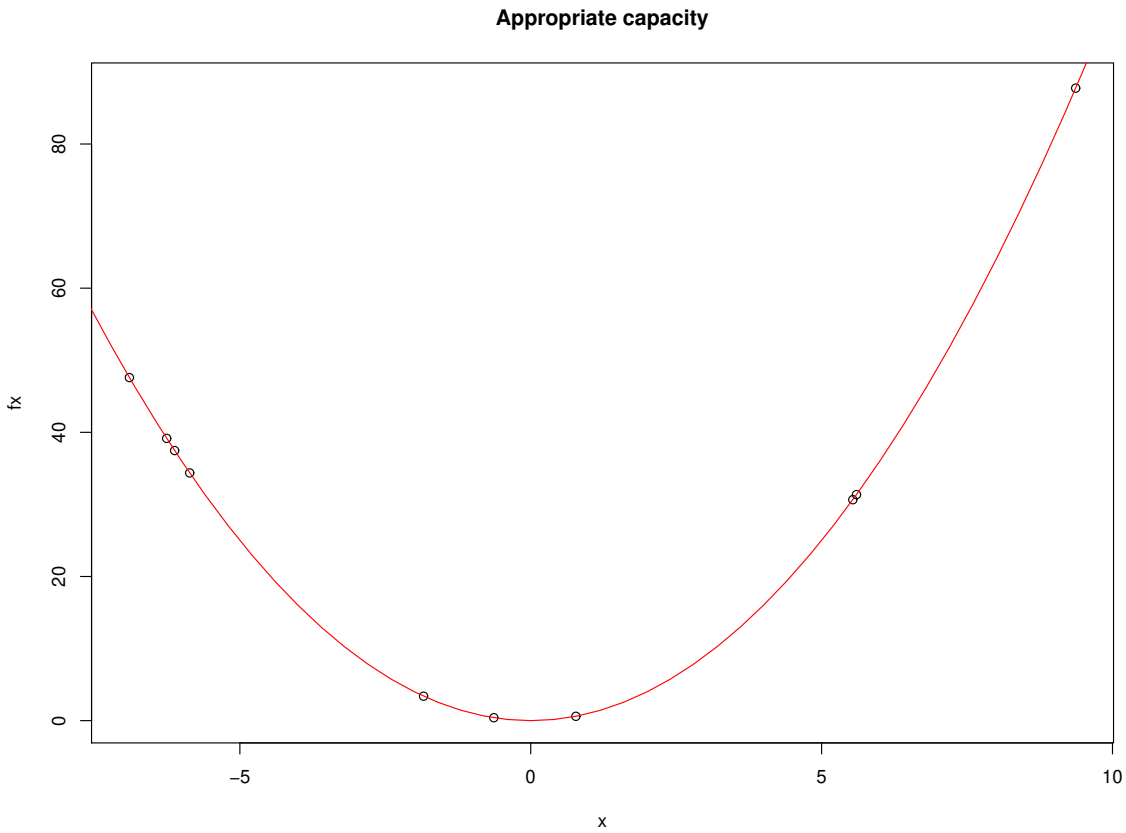
A random training set for $y = x^2$.

Underfitting



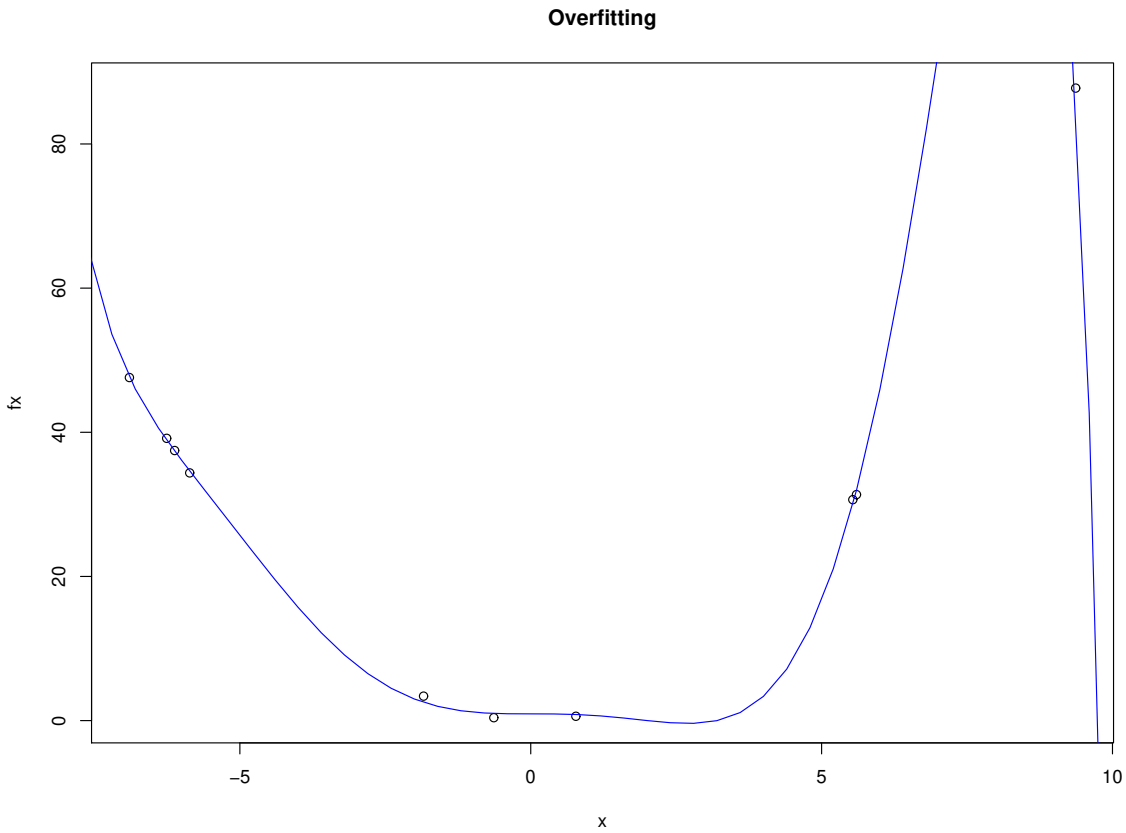
We fit the model $y = \beta_0 + \beta_1 x$.
Not enough capacity to fit the quadratic term.

Appropriate Capacity



We fit the model $y = \beta_0 + \beta_1 x + \beta_2 x^2$.
Appropriate capacity: Correct model selection.

Overfitting



We fit the model $y = \beta_0 + \beta_1 x^3 + \beta_2 x^4 + \beta_3 x^5 + \beta_4 x^6 + \beta_5 x^9$.
Overfitting: Almost no training error, but wrong functional form!

Decomposing the Mean Square Error: Bias and Variance I

Definition

For predicting a variable y given an input vector $\mathbf{x}$ we assume that there exists a function $f(\mathbf{x})$ which describes the relationship between $y = f(\mathbf{x}) + \epsilon$. We call the function $\hat{f}$ which approximates f a (function) **estimator**. Estimation of $\hat{f}$ is the same as estimation of the parameters $\hat{\theta}$ of $\hat{f}$.

Definition

The **bias of an estimator** is defined as

$$\text{bias}(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta \tag{1}$$

where the expectation is over the data (as a set of samples from a random variable) and θ is the true set of parameters of the data generating distribution.

Note: $\hat{\theta}_m$ is the parameter vector estimated from m random data points. We have used 10 points drawn from $[-10, 10]$ in our examples of underfitting, appropriate capacity, and overfitting.

Decomposing the Mean Square Error: Bias and Variance II

Definition

- An estimator $\hat{\theta}_m$ is **unbiased** if $\text{bias}(\hat{\theta}_m) = 0$.
- An estimator $\hat{\theta}_m$ is **asymptotically unbiased** if $\lim_{m\to\infty} \text{bias}(\hat{\theta}_m) = 0$.

Definition

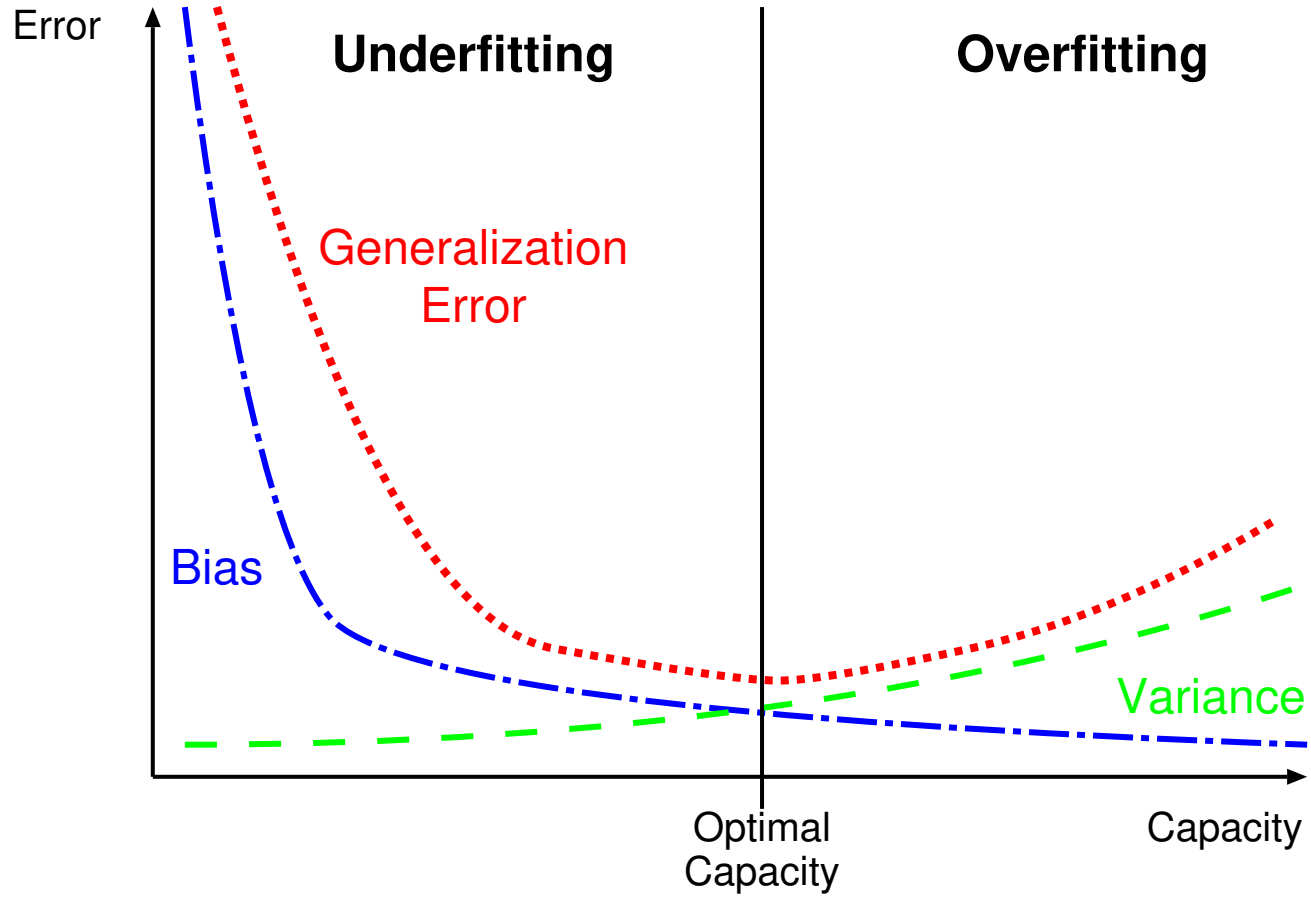
The **variance** of an estimator is simply $\text{Var}(\hat{\theta})$ where the random variable is the training set.

Definition

The mean squared error (MSE) of the estimates can be decomposed as follows:

$$\text{MSE} = E((\hat{\theta}_m - \theta)^2) = \text{bias}(\hat{\theta}_m) + \text{Var}(\hat{\theta}_m) \tag{2}$$

The Relationship of Bias and Variance with Over- and Underfitting



Consistency

Consistency formalizes the desired behavior of an estimator as the training data set grows.

Definition

An estimator is **weakly consistent** iff $\text{plim}_{m\to\infty} \hat{\theta}_m = \theta$.

Definition

plim means **convergence in probability**:
For any $\epsilon > 0$, $P(|\ \hat{\theta}_m - \theta\ | > \epsilon) \to 0$ as $m \to \infty$.

Definition

An estimator is **strongly consistent** iff $\hat{\theta}$ converges almost surely to θ.

Definition

Almost sure convergence of a sequence of random variables $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ to a value $\mathbf{x}$ occurs when $P(\text{plim}_{m\to\infty} \mathbf{x}^{(m)} = \mathbf{x}) = 1$.

Choosing a Model's Capacity

- In the regression example we have used just one way to adapt the model's capacity, namely by adapting the number of input features (e.g. powers of x) and simultaneously adding new parameters with these features.
- However, as we will see in this lesson, there are many ways of adapting the capacity of a model. For example, the model can also specify which family of functions the learning algorithm can choose from (see Architecture Design of a Deep Neural Network).
- This is called the **representational capacity** of the model.
- Often, finding the best function out of this family is a difficult optimization problem.
- A learning algorithm may have limitations of reaching the best model in a model family. This implies that the **effective capacity** of the learning algorithm is below the representational capacity of the model.

The Model's Capacity

- Since the medieval times scientists use Occam's principle [King, 2005] stated in his book *Summa Logicae*:
  Among competing hypothesis that explain known observations well, choose the **simplest** *one!*

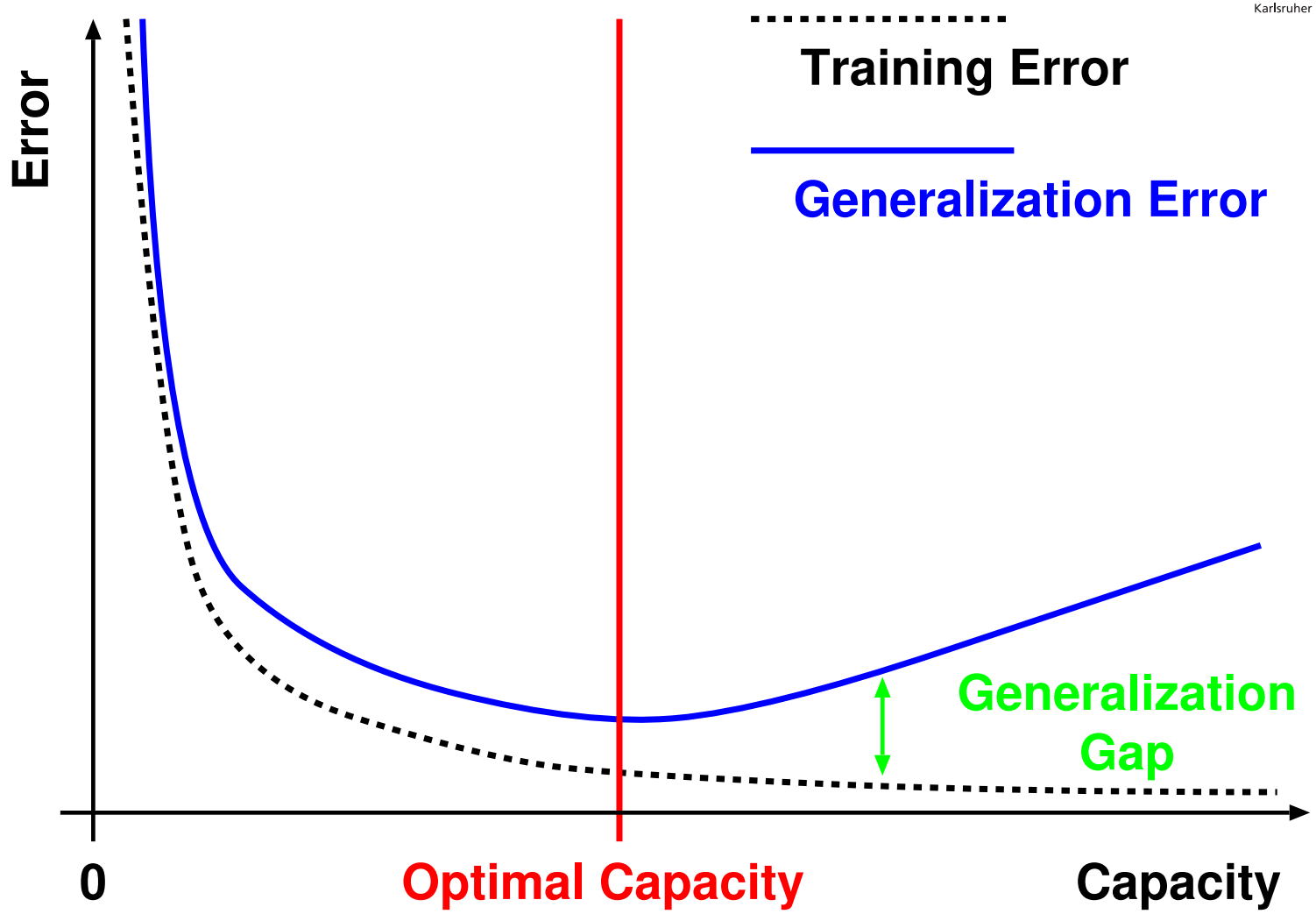
- In the last quarter of the 20th century, statistical learning theory quantified the concept of model capacity in several ways (see e.g. [Vapnik and Chervonenkis, 1971] and [Vapnik, 1995]).

  Widely known is the **Vapnik-Chervonenkis (VC) dimension** which measures the capacity of a binary classifier. The VC dimension is defined as the largest possible value of m for which there exists a training set of m different $\mathbf{x}$ points that the classifier can label arbitrarily.

- The discrepancy between training error and generalization error is bounded from above by a quantity that grows as the model capacity grows, but shrinks as the number of training examples increases.

- These bounds are rarely used in practice, because they are quite loose and it is difficult to determine the capacity of a deep learning model.

The Relationship between Capacity and Error



The generalization error as a function of model capacity is U-shaped.

Parametric and Non-Parametric Models

- Parametric models learn a function described by a parameter vector with **finite and fixed** size before any data is observed.
- Non-parametric models do not have this limitation and can reach arbitrarily high capacity.

  Example: Nearest Neighbor Regression.

  - The model is the entire dataset $\mathbf{X}$ and $\mathbf{y}$.
  - When asked to classify a test point $\mathbf{x}$, the model looks up the nearest entry in the training set and returns the associated dependend value $y_i = \arg\min || \mathbf{X}_{i,:} - \mathbf{x} ||_2^2$.
  - Breaking ties means averaging the y_i for all tied points $\mathbf{X}_{i,:}$.
  - The algorithm is able to achieve the minimal possible training error on any regression data set.
- An other way to generate a non-parametric learning algorithm is e.g.

  1. to specify a model family as a formal language for a genetic programming algorithm
  2. which uses a parametric learning algorithm for estimating the parameters.

The Ideal Model and the Bayes Error

- The *ideal* model is an **oracle** which knows the *true* probability distribution that generate the data (the box **Data Generating Process**).
- However, even an ideal probabilistic model still makes some errors.

Definition

The error incurred by an oracle making predictions from the true distribution $p(\mathbf{x}, y)$ is called **Bayes error**.

- Expected generalization error never increases with the size of the training data set.
- With more training data, non-parametric models decrease the generalization error until the Bayes error is reached.
- Fixed parametric models with less than optimal capacity asymptotically converge to an error limit which is higher than the Bayes error.
- Models with optimal capacity which still have a large gap between training and generalization error exist. By increasing the training data set it may be possible to reduce this gap.

The No Free Lunch Theorem (Informal Version)

Theorem

Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying unobserved points. No machine learning algorithm is universally better than any other.

Implications:

- "(I)f an algorithm does particularily well on average for one class of problems, then it must do worse on average over the remaining problems"
  [Wolpert and Macready, 1997, p. 70].
- Learning algorithms cannot beat random search **on the (universal) class of all problems**. (However, this is of little practical relevance).
- Comparisons reporting the performance of a particular algorithm with a particular parameter setting are of limited utility:
  - These results **do hold** on the **narrow range** of problems considered.
  - **Beware of overgeneralization** of these results to other problems.

See e.g. [Wolpert and Macready, 1997].

Regularization of the Square Function: Ridge Regression

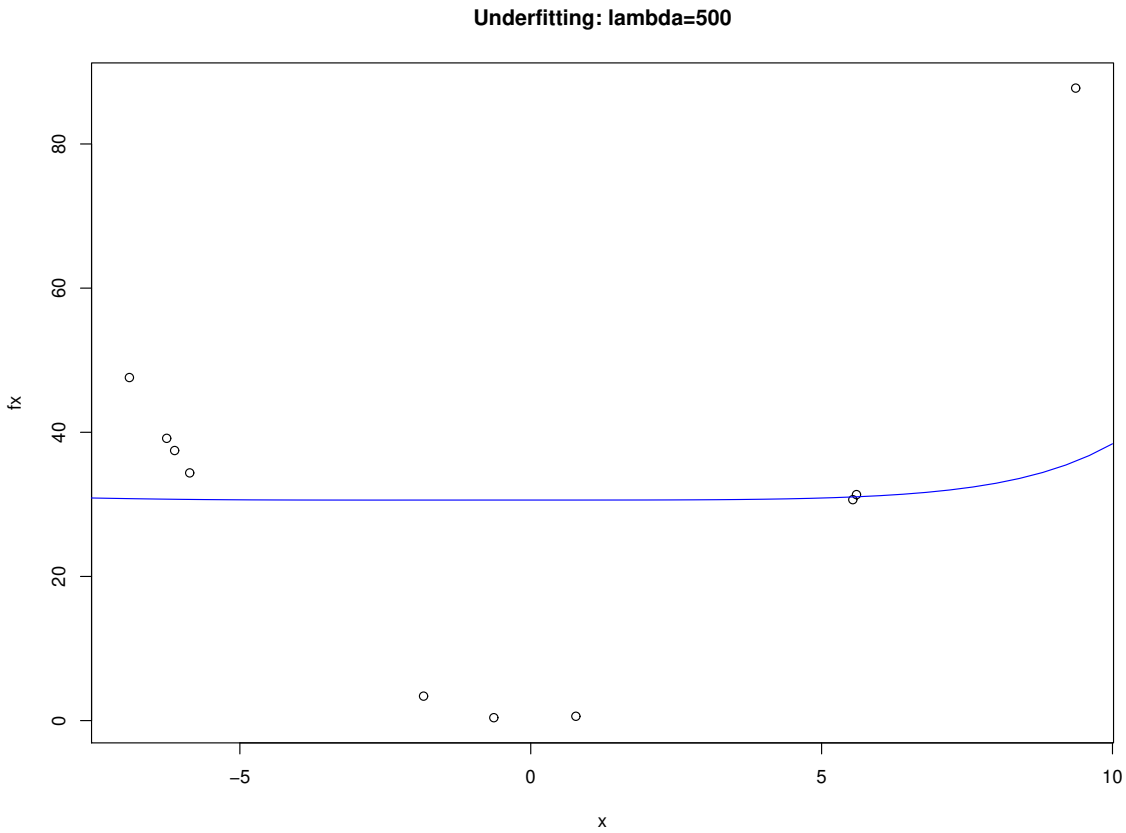
- One possibility (well known in econometrics) is to modify the objective function of the linear regression problem with **weight decay**.

  We do this by adding a penalty term to the objective function of the least squares problem:

$$J(\theta) = \underbrace{(\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta)}_{\text{Mean Square Error}} + \underbrace{\lambda\theta^T\theta}_{\text{Penalty Term on Square of Parameters}}$$

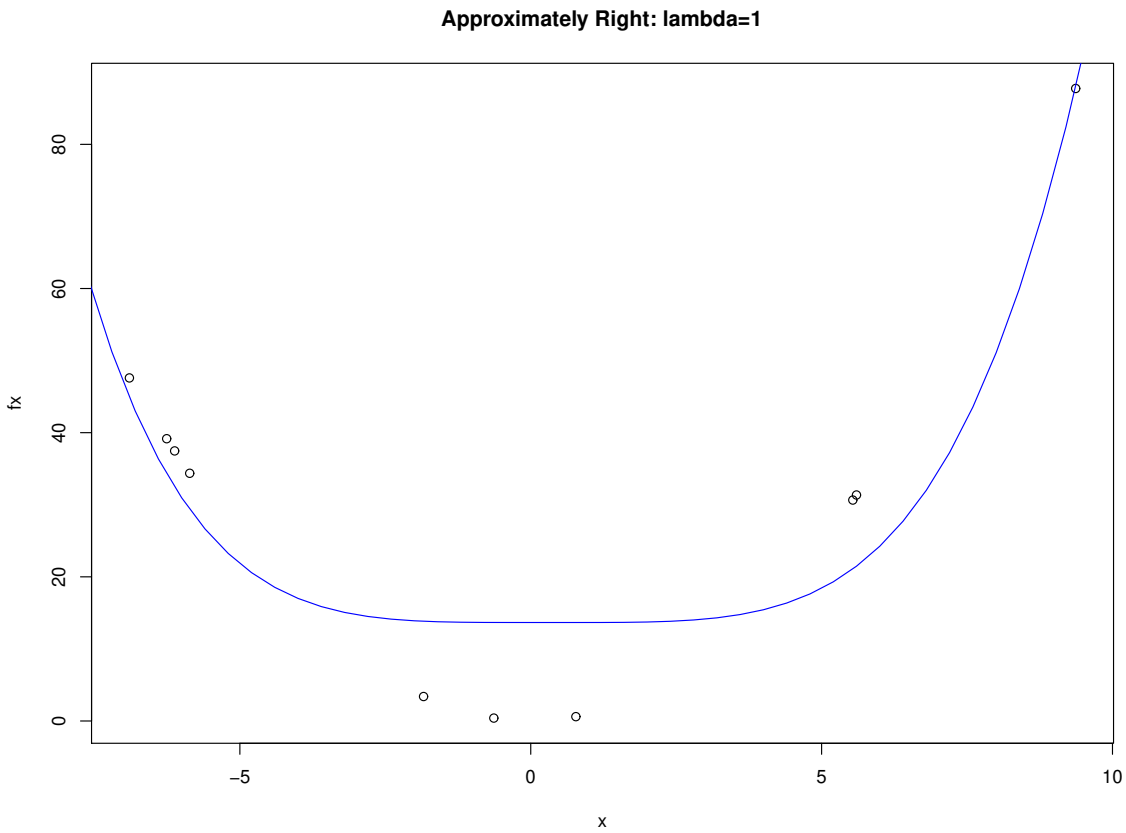
- This is known as **ridge regression** in econometrics. See [Hoerl and Kennard, 1988].
- λ is now a hyperparameter which we vary to search for the minimal **generalization error**. (See next slides!)

Ridge Regression: Underfitting $\lambda = 500$



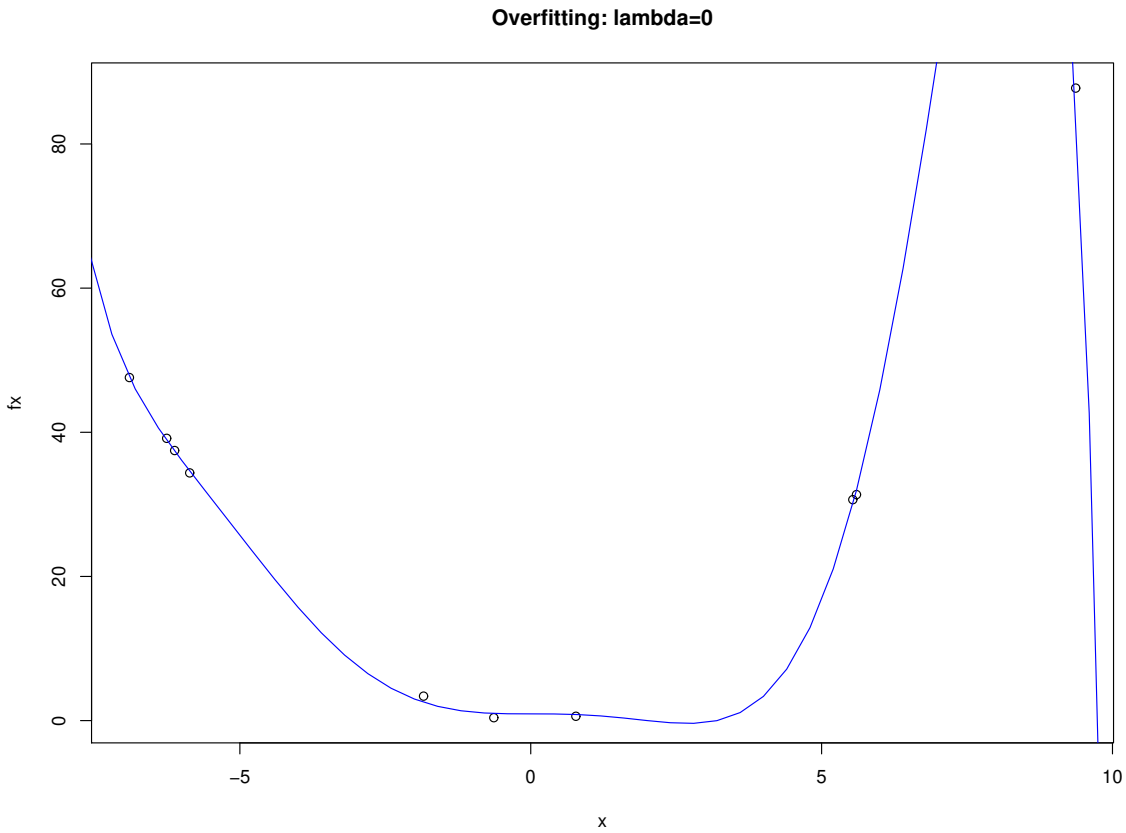
We fit the model $y = \beta_0 + \beta_1 x^3 + \beta_2 x^4 + \beta_3 x^5 + \beta_4 x^6 + \beta_5 x^9$.
We set $\lambda = 500$ to force underfitting.

Ridge Regression: Approximately right $\lambda = 1$



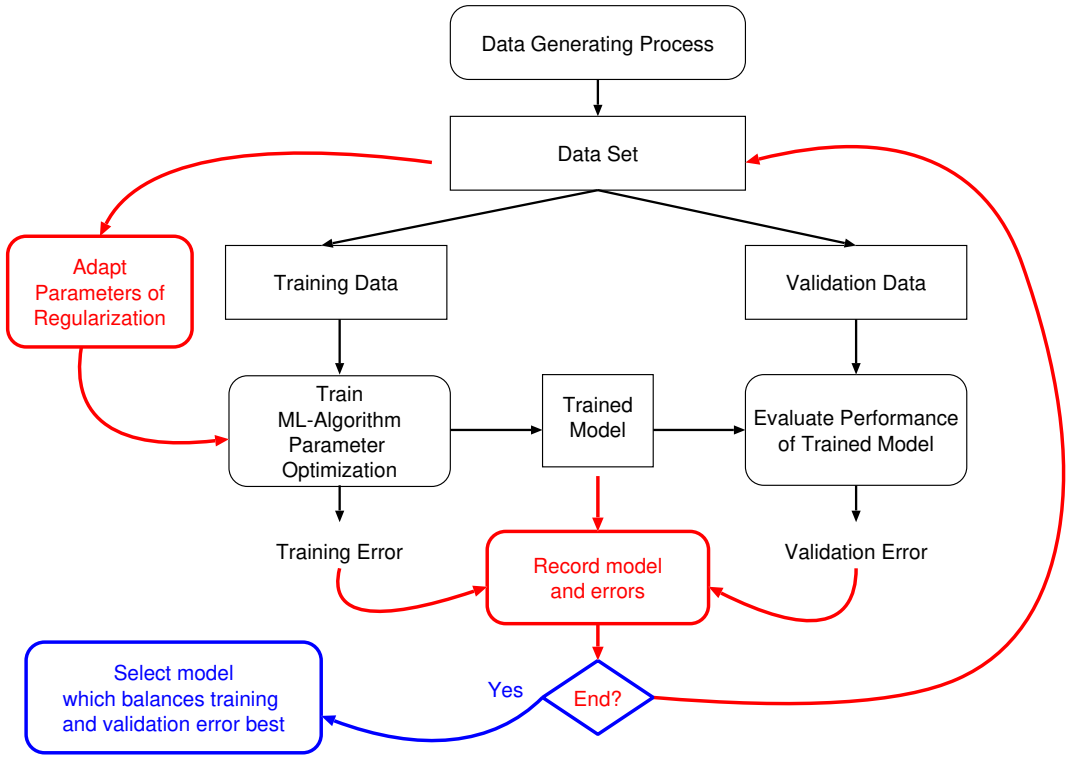
We fit the model $y = \beta_0 + \beta_1 x^3 + \beta_2 x^4 + \beta_3 x^5 + \beta_4 x^6 + \beta_5 x^9$.
We set $\lambda = 1$ which is a crude approximation.

Ridge Regression: Overfitting $\lambda = 0$



We fit the model $y = \beta_0 + \beta_1 x^3 + \beta_2 x^4 + \beta_3 x^5 + \beta_4 x^6 + \beta_5 x^9$.
We set $\lambda = 0$ which reproduces the overfitting solution of least squares.

The Learning Process with Regularization



Goal: Minimize training and validation error.

- **Hyper Parameter Tuning** (Outer loop): Minimize validation error on the validation data by varying the hyperparameters of the regularization method chosen (Red loop).
- **Parameter Training** (Inner "loop"): Minimize the training error by varying the parameters of the model with the machine learning algorithm (In black).

The Problem of Verification and Validation of the Trained Deep Neural Network Learning Algorithm

From a scientist's (or a manager's) point of view, our efforts still are insufficient:

- Does the deep neural network learning model correspond to reality?

  This is also a philosophical question e.g. about causality.
- Does the deep neural network learning model implement the conceptual (mathematical model)?

  In practice, this is about the properties of the framework we use ...
- Does the model generalize?

  Since we have used the validation data in the regularization process, **validation data should not be used in the model validation process**.

  We require **independence** between training data, validation data, and test data.

  The test data is used to assess the generalization error.

Terminology

Data sets are named differently in scientific & industrial communities:

	Statistics/Machine Learning [James et al., 2013, p. 176-178]	OR, Simulation, Engineering [Oberkampf and Roy, 2010]
Parameter Optimization	Training data.	Training data.
Hyperparameter Tuning	Validation data. Hold-Out data.	Test Data.
Model validation	Test data.	Validation data.

We use the statistics and machine learning terminology.

Complex Models and the Need for Verification and Validation

Deep neural networks are **complex models**. Complex models immediately give rise to fundamental problems with

1. defining system as opposed to external influences,
2. issues of causality,
3. human behavior,
4. measuring systems responses,
5. assessing the accuracy of the model.

Definition

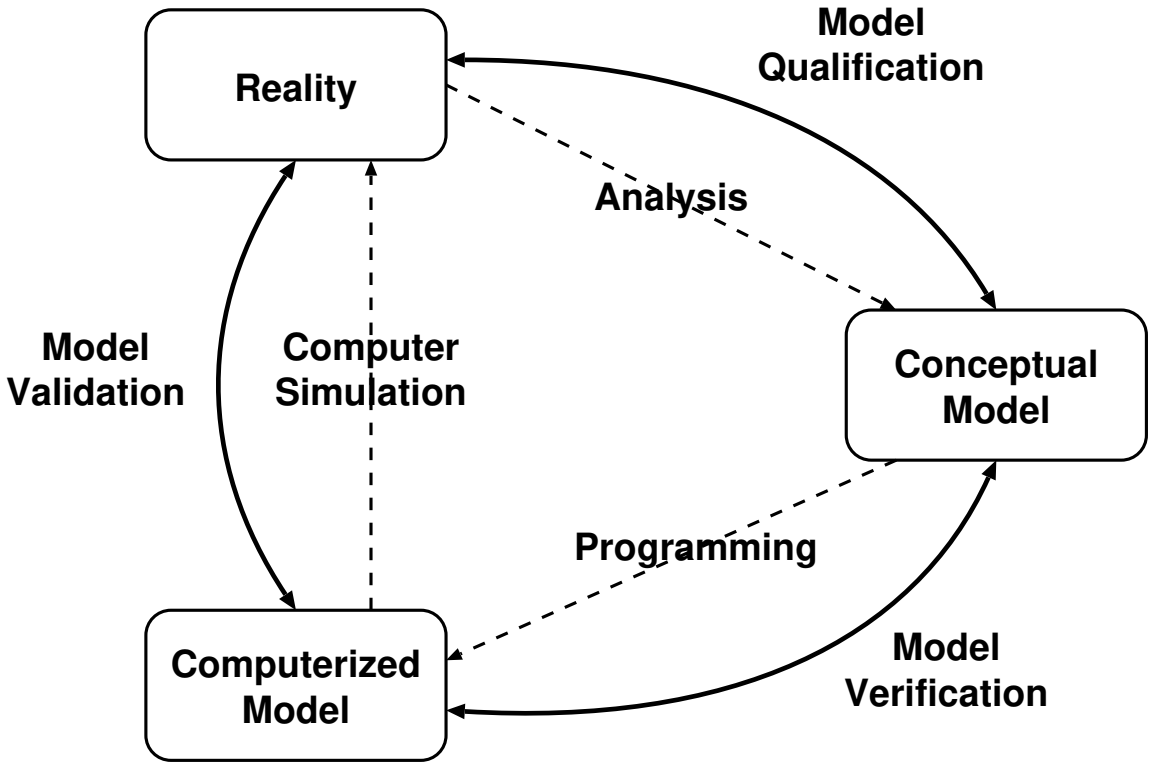
Model verification means substantiation that the computerized model represents a conceptual model within specific limits of accuracy.

Definition

Model validation means substantiation that the computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model.

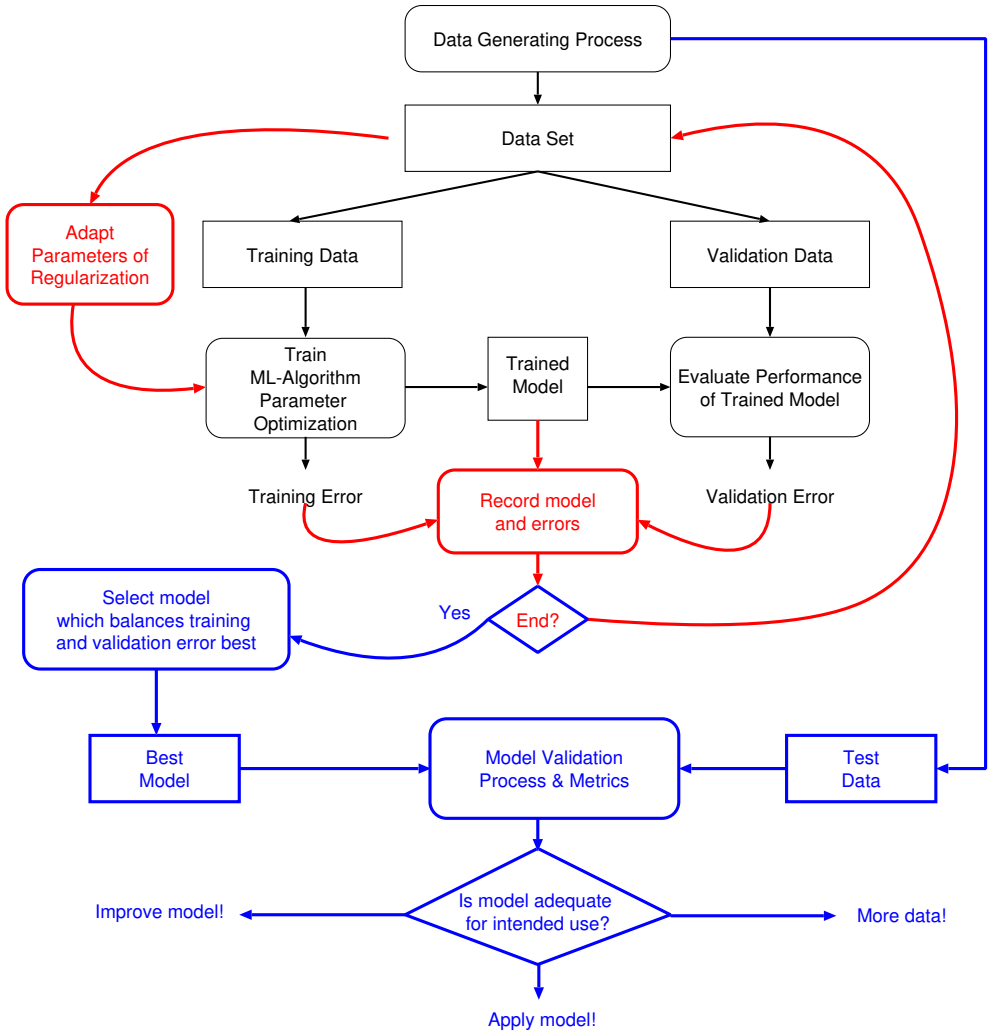
See [Schlesinger, 1979].

The Relation of Modeling and Simulation to Verification and Validation



From [Schlesinger, 1979].

The Learning Process with Regularization and Validation

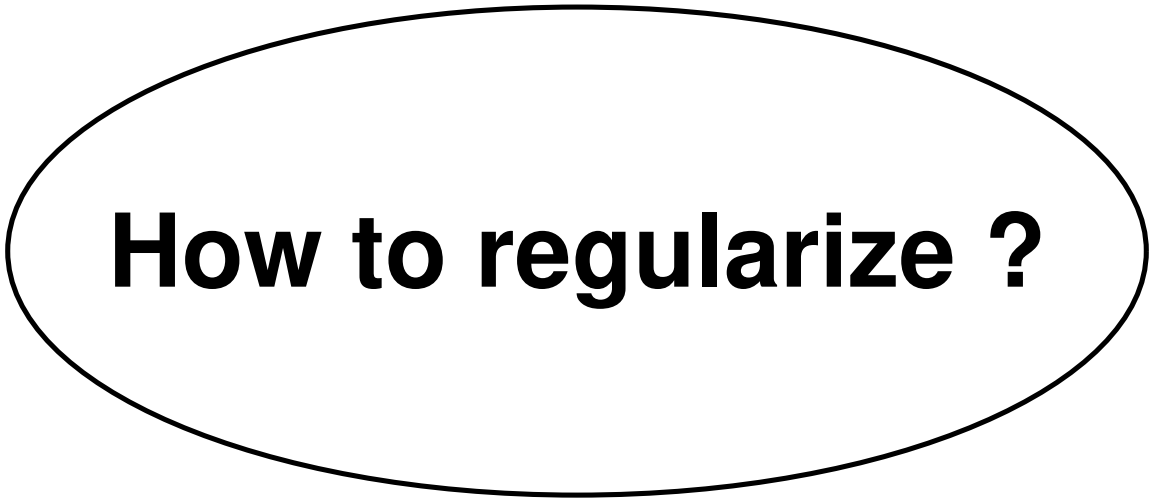


What is Regularization?

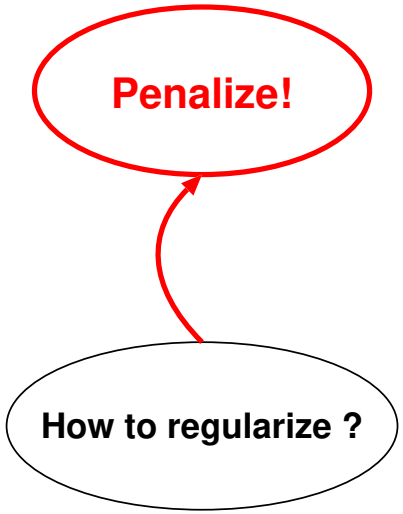
Definition

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. [Goodfellow et al., 2016b, p. 117]

How to Regularize?



Penalize!

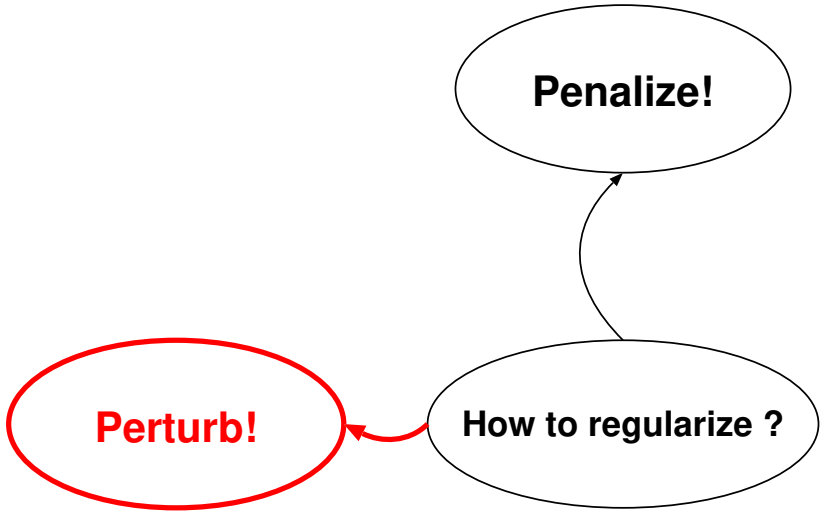


Definition

To reduce the test error, the regularization strategy consists in

1. adding a weighted new term (the regularizer) to the objective function and
2. in searching the neural network model with the lowest test error by repeatedly retraining the neural network with different weights of the regularizer.

Perturb!

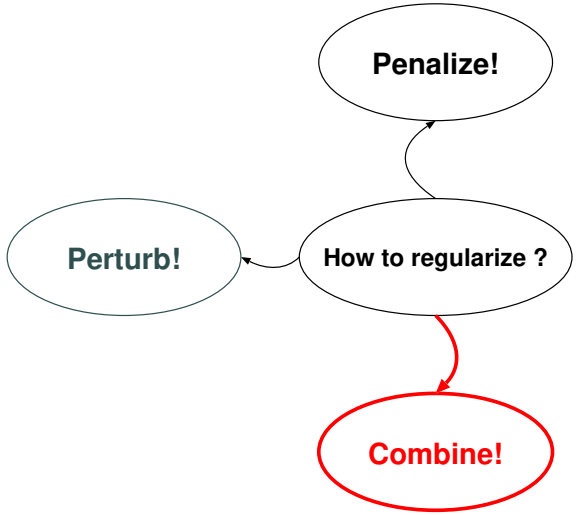


Definition

To reduce the test error, the regularization strategy consists in perturbing with noise

1. the input data or
2. the output data or
3. the weights or
4. the structure of the network
5. or any combination of these.

Combine!

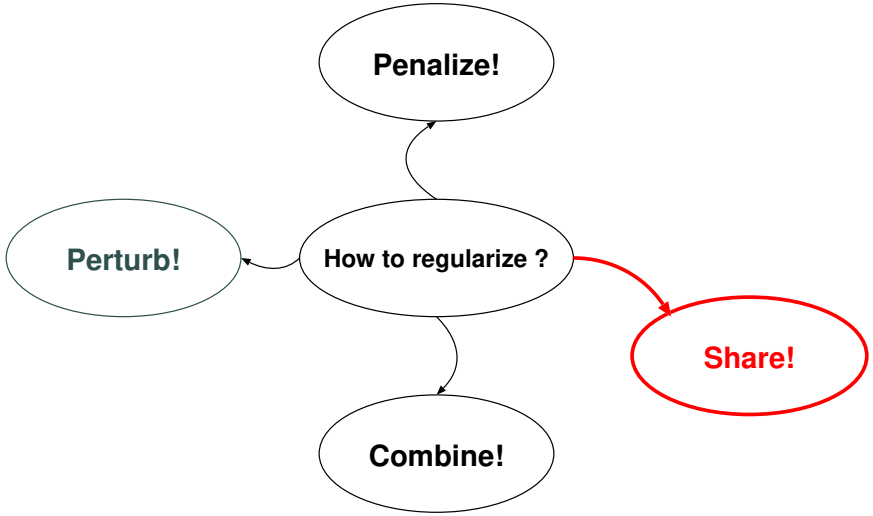


Definition

To reduce the test error, the regularization strategy consists in combining

1. examples or
2. models or
3. training and test error or
4. or any combination of these.

Share!

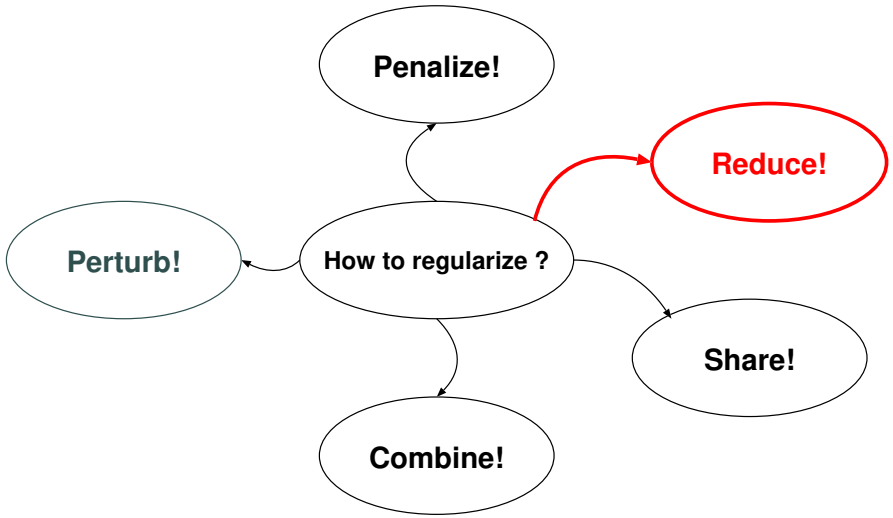


Definition

To reduce the test error, the regularization strategy consists in sharing (reusing)

1. parameters or
2. features or
3. layers or
4. any any other part of a neural network model.

Reduce!

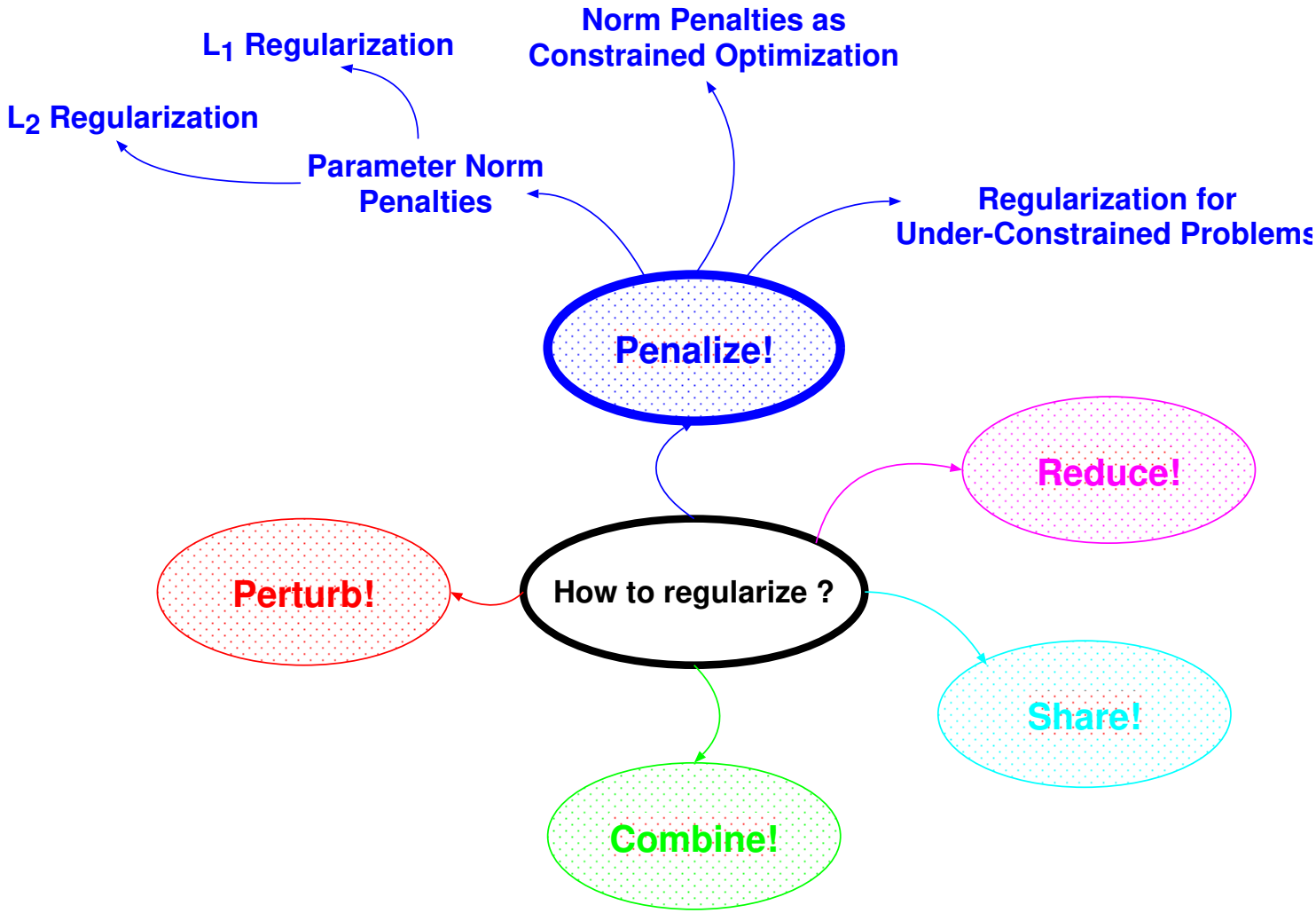


Definition

To reduce the test error, the regularization strategy consists in reducing

1. the dimensionality of the data space by assuming the data is on the surface of a low-dimensional manifold.

Penalize!


L_1 Regularization
Norm Penalties as Constrained Optimization
L_2 Regularization
Parameter Norm Penalties
Regularization for Under-Constrained Problems
Penalize!
Reduce!
Perturb!
How to regularize ?
Share!
Combine!

The Regularized Objective Function

Many regularization approaches are based on **limiting** the capacity of models (neural networks, linear regression, logistic regression) by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J to get a regularized objective function $\bar{J}$:

$$\bar{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha\Omega(\theta) \tag{3}$$

where $\alpha \in [0, \infty]$ is a hyperparameter that weights the relative contribution of the norm penalty Ω to the standard objective function J. $\alpha = 0$ means no regularization.

- Usually for neural networks only the weights $\mathbf{W}^{(i)}$ are penalized.
  Reasons: The $\mathbf{b}^{(i)}$ require less data to estimate and regularizing them can introduce a significant underfit.
- In the following θ are all parameter of a model and ω denotes subset of regularized parameters of a model.
- In deep neural networks, we may use a set of weight parameters for groups of parameters (all weights of a layer, all weights of column, ...). However, this makes hyperparameter tuning more expensive.

The Regularized Objective Function. Repetition

- The basic idea of regularization is the application of the **method of Lagrange** to add constraints/penalties on the parameters of the deep neural network.
- The mathematical basis of norm regularization is the method of Lagrange and Karush-Kuhn-Tucker theorem.
- See the section on **constraint nonlinear optimization** in the unit on the binary coded genetic algorithm as function optimizer (`03INNGATheGAAsFunctionOptimizer`).

L^2 Parameter Regularization

- ... aka (also kown as) weight decay, ridge regression, Tikhonov regularization
- The regularization term added to the objective function is $\Omega(\theta) = \frac{1}{2} \mid\mid \omega \mid\mid_2^2$. (This is the special case of shrinking the parameter vector towards 0.)
- Let $\theta = \omega$. The regularized objective function is:

$$\bar{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\omega; \mathbf{X}, \mathbf{y}) + \frac{\alpha}{2} \omega^T \omega \tag{4}$$

with the parameter gradient

$$\nabla_\omega \bar{J}(\omega; \mathbf{X}, \mathbf{y}) = \nabla_\omega J(\omega; \mathbf{X}, \mathbf{y}) + \alpha\omega \tag{5}$$

- The weight update is

$$\omega \leftarrow \omega - \epsilon(\alpha\omega + \nabla_\omega J(\omega; \mathbf{X}, \mathbf{y})) \tag{6}$$

Separating gradient and regularization effect on ω

$$\omega \leftarrow (1 - \epsilon\alpha)\omega - \epsilon\nabla_\omega J(\omega; \mathbf{X}, \mathbf{y}) \tag{7}$$

This means that the parameter vector ω shrinks by a constant factor in **each** step.

L^2 Parameter Regularization: What happens at the End of the Training? Case 1: α shrinks to 0.

- The parameters ω^* with minimal unregularized training costs are $\omega^* = \arg\min_\omega J(\omega)$.
- We approximate J by the first two terms of its Taylor series expansion:

$$\hat{J}(\omega) = J(\omega^*) + \frac{1}{2}(\omega - \omega^*)^T \mathbf{H}(\omega - \omega^*) + R \tag{8}$$

  For a linear regression, $R = 0$. We assume R is small.
- At the minimum of $\hat{J}$ we have $\nabla_\omega \hat{J}(\omega) = \mathbf{H}(\omega - \omega^*) = \mathbf{0}$.
- Let $\tilde{\omega}$ represent the parameters at the minimum of the regularized objective function:

$$\alpha\tilde{\omega} + \mathbf{H}(\tilde{\omega} - \omega^*) = \mathbf{0} \tag{9}$$

$$(\mathbf{H} + \alpha\mathbf{I})\tilde{\omega} = \mathbf{H}\omega^* \tag{10}$$

$$\tilde{\omega} = (\mathbf{H} + \alpha\mathbf{I})^{-1}\mathbf{H}\omega^* \tag{11}$$

  As $\alpha \to 0$, $\tilde{\omega}$ approaches ω^*.

L^2 Parameter Regularization: What happens at the End of the Training? Case 2: α grows.

- **H** is symmetric and positive definite.
- The orthonormal decomposition of **H** is $\mathbf{H} = \mathbf{Q}\Lambda\mathbf{Q}^T$ where Λ is the diagonal matrix of eigenvalues λ_i of **H** and **Q** is an orthonormal basis of eigenvectors of **H**.
- We rewrite equation (11) by substituting **H** by its decomposition as

$$\tilde{\omega} = \mathbf{Q}(\Lambda + \alpha\mathbf{I})^{-1}\mathbf{Q}^T\omega^* \tag{12}$$

- Regularization means rescaling of ω_i^*:

$$\omega_i^* = \frac{\lambda_i}{\lambda_i + \alpha} \tag{13}$$

1. The effects of regularization are rather small along the directions for which $\lambda_i >> \alpha$
2. Components for which $\lambda_i << \alpha$ are shrunk to nearly zero.

L^1 Regularization

- ... aka LASSO (least absolut shrinkage and selection operator) for linear regression [Tibshirani, 1996].
- The (general) L^1 Regularizer is $\Omega(\theta) = || \omega - \omega^{(o)} ||_1 = \sum_i | \omega_i - \omega_i^{(o)} |$ where $\omega^{(o)}$ is the paramater vector towards which we want to shrink ω. (Usually, $\omega^{(o)} = \mathbf{0}$.)
- The regularized objective function is:

$$\bar{J}(\omega; \mathbf{X}, \mathbf{y}) = J(\omega; \mathbf{X}, \mathbf{y}) + \alpha || \omega ||_1 \tag{14}$$

with the parameter gradient

$$\nabla_\omega \bar{J}(\omega; \mathbf{X}, \mathbf{y}) = \nabla_\omega J(\theta; \mathbf{X}, \mathbf{y}) + \alpha \, \text{sign}(\omega) \tag{15}$$

where $\text{sign}(\omega)$ is simply the vector of signs of ω.

- The update rule is

$$\omega \leftarrow \omega - \epsilon(\alpha \, \text{sign}(\omega) + \nabla_\omega J(\omega; \mathbf{X}, \mathbf{y})) \tag{16}$$

This means that the parameter vector ω is updated with a constant factor with the sign of $\text{sign}(\omega_1)$.

L^1 Regularization.

- We approximate J by the first two terms of its Taylor series expansion at its minimum ω^*:

$$\hat{J}(\omega; \mathbf{X}, \mathbf{y}) = J(\omega^*; \mathbf{X}, \mathbf{y}) + \sum_i \left(\frac{1}{2} H_{i,i} (\omega_i - \omega_i^*)^2 + \alpha \mid \omega_i \mid \right) + R \tag{17}$$

- The analytical solution of minimizing this approximate cost function is:

$$\omega_i = \text{sign}(w_i^*) \max \left(\mid \omega_i \mid - \frac{\alpha}{H_{i,i}}, 0 \right) \tag{18}$$

L^1 Regularization. 4 Cases

1. Let $\omega_i > 0$ for all i.
   $w_i^* \leq \frac{\alpha}{H_{i,i}}$: The optimal value of ω_i is simply $\omega_i = 0$.
2. Let $\omega_i > 0$ for all i.
   $w_i^* > \frac{\alpha}{H_{i,i}}$:
   The optimal value of ω_i is $\omega_i - \frac{\alpha}{H_{i,i}}$.
3. Let $\omega_i < 0$ for all i.
   $\mid w_i^* \mid \leq \frac{\alpha}{H_{i,i}}$:
   The optimal value of ω_i is simply $\omega_i = 0$.
4. Let $\omega_i < 0$ for all i.
   $\mid w_i^* \mid > \frac{\alpha}{H_{i,i}}$:
   The optimal value of ω_i is $\omega_i + \frac{\alpha}{H_{i,i}}$.

The Sparsity of L^1 Regularization.

- The consequence of this is that L^1 regularization results in a solution that is more **sparse** than the solution of a L^2 regularization.
- L^1 regularization has been used extensively as a feature selection mechanism. This simplifies machine learning problems by choosing a small (sparse) set of features out of the set of all available features.
- The LASSO model of [Tibshirani, 1996] integrates an L^1 penalty with a linear model with a least squares cost function.

  The L^1 penalty forces a subset of weights to zero indicating the features which may be safely discarded.

The Interpretation of Regularization as Maximum A Posteriori (MAP) Bayesian Estimation

- L^2 regularization is equivalent to MAP Bayesian inference with a Gaussian prior on the weights.
- For L^1 regularization the penalty is $\alpha\Omega(\omega) = \alpha \sum_i \mid \omega_i \mid$.
- This is equivalent to a log-prior of an isotropic Laplace distribution over $\omega \in R^n$:

$$\log p(\omega) = \sum_i \log \text{Laplace}(\omega_i; 0, \frac{1}{\alpha}) \quad (19)$$

$$= -\alpha \mid\mid \omega \mid\mid_i + n \log \alpha - n \log 2 \quad (20)$$

- We ignore $n \log \alpha - n \log 2$, because this term does not depend on ω.

Norm Penalties as Constrained Optimization

Consider once again the regularized objective function (equation (3)):

$$\bar{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha\Omega(\theta) \tag{21}$$

We may constrain $\Omega(\theta) \leq k$. This leads to the following Lagrange function:

$$\mathcal{L}(\theta, \alpha; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha(\Omega(\theta) - k) \tag{22}$$

The solution of the constrained problem is:

$$\theta^* = \arg\min_{\theta} \max_{\alpha \geq 0} \mathcal{L}(\theta, \alpha; \mathbf{X}, \mathbf{y}) \tag{23}$$

An optimal value $\alpha^* > 0$ will encourage $\Omega(\theta)$ to shrink so that $\Omega(\theta) = k$.
Let us fix α at α^*.

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta, \alpha^*; \mathbf{X}, \mathbf{y}) = \arg\min_{\theta} J(\theta; \mathbf{X}, \mathbf{y}) + \alpha^*\Omega(\theta) \tag{24}$$

This is exactly the same as minimizing the regularized training function in equation (21).
So why should we care?

Norm Penalties as Constrained Optimization

- Equation (24) gives us an insight into the interpretation of choosing α:
  1. Larger α will result in a smaller constraint region of the parameters.
  2. Smaller α will result in a larger constraint region of the parameters.
- We can use explicit constraints on parameters instead of penalties directly by modifiying the stochastic gradient descent algorithm by back projection of the parameters:
  1. Take a step downhill on $J(\theta)$.
  2. Project θ back to the nearest point that satisfies $\Omega(\theta) < k$.
- **Example:** For a Bayesian interpretation of the weights of a layer, we renormalize the weights of the layer to 1.
- Using back projection instead of penalties avoids getting stuck in local minima.
- Back projection can make optimization procedures more stable, because the parameter values can not explode.
- Variations are possible: Constraining the norm of each column of the weight matrix of a neural network layer, rather than constraining the Frobenius norm of the entire weight matrix ($\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} \mid x_{i,j} \mid^2}$).

Regularization and Under-Constrained Problems

Many linear models (regression and PCA) depend on inverting the matrix $\mathbf{X}^T\mathbf{X}$.
This is not possible, if $\mathbf{X}^T\mathbf{X}$ is singular:

- the data-generating distribution truly has no variance in some direction or
- when no variance is observed in some direction:
  Fewer examples (rows of $\mathbf{X}$) than input features (columns of $\mathbf{X}$).

In this case, many forms of regularization correspond to inverting

$$\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I}$$

This is guaranteed to be invertible.

Regularization and Under-Constrained Problems

- Logistic regression is underdetermined, if applied to a problem with linearly separable classes:

  A consequence is that if ω achieves optimal classification, so will $k\omega$ for $k = 1, 2, 3, \ldots$.

  Therefore, a stochastic gradient algorithm will run until numerical overflow occurs.

  Weight decay (by regularization) will solve this problem: The algorithm will stop when the slope of the likelihood is equal to the weight decay coefficient.

- We can also interprete the Moore-Penrose pseudoinverse $\mathbf{X}^+$ of $\mathbf{X}$ as performing a linear regression with weight decay:

$$\mathbf{X}^+ = \lim_{\alpha \to 0} (\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})^{-1}\mathbf{X}^T$$

  The pseudoinverse stabilizes the underdetermined problem by regularization.

Sparse Representations

A neural network is represented by

1. its weights and
2. and its activiations (data $\mathbf{x} = \mathbf{h}^0$ and hidden activation values $\mathbf{h}^i$).

We have already used L_1 regularization to induce a sparse representation of the weights of a neural network.
However, we can use the same approach on data and hidden activation units:

$$\bar{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha\Omega(\mathbf{h}) \tag{25}$$

where $\alpha \in [0, \infty)$ weights the relative contribution of the norm penalty.
For L_1 regularization:

$$\Omega(\mathbf{h}) = || \mathbf{h} ||_1 = \sum_i | h_i |$$

Sparse Representations. Variants

- For representations with elements in the unit interval the penalty may be derived from
  - a Student t-prior on the representation ([Olshausen and Field, 1996], [Bergstra, 2011]) or
  - a Kullback-Leibler divergence [Larochelle and Bengio, 2008].
- For regularizing the average activiation over several examples, use $\frac{1}{m}\sum_i \mathbf{h}^{(i)}$. See [Lee et al., 2008] and [Goodfellow et al., 2009].
- In **orthogonal matching pursuit** (OMP, [Pati et al., 1993]), **W** is assumed to be orthogonal and an input **x** is encoded with the representation **h** that solves

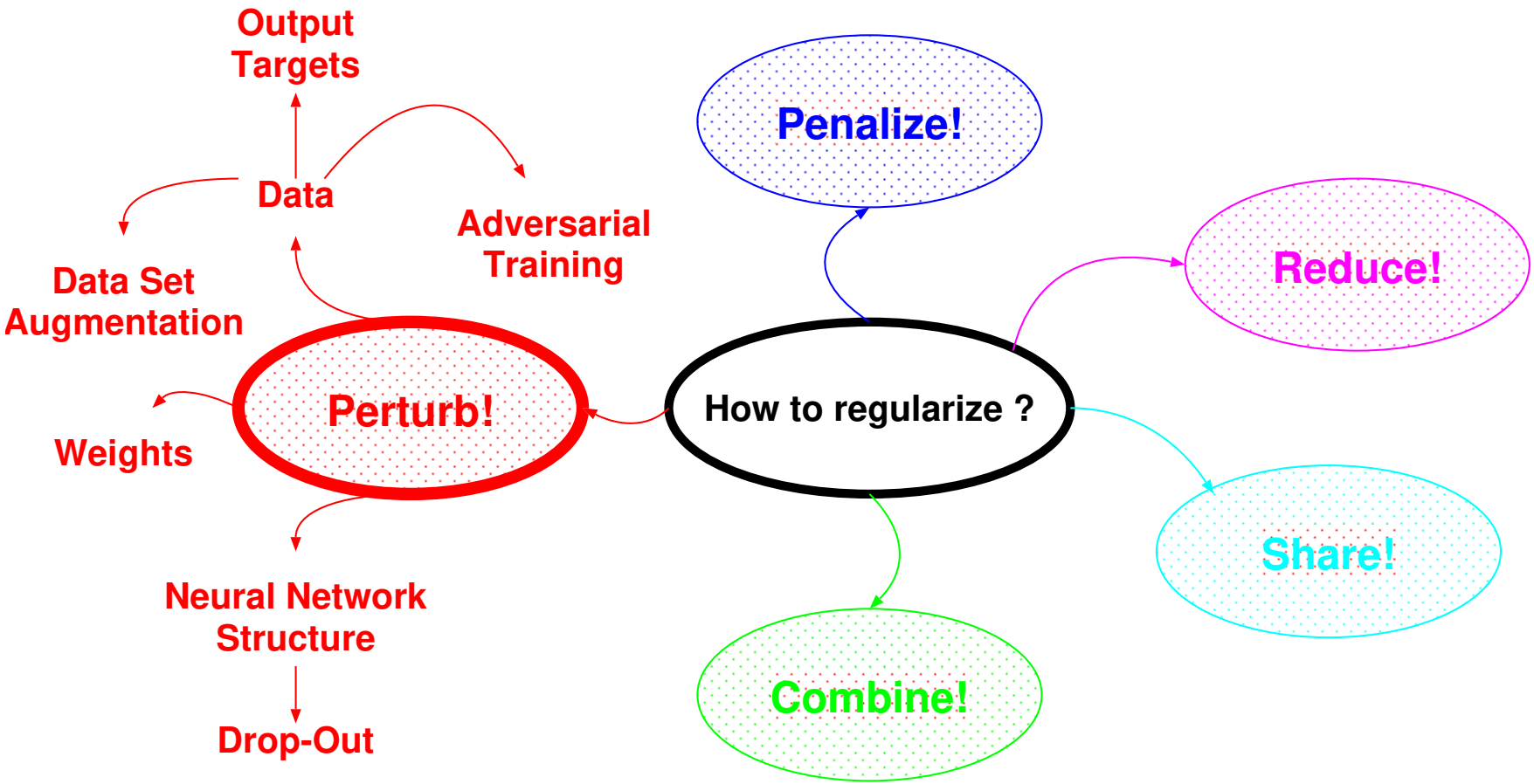
$$\arg \min_{\mathbf{h}, ||\mathbf{h}||_0 < k} || \mathbf{x} - \mathbf{W}\mathbf{h} ||^2$$

  where $|| \mathbf{h} ||_0$ is the number of nonzero entries of **h**.

  This can be uses as a feature extractor for deep architectures [Coates and Ng, 2011].

(Not relevant for the exam. For the curious!)

Perturb!


Output Targets
Data
Adversarial Training
Data Set Augmentation
Perturb!
Weights
Neural Network Structure
Drop-Out
How to regularize ?
Penalize!
Reduce!
Share!
Combine!

Data Set Augmentation

- **More data** leads to **better generalization** of machine learning models.
- Data set augmentation combines the original data with fake data.
- For classification tasks, fake data generation is easy:

  For $(\mathbf{x}, y)$, we need to subject the high-dimensional vector $\mathbf{x}$ to a variety of transformations to make the classifier invariant to these transformations.
- This approach has been very successful for object recognition: Images are high dimensional and are subject to an enormous range of factors of variation. E.g. shifting, rotating, scaling a few pixels in each direction, slight changes in colour coding have proved quite effective.

  Warning: Transformations should not change the class of an object: No horizontal flips and 180 degree rotations for *b* and *d* or *6* and *9*.

  Warning: Some transformations are hard or impossible to implement: Out-of-plane rotation of 3D-objects in a single image.

  For speech recognition, see [Jaitly and Hinton, 2013].
- Designing proper transformations is based on knowledge about the data and the domain.

Data Set Augmentation. Noise Injection

Problem: Neural networks are not robust to noise in images which are not in the training set [Tang and Eliasmith, 2010].
Solution:
Injection of noise into

- the input data of the neural network [Sietsma and Dow, 1991].
  Input noise injection has been integrated e.g. into the denoising autoencoder [Vincent et al., 2008].
- the hidden units (before the nonlinearity and at the hidden unit activations) [Poole et al., 2014].
  In this framework **dropout** can be seen as a process of constructing new inputs by multiplying activations by random zeroes.

(Not relevant for the exam. For the curious!)

Noise Robustness. Flat Regions

Consider a training set of m labeled examples $\Big\{(\mathbf{x}^(1), y^1), \ldots (\mathbf{x}^(m), y^m), \Big\}$:
The cost function is the Ordinary Least Squares (OLS) function:

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

With each input representation, the weights $\mathbf{W}$ are perturbed by adding $\epsilon_{\mathbf{W}} \sim N(\mathbf{0}, \sigma\mathbf{I})$.
The network is a fully connected standard l-layer multi-layer perceptron. The perturbed model is $\hat{y}_{\epsilon_{\mathbf{W}}}$ with the cost function

$$\tilde{J}_{\mathbf{W}} = E_{p(x,y,\epsilon_{\mathbf{W}})}((\hat{y}_{\epsilon_{\mathbf{W}}}(\mathbf{x}) - y)^2)$$

For small σ, minimization of $\tilde{J}_{\mathbf{W}}$ equals minimization of $J + \sigma E_{p(x,y)}(|| \nabla_{\mathbf{W}} \hat{y}((x) ||^2)$
This type of regularization pushes the model into regions where the minima are surrounded by flat regions [Hochreiter and Schmidhuber, 1994]:

- The model becomes insensitive to small perturbations.

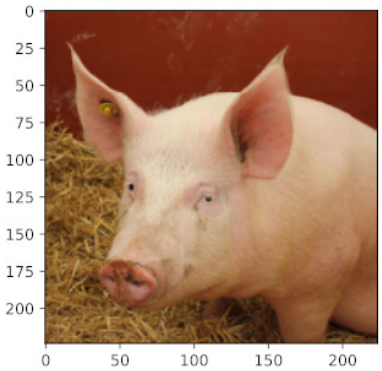
Noise Robustness. Label Smoothing

- Most data sets used in classification tasks have errors in the y labels.
- Maximum likelihood estimation of a softmax classifier and hard targets $(0/1)$ may have convergence problems.
- For this situation, it is recommended to assume that the label is correct with probability $1 - \epsilon$ and otherwise any of the other labels is correct.
- **Label smoothing** models this by using a softmax unit with k-output values with target values of $1 - \epsilon$ and $\frac{\epsilon}{k-1}$.
  The standard cross-entropy loss function is then used with these soft targets.
- For a more detailed explanation, see [Szegedy et al., 2016].

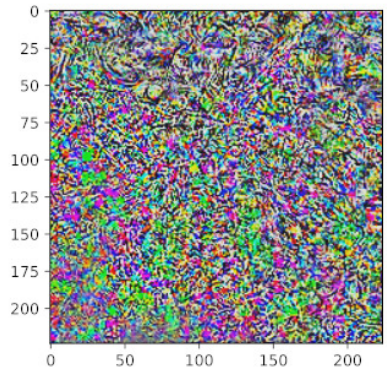
Dropout

Dropout can be seen as a **perturbation** of the perception of an agent, because some random activation values of his neural network are set to zero for some training steps.

Adversarial Training. Motivation

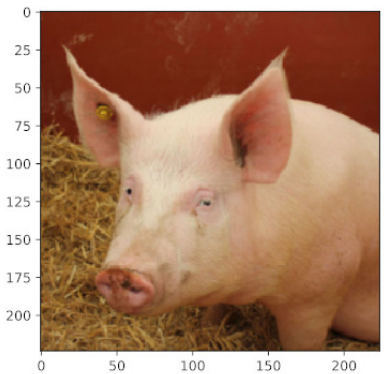


Pig

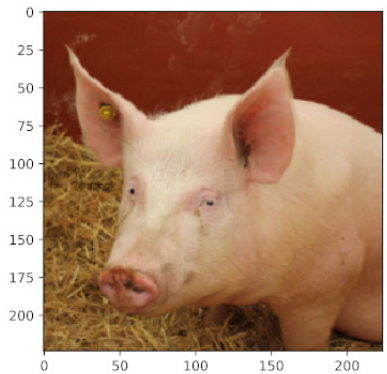


AirlineNoise

Classified as:



Pig!



Airline!

From [Kolter and Madry, 2018].

Adversarial Training

1. Find adversarial examples means maximize the loss of the perturbation δ

$$\arg\max_{||\delta|| \leq \epsilon} J(h_\theta(\mathbf{x} + \delta, y)$$

2. Train a new network with some adversarial examples added. This implies optimization of

$$\min_\theta \frac{1}{|S|} \sum_{x,y \in S} \max_{||\delta|| \leq \epsilon} J(h_\theta(\mathbf{x} + \delta, y)$$

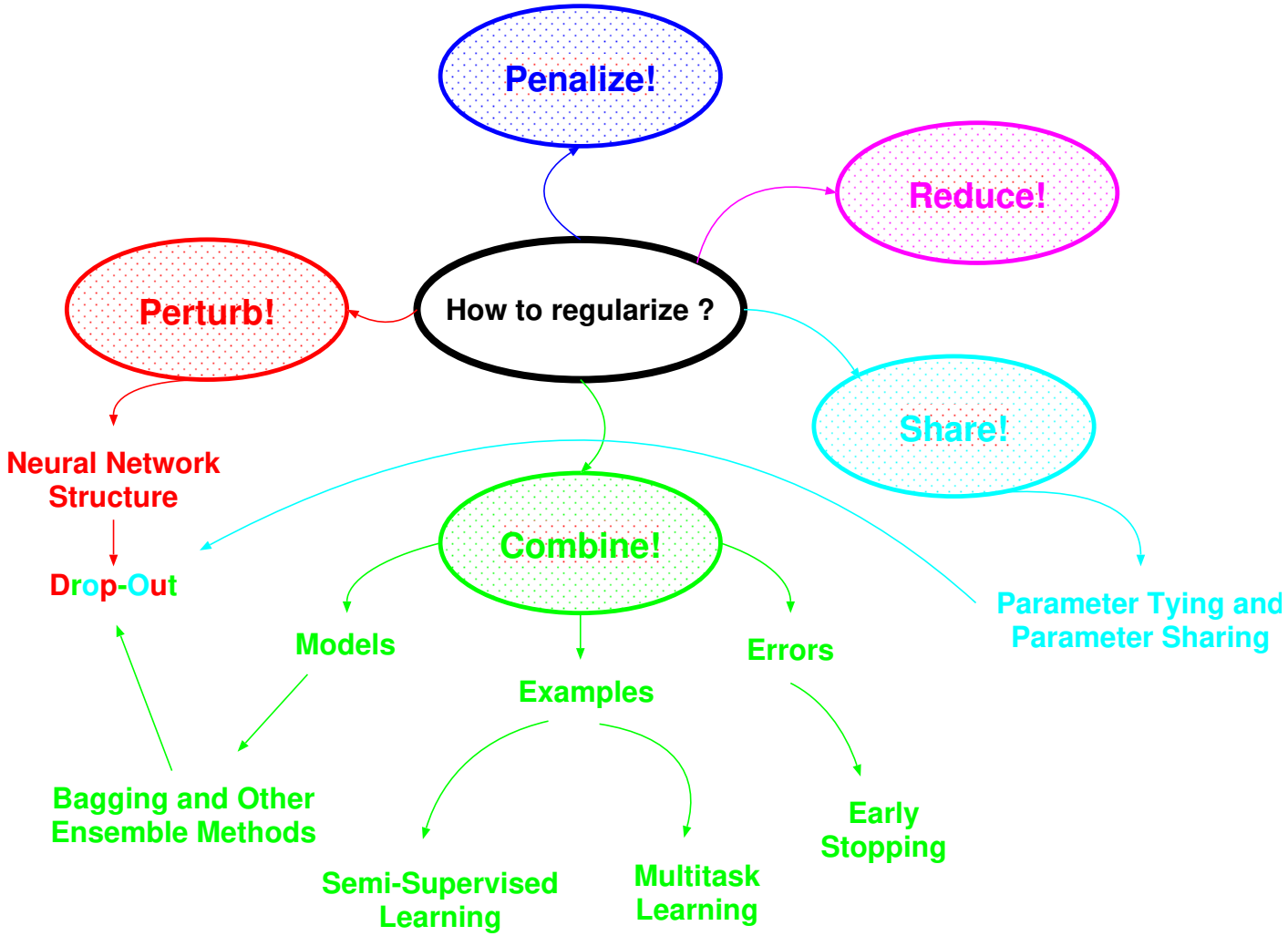
3. Note that the best we can do is to produce an empirically adversarially robust classifier.
   There is no guarantee that this solution is provably robust.

Further References to Adversarial Training

- The first adversarial examples for neural networks performing at a *human-level* were given by [Szegedy et al., 2014] and [Goodfellow et al., 2015].
- The NeurIPS 2018 tutorial of [Kolter and Madry, 2018] also provides a set of jupyter notebooks with python code for this topic *https://adversarial-ml-tutorial.org/*.

(Not relevant for the exam. For the curious!)

Combine!


Penalize!
Reduce!
Perturb!
How to regularize ?
Share!
Neural Network Structure
Drop-Out
Combine!
Parameter Tying and Parameter Sharing
Models
Examples
Errors
Bagging and Other Ensemble Methods
Semi-Supervised Learning
Multitask Learning
Early Stopping

Semi-Supervised Learning

Definition

Semi-supervised learning combines labeled examples from $P(\mathbf{x}, y)$ and unlabeled examples from $P(\mathbf{x})$ to estimate $P(y \mid \mathbf{x})$ or to predict y from $\mathbf{x}$.

- Learning from unlabeled examples is only possible, if an additional assumption holds. If not, it can even lead to worse results.
- An assumption is: Examples that are near together in the input space should also be near together in the output space (The Smoothness Assumption).

  A linear classifier in the new space often achieves better generalizations (see e.g. [Belkin and Niyogi, 2002], [Chapelle et al., 2002])
- Another assumption is: The high-dimensional examples lie (roughly) on a low dimensional manifold (surface/subspace) (The Manifold Assumption).

  In this situation, principal components analysis is used as a preprocessing step to project the data into the new space before applying a classifier on the data in the new space.

Semi-Supervised Learning

- In the context of deep learning, semi-supervised learning usually refers to learning a representation $\mathbf{h} = f(\mathbf{x})$.
- The goals is to learn a representation so that examples from the same class have similar representations. Unsupervised learning can be useful for this.
- E.g. In a character recognition task to learn whether a pixel is a part of the character or a part of the background e.g. by k-means clustering.

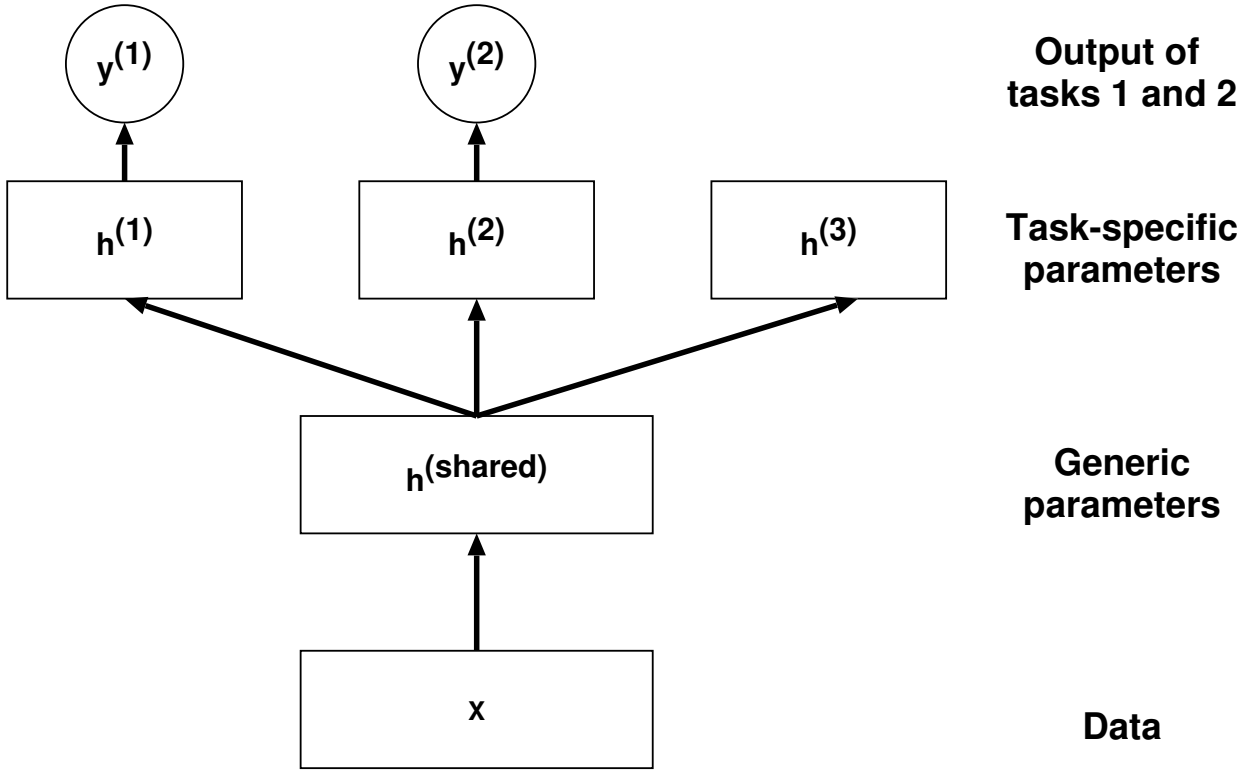
Further References for Semi-Supervised Learning

- An introduction and an overview of the state-of-the-art in semi-supervised learning is given in [Chapelle et al., 2006]. Additional references can be found in the technical report of [Zhu, 2008].
- We distinguish the discriminative model $P(y \mid \mathbf{x})$ and the generative models $P(\mathbf{x})$ and $P(\mathbf{x}, y)$.
- One can construct models in which a generative model shares parameters with a discriminative model. Bishop and Lasserre [Bishop and Lasserre, 2007] show how the results of generative and discriminative models can be combined by factoring the Likelihood function of a joint model with parameter sets θ (for the discriminative model) and $\tilde{\theta}$ (for the generative model) and how a tradeoff between the objective functions $-\log P(y \mid \mathbf{x})$ $-\log P(\mathbf{x})$ and $-\log P(\mathbf{x}, y)$ can be found. In the combined model, the generative model of the distribution of $\tilde{\theta}$ for expresses a prior distribution on the parameters θ of the discriminative model $P(y \mid \mathbf{x})$.

  See also [Lasserre et al., 2006] and [Larochelle and Bengio, 2008].
- Hinton and Salakhutdinov [Hinton and Salakhutdinov, 2008] describe a method for learning the kernel function of a kernel machine used for regression for which the use of unlabeled data for modeling $P(\mathbf{x})$ improves $P(y \mid \mathbf{x})$ significantly.

(Not relevant for the exam. For the curious!)

Multitask Learning



Train a neural network which computes the following 8-bit boolean functions:

1. Task 1: `B1 OR Parity(B2-B8)`
2. Task 2: `NOT(B1) AND Parity(B2-B8)`

Multitask Learning

Definition

Multitask learning is a way to improve generalization by pooling the examples arising out of several related tasks [Caruana, 1993].

- Multitask learning works if a part of the model is shared between the tasks. In the example, e.g. the computation of the 7-bit `Parity` function is shared between the tasks.
- The reason is that the shared parts of the model benefit from the examples of all tasks whereas the task-specific parts of the model benefit only from the task relevant examples.

  For a formal proof of this claim, see [Baxter, 1995].
- The assumption is that among the factors that explain the variations observed in the data associated with the different tasks, some are shared across two or more tasks.

Bagging and Other Ensemble Methods

Definition

Ensemble methods **combine** the **results** of a set of machine learning models by voting (for classification type problems) or by some kind of averaging (for regression type problems).

- Ensemble methods are meta-level methods which improve almost all machine learning models (NNs, decision trees, kNN-methods, ...)
- To work, each single machine learning model must be better than random guessing. The models should make different errors.
- Ensemble learning models work at least as good as the best single machine learning model in the set and often better.
- Ensemble methods require more computing ressources both in training and actual prediction.
- Ensemble methods differ by the mathematical assumptions controlling the variations of the machine learning models in the ensemble.

Bagging

1. Generate k different datasets of size m by **resampling with replacement** from the original dataset of size m. The proportion of missing examples is $\lim_{m\to\infty}(1 - 1/m)^m = e^{-1} = 0.3678794$.
2. Train a neural network algorithm on each data set. Note that e.g. the random initialization of weights also contributes to the variation.
3. Decision rule for result:
   - Classifier: Majority vote of k votes.
   - Prediction: Average of k results.

Bagging. Why Does It Work?

- Consider k regression models with the error ϵ distributed according to the multivariate normal distribution $N(0, \Sigma)$.

  The expectation of the error variance: $v = E(\epsilon_i^2)$.

  The expectation of the error covariances: $c = E(\epsilon_i \epsilon_j)$.

- The expected squared error of the ensemble predictor is

$$E\left(\left[\frac{1}{k}\sum_i \epsilon_i\right]^2\right) = \frac{1}{k^2} E\left(\sum_i [\underbrace{\epsilon_i^2}_{\text{uncorrelated error}} + \underbrace{\sum_{i \neq j} \epsilon_i \epsilon_j}_{\text{correlated error}}]\right)$$

$$E\left(\left[\frac{1}{k}\sum_i \epsilon_i\right]^2\right) = \frac{1}{k} v + \frac{k-1}{k} c$$

- Errors perfectly correlated: $c = v$. Model averaging does not help.
- Errors perfectly uncorrelated: $c = 0$. Ensemble error is $\frac{1}{k} v$.

Other Ensemble Methods

- Boosting: Boosting incrementally builds an ensemble of models by first training an algorithm on a training data set consisting of observations with equal weight. In the next steps, further algorithms are trained with observations weighted proportional to the error in the previous step. See [Freund and Shapire, 1999].
- Bayesian Model Averaging (BMA) and Bayesian Model Combination (BMC): Predictions are based an averaging the results of several models with weights given by the posterior probability of each model (BMA) (or each model combination (BMC)) given the data. (See [Hoeting et al., 1999] and [Hoeting et al., 2000])
- Stacking: Training a learning algorithm to combine the predictions pf other learning algorithms. E.g. [Wolpert and Macready, 1999].
- Machine learning contests are won by ensemble methods. E.g. [Koren, 2009].

(Not relevant for the exam. For the curious!)

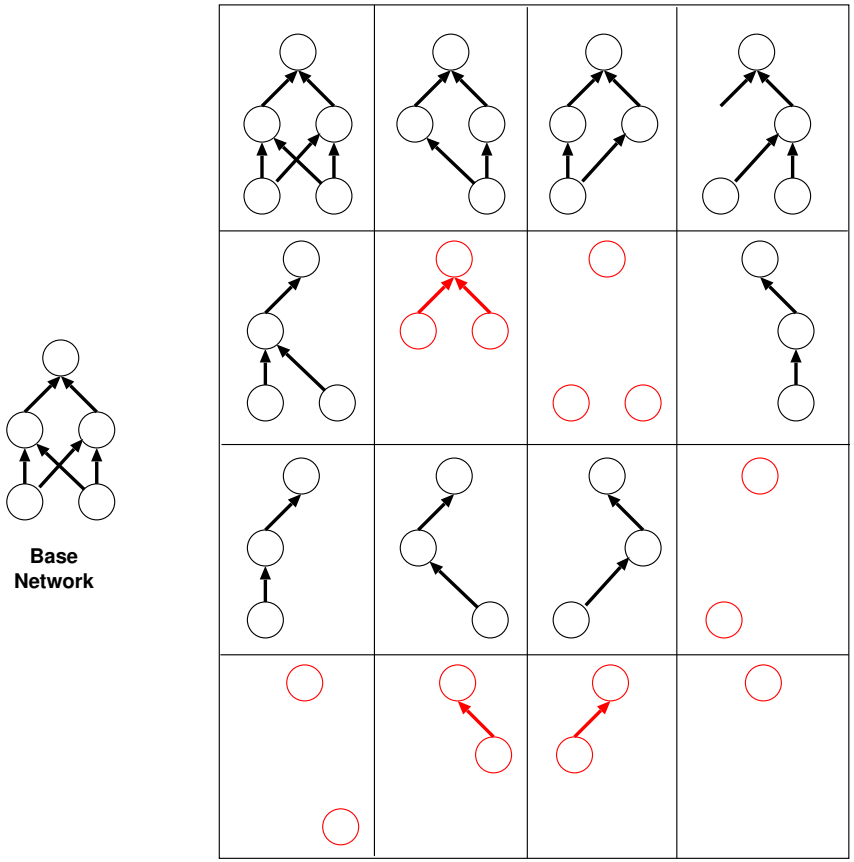
Dropout

Definition

- Dropout randomly removes units from neural networks in each training step.
- This can be implemented by generating random 0/1 mask vectors with which the output vectors ($\mathbf{h}^{k-1}, k \in 1, \ldots, l$) of all neurons is multiplied elementwise at the beginning of each training step. Forward propagation, back-propagation, and the update of weights is run as usual. The probability that a mask variable equals 1 is a hyperparameter.
- Let the mask vector μ specify which units to include and let $J(\theta, \mu)$ be the cost of the model defined by the model parameter vector θ and the mask vector μ.
- Dropout training minimizes $E_\mu(J(\theta, \mu)$. Since there exist 2^n different masks of size n, we obtain n unbiased estimate of its gradient by sampling values of μ.

Dropout provides a computationally inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

Dropout. Ensemble of Subnetworks



Ensemble of Subnetworks (Red: Non functioning networks)

The problem of non-functioning networks (no inputs or no path from input nodes to output nodes) reduces with the size of the layers of the networks. Often, the probability of ones in the mask can be parametrized for each layer separately before training starts.

Dropout versus Bagging

Dropout:

- Models share parameters and inherit different subsets of parameters from parent model.
- Represents an exponential number of models with limited memory.
- Most models are not explicitly trained at all. A tiny fraction is trained for a single step.
- The prediction is $\sum_{\mu} P(\mu)P(y \mid \mathbf{x}, \mu)$.
- Sampling required! Often 10-20 masks are sufficient for good performance.

Bagging:

- Models are all independent.
- Each model is trained to convergence on its data set.
- Each model i produces $P^i(y \mid \mathbf{x})$.
- The prediction is $\frac{1}{k} \sum_{i=1}^{k} P^i(y \mid \mathbf{x})$.

Advantages of Dropout

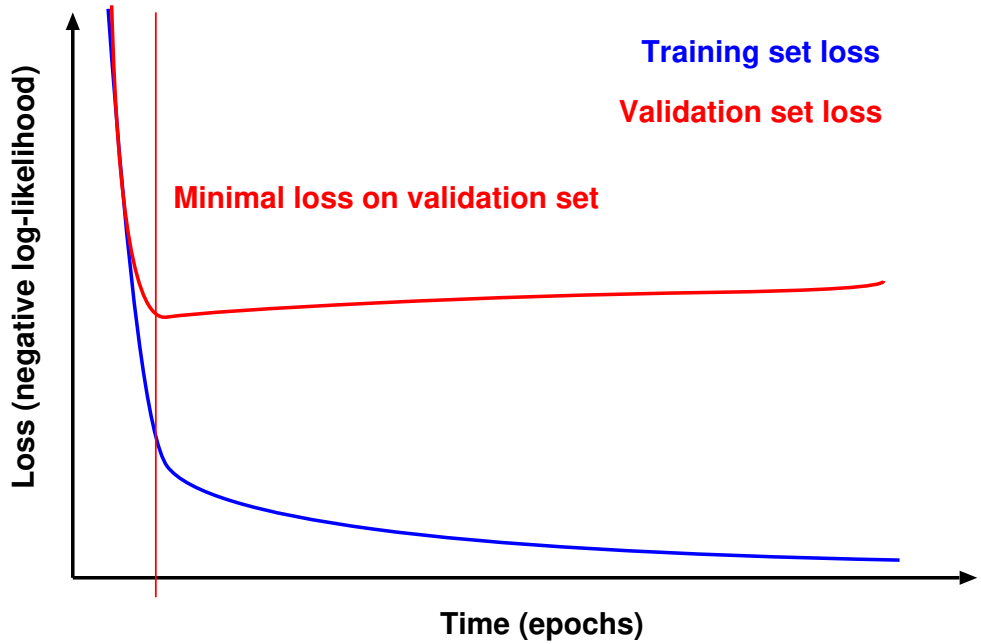
- More effectice regularizer than e.g. weight decay ...
- Computationally **very** cheap!
- Does not significantly limit the type of model and the type of training procedure used.
- Regularizes hidden units to be not only a good feature but a feature good in many contexts.
- Noise applied to hidden units destroys features. This implies that features must be learned redundantly, either by redundant encoding of the destroyed features or by detecting properties by using other than the destroyed features.

Further Results on Dropout

- Seminal papers: Dropout [Hinton et al., 2012] and [Srivastava et al., 2014].
- The weight-scaling inference heuristic to approximate $P_{ensemble}$ by evaluating $P(y \mid \mathbf{x})$ from a single model by multiplying the weights out of unit i with probability of including the unit in the model.
- Use geometric mean instead of arithmetic mean for aggregation of votes [Warde-Farley et al., 2014].
- For integrating dropout in restricted Boltzmann machines see [Srivastava et al., 2014] and into recurrent neural networks see e.g. [Bayer and Osendorfer, 2015].

(Not relevant for the exam. For the curious!)

Early Stopping. Intuition



The sketch above shows the usually observed development of training and validation error of a large model with increasing number of iterations over the data set:

- The training loss decreases over time.
- The validation loss is asymmetrically U-shaped:
  Overfitting starts as soon as the validation loss rises again.

Conclusion: We should stop training as soon as the validation set loss starts to rise.

Early Stopping Algorithm (Pseudocode)

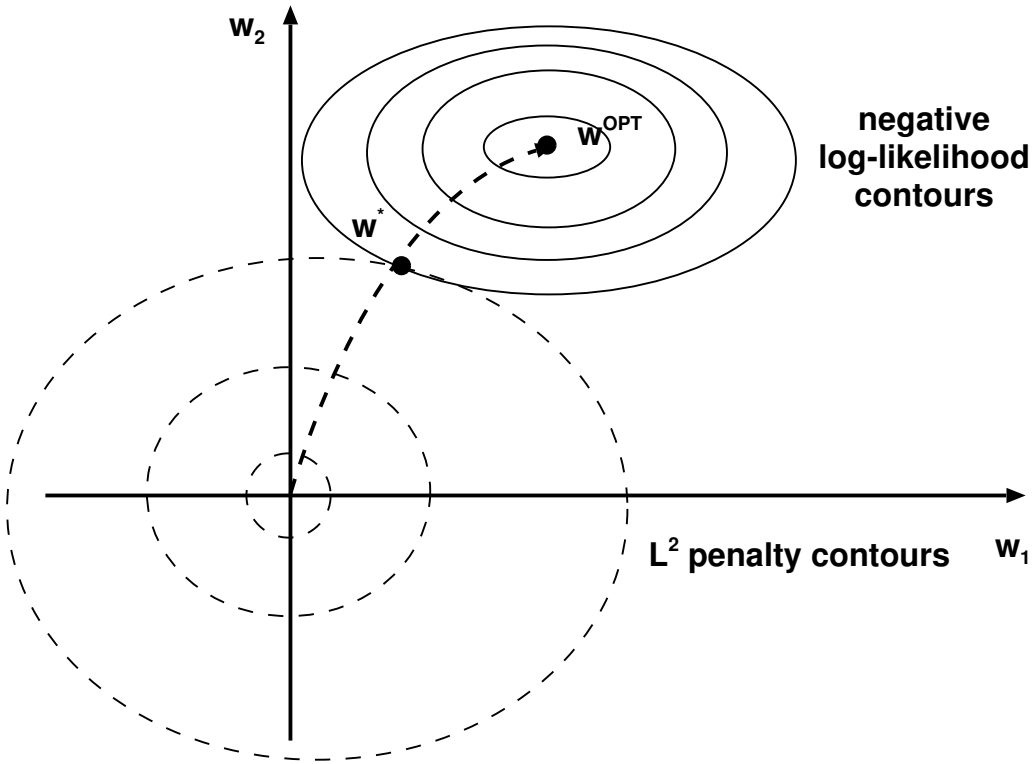
```
1  EarlyStopping(n, p)
2  //* n: number of training steps between evaluations
3  //* p: patience (number of time of observing worsening
       validation error v)
4  //* theta0: start parameters
5  thetaStar <- theta <- theta0
6  iStar <- i <- j <- 0$ //* counter of steps: i, counter of
       worsening v: j
7  while (j<p) do {
8     theta<-trainingAlgorithm(theta, n) //* train for n
          steps
9     i<-i+n
10    vNew<-ValidationSetError(theta)
11    if (vNew<v) then { //* Update j, v, thetaStar, iStar
12    j<-0
13    thetaStar<-theta
14    iStar<-i
15    v<-vNew
16      }
17    else {j<-j+1} // end if
18 } //* end while
19 return(thetaStar, iStar) //* best parameters and steps
```

Early Stopping

Early stopping combines information from

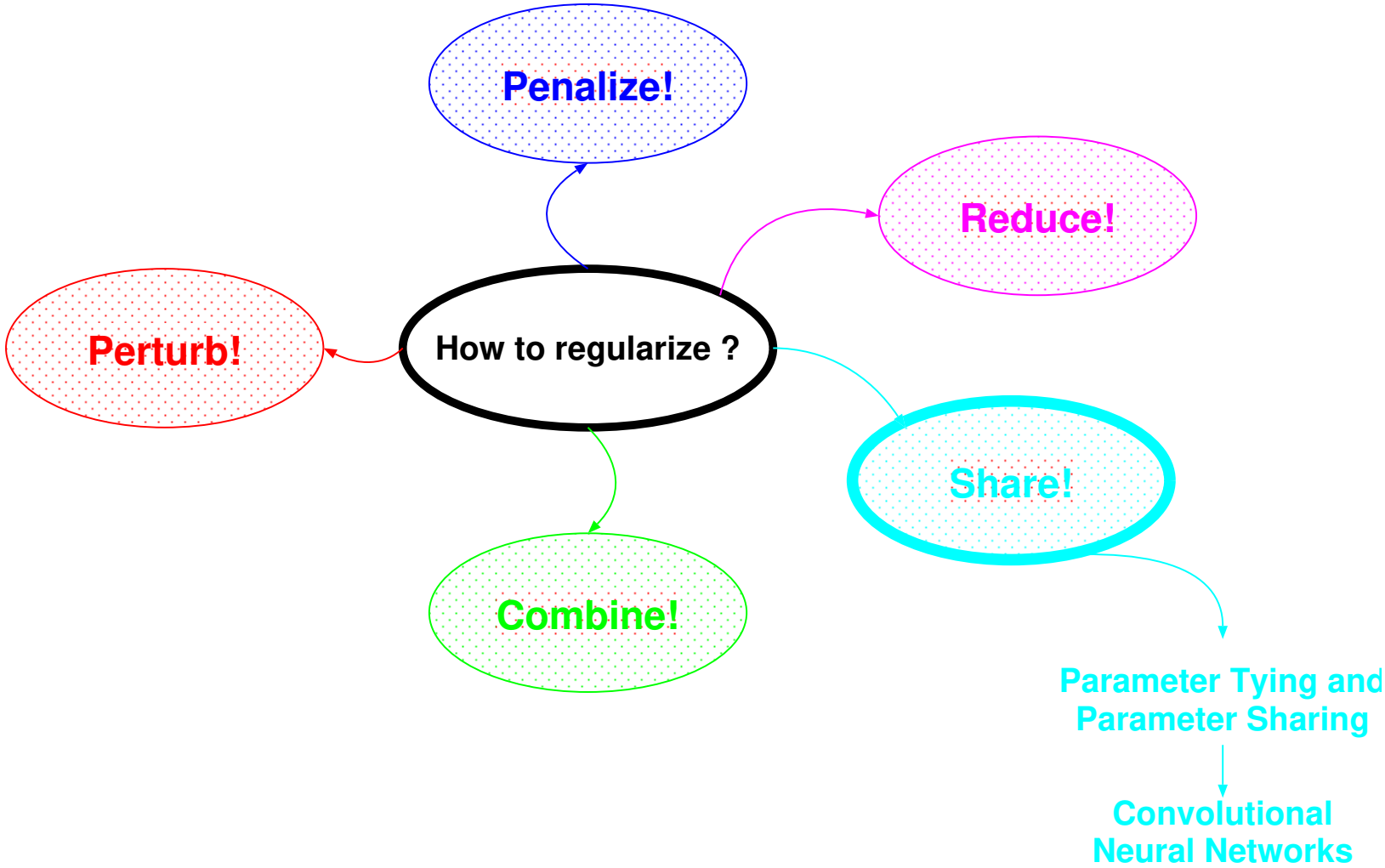
1. the **training error** (for parameter improvement) and
2. the **validation error** (for determining the stopping time).

Early Stopping as a Regularizer



- The trajectory of the early stopping algorithm stops at a point which corresponds to a minimum of a L^2-regularized objective.
- The advantage of the early stopping algorithm is that it automatically determines the proper amount of regularization, whereas finding the the proper α requires many training experiments.

Share!


Penalize!
Reduce!
Perturb!
How to regularize ?
Share!
Combine!
Parameter Tying and
Parameter Sharing
Convolutional
Neural Networks

Parameter Tying

- Often we want to model a dependency between two models for a similar classification task (e.g. number recognition):

  We have model A: $\mathbf{w}^A$ and model B: $\mathbf{w}^B$.

  We want to express that $\forall i : w_i^A$"near"w_i^B
- By the method of Lagrange we can express this as a parameter norm penalty $\Omega(\mathbf{w}^A, \mathbf{w}^B)$ (e.g. as a L_2-penalty term):

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

- Note that we can adapt this to penalize only the weights of the first layer:

  By this we express our belief that the activations of the first layer encode a set of features which are present in the input vector (e.g. for number recognition: a vertical bar, a horizontal line, a single circle, two circles, ...).
- See [Lasserre et al., 2006].

Parameter Sharing

Example: Recognition of the numbers 0, 1, ..., 9 from 64×64 colour images coded on a RGB-palette as a three element vector with values in 0 to 255.
"Naive" learning requires 12288 input nodes.
Observation: Reading usually requires only a discrimination between pixels in the foreground (which usually form the numbers) and pixels in the background.
So a first layer which maps the vector triple into $0/1$ would reduce the input vector to the feature recognition layer to 4096 input nodes.

- To learn the discrimination between foreground and background pixels, we train a single network on the pixels of the data set:
- The first layer of this network consists of a single layer network with three input nodes, perhaps three hidden nodes and a single output node.
- In the composed network, we use 4096 networks of this type **sharing the same parameters** as a preprocessing layer.
- See *Convolutional Networks*, [Goodfellow et al., 2016c].

(Not relevant for the exam. For the curious!)

Parameter Sharing: Convolutional Neural Networks

- Computer vision is the most popular application of convolutional neural networks (CNNs) which use parameter sharing.
- In natural images, many (statistical) properties exist which are **invariant** e.g. to translation.

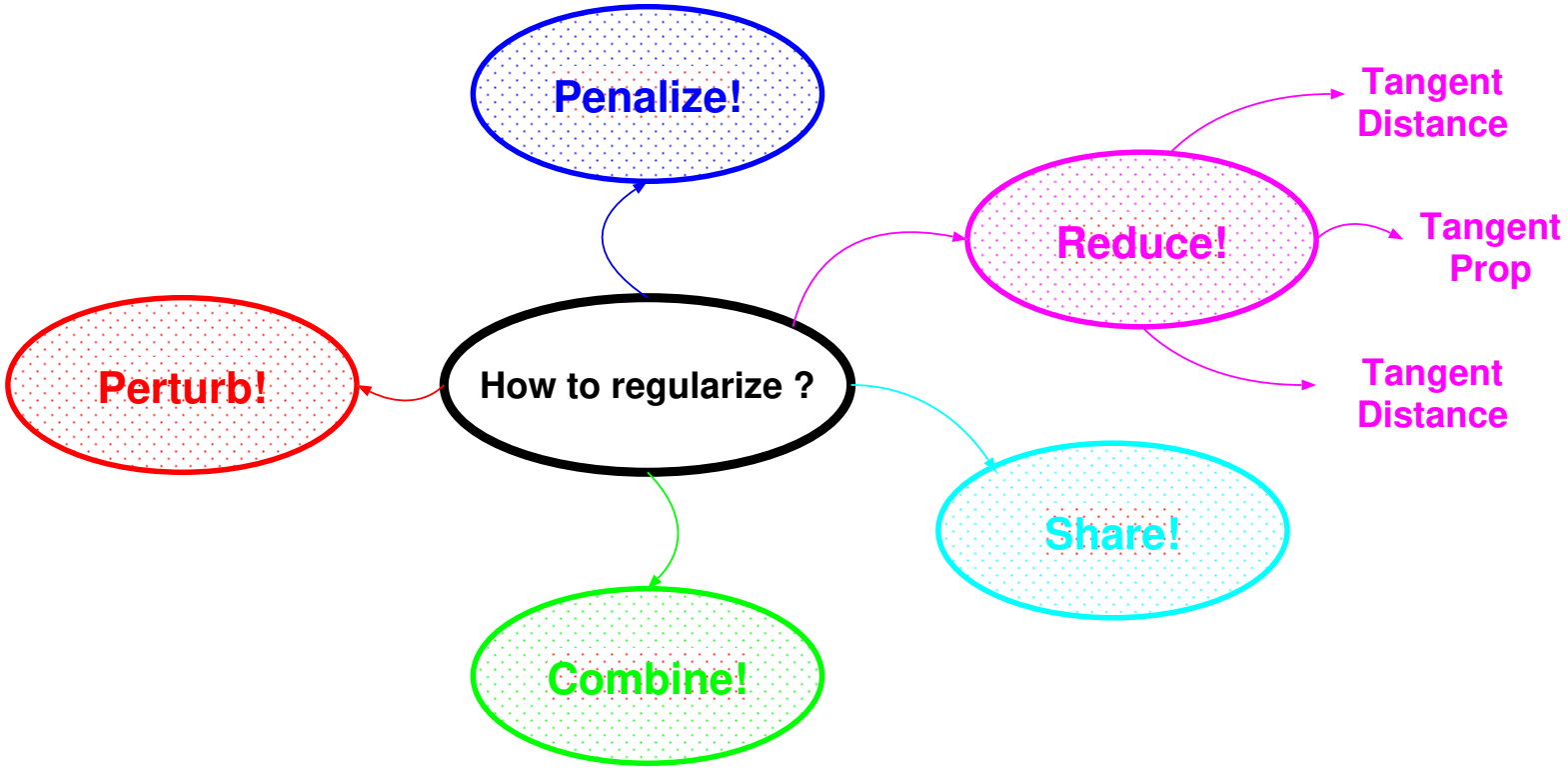
  The photo of a certain person remains this person even if the person is shifted a pixel to the right.
- CNNs use this property by sharing parameters between image locations: The same feature (a hidden unit with the same weights) is computed over different locations in the input.

  We find the person with the same person detector independent of the actual position of the person in the photo.

Dropout

Dropout **shares** the parameters of the subnetworks it trains.

Reduce!


Penalize!
Tangent Distance
Reduce!
Tangent Prop
Tangent Distance
Perturb!
How to regularize ?
Share!
Combine!

The Low Dimensional Manifold Assumption

Definition

High-dimensional examples lie (roughly) on a low dimensional manifold (surface/subspace) (The Manifold Assumption (Informal)).

Example:

- A mountain road is a 1-D manifold embedded in the 3-D GPS coordinate system of our world.
- The driving distances between 2 postal addresses on the road are an intrinsic property of the road.

The Concept of a Manifold

Definition

Let U and V open sets in R^n. A differentiable function $h : U \to V$ with a differentiable inverse $h^{-1} : V \to U$ will be called a **diffeomorphism** [Spivak, 1965, p. 109].

Definition

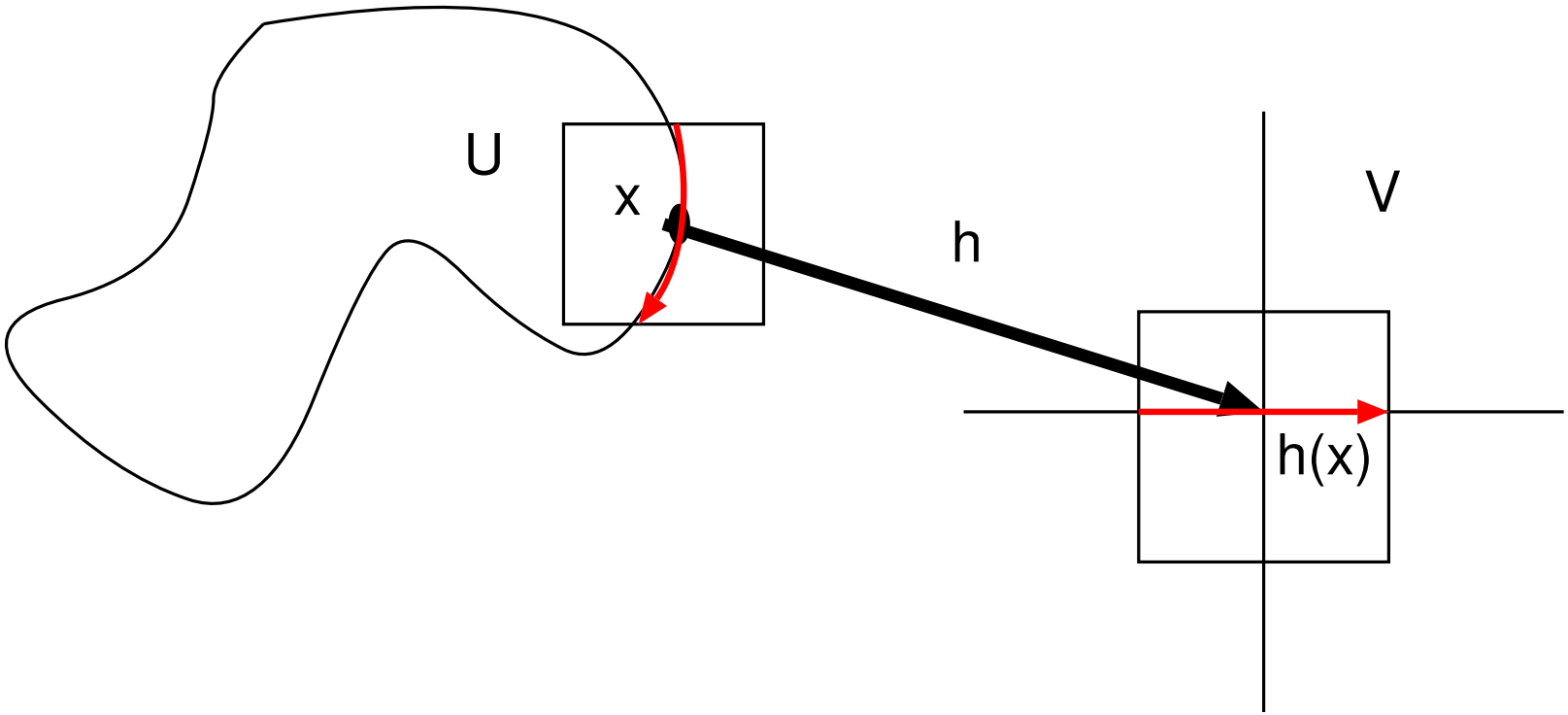
A subset M of R^n is called a k-dimensional manifold in R^n if for every point $\mathbf{x} \in M$ the following condition holds:
There is an open set U containing $\mathbf{x}$, an open set $V \subset R^n$, and a diffeomorphism $h : U \to V$ such that

$$h(U \cap M) = V \cap (R^k \times \{0\}) = \{y \in V : y^{k+1} = \cdots = y^n = 0\}$$

[Spivak, 1965, p. 109].

Examples: A point in R^n is a 0-dimensional manifold, and an open subset of R^n is an n-dimensional manifold.

A 1-Dimensional Manifold



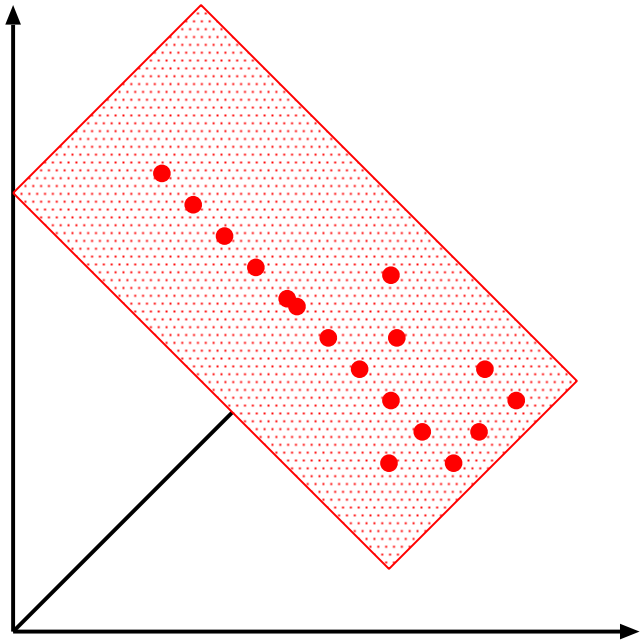
- From any given point the manifold locally appears to be a Euclidean space.
- We experience the world as a 2-D plane, but it is in fact a spherical manifold in 3-D space.
- In machine learning, the dimensionality of the manifold may vary from point to point: 8 is a 1-D manifold except at the intersection point which is a 2-D manifold.

In Favor of the Manifold Hypothesis

- The probability distribution over images, texts, strings, and sounds that occur in real life is highly concentrated. Uniform noise essentially never resembles structured input from these domains.
- We have a good informal understanding of such neighborhoods and transformations. E.g. in the case of images, we formally know scaling, rotation, and linear translation operations ...
- Computational experiments support the hypothesis for a large class of datasets in AI. See e.g. [Cayton, 2005].
- In practice, we extract the manifold coordinates in a first step and then perform the machine learning task on the data represented in the low-dimensional manifold coordinates.

h Is Linear: Principal Component Analysis (PCA)

- We consider a set of points in a 3-dimensional space which are on or near a 2-dimensional plane.
- PCA will return two vectors that span the plane and a third vector which is orthogonal to the plane.
- The two vector which span the two dimensional plane will have a positive weight, the last will have a weight of zero (all points exactly on the plane).
- PCA finds a basis for the linear subspace and it allows to disregard irrelevant features (the third vector).



Principal Component Analysis (PCA)

- The D-dimensional data set: $\mathbf{X} \in R^{n \times D}$.
- We are looking for the d-dimensional linear subspace of R^D along which the data has maximum variance:

$$\max_{\mathbf{V}} \text{var}(\mathbf{XV})$$

  where $\mathbf{V}$ is an orthogonal $D \times d$ matrix.
- The columns of $\mathbf{V}$ give the d dimensions that we project the data onto.
- The d-dimensional PCA solution is $\mathbf{XV}$ where the d columns of $\mathbf{V}$ are the *top* d eigenvectors of the eigenvector equation

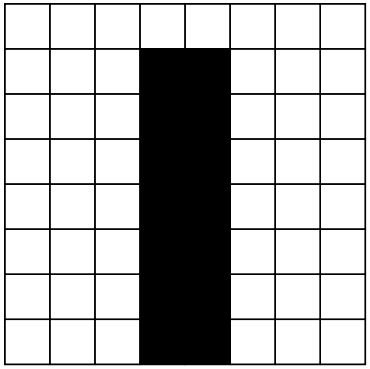
$$\lambda_i = \mathbf{v}_i^T \mathbf{X}^T \mathbf{X} \mathbf{v}_i$$

  top means sorted by $\mid \lambda_i \mid$.
- Note that $\mathbf{X}$ must be a standardized data set with mean 0 and variance 1. For the reformulation of $\max_{||\mathbf{v}||=1} \text{var}(\mathbf{Xv})$ as a form of Rayleigh's quotient $\max_{||\mathbf{v}||=1} \mathbf{v}^T \mathbf{X}^T \mathbf{X} \mathbf{v}$, see [Cayton, 2005].
- R-Code:
  `pca1 <- prcomp(X, scale=TRUE)`

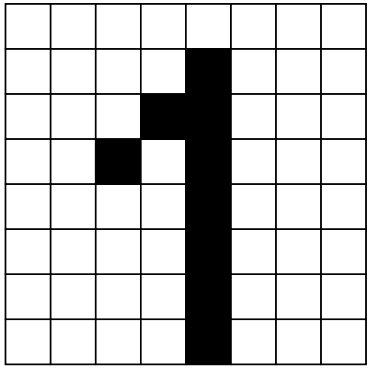
Motivation for Tangent Distance

- The simplest classification systems used in practice simply store all the examples together with their labels in memory. This type of systems needs:
  1. a distance measure to compare inputs to prototypes,
  2. an output function to produce an output by combing the labels of the prototype, and
  3. and a storage scheme to build the set of prototypes.
- Using a simple distance measure (Euclidean distance) between a vector of raw inputs and a very large set of prototypes usually leads to inefficient pattern recognition systems.

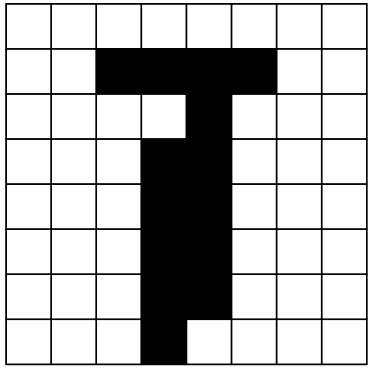
Why simple distance measures do not work



Pattern: 1



Prototype: 1



Prototype: 7

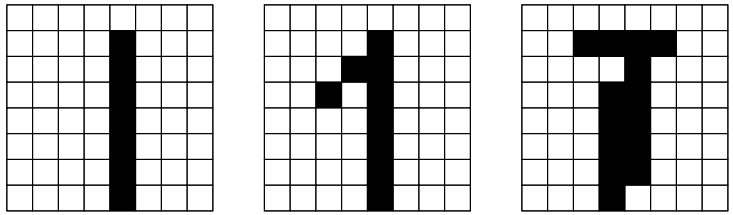
EuclideanDistance(Pattern 1, Prototype 1) = 7
EuclideanDistance(Pattern 1, Prototype 7) = 4

- Simple distance measures fail, because they do not handle distortions of the pattern well. E.g. variations in line thickness.
- For character/number recognition problems, we require representations which are invariant with respect to position, size, small rotations, distortions, changes in line thickness, small variations in colour, ...

Two Solution Approaches

1. Design and implement a set of feature extractors which are appropriate for the problem.

   In the example: Thinning and horicontally shifting the pattern will work.



Thinned Pattern: 1 Prototype: 1 Prototype: 7

EuclideanDistance(Thinned Pattern 1, Prototype 1) = 2
EuclideanDistance(Thinned Pattern 1, Prototype 7) = 9

   The construction of an appropriate set of feature extractors which work for an application is a major bottleneck in machine learning.

2. Construct an **invariant distance measure**!

Construction of an Invariant Distance Measure

1. Define a transformation group Γ of operations γ on the representation of a prototype P. This defines a compact representation of a potentially infinite set of equivalent representations of P.
2. The invariant distance is

$$\inf_{\gamma \in \Gamma} d(X, \gamma P) \tag{26}$$

   where X is a pattern, and d a distance between the representation of the pattern and the representation of the transformed prototype.
3. The effect of using an invariant distance measure is that the distance between pattern and prototype is not affected by irrelevant transformation (i.e. the transformations in Γ).
4. However, the design of the transformation group requires a priori knowledge and solving equation (26) may become very expensive.
5. In machine learning γP is called a **deformable prototype**.

Construction of an Invariant Distance Measure for Character Recognition

- For character recognition, candidates for transformations are e.g. rotation, scaling, and vertical and horizontal shifts. All of these operations are continuous linear matrix operators. See e.g. [Meyer, 2000, pp. 330-334].
- Consider 16×16 images (patterns and prototypes) represented as 256 dimensional pixel vectors $\mathbf{x}$. Transforming a pattern according to a transformation γ which depends on a single parameter α (e.g. the angle of rotation), the set of all transformed patterns

$$S_P = \{\mathbf{x} \mid \exists \alpha \text{ for which } x = \gamma(P, \alpha)\}$$

  is a one-dimensional manifold (curve) in the 256 dimensional vector space of the inputs. We assume that γ is differentiable with respect to P and α, and that $\gamma(P, 0) = P$.
  If the set of transformations is parametrized by n parameters, the intrinsic dimension of the manifold S_P is at most n.
- Matching a deformable prototype with an incoming pattern means finding the point on the surface of the manifold which has the minimum distance from the incoming pattern.
  This is a **highly nonlinear optimization problem** in pixel space.

Tangent Plane

- The main idea to eliminate the nonlinearity of this optimization problem is to approximate the surface of the manifold S_P at the point P by the Taylor series expansion of γ around $\alpha = 0$:

$$\gamma(P, \alpha) = \gamma(P, 0) + \alpha \frac{\partial \gamma(P, \alpha)}{\partial \alpha} + R(\alpha^2) \approx P + \alpha T$$

  where $T = \frac{\partial \gamma(P, \alpha)}{\partial \alpha}$ is the tangent vector.
  For small α, the approximation is usually very good.

- The tangent hyperplane L_P to S_P at P is given by the columns of the following matrix:

$$L_P = \left. \frac{\partial \gamma(P, \alpha)}{\partial \alpha} \right|_{\alpha=0} = \left[\frac{\partial \gamma(P, \alpha)}{\partial \alpha_1}, \ldots \frac{\partial \gamma(P, \alpha)}{\partial \alpha_m} \right]_{\alpha=0}$$

Tangent Distance

Let E and P be two patterns to be compared. We replace the Euclidean distance between the patterns by the locally linear tangent planes of the manifolds S_P and S_E and define the tangent distance by

$$D(E,P) = \min_{\mathbf{x}\in T_E, \mathbf{y}\in T_P} || x - y ||^2$$

The tangent planes T_E and T_P are

$$\begin{aligned} E'(\alpha_E) &= E + L_E \times \alpha_E \\ P'(\alpha_E) &= P + L_P \times \alpha_P \end{aligned}$$

Computing the tangent distance

$$D(E,P) = \min_{\alpha_E, \alpha P} || E'(\alpha_E) - P'(\alpha_P) ||^2$$

requires solving a linear least squares problem. For the details, see [Simard et al., 1998].

Smoothing the Input Space

- The tangent vectors are the Lie derivatives of the transformation function $\gamma(P, \alpha)$.

$$L_P = \left.\frac{\partial \gamma(P, \alpha)}{\partial \alpha}\right|_{\alpha=0} = \lim_{\epsilon \to 0} \frac{\partial \gamma(P, \epsilon) - \partial \gamma(P, 0)}{\epsilon}$$

- $\gamma(P, \epsilon)$ must be differentiable. If P is discrete, (remember, P is a pixel image), we need a smoothing interpolation function C_ρ for preprocessing P so that $\gamma(C_\rho(P), \alpha)$ is differentiable with regard to $C_\rho(P)$.

  For example, $C_\rho(P)$ can be a convolution of P with a Gaussian function with standard deviation ρ. The standard deviation ρ has the role of a smoothing factor which controls the locality of the invariance.

- In general, the best smoothing factor is the maximum smoothing which does not blur the image. E.g. for handwritten character recognition on 16×16 pixel images, a standard deviation of 1 yielded the best results.

Tangent Prop(agation)

Alternatively (if we want a NN instead of storing many prototypes),
we can modify the backpropagation algorithm to make the network invariant with regard to a Lie transformation group.

- The modification requires a regularizing penalty term:

$$\Omega(f) = \lambda \sum_i \left((\nabla_{\mathbf{x}} f(\mathbf{x}))^T \mathbf{v}^{(i)} \right)$$

  where $\mathbf{v}^{(i)}$ is $L_{\mathbf{x}}$ (the manifold tangent vectors at $\mathbf{x}$) and λ a hyperparameter scaling the regularizer.

- Local invariance requires that the gradient $\nabla_{\mathbf{x}}$ at $\mathbf{x}$ is orthogonal to the known manifold tangent vectors $\mathbf{v}^{(i)}$ at $\mathbf{x}$.

  This implies that the directional derivative of f at $\mathbf{x}$ is small in the directions $\mathbf{v}^{(i)}$. This is enforced by adding the penalty term $\Omega(f)$ above.

The Manifold Tangent Classifier

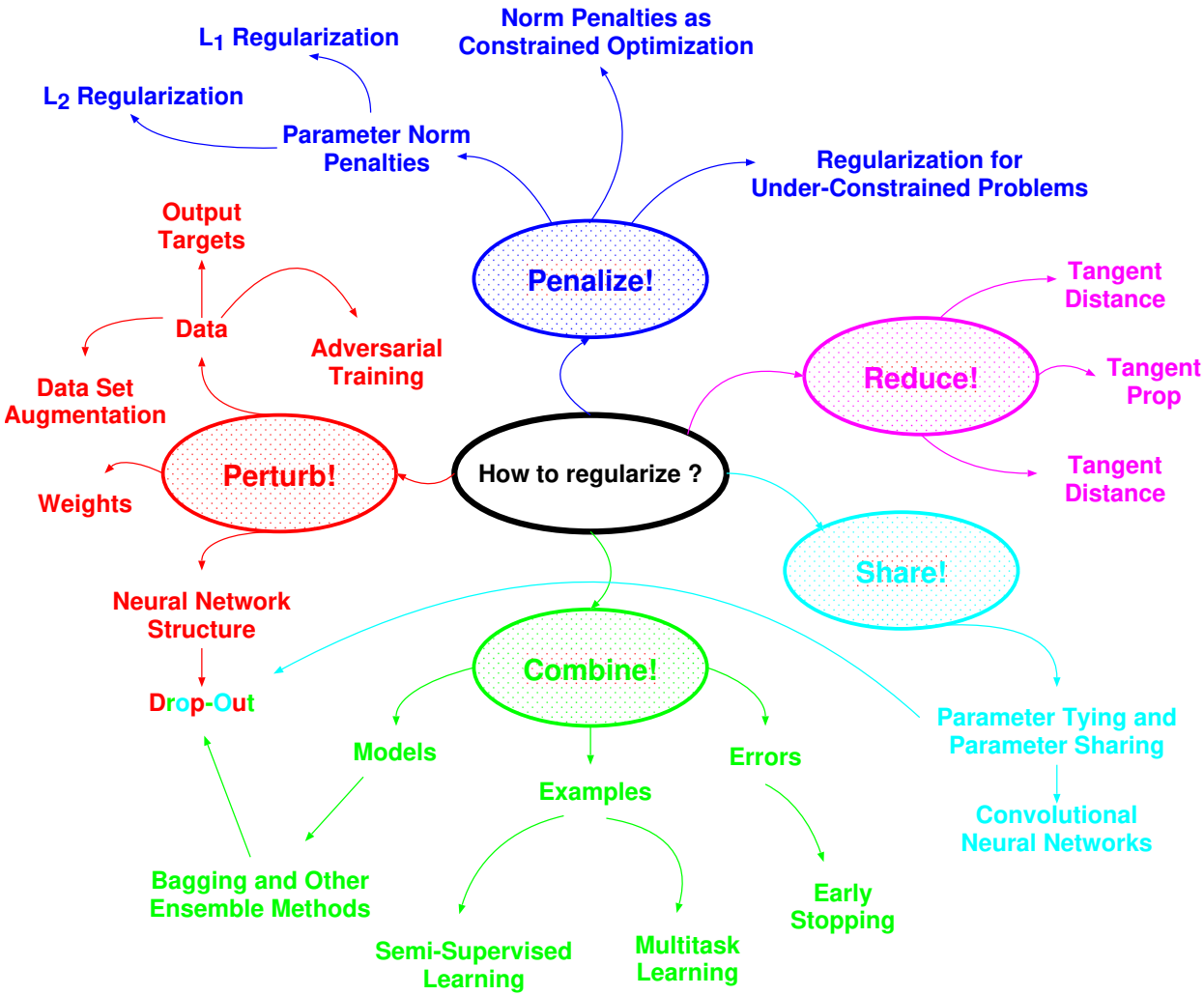
The manifold tangent classifier eliminates the need to know the tangent vectors a priori. The algorithm proposed for this is simple [Rifai et al., 2011]:

1. Use an auto-encoder to learn the manifold structure (the tangent vectors) by unsupervised learning.
2. Use the tangent vectors to regularize a neural network classifier by the penalty term $\Omega(f)$ given for tangent prop.

Tangent Distance, Tangent Prop and Manifold Tangent Classifier

- The tangent distance algorithm [Simard et al., 1993], [Simard et al., 1998].
- The tangent prop(agation) algorithm [Simard et al., 1991], [Simard et al., 1998].
- The manifold tangent classifier [Rifai et al., 2011].

Summary: A Map of Regularization Methods



Recommended Reading I

- This lecture follows mainly chapter 7 *Regularization* in [Goodfellow et al., 2016b].
- For an introduction of the terminology on training machine learning models we refer to [Goodfellow et al., 2016a] and [Koller and Friedman, 2009].
- However, in addition we strongly recommend *Verification and Validation in Scientific Computing* [Oberkampf and Roy, 2010]. The book contains a complete description of the verification and validation processes needed for deploying critical systems (in aerospace, defence, engineering, ...).
- The sources for **No Free Lunch Theorems** are [Wolpert and Macready, 1997] and [Wolpert, 1996].

  Note that properly formulazing the conditons for these theorems to hold requires care as the paper [Wolpert, 1996] shows which discusses the conditions of existence of a priori distictions between learning algorithms.
- Regularizion methods:

  1. Double Backpropagation: [Drucker and Le Cun, 1992]

References I

Baxter, J. (1995).

Learning internal representations.

In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, COLT '95, pages 311 – 320, New York, NY, USA. ACM.

Bayer, J. and Osendorfer, C. (2015).

Learning stochastic recurrent networks.

arXiv, 2015.

Belkin, M. and Niyogi, P. (2002).

Laplacian eigenmaps and spectral techniques for embedding and clustering.

In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 585 – 591. MIT Press.

Bergstra, J. (2011).

Incorporating Complex Cells into Neural Networks for Pattern Classification.

PhD thesis, Departement d'informatiqures et de recherche operationnelle, Universite de Montreal, Montreal.

References II

Bishop, C. M. and Lasserre, J. (2007).

Generative or discriminative? getting the best of both worlds.

In Bernardo, J. M., Bayarri, M. J., Berger, J. O., Dawid, A. P., Heckerman, D., Smith, A. F. M., and Mike, W., editors, *Bayesian Statistics 8. Proceedings of the Eight Valencia International Meeting*, chapter 1, pages 3 – 24. Oxford University Press, Oxford.

Caruana, R. A. (1993).

Multitask connectionist learning.

In *Proceedings of the 1993 Connectionist Models Summer School*, pages 372 – 379.

Cayton, L. (2005).

Algorithms for manifold learning.

Technical Report CS2008-0923, University of California at San Diego, San Diego.

Chapelle, O., Schölkopf, B., and Zien, A., editors (2006).

Semi-Supervised Learning.

MIT Press, Cambridge.

References III

Chapelle, O., Weston, J., and Schölkopf, B. (2002).

Cluster kernels for semi-supervised learning.

In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, pages 601 – 608, Cambridge, MA, USA. MIT Press.

Coates, A. and Ng, A. Y. (2011).

The importance of encoding versus training with sparse coding and vector quantization.

In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 921 – 928, Madison, WI, USA. Omnipress.

Drucker, H. and Le Cun, Y. (1992).

Improving generalization performance using double backpropagation.

IEEE Transactions on Neural Networks, 3(6):991 – 997.

Freund, Y. and Shapire, R. E. (1999).

A short introduction to boosting.

Journal of Japanese Society of Artificial Intelligence, 14(5):771 – 780.

References IV


Goodfellow, I., Bengio, Y., and Courville, A. (2016a).
Machine learning basics.
In *Deep Learning*, chapter 5, pages 95 – 160. MIT Press, Cambridge, MA.


Goodfellow, I., Bengio, Y., and Courville, A. (2016b).
Regularization for deep learning.
In *Deep Learning*, chapter 7, pages 221 – 266. MIT Press, Cambridge, MA.


Goodfellow, I., Bengio, Y., and Courville, A. (2016c).
Regularization for deep learning.
In *Convolutional Networks*, chapter 9, pages 321 – 361. MIT Press, Cambridge, MA.


Goodfellow, I. J., Le, Q. V., Saxe, A. M., Lee, H., and Ng, A. Y. (2009).
Measuring invariances in deep networks.
In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 646 – 654, Red Hook, NY, USA. Curran Associates Inc.

References V


Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015).
Explaining and harnessing adversarial examples.
arXiv, 2015:1 – 11.


Hinton, G. E. and Salakhutdinov, R. R. (2008).
Using deep belief nets to learn covariance kernels for gaussian processes.
In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 1249 – 1256. Curran Associates, Inc.


Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012).
Improving neural networks by preventing co-adaptation of feature detectors.
arXiv, 2012.


Hochreiter, S. and Schmidhuber, J. (1994).
Simplifying neural nets by discovering flat minima.
In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, pages 529 – 536, Cambridge, MA, USA. MIT Press.

References VI


Hoerl, A. E. and Kennard, R. W. (1988).
Ridge regression.
In Kotz, S., Johnson, N. L., and Read, C. B., editors, *Encyclopedia of Statistical Sciences (Vol. 8). Regressograms To St. Petersburg Paradox, The*, volume 8 of *Encyclopedia of Statistical Sciences*, chapter 129, pages 129 – 136. Wiley, New York.


Hoeting, J. A., Madigan, D., Raftery, A. E., and Volinsky, C. T. (1999).
Bayesian model averaging. a tutorial.
Statistical Science, 14(4):382 – 401.


Hoeting, J. A., Madigan, D., Raftery, A. E., and Volinsky, C. T. (2000).
Correction to “bayesian model averaging: A tutorial”.
Statistical Science, 15(3):193 – 195.


Jaitly, N. and Hinton, G. E. (2013).
Vocal tract length perturbation (vtlp) improves speech recognition.
In *Proceedings of the 30th International Conference on International Conference on Machine Learning – Volume 28*, ICML’13, pages 1 – 5, Atlanta, GA, USA. JMLR.org.

References VII

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013).

An Introduction to Statistical Learning, volume 103 of *Springer Texts in Statistics*.

Springer New York.

King, P. (2005).

William of ockham: Summa logicae.

In John, S., editor, *Central Works of Philosophy: Ancient and Medieval*, volume 1, pages 243 – 274. Acumen, Chesham.

Koller, D. and Friedman, N. (2009).

Learning graphical models. overview.

In *Probabilistic Graphical Models*, chapter 16, pages 697 – 715. MIT Press, Cambridge.

Kolter, Z. and Madry, A. (2018).

Adversarial Robustness. Theory and Practice. NeurIPS 2018 Tutorial.

References VIII

Koren, Y. (2009).

The bellkor solution to the netflix grand prize.

Larochelle, H. and Bengio, Y. (2008).

Classification using discriminative restricted boltzmann machines.

In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, ICML-08, pages 536 – 543, New York, NY, USA. ACM.

Lasserre, J. A., Bishop, C. M., and Minka, T. P. (2006).

Principled hybrids of generative and discriminative models.

In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 87 – 94.

Lee, H., Ekanadham, C., and Ng, A. Y. (2008).

Sparse deep belief net model for visual area v2.

In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 873 – 880. Curran Associates, Inc.

References IX

Meyer, C. D. (2000).

Matrix Analysis and Applied Linear Algebra.

Society for Industrial and Applied Mathematics, Philadelphia.

Oberkampf, W. L. and Roy, C. J. (2010).

Verification and Validation in Scientific Computing.

Cambridge University Press, 1 edition.

Olshausen, B. A. and Field, D. J. (1996).

Emergence of simple-cell receptive field properties by learning a sparse code for natural images.

Nature, 381:607 – 609.

Pati, Y., Rezaiifar, R., and Krishnaprasad, P. (1993).

Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition.

In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pages 40 – 44.

References X

Poole, B., Sohl-Dickstein, J., and Ganguli, S. (2014).
Analyzing noise in autoencoders and deep networks.
CoRR, abs/1406.1831.

Rifai, S., Dauphin, Y. N., Vincent, P., Bengio, Y., and Muller, X. (2011).
The manifold tangent classifier.
In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2294 – 2302, Red Hook, NY, USA. Curran Associates Inc.

Schlesinger, S. (1979).
Terminology for model credibility.
Simulation, 32(3):103 – 104.

Sietsma, J. and Dow, R. J. F. (1991).
Creating artificial neural networks that generalize.
Neural Networks, 4(1):67 – 79.

References XI

Simard, P., LeCun, Y., and Denker, J. S. (1993).

Efficient pattern recognition using a new transformation distance.

In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 50 – 58. Morgan-Kaufmann.

Simard, P., LeCun, Y., Denker, J. S., and Victorri, B. (1998).

Transformation invariance in pattern recognition-tangent distance and tangent propagation.

In Orr, G. B. and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, chapter 12, pages 239 – 274. Springer Berlin Heidelberg, Berlin, Heidelberg.

Simard, P., Victorri, B., Le Cun, Y., and Denker, J. (1991).

Tangent prop. a formalism for specifying selected invariances in an adaptive network.

In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pages 895 – 903, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

References XII

Spivak, M. (1965).

Calculus on Manifolds.

W. A. Benjamin, Inc., New York.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).

Dropout. A simple way to prevent neural networks from overfitting.

Journal of Machine Learning Research, 15(56):1929 – 1958.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016).

Rethinking the inception architecture for computer vision.

In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818 – 2826, Los Alamitos, CA, USA. IEEE Computer Society.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014).

Intriguing properties of neural networks.

arXiv, 2014.

References XIII

Tang, Y. and Eliasmith, C. (2010).

Deep networks for robust visual recognition.

In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, ICML-2010, pages 1055 – 1062, Haifa, Israel. Omnipress.

Tibshirani, R. (1996).

Regression shrinkage and selection via the lasso.

Journal of the Royal Statistical Society. Series B (Methodological), 58(1):267 – 288.

Vapnik, V. N. (1995).

The Nature of Statistical Learning Theory.

Springer-Verlag, New York.

Vapnik, V. N. and Chervonenkis, A. Y. (1971).

On the uniform convergence of relative frequencies of events to their probabilities.

Theory of Probability & Its Applications, 16(2):264 – 280.

References XIV

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008).

Extracting and composing robust features with denoising autoencoders.

In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, ICML-08, pages 1096 – 1103, New York, NY, USA. ACM.

Warde-Farley, D., Goodfellow, I. J., Courville, A., and Bengio, Y. (2014).

An empirical analysis of dropout in piecewise linear networks.

arXiv, 2014:1 – 10.

Wolpert, D. and Macready, W. (1997).

No free lunch theorems for optimization.

IEEE Transactions on Evolutionary Computation, 1(1):67 – 82.

Wolpert, D. H. (1996).

The lack of a priori distinctions between learning algorithms.

Neural Computation, 8(7):1341 – 1390.

References XV

Wolpert, D. H. and Macready, W. G. (1999).
An efficient method to estimate bagging's generalization error.
Machine Learning, 35:41 – 55.

Zhu, X. (2008).
Semi-supervised learning literature survey.
Technical Report 1530, Computer Sciences, University of Wisconsin, Madison.