Regularization Methods

Karim Baidar, Alexander Zorn

MA-INF 4209
Seminar Principles of Data Mining and Learning Algorithms

24^{st} November of 2020

Outline

Introduction

Main Problem: Overfitting!

Goal: Want to perform better on test data to a possible extend of worser performance on training data.

Definition: Regularization

Define Regularization as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

In practice those methods are more promising then searching for a model with suited capacity!

Outline

L1 Regularization or Ridge Regression

Regularized Objective Function:

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

Where

$$\alpha = [0, \infty] \textit{ is a hyperparameter}$$

Regularization Continued

L2 Parameter Regularization: Ridge Regression or Tikonohov Regularization or Weight Decay

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

Regularization Continued

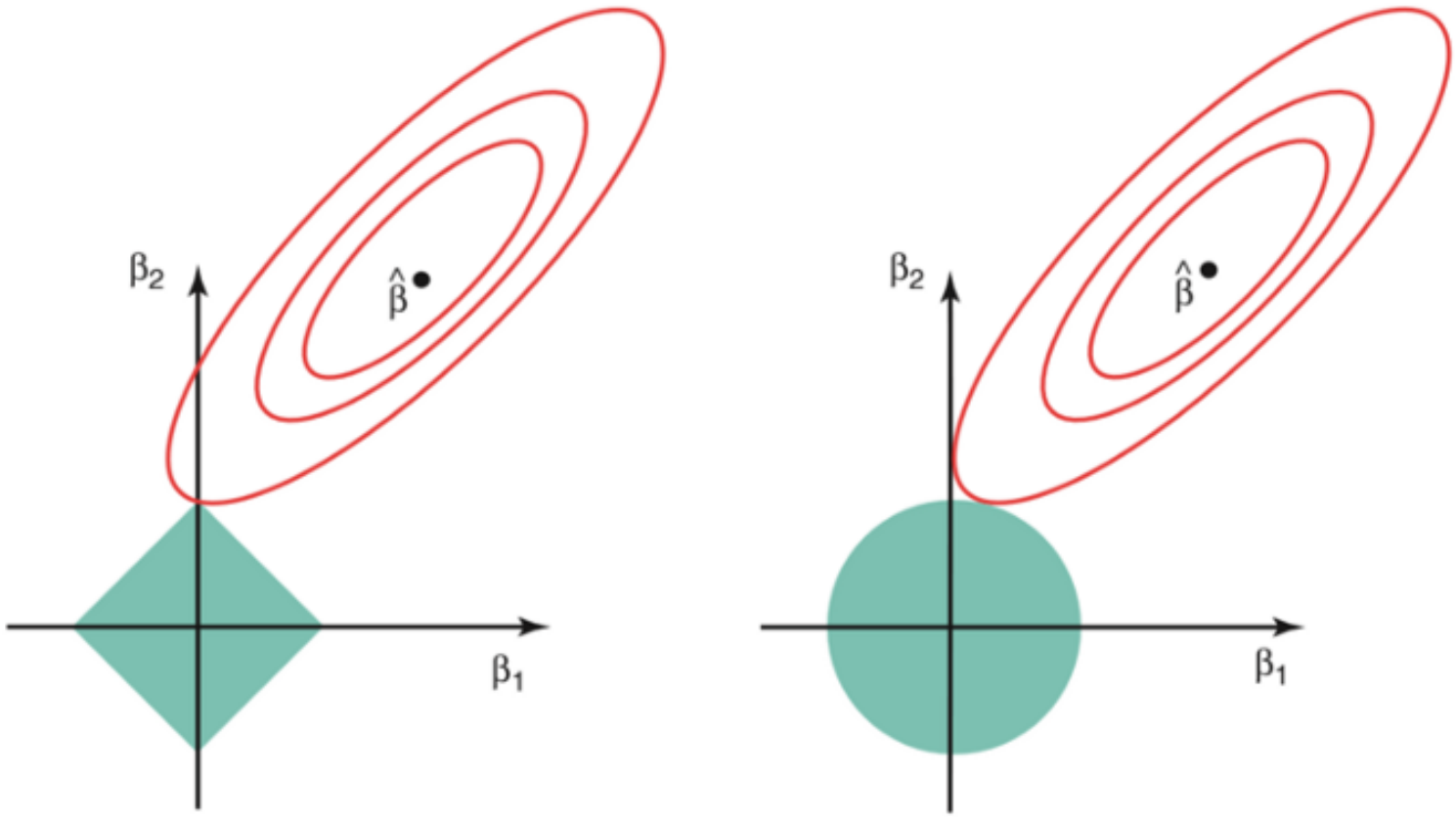
L1: LASSO (Least Absolute Shrinkage and Selection Operator) Regression

$$\tilde{J}(w; X, y) = \alpha \|w\|_1 + J(w; X, y)$$

where

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

Regularization Continued



Outline

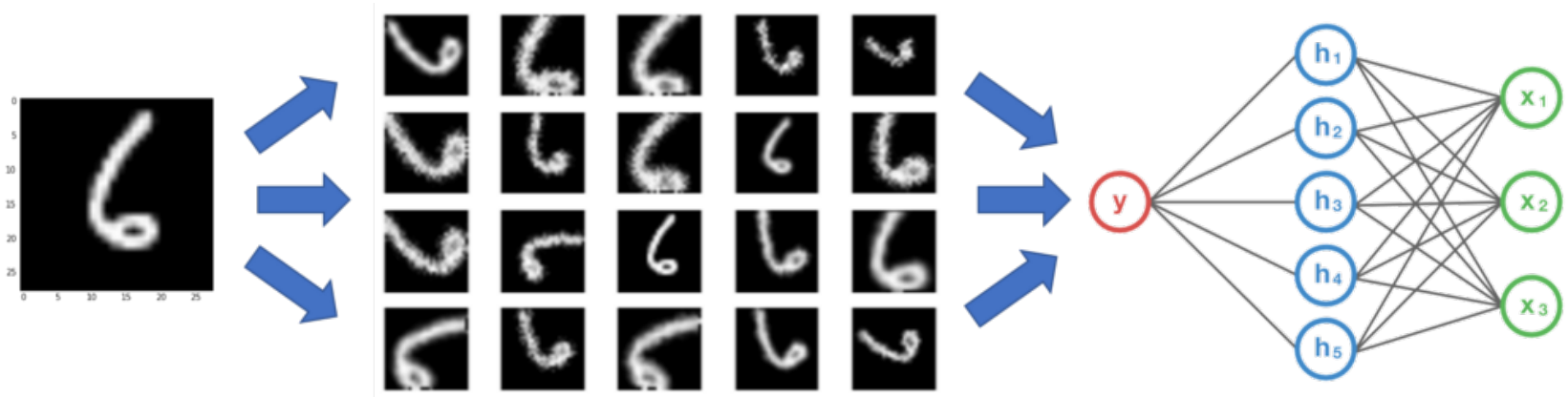
Data Augmentation

Idea: Solve the problem of processing Limited Data with less diversity in order to get efficient results from the neural nets

Offline Augmentation: For smaller Dataset
Online Augmentation: For larger Dataset
where can we apply:
1. Speech Recognition
2. Object Recognition
3. Injecting Noise in the input to a neural network

Data Augmentation Techniques
1. Flip
2. Rotation
3. Scale
4. Crop

Data Augmentation:



[Figure : www.kdnuggets.com]

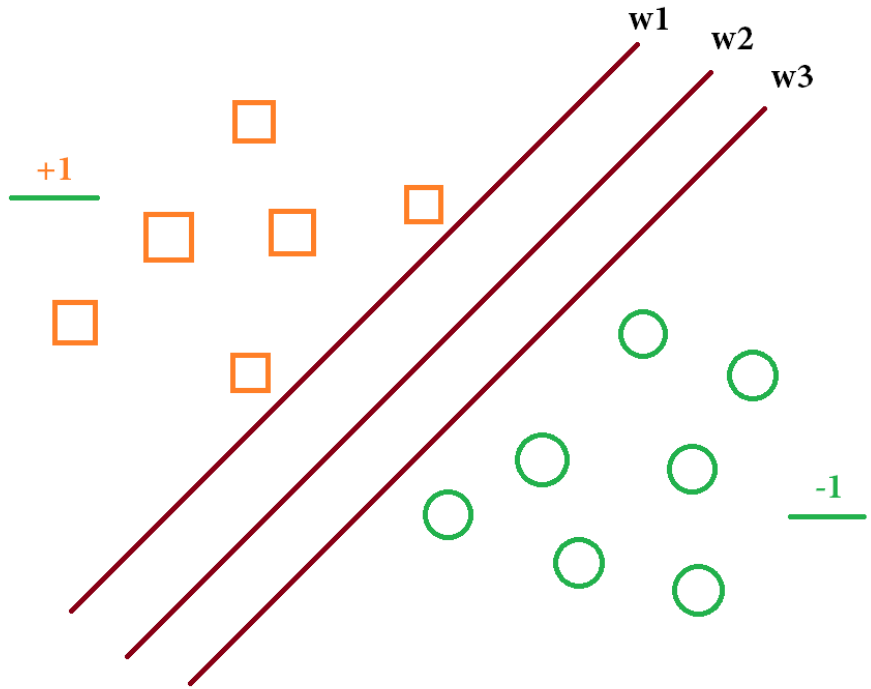
Outline

Noise Robustness

In some models Addition of noise at the input with variance is equivalent to adding:

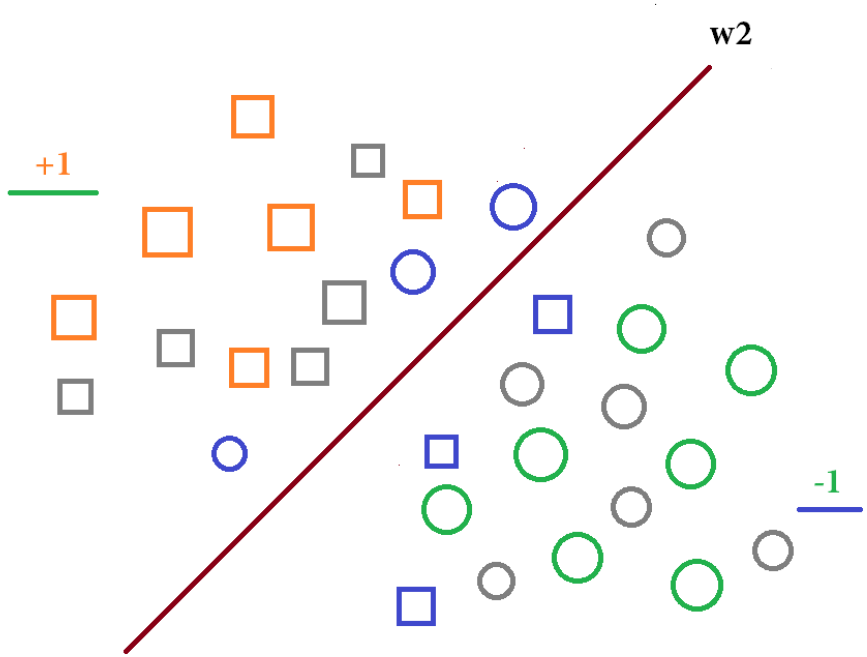
$$\Omega(\theta) = \sum_i |w_i|$$

[Figure : www.kdnuggets.com]



[Figure : Idea from Support vector Machines]

Noise Robustness



[Figure : Idea from Support vector Machines]

Example

where

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

$$\tilde{x}_i = x_i + \epsilon_i$$

$$\hat{y} = \sum_{i=1}^{n} w_i x_i$$

$$\tilde{y} = \sum_{i=1}^{n} w_i \tilde{x}_i$$

$$\tilde{y} = \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} w_i \epsilon_i$$

$$\tilde{y} = \hat{y} + \sum_{i=1}^{n} w_i \epsilon_i$$

Noise Robustness

We are interested in

$$E[(\tilde{y}-y)^2]$$

$$E[(\tilde{y}-y)^2] = E[(\hat{y} + \sum_{i=1}^{n} w_i \epsilon_i - y)^2$$

$$E[(\hat{y}-y)^2] = E[((\hat{y} - y) + (\sum_{i=1}^{n} w_i \epsilon_i))^2]$$

Noise Robustness

$$E[(\tilde{y}-y)^2] = E[(\hat{y}-y)^2] + E[2(\hat{y}-y)\sum_{i=1}^{n} w_i\epsilon_i] + E[(\sum_{i=1}^{n} w_i\epsilon_i)^2]$$

$$E[(\tilde{y}-y)^2] = E[(\hat{y}-y)^2] + 0 + E[\sum_{i=1}^{n} w_i^2\epsilon_i^2]$$

$$E[(\tilde{y}-y)^2] = E[(\hat{y}-y)^2] + \sigma^2\sum_{i=1}^{n} w_i^2$$

Semi Supervised Learning

Idea: Combines a small amount of labeled data with a large amount of unlabeled data during training to Improve Learning Accuracy

[Figure from semanticscholar.org]

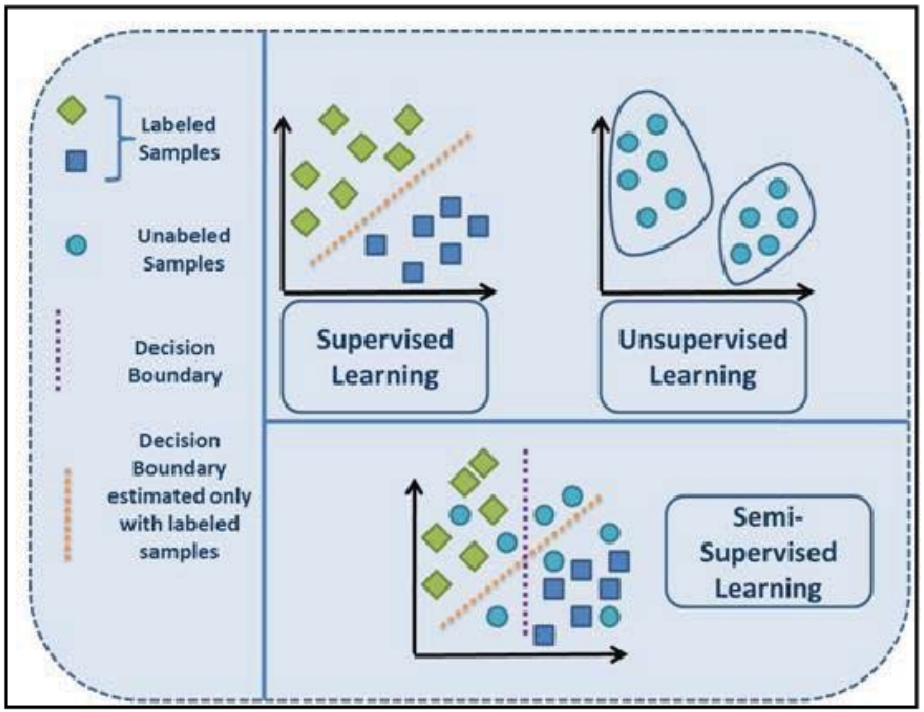


Fig. 1Semi-Supervised Learning

Semi Supervised Learning

Another Example:

1. P(x,y) denotes the joint distribution of x and y, i.e., corresponding to a training sample x, and have a label y

2. P(x) denotes the marginal distribution of x

3. In Semi-supervised Learning, we use both P(x,y) and P(x) to estimate P(y|x)

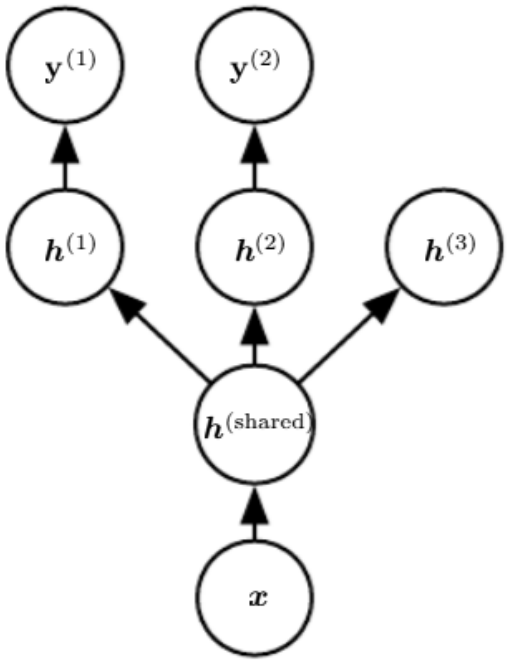
4. We want to learn some representation h = f(x) such that samples which are closer in the input space have similar representations and a linear classifier in the new space achieves better generalization error.

5. we can have a generative model of P(x) (or P(x, y)) which shares parameters with the discriminative model.

Multi Task Learning

Idea: Learning together can be a good regularizer

Multitask learning leads to better generalization when there is actually some relationship between the tasks

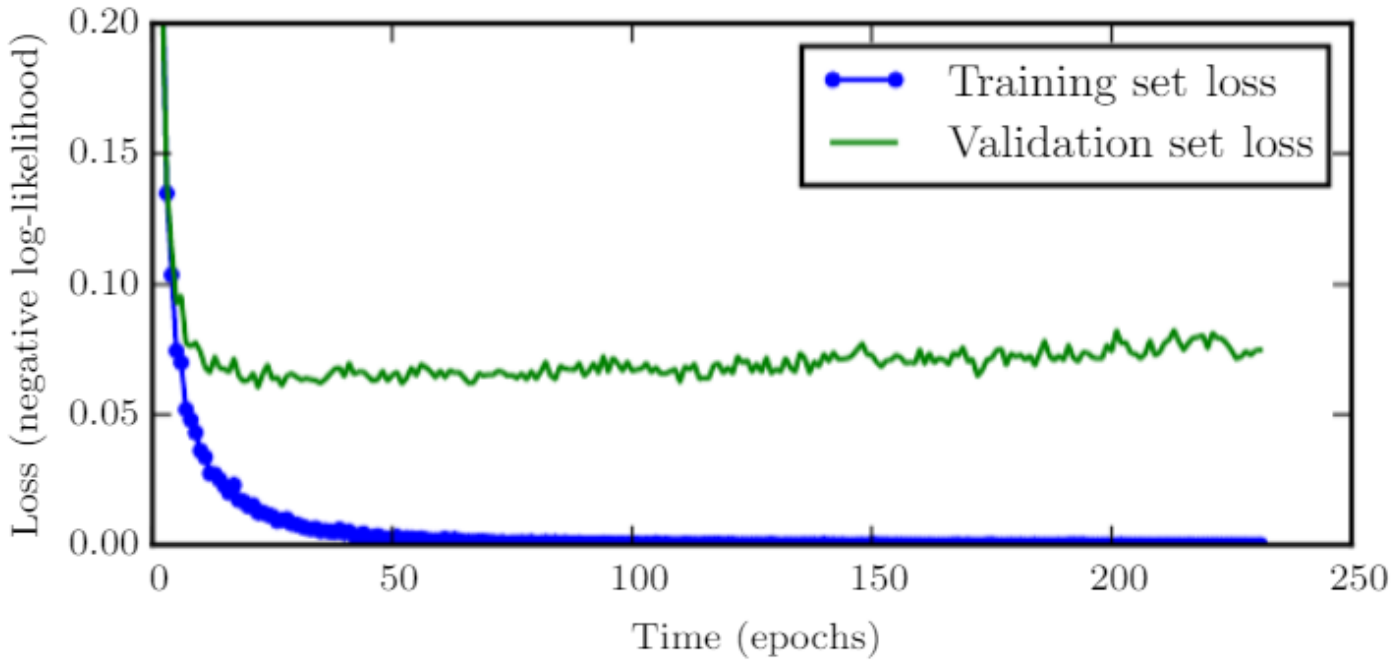


Outline

Early Stopping

Idea: One way to think of early stopping is as a very efficient hyperparameter selection algorithm..

Early Stopping



Early Stopping

Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

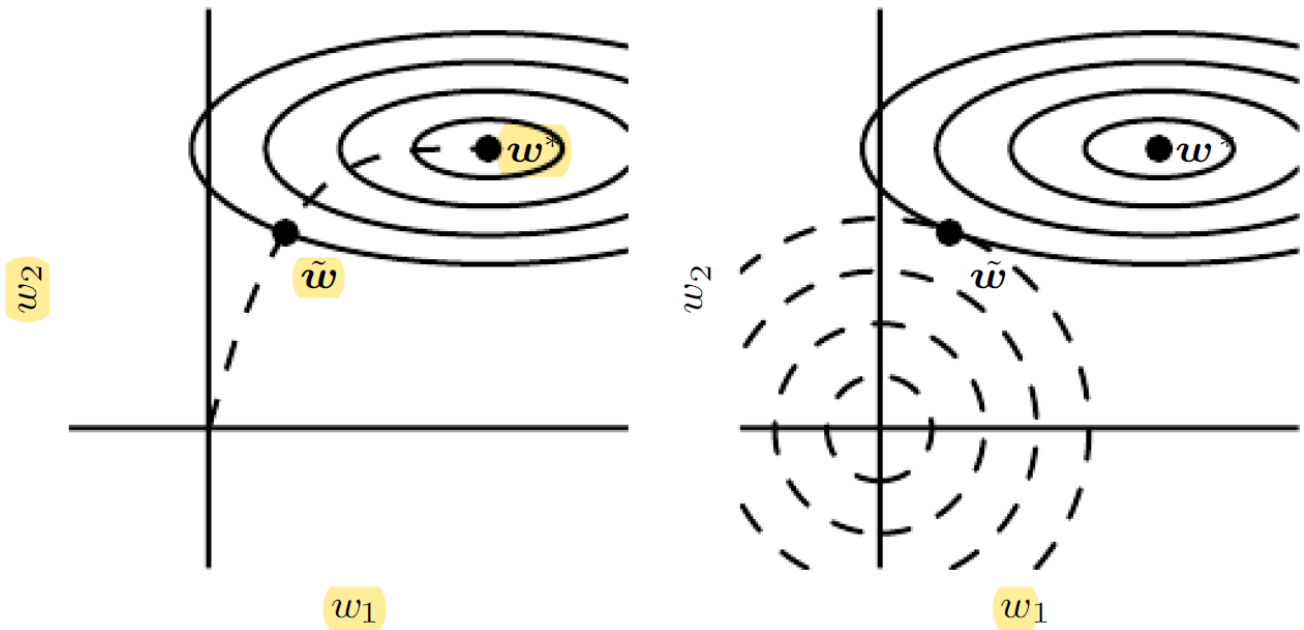
Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o \quad j \leftarrow 0 \quad \boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i \leftarrow 0 \quad v \leftarrow \infty \quad i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0 \quad i^* \leftarrow i$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta} \quad v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Early Stopping

Algorithm 7.2 A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.
Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for i^* steps.

Early Stopping



Outline

Parameter Tying and Sharing

The previous techniques often penalized the parameters when deviating from a constant.
Prior knowledge often need more tools in that direction.

Parameter Tying: Express that certain parameters should be close to each other. Typically from two different models. Using L_2 Regularization:

- Consider models A with $w^{(A)}$ and B with $w^{(B)}$.
- Outputs: $\hat{y}^{(A)} = f(w^{(A)}, x)$, $\hat{y}^{(B)} = f(w^{(B)}, x)$
- Change Loss with additive Regularization term: $\Omega(w^{(A)}, w^{(B)}) := ||w^{(A)} - w^{(B)}||_2^2$

Parameter Tying and Sharing

Parameter Sharing: force sets of parameters (in one or over multiple models) to be equal.

Advantage: For each set only one weight has to be stored. Enables huge networks!

Used heavily on **CNN** training in computer vision:

- When detecting features (e.g lines) the feature detector applied to every pixelarea shares his weights with all detectors for the same feature.
- Ability to recognise object independently of their translation or size in the image.
- More details: 12^{st} Jan. of 2021 Seminar Talk about CNNs

Dropout

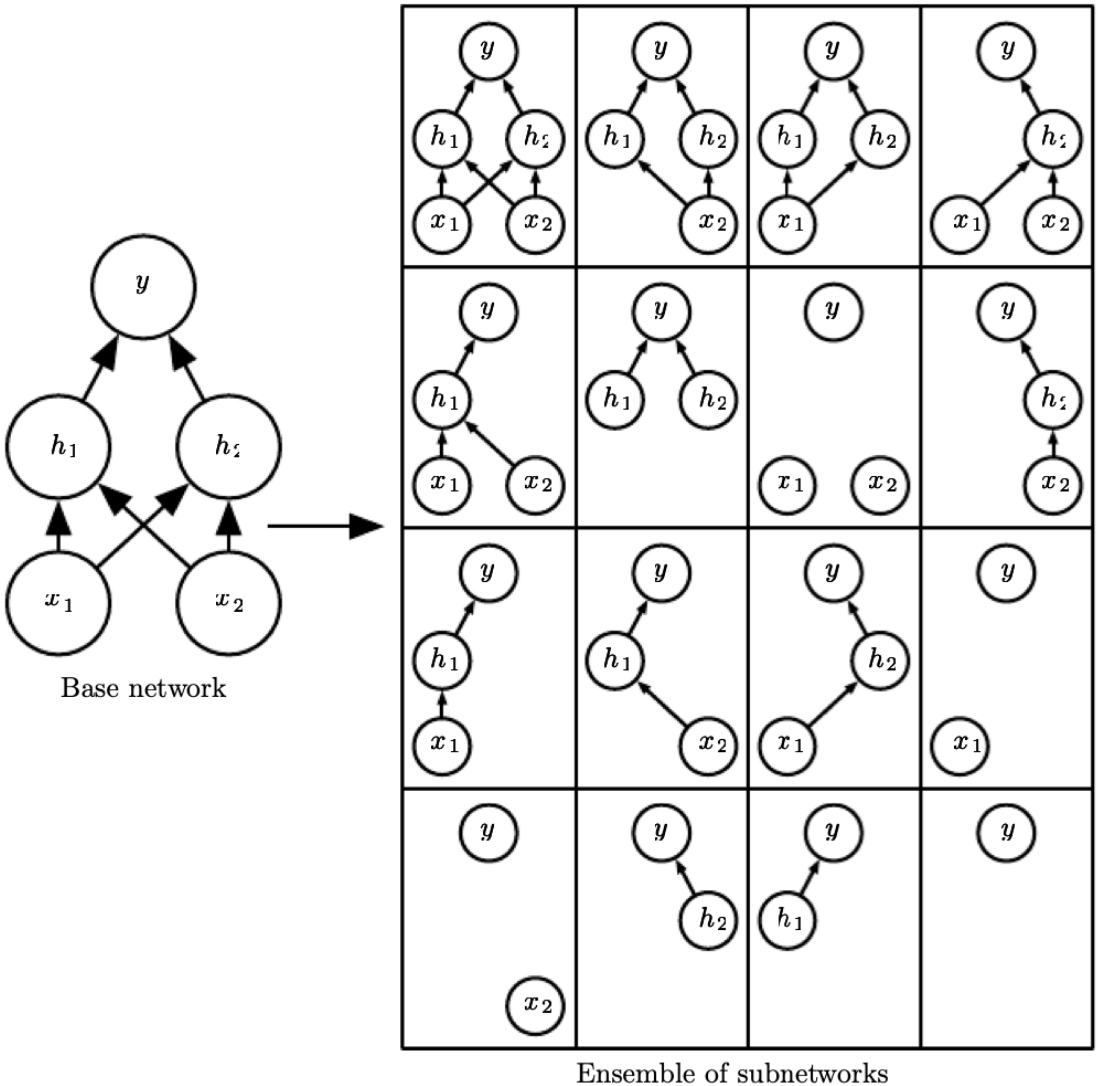
Goal: Improve generalization and speed up training.

Short: Training ensemble of all sub-networks of a NN.

Setting:

- **Sub-networks:** NN reduced by a set S of non output neurons.
- **Bagging:** Special kind of Bagged learning -> Manage k different models and train them on k different datasets.
- **Sample Sub-networks:** Each Neuron from a non output layer l gets assigned to S with probability $1 - p_l$.
  - typically 0.5 for hidden and 0.8 for input layers

Sample Sub-networks:



Training with Dropout

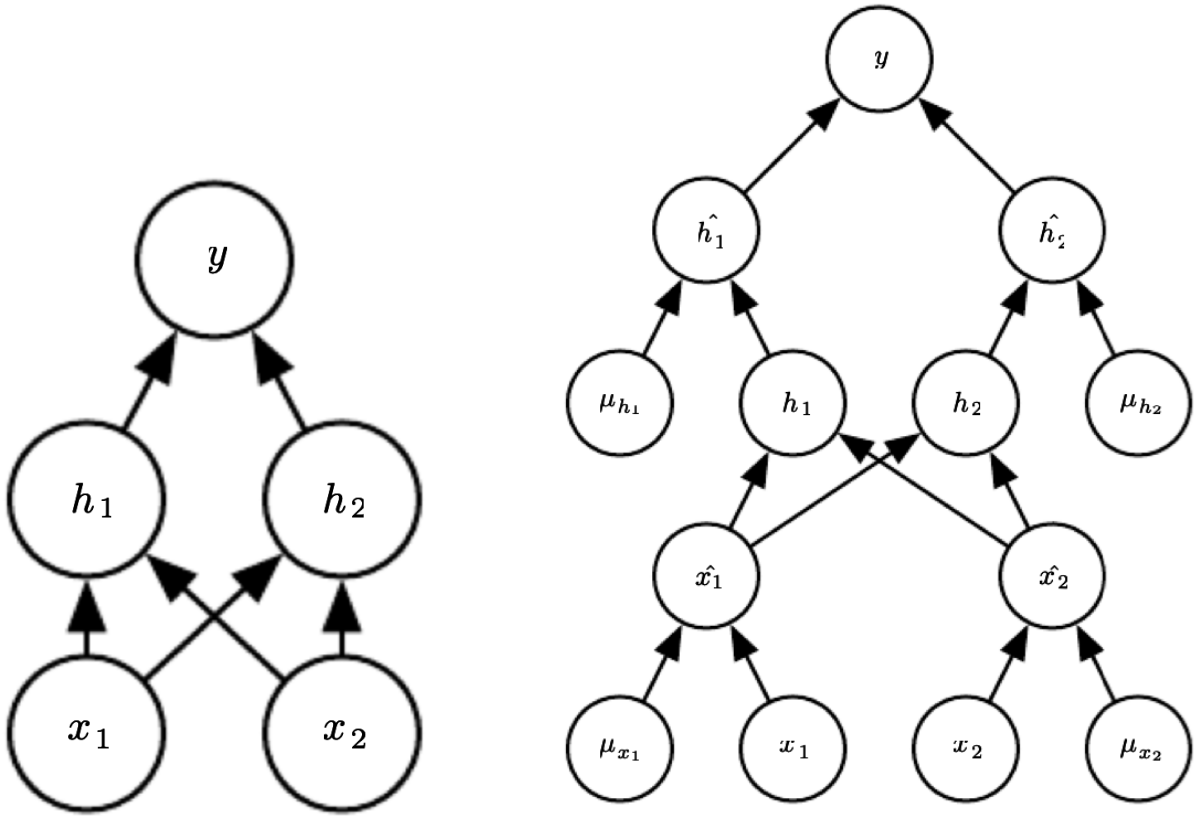
Train using the following routine:

- Sample a mask μ corresponding to a subnet N
- Train N on a minibatch with loss: $J(\theta, \mu)$, stochastic gradient descent
- change the weights of the orig NN
- iterate

This algorithm seeks to minimize $\mathbb{E}_\mu(J(\theta, \mu))$ unbiased estimator but with exponentially many terms.

Only a tiny fraction of all models can be trained.

Network transformation:



Very cheap additive costs for implementation!

Prediction with Dropout

Not able to make predictions with the net as usual because e.g two subnet with non intersecting hidden nodes are trained on the same task their results would be added up in the whole NN.

Most used and computationally efficient method: **weight scaling inference rule**

When doing a forward propagation for a prediction replace each weight w_n outgoing from neuron n (in layer l) with $w_n * p_l$.

Dropout Summary

Summarizing Dropout:

- computationally inexpensive fast tool for learning large models (with n neurons)
  - $O(n)$ for constructing the transformed NN
  - $O(n)$ effort to sample a subnet
  - training as usual but faster since smaller net
- More effective than the common regularization techniques (e.g weight decay) due to [Srivastava et al. (2014)]
- Standard widely used tool for training and prior knowledge integration.

Outline

Adversarial Training

Motivation: NNs that solve a task on human level usually don't reach human understanding.

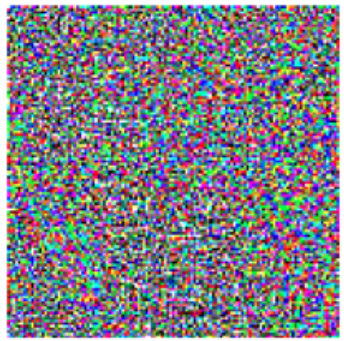
Suspected problem: Heavy use of linearity of NNs. A change of ϵ on each output can change the output up to $\epsilon||w||_1$

Goal: local constancy around training examples.

Solution: Train model on slightly modified (random noise) instances where the output of the model is very different from the teached one (**adversarial example**).

Adversarial example

$+ .007 \times$



$=$

$\boldsymbol{x}$

$y =$"panda"
w/ 57.7%
confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
w/ 8.2%
confidence

$\boldsymbol{x} + \epsilon \, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
w/ 99.3 %
confidence

References I

[1] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613.

All pictures (if not referenced otherwise) are from [2].

Discussion