Deep Learning

Lecture 3

Fernando Perez-Cruz
based on Thomas Hofmann lectures

Swiss Data Science Center
ETH Zurich and EPFL – `datascience.ch`

October 8th, 2018

Overview

1. Feedforward Networks

2. Output Units and Objectives

3. Regularization in Deep Learning

Section 1

Feedforward Networks

Feedforward Networks

Definition: Feedforward Network

A set of computational units arranged in a DAG (directed acyclic graph) are called a feedforward network.

- cf. Chapter 6 of DL book
- in contrast: output of network fed back into the computation $\Longrightarrow$ recurrent networks
- most cases of interest: layer-wise processing (cf. above)

$$F = F^L \circ \cdots \circ F^1, \quad F^l = \sigma_l \circ \bar{F}^l, \quad \bar{F}^l(\mathbf{h}^{l-1}) = \mathbf{W}^l \mathbf{h}^{l-1} + \mathbf{b}$$

  - σ^l: element-wise non-linearity of layer l.
  - $\bar{F}^l$: linear function in layer l. $\bar{F}^l \in \mathbb{R}^{m^l}$
  - $\mathbf{h}^0 := \mathbf{x}$.

Hidden Layers

Definition: Hidden Layer

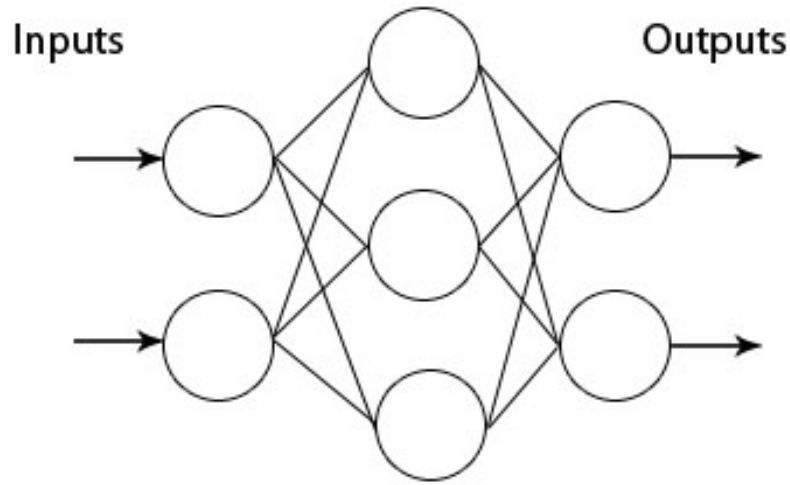
A layer that is neither the input, nor the output layer is called a hidden layer.

- some think of inputs as a "layer" (not in our notation)
- outputs are specified by targets (e.g. during training)
- hidden layer = intermediate representation
- "hidden" = not directly specified by training data
- probabilistic models = latent variables

Multilayer Perceptron

Canonical architecture (since the 1980ies)

- three layer architecture:
  input - hidden - output
- classical: sigmoid activation
  at hidden layer
- modern: ReLU activation at
  hidden layer(s), deeper
- problem-dependent
  activation at outputs



Function Approximation

A feedforward neural network represents family of functions $\mathcal{F}$.

Functions $F_\theta \in \mathcal{F}$ are parameterized by parameters θ (i.e. weight matrices $\mathbf{W}^l$ and biases $\mathbf{b}^l$, $l = 1, \ldots, L$).

How can we find the most appropriate function in $\mathcal{F}$ based on training data $\{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \ldots, N\}$ (i.e. N input-output pairs)?

This is what we turn our attention towards now...

Section 2

Output Units and Objectives

Outputs in (Deep) Learning

- It tell us how to know that we are doing good.
- Two options:
  - Decision theory
  - Maximum Likelihood.
- Both of them lead to similar loss functions.

Decision Theory

- Minimum Risk:

$$F^* = \arg\min_F \sum_y \int \ell(y, F(\mathbf{x})) p(\mathbf{x}, y) d\mathbf{x}.$$

  We need to know $\ell(y, F(\cdot))$, $p(\mathbf{x}, y)$.

- $\ell(y, F(\cdot))$ is the loss function. What is it?
- But we do not know $p(\mathbf{x}, y)$, we are only given:

$$\mathcal{S}_N := \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\},$$

  drawn independently and identically distributed from $p(\mathbf{x}, y)$.

Loss Function

For learning, we need to assess the goodness-of-fit of a network.

Definition: Loss function

A loss (or cost) function is a non-negative function

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}, \quad (\mathbf{y}^*, \mathbf{y}) \mapsto \ell(\mathbf{y}^*, \mathbf{y})$$

such that $\ell(\mathbf{y}, \mathbf{y}) = 0$ $(\forall \mathbf{y} \in \mathcal{Y})$ and $\ell(\mathbf{y}^*, \mathbf{y}) > 0$ $(\forall \mathbf{y} \neq \mathbf{y}^*)$.

- here: $\mathcal{Y}$: output space
- general understanding: $\mathbf{y}^*$ is truth and $\mathbf{y}$ predicted

Loss Function: Examples

Example: squared-error:

$$\mathcal{Y} = \mathbb{R}^m, \quad \ell(\mathbf{y}^*, \mathbf{y}) = \frac{1}{2}\|\mathbf{y}^* - \mathbf{y}\|_2^2 = \frac{1}{2}\sum_{i=1}^{m}(y_i^* - y_i)^2$$

Example: classification error:

$$\mathcal{Y} = [1:m], \quad \ell(\mathbf{y}^*, \mathbf{y}) = 1 - \delta_{\mathbf{y}^*\mathbf{y}}$$

with Kronecker delta: $\delta_{ab} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$

Expected Risk

Definition: Expected Risk

Assume inputs and outputs are governed by a distribution $p(\mathbf{x}, \mathbf{y})$ over $\mathcal{X} \times \mathcal{Y}$, $\mathcal{X} \subseteq \mathbb{R}^n$. The expected risk of F is given by

$$\mathcal{R}^*(F) = \mathbf{E}_{\mathbf{x},\mathbf{y}} \left[\ell(\mathbf{y}, F(\mathbf{x})) \right]$$

- as distribution p is generally unknown, we cannot evaluate $\mathcal{R}^*$ directly, but it serves as a point of reference in learning theory
- $\mathcal{R}^*$ is a functional (mapping functions to scalars)
- parameterized functions $\{F_\theta : \theta \in \Theta\} \Longrightarrow \mathcal{R}^*(\theta) := \mathcal{R}^*(F_\theta)$

Training Risk

Definition: Training Risk

Assume we have a random sample of N input-output pairs,

$$\mathcal{S}_N := \{(\mathbf{x}_i, \mathbf{y}_i) \overset{\text{i.i.d.}}{\sim} p : i = 1, \ldots, N\}.$$

The training risk of F on a training sample is defined as

$$\mathcal{R}(F; \mathcal{S}_N) = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{y}_i, F(\mathbf{x}_i))$$

- training risk = expected risk under the empirical distribution induced by the sample $\mathcal{S}_N$

Empirical Risk Minimization

For a family $\mathcal{F} = \{F_\theta : \theta \in \Theta\}$ (e.g. neural network) and training data $\mathcal{S}_N$: find function with lowest empirical risk.

Definition: Empirical risk minimization

The empirical risk minimizer is defined as

$$\hat{F}(\mathcal{S}_N) = \operatorname*{argmin}_{F \in \mathcal{F}} \mathcal{R}(F; \mathcal{S}_N)$$

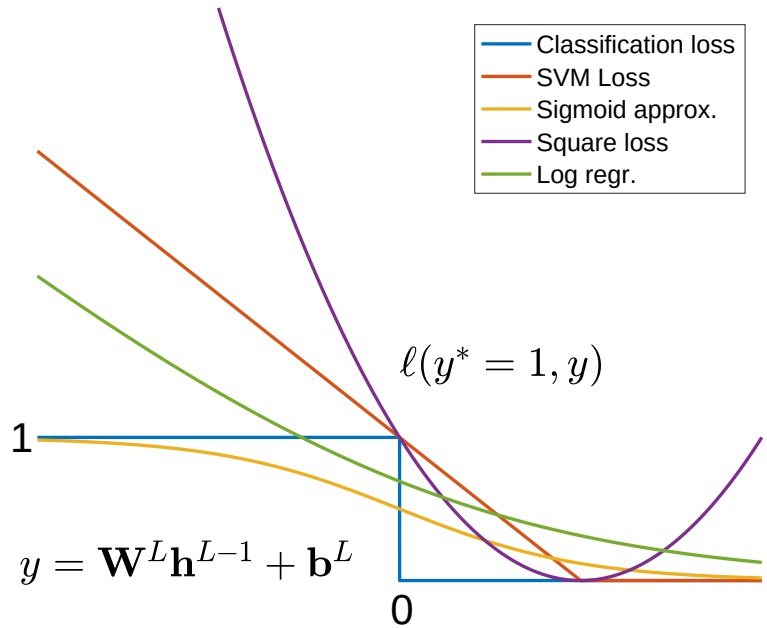
with corresponding parameters $\hat{\theta}(\mathcal{S}_N)$.

- one may also add a regularizer $\Omega(F)$ or $\Omega(\theta)$ to the risk (more on that later)
- finding $\hat{F} \in \mathcal{F}$ amounts to solving an optimization problem

Classification loss

Classification loss derivate is either zero or undefined

- SVM loss: $\max(0, 1 - y)$
- Square loss: $(1 - y)^2$
- Log regression: log(1+exp(-y))



Probability Distributions as Outputs

It is often constructive to think of functions F as mappings from inputs to distributions $\mathcal{P}(\mathcal{Y})$ over outputs $\mathbf{y} \in \mathcal{Y}$.

$$F : \mathbb{R}^n \to \mathbb{R}^m, \quad \mathbf{x} \mapsto \mu, \quad \mu \overset{\text{fixed}}{\mapsto} p(\mathbf{y}; \mu) \in \mathcal{P}(\mathcal{Y}), \quad \mathbf{y} \sim p(\mathbf{y}; \mu)$$

Each F effectively defines a conditional probability distribution (or conditional probability density function) via

$$p(\mathbf{y}|\mathbf{x}; F) = p(\mathbf{y}; \mu = F(\mathbf{x}))$$

Example: Multivariate Normal Distribution

Example: μ = mean of a normal distribution
(with fixed covariance matrix $\gamma^2\mathbf{I}$)

$$p(\mathbf{y}|\mathbf{x}; F) = \left[\frac{1}{\sqrt{2\pi}\gamma}\right]^m \exp\left[-\frac{1}{2\gamma^2}\|\mathbf{y} - F(\mathbf{x})\|^2\right]$$

so that

$$-\log p(\mathbf{y}|\mathbf{x}; F) = C(\gamma) + \frac{1}{2\gamma^2}\|\mathbf{y} - F(\mathbf{x})\|^2$$

which is equivalent to the squared error loss.

- $F(\mathbf{x}) = \mu$ and $\mathbf{y}$ live in same space ($\mathbb{R}^m$)

Generalized Linear Models

Generalized linear models: predict the mean of the output distribution. Use the form

$$\mathbf{E}[\mathbf{y}|\mathbf{x}] = \sigma(\mathbf{w}^\top \mathbf{x})\,.$$

where σ is invertible and σ^{-1} is called the link function.

- can be extended to also predict variances or dispersions

Example: Logistic Regression

Example: Logistic regression

$\mathcal{Y} = \{0, 1\}$, $\mathcal{P}(\mathcal{Y}) = [0; 1]$, $\sigma(x) = 1/(1 + e^{-x})$, then:

$$\mathbf{E}[y|\mathbf{x}] = p(1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

- link function: logit

$$\sigma^{-1}(t) = \log\left(\frac{t}{1-t}\right), \quad t \in (0; 1)$$

Example: Multinomial Logistic Regression

Example: Multinomial logistic regression:

$\mathcal{Y} = [1 : m]$, $\mathcal{P}(\mathcal{Y})$ can be represented via soft-max

$$p(y|\mathbf{x}) = \frac{\exp[z_y]}{\sum_{i=1}^{m} \exp[z_i]}, \quad z_i := \mathbf{w}_i^\top \mathbf{x}, \quad i = 1, \ldots, m$$

- over-parametrized model: set $\mathbf{w}_1 = \mathbf{0}$, s.t. $z_1 = 0$ (w.l.o.g.)
- generalizes (binary) logistic regression (*left as exercise*)

Generalized Linear Units

In neural networks:

- non-linear functions replace linear functions
- output layer units implement inverse link function

Examples:

- normal model = linear output layer

$$\mathbf{E}\left[\mathbf{y}|\mathbf{x}\right] = \bar{F}(\mathbf{x}) = \mathbf{W}^L \left(F^{L-1} \circ \cdots \circ F^1\right)(\mathbf{x}) + \mathbf{b}^L$$

- logistic regression = sigmoid output layer

$$\mathbf{E}\left[y|\mathbf{x}\right] = \sigma(\bar{F}(\mathbf{x}))$$

Log-Likelihood

Canonical way of defining risk functions: negative of a log-likelihood function.

Use conditional probability distribution to define generalized loss (between target value $\mathbf{y} \in \mathcal{Y}$ and a distribution over $\mathcal{Y}$)

$$\mathcal{R}(\theta; (\mathbf{x}, \mathbf{y})) = -\log p(\mathbf{y}|\mathbf{x}; \theta)$$

- non-linearity of output layer is "absorbed" in risk/link function
- i.e. $\mathcal{R}$ depends on $\bar{F}$
- provides a "template" for generalized loss/risk functions

Logistic Log-Likelihood

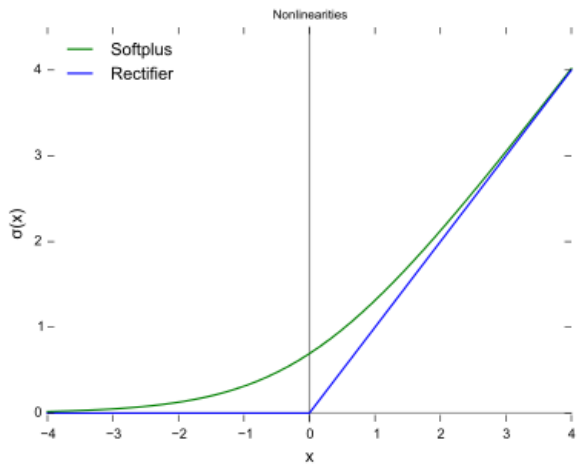
Let us look at the (implied) risk function for the logistic function.

Use shorthand $z := \bar{F}(\mathbf{x}) \in \mathbb{R}$ then

$$-\log p(y|z) = -\log \sigma((2y-1)z) = \zeta((1-2y)z)$$

where $\zeta = \log(1+\exp(\cdot))$ is the soft-plus function.

also commonly called:
cross entropy loss



Multinomial Log-Likelihood

More generally (multinomial logistic regression), use shorthand:

$$\mathbf{z} := \bar{F}_i(\mathbf{x}) \in \mathbb{R}^m$$

then:

$$\mathcal{R}(F; (\mathbf{x}, y)) = -\log p(y|\mathbf{x}; F) = -\log \left[\frac{e^{z_y}}{\sum_{i=1}^m e^{z_i}} \right]$$

$$= -z_y + \underbrace{\log \sum_{i=1}^m \exp[z_i]}_{\text{log-partition fct.}} = \log \left[1 + \sum_{i \neq y} \exp[z_i - z_y] \right]$$

Section 3

Regularization in Deep Learning

Decision Theory (recap)

- Minimum Risk:

$$F^* = \arg\min_F \sum_y \int \ell(y, F(\mathbf{x}))p(\mathbf{x}, y)d\mathbf{x}.$$

  We need to know $\ell(y, F(\cdot))$, $p(\mathbf{x}, y)$.

- But we do not know $p(\mathbf{x}, y)$, we are only given:

$$\mathcal{S}_N := (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n),$$

  drawn independently and identically distributed from $p(\mathbf{x}, y)$.

- What can we do?

Density Estimation?

- Use the n given samples to estimate $p(\mathbf{x}, y)$.
- We can use our estimate $\widehat{p}(\mathbf{x}, y)$ to solve:

$$\widehat{F} = \arg\min_{F} \sum_{y} \int \ell(y, F(\mathbf{x})) \widehat{p}(\mathbf{x}, y) d\mathbf{x}.$$

- We transform our classification problem into a probability density estimation problem.
- We can now use the machinery in the previous lecture.
- End of story?
- Which are the limitations to this approach?

Empirical Risk Minimization

- Minimum Risk:

$$\mathcal{R}^*(F) = \sum_y \int \ell(y, F(\mathbf{x}))p(\mathbf{x}, y)d\mathbf{x}.$$

- Empirical Risk of is the average over the training data set:

$$\mathcal{R}(F; \mathcal{S}_N) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, F(\mathbf{x}_i))$$

- Implicit density estimation:

$$p(\mathbf{x}, y) = \frac{1}{n}\sum_{i=1}^{n} d(\mathbf{x} - \mathbf{x}_i, y - y_i)$$

$d(\cdot, \cdot)$ dirac-delta function.

Empirical Risk Minimization

- We want it to "fit the data". We can find

$$\widehat{F} = \arg\min_{F \in \mathcal{F}} \mathcal{R}(F; \mathcal{S}_N) = \arg\min_{F \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, F(\mathbf{x}_i))$$

- We can find this by equating to zero its derivative:

$$\frac{\partial \mathcal{R}(F; \mathcal{S}_N)}{\partial F} = \sum_{i=1}^{n} \frac{\partial \ell(y_i, F(\mathbf{x}_i))}{\partial F} = 0$$

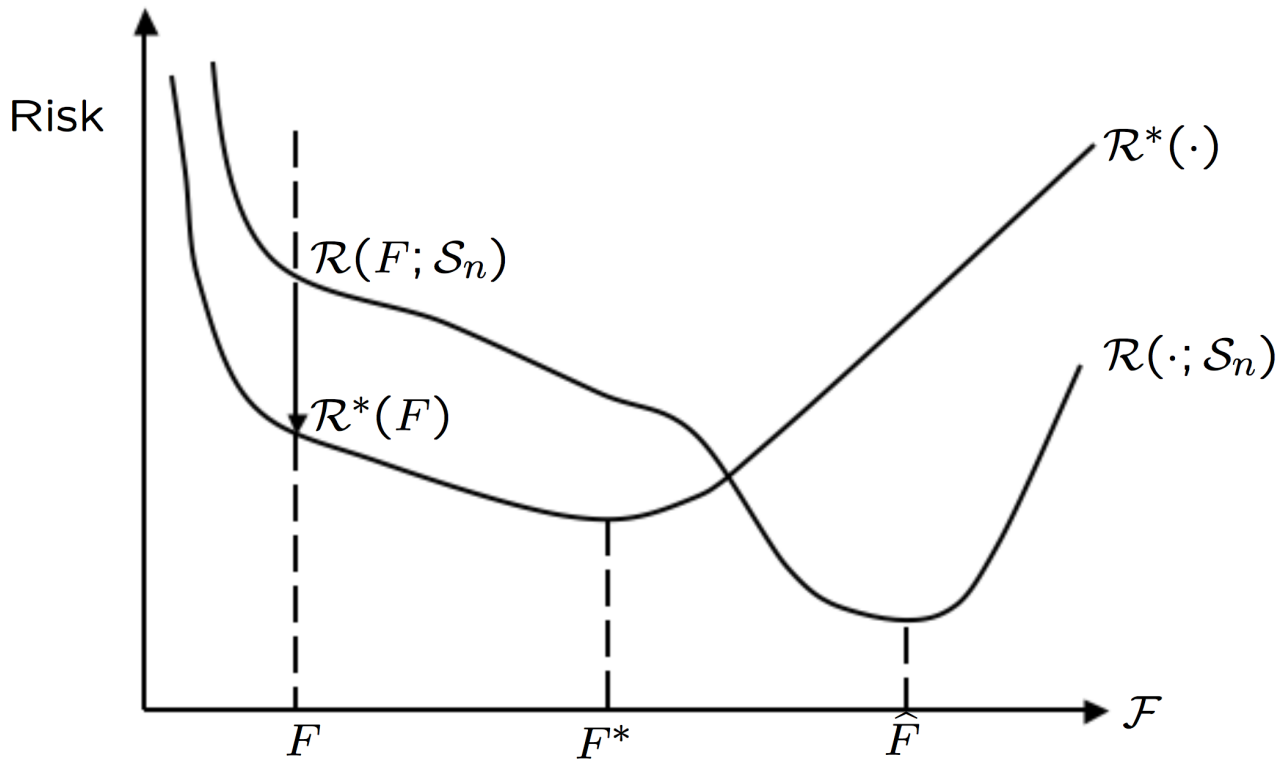
- To solve this equation, we need:
  - to know what $\ell(\cdot, \cdot)$ measures. ✓
  - to know the space $\mathcal{F}$, where we need to look for $F(\cdot)$.
- Will this always give a good answer?

Consistency

- We want that as n tends to infinity that

$$\mathcal{R}(\widehat{F}; \mathcal{S}_n) \quad \longrightarrow \quad R^*(F^*)$$

- As n grows we want to do as good as we we did know $p(\mathbf{x}, y)$.

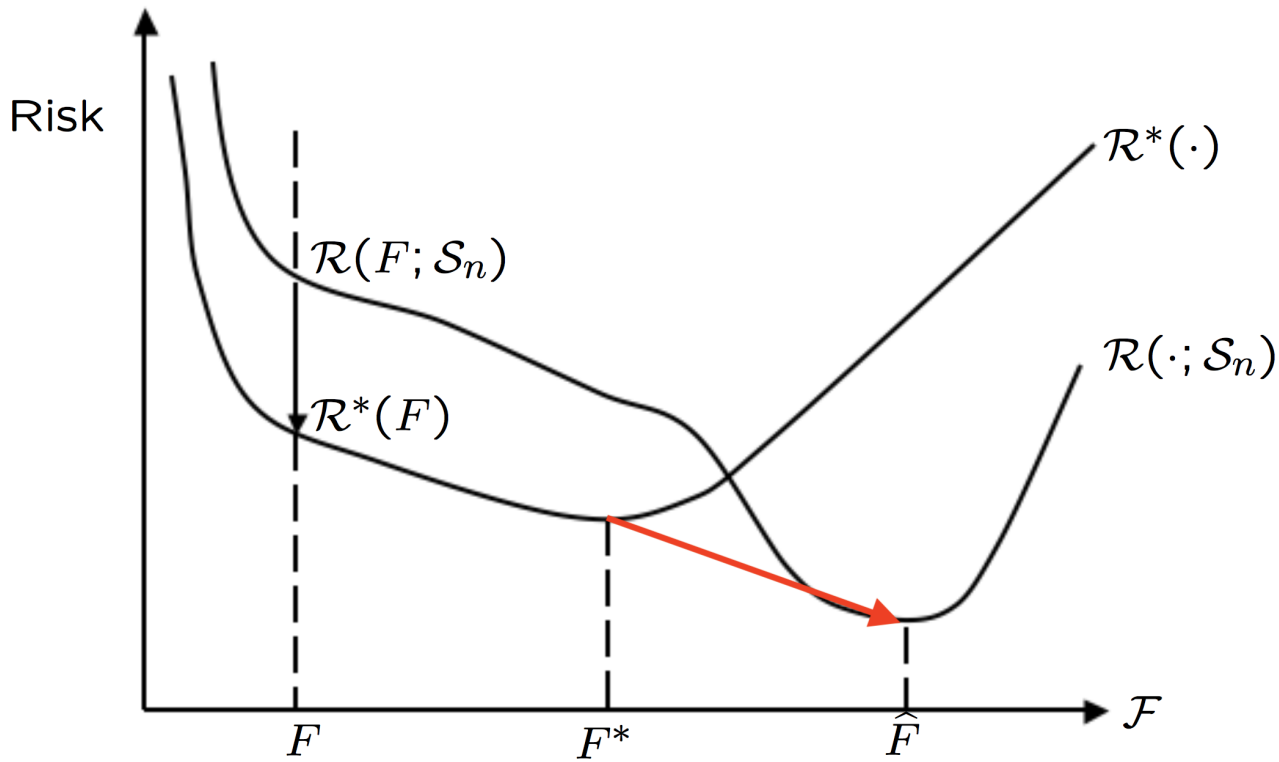


Consistency

- We want that as n tends to infinity that

$$\mathcal{R}(\widehat{F}; \mathcal{S}_n) \quad \longrightarrow \quad R^*(F^*)$$

- As n grows we want to do as good as we we did know $p(\mathbf{x}, y)$.

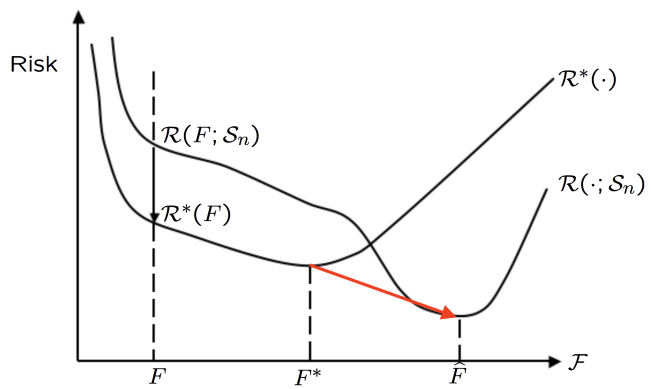


Consistency

- We want that as n tends to infinity that

$$\mathcal{R}(\widehat{F}; \mathcal{S}_n) \quad \longrightarrow \quad R^*(F^*)$$

- As n grows we want to do as good as we we did know $p(\mathbf{x}, y)$.
- To prove this convergence we need a *worse case analysis*, i.e. that all functions in $\mathcal{F}$ converge to the true risk.
- We can only do that if we restrict the class of functions $\mathcal{F}$.
- Regularization is one way of enforcing consistency.



Regularization

- Minimizing the loss for flexible models $\longrightarrow$ **Overfitting**.
- We know that the weights do not need to grow indefinitely to get a good approximation.
- Regularization Theory:

$$\mathcal{R}_r(F; \mathcal{S}_n) = \mathcal{R}(F; \mathcal{S}_n) + \nu\Omega(||F||)$$

- $\mathcal{R}(F; \mathcal{S}_n)$ is the empirical risk.
- $\Omega(||F||) \geq 0$ regularizer.
- ν trade-off hyperparameter.

Weight Decay

$$\Omega(||F||) = \|\mathbf{w}\|^2$$

- It prefers lower weights.
- If a weight increase does not significantly reduce $\mathcal{R}(F; \mathcal{S}_n)$, then $\Omega(||F||)$ will keep it low.
- It is also known as:
  1. Ridge Regression.
  2. Logarithm of Gaussian Prior (MAP).
  3. Maximum margin (SVMs).
  4. It is related to: early stopping and training with noise.

Other regularization approaches

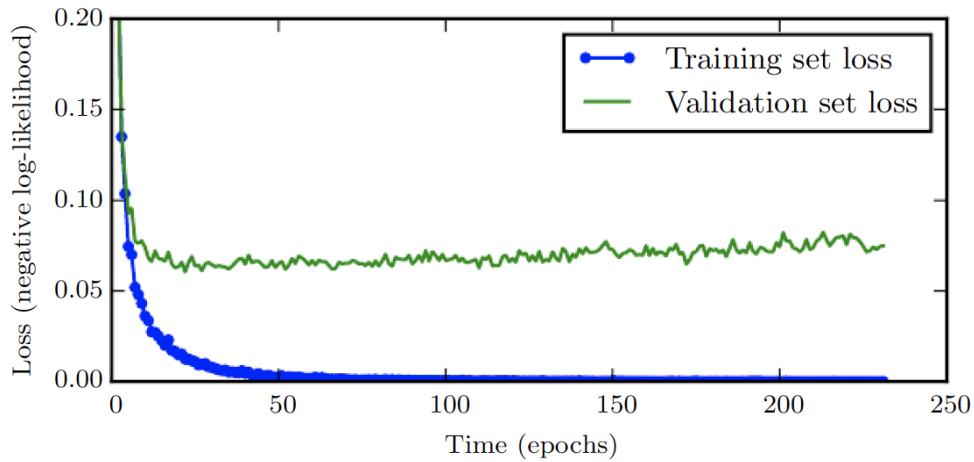
- Data Augmentation.

> The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted patches[4]. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five 224×224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

- Noise Robustness:
  1. Adding it to the input.
  2. Adding it to the weights.
  3. Adding it to the labels.
- Semisupervised learning.

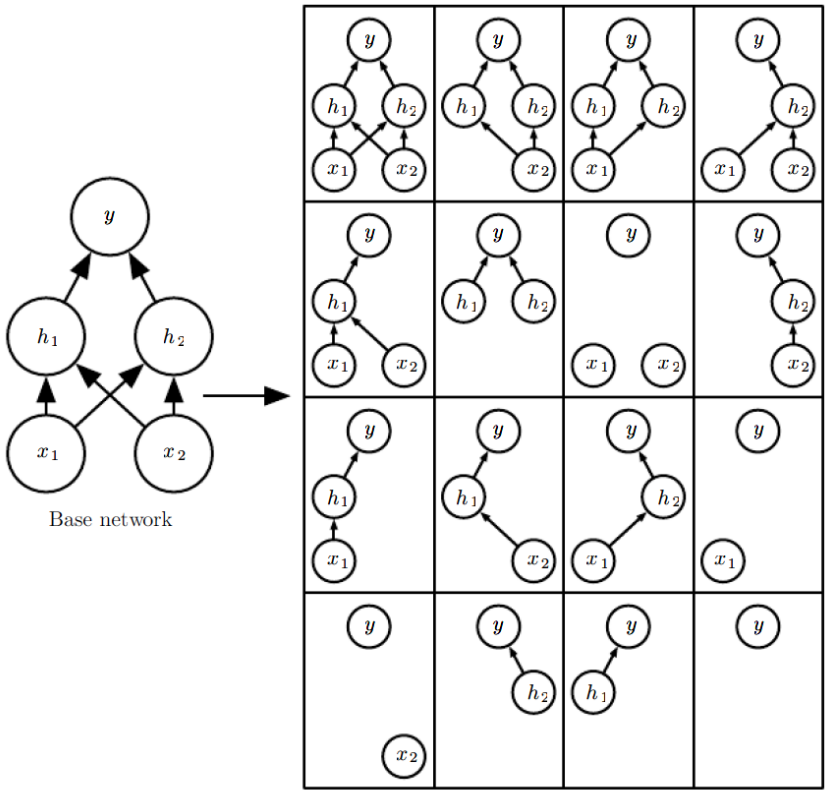
Other regularization approaches II

- Multi Task Learning.
- Early Stopping



- Parameter Sharing.
- Ensembles.

Dropout



$$p_{ensemble}(y|\mathbf{x}) = \sum_{\mu} p(\boldsymbol{\mu}) p(y|\mathbf{x}, \boldsymbol{\mu})$$

$$p_{ensemble}(y|\mathbf{x}) = \sqrt[d]{\prod_{\mu} p(\boldsymbol{\mu}) p(y|\mathbf{x}, \boldsymbol{\mu})}$$

Bibliography

- I. Goodfellow, Y. Bengio and A. Courville, (2016). "Deep Learning". MIT Press (Sections 5.1-5.6, 6.1-6.4, 7.1-7.12).
- D. Mackay, (2003). " Information Theory, Inference and Learning Algorithms". Cambridge University Press (Chapters 39, 41 and 44).
- K. Murphy, (2012), "Machine Learning: A probabilistic Approach". MIT Press (Chapter 8: Logistic regression).
- B Schölkopf and A. Smola, (2003), "Learning with Kernels". MIT Press (Chapter 5: Elements of Statistical Learning Theory).