Deep Learning

Lecture 7

Nathanaël Perraudin
based on Thomas Hofmann lectures

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

November 05, 2018

Overview

1. Optimization Heuristics
2. Norm-based Regularization
3. Dataset Augmentation
4. Dropout

Section 1

Optimization Heuristics

Polyak Averaging

Iterate or Polyak averaging: [DL 8.7.3]
average over iterates (instead of outputting the final iterate)

Linear averaging (common in convex case)

$$\bar{\theta}(t) = \frac{1}{t} \sum_{s=1}^{t} \theta(s)$$

- often strong theoretical guarantees

Running averages (common in non-convex case)

$$\bar{\theta}(t) = \alpha\theta(t-1) + (1-\alpha)\theta(t), \quad \text{time constant} \;\; \alpha \in [0;1)$$

- often good practical results

Batch Normalization

[DL 8.7.1, Ioffe & Szegedy, 2015]

Strong dependencies between weights in layers exist.
Hard to find suitable learning rate.

Toy example: deep network with one unit per layer

$$y = w_1 \cdot \cdots \cdot w_L x \Longrightarrow y^{new} = \prod_{l=1}^{L} \left(w_l - \eta \frac{\partial \mathcal{R}}{\partial w_l} \right)$$

multipling out leads to terms of up to order L.

Higher order terms (in η) may be significant, despite the damping

Batch Normalization

[DL 8.7.1, Ioffe & Szegedy, 2015]

Key idea: normalize the layer activations $\Longrightarrow$ batch normalization (and backpropagate through normalization!)

- fix layer l, fix set of examples $I \subseteq [1:N]$
- mean activity, vector of standard deviation

$$\mu_j^l := \frac{1}{|I|} \sum_{i \in I} (F_j^l \circ \cdots \circ F^1)(\mathbf{x}[i]) \in \mathbb{R}^{m_l}$$

$$\sigma_j^l := \sqrt{\delta + \frac{1}{|I|} \sum_i \Big((F_j^l \circ \cdots \circ F^1)(\mathbf{x}[i]) - \mu_j \Big)^2}, \quad \delta > 0$$

- μ and σ are functions of the weights: can be differentiated

Batch Normalization

[DL 8.7.1, Ioffe & Szegedy, 2015]

Normalized activities (cf. z-score in statistics)

$$\tilde{\mathbf{x}}_j^l := \frac{\mathbf{x}_j^l - \mu_j}{\sigma_j}$$

Regain representational power

$$\tilde{\tilde{\mathbf{x}}}_j^l = \alpha_j \tilde{\mathbf{x}}_j^l + \beta_j$$

- in principle: can exactly undo the batch normalization
- however: different learning dynamics, mean activation (and scale) can now be directly controlled

Batch Normalization - implementation details

[DL 8.7.1, Ioffe & Szegedy, 2015]

- The bias term before batch normalization should be removed.
- At training time, the statistics are computed over a batch.
- At test time, μ and σ may be replaced by running averages that were collected during training time

Many more heuristics ...

- Curriculum learning and non-uniform sampling of data points $\Longrightarrow$ focus on most relevant examples [Bengio, Louradour, Collobert, Weston, 2009; DL, 8.7.6]
- Continuation methods: define a family of (simpler) objective functions and track solutions [DL, 8.7.6]
- Heuristics for initialization [DL, 8.4] and pre-training [DL, 8.7.4]

Section 2

Norm-based Regularization

Regularization in Machine Learning

[DL, Chapter 7.1-7.2]

What is regularization?

Any aspect of a learning algorithm that is intended to lower the generalization error but not the training error (cf. DL, p.228)

- Informed regularization: encode specific prior knowledge
- Simplicity bias: preference for simpler models (Occam's razor)
- Data augmentation and cross-task learning
- Model averaging, e.g. ensemble methods, drop-out

Norm-based Regularization

Standard regularization method (convex models)

$$\mathcal{R}_\Omega(\theta; \mathcal{S}) = \mathcal{R}(\theta; \mathcal{S}) + \Omega(\theta).$$

- Ω: functional that does not depend on training data

Common choice: L_2/Frobenius–norm penalty for deep networks

$$\Omega(\theta) = \frac{1}{2} \sum_{l=1}^{L} \mu^l \|\mathbf{W}^l\|_F^2, \quad \mu^l \geq 0$$

- common practice: only penalize weights, not biases
- single μ weight or one μ^l per layer

Weight Decay

Regularization based on L_2-norm is also called weight decay as

$$\frac{\partial \Omega}{\partial w_{ij}^l} = \mu^l w_{ij}^l$$

- weights in l-th layer get pulled towards zero with "gain" μ^l (in the following assume $\mu^l = \mu$, ignore biases)
- naturally favors weights of small magnitude

Gradient descent gets modified as

$$\theta(t+1) \; = \; \underbrace{(1-\mu)\cdot\theta(t)}_{\text{weight decay}} - \underbrace{\eta}_{\text{step size}} \cdot \underbrace{\nabla_\theta \mathcal{R}}_{\text{data dep.}}$$

Weight Decay: Analysis

Quadratic (Taylor) approximation of $\mathcal{R}$ around optimal θ^*

$$\mathcal{R}(\theta) \approx \mathcal{R}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^\top \mathbf{H}\, (\theta - \theta^*),$$

where $\mathbf{H}_\mathcal{R}$ is the Hessian of $\mathcal{R}$

$$\mathbf{H}_\mathcal{R} = \left(\frac{\partial^2 \mathcal{R}}{\partial \theta_i \partial \theta_j} \right), \quad \text{and} \quad \mathbf{H} := \mathbf{H}_\mathcal{R}\Big|_{\theta = \theta^*}$$

Weight Decay: Analysis

First order optimality condition

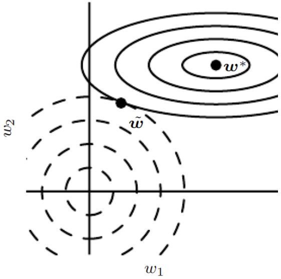
$$\begin{aligned}\nabla_\theta \mathcal{R}_\Omega \overset{!}{=} 0 &\iff \mathbf{H}(\theta - \theta^*) + \mu\theta = \mathbf{0} \\ &\iff (\mathbf{H} + \mu\mathbf{I})\,\theta = \mathbf{H}\,\theta^* \\ &\iff \theta = (\mathbf{H} + \mu\mathbf{I})^{-1}\,\mathbf{H}\,\theta^* = \mathbf{Q}\,\underbrace{(\mathbf{\Lambda} + \mu\mathbf{I})^{-1}\,\mathbf{\Lambda}}_{=\text{diag}\left(\frac{\lambda_i}{\lambda_i+\mu}\right)}\,\mathbf{Q}^\top\theta^*\end{aligned}$$

with diagonalization $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$, $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_d)$.

Weight Decay: Interpretation

Along directions in parameter space with large eigenvalues of $\mathbf{H}$, i.e. $\lambda_i \gg \mu$: vanishing effect

Along directions in parameter space with small eigenvalues of $\mathbf{H}$, i.e. $\lambda_i \ll \mu$: shrunk to nearly zero magnitude



Weight Decay: Linear Regression

Perform analysis exactly for special case: linear regression

$$\mathcal{R}_\Omega(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^\top(\mathbf{X}\theta - \mathbf{y}) + \frac{\mu}{2}\|\theta\|^2$$

for which the modified normal equations are given by

$$\begin{aligned} \nabla_\theta \mathcal{R}_\Omega(\theta) \overset{!}{=} 0 &\iff \mathbf{X}^\top(\mathbf{X}\theta - \mathbf{y}) + \mu\theta = 0 \\ &\iff \theta = \left(\mathbf{X}^\top\mathbf{X} + \mu\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y} \end{aligned}$$

- $\mu \to 0$: Moore-Penrose pseudoinverse

L1 Regularization

Sparsity inducing choice

$$\Omega(\theta) = \sum_{l=1}^{L} \mu^l \|\mathbf{W}^l\|_1 = \sum_{l=1}^{L} \mu^l \sum_{ij} |\mathbf{W}_{ij}^l|, \quad \mu^l \geq 0$$

One dimensional problem

$$w^* = \underset{w}{\operatorname{argmin}} (w_0 - w)^2 + \mu |w|$$

Using the subderivative, we can derive the soft thresholding operator

$$w^* = \mathsf{sign}(w_0)(\max\{|w_0| - \mu, 0\}$$

Regularization via Constrained Optimization

Alternative view on regularization: for given $r > 0$ solve

$$\min_{\{\theta : \|\theta\| \le r\}} \mathcal{R}(\theta)$$

Simple optimization approach: projected gradient descent

$$\theta(t+1) = \Pi_r\left(\theta(t) - \eta \nabla \mathcal{R}\right), \quad \Pi_r(\mathbf{v}) := \min\left\{1, \frac{r}{\|\mathbf{v}\|}\right\} \mathbf{v}$$

Discussion [Hinton et al, 2013]:

- constraints do not affect initial learning (small weights), only become active, once weights are large (plus!)
- constrain norm of vector of incoming weights for each unit (row norms) $\Longrightarrow$ practical success, stabilization

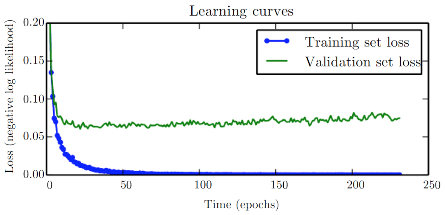
Early Stopping

Iterative methods like gradient-descent typically evolve solutions from simple and robust to complex and sensitive.

Early stopping: stop learning after finite (small) number of iterations.

Rely on validation data (hold-out) to estimate and track expected risk. Stop when flat or worsening. Keep best solution.

Conceptually easy, computationally attractive $\Longrightarrow$ high popularity



Early Stopping: Analysis

Study gradient descent trajectories: quadratic approximation

By Taylor series approximation of gradient around optimal θ^*:

$$\nabla_\theta \mathcal{R}\Big|_{\theta_0} \approx \nabla_\theta \mathcal{R}\Big|_{\theta^*} + \mathbf{J}_{\nabla \mathcal{R}}\Big|_{\theta^*}(\theta_0 - \theta^*) = \mathbf{H}(\theta_0 - \theta^*)$$

as the Jacobian of the gradient map is the Hessian. Furthermore

$$\theta(t+1) = \theta(t) - \eta \nabla_\theta \mathcal{R}\Big|_{\theta(t)} \approx \theta(t) - \eta \mathbf{H}(\theta(t) - \theta^*)$$

which yields (after subtracting θ^* on both sides)

$$\theta(t+1) - \theta^* \approx (\mathbf{I} - \eta \mathbf{H})(\theta(t) - \theta^*)$$

Early Stopping: Analysis

Change to basis that diagonalizes Hessian $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$

$$\tilde{\theta}(t+1) - \tilde{\theta}^* = (\mathbf{I} - \eta\mathbf{\Lambda})(\tilde{\theta}(t) - \tilde{\theta}^*), \quad \tilde{\theta} := \mathbf{Q}^\top\theta$$

Assuming $\theta(0) = \mathbf{0}$ and small η ($|1 - \eta\lambda_i| < 1$) one gets explicitly

$$\tilde{\theta}(t) = \left[\mathbf{I} - (\mathbf{I} - \eta\mathbf{\Lambda})^t\right]\tilde{\theta}^*$$

and thus (comparing to previous analysis) if we can chose t, η s.t.

$$(\mathbf{I} - \eta\mathbf{\Lambda})^t \overset{!}{=} \mu\left(\mathbf{\Lambda} + \mu\mathbf{I}\right)^{-1}$$

which for $\eta\lambda_i \ll 1$, $\lambda_i \ll \mu$ can be achieved approximately via performing $t = \frac{1}{\eta\mu}$ steps.
Early stopping can thus be seen as an approximate L_2-regularizer.

Early Stopping: Analysis (Details 1 of 2)

- Early stopping

$$\begin{aligned}\tilde{\theta}(t) - \tilde{\theta}^* &= (\mathbf{I} - \eta\boldsymbol{\Lambda})(\tilde{\theta}(t-1) - \tilde{\theta}^*) \\ &\overset{\text{Ind}}{=} (\mathbf{I} - \eta\boldsymbol{\Lambda})^t(\tilde{\theta}(0) - \tilde{\theta}^*) \overset{\tilde{\theta}(0)=\mathbf{0}}{=} -(\mathbf{I} - \eta\boldsymbol{\Lambda})^t\tilde{\theta}^* \end{aligned}$$

$$\implies \quad \tilde{\theta}(t) = \left[\mathbf{I} - \underbrace{(\mathbf{I} - \eta\boldsymbol{\Lambda})^t}_{:=\mathbf{A}(t)}\right]\tilde{\theta}^*$$

- Weight decay (running until convergence)

$$\begin{aligned}\lim_{t\to\infty}\tilde{\theta}(t) &= (\boldsymbol{\Lambda} + \mu\mathbf{I})^{-1}\boldsymbol{\Lambda}\tilde{\theta}^* = \text{diag}\left(\frac{\lambda_i}{\mu+\lambda_i}\right)\tilde{\theta}^* \\ &= \text{diag}\left(1 - \frac{\mu}{\mu+\lambda_i}\right)\tilde{\theta}^* = \left(\mathbf{I} - \underbrace{\mu(\boldsymbol{\Lambda} + \mu\mathbf{I})^{-1}}_{=:\mathbf{B}(\mu)}\right)\tilde{\theta}^*\end{aligned}$$

Early Stopping: Analysis (Details 2 of 2)

- Simplification (small step sizes)

$$\mathbf{A}(t) = \mathbf{I} - t\eta\mathbf{\Lambda} + \mathbf{O}([\eta\lambda_i]^2)$$

- Von-Neuman series (large regularization strength μ)

$$\left(\frac{1}{\mu}\mathbf{\Lambda} + \mathbf{I}\right)^{-1} = \sum_{k=0}^{\infty}\left(-\frac{1}{\mu}\mathbf{\Lambda}\right)^k = \mathbf{I} - \frac{1}{\mu}\mathbf{\Lambda} + \mathbf{O}\left(\left[\frac{\lambda_i}{\mu}\right]^2\right)$$

- Equating coefficients

$$t\eta = \frac{1}{\mu} \quad \Longleftrightarrow \quad t = \frac{1}{\eta\mu}$$

Section 3

Dataset Augmentation

Invariances through Virtual Examples

[DL, Chapter 7.4]

Often one knows *a priori* that inputs can be subjected to certain transformations τ without changing the target output.

Input invariances (global or local)

Generate virtual examples by applying each τ to each training example $(\mathbf{x}, \mathbf{y})$ to get $(\tau(\mathbf{x}), \mathbf{y})$.

Significant blow-up of training data
(but: can be nicely combined with SGD)

Invariances: Images

Exploiting invariances is highly domain specific.

Example: Images [Krizhevsky, Sutskever, Hinton, 2012]

- scale changes (size): cropping and resizing
- rotations and reflections (e.g. horizontal)
- adding transformations through PCA

Invariant architectures

- For some problems, it might be better to build and invariant architecture than to augment the dataset.
- Less parameters to learn $\rightarrow$ faster and more stable convergence
- Example: because convolution is equivariant to translation (translated input $\rightarrow$ translated output), a fully convolutional network with a statistical layer (mean, variance, ...) does not need translation augmentation.

Injection of Noise

Various schemes to inject noise during training of deep networks.

- adding noise to inputs: ideally realistic noise (e.g. background noise in acoustic or image processing)
- adding noise to weights: regularizing effect – find weights where small perturbations have small effects on outputs
- adding noise to targets: soft targets (e.g. probability distributions over labels, robustness w.r.t. label errors)

Semi-supervised Training

[DL, Chapter 7.6]

Typical setting: more unlabeled data than labeled data.

Define a generative model with a corresponding log-liklihood.

Optimize additive combination of supervised and unsupervised risk, while sharing parameters.

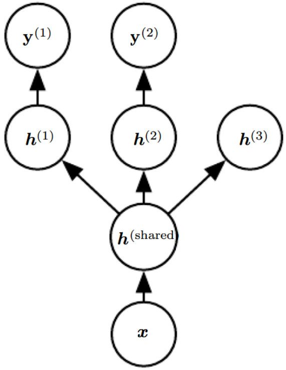
Cheaper (but generally less effective) approach: pre-training

Multi-Task Learning

[DL, Chapter 7.7]

Share representations across tasks and learn jointly (i.e. minimize combined objective).

- typical architecture: share low level representations, learn high level representations per task
- sometimes even task-specific linear layers are sufficient



Section 4

Dropout

Ensemble Methods: Bagging

[DL, Chapter 7.11]

Bagging: Ensemble method that combines models trained on bootstrap samples.

- bootstrap sample $\tilde{\mathcal{S}}_N^k$: sample N times from $\mathcal{S}_N$ with replacement for $k = 1, \ldots, K$
  (many duplicates, on average $\approx 2/3$ distinct examples)
- train model on $\tilde{\mathcal{S}}_N^k \longrightarrow \theta^k$.
- prediction: average model output probabilities $p(\mathbf{y}|\mathbf{x}; \theta^k)$

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} p(\mathbf{y}|\mathbf{x}; \theta^k)$$

Dropout

[DL, Chapter 7.12]

Dropout idea: randomly "drop" subsets of units in network. [Hinton at al, 2012]

More precisely, define "keep" probability π_i^l for unit i in layer l.

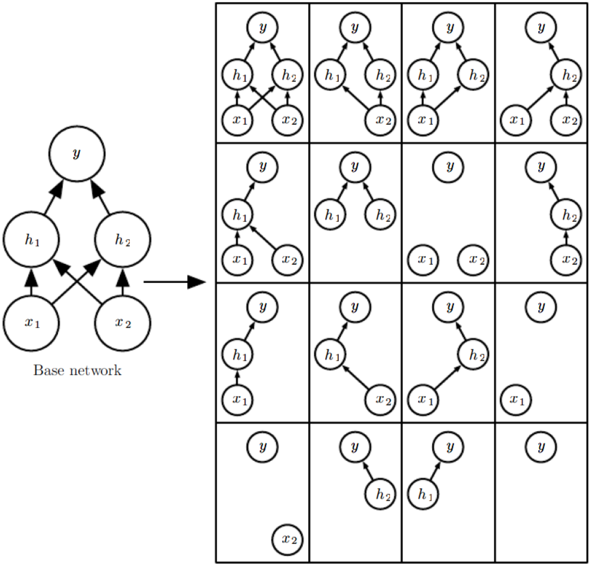
- typically: $\pi_i^0 = 0.8$ (inputs) and $\pi_i^{l \geq 1} = 0.5$ (hidden units)
- realization: sampling bit mask and zeroing out activations
- effectively defines an exponential ensemble of networks (each of which is a sub-network of the original one)
- all models share same weights
- standard backpropagation applies

Dropout: Motivation

... "overfitting" is greatly reduced by randomly omitting half of the feature detectors on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate.

(Hinton et al. 2012)

Dropout Ensembles



Base network

Ensemble of Sub-Networks

Dropout Ensembles

Dropout realizes an ensemble

$$p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{Z}} p(\mathbf{Z}) p(\mathbf{y}|\mathbf{x};\mathbf{Z}),$$

where $\mathbf{Z}$ denotes the binary "zeroing" mask.
(note that $p(\mathbf{Z})$ is defined via probabilities π_i^l)

Gradient based learning: ok, as unbiased gardient estimates
(provided $\mathbf{Z}$ is sampled according to $p(\mathbf{Z})$).

Prediction: no analytic solution for deep networks known.
Practically: sample $\mathbf{Z}$ and average results ($\approx 10 - 20$ repetitions)

Weight Rescaling

Simple, often effective heuristic (to avoid 10-20x sampling blowup):

Scale each weight w_{ij}^l by probability of unit j being active,

$$\tilde{w}_{ij}^l \leftarrow \pi_j^{l-1} w_{ij}^l$$

Makes sure, net input to unit i is calibrated, i.e.

$$\sum_j \tilde{w}_{ij}^l x_j \overset{!}{=} \mathbf{E}_{\mathbf{Z}} \sum_j z_j^{l-1} w_{ij}^l x_j = \sum_j \pi_j^{l-1} w_{ij}^l x_j \quad \checkmark$$

Can be shown to be an approximation (sometimes exact) to a geometrically averaged ensemble (see [DL, 7.12]).