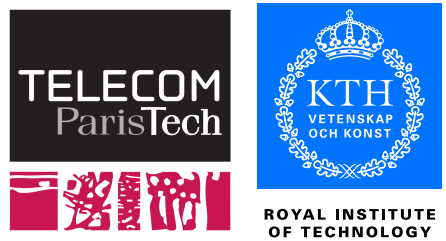


Deep Learning

Lecture 6: Large-scale Training for Deep Neural Networks

University of Agder,
Kristiansand, Norway

Prepared by: Hadi Ghauch[⋆], Hossein S. Ghadikolaei[†]

[⋆] Telecom ParisTech

[†] Royal Institute of Technology, KTH

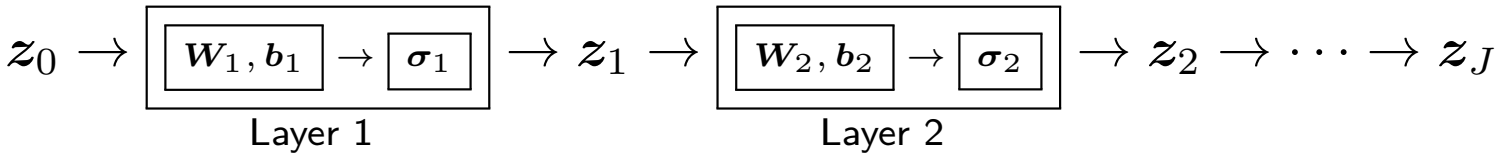
`https://sites.google.com/view/fundl/home`

Outline

1. Challenges for DNN optimization
2. Large-Scale methods for DNN training
3. Regularization in DNNs
4. Some tricks of the trade

Recap: Deep Neural Networks (DNNs)

DNN with J-layers: cascade of J filters + activation



In compact form: $\boldsymbol{z}_j = \boldsymbol{\sigma}_j(\boldsymbol{W}_j \boldsymbol{z}_{j-1} + \boldsymbol{b}_j)$

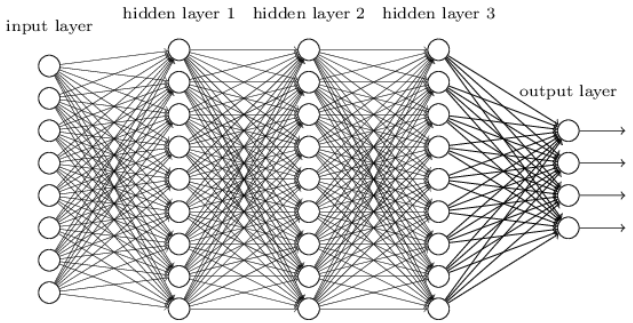
$\boldsymbol{z}_j \in \mathbb{R}^{d_j}$: input to layer j

$\boldsymbol{W}_j \in \mathbb{R}^{d_j \times d_{j-1}}$: **Weights** for layer j (drop bias for simplicity)

$\boldsymbol{\sigma}_j : \mathbb{R}^{d_j} \mapsto \mathbb{R}^{d_j}$: **Activation func** for layer j
$[\boldsymbol{\sigma}_j(\boldsymbol{u})]_i = \boldsymbol{\sigma}_j([\boldsymbol{u}]_i),\ \boldsymbol{u} \in \mathbb{R}^d,\ \ \forall i \in [N]$

Recap: DNN training and non-convex optimization

The loss for sample $(\boldsymbol{x}_i, \boldsymbol{y}_i)$:
$\|\boldsymbol{y}_i - \sigma_J(\boldsymbol{W}_J \cdots \sigma_2(\boldsymbol{W}_2\sigma_1(\boldsymbol{W}_1\boldsymbol{x}_i)))\|_2^2, i \in [N]$



Let $\boldsymbol{w} \in \mathbb{R}^d$ be the total number of weights in DNN (from all layers)
$d \geq 10^6$ in current deep learning application

The resulting DNN training problem:

$$\min_{\boldsymbol{w}\in\mathcal{D}} \frac{1}{N}\sum_{i=1}^{N} f_i((\boldsymbol{x}_i, \boldsymbol{y}_i); \boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 := f(\boldsymbol{w})$$

non-convex optimization problem

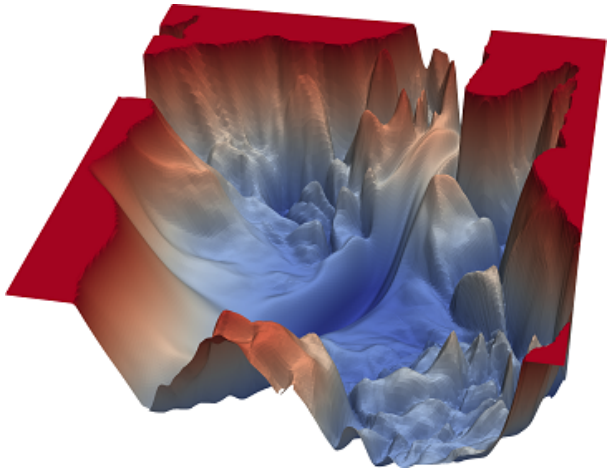
Outline

1. Challenges for DNN optimization
2. Large-Scale methods for DNN training
3. Regularization in DNNs
4. Some tricks of the trade

Loss Surface of DNNs

DNN training in fully-parametric form:

$$\arg\min_{\boldsymbol{w}} \; f(\boldsymbol{w}) := \sum_{i=1}^{N} f_i(\boldsymbol{w}) \qquad (1)$$

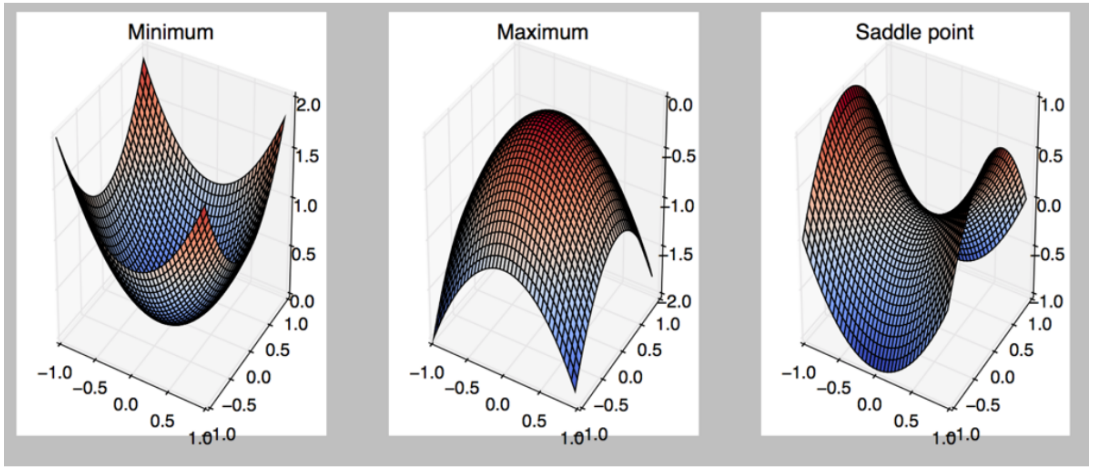


- $\boldsymbol{w} \in \mathbb{R}^d$: **all weights in DNN** ($d \gg 1$)
- $f()$: **regularized loss function**, $f_i()$: loss func for sample $i \in [N]$
- $f()$: non-convex for general DNN training problems
- $\boldsymbol{g}^{(k)} := \nabla_{\boldsymbol{w}} \; f(\boldsymbol{w}^{(k)}) \in \mathbb{R}^d$: gradient at $\boldsymbol{w}^{(k)}$
- $\boldsymbol{H}^{(k)} := \nabla^2_{\boldsymbol{w}} \; f(\boldsymbol{w}^{(k)}) \in \mathbb{R}^{d \times d}$: Hessian at $\boldsymbol{w}^{(k)}$
- $\{ \; \lambda_i[\boldsymbol{H}^{(k)}] \; \}_{i=1}^{d}$: eigenvalues of $\boldsymbol{H}^{(k)}$

Loss Surface of DNNs

Only first-order methods used in training
due to complexity of optimization

- first-order methods converge to a **stationary point**
- Recall definition of **stationary point**: $\boldsymbol{w}$ is stationary $\Leftrightarrow \nabla f(\boldsymbol{w}) = \mathbf{0}$
- Problematic in non-convex setting



All positive eigenvalues | All negative eigenvalues | Some positive and some negative

- Particularly problematic for DNN training
  Because loss surface exponentially many saddle points

Loss Surface of DNNs

How many saddle points or local minima ?

- Eigenvalues of $\boldsymbol{H}$ are Bernoulli distributed:
- $\lambda_i[\boldsymbol{H}^{(k)}] > 0$ with prob p (independently of other ones).
- local min $\Leftrightarrow$ all eigenvalues of Hessian are positive.
  Probability of having a local min: $\propto p^d$ (where $d \gg 1$)
- local minima **exponentially less likely** to occur than saddle points
- argument based on random matrix theory

Almost all stationary pts are saddle

- Result validated empirically

However, most minima give a 'good enough' loss value

Analysis of Gradient Descent

Improvement in f with one GD step? $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha^{(k)}\boldsymbol{g}^{(k)}$
- intractable in closed form: use 2nd order Taylor approx at $\boldsymbol{w}^{(k)}$

$$f(\boldsymbol{w}^{(k)} - \alpha^{(k)}\boldsymbol{g}^{(k)}) \approx f(\boldsymbol{w}^{(k)}) - \alpha^{(k)}\|\boldsymbol{g}^{(k)}\|_2^2 + \frac{(\alpha^{(k)})^2}{2}(\boldsymbol{g}^{(k)})^T\boldsymbol{H}^{(k)}\boldsymbol{g}^{(k)}$$

Use Taylor approx to find **decay after one GD step:**

$$\Delta^{(k)} := f(\boldsymbol{w}^{(k+1)}) - f(\boldsymbol{w}^{(k)}) \approx \underbrace{-\alpha^{(k)}\|\boldsymbol{g}^{(k)}\|_2^2}_{\text{decay from grad. norm}} \underbrace{+\frac{(\alpha^{(k)})^2}{2}(\boldsymbol{g}^{(k)})^T\boldsymbol{H}^{(k)}\boldsymbol{g}^{(k)}}_{\text{depends on alignment of } \boldsymbol{g}^{(k)}, \boldsymbol{H}^{(k)}}$$

Conditions on $\Delta^{(k)}$: $\Delta^{(k)} \leq 0$ and $|\Delta^{(k)}|$ as **large as possible**. why ?

- balance b/w term decays cost (depends on $\|\boldsymbol{g}^{(k)}\|_2^2$), and increase or decay (depend on $\|\boldsymbol{g}^{(k)}\|_2^2$ and $\lambda_{\min}[\boldsymbol{H}^{(k)}]$)
- one way to satisfy conditions on $\Delta^{(k)}$ is pick **stepsize optimally**: $\alpha^{(k)^\star} = \|\boldsymbol{g}^{(k)}\|_2^2/(\boldsymbol{g}^{(k)})^T\boldsymbol{H}^{(k)}\boldsymbol{g}^{(k)}$
  not used in practice (need Hessian): sequence $\alpha^{(k)}$ selected in advance

Ill-conditioning

Ill-Conditioning: when curvature of f not even across dim
- large difference b/w eigenvalues of $\boldsymbol{H}^{(k)}$
- sensitivity of loss fn different across features

Different perspective on ill-conditioning for **non-convex** f

Mathematically ill-conditioning happens:

$$\underbrace{\frac{(\alpha^{(k)})^2}{2}(\boldsymbol{g}^{(k)})^T\boldsymbol{H}^{(k)}\boldsymbol{g}^{(k)}}_{\text{increase/decay from alignment of } \boldsymbol{g}^{(k)},\boldsymbol{H}^{(k)}} \geq \underbrace{\alpha^{(k)}\|\boldsymbol{g}^{(k)}\|_2^2}_{\text{decay from grad. norm}}$$

recall $f(\boldsymbol{w})$ non-convex: $\lambda_i[\boldsymbol{H}^{(k)}]$ may take any value
$\Rightarrow \Delta^{(k)} \geq 0 \Rightarrow f(\boldsymbol{w}^{(k)}) \leq f(\boldsymbol{w}^{(k+1)})$: GD step **increases the cost**
ill-conditioning more harmful in non-convex setting

Preconditioning

Mitigate ill-conditioning ?

most DNN training methods find a **direction**, $\boldsymbol{d}^{(k)}$, to move the solution

then they **scale each coordinate** by a single factor,$\alpha^{(k)}$

$$\begin{bmatrix} w_1^{(k+1)} \\ w_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \end{bmatrix} - \alpha^{(k)} \begin{bmatrix} d_1^{(k)} \\ d_2^{(k)} \end{bmatrix}$$

true for methods seen so far: GD, SGD, momentum, etc.

Preconditioning: scale coordinates in direction differently

$$\begin{bmatrix} w_1^{(k+1)} \\ w_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \end{bmatrix} - \alpha^{(k)} \begin{bmatrix} p_1^{(k)} & p_2^{(k)} \\ p_3^{(k)} & p_4^{(k)} \end{bmatrix} \begin{bmatrix} d_1^{(k)} \\ d_2^{(k)} \end{bmatrix}$$

$$\begin{bmatrix} w_1^{(k+1)} \\ w_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \end{bmatrix} - \alpha^{(k)} \begin{bmatrix} p_1^{(k)} \\ p_3^{(k)} \end{bmatrix} \circ \begin{bmatrix} d_1^{(k)} \\ d_2^{(k)} \end{bmatrix}$$

adapt stepsize using more info than gradient (w/out Hessian)

Outline

1. Challenges for DNN optimization
2. Large-Scale methods for DNN training
3. Regularization in DNNs
4. Some tricks of the trade

Stochastic Gradient Descent (SGD)

Recall that computing full-gradient (all samples) is too expensive

SGD: Do gradient step for one training sample

- Pick i uniformly at random from $[N]$
- Compute SG w.r.t. sample i: $\widehat{\boldsymbol{g}}_i^{(k)} := \nabla\ f_i(\boldsymbol{w}^{(k)})$
- Update weights: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha^{(k)}\ \widehat{\boldsymbol{g}}_i^{(k)}$

where $\nabla\ f_i()$ gradient of f_i, k iteration index, $\alpha^{(k)}$ stepsize

Pros: low computational footprint

Cons: estimating full-gradient $\nabla\ f$, using gradient of sample i, $\nabla\ f_i$. Too noisy!

Mini-batch SGD

Solution to noisy gradient estimate in SGD

Mini-batch SGD: gradient over batch of training samples

- Pick batch of samples, $\mathcal{B}$, uniformly at random from $[N]$
- Compute SG w.r.t. batch $\mathcal{B}$: $\widehat{\boldsymbol{g}}_i^{(k)} := \sum_{i \in \mathcal{B}} \nabla f_i(\boldsymbol{w}^{(k)})$
- Update weights: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha^{(k)} \widehat{\boldsymbol{g}}_i^{(k)}$

Improves the gradient estimate wrt SGD

$\mathcal{B} = i$ recovers SGD, and $\mathcal{B} = [N]$ recovers full GD

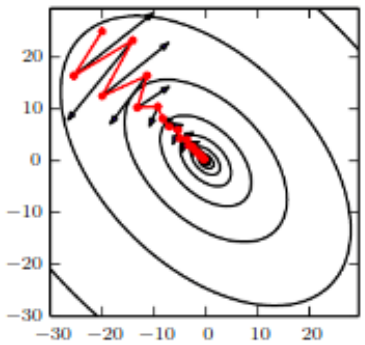
Pros: Changing $\mathcal{B}$ trades-off accuracy of gradient estimation for complexity

Momentum Methods

SGD with momentum

- Pick i uniformly at random from $[N]$
- Compute SG w.r.t. sample i: $\widehat{\boldsymbol{g}}_i^{(k)} := \nabla\ f_i(\boldsymbol{w}^{(k)})$
- Update momentum: $\boldsymbol{t}_i^{(k+1)} = \eta \widehat{\boldsymbol{g}}_i^{(k)}, \quad 0 < \eta < 1$
- Update weights: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha^{(k)} \widehat{\boldsymbol{g}}_i^{(k)} - \boldsymbol{t}_i^{(k+1)}$

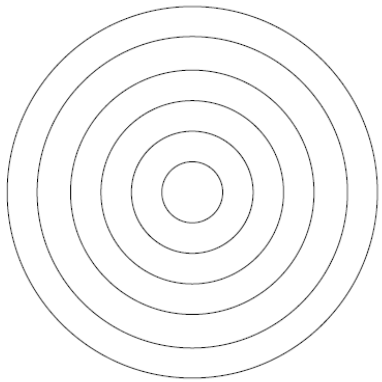
Faster conv (wrt SGD) when loss has
- high curvature and $\widehat{\boldsymbol{g}}$ too noisy
- norm of $\widehat{\boldsymbol{g}}$ vary across surface
- momentum 'smoothes-out' variation in $\widehat{\boldsymbol{g}}$
- similar behavior as Nesterov and heavy ball methods (lec 2)



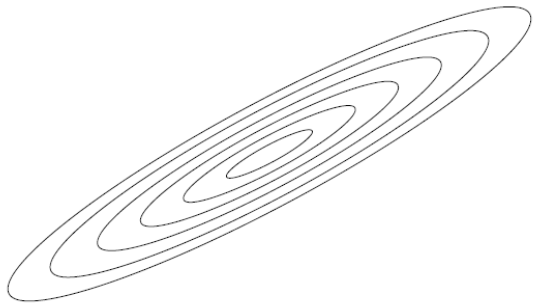
Choice of Stepsize

Stepsize, $\alpha^{(k)}$, aka, **learning rate** in backpropagation

- chosen apriori **independently of gradient/Hessian**
- examples: $\alpha^{(k)} = c/\sqrt{k}$, $\alpha^{(k)} = c/k$, $\alpha^{(k)} = c(c << 1)$
- Problematic when **curvature is uneven**



$\lambda_1 = \lambda_2$ (ideal)



$\lambda_1 >> \lambda_2$ (bad for GD)

- Stepsize should be adapted to compensate for uneven curvature, **w/out using Hessian**:
  idea behind **Adaptive rate methods**

Adaptive Gradient (ADAGRAD)

Adapt stepsize using norm of all previous gradients
Intuition: Shrink stepsize for sensitive features (high grad norm)

ADAGRAD:

- Pick i uniformly at random from $[N]$
- Compute SG w.r.t. sample i: $\widehat{\boldsymbol{g}}_i^{(k)} := \nabla\ f_i(\boldsymbol{w}^{(k)})$
- Compute all previous norms of SG: $\boldsymbol{u}_i^{(k+1)} = \boldsymbol{u}_i^{(k)} + \widehat{\boldsymbol{g}}_i^{(k)} \circ \widehat{\boldsymbol{g}}_i^{(k)}$
  // $\boldsymbol{u}_i^{(k)}$ is running a sum of all previous squared grad norms
- Update weights: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left((\boldsymbol{u}_i^{(k)})^{1/2} + \delta\mathbf{1}\right)^{-1} \circ\ \widehat{\boldsymbol{g}}_i^{(k)}$
  // $\left((\boldsymbol{u}_i^{(k)})^{1/2} + \delta\mathbf{1}\right)^{-1}$: behaves like 'stepsize' (not strictly though)

$(-)^{1/2}$ and $(-)^{-1}$ are element-wise operations.
Stepsize inversely prop to **gradient norm**. why ?
weight update step is form of **preconditioning**. why?

Limitations: Stepsize decays too much due to scaling with sum of all prev grad ; add exponential decay to gradients $\Rightarrow$ **RMSProp**

RMSProp

Intuition: add exponentially decaying sum of grad norm

RMSProp:

- Pick i uniformly at random from $[N]$
- Compute SG w.r.t. sample i: $\widehat{\boldsymbol{g}}_i^{(k)} := \nabla\ f_i(\boldsymbol{w}^{(k)})$
- Exp decaying sum of prev SGs: $\boldsymbol{u}_i^{(k+1)} = \beta \boldsymbol{u}_i^{(k)} + (1-\beta)\widehat{\boldsymbol{g}}_i^{(k)} \circ \widehat{\boldsymbol{g}}_i^{(k)}$
  // $\boldsymbol{u}_i^{(k)}$ *is exponentially decaying sum of all previous squared grad norms*
- Update weights: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left((\boldsymbol{u}_i^{(k)})^{1/2} + \delta \mathbf{1}\right)^{-1} \circ\ \widehat{\boldsymbol{g}}_i^{(k)}$

$(-)^{1/2}$ and $(-)^{-1}$ are element-wise operations.

Difference with ADAGRAD: exponentially decreasing sum for all the gradients.
put **less emphasis on past values** of gradient norm
weight update step is a form of **preconditioning**

add **bias correction** for first/second moments $\Rightarrow$ ADAM

Adaptive Moments (ADAM)

Intuition: $\widehat{\boldsymbol{g}}_i^{(k)}$ may be 'bad' estimator of true gradient
approximate bias for FoM/SoM of $\widehat{\boldsymbol{g}}_i^{(k)}$, and **compensate** for it

ADAM:

- pick i uniformly at random from $[N]$
- compute SG w.r.t. sample i: $\widehat{\boldsymbol{g}}_i^{(k)} := \nabla\ f_i(\boldsymbol{w}^{(k)})$
- approximate bias for FoM: $\boldsymbol{u}_i^{(k+1)} = \beta_1 \boldsymbol{u}_i^{(k)} + (1-\beta_1)\widehat{\boldsymbol{g}}_i^{(k)}$
  // exponentially decaying sum of prev SGDs
- approximate bias for SoM: $\boldsymbol{v}_i^{(k+1)} = \beta_2 \boldsymbol{v}_i^{(k)} + (1-\beta_2)\widehat{\boldsymbol{g}}_i^{(k)} \circ \widehat{\boldsymbol{g}}_i^{(k)}$
  // exponentially decaying sum of prev SGD norms
- compute bias term of FoM: $\widehat{\boldsymbol{u}}_i^{(k+1)} = \boldsymbol{u}_i^{(k+1)}/(1-\beta_1^k)$
- compute bias term of SoM: $\widehat{\boldsymbol{v}}_i^{(k+1)} = \boldsymbol{v}_i^{(k+1)}/(1-\beta_2^k)$
- update weights: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left((\widehat{\boldsymbol{v}}_i^{(k)})^{1/2} + \delta \mathbf{1} \right)^{-1} \circ \widehat{\boldsymbol{u}}_i^{(k)}$

Difference? move along **updated FoM bias**, $\widehat{\boldsymbol{u}}_i^{(k)}$, not the SG.
precondition using updated SoM bias $\widehat{\boldsymbol{v}}_i^{(k)}$.

Outline

1. Challenges for DNN optimization
2. Large-Scale methods for DNN training
3. Regularization in DNNs
4. Some tricks of the trade

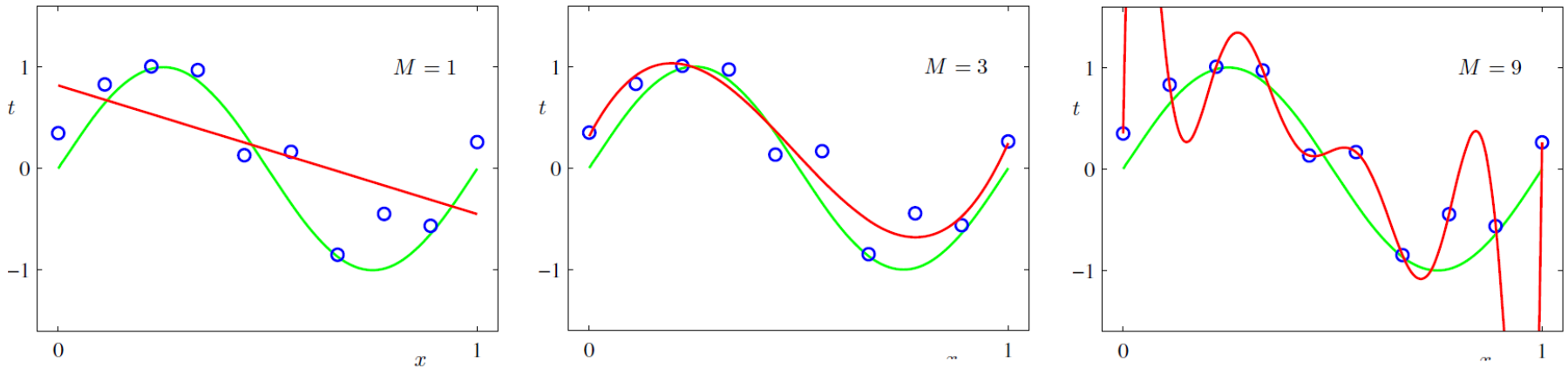
Regularization in DNNs

Regularization is critical in DDN:

- huge number of parameters: 1 Billion weights in modern DNN
- need to avoid *overfitting*

Reduce **test error** at the expense of higher **training error**

- trading increased bias for reduced variance of the estimator (Lec 2)
- reduce the model complexity to match the data-generating process



source: C. Bishop, 2006

Regularization: Parameter norm penalty

Regularization via **Parameter norm penalty**

- $\min_{\boldsymbol{w}} \tilde{f}(\boldsymbol{w}) := f(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_p$, $p \in \{1, 2\}$
- $f()$: loss function (from ERM problem)
- $\tilde{f}()$: regularized loss function
- no regularization needed for biases, $\boldsymbol{b}_j$

In deep learning:

- common to start with **overpametrized model** (lots of weight)
- and gradually increase **regularization parameter**, λ
- **hyperparameter tuning** done using **cross-validation**. how?
  grid search over set of possbile values for λ
  pick value for λ that minimizes **test error (over test set)**
  see also K-fold cross validation

Regularization: Parameter norm penalty

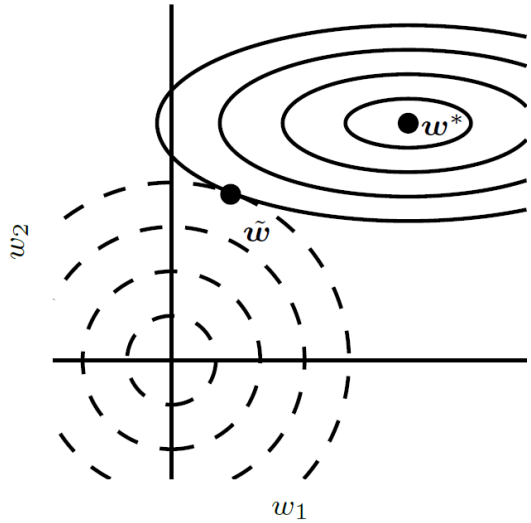
Weight Decay: Tikhonov regularization $(p = 2)$

- $\boldsymbol{w}^\star = \arg\min \ f(\boldsymbol{w})$, $\tilde{\boldsymbol{w}} = \arg\min \ \tilde{f}(\boldsymbol{w})$,

Effect of weight decay on solution ?

- if $\lambda_i[\nabla^2 f(\boldsymbol{w}^\star)] \gg \lambda$: direction i sensitive
  $\tilde{w}_i \approx w_i^\star$: no change in sol
- if $\lambda_i[\nabla^2 f(\boldsymbol{w}^\star)] \ll \lambda$: direction i not sensitive
  $\tilde{w}_i \approx 0$: shrink component to zero

no change in solution along sensitive direction.
lower model complexity: $\|\tilde{\boldsymbol{w}}\|_2 \leq \|\boldsymbol{w}^\star\|_2$



solid line $= f$, dashed line $= r$. $\tilde{w}$ closer to origin (lower magnitude) than $w^\star$. Reg pushes optimal solution closer to 0 (lower norm)

Regularization: Parameter norm penalty

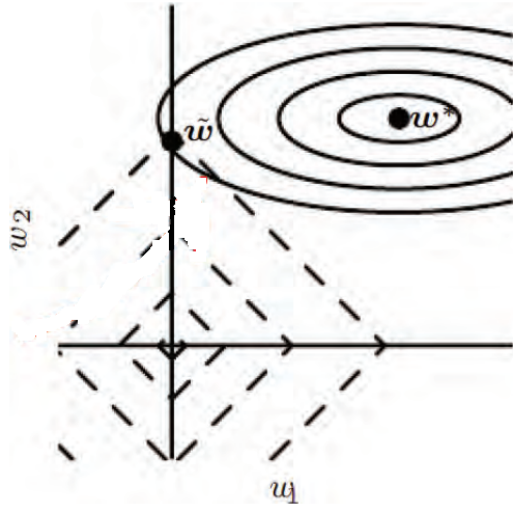
L_1-**regularization:** (sparse regularization)

- $\boldsymbol{w}^\star = \arg\min\ f(\boldsymbol{w})$, $\tilde{\boldsymbol{w}} = \arg\min\ \tilde{f}(\boldsymbol{w})$,

Effect on solution ?

- if $w_i^\star \leq \lambda / [\nabla^2\ f(\boldsymbol{w}^\star)]_{i,i}$: direction i not sensitive. set $\tilde{w}_i = 0$ to zero
- if $w_i^\star \geq \alpha / [\nabla^2\ f(\boldsymbol{w}^\star)]_{i,i}$: direction i sensitive. set $\tilde{w}_i = w_i^\star$ (no change)

Round $w_i^\star$ to zero if below threshold



solid line $= f$, dashed line $= r$. Due to shape of ℓ_1 norm, $\tilde{w}$ has one non-zero component. Reg makes solution sparse.

L_1-regularization gives *sparse* solutions.
- Insignificant components of model (small entries in $\boldsymbol{w}$) set to zero

Overfitting

- *Data Augmentation:* Avoid overfitting with more data in training set

  apply transformations on training set to get more data
- *Early stopping:* evaluate training and test error with each epoch.

  return model with lower test error
- *Dropout training:* drop weights in DNN following a distribution

see chapter 7 [Goodfellow, 2016] for detailed treatment of regularization

Outline

1. Challenges for DNN optimization
2. Large-Scale methods for DNN training
3. Regularization in DNNs
4. Some tricks of the trade

Batch normalization

Input normalization to bring all the features on the same scale

improves learning

Why not applying this idea to every layer? two new parameters to learn per layer (γ_j and β_j), same training pipeline, almost same backprop.

Batch normalization for layer j [Ioffe-Szegedy-2015]

Consider a minibatch of size N_j, $\mathcal{B} = \{\boldsymbol{z}_j^{[i]}\}_{i \in [N_j]}$

Compute batch mean $m_j^{[\mathcal{B}]}$ and batch standard deviation $\Sigma_j^{[\mathcal{B}]}$

$$m_j^{[\mathcal{B}]} = \frac{1}{N_j} \sum\nolimits_{i \in \mathcal{B}} \boldsymbol{z}_j^{[i]} \quad , \quad \Sigma_j^{[\mathcal{B}]} = \frac{1}{N_j} \sum\nolimits_{i \in \mathcal{B}} \left(\boldsymbol{z}_j^{[i]} - m_j^{[\mathcal{B}]} \right)^2$$

For every $i \in \mathcal{B}$, normalize the outputs by $\boldsymbol{z}_{\text{norm},j}^{[i]} = \dfrac{\boldsymbol{z}_j^{[i]} - m_j^{[\mathcal{B}]}}{\sqrt{\Sigma_j^{[\mathcal{B}]} + \epsilon}}$

Use the new output: $\hat{\boldsymbol{z}}_j^{[i]} = \gamma_j \boldsymbol{z}_{\text{norm},j}^{[i]} + \beta_j$

Tricks of the trade

Initialization of algo critical in *non-convex opt* (unlike convex). Why ?
- *Heuristics* (verified empirically). No principled approach for DNNs

Parameter Initialization Strategies for Weight

Recall model for layer j: $\boldsymbol{z}_j = \boldsymbol{\sigma}_j(\boldsymbol{W}_j \boldsymbol{z}_{j-1})$, where $\boldsymbol{W}_j \in \mathbb{R}^{d_j \times d_{j-1}}$

1. *Random Init*: $[\boldsymbol{W}_j]_{i,k} \sim U(\frac{-1}{\sqrt{d_{j-1}}}, \frac{1}{\sqrt{d_{j-1}}})$
2. *Xavier Init (2015)*: $[\boldsymbol{W}_j]_{i,k} \sim U(\frac{-6}{\sqrt{d_j d_{j-1}}}, \frac{6}{\sqrt{d_j d_{j-1}}})$
   - takes into account matrix mult (w/out non-linearity)
3. *Saxe Init (2013)*: Select weight as random orthogonal matrices.
   - increase norm of activation fnc (avoid vanishing grad)
4. *Martens Init (2010)*: Initial weights are sparse.

Conclusions

large scale training for DNNs

- Challenges:
  non-convex loss, exponential number of saddle points, ill and poor conditioning
- SoA Methods:
  SGD, Nesterov, minibatch SGD, ADAGRAD, RMSProp, ADAM
- Regularization: weight decay, sparse
- Heuristics: batch normalization, initialization
- Many SoA methods not covered:
  Nesterov acceleration, Polyak averaging, Dropout training,

Some references

A. Goodfellow, Y. Bendgio, A Courville, “Deep Learning”, MIT Press, Chap. 7,8

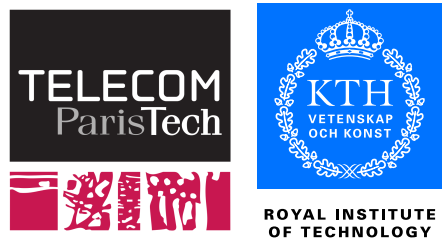
Y. LeCun, L Bottou, G. Orr, K-B Muller, “Efficient Backprop”, Neural Networks: Tricks of the trade

C. Bishop, “Pattern recognition and machine learning”, 2006, Springer, Chap. 1

S. Trivedi, R. Kondor, CMSC 35246 Deep Learning, spring 2017

S. Bach, Stochastic gradient descentand robustness to ill-conditioning, May 2016

L. Bottou, F. E. Curtis, J Norcedal, “Optimization Methods for large-scale machine learning ”, Sec. 2-3



Deep Learning

Lecture 6: Large-scale Training for Deep Neural Networks

University of Agder,
Kristiansand, Norway

Prepared by: Hadi Ghauch[⋆], Hossein S. Ghadikolaei[†]

[⋆] Telecom ParisTech

[†] Royal Institute of Technology, KTH

`https://sites.google.com/view/fundl/home`