Deep Learning

Lecture 6: Fundamentals of Deep Neural Networks

University of Agder,
Kristiansand, Norway

Prepared by: Hadi Ghauch*, Hossein S. Ghadikolaei†

* Telecom ParisTech

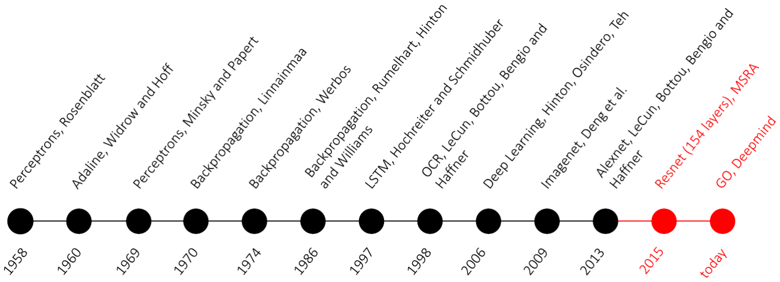
† Royal Institute of Technology, KTH

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

Outline

1. Mathematical Models and Taxonomy
2. Statistical Learning and DNNs
3. BackPropagation
4. Practical Issues

Brief History



Source: Efstratios Gavves (UVA), Deep Learning, 2015

Basics of NN

Percepteron: atom of NNs

Linear mapping + activation function σ

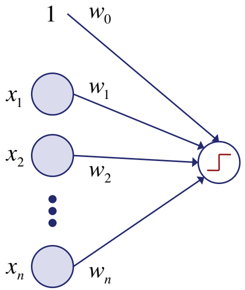
One weight per input

Output: $z = \sigma(\boldsymbol{w}^T \boldsymbol{x})$

Binary classification:

$\sigma(t) = 1$ if $t \geq 0$, and 0 otherwise

Other examples of σ: sigmoid, tanh, ReLU, ...



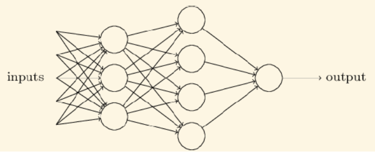
percepteron

Percepteron = Linear regression + activation function

nonlinearity σ is important

Basics of Neural Networks

Neural Network: Networks of percepterons (neurons)

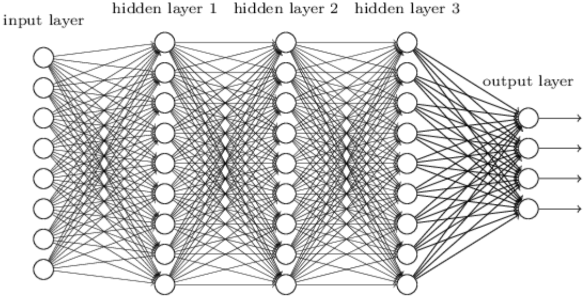


- Each Neuron has activation function: $\sigma()$
  sigmoid, tanh, ReLU, etc.

Basics of Neural Networks (NN)

Multi-layer Percepteron:

- Output from some perceptrons are used in the inputs to other perceptrons



$\boldsymbol{W}_1$ $\boldsymbol{W}_2$ $\boldsymbol{W}_3$ Output

Deep Neural Network (DNN):

- Multi-layer Percepteron with 'many' hidden layers

Outline

Mathematical Models

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

$$\boldsymbol{z}_0 \rightarrow \underset{\text{Layer 1}}{\boxed{\boxed{\boldsymbol{W}_1, \boldsymbol{b}_1} \rightarrow \boxed{\boldsymbol{\sigma}_1}}} \rightarrow \boldsymbol{z}_1 \rightarrow \underset{\text{Layer 2}}{\boxed{\boxed{\boldsymbol{W}_2, \boldsymbol{b}_2} \rightarrow \boxed{\boldsymbol{\sigma}_2}}} \rightarrow \boldsymbol{z}_2 \rightarrow \cdots \rightarrow \boldsymbol{z}_J$$

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)

$\boldsymbol{\sigma}_j() \in \mathbb{R}^{d_j \times d_j}$: **Activation func** for layer j

$\boldsymbol{\sigma}_j()$ is element-by-element func, assumed differentiable

A DNN is a composition of **non-linear layers**, $\boldsymbol{W}_1, ..., \boldsymbol{W}_J$.

DNN output for input sample $\boldsymbol{x}_i$?

$$\sigma_J(\boldsymbol{W}_J \cdots \sigma_2(\boldsymbol{W}_2 \sigma_1(\boldsymbol{W}_1 \boldsymbol{x}_i))), \ i \in [N]$$

Mathematical Models

Another way to abstract the DNN operation:
a composition of J non-linear operators, $\{\boldsymbol{f}_j\}_{j=1}^{J}$

- $\boldsymbol{f}_j() \in \mathbb{R}^{d_j \times d_{j-1}},\ j \in [J]$: **Model for layer** j
- $\boldsymbol{f}() \in \mathbb{R}^{d_J \times d_0}$: **Model** for the entire DNN
- The model, $\boldsymbol{f}$, is composition of func of each layer $\boldsymbol{f}_1, ..., \boldsymbol{f}_J$

DNN output (or prediction) for sample $\boldsymbol{x}_i$:

$$\boldsymbol{f}_J(\cdots \boldsymbol{f}_2(\boldsymbol{f}_1(\boldsymbol{x}_i)) \quad) := \boldsymbol{f}(\boldsymbol{x}_i),\ \forall i \in [N]$$

$\boldsymbol{f}$: **input-output relation** that models action of the DNN
$\boldsymbol{f}$: model that we learn using training set

Taxonomy of Neural Networks

Mathematical model recovers many DDNs variants

Feed-forward NN:
only forward connections in each layer

Convolutional NN:
$\boldsymbol{W}_j$ is circulant matrix: $\boldsymbol{W}_j \boldsymbol{z}_{j-1}$ is convolution of $\boldsymbol{z}_{j-1}$

Fully-Connected NN:
Every input connected to every hidden layer (no zeros in $\boldsymbol{W}_j$)

Recurrent NN:
allow for loops in network

Deep Linear Networks:
Activation fncs are set to identity function ($\boldsymbol{\sigma}_j = \boldsymbol{I}$)

Outline

Statistical Learning Perspective on DNNs

Single layer NN with non-linear activation,

- can approximate $\boldsymbol{f}$ with arbitrarily small error
- if size of layer is large enough (Hornik '89, Cybenko '90)

Why Multilayer NN ?

- Training algorithm may not learn $\boldsymbol{f}$
- Algorithm chooses wrong fnc: overfitting (Lec 7)

NP-hardness of the training [Blum-Rivest-89]

- Training a 3-node NN ($\boldsymbol{W}_1 \in \mathbb{R}^{2\times d}, \boldsymbol{W}_2 \in \mathbb{R}^{1\times 2}$) is NP-complete
- Does does not negate universal approximation result (violates assumption "size of each layer is large enough")
- Limited resources: insufficient data (overfit)

Statistical Learning Perspective on DNNs

How many layers needed ?

- Barron '93: Bounds on size of layer (too loose)
- Number/Size of layers found experimentally (cross-validation)

The role of non-linearities:

- Montufar etal '14: DNN with piecewise linear activation can represent functions $\boldsymbol{f}$, having exponentially many regions with the number of layers.
- More layers $\Rightarrow$ more representation power
  - if overfitting can be prevented

DNNs assume that the model, $\boldsymbol{f}$, is a **composition of several simple fncs**, $\boldsymbol{f}_j$
captured by the **layered architecture**

Training DNNs

Training samples: $\{(\boldsymbol{x}_1, \boldsymbol{y}_1) \cdots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, $\boldsymbol{x}_i \in \mathcal{X}, \boldsymbol{y}_i \in \mathcal{Y}$
iid samples from joint distribution $P_{\boldsymbol{x},\boldsymbol{y}}$ (main assumption in SL)
data generating distribution, $P_{\boldsymbol{x},\boldsymbol{y}}$, unknown (recall Lec 1)

Using the ℓ_2 norm, loss for training sample i:

$$\| \underbrace{\boldsymbol{y}_i}_{\text{true value}} - \underbrace{\boldsymbol{f}(\boldsymbol{x}_i)}_{\text{prediction}} \|_2^2 = \|\boldsymbol{y}_i - \boldsymbol{\sigma}_J(\boldsymbol{W}_J \cdots \boldsymbol{\sigma}_2(\boldsymbol{W}_2\ \boldsymbol{\sigma}_1(\boldsymbol{W}_1 \boldsymbol{x}_i)))\|_2^2 \qquad (1)$$

Empirical Risk Minimization (ERM) problem:

$$\min_{\{\boldsymbol{W}_j\}_{j=1}^J} f(\boldsymbol{W}_1, \cdots, \boldsymbol{W}_J) := \frac{1}{N} \sum_{i=1}^N \|\boldsymbol{y}_i - \boldsymbol{\sigma}_J(\boldsymbol{W}_J \cdots \boldsymbol{\sigma}_2(\boldsymbol{W}_2\ \boldsymbol{\sigma}_1(\boldsymbol{W}_1 \boldsymbol{x}_i)))\|_2^2 \qquad (2)$$

Not jointly convex in set of all weights. why ?
due to composition and coupling among each layer.
simple two layer linear NN example

Strategies for training DNNs

Globally optimal solution to (2) ? $\rightarrow$ Not possible since (2) is NP-hard

First-order methods common (for scalability)

How to compute gradient for w.r.t layer $\boldsymbol{W}_j$?
using **BackPropagation (BP)**. See these video to get an intuition
`https://www.youtube.com/watch?v=tIeHLnjs5U8`
`https://www.youtube.com/watch?v=GlcnxUlrtek`
Check the full derivations in Haykin, 2009 (Chap 4)

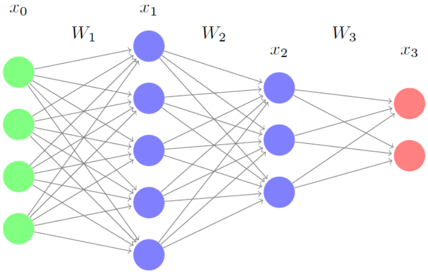
Computational complexity large ?
run SGD or mini-batch GD
applications of SGD and stochastic optim to DNN training (Lec 7)

Block-coordinate descent (BCD):
Optimize layer j, $\boldsymbol{W}_j$, while fixing others of other layers
suitable for DNN without activation func **(deep linear networks)**

Outline

1. Mathematical Models and Taxonomy
2. Statistical Learning and DNNs
3. BackPropagation
4. Practical Issues

Backprop: Illustrative Example



Look at one training sample $(\boldsymbol{z}_0, \boldsymbol{y})$. DNN training:

$f = \|\boldsymbol{y} - \boldsymbol{\sigma}_3(\boldsymbol{W}_3\boldsymbol{\sigma}_2(\boldsymbol{W}_2(\boldsymbol{\sigma}_1(\boldsymbol{W}_1\boldsymbol{z}_0))))\|_2^2$

Each layer optimized using GD :

$$\boldsymbol{W}_j^{(k+1)} = \boldsymbol{W}_j^{(k)} - \alpha_j^{(k)} \frac{\partial f}{\partial \boldsymbol{W}_j}\Big|_{\boldsymbol{W}_j = \boldsymbol{W}_j^{(k)}}, \; j = 1, 2, 3$$

Backprop: Mechanism to compute $\frac{\partial f}{\partial \boldsymbol{W}_j}$

Backprop in Action

Layer 3:
Will need eqts: $e.1)\ \boldsymbol{z}_3 = \boldsymbol{\sigma}_3(\boldsymbol{W}_3\boldsymbol{z}_2)$, $e.2)\ f = \|\boldsymbol{y} - \boldsymbol{z}_3\|_2^2$
Find $\frac{\partial f}{\partial \boldsymbol{W}_3}$ using Chain Rule (CR):

$$\frac{\partial f}{\partial \boldsymbol{W}_3} = \frac{\partial f}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{W}_3} \overset{e.2)}{=} 2(\boldsymbol{z}_3 - \boldsymbol{y})\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{W}_3} \overset{e.1)}{=} 2(\boldsymbol{z}_3 - \boldsymbol{y})\frac{\partial(\boldsymbol{\sigma}_3(\boldsymbol{W}_3\boldsymbol{z}_2))}{\partial \boldsymbol{W}_3}$$
$$\overset{CR}{=} [2(\boldsymbol{z}_3 - \boldsymbol{y}) \circ \boldsymbol{\sigma}_3']\frac{\partial(\boldsymbol{W}_3\boldsymbol{z}_2)}{\partial \boldsymbol{W}_3} = \boldsymbol{\delta}_3\boldsymbol{z}_2^T$$

where $\boldsymbol{\delta}_3 \triangleq 2(\boldsymbol{z}_3 - \boldsymbol{y}) \circ \boldsymbol{\sigma}_3'$.

$\circ$ is the Hadamard product.
Let $\boldsymbol{u} \in \mathbb{R}^d$, and define $\boldsymbol{\sigma}_j' \triangleq \frac{\partial \boldsymbol{\sigma}(\boldsymbol{u})}{\partial \boldsymbol{u}} = [\frac{\partial \sigma(u_1)}{\partial u_1}, ..., \frac{\partial \sigma(u_d)}{\partial du}]^T$, $j = 1, 2, 3$. Then, $\boldsymbol{\sigma}_j'$ depends on characteristics of activation fnc

Backprop in Action

Layer 2:
Will need eqts: $e.1)\ \boldsymbol{z}_2 = \boldsymbol{\sigma}_2(\boldsymbol{W}_2\boldsymbol{z}_1)$, $e.2)\ f = \|\boldsymbol{y} - \boldsymbol{z}_3\|_2^2$
Find $\frac{\partial f}{\partial \boldsymbol{W}_2}$ using Chain Rule (CR):

$$\begin{aligned}
\frac{\partial f}{\partial \boldsymbol{W}_2} &= \frac{\partial f}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{W}_2} \overset{e.2)}{=} 2(\boldsymbol{z}_3 - \boldsymbol{y})\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{W}_2} = 2(\boldsymbol{z}_3 - \boldsymbol{y})\frac{\partial(\boldsymbol{\sigma}_3(\boldsymbol{W}_3\boldsymbol{z}_2))}{\partial \boldsymbol{W}_2} \\
&\overset{CR}{=} [2(\boldsymbol{z}_3 - \boldsymbol{y}) \circ \boldsymbol{\sigma}_3']\frac{\partial(\boldsymbol{W}_3\boldsymbol{z}_2)}{\partial \boldsymbol{W}_2} \overset{CR}{=} \boldsymbol{\delta}_3\frac{\partial(\boldsymbol{W}_3\boldsymbol{z}_2)}{\partial \boldsymbol{z}_2}\frac{\partial(\boldsymbol{z}_2)}{\partial \boldsymbol{W}_2} \\
&\overset{e.1}{=} \boldsymbol{W}_3^T\boldsymbol{\delta}_3\frac{\partial(\boldsymbol{\sigma}_2(\boldsymbol{W}_2\boldsymbol{z}_1))}{\partial \boldsymbol{W}_2} = \boldsymbol{W}_3^T\boldsymbol{\delta}_3\frac{\partial(\boldsymbol{z}_2)}{\partial \boldsymbol{W}_2} \overset{CR}{=} [(\boldsymbol{W}_3^T\boldsymbol{\delta}_3) \circ \boldsymbol{\sigma}_2']\frac{\partial(\boldsymbol{W}_2\boldsymbol{z}_1)}{\partial \boldsymbol{W}_2} \\
&= \boldsymbol{\delta}_2\boldsymbol{z}_1^T
\end{aligned}$$

where $\boldsymbol{\delta}_2 \triangleq (\boldsymbol{W}_3^T\boldsymbol{\delta}_3) \circ \boldsymbol{\sigma}_2'$.

Backprop in Action

Layer 3:
Similarly, find $\frac{\partial f}{\partial \boldsymbol{W}_1}$ using Chain Rule (CR):

$$\frac{\partial f}{\partial \boldsymbol{W}_1} = ... = \boldsymbol{\delta}_1 \boldsymbol{z}_0^T$$

where $\boldsymbol{\delta}_1 \triangleq (\boldsymbol{W}_2^T \boldsymbol{\delta}_2) \circ \boldsymbol{\sigma}_1'$.

BP: General Case

BP for J-layer DNN with l_2 loss

1. Cost func for a J-layer network, for one training sample $(\boldsymbol{z}_0, \boldsymbol{y})$:

$$f = \|\boldsymbol{y} - \boldsymbol{\sigma}_J(\boldsymbol{W}_J \cdots \boldsymbol{\sigma}_2(\boldsymbol{W}_2\ \boldsymbol{\sigma}_1(\boldsymbol{W}_1 \boldsymbol{z}_0)))\|_2^2$$

2. Compute gradient of each layer: $\frac{\partial f}{\partial \boldsymbol{W}_j} = \boldsymbol{\delta}_j \boldsymbol{z}_{j-1}^T$ where

$$\boldsymbol{\delta}_j \triangleq \begin{cases} (\boldsymbol{W}_{j+1}^T \boldsymbol{\delta}_{j+1}) \circ \boldsymbol{\sigma}'_j, & j < J \\ 2(\boldsymbol{x}_J - \boldsymbol{y}) \circ \boldsymbol{\sigma}'_J, & j = J \end{cases}$$

3. Update Weights of each layer:

$$\boldsymbol{W}_j^{(k+1)} = \boldsymbol{W}_j^{(k)} - \alpha_j^{(k)} \frac{\partial f}{\partial \boldsymbol{W}_j}|_{\boldsymbol{W}_j^{(k)}},\ \forall j \in [J]$$

derivations are for single sample.
don't forget to sum gradients over all other samples (step 2)

Outline

Practical Issues

- BP with full gradient is computationally demanding
- Vanishing gradient (numbers below machine precision)
- How to choose stepsize, $\alpha_j^{(k)}$.
  step-size is called **learning rate** in BP
  constant: $\alpha^{(k)} = c$ (c small),
  decaying: $\alpha^{(k)} = 1/k$, $\alpha^{(k)} = 1/\sqrt{k}$
  adaptive: ADAM, ADAGRAD, RMSProp (next lecture)

Practical Issues

BackProp (BP): $\boldsymbol{W}_j^{(k+1)} = \boldsymbol{W}_j^{(k)} - \alpha_j^{(k)} \nabla_{\boldsymbol{W}_j} f(\boldsymbol{W}_j^{(k)})$
- do gradient step for **each layer** separately
- BP is an **application** of GD for DNN training

Gradient Descent: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha^{(k)} \nabla f(\boldsymbol{w}^{(k)})$
- **implicit assumption:** update **all weights** simultaneously

mismatch b/w theoretical formulation (GD) and its implementation in a DNN training problem (BP)

Practical Issues

Theoretical form (optimization method):

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha^{(k)} \boldsymbol{d}^{(k)} \qquad (u.1)$$

- parametrize all DNN weights in one vector, $\boldsymbol{w} \in \mathbb{R}^d$
- update **all weights** simultaneously, along a descent direction
- includes GD, SGD, acceleration methods, etc.

Practical form (apply optim method to DNN training):

$$\boldsymbol{W}_j^{(k+1)} = \boldsymbol{W}_j^{(k)} - \alpha_j^{(k)} \boldsymbol{D}_j^{(k)} \qquad (u.2)$$

- in practice update **each layer** independently

In general, there is a mismatch b/w
- **theoretical form** of weight updates (an optimization method), $(u.1)$
- **practical implementation** of the update for each layer, $(u.2)$

Practical Issues

Mismatch b/w theoretical for optimization method $(u.1)$
and its application to DNN training $(u.2)$

raison d'etre for **batch normalization** (Lec 7)

converge of application to DNN training $(u.2)$ open. why ?

updates $(u.1)$ and $(u.2)$ correspond to **non-equivalent optim prob**
- **convergence results for optim methods** applicable to $(u.1)$
- but their **application to DNN training** $(u.2)$ is open
- convergence of most methods for DNN training (Lec 7) is open

Stochastic vs Batch Optimization

Preview/motivation for next lecture

Batch Optimization: Use full-gradient when updating weights

- vanilla gradient descent
- $\nabla_{\boldsymbol{W}_j} f$ depends on all training samples. Too expensive!

Stochastic Optimization: Do gradient step for one sample

- do gradient wrt sample i in training set, and for layer $\boldsymbol{W}_j$
- Low computation / memory requirement
- Widespread in large-scale optimization

Conclusions

Fundamentals of DNNs

- Building blocks of DNNs: percepteron, multi-layer NNs, taxonomy

- Mathematical model: DNNs as a composition of J non-linear functions

- Insights from statistical learning theory: expressiveness, number of layers

- Training DNNs with backprop: Intuition derivation of backprop for small example

- Backprop eqts for general DNNs

Some references

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

A. Goodfellow, Y. Bendgio, A Courville, “Deep Learning”, MIT Press, Chap. 6

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

E. Gavves, “Introduction to Neural Networks and Deep Learning,” Lecture Notes

T. B. Arnold, “Introduction to Neural Networks”, February 2016

C. M. Bishop, “Pattern Recognition and Machine Learning”, Springer-Verlag, 2006

S. Haykin, “Neural Networks and Learning Machines”, 3rd, 2009, Chap 4

Check Tutorial by Ruslan Salakhutdinov (http://www.cs.cmu.edu/ rsalakhu/)

Deep Learning

Lecture 6: Fundamentals of 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