OUTLINE

- Introduction to multi-layered neural network
- Optimization (back-propagation)
- Regularization and Dropout
- The vanishing gradient issue
- Toolkit
OUTLINE

- Introduction to multi-layered neural network
- Optimization (back-propagation)
- Regularization and Dropout
- The vanishing gradient issue
- Toolkit
**FEATURE ENGINEERING**

Most current machine learning works well because of human-designed representations and input features.

- Time consuming and task/domain dependant
- Features are often both over-specified and incomplete
- Machine learning $\Leftrightarrow$ optimizing parameters to make the best prediction
Representation learning attempts to automatically learn useful features.

- Learning a hierarchical and abstract representation
- That can be shared among tasks
- Almost all data is unlabeled ⇒ unsupervised learning
THE CURSE OF DIMENSIONALITY

In high-dimensional space, training data becomes sparse

To generalize:
- Use the distance to define some sort of “near-ness”
- Spread the probability mass around training examples (smooth the empirical distribution)

http://www.edupristine.com/blog/curse-dimensionality
THE CURSE OF DIMENSIONALITY (DIMENSION=3)

In 2-dimensions, two points are near if one falls within a certain radius of another.

In 2-d, which proportion of uniformly spaced points within black square fall inside the red circle?

$$\frac{\pi r^2}{4r^2} = \frac{\pi}{4} \approx 78\%$$

This proportion drops to 52% in 3-d, and to 0.24% in 10-d.

Consequence

In high-dimensional space, the distance does not define a useful similarity.
THE CURSE OF DIMENSIONALITY (DIMENSION=3)

Smoothness distribution

- The mass is spread around the examples.
- While plausible in this 2-dimensional case, in higher dimensions, the balls will leave holes or be too large in high probability regions.

Manifold

- If we can discover a representation of the probability concentration,
- a lower dimensional (non-linear) manifold,
- we can "flatten" it by changing the representation
- for which the distance is useful for density estimation, interpolation, ...
INTRODUCTION

NEURAL NETWORKS

Output layer
prediction of the supervised target

Hidden Layers
learn more abstract features as you head up

Raw inputs

(from Bengio, 2015)
ILLUSTRATION AT DIFFERENT LAYERS?

(Lee et al. 2009)
DEEP LEARNING AND NEURAL NETWORKS: A SUCCESS STORY

Since 2009, deep learning approaches won several challenges

- ImageNet since 2012 (Krizhevskiy et al. 2012)
- Handwritting recognition since 2009 (Graves and Schmidhuber 2009) based on (Hochreiter and Schmidhuber 1997)
- Automatic Speech recognition (Hinton et al. 2012)
Since 2009, deep learning approaches won several challenges:

- **ImageNet since 2012** (Krizhevsky et al. 2012)
- **Handwriting recognition since 2009** (Graves and Schmidhuber 2009) based on (Hochreiter and Schmidhuber 1997)
- **Automatic Speech recognition** (Hinton et al. 2012)
The breakthrough of 2006

The expression *Deep Learning* was coined around 2006 with papers on unsupervised pre-training of neural nets (Hinton et al. 2006; Hinton and Salakhutdinov 2006; Bengio et al. 2007)

And before ? (just a few dates)

- 1958 Rosenblatt proposed the perceptron (Rosenblatt 1958), following the work of McCulloch and Pitts in 1943 and Hebb in 1949.
- 1980 Neocognitron (Fukushima 1980) or the multilayered NNets
- 1982 Hopfield network with memory and recurrence (Hopfield 1982), the unsupervised SOM (Kohonen 1982), Neural PCA (Oja 1982)
- 1986 Multilayer perceptrons and backpropagation (Rumelhart et al. 1986b)
- 1989 Autoencoders (Baldi and Hornik 1989), Convolutional network (LeCun et al. 1989)
- 1993 Sparse coding (Field 1993)
BUT, WHAT IS NEW?

Why today?

- The huge amount of data and the growth of computational power.
- Regularization
- and ...
INTRODUCTION

BUT, WHAT IS NEW?

Why today?
- We have connected the dots, e.g. (Probabilistic) PCA / Neural PCA / Autoencoder
- We understand learning better (regularization, architecture)
- No need to be scared of non-convex optimization (initialization)
- The huge amount of data and the growth of computational power.

What is the difference between a NNet and a Deep Network?
An intensive empirical exploration of different issues
LOGISTIC REGRESSION

Logistic regression (binary classification)

\[ y_1 = f(\mathbf{w}_1^t \mathbf{x}) \]

\[ f(a = \mathbf{w}_1^t \mathbf{x}) = \frac{1}{1 + e^{-a}} \]

A single artificial neuron

pre-activation : \[ a_1 = \mathbf{w}_1^t \mathbf{x} \]

\[ y_1 = f(\mathbf{w}_1^t \mathbf{x}), \text{ } f \text{ is the activation function of the neuron} \]
LOGISTIC REGRESSION

From binary classification to $K$ classes (Maxent)

A simple neural network

- $x$: input layer
- $y$: output layer
- each $y_k$ has its parameters $w_k$
- $f$ is the softmax function
LOGISTIC REGRESSION

\[ y = f(Wx) \]

- \( f \) is usually a non-linear function
- \( f \) is a component wise function
- \textit{e.g.} the softmax function:

\[
y_k = P(c = k|x) = \frac{e^{w_k^t x}}{\sum_{k'} e^{w_{k'}^t x}} = \frac{e^{W_{k,:}^t x}}{\sum_{k'} e^{W_{k',:}^t x}}
\]

- tanh, sigmoid, relu, ...
MAXENT CLASSIFIERS

Word representation
For each word in the context
- surface form (one-hot vector)
- prefix
- suffix
- ...

A rich representation of the input for a better generalization.
NEURAL NETWORKS WITH A HIDDEN LAYER

\[ x : \text{raw input representation} \]

\[ h = f(W^{(1)}x) \]

\[ y = f(W^{(2)}h) \]

the internal and tailored representation

Intuitions

- Learn an internal representation of the raw input
- Apply a non-linear transformation
- The input representation \( x \) is transformed/compressed in a new representation \( h \)
- Adding more layers to obtain a more and more abstract representation
INTRODUCTION

HOW DO WE LEARN THE PARAMETERS?

For a supervised single layer neural net

Just like a maxent model:

- Calculate the gradient of the objective function and use it to iteratively update the parameters.
- Conjugate gradient, L-BFGS, ...
- In practice: **Stochastic gradient descent (SGD)**

With one hidden layer

- The internal ("hidden") units make the function non-convex ... just like other models with hidden variables:
  - hidden CRFs (Quattoni et al.2007), ...
- But we can use the same ideas and techniques
- Just without guarantees ⇒ **backpropagation** (Rumelhart et al.1986b)
OUTLINE

- Introduction to multi-layered neural network
- Optimization (back-propagation)
- Regularization and Dropout
- The vanishing gradient issue
- Toolkit
OPTIMIZATION OF A SINGLE LAYER NETWORK FOR CLASSIFICATION

The set of parameters is denoted $\theta$, in this case: $$\theta = (W)$$

The log-loss (conditional log-likelihood)

Assume the dataset $\mathcal{D} = (\mathbf{x}(i), c(i))_{i=1}^{N}$, $c(i) \in \{1, 2, \ldots, C\}$

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} l(\theta, \mathbf{x}(i), c(i)) = \sum_{i=1}^{N} \left( - \sum_{c=1}^{C} \mathbb{I}\{c = c(i)\} \log(P(c|\mathbf{x}(i))) \right)$$ (1)

$$l(\theta, \mathbf{x}(i), c(i)) = - \sum_{k=1}^{C} \mathbb{I}\{k = c(i)\} \log(y_k)$$ (2)
OPTIMIZATION USING SGD

Stochastic Gradient Descent (Bottou2010)

For ( $t = 1$ ; until convergence ; $t++$ ) :

- Pick randomly a sample ($\mathbf{x}_i, c_i$)
- Compute the gradient of the loss function w.r.t the parameters ($\nabla \theta$)
- Update the parameters : $\theta = \theta - \eta_t \nabla \theta$

Questions

- convergence : what does it mean ?
- what do you mean by $\eta_t$ ?
  - convergence if $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$
  - $\eta_t \propto t^{-1}$
  - and lot of variants like Adagrad (Duchi et al.2011), Down scheduling, ... see (LeCun et al.2012)
GRADIENT COMPUTATION AT FIRST LAYER

Inference chain:

\( x(i) \rightarrow (a = Wx(i)) \rightarrow (y = f(a)) \rightarrow l(\theta, x(i), c(i)) \)

The gradient for \( w_{kj} \)

\[
\nabla_{w_{kj}} = \frac{\partial l(\theta, x(i), c(i))}{\partial w_{kj}} = \frac{\partial l(\theta, x(i), c(i))}{\partial y} \times \frac{\partial y}{\partial a} \times \frac{\partial a}{\partial w_{kj}} \\
= -(\mathbb{1}\{k = c(i)\} - y_k)x_j = \delta_k x_j
\]
OPTIMIZATION

GRADIENT COMPUTATION AT SECOND LAYER

Ex. 1: compute the gradient - 2

\[ w_{kj} \times f(W^t x_t) = y_k \Rightarrow \delta \]

Generalization

\[ \nabla W = \delta x^t \]

\[ \delta_k = - \left( \mathbb{I}\{k = c_i\} - y_k \right) \]

with \( \delta \) the gradient at the pre-activation level.
Inference: a forward step
- Matrix multiplication with the input $\mathbf{x}$
- Application of the activation function

One training step: forward and backward steps
- Pick randomly a sample $(\mathbf{x}^{(i)}, c^{(i)})$
- Compute $\delta$
- Update the parameters: $\theta = \theta - \eta t \delta x^t$
FEED-FORWARD NETWORKS WITH A MULTI-LAYERS

One layer, indexed by \( l \)

\[
\begin{align*}
\mathbf{x}^{(l)} & \quad \mathbf{W}^{(l)} \quad \mathbf{y}^{(l)} \\
| & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad | \\
\cdots & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \cdots \\
\mathbf{x}^{(l)} : & \text{input of the layer } l \\
\mathbf{y}^{(l)} : & \mathbf{f}^{(l)}(\mathbf{W}^{(l)} \mathbf{x}^{(l)}) \\
\text{stacking layers} : & \mathbf{y}^{(l)} = \mathbf{x}^{(l+1)} \\
\mathbf{x}^{(1)} : & \text{a data example}
\end{align*}
\]

\[
\begin{align*}
\mathbf{x}^{(1)} & \quad \mathbf{x}^{(2)} & \quad \mathbf{x}^{(3)} & \quad \cdots & \quad \mathbf{x}^{(L)} \\
| & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad | \\
\mathbf{W}^{(1)} & \quad \mathbf{W}^{(2)} & \quad \mathbf{W}^{(3)} & \quad \cdots & \quad \mathbf{W}^{(L)} \\
\mathbf{y}^{(1)} & \quad \mathbf{y}^{(2)} & \quad \mathbf{y}^{(3)} & \quad \cdots & \quad \mathbf{y}^{(L)} : \text{output}
\end{align*}
\]
EXAMPLE OF TWO LAYERS

Gradient for the output layer
As in the Ex. 1:

\[ y \rightarrow y^{(2)} \]
\[ W \rightarrow W^{(2)} \]
\[ x \rightarrow x^{(2)} = y^{(1)} \]

\[ \nabla W^{(2)} = \delta^{(2)} x^{(2)^t}, \text{ with} \]
\[ \delta^{(2)}_k = -\mathbb{I}\{k = c(i)\} - y_k \]
**BACK-PROPAGATION OF THE LOSS GRADIENT**

Inference chain part 1:

\[
\begin{align*}
y^{(1)} &= f^{(1)}(a^{(1)}) \rightarrow (a^{(2)} = W^{(2)}y^{(1)}) \rightarrow (y^{(2)} = f^{(2)}(a^{(2)})) \rightarrow l(\theta, x_{(i)}, c_{(i)}) \\
\n\nabla a_j^{(1)} &= \frac{\partial l(\theta, x_{(i)}, c_{(i)})}{\partial a_j^{(1)}} = \frac{\partial l(\theta, x_{(i)}, c_{(i)})}{\partial y^{(2)}} \times \frac{\partial y^{(2)}}{\partial a^{(2)}} \times \frac{\partial a^{(2)}}{\partial y_j^{(1)}} \times \frac{\partial y_j^{(1)}}{\partial a_j^{(1)}} \\
&= \sum_k (\mathbb{I}\{k = c_{(i)}\} - y_k^{(2)}) w_{k,j}^{(2)} f'(1)(a_j) = f'(1)(a_j) (W_{:,j}^{(2)} \delta^{(2)^t})
\end{align*}
\]
The back-propagation signal $\nabla_{y^{(1)}} \leftarrow$

$$\nabla_{y^{(1)}} = W^{(2)^t} \delta^{(2)}$$
then

$$\delta^{(1)} = \nabla_{a^{(1)}} = f^{(1)'}(a^{(1)}) \circ (W^{(2)^t} \delta^{(2)})$$
As for the output layer, the gradient is:

$$\nabla W^{(1)} = \delta^{(1)} x^{(1)^t}, \text{ with}$$

$$\delta^{(1)}_j = \nabla a^{(1)}_j$$

$$\delta^{(1)} = f^{(1)}(a^{(1)}) \circ (W^{(2)^t} \delta^{(2)})$$

The term \((W^{(2)^t} \delta^{(2)})\) comes from the upper layer.
For a hidden layer $l$:

- The gradient at the pre-activation level:

$$\delta^{(l)} = f'^{(l)}(a^{(l)}) \circ (W^{(l+1)^t} \delta^{(l+1)})$$

- The update is as follows:

$$W^{(l)} = W^{(l)} - \eta_t \delta^{(l)} x^{(l)^t}$$

The layer should keep:

- $W^{(l)}$: the parameters
- $f^{(l)}$: its activation function
- $x^{(l)}$: its input
- $a^{(l)}$: its pre-activation associated to the input
- $\delta^{(l)}$: for the update and the back-propagation to the layer $l - 1$
BACK-PROPAGATION: ONE TRAINING STEP

Pick a training example: \( \mathbf{x}^{(1)} = \mathbf{x}(i) \)

Forward pass

For \( l = 1 \) to \( (L - 1) \)
- Compute \( \mathbf{y}^{(l)} = f^{(l)}(\mathbf{W}^{(l)} \mathbf{x}^{(l)}) \)
- \( \mathbf{x}^{(l+1)} = \mathbf{y}^{(l)} \)

\( \mathbf{y}^{(L)} = f^{(L)}(\mathbf{W}^{(L)} \mathbf{x}(L)) \)

Backward pass

Init: \( \delta^{(L)} = \nabla_{\mathbf{a}^{(L)}} \)

For \( l = L \) to \( 2 \) // all hidden units
- \( \delta^{(l-1)} = f'(l-1)(\mathbf{a}^{(l-1)}) \circ (\mathbf{W}^{(l)t} \delta^{(l)}) \)
- \( \mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \eta_t \delta^{(l)} \mathbf{x}^{(l)t} \)
- \( \mathbf{W}^{(1)} = \mathbf{W}^{(1)} - \eta_t \delta^{(1)} \mathbf{x}^{(1)t} \)
INITIALIZATION RECIPES

A difficult question with several empirical answers.

One standard trick

\[ W \sim \mathcal{N}(0, \frac{1}{\sqrt{n_{in}}}) \]

with \( n_{in} \) is the number of inputs

A more recent one

\[ W \sim \mathcal{U}
\left[
-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\\
\right]
\]

with \( n_{in} \) is the number of inputs
OUTLINE

- Introduction to multi-layered neural network
- Optimization (back-propagation)
- Regularization and Dropout
- The vanishing gradient issue
- Toolkit
The basic way:

\[
\mathcal{L}(\theta) = \sum_{i=1}^{N} l(\theta, x(i), c(i)) + \frac{\lambda}{2} \|\theta\|^2
\]

- The second term is the regularization term.
- Each parameter has a gaussian prior: \( \mathcal{N}(0, 1/\lambda) \).
- \( \lambda \) is a hyperparameter.
- The update has the form:

\[
\theta = (1 + \eta_t \lambda) \theta - \eta_t \nabla \theta
\]
Dropout serves to separate effects from strongly correlated features and prevents co-adaptation between units.

It can be seen as averaging different models that share parameters.

It acts as a powerful regularization scheme.

For each training example:
randomly turn-off the neurons of hidden units (with $p = 0.5$)

At test time, use each neuron scaled down by $p$
OUTLINE

- Introduction to multi-layered neural network
- Optimization (back-propagation)
- Regularization and Dropout
- The vanishing gradient issue
- Toolkit
EXPERIMENTAL OBSERVATIONS (MNIST TASK) - ONE LAYER

The MNIST database

Comparison of different depth for feed-forward architecture

- Hidden layers have a sigmoid activation function.
- The output layer is a softmax.
VANISHING GRADIENTS

EXPERIMENTAL OBSERVATIONS (MNIST TASK) – TWO LAYERS

Varying the depth

- Without hidden layer: \( \approx 88\% \) accuracy
- 1 hidden layer (30): \( \approx 96.5\% \) accuracy
- 2 hidden layer (30): \( \approx 96.9\% \) accuracy
- 3 hidden layer (30): \( \approx 96.5\% \) accuracy
- 4 hidden layer (30): \( \approx 96.5\% \) accuracy

(From http://neuralnetworksanddeeplearning.com/chap5.html)
**INTUITIVE EXPLANATION**

Let consider the simplest deep neural network, with just a single neuron in each layer.

\[ w_i, b_i \text{ are resp. the weight and bias of neuron } i \text{ and } C \text{ some cost function.} \]

Compute the gradient of \( C \) w.r.t the bias \( b_1 \)

\[
\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4} \times \frac{\partial y_4}{\partial a_4} \times \frac{\partial a_4}{\partial y_3} \times \frac{\partial y_3}{\partial a_3} \times \frac{\partial a_3}{\partial y_2} \times \frac{\partial y_2}{\partial a_2} \times \frac{\partial a_2}{\partial y_1} \times \frac{\partial y_1}{\partial a_1} \times \frac{\partial a_1}{\partial b_1} \quad (3)
\]

\[
= \frac{\partial C}{\partial y_4} \times \sigma'(a_4) \times w_4 \times \sigma'(a_3) \times w_3 \times \sigma'(a_2) \times w_2 \times \sigma'(a_1) \quad (4)
\]
INTUITIVE EXPLANATION

The derivative of the activation function : $\sigma'$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

But weights are initialize around 0.

The different layers in our deep network are learning at vastly different speeds:

- when later layers in the network are learning well,
- early layers often get stuck during training, learning almost nothing at all.
SOME HEURISTICS

Change the activation function (Rectified Linear Unit or ReLU)

- Avoid the vanishing gradient
- Some units can ”die”

See (Glorot et al.2011) for more details

Do pre-training when it is possible

See (Hinton et al.2006; Bengio et al.2007) :

- when you cannot really escape from the initial (random) point, find a good starting point.

More details

See (Hochreiter et al.2001; Glorot and Bengio2010; LeCun et al.2012)
OUTLINE

• Introduction to multi-layered neural network
• Optimization (back-propagation)
• Regularization and Dropout
• The vanishing gradient issue
• Toolkit
USEFUL TOOLS

Theano: http://deeplearning.net/software/theano/
- in python, works on CPU and GPU and several wrappers
- http://lasagne.readthedocs.org/
- http://keras.io/
- https://www.cs.cmu.edu/~ymiao/pdnntk.html
- http://deeplearning.net/software/pylearn2/
- http://blocks.readthedocs.org/

Torch7: http://torch.ch/
Lua interface to C/CUDA

TensorFlow https://www.tensorflow.org
API in C++ and Python
READINGS (PAPERS)

Basics of deep learning

- Intro to deep learning: http://www.deeplearningbook.org/contents/intro.htm
- Feedforward multi-layer nets: http://www.deeplearningbook.org/contents/
- Learning deep architectures for AI
- Practical recommendations for gradient-based training of deep architectures
- Quick’n’dirty introduction to deep learning: Advances in Deep Learning
- A fast learning algorithm for deep belief nets
- Greedy Layer-Wise Training of Deep Networks
- Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion
- Contractive auto-encoders: Explicit invariance during feature extraction
- Why does unsupervised pre-training help deep learning?
- An Analysis of Single Layer Networks in Unsupervised Feature Learning
- The importance of Encoding Versus Training With Sparse Coding and Vector Quantization
- Representation Learning: A Review and New Perspectives
- Deep Learning of Representations: Looking Forward
- Measuring Invariances in Deep Networks
- Neural networks course at USherbrooke [youtube]
FEEDFORWARD NETS

- [http://www.deeplearningbook.org/contents/mlp.html](http://www.deeplearningbook.org/contents/mlp.html)
- "Improving Neural Nets with Dropout" by Nitish Srivastava
- Batch Normalization
- "Fast Drop Out"
- "Deep Sparse Rectifier Neural Networks"
- "What is the best multi-stage architecture for object recognition?"
- "Maxout Networks"

MCMC

- Iain Murray’s MLSS slides
- Radford Neal’s Review Paper (old but still very comprehensive)
- Better Mixing via Deep Representations
- Bayesian Learning via Stochastic Gradient Langevin Dynamics

RESTRICTED BOLTZMANN MACHINES

- Unsupervised learning of distributions of binary vectors using 2-layer networks
- A practical guide to training restricted Boltzmann machines
- Training restricted Boltzmann machines using approximations to the likelihood gradient
- Tempered Markov Chain Monte Carlo for training of Restricted Boltzmann Machine
- How to Center Binary Restricted Boltzmann Machines
- Enhanced Gradient for Training Restricted Boltzmann Machines
- Using fast weights to improve persistent contrastive divergence
- Training Products of Experts by Minimizing Contrastive Divergence
READINGS (PAPERS)

Boltzmann Machines
- [Deep Boltzmann Machines](Salakhutdinov & Hinton)
- Multimodal Learning with Deep Boltzmann Machines
- Multi-Prediction Deep Boltzmann Machines
- [A Two-stage Pretraining Algorithm for Deep Boltzmann Machines](

Regularized Auto-Encoders
- [The Manifold Tangent Classifier](
- DL book chapter on autoencoders:
  [http://www.deeplearningbook.org/contents/autoencoders.html](http://www.deeplearningbook.org/contents/autoencoders.html)
- DL book chapter on representation learning:
- [Representation Learning: A Review and New Perspectives](in particular section 7).

Regularization

Stochastic Nets & GSNs
- [Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation](
- Learning Stochastic Feedforward Neural Networks
- [Generalized Denoising Auto-Encoders as Generative Models](
- Deep Generative Stochastic Networks Trainable by Backprop
READINGS (PAPERS)

Others
- Slow, Decorrelated Features for Pretraining Complex Cell-like Networks
- What Regularized Auto-Encoders Learn from the Data Generating Distribution
- Generalized Denoising Auto-Encoders as Generative Models
- Why the logistic function?

Recurrent Nets
- DL book chapter on recurrent nets
- Learning long-term dependencies with gradient descent is difficult
- Advances in Optimizing Recurrent Networks
- Learning recurrent neural networks with Hessian-free optimization
- On the importance of momentum and initialization in deep learning.
- Long short-term memory (Hochreiter & Schmidhuber)
- Generating Sequences With Recurrent Neural Networks
- Long Short-Term Memory in Echo State Networks: Details of a Simulation Study
- The "echo state" approach to analysing and training recurrent neural networks
- Backpropagation-Decorrelation: online recurrent learning with O(N) complexity
- New results on recurrent network training: Unifying the algorithms and accelerating convergence
- Audio Chord Recognition with Recurrent Neural Networks
- Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription
Convolutional Nets

- Generalization and Network Design Strategies (LeCun)
- On Random Weights and Unsupervised Feature Learning

Optimization issues with DL

- Curriculum Learning
- Evolving Culture vs Local Minima
- Knowledge Matters: Importance of Prior Information for Optimization
- Efficient Backprop
- Practical recommendations for gradient-based training of deep architectures
- Batch Normalization
- Natural Gradient Works Efficiently (Amari 1998)
- Hessian Free
- Natural Gradient (TONGA)
- Revisiting Natural Gradient
**READINGS (BOOKS)**

- *Deep Learning Book* by Yoshua Bengio, Ian Goodfellow and Aaron Courville (http://www.iro.umontreal.ca/~bengioy/dlbook/)

- *Deep Learning for Natural Language Processing* by Stanford University http://cs224d.stanford.edu/


- *Reading lists for new LISA students* by University of Montreal.