Aprendizado Profundo (Deep Learning)

Optimization and Generalization

Dario Oliveira
(dario.oliveira@fgv.br)

Overview

1. Stochastic Gradient Descent
2. Stochastic Gradient Descent + Momentum
3. Nesterov Momentum
4. AdaGrad
5. RMSProp
6. Adam

Stochastic Gradient Descent

At each step sample uniformly a **minibatch** of samples $\mathbb{B} = \{x^{(1)}, \ldots, x^{(m)}\}$, for a "small" m.

Estimate the gradient as

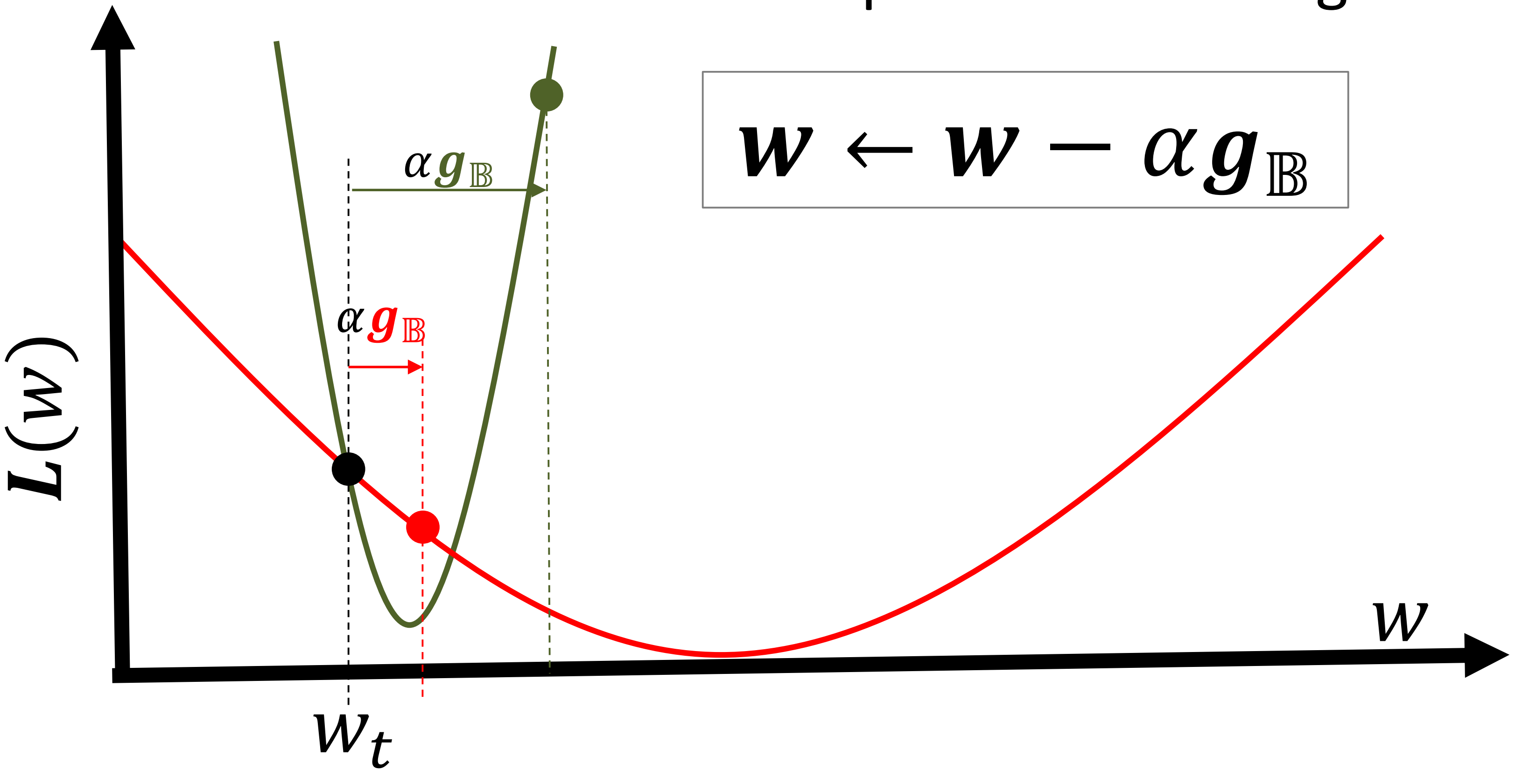
$$\boldsymbol{g}_{\mathbb{B}} \leftarrow \nabla_{\theta_d} \frac{1}{m} \sum_{i \in \mathbb{B}} L_i[f(\boldsymbol{w}, \boldsymbol{x}_i), y_i] - \lambda R(\boldsymbol{w})$$

and updated $\boldsymbol{W}$ as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{g}_{\mathbb{B}}$$

SGD: problems 1

Consider the two unidimensional loss functions below. In which case the update will be larger?

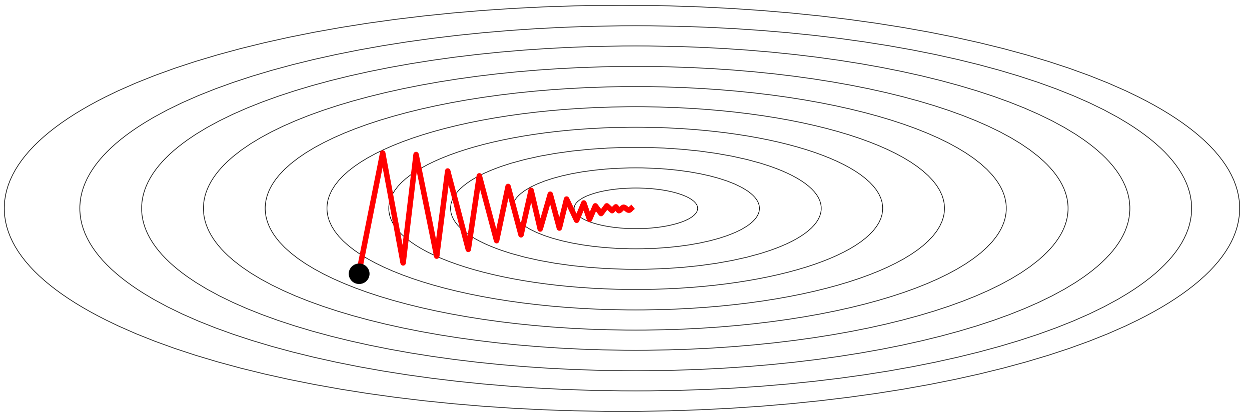


$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\boldsymbol{g}_{\mathbb{B}}$$

SGD: problems 1

What if loss changes quickly in one direction and slowly in another?

Slow progress along shallow dimension, jitter along steep direction.

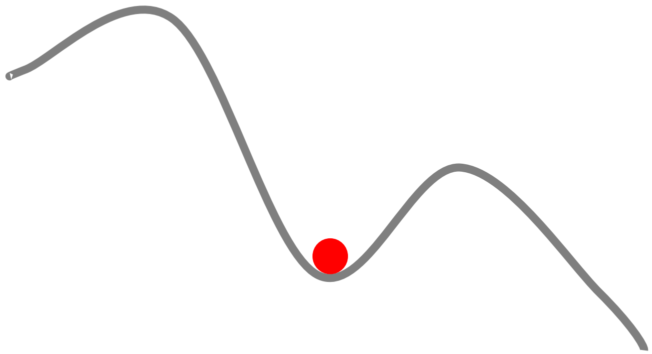


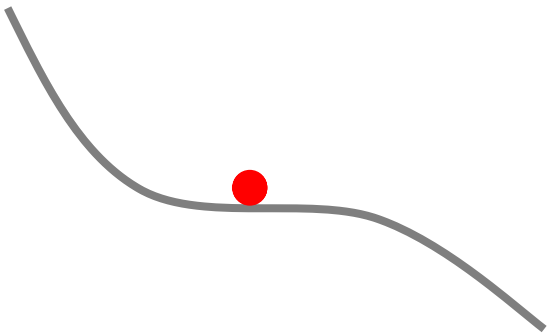
i.e., when the ratio of largest to smallest singular value of the Hessian matrix is large (high **condition number**)

SGD: problems 2

What if loss has a local minima or a saddle point?

Gradient descent gets stuck.





Saddle points are more common than local minima in high dimension.

Overview

1. Stochastic Gradient Descent
2. Stochastic Gradient Descent + Momentum
3. Nesterov Momentum
4. AdaGrad
5. RMSProp
6. Adam

SGD + Momentum

Build up “velocity” as a running mean of updates.

The larger ρ is relative to α, the more previous gradients affect the current direction

SGD

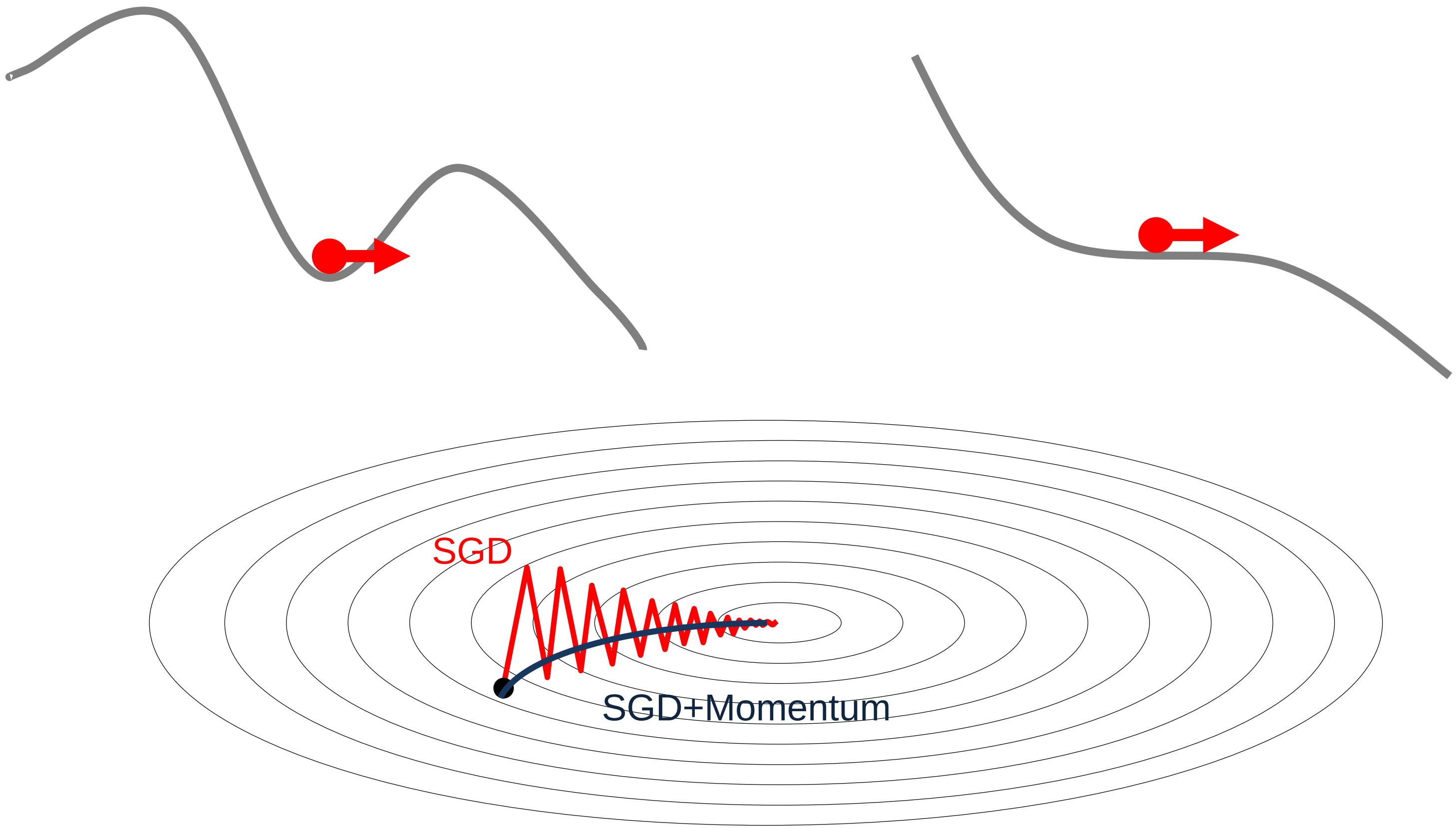
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla L(\boldsymbol{w})$$

SGD + Momentum

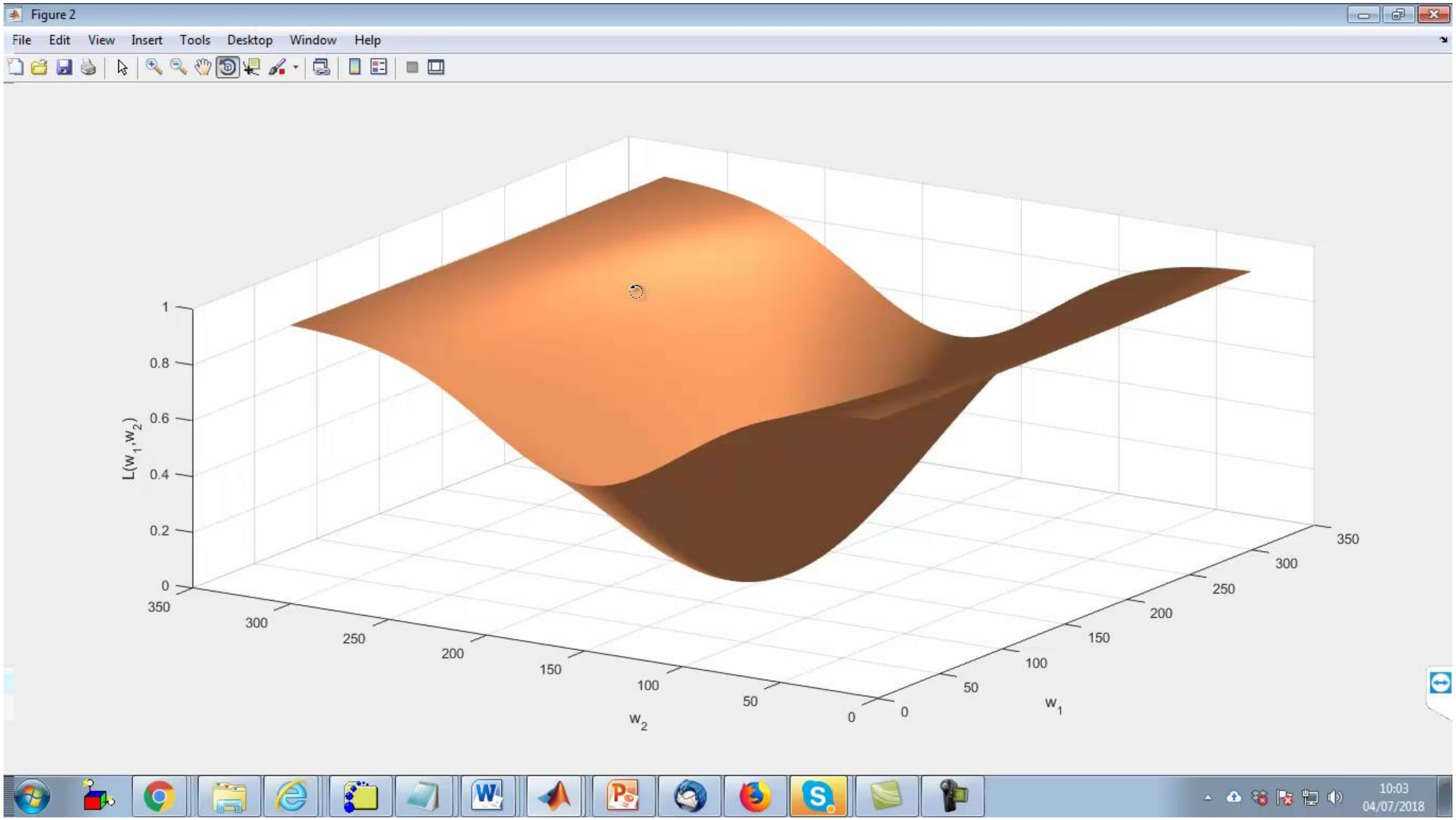
$$\boldsymbol{v} \leftarrow \rho \boldsymbol{v} - \alpha \nabla L(\boldsymbol{w})$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}$$

SGD vs. SGD+Momentum



SGD vs. SGD+Momentum

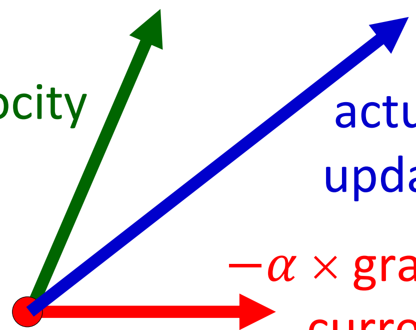


Overview

1. Stochastic Gradient Descent
2. Stochastic Gradient Descent + Momentum
3. Nesterov Momentum
4. AdaGrad
5. RMSProp
6. Adam

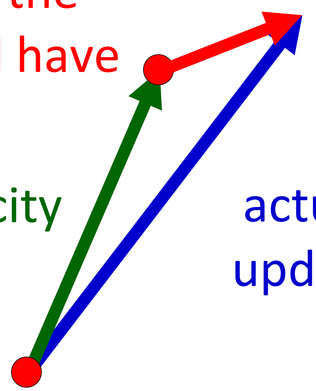
Nesterov Momentum

Momentum update



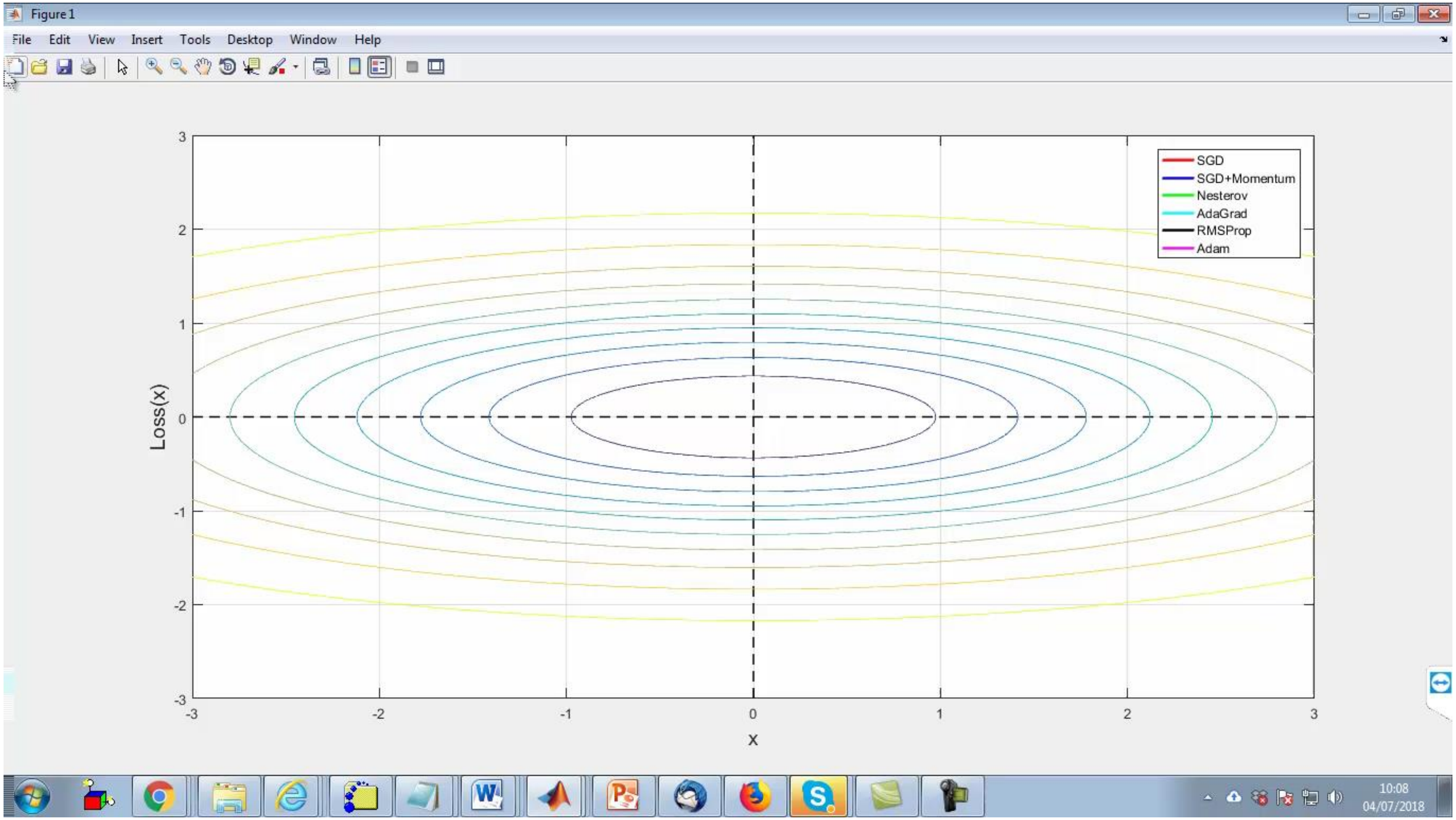
$$\boldsymbol{v} \leftarrow \rho \boldsymbol{v} - \alpha \nabla L(\boldsymbol{w})$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}$$

Nesterov Momentum



$$\boldsymbol{v} \leftarrow \rho \boldsymbol{v} - \alpha \nabla L(\boldsymbol{w} + \rho \boldsymbol{v})$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}$$

Nesterov Momentum

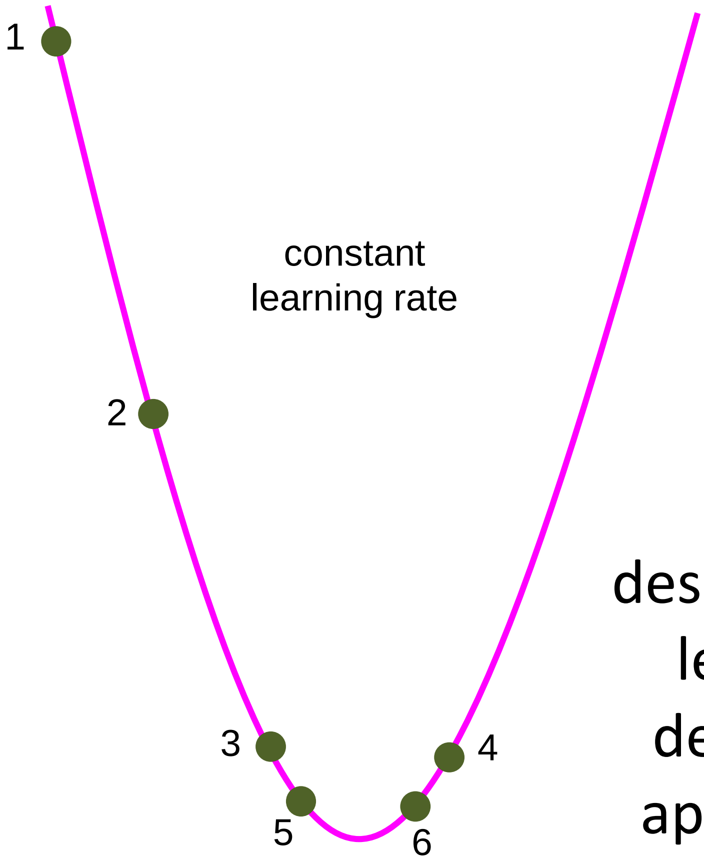


Overview

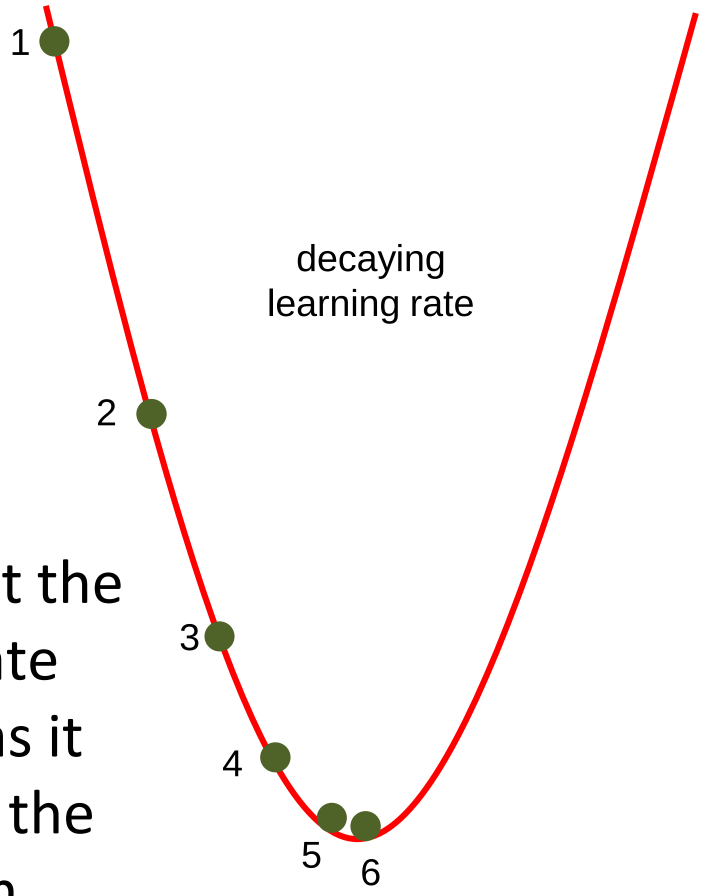
1. Stochastic Gradient Descent
2. Stochastic Gradient Descent + Momentum
3. Nesterov Momentum
4. AdaGrad
5. RMSProp
6. Adam

AdaGrad

Motivation



desirable that the learning rate decreases as it approaches the minimum



AdaGrad

AdaGrad adapts the learning rates in all dimensions by scaling them proportional to the square root of sum of squares ($\boldsymbol{r}$) of all historical squared values of gradients.

$$\boldsymbol{r} = \mathbf{0}$$

while true

element-wise (Hadamard) product

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \nabla L(\boldsymbol{w}) \odot \nabla L(\boldsymbol{w})$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{\sqrt{\boldsymbol{r}} + \boldsymbol{\varepsilon}} \odot \nabla L(\boldsymbol{w})$$

small constant to avoid division by zero

AdaGrad

$\boldsymbol{r} = \mathbf{0}$

while true

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \nabla L(\boldsymbol{w}) \odot \nabla L(\boldsymbol{w})$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{\sqrt{\boldsymbol{r}} + \boldsymbol{\varepsilon}} \odot \nabla L(\boldsymbol{w})$$

What happens with high condition number?

The movement along all dimensions tend to be equal.

What happens to the step size over long time?

The steps gets smaller and smaller

AdaGrad

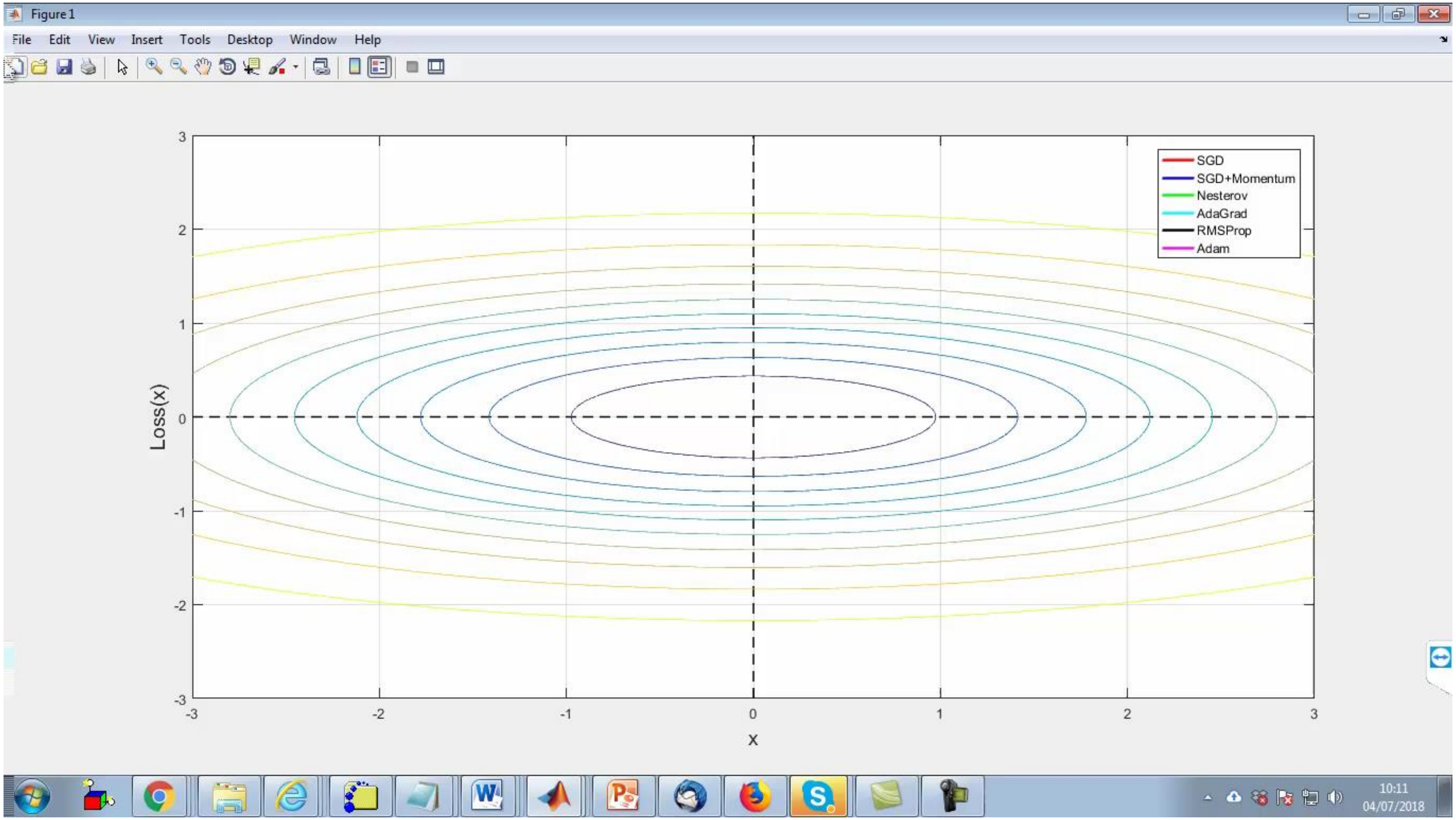


Figure 1
File Edit View Insert Tools Desktop Window Help
SGD
SGD+Momentum
Nesterov
AdaGrad
RMSProp
Adam
Loss(x)
x
3
2
1
0
-1
-2
-3
10:11
04/07/2018

Overview

1. Stochastic Gradient Descent
2. Stochastic Gradient Descent + Momentum
3. Nesterov Momentum
4. AdaGrad
5. RMSProp
6. Adam

RMSProp

RMSProp changes the gradient accumulation into a weighted moving average.

AdaGrad

$\boldsymbol{r} = \mathbf{0}$

while true

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \nabla L(\boldsymbol{w}) \odot \nabla L(\boldsymbol{w})$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{\sqrt{\boldsymbol{r}} + \boldsymbol{\varepsilon}} \odot \nabla L(\boldsymbol{w})$$

RMSProp

$\boldsymbol{r} = \mathbf{0}$

while true

$$\boldsymbol{r} \leftarrow \delta * \boldsymbol{r} + (1 - \delta) * \nabla L(\boldsymbol{w}) \odot \nabla L(\boldsymbol{w})$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{\sqrt{\boldsymbol{r}} + \boldsymbol{\varepsilon}} \odot \nabla L(\boldsymbol{w})$$

δ is the decay rate

RMSProp

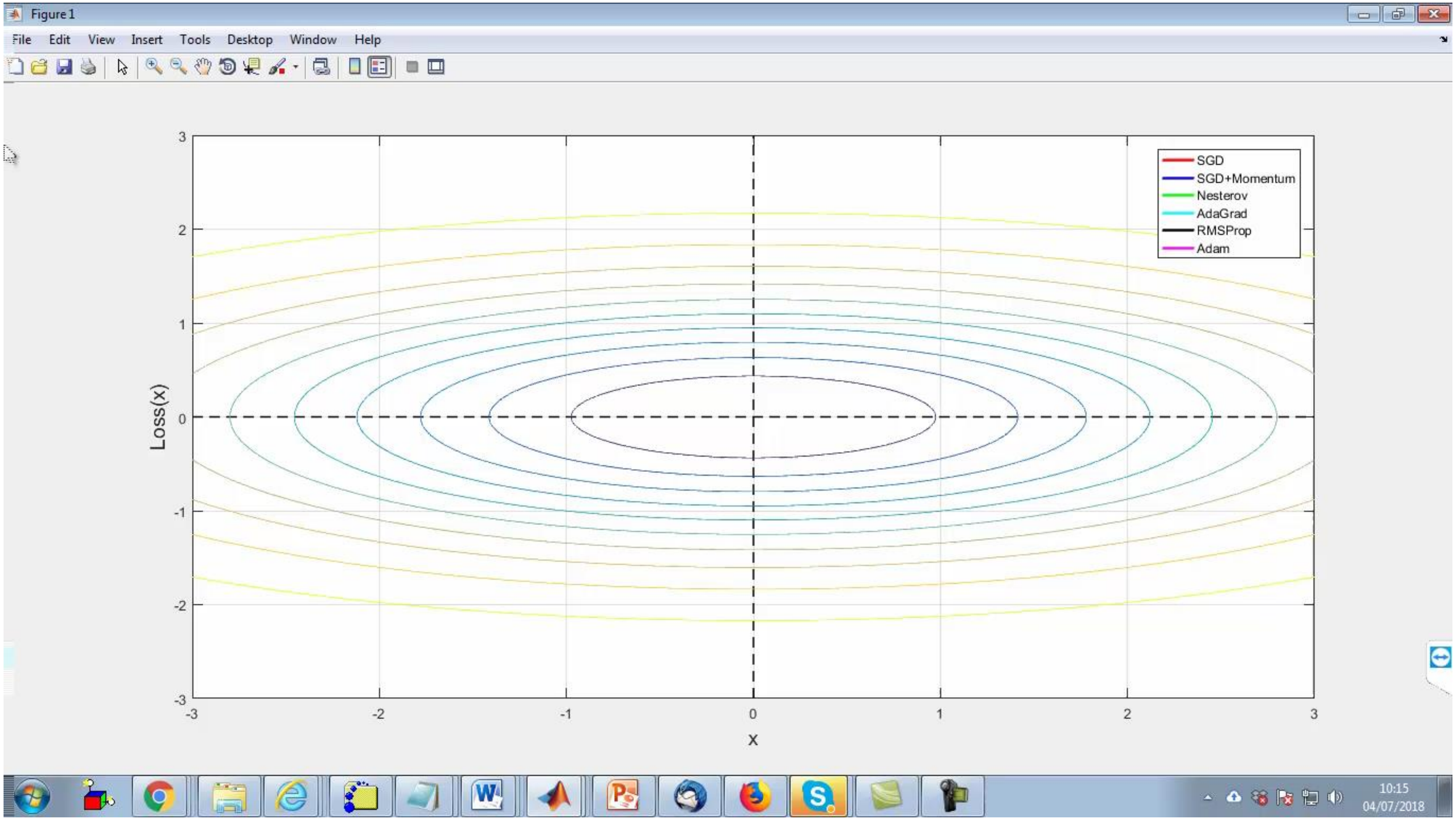


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
SGD
SGD+Momentum
Nesterov
AdaGrad
RMSProp
Adam
Loss(x)
x
3
2
1
0
-1
-2
-3
10:15
04/07/2018

Overview

1. Stochastic Gradient Descent
2. Stochastic Gradient Descent + Momentum
3. Nesterov Momentum
4. AdaGrad
5. RMSProp
6. Adam

Adaptive moments - Adam

Adam combines both, momentum and adaptive learning rate.

$\boldsymbol{r} = \mathbf{0}; \boldsymbol{v} = \mathbf{0};$

for t**=** 0 **to** *total_iterations*

1st moment update $$\boldsymbol{v} \leftarrow \rho_1 \boldsymbol{v} - (1 - \rho_1) \nabla L(\boldsymbol{w})$$

2nd moment update $$\boldsymbol{r} \leftarrow \rho_2 * \boldsymbol{r} + (1 - \rho_2) * \nabla L(\boldsymbol{w}) \odot \nabla L(\boldsymbol{w})$$

1st/2nd moment bias correction (both start at zero)

$$\left\{ \begin{array}{l} \boldsymbol{v} \leftarrow \boldsymbol{v} / (1 - \rho_1^t) \\ \boldsymbol{r} \leftarrow \boldsymbol{r} / (1 - \rho_2^t) \end{array} \right.$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{v} / \left(\sqrt{\boldsymbol{r}} + \boldsymbol{\varepsilon} \right)$$

Suggested values:

$\rho_1 = 0.9$
$\rho_2 = 0.999$
$\alpha = 10^{-3}$

Adam

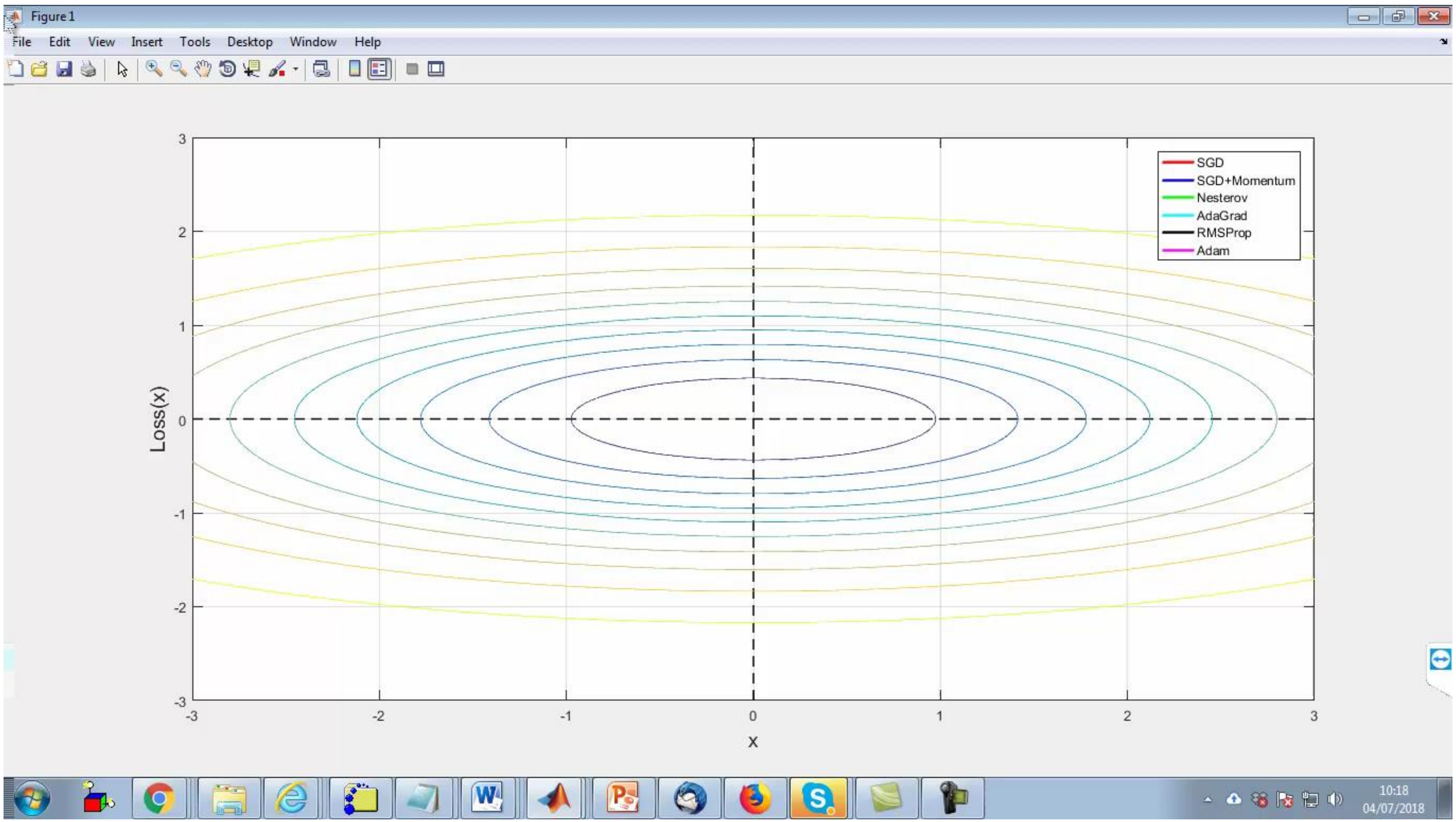
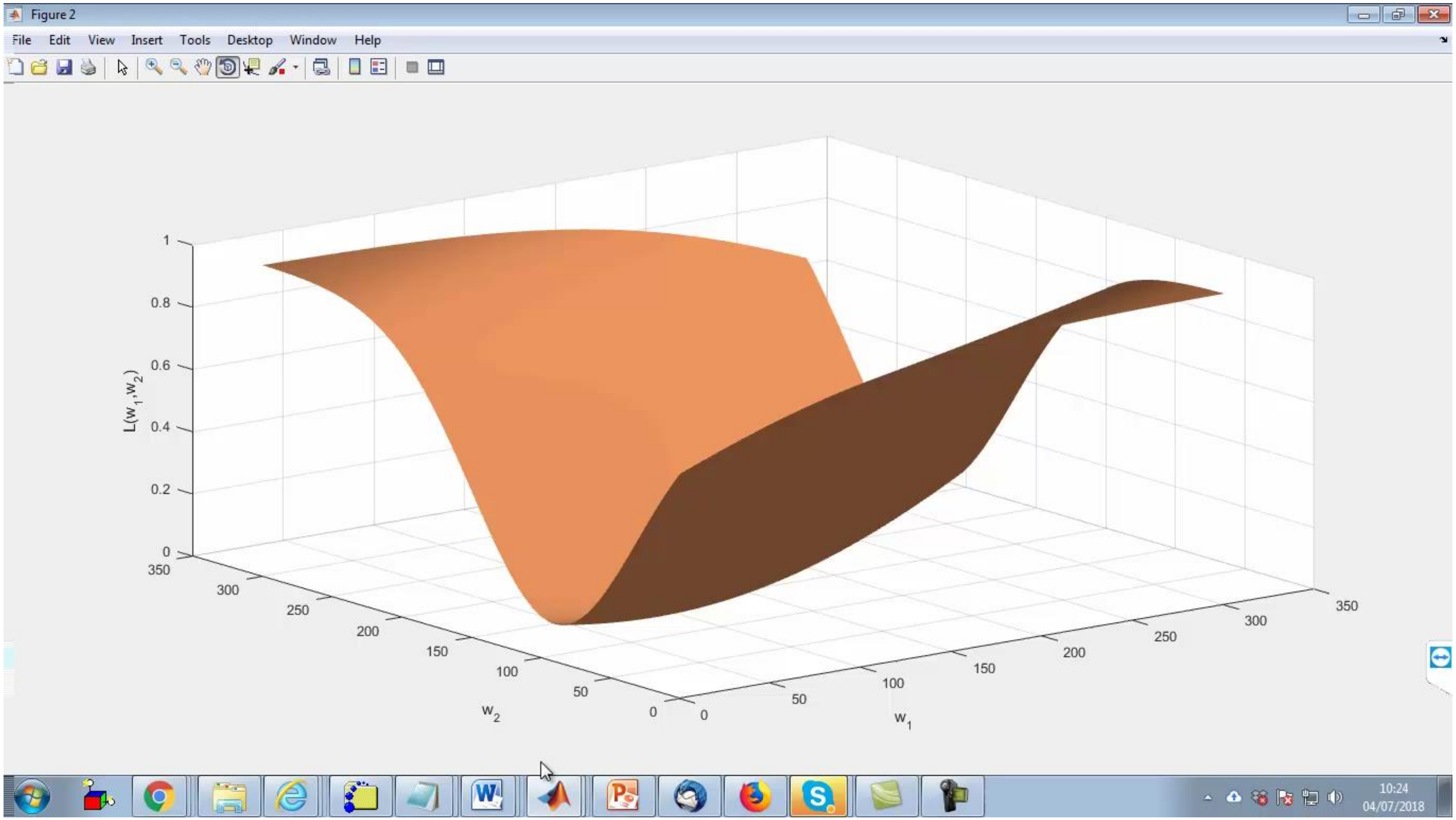


Figure 1
File Edit View Insert Tools Desktop Window Help
SGD
SGD+Momentum
Nesterov
AdaGrad
RMSProp
Adam
Loss(x)
x
3
2
1
0
-1
-2
-3
-3 -2 -1 0 1 2 3
10:18
04/07/2018

One more example



Choosing the right algorithm

The list of optimization algorithms is longer than that.

“The choice of which algorithm to use, at this point, seems to depend largely on the user’s familiarity with the algorithm.”

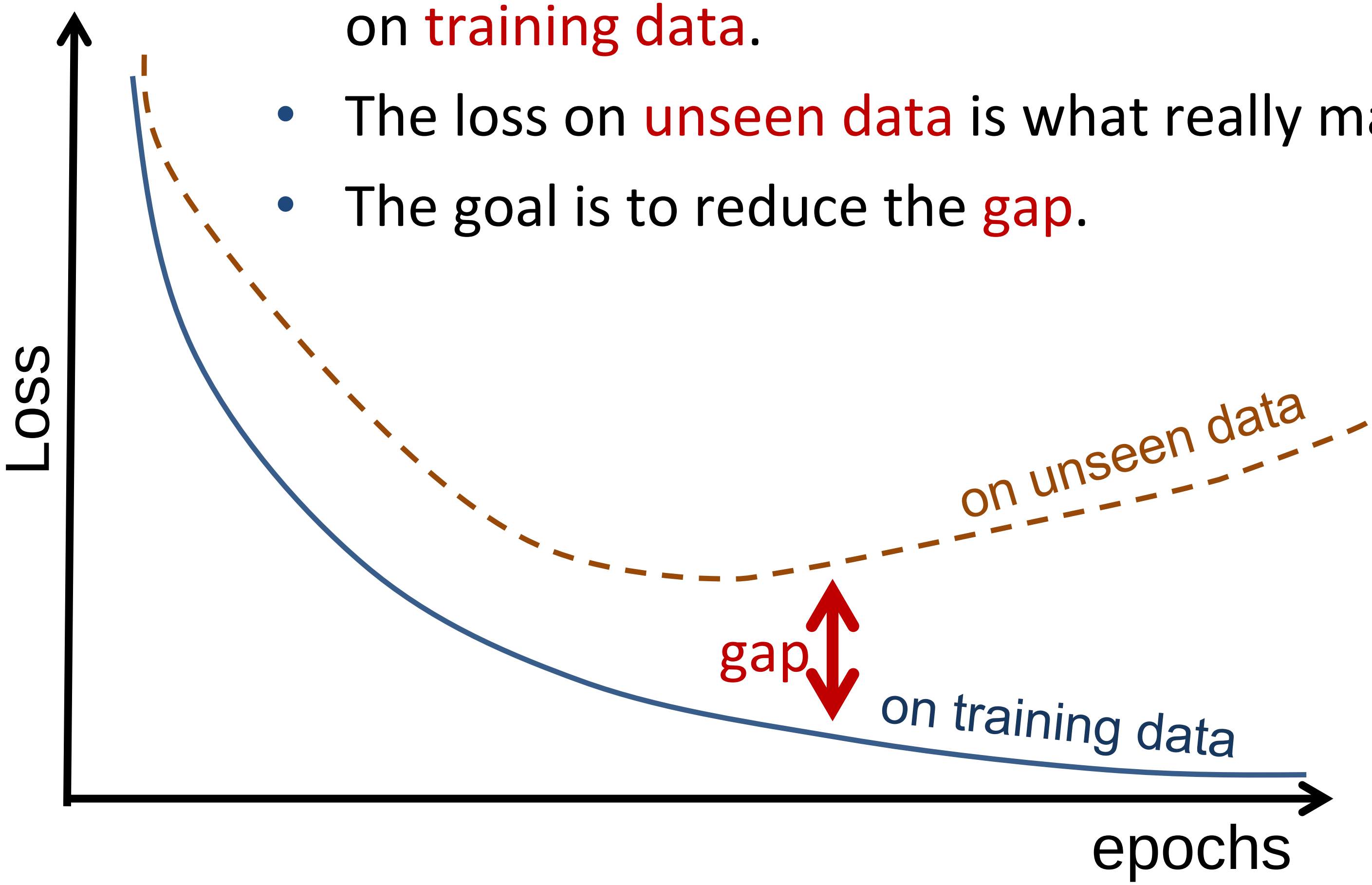
Deep Learning, Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016, MIT Press, p. 302

Overview

1. Early Stopping
2. Ensembles
3. Regularization
4. Dropout
5. Data Augmentation
6. Transfer Learning

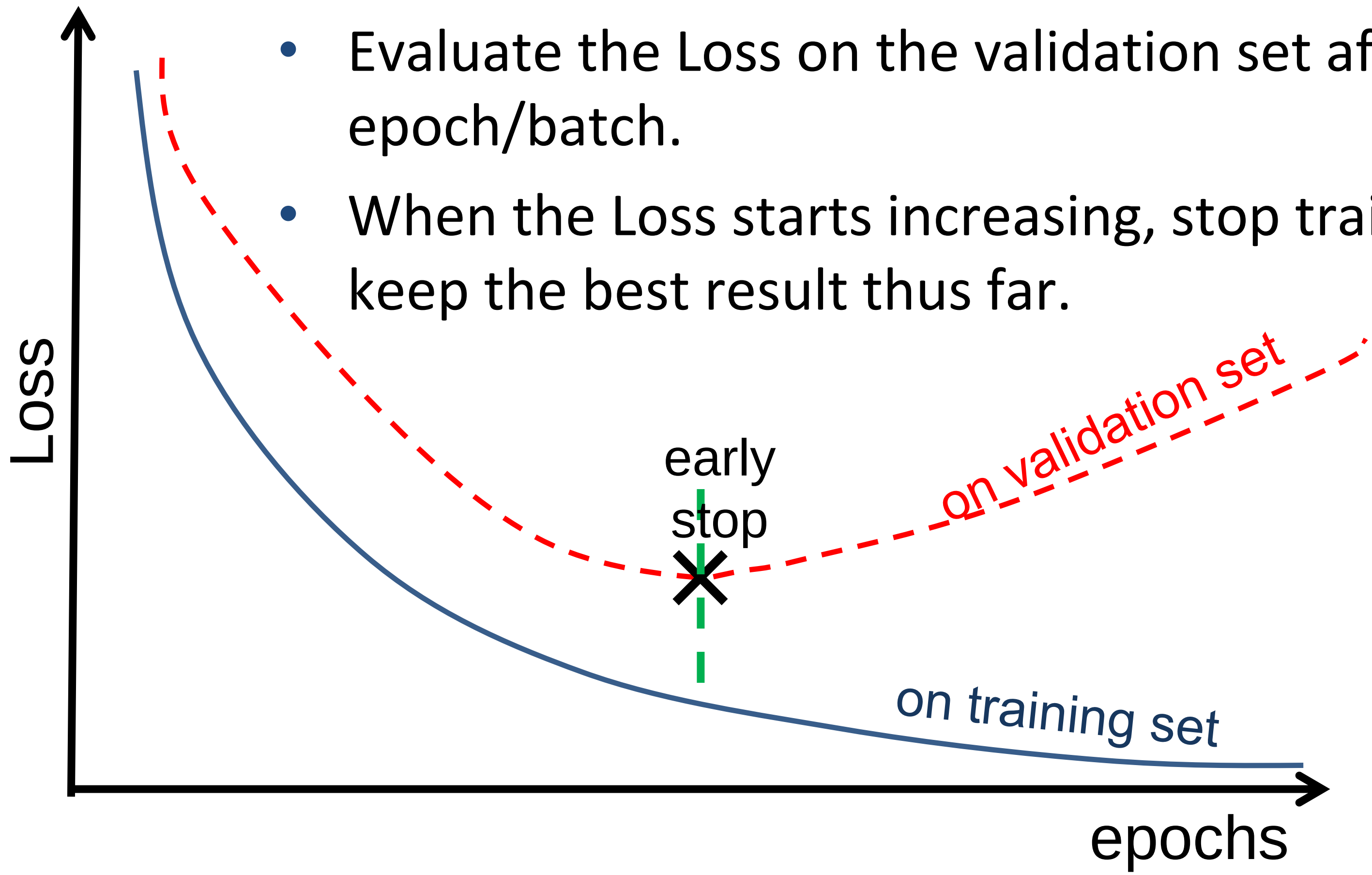
Beyond the training error

- Optimization algorithms minimize the loss function on training data.
- The loss on unseen data is what really matters.
- The goal is to reduce the gap.



Early Stopping

- Separate available labeled samples in two groups: training and validation sets.
- Evaluate the Loss on the validation set after each epoch/batch.
- When the Loss starts increasing, stop training and keep the best result thus far.



Overview

1. Early Stopping
2. Ensembles
3. Regularization
4. Dropout
5. Data Augmentation
6. Transfer Learning

Model Ensembles

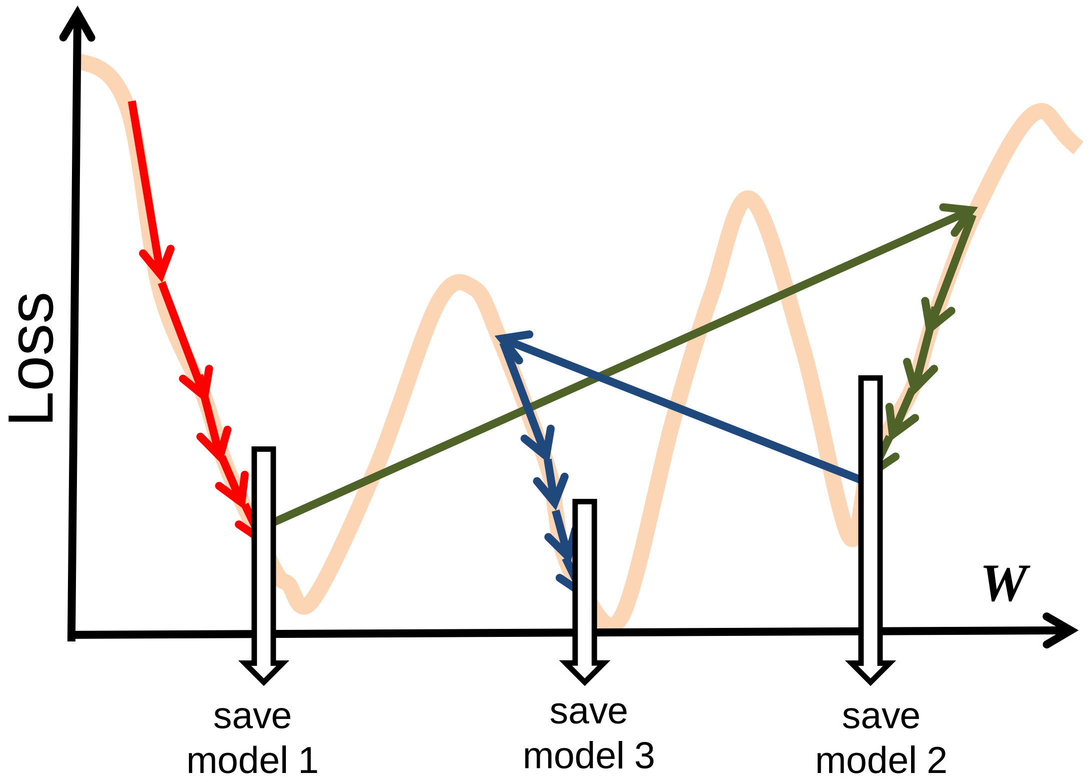
1. Train multiple independent models from different random initial restarts
2. At test time average their results / voting

   $\rightarrow$ Enjoy moderate but consistent improvements.

The problem is that training each network may take weeks.

Snapshot Ensembles

- The loss function has typically millions of local minima.
- Use SGD to find a minimum and save the parameters.
- Increase the learning rate to scape from that minimum and start again
- Keep on doing it until you get M models



Huang, et al., 2017, Snapshot Ensembles: Train 1, Get M for Free, ICLR, 2017

Overview

1. Early Stopping
2. Ensembles
3. Regularization
4. Dropout
5. Data Augmentation
6. Transfer Learning

Regularization: Add term to loss

regulation weight

$$L(\boldsymbol{W}) = \frac{1}{N}\sum_i \underbrace{L_i[f(\boldsymbol{W}, \boldsymbol{x}_i), y_i]}_{\substack{\textbf{Data loss} \\ \text{match between model and data}}} + \lambda \underbrace{R(\boldsymbol{W})}_{\substack{\textbf{Regularization} \\ \text{model complexity}}}$$

L2 (weight decay) $\rightarrow$ $R(\boldsymbol{W}) = \sum_k \sum_l \boldsymbol{W}_{k,l}^2$

L1 $\rightarrow$ $R(\boldsymbol{W}) = \sum_k \sum_l |\boldsymbol{W}_{k.l}|$

Elastic net $\rightarrow$ $R(\boldsymbol{W}) = \sum_k \sum_l \beta \boldsymbol{W}_{k,l}^2 + |\boldsymbol{W}_{k.l}|$

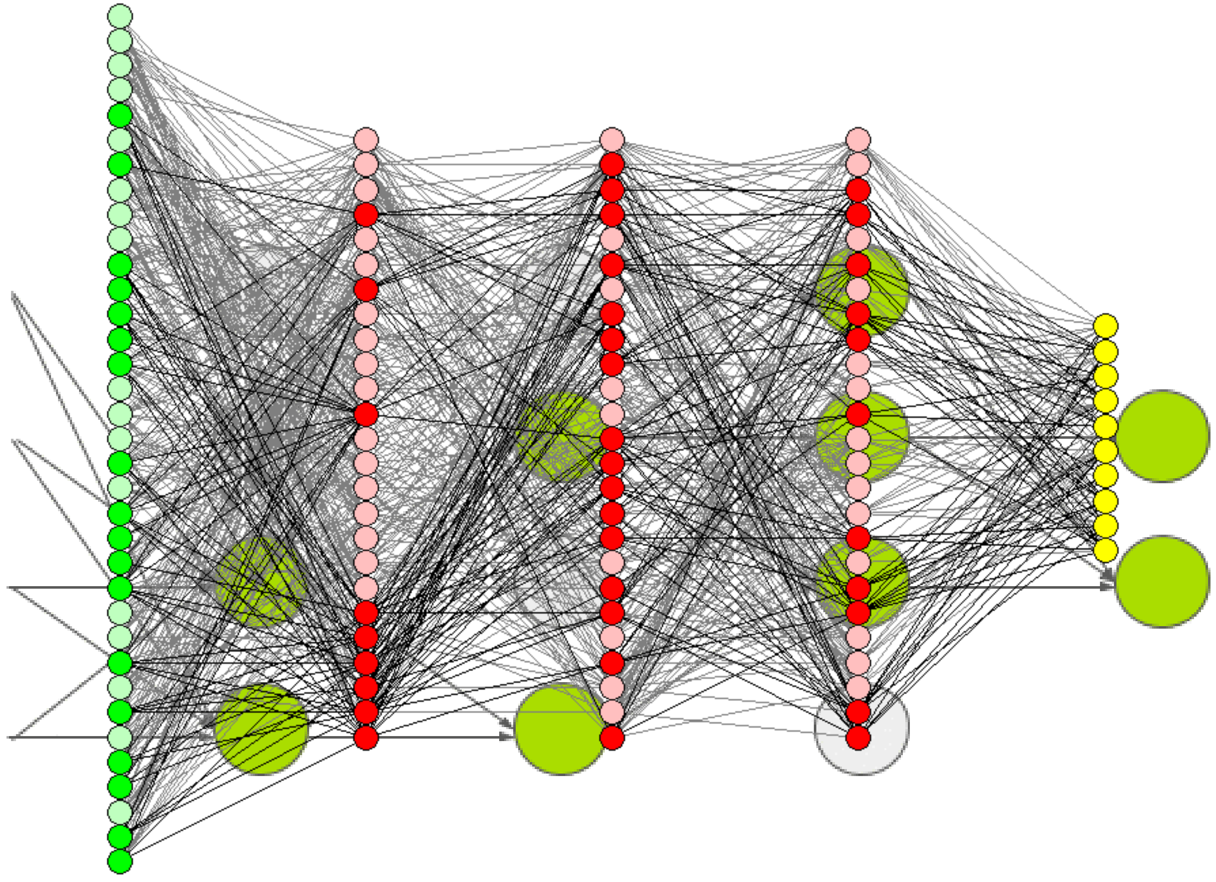
Overview

1. Early Stopping
2. Ensembles
3. Regularization
4. Dropout
5. Data Augmentation
6. Transfer Learning

Dropout

In each forward pass, randomly set some neurons to zero.

Usual probability of dropping around 0.5.

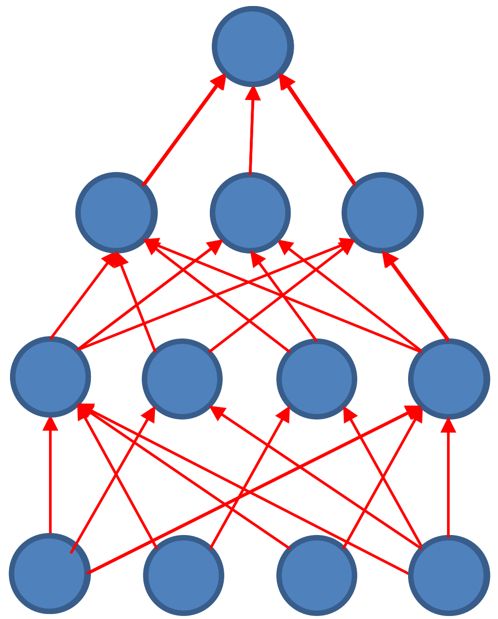


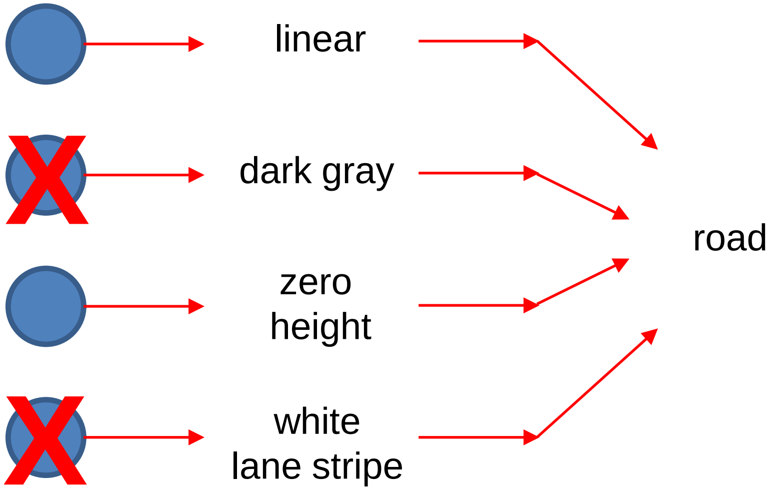
Dropout: why beneficial?

- Forces the network to learn redundant representations

- Prevents co-adaptation (when the network rely on many features).

- Easy to implement





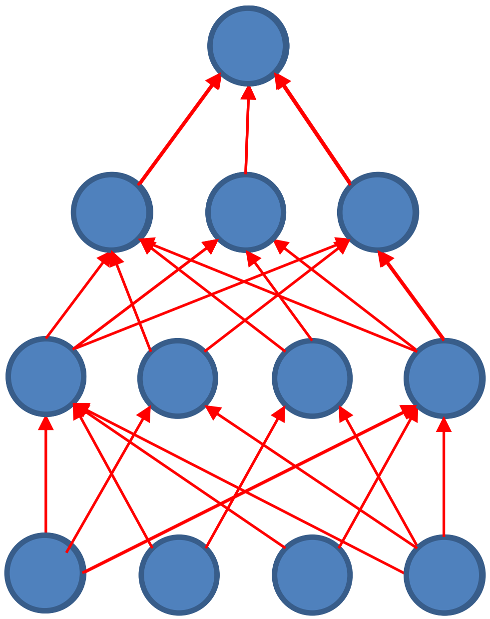
Dropout: why beneficial?

Another interpretation:

- Dropout is like training a large **ensemble** of models (that share parameters).

- Each binary mask is one model.

  e.g., an FC layer with 4096 neurons has $2^{4096} \sim 10^{1233}$ possible masks.



Dropout at test time

Dropout adds noise to the input

$$y = f_{\mathbf{w}}(x, z)$$

Random noise
Input image
Output (label)

To "average out" the randomness at test time

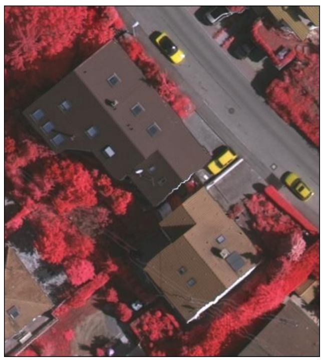
$$y = f_{\mathbf{w}}(x) = \mathbb{E}_z[f_{\mathbf{w}}(x, z)] = \int p(z) f_{\mathbf{w}}(x, z) dz$$

Overview

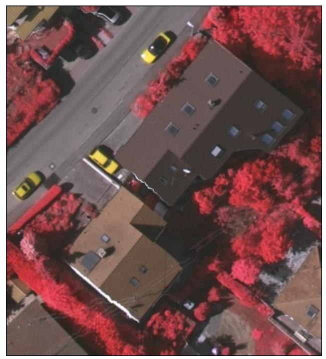
1. Early Stopping
2. Ensembles
3. Regularization
4. Dropout
5. Data Augmentation
6. Transfer Learning

Data Augmentation

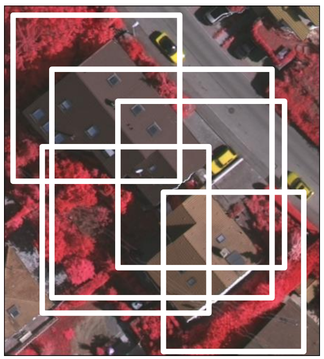
Synthesize new (noisy) samples by



original



mirroring

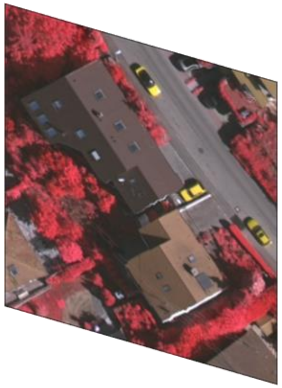


crops/scale

use your imagination with some care

rotation

color jittering



geometric transf.

Overview

1. Early Stopping
2. Ensembles
3. Regularization
4. Dropout
5. Data Augmentation
6. Transfer Learning

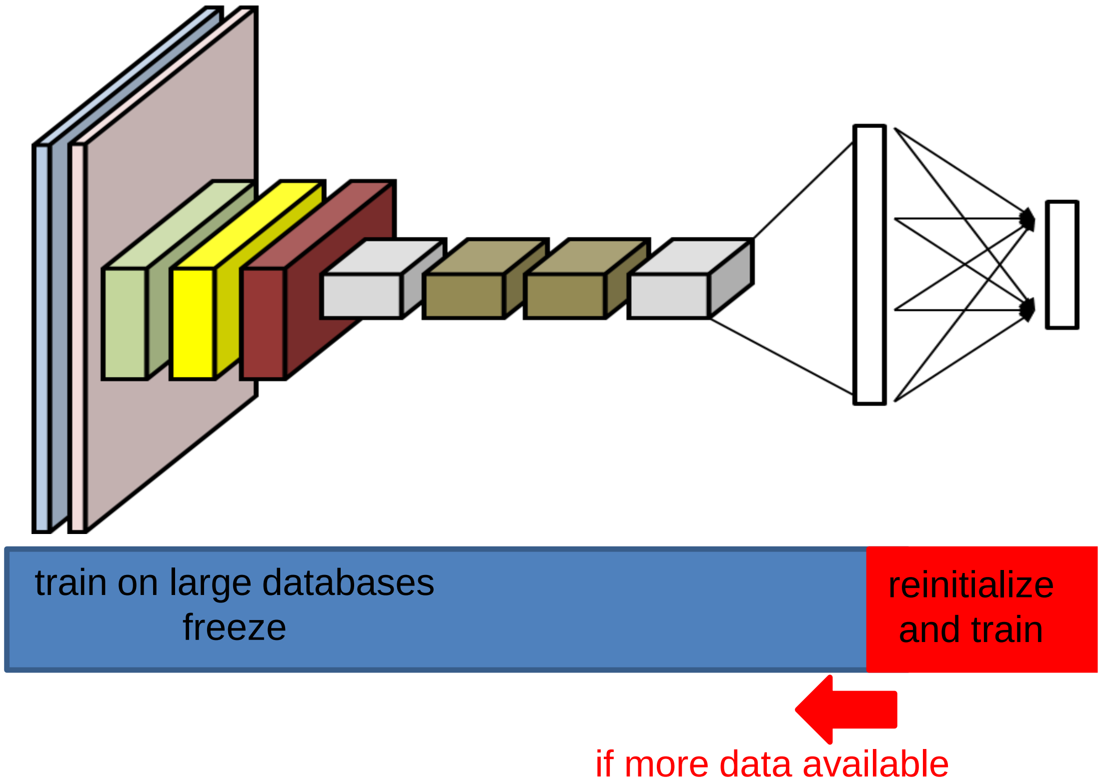
Transfer Learning

Motivation:

- CNNs might contain hundreds of millions of parameters to learn.
- This imposes a huge demand on labeled samples.
- Labeled datasets are scarce.
- Training CNNs from scratch may be impractical.
- Use ConvNets as feature extractors.

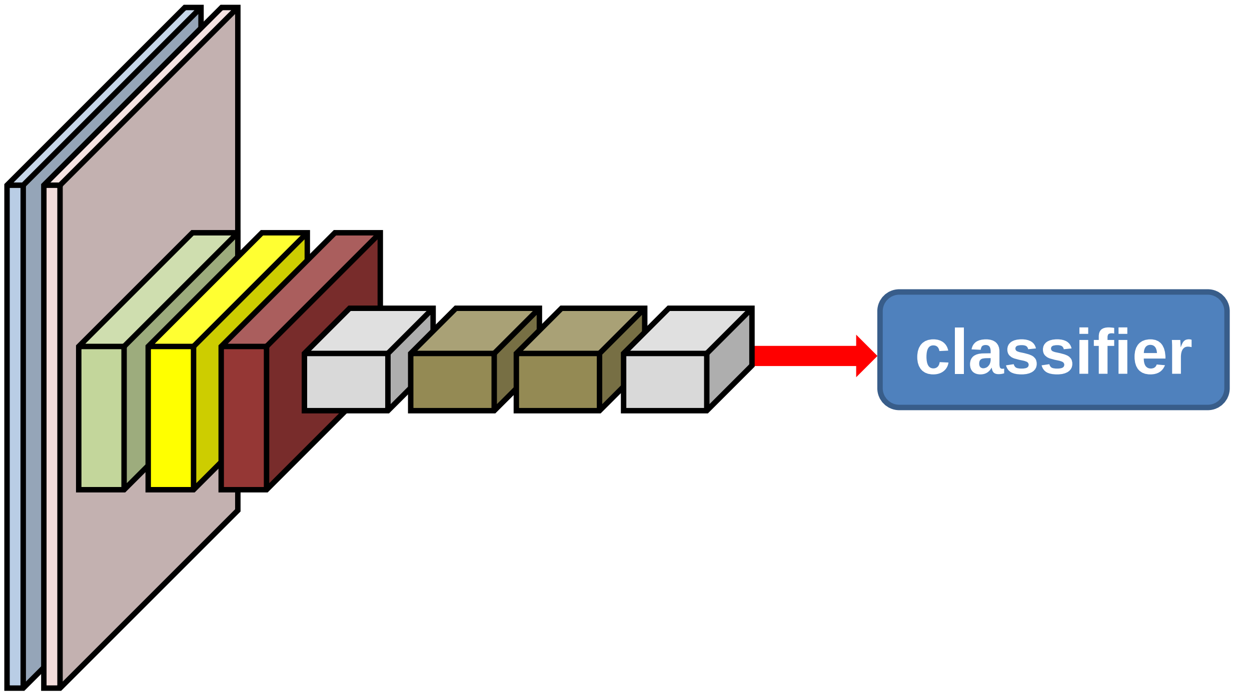
Transfer Learning

1) Take a network trained on a large dataset of a related domain
2) Freeze earlier trained layers and retrain the final layers (fine-tuning)

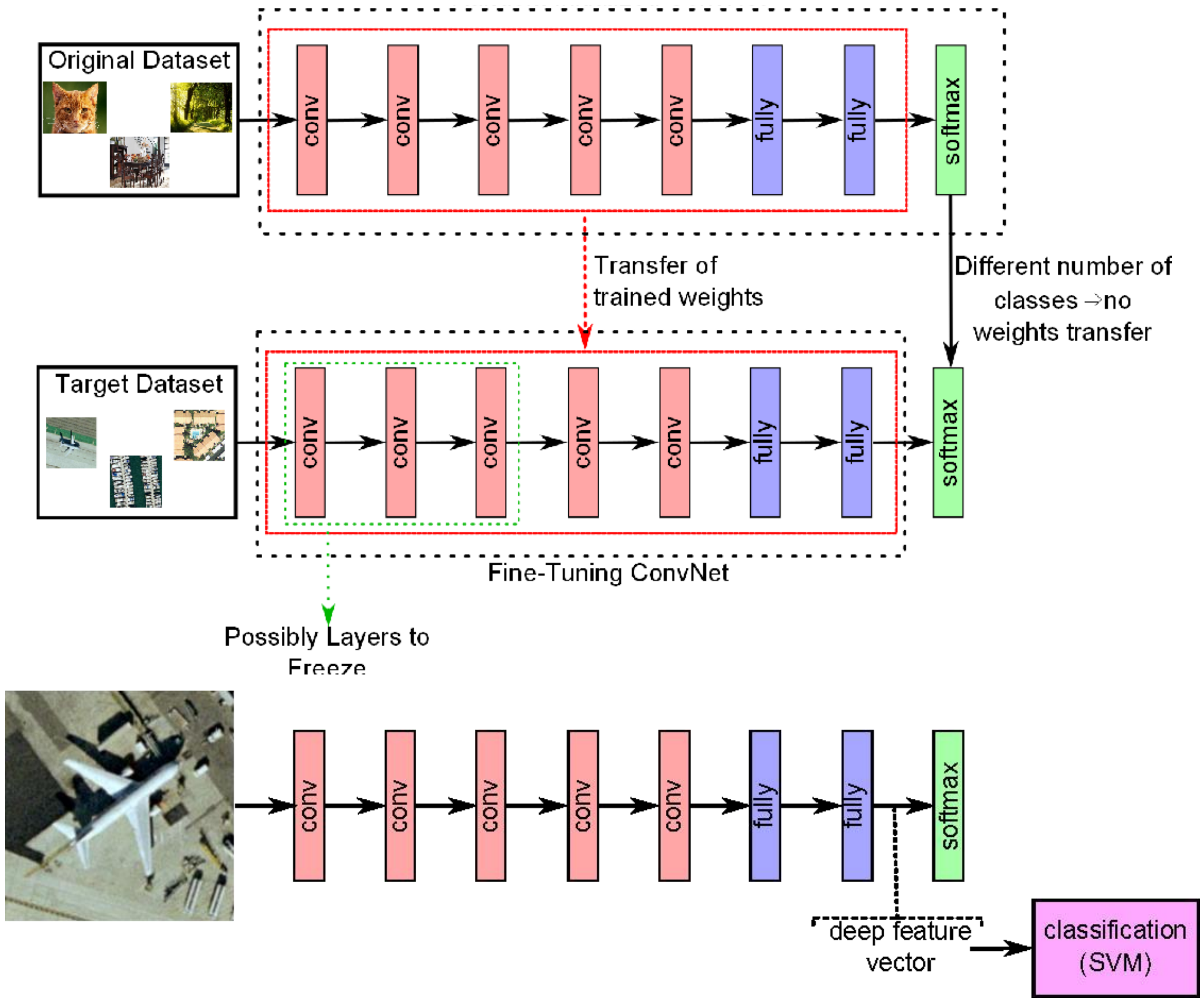


ConvNet as feature extractor

3) Alternatively, use another ML classifier



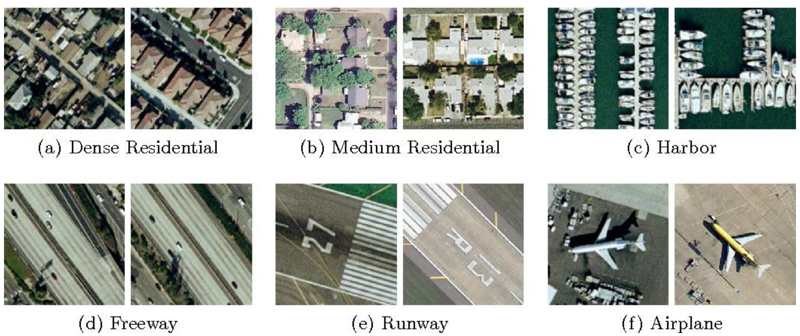
Transfer Learning is the rule



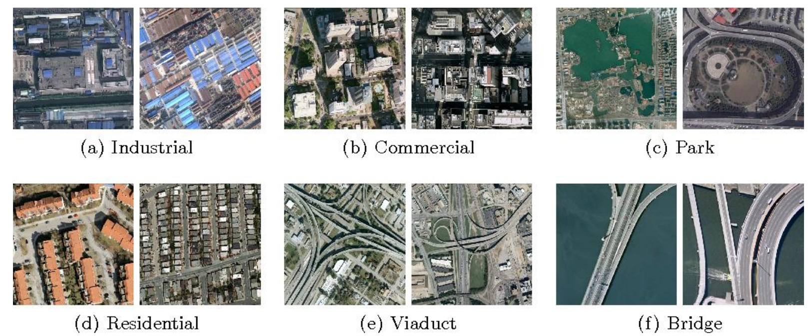
Nogueira, K., (2017), Towards better exploiting convolutional neural networks for remote sensing scene classification, Pattern Recognition

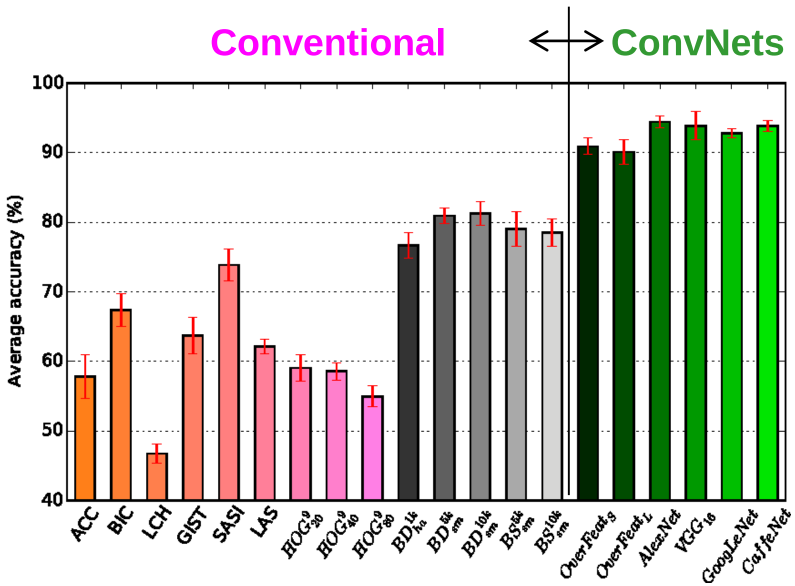
Transfer Learning is the rule

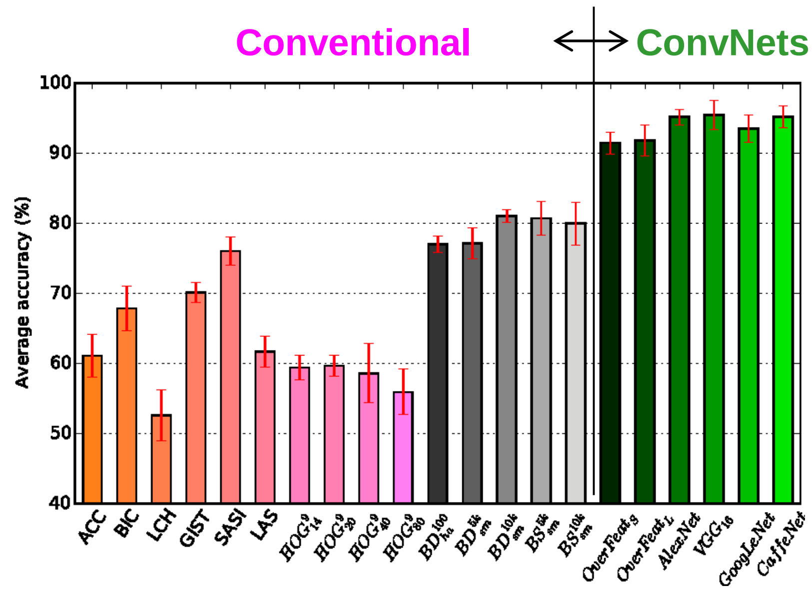
UCMercerd – 2,100 aerial scenes



RS19- 1005 from Google Earth



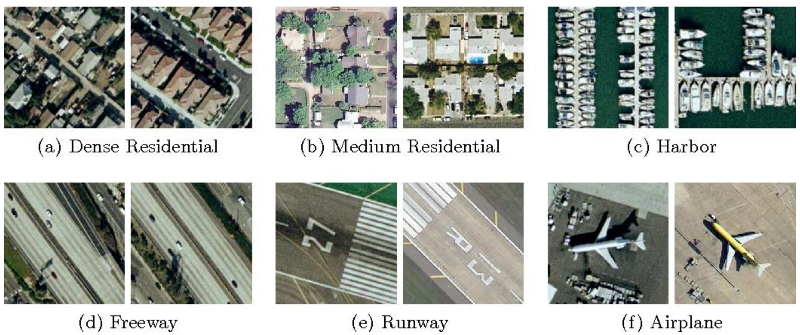




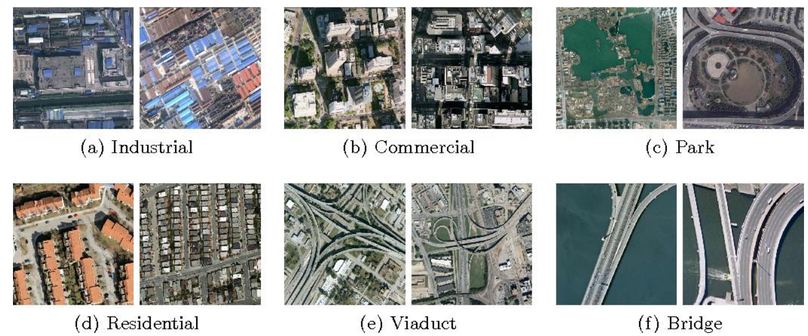
Nogueira, K., (2017), Towards better exploiting convolutional neural networks for remote sensing scene classification, Pattern Recognition

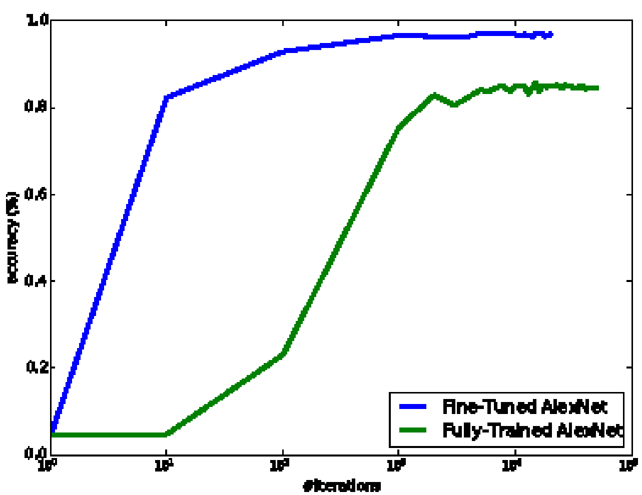
Transfer Learning is the rule

UCMercerd – 2,100 aerial scenes

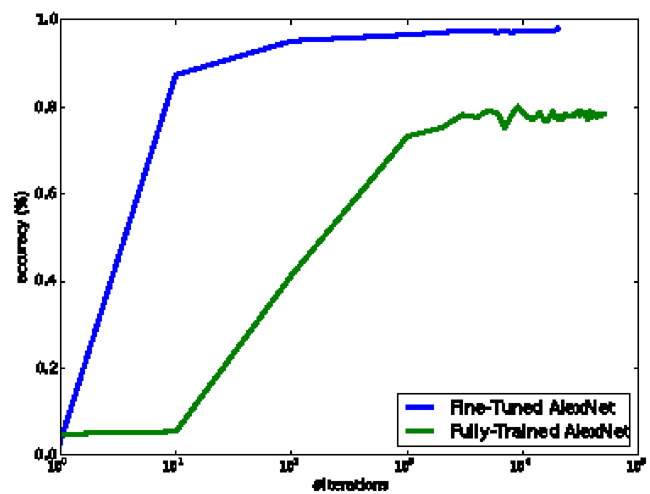


RS19- 1005 from Google Earth





(a) UCMerced Land-use dataset



(b) RS19 dataset

Nogueira, K., (2017), Towards better exploiting convolutional neural networks for remote sensing scene classification, Pattern Recognition

Model Zoo (non RS)

Deep learning frameworks provide a “Model Zoo” of pre-trained models (mostly, not from RS domain):

- Model zoo
- Model Zoos of machine and deep learning technologies
- Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
- TensorFlow: https://github.com/tensorflow/models
- PyTorch: https://github.com/pytorch/vision
- . . .

Available RS Datasets for Training

- UC Merced data set
- Aerial Image data set (AID)
- Northwestern Polytechnical University–Remote Sensing Image Scene Classification 45 data set
- Zurich Summer data set
- Zeebruges, or the Data Fusion Contest 2015 data set
- ISPRS 2-D semantic labeling challenge
- SARptical data set

Source: Zhu, X. X., Tuia, D., Mou, L., Xia, G.-S., Zhang, L., Xu, F., Fraundorfer, F., 2017. Deep learning in remote sensing: A comprehensive review and list of resources. IEEE Geoscience and Remote Sensing Magazine 5(4): 8-36.

Next Lecture

Lab 1

Convolutional Neural Networks
Architectures

See you next class! ☺