Optimization on Deep Learning

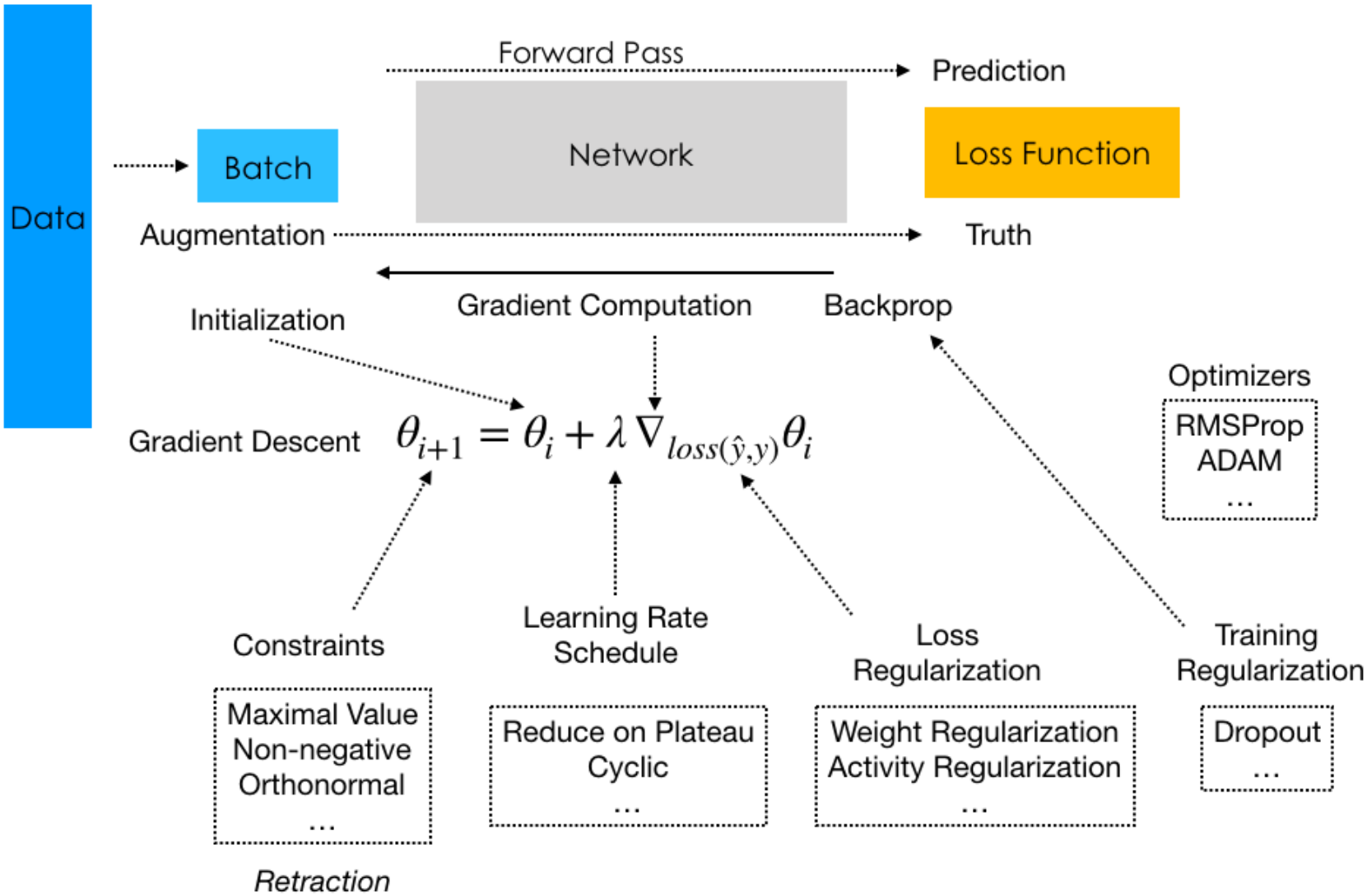
Santiago VELASCO-FORERO
http://cmm.ensmp.fr/ velasco/

MINES ParisTech
PSL Research University
Center for Mathematical Morphology
January 20, 2023

Contents

Contents

This course in one slide



Contents

Machine Learning as an optimization problem 1/2

- We have seen that most Machine Learning methods can be written as an optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) + \lambda \mathcal{R}(\boldsymbol{\theta})$$

  where $\boldsymbol{\theta}$ are the parameters of the classifier.
- The function we want to minimize is called **objective function**.
- The $*$ denotes the solution of the optimization problem (argument that solves the optimization problem).
- $\mathcal{R}(\boldsymbol{\theta})$ is called the regularization term in machine learning (shrinkage in statistics)
- λ is called the regularization coefficient.

Regularization for linear models

For linear models, we assume that our process can be approximated by $\hat{\mathbf{y}} := \mathbf{X}\boldsymbol{\theta}$, where $\mathbf{X}_{N \times P}$ is a design matrix of N samples in P variables.

Regularization for linear models

For linear models, we assume that our process can be approximated by $\hat{\mathbf{y}} := \mathbf{X}\boldsymbol{\theta}$, where $\mathbf{X}_{N \times P}$ is a design matrix of N samples in P variables.

The least-squares estimator (i.e., l_2-loss) and a l_2- penalty (regularization in l_2 norm) has a closed-form as follows,

$\arg\min_{\boldsymbol{\theta}} ||(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})||_2^2 + \lambda ||\boldsymbol{\theta}||_2^2$

Regularization for linear models

For linear models, we assume that our process can be approximated by $\hat{\mathbf{y}} := \mathbf{X}\boldsymbol{\theta}$, where $\mathbf{X}_{N\times P}$ is a design matrix of N samples in P variables.

The least-squares estimator (i.e., l_2-loss) and a l_2- penalty (regularization in l_2 norm) has a closed-form as follows,

$\arg\min_{\boldsymbol{\theta}} ||(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})||_2^2 + \lambda||\boldsymbol{\theta}||_2^2$

$$
\begin{aligned}
L(\boldsymbol{\theta}) &= (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta} \\
&= (\mathbf{y}^T - \boldsymbol{\theta}^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta} \\
&= \mathbf{y}^T\mathbf{y} - \boldsymbol{\theta}^T\mathbf{X}^T - \mathbf{y}^T\mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta} \\
&= \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\theta}^T\mathbf{X}^T\mathbf{y}^T + \boldsymbol{\theta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta}
\end{aligned}
$$

Regularization for linear models

For linear models, we assume that our process can be approximated by $\hat{\mathbf{y}} := \mathbf{X}\boldsymbol{\theta}$, where $\mathbf{X}_{N\times P}$ is a design matrix of N samples in P variables.
The least-squares estimator (i.e., l_2-loss) and a l_2- penalty (regularization in l_2 norm) has a closed-form as follows,
$\arg\min_{\boldsymbol{\theta}} ||(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})||_2^2 + \lambda||\boldsymbol{\theta}||_2^2$

$$\begin{aligned} L(\boldsymbol{\theta}) &= (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta} \\ &= (\mathbf{y}^T - \boldsymbol{\theta}^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta} \\ &= \mathbf{y}^T\mathbf{y} - \boldsymbol{\theta}^T\mathbf{X}^T - \mathbf{y}^T\mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta} \\ &= \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\theta}^T\mathbf{X}^T\mathbf{y}^T + \boldsymbol{\theta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^T\boldsymbol{\theta} \end{aligned}$$

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -2\mathbf{X}^T\mathbf{y} + 2\boldsymbol{\theta}\mathbf{X}^T\mathbf{X} + 2\lambda\boldsymbol{\theta} = 0$$

$$\begin{aligned} \Rightarrow \boldsymbol{\theta}\mathbf{X}^T\mathbf{X} + \lambda\boldsymbol{\theta} &= \mathbf{X}^T\mathbf{y} \\ \boldsymbol{\theta}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}) &= \mathbf{X}^T\mathbf{y} \end{aligned}$$

Regularization for linear models

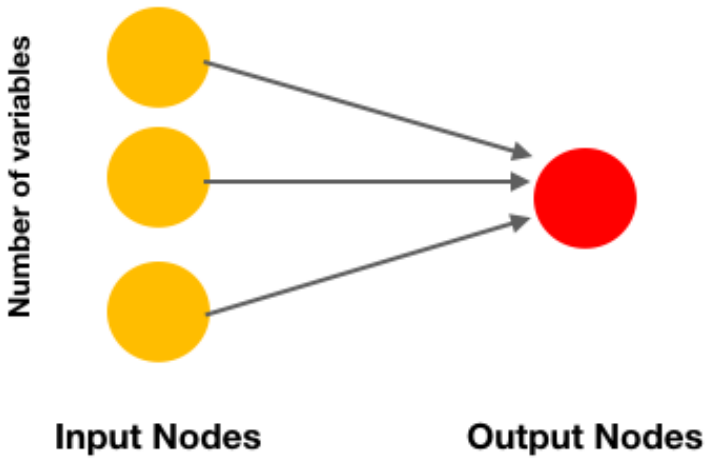
$$\arg\min_{\boldsymbol{\theta}} ||(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})||_2^2 + \lambda||\boldsymbol{\theta}||_2^2 \Rightarrow \boldsymbol{\theta}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

Regularization for linear models

$\arg\min_{\boldsymbol{\theta}} ||(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})||_2^2 + \lambda||\boldsymbol{\theta}||_2^2 \Rightarrow \boldsymbol{\theta}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

1. If $\lambda = 0$, is the solution of regular least-squares linear regression.
$$\boldsymbol{\theta}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{1}$$
2. If $\lambda = \infty$, is the solution $\boldsymbol{\theta}^* \to 0$
3. Positive λ will cause the magnitude of the weights to be smaller than regular linear solution. This is called ridge regression, and it is a special case of Tikhonov regularization.

Summary: Regularized Linear Model



1. Linear model without activation function, the prediction of the model are $\texttt{Model}(\mathbf{x}) := (\mathbf{x}\boldsymbol{\theta}^*)^T$
2. The optimal parameters $\boldsymbol{\theta}^*$ are found by minimising the loss function: $L(\boldsymbol{\theta}) = ||(\mathbf{y} - \hat{\mathbf{y}})||_2^2 + \lambda||\boldsymbol{\theta}||_2^2$
3. $N > P$, one have an exact solution.

Supervised Learning (Machine Learning)

- **Model**: $\texttt{Model}(\mathbf{x}) := \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})$ of features $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^P$
  Prediction as linear mapping of features
- **Data**: N observations $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, N$, **i.i.d.**
- **Minimization of Regularized Empirical Risk**: We would like to find $\boldsymbol{\theta}^*$ the solution of:

$$\boldsymbol{\theta}^* := \min_{\boldsymbol{\theta} \in \mathbb{R}^P} \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i)) \quad + \quad \lambda \mathcal{R}(\boldsymbol{\theta})$$

$$\texttt{Data fitting} \quad + \quad \texttt{regularizer}$$

where $L(\cdot, \cdot)$ is called the loss function.

Other loss functions, other models

1. Support Vector Machine (SVM): "Hinge" Loss

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \max\{1 - y\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}), 0\} \tag{2}$$

2. Logistic Regression:

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \log\big(1 + \exp\big(-y\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})\big)\big) \tag{3}$$

3. Others ...

Minimizing Empirial Risk = Problems!

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$

Minimizing Empirial Risk = Problems!

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$
  Loss in a training set

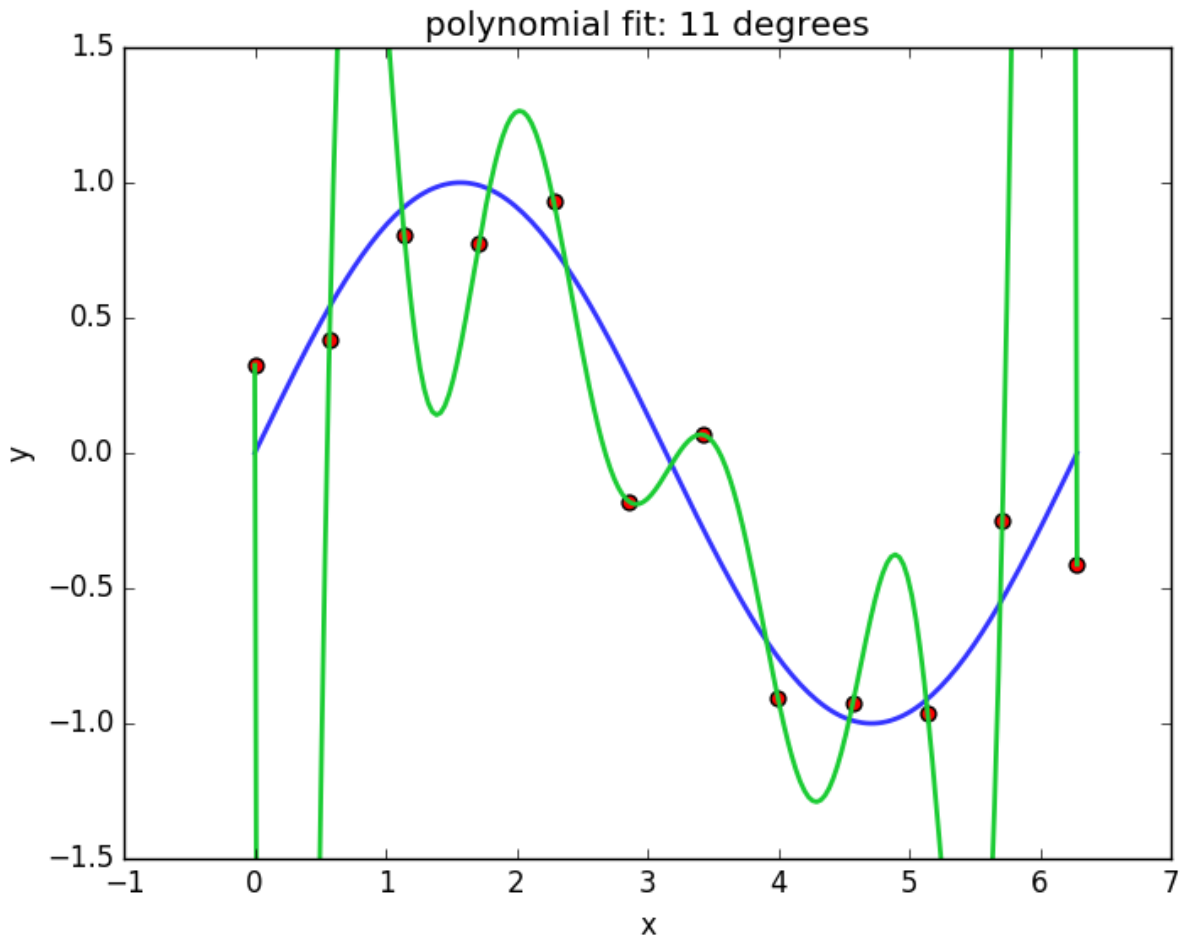


Figure: There are infinity minimizer of the empirical risk!

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$
  Loss in a training set
- Expected Risk : $f(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},y)} L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))$

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$
  Loss in a training set
- Expected Risk : $f(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},y)} L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))$
  Loss in a testing set

There are infinity minimizers of the empirical risk, but most of them have a large expected risk (overfitting).

Bias/Variance Tradeoff

Let $\hat{y} := \texttt{Model}(\mathbf{x})$ the prediction of a deterministic model evaluated at $\mathbf{x}$

$$\mathbb{E}_{(\mathbf{x},y)}\left[(y - \texttt{Model}(\mathbf{x}))^2\right] =$$

$$Var\left[y\right] + Var\left[\texttt{Model}(\mathbf{x})\right] + \left(Bias[\texttt{Model}(\mathbf{x})]\right)^2$$

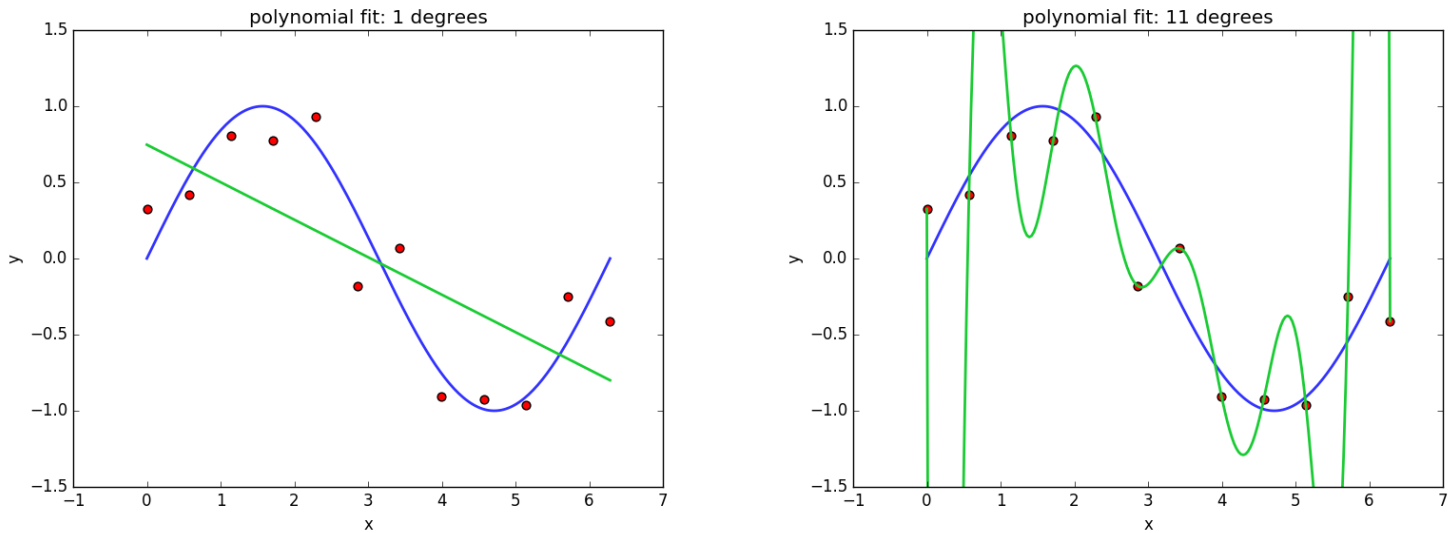


Figure: Underfitting / Overfitting

Underfitting : Prediction with less variance but more bias
Overfitting : Prediction with more variance but less bias

Deep Neural Networks

A Deep Neural Network (DNN) with parameter denoted as $\boldsymbol{\theta} = [\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(d)}]^T$, and activation functions $f^{(1)}, \ldots, f^{(d)}$ is a map that may be written as follows:

$$\hat{\mathbf{y}}_{f^{(d)},\ldots,f^{(1)};\boldsymbol{\theta}} := f^{(d)}(\mathbf{W}^{(d)} \cdots f^{(1)}(\mathbf{W}^{(1)}\mathbf{X}))$$

where the variable d denotes the total number of layers in the DNN or depth of the model.

Deep Neural Networks

A Deep Neural Network (DNN) with parameter denoted as $\boldsymbol{\theta} = [\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(d)}]^T$, and activation functions $f^{(1)}, \ldots, f^{(d)}$ is a map that may be written as follows:

$$\hat{\mathbf{y}}_{f^{(d)},\ldots,f^{(1)};\boldsymbol{\theta}} := f^{(d)}(\mathbf{W}^{(d)} \cdots f^{(1)}(\mathbf{W}^{(1)}\mathbf{X}))$$

where the variable d denotes the total number of layers in the DNN or depth of the model.

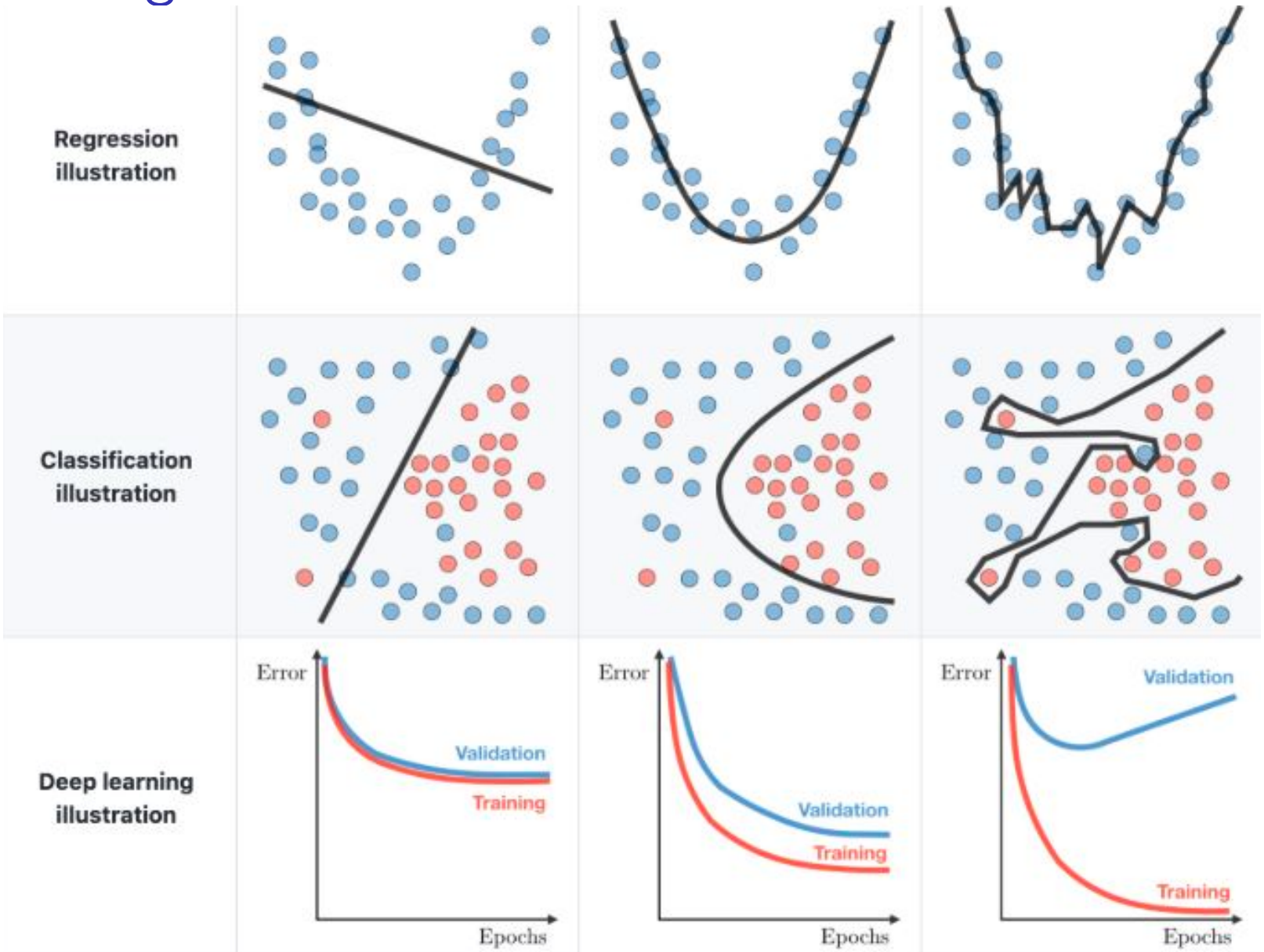
- These models are called feedforward because information flows through the function being evaluated from $\mathbf{X}$

Universal Approximation

- The universal approximation theorem means that regardless of what function we are trying to learn, we know that a large MLP will be able to represent this function. We are not guaranteed, however, that the training algorithm will be able to learn that function.
- Universal approximation theorems have also been proved for a wider class of activation functions, which includes the now commonly used rectified linear units [Leshno et al., 1993], max-plus layers, max-out layers, etc.

Contents

How to judge if a deep machine learning model is overfitting or not?

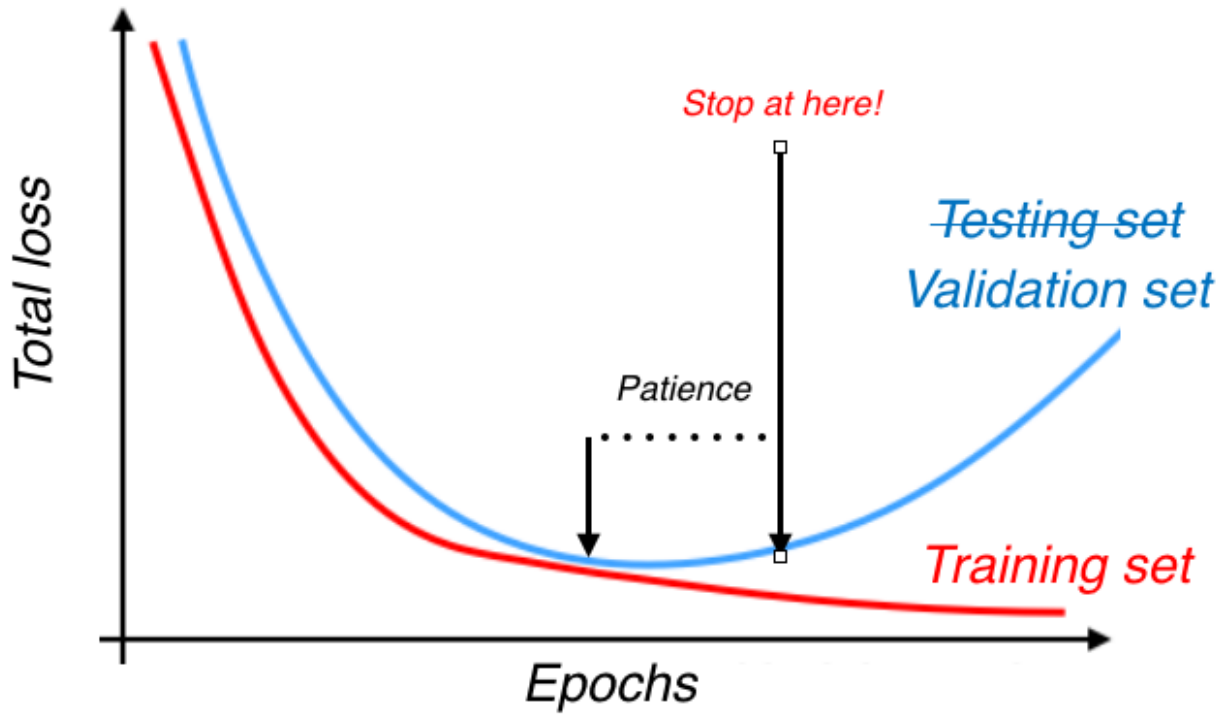


- **Underfitting**: High training error, training error can be close to validation error $\rightarrow$ high bias.
  `What to do?`

- **Underfitting**: High training error, training error can be close to validation error $\rightarrow$ high bias.
  `What to do?` Complexify model, add more features, change initializations, change learning rate / batch size.
- **Overfitting**: Very low training error, large gap between training and validation error $\rightarrow$ high variance.
  `What to do?`

- **Underfitting**: High training error, training error can be close to validation error $\rightarrow$ high bias.
  `What to do?` Complexify model, add more features, change initializations, change learning rate / batch size.
- **Overfitting**: Very low training error, large gap between training and validation error $\rightarrow$ high variance.
  `What to do?` ~~Reduce model complexity~~, add more data, model regularization.

1. Early Stopping / ReduceLROnPlateau / Learning Rate Scheduler



1. Early Stopping / ReduceLROnPlateau / Learning Rate Scheduler

```
ES=EarlyStopping(monitor='loss', patience=3)
```

```
#Reduce learning rate when a metric has stopped improving.
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,patience=5, min_lr=0.001)
```

```
#Only Using Early Stopping
model.fit(x_train, y_train,epochs=100,batch_size=64,shuffle=True,callbacks=[ES])

#Using Early Stopping and ReduceLRonPLateau
model.fit(x_train, y_train,epochs=100,batch_size=64,shuffle=True,callbacks=[ES, reduce_lr])
```

Figure: Callbacks in Keras

2. We need more data!

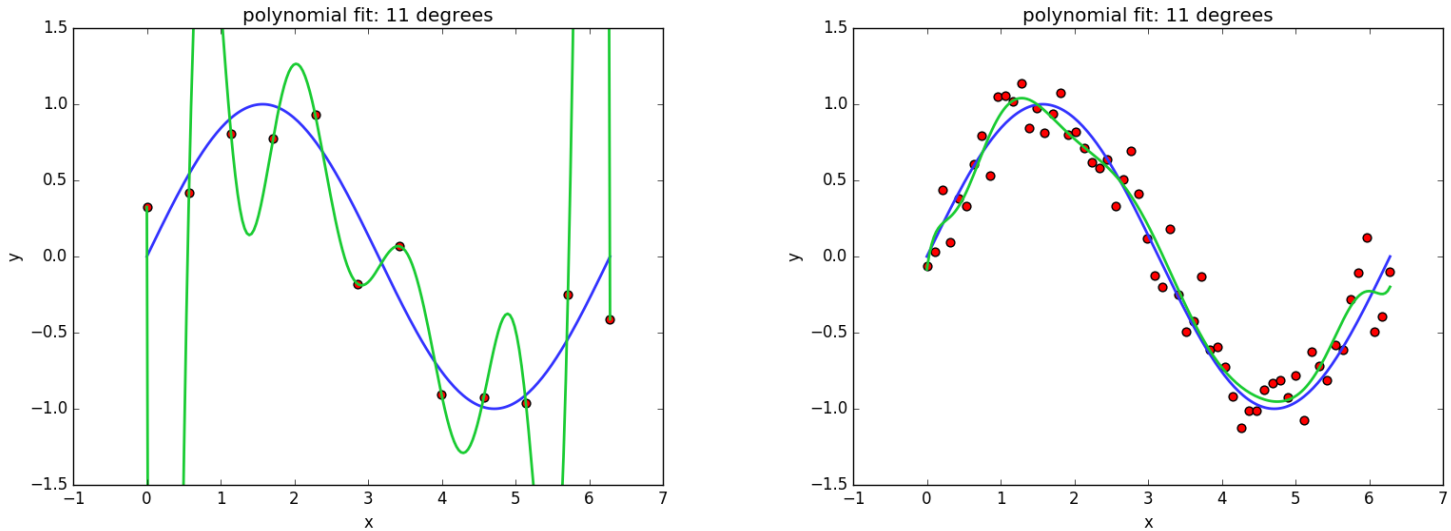


Figure: Same model complexity but more data

1. Additive Gaussian noise.
2. Data augmentation.
3. Adversarial Training (Not in this talk)

Expected Risk (again)

The expected risk

$$\mathbb{E}_{(\mathbf{x},y)} L(y, \texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x})) dP(\mathbf{x}, y) := R(\texttt{Model})$$

Expected Risk (again)

The expected risk

$$\mathbb{E}_{(\mathbf{x},y)} L(y, \texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x})) dP(\mathbf{x}, y) := R(\texttt{Model})$$

But the distribution P is unknown in most practical situations.

Expected Risk (again)

The expected risk

$$\mathbb{E}_{(\mathbf{x},y)} L(y, \texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x})) dP(\mathbf{x}, y) := R(\texttt{Model})$$

But the distribution P is unknown in most practical situations. We usually have access to a set of training data $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $(\mathbf{x}_i, y_i) \sim P$, for all $i = 1, \ldots, N$. Thus, we may approximate P by the empirical distribution:

$$P_\delta(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} = \mathbf{x}_i, y = y_i)$$

Empirical Risk (again)

Using the empirical distribution P_δ, we can now approximated the expected risk, by the called empirical risk

$$R_\delta(\texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x}))dP_\delta(\mathbf{x}, y) = \frac{1}{N}\sum_{i=1}^{N} L(\texttt{Model}(\mathbf{x}_i), y_i) \tag{4}$$

Learning the function f by minimizing (4) is known as the Empirical Risk Minimization (ERM) principle [Vapnik, 1999] (Vapnik, 1998). If the number of parameters are comparable to N, one trivial way to minimize (4) is to memorize the whole set of training data (overfitting).

Vicinal Risk Minimization (VRM)

P_δ is only one of the possibility to approximate the true distribution P. [Chapelle et al., 2001] proposed to approximate P by:

$$P_v(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i)$$

where v is vicinity distribution that measure the probability for a "virtual" pair $(\tilde{\mathbf{x}}, \tilde{y})$ to be in the vicinity of the training pair $(\mathbf{x}, y)$.

Vicinal Risk Minimization (VRM)

P_δ is only one of the possibility to approximate the true distribution P. [Chapelle et al., 2001] proposed to approximate P by:

$$P_v(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i)$$

where v is vicinity distribution that measure the probability for a "virtual" pair $(\tilde{\mathbf{x}}, \tilde{y})$ to be in the vicinity of the training pair $(\mathbf{x}, y)$.

1. Gaussian vicinities: $v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i) = \mathcal{N}(\tilde{\mathbf{x}} - \mathbf{x}, \sigma^2 \mathbf{I}) \delta(\tilde{y} = y)$

Vicinal Risk Minimization (VRM)

P_δ is only one of the possibility to approximate the true distribution P. [Chapelle et al., 2001] proposed to approximate P by:

$$P_v(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i)$$

where v is vicinity distribution that measure the probability for a "virtual" pair $(\tilde{\mathbf{x}}, \tilde{y})$ to be in the vicinity of the training pair $(\mathbf{x}, y)$.

1. Gaussian vicinities: $v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i) = \mathcal{N}(\tilde{\mathbf{x}} - \mathbf{x}, \sigma^2\mathbf{I})\delta(\tilde{y} = y)$

which is equivalent to augmenting the training data with additive Gaussian noise

Keras Gaussian Layer for Gaussian Noise

```
import keras
from keras import Input
from keras.layers import GaussianNoise
dim_input=(256,256)
input_img = Input(shape=dim_input)
x =GaussianNoise(.01)(input_img)
...
```

Figure: Gaussian vicinities in Keras

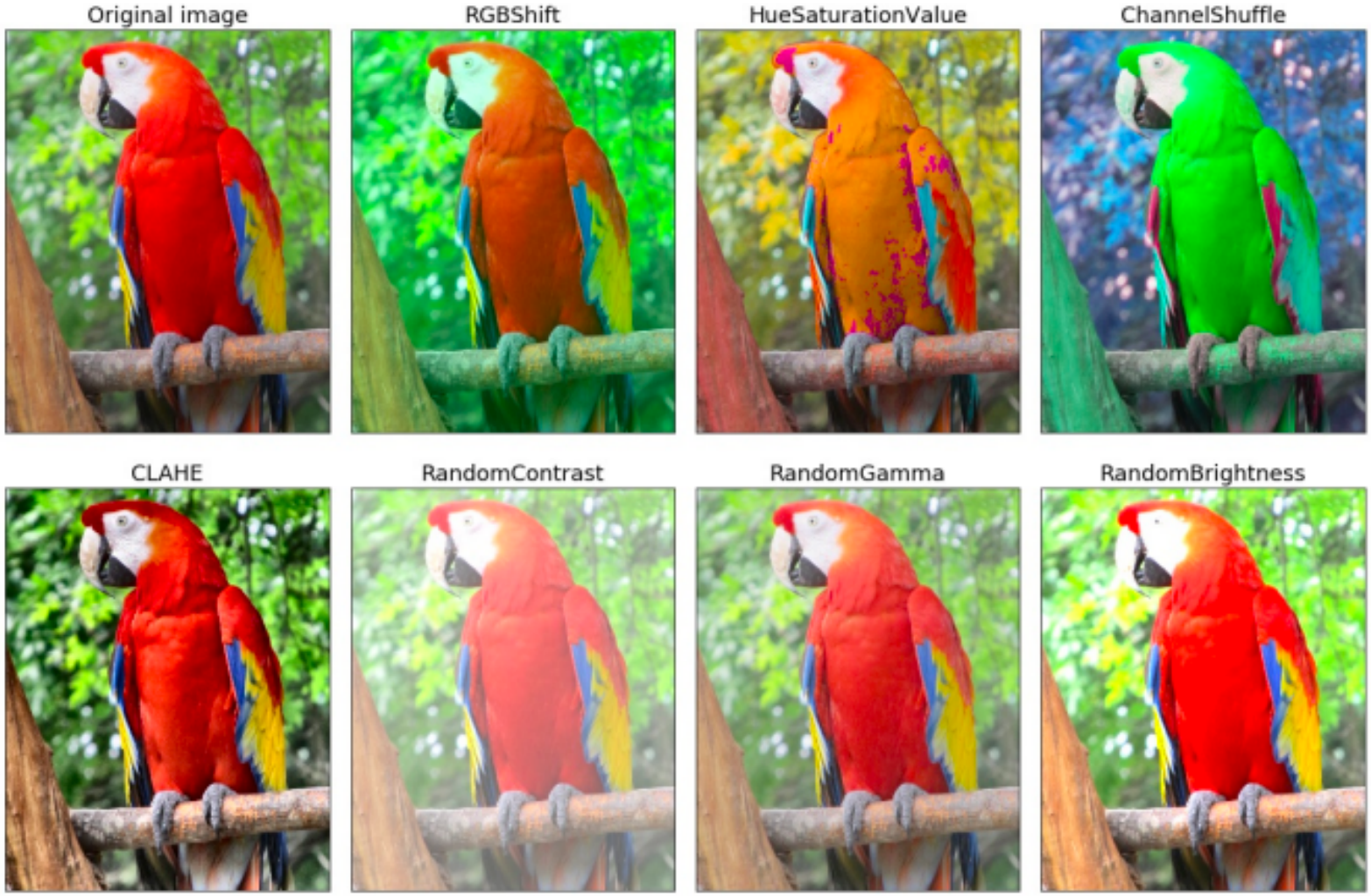
Why do we need Data-augmentation?

2 Data-augmentation based vicinities:
$P_{agg}(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\tilde{\mathbf{x}}, y_i)$, where $\tilde{\mathbf{x}}$ is a random transformation applied $\mathbf{x}$

Figure: Example of a set of image produce by random spatial transformations (translations, rotations, zooming, ...)

Why do we need Data-augmentation?

2 Data-augmentation based vicinities:
$P_{agg}(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\tilde{\mathbf{x}}, y_i)$, where $\tilde{\mathbf{x}}$ is a random transformation applied $\mathbf{x}$



Recent improvement on image augmentation.

- *CutOut Augmentation* removes contiguous section from the input image while training [DeVries and Taylor, 2017].

Recent improvement on image augmentation.

- *Mixup Augmentation* creates virtual examples by, $(\tilde{\mathbf{x}}, \tilde{y}) = \lambda(\mathbf{x}_i, y_i) + (1 - \lambda)(\mathbf{x}_j, y_j)$, λ is drawn from $\texttt{Beta}(\alpha)$ distribution. Where α controls the strength of interpolation in the input / target space[Zhang et al., 2017]

Recent improvement on image augmentation.

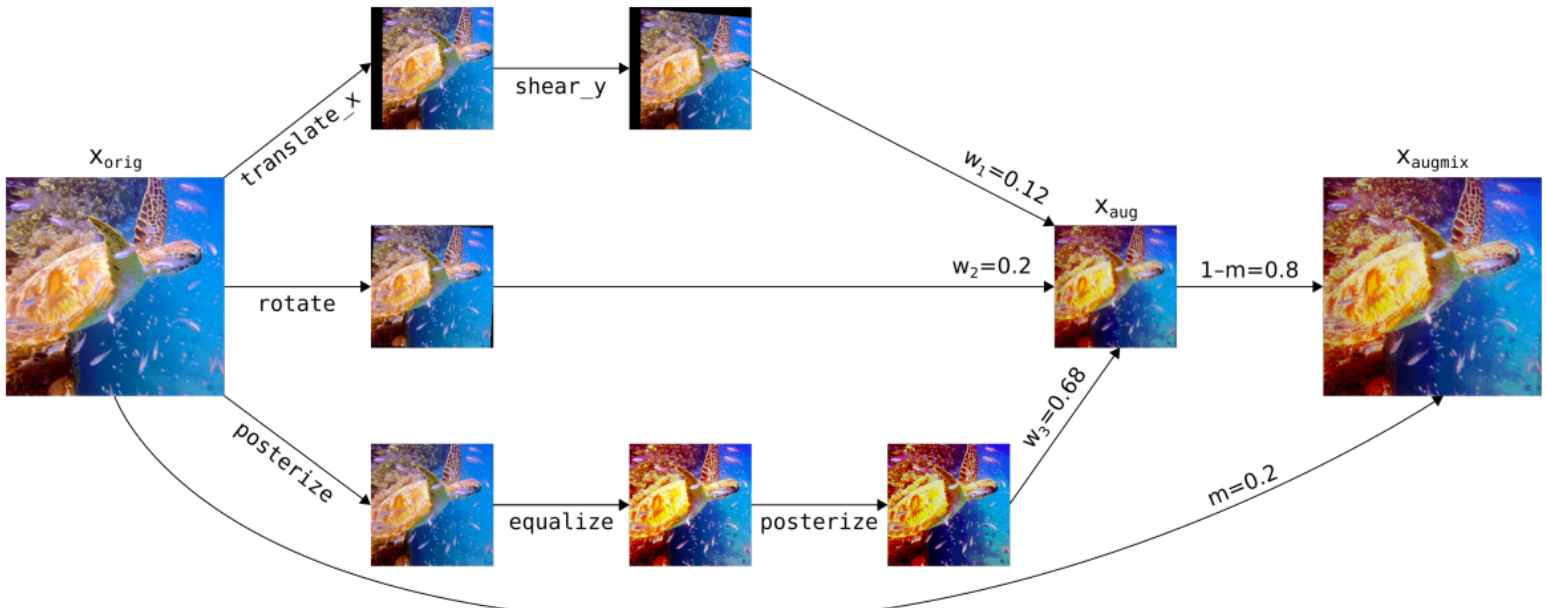
- `CutMix Augmentation` generates virtual examples by:

$$\tilde{\mathbf{x}} = M\mathbf{x}_i + (1 - M)\mathbf{x}_j$$
$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j$$

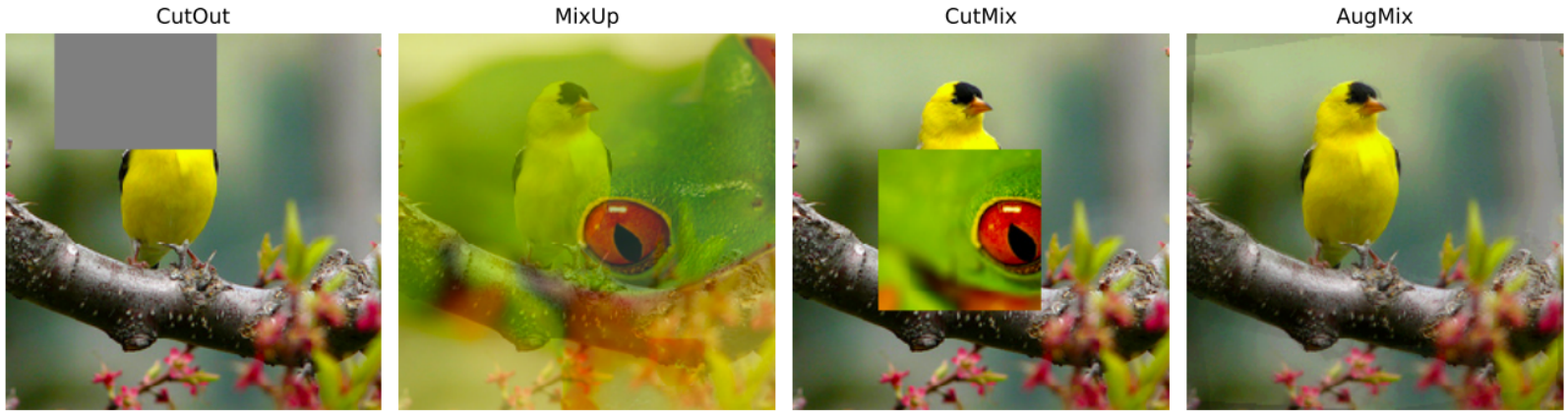
where M is a binary mask(often square) indicating the Cutout and the fill-in regions from the two randomly drawn images [Yun et al., 2019]. As in Cutout augmentation, λ is drawn from $\texttt{Beta}(\alpha)$ distribution.

Recent improvement on image augmentation.

- AugMix Augmentation [Hendrycks et al., 2019]: Given a composition of augmentation operators, the images are mixed using a random element-wise convex combination. A term of Consistency loss to ensure smoother neural network response is recommended, i.e: $L(\mathbf{x}, y) + \lambda JS(\mathbf{x}, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2)$, where $JS(\mathbf{x}, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2) = \frac{1}{3}(KL(\mathbf{x}||M) + KL(\mathbf{x}||M) + KL(\mathbf{x}||M))$, and KL is the Kullback–Leibler divergence evaluated in the logits of the model and M is the average of three logits.



Summarising: Image Augmentation



- Augmentations = Vicinal Risk Minimization (VRM)
- Reported empirical results in CIFAR10/100: (augmix > cutmix > mixup > cutout > baseline)
- Occlusion are important in image recognition task.
- Augmix motivates a smooth neural network response in the space of transformation.
- Define your own data augmentation according to the application.

Regularization vs Overfitting

1. Classical Regularization ,a.k.a. Weight Decay (Direct Model Regularization)
2. Early Stopping (Regularization in the training process)
3. More data! : Data Augmentation / Adversarial Examples (Regularization on data)
4. Summing-up: Dropout (Indirect regularization on model during training)

How can we reduce the variance of ?

- Remember: given N i.i.d. observations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ each of them with variance equal to σ^2.
- What is the variance of $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$?

How can we reduce the variance of ?

- Remember: given N i.i.d. observations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ each of them with variance equal to σ^2.
- What is the variance of $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$? $\frac{\sigma^2}{N^2}$
- Classical Approach: Train model on different training sets, and use the mean of predictions as final model of prediction.

Dropout: [Srivastava et al., 2014]

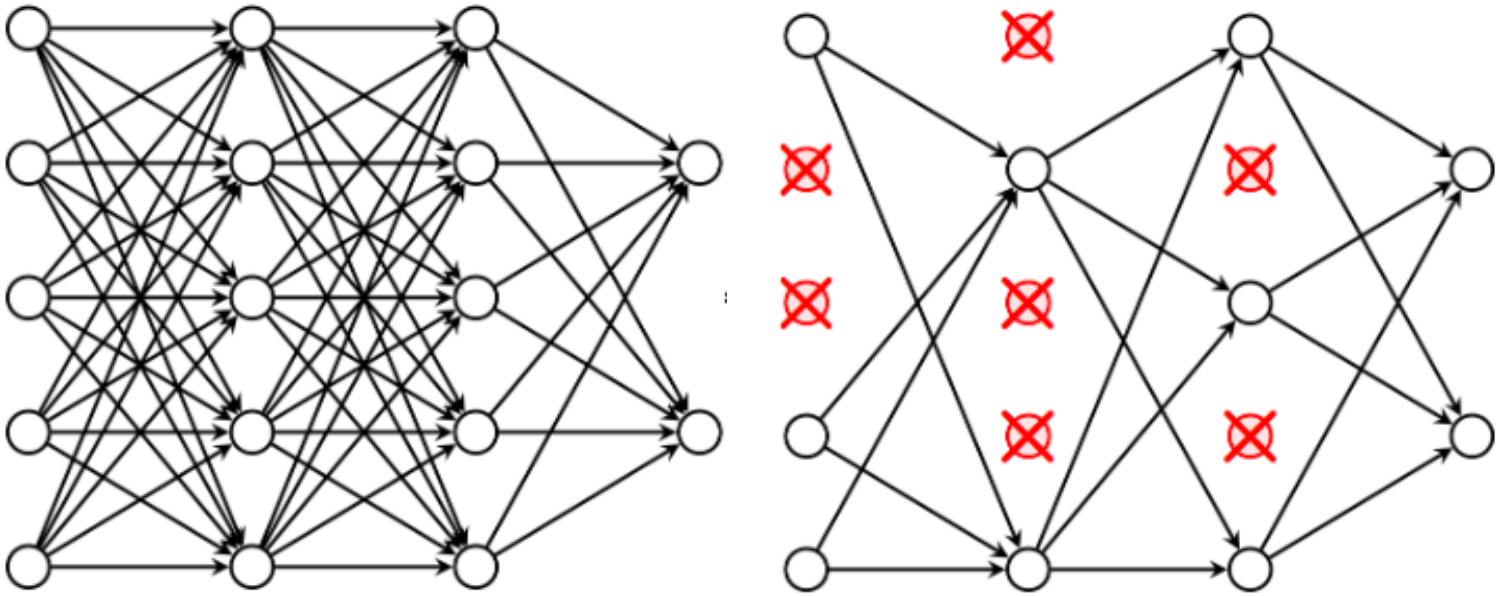


Figure: Left: No Dropout Right: Dropout

- Avoid memorization!
- Dropout forces to learn more robust features that are useful in conjunction with many different random subsets.
- With H hidden units, each of which can be dropped, we have 2^H possible models!

Dropout

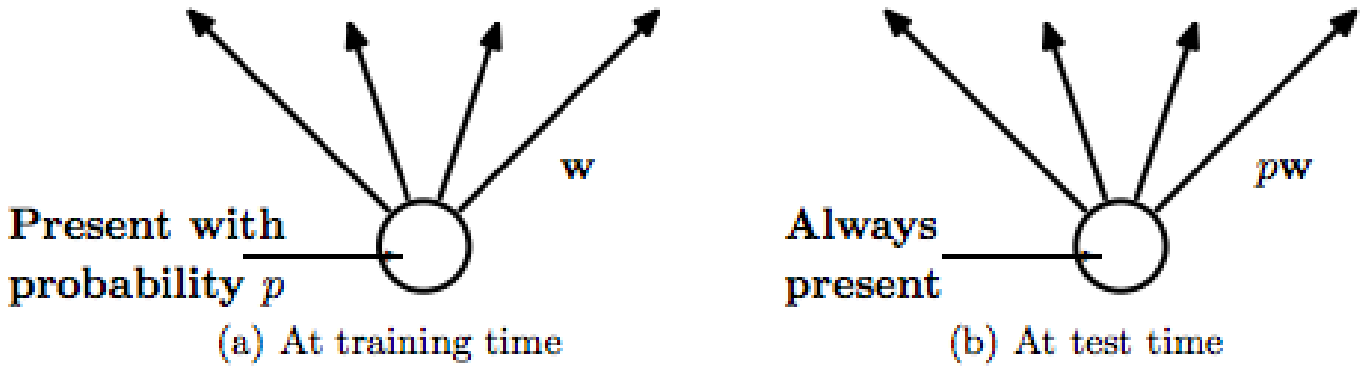


Figure: Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{W}$. Right: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Dropout

```
from keras.layers import Dropout
...
x=Dense(50, activation='relu')(x)
x=Dropout(0.5)(x)
...
```

Figure: Dropout in Keras. Layer with a dropout of $p = .5$

Contents

Gradient Descent: the intution [Cauchy, 1847]

Given a dataset $\mathbf{X}$ and a model with parameters $\boldsymbol{\theta}$, we would like to find the best values for $\boldsymbol{\theta}$ such that minimize a given loss function L evaluated on $\mathbf{X}$.

Algorithm 1 pseudocode gradient descent

given initial learning rate $\eta \in \mathbb{R}$ and dataset $\mathbf{X}$
initialize time step $t = 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^P$,
repeat
    t=t+1
    $\mathbf{g}_t = \nabla L_i(\mathbf{X}, \boldsymbol{\theta}_{t-1})$ `Return parameter gradient`
    $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_i$

We note that in this first version, the gradient is calculated for the entire training set.

Stochastic Gradient Descent [Robbins and Monro, 1985]

Algorithm 2 pseudocode for stochastic gradient descent

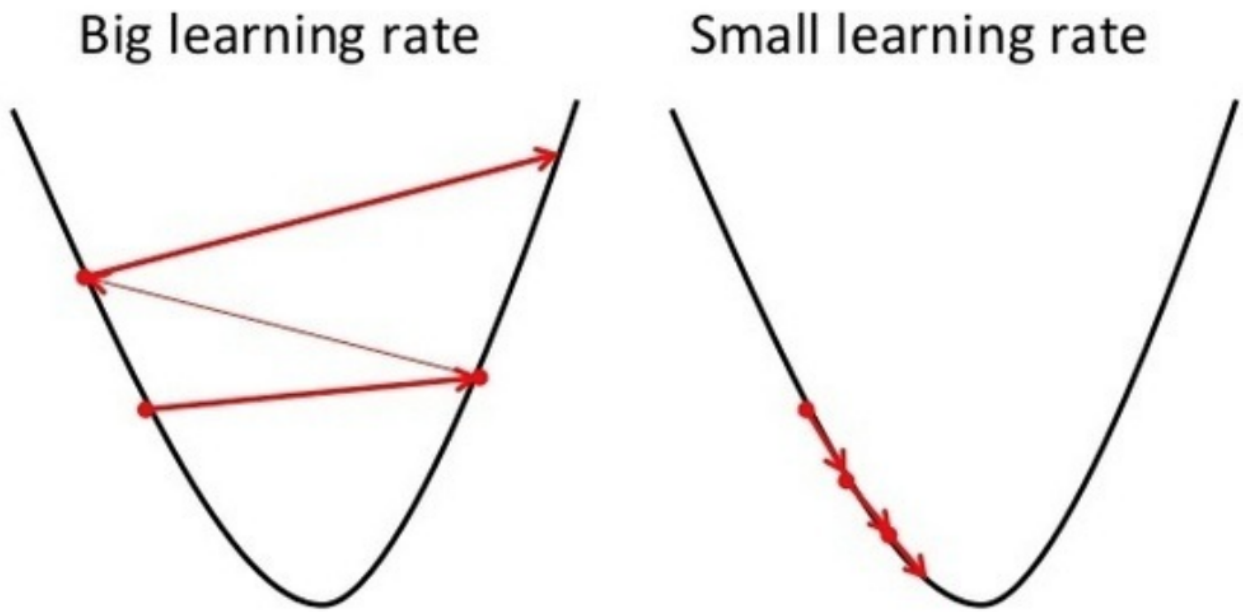
given initial learning rate $\eta \in \mathbb{R}$ and dataset $\mathbf{X}$
initialize time step $t = 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^P$,
repeat
    t=t+1
    $\mathbf{X}_t = \texttt{SelectBatch}(\mathbf{X})$ `Select a batch from data, whole data, only one, ...`
    $\mathbf{g}_t = \nabla L_i(\mathbf{X}_t, \boldsymbol{\theta}_{t-1})$ `Return parameter gradient`
    $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_i$

Here, we draw m samples to calculate the gradient.

SelectBatch($\mathbf{X}$)

- The larger the batch-size m, the more accurate the gradient is, but the longer the computation takes.
- Optimization algorithms converge faster (in terms of total number of computation) for $m < N$.
  - Better more approximate updates than few very precise updates.
  - Training sets are partly redundant.
- Small batch sizes have a regularizing effect.
- Too small batch sizes underutilize the parallel hardware architectures.
- The optimal mini-batch-size also depends on the algorithm.

The step size η: a critical parameter



- If η is set too low, the optimization takes a long time, and the algorithm can get stuck in a local minimum.
- If η is set too high, a good path might not be found in regions of high curvature.

Contents

SGD with Learning Rate Schedule

Algorithm 3 pseudocode for SGD with Learning Rate Schedule

given initial learning rate $\eta \in \mathbb{R}$ and a dataset $\mathbf{X}$
initialize time step $t = 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^P$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
    t=t+1
    $\mathbf{X}_t = \texttt{SelectBatch}(\mathbf{X})$ `Select a batch from data, whole data, only one, ...`
    $\mathbf{g}_t = \nabla L_i(\mathbf{X}_t, \boldsymbol{\theta}_{t-1})$ `Return parameter gradient`
    $\eta_t = \text{SetScheduleMultiplier}(t)$ Can be fixed,decay, warm restarts, ...
    $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \eta \mathbf{g}_t$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_i$

The schedule can take various forms.

Schedules 1/3

- One option is to successively reduce the learning rate over time.
- Linear rate decay:

$$\eta_k = \left(1 - \frac{k}{\tau}\right)\eta_0 + \frac{k}{\tau}\eta_\tau$$

  until iteration τ. After this, the learning rate remains constant.
- The underlying idea is that in the beginning, the algorithm should explore the parameter space in order to identify a good region.
- In order to further increase and to avoid bouncing between different slopes, we need to successively decrease the learning rate.
- This is the simplest schedule, and is widely used in practice.

Schedules 2/3

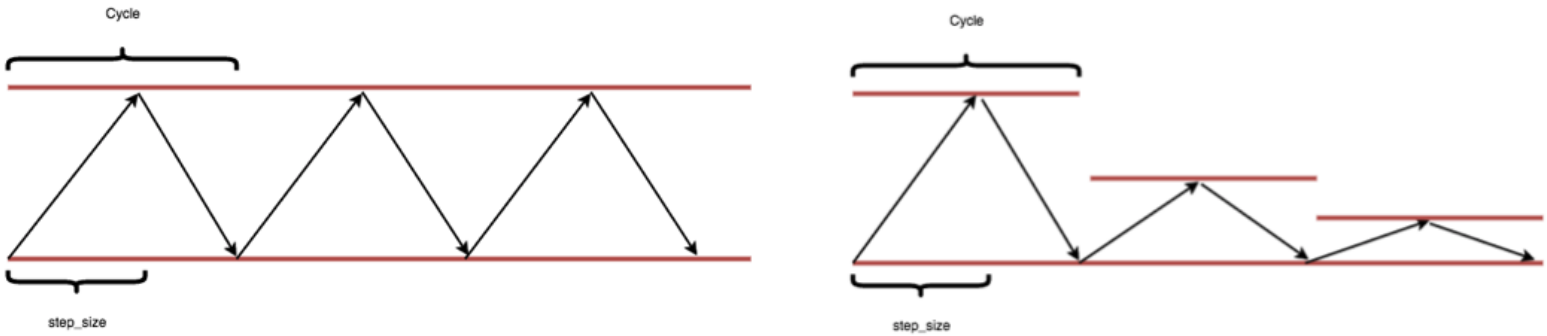


Figure: Circular Schedules [Smith, 2017]

- Circular schedules with or without decay can be used to move out sharp minima.
- Sharp minima are suspected to lead to poor generalization (small changes in parameters lead to strongly increasing losses).
- Increasing the learning rate can also allow for "more rapid" traversal of saddle point plateaus.

Schedules 3/3

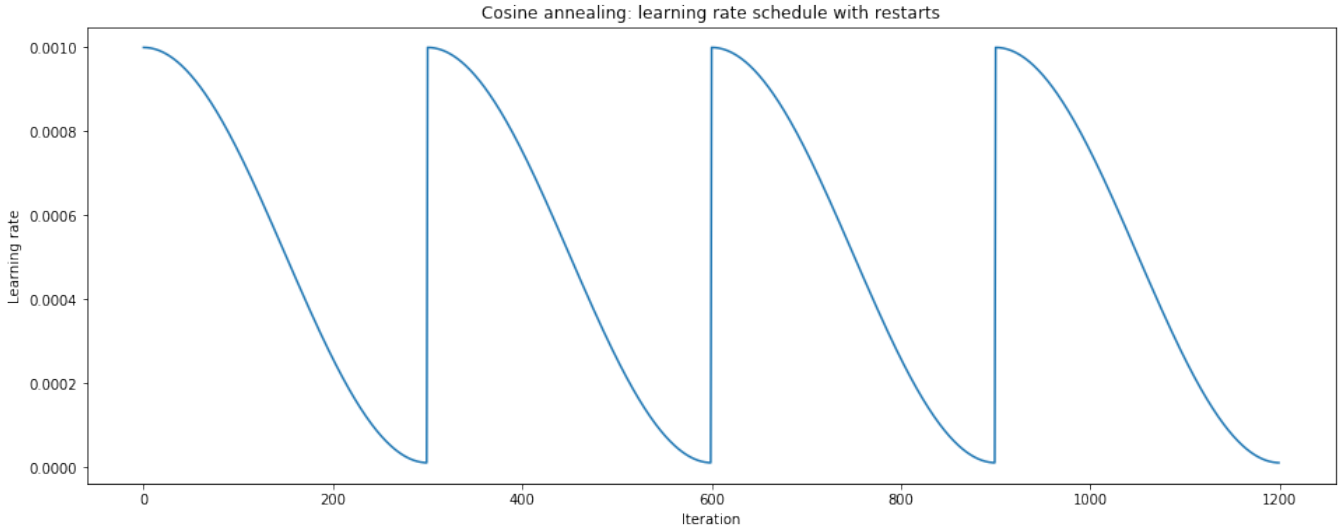


Figure: Stochastic Gradient Descent with Warm Restarts (SGDR) [Loshchilov and Hutter, 2016]

- This strategy also aims at moving out of local minima, in particular sharp local minima.
- An additional idea is to keep track of the models at different minima (end of the cycles).
- These models can then be averaged. We therefore get an ensemble of Neural Networks by only one training procedure [Huang et al., 2017].

Optimizer in Keras

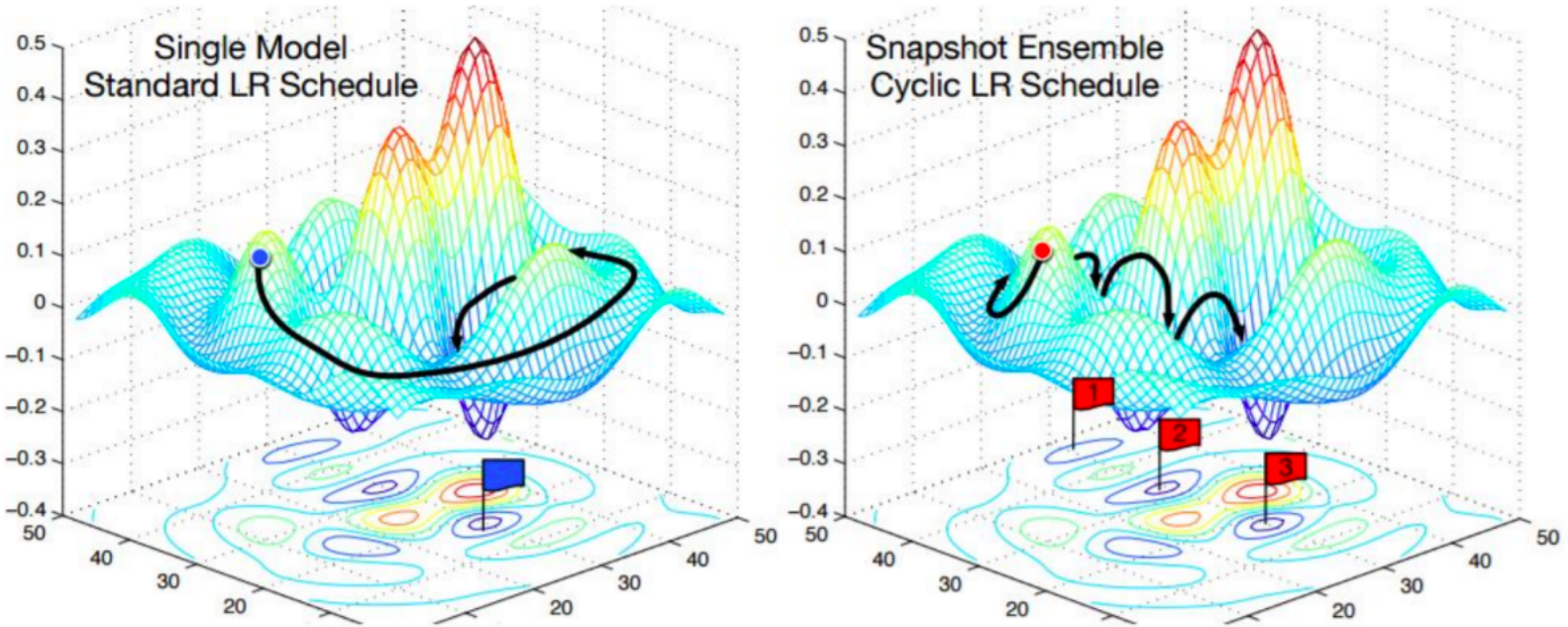


Figure: [Huang et al., 2017]

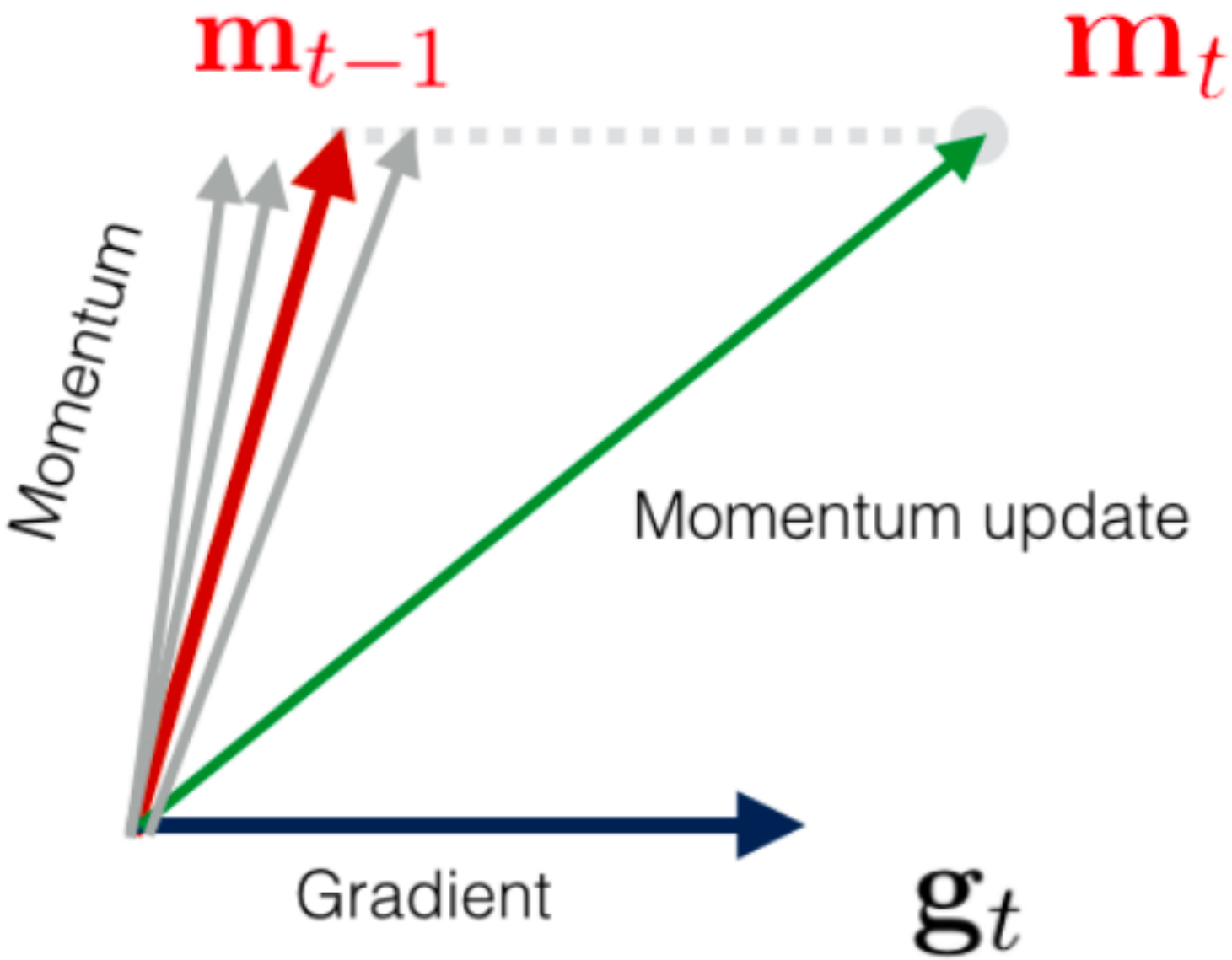
```
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

SGD with momentum [Qian, 1999]

Algorithm 4 pseudocode for stochastic gradient descent with Momentum

given initial learning rate $\epsilon \in \mathbb{R}$, dataset $\mathbf{X}$, momentum factor $\beta_1 \in \mathbb{R}$
initialize time step $t = 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^P$, first momentum factor $\mathbf{m}_{t=0} = 0 \in \mathbb{R}^P$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
    t=t+1
    $\mathbf{X}_t = \texttt{SelectBatch}(\mathbf{X})$ `Select a batch from data`
    $\mathbf{g}_t = \nabla L_t(\mathbf{X}_t, \boldsymbol{\theta}_{t-1})$ `Return parameter gradient`
    $\eta_t = \text{SetScheduleMultiplier}(t)$ Can be fixed,decay, warm restarts, ...
    $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + \eta_t \eta \mathbf{g}_t$
    $\boldsymbol{\theta}_i = \boldsymbol{\theta}_{t-1} - \mathbf{m}_t$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_i$

Momentum


$\mathbf{m}_{t-1}$
$\mathbf{m}_t$
Momentum
Momentum update
Gradient
$\mathbf{g}_t$

Momentum

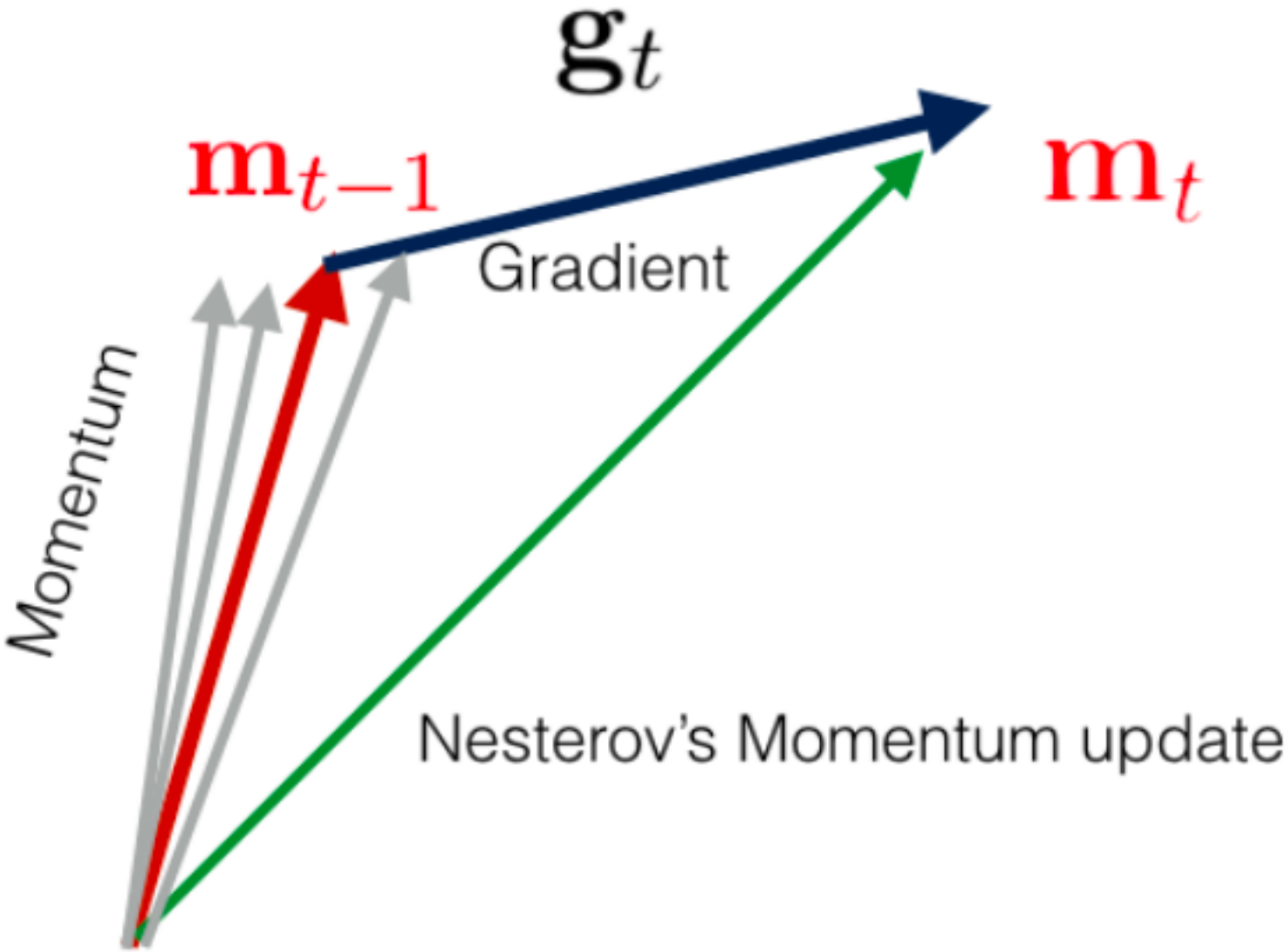
- Objective: accelerate optimization
- Mechanism: accumulation of an exponentially decaying moving average
- If the gradient always points to the same direction, it is amplified.
- In the case of abrupt changes in the gradient, the update direction is smoothened.

SGD with Nesterov's momentum [Nesterov, 1983]

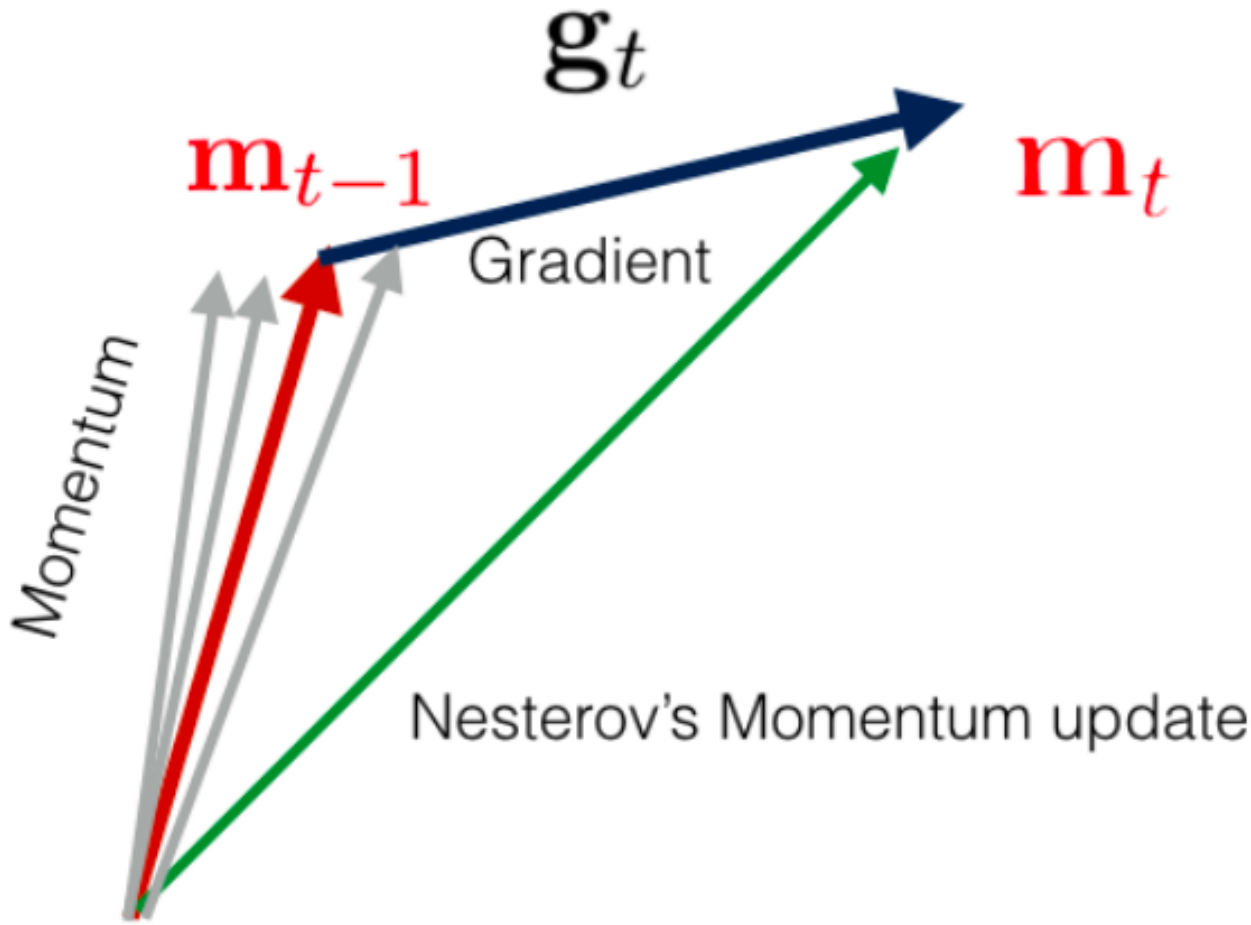
Algorithm 5 pseudocode for stochastic gradient descent with Momentum

given initial learning rate $\epsilon \in \mathbb{R}$, dataset $\mathbf{X}$, momentum factor $\beta_1 \in \mathbb{R}$
initialize time step $t = 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^P$, first momentum factor $\mathbf{m}_{t=0} = 0 \in \mathbb{R}^P$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
    t=t+1
    $\mathbf{X}_t = \texttt{SelectBatch}(\mathbf{X})$ `Select a batch from data`
    $\mathbf{g}_t = \nabla L_t(\mathbf{X}_t, \boldsymbol{\theta}_{t-1} + \mathbf{m}_{t-1})$ `Return parameter gradient`
    $\eta_t = \text{SetScheduleMultiplier}(t)$ Can be fixed,decay, warm restarts, ...
    $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + \eta_t \eta \mathbf{g}_t$
    $\boldsymbol{\theta}_i = \boldsymbol{\theta}_{t-1} - \mathbf{m}_t$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_i$

Nesterov's Momentum



Nesterov's Momentum



- The difference to standard momentum is where the gradient is evaluated.
- This can lead to further acceleration. For the optimization of NN, this does not play a major role.

Table: Summary: Some Optimizers I

Method	Update
SGD	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}(\boldsymbol{\theta}_t)$
Momentum	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{m}_t,\ \mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} - \eta \mathbf{g}(\boldsymbol{\theta}_t)$
Nesterov Mom.	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{m}_t\ ,\ \mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} - \eta \mathbf{g}(\boldsymbol{\theta}_t + \beta_1 \mathbf{m}_{t-1})$

Motivation

1. SGD bounces around in high curvature directions and makes slow progress in low curvature directions.
2. This can happen with highly correlated parameters.

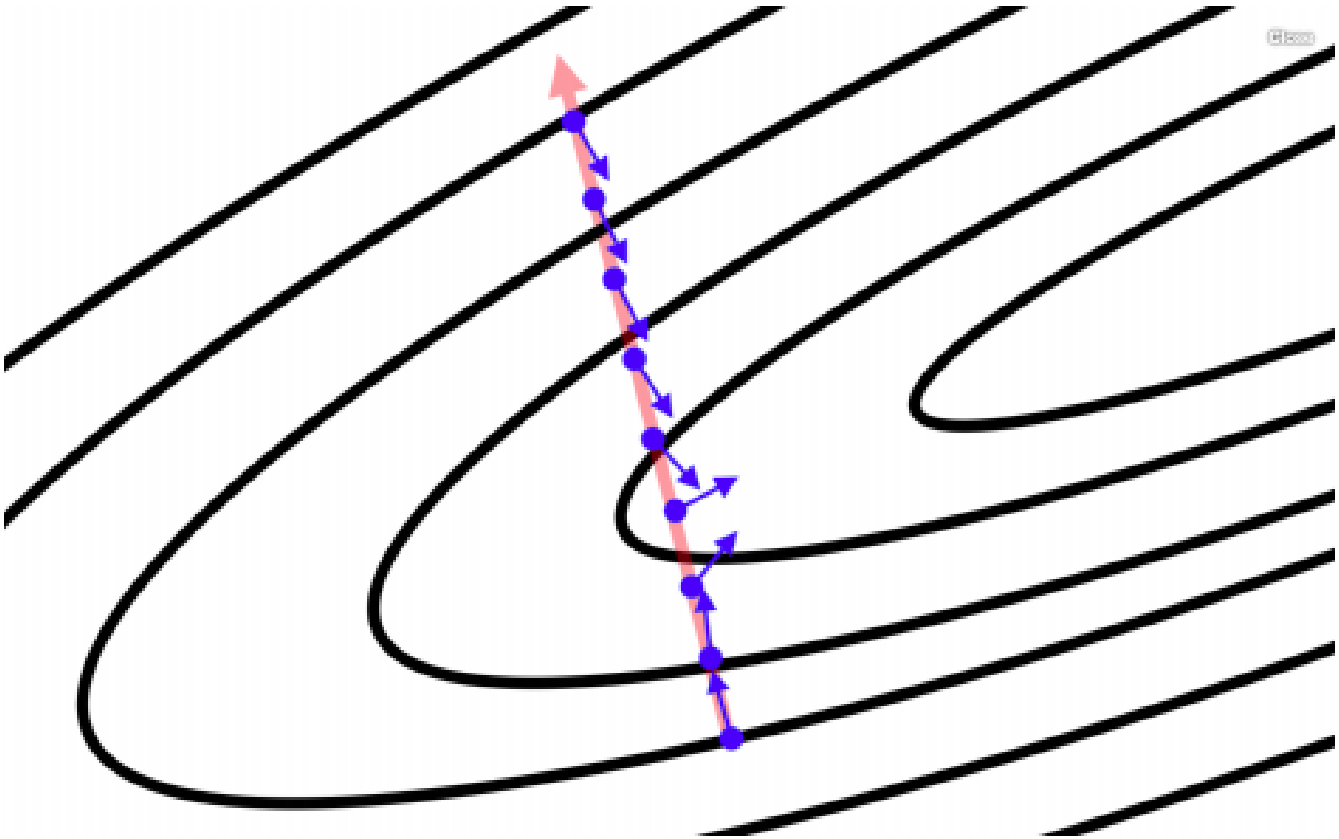


Figure: SGD

First Order Gradient Descend

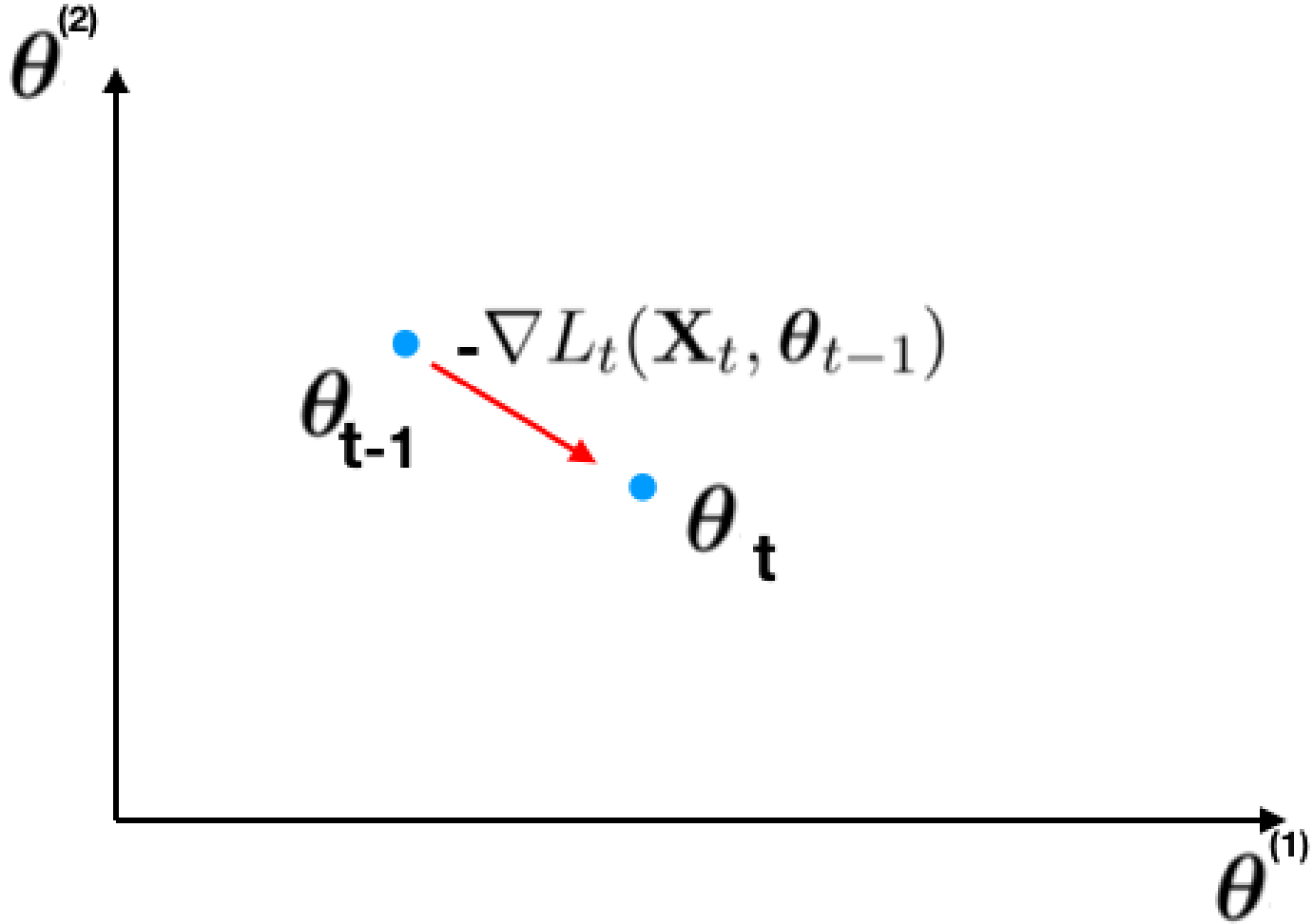


Figure: Gradient Descend

Second Order Gradient Descend

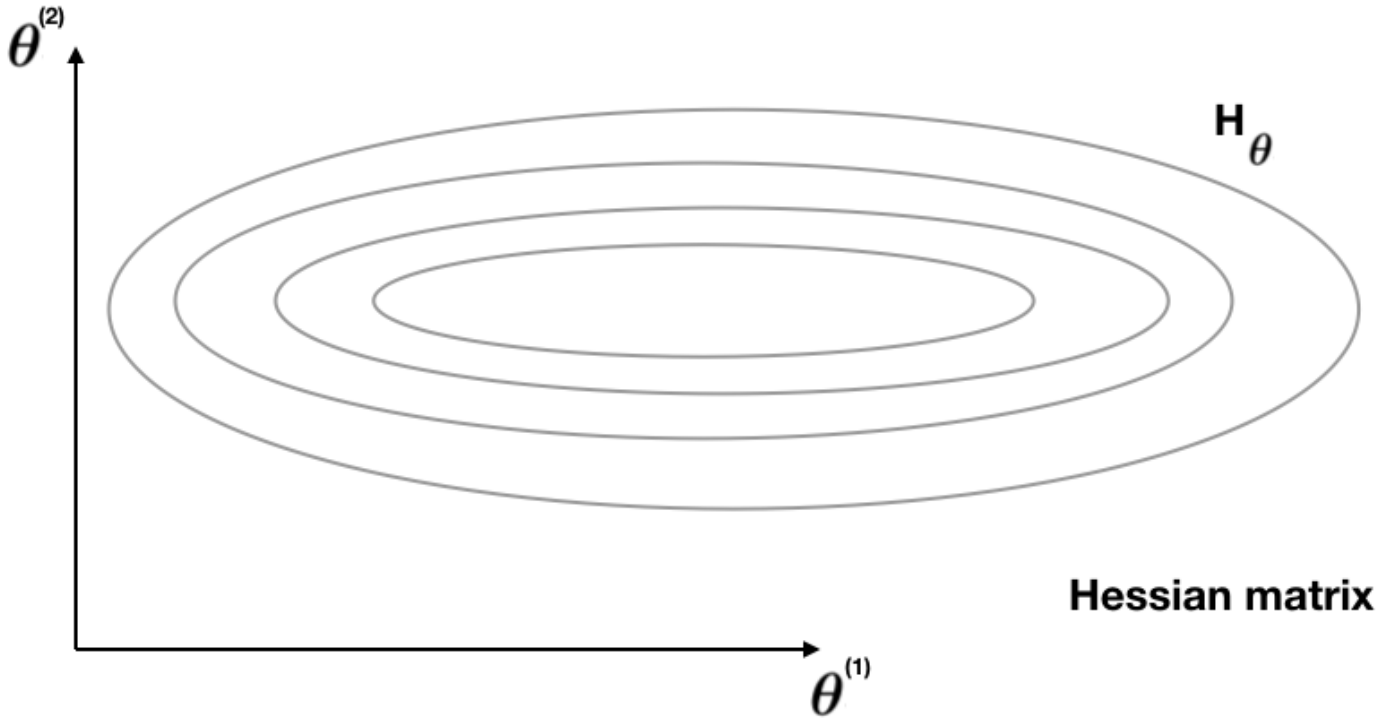


Figure: Hessian Matrix

Second Order Gradient Descend

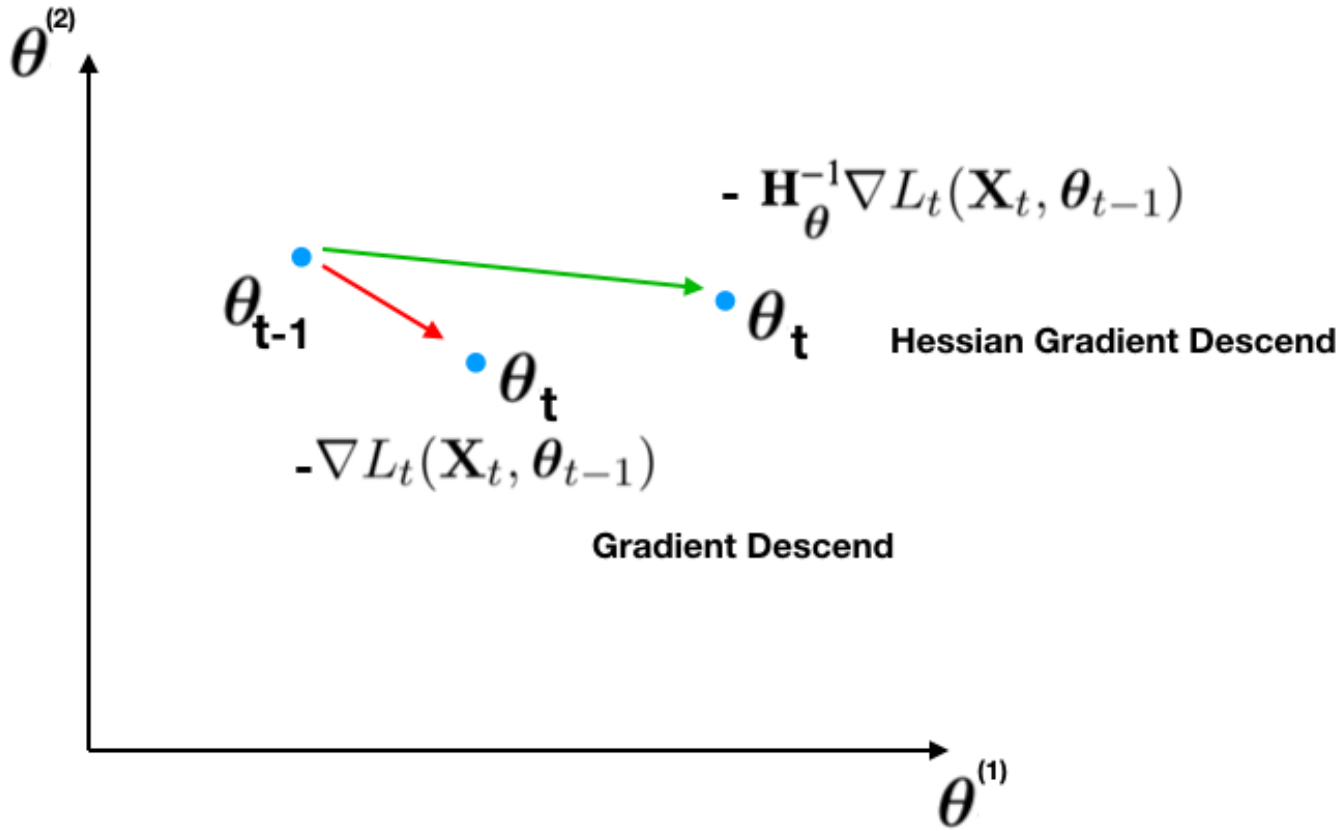


Figure: Second Order Gradient Descend

Information theory point of view

Table: Natural Gradient Descend [Amari et al., 2000]

Method	Update
SGD	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}(\boldsymbol{\theta}_t)$
Natural Gradient Descend	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{\Sigma}_{\boldsymbol{\theta}_t}^{-1} \mathbf{g}(\boldsymbol{\theta}_t)$
Second Order Gradient Descend	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{H}_{\boldsymbol{\theta}}^{-1} \mathbf{g}(\boldsymbol{\theta}_t)$

Two different interpretations for similar methods:

1. Second order optimization: Newton Algorithm [LeCun et al., 2012]
2. Gradient Descend in the Riemannian Manifold [Amari, 1998] = Fisher Information Matrix (empirical version)

Practical Point of view

What is the practical problem with the computation of Hessian matrix?

1. $\mathbf{H}$ is a $P \times P$
2. How to calculate $\mathbf{H}$ on batches?
3. $\mathbf{H}$ can be bad-conditioning, i.e., the calcule of inverse matrix can produce numerical error.

Practical Solutions

1. Only estimate diagonal element of Hessian matrix.
2. Use block estimation by layer.
3. Use variance of gradient as a normalization of scale.

Adagrad [Duchi et al., 2011],

The parameter update is defined as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{G_{ii,t} + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t) \tag{5}$$

where $G_{ii,t}$ is the sum of the squares of the gradients of $\boldsymbol{\theta}_i$ up to time step t while ϵ is a smoothing term that avoids division by zero. $G_{ii,t} = \sum_{j=1}^{t} \mathbf{g}_i^2(\boldsymbol{\theta}_j)$, where $\mathbf{g}_i$ denote the i-th component of $\mathbf{g}$.

- Each dimension is thus weighted by its gradient component: the learning rate is lower for features with a "high-gradient-past".
- This allows slowly evolving features to get a higher learning rate.
- Note that only elements in the diagonal of $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}$ are estimated.

RMSprop [Hinton et al., 2012]

In RMSprop [Hinton et al., 2012] proposes a rule update model's parameter by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\texttt{RMS}(G_{ii})_t + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t) \tag{6}$$

where $\texttt{RMS}(G_{ii,t})$ is a momentum estimation of $\sqrt{G_{ii,t} + \epsilon}$, i.e., defining the sequence

$$E(\mathbf{g}^2)_t = \rho E(\mathbf{g}^2)_{t-1} + (1-\rho)\mathbf{g}_t^2,$$

and we define $\texttt{RMS}(G_{ii,t}) = E(\mathbf{g}^2)_t + \epsilon$.

RMSprop

- RMSprop replaces the total sum (Adagrad) by an expontentially weighted moving average.
- This loss in memory is important in the presence of strong non-convexities: the landscape can change profoundly, and the extreme past is probably not useful in such a case.

Adadelta [Zeiler, 2012]

However, the unit in the learning rate don't correspondent with unit in the denominator. Thus, Adadelta optimiser [Zeiler, 2012] update by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\texttt{RMS}(\partial\boldsymbol{\theta})_{t-1}}{\sqrt{\texttt{RMS}(G_{ii,t} + \epsilon)}}\mathbf{g}(\boldsymbol{\theta}_t) \tag{7}$$

here is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. $\boldsymbol{\theta}_i$ up to time step t

Table: Summary: Some Optimizers II

Method	Update
SGD	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}(\boldsymbol{\theta}_t)$
Momentum	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{m}_t,\ \mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} - \eta \mathbf{g}(\boldsymbol{\theta}_t)$
Nesterov Mom.	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{m}_t\ ,\ \mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} - \eta \mathbf{g}(\boldsymbol{\theta}_t + \beta_1 \mathbf{m}_{t-1})$
Adagrad	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{G_{ii,t} + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t)$
RMSprop	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\texttt{RMS}(G_{ii,t}) + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t)$
Adadelta	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\texttt{RMS}(\partial \boldsymbol{\theta})_{t-1}}{\sqrt{\texttt{RMS}(G_{ii})_t + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t)$

ADAM

Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum
[Kingma and Ba, 2014]

Table: Summary: Some Optimizers

Method	Update
SGD	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}(\boldsymbol{\theta}_t)$
Momentum	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{m}_t,\ \mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} - \eta \mathbf{g}(\boldsymbol{\theta}_t)$
Nesterov Mom.	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{m}_t\ ,\ \mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} - \eta \mathbf{g}(\boldsymbol{\theta}_t + \beta_1 \mathbf{m}_{t-1})$
Adagrad	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{G_{ii,t} + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t)$
RMSprop	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\mathtt{RMS}(G_{ii})_t + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t)$
Adadelta	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\mathtt{RMS}(\partial \boldsymbol{\theta})_{t-1}}{\sqrt{\mathtt{RMS}(G_{ii})_t + \epsilon}} \mathbf{g}(\boldsymbol{\theta}_t)$
ADAM	Momentum and RMSprop
NADAM	Nesterov Momentum and RMSprop

Contents

Initialization Issues

1. **Gaussian**: From Imagenet 2012, [Krizhevsky et al., 2012] recommends initialization with $\mathcal{N}(0, .01)$ and adding bias equal to one for some layers become very popular. It is not possible to train very deep network from scratch with it [Simonyan and Zisserman, 2014]. The problem is caused by the activation (and/or) gradient magnitude in final layers [He et al., 2016].
2. **Glorot**: [Glorot and Bengio, 2010] proposed a formula for estimating the standard deviation on the basis of the number of input and output channels of the layers under assumption of no non-linearity between layers. Despite invalidity of the assumption, Glorot initialization works well.
3. **Orthogonal**: Saxe et al. [Saxe et al., 2014] showed that orthonormal matrix initialization works much better for linear networks than Gaussian noise, which is only approximate orthogonal. It also work for networks with non-linearities.
4. Recommended lecture: All you need is a good init [Mishkin and Matas, 2016]

Vanishing / Exploding Gradients

Vanishing gradients:

1. If the gradient of a node vanishes (becomes 0), the entry value is not updated.
2. This means that there is no information that flows through the node backwards.
3. In this case, the networks cannot learn anymore: the network becomes untrainable.
4. This problem is particularly striking for very deep networks.

In exploding gradients, the weights explode and become too big. Usually related to a gradient descent approach with huge steps,

Fighting against Vanishing/Exploding Gradient

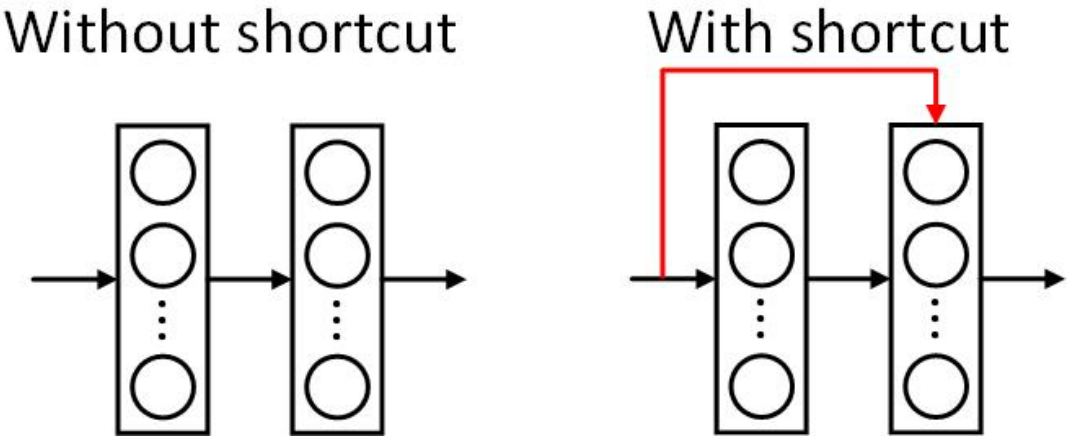
1. Gradient clipping: clips parameters gradients during backpropation by a maximum value or maximum norm

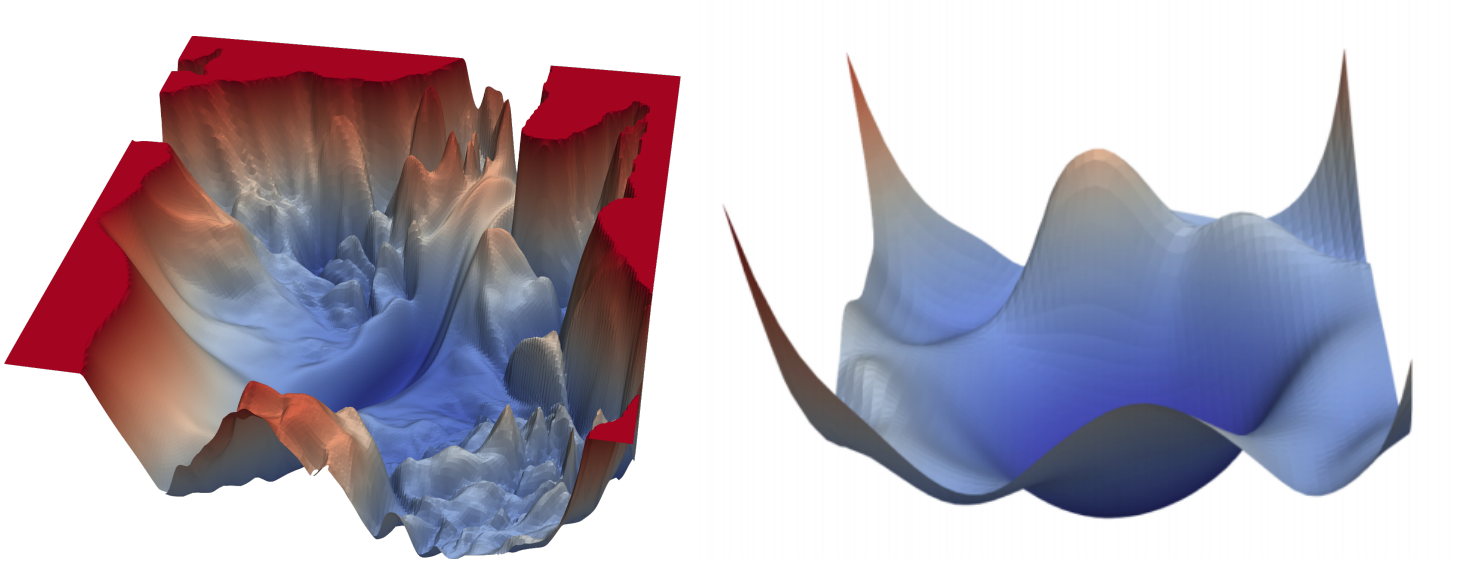
```
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum value of 0.5 and
# a minimum value of -0.5.
sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)

# All parameter gradients will be clipped to
# a maximum norm of 1.
sgd = optimizers.SGD(lr=0.01, clipnorm=1.)
```

2. Use `ReLu` or variants. They are much more robust to this problem, because a sigmoid maps a large input space to a $[0, 1]$ output. If the input is large, the gradient becomes small. ReLus do not suffer from this problem. `LeakyReLu` similar to ReLU, but it relaxes sparsity constraints by allowing small negative activation values.
3. Use of residual networks.

Fighting against Vanishing/ExplodingGradient: skip connections

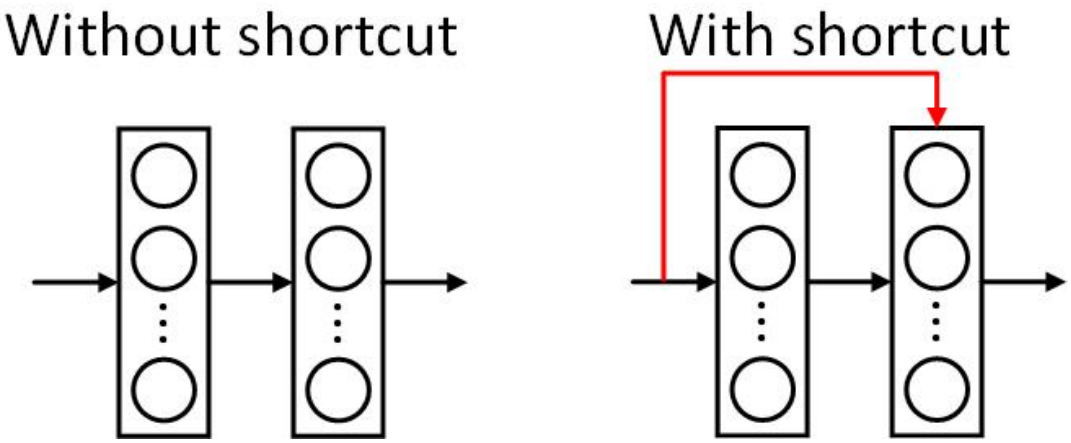
Skip connections or Shortcuts (Residual Networks):





Fighting against Vanishing/Exploding Gradient

3 Skip connections or Shortcuts (Residual Networks):



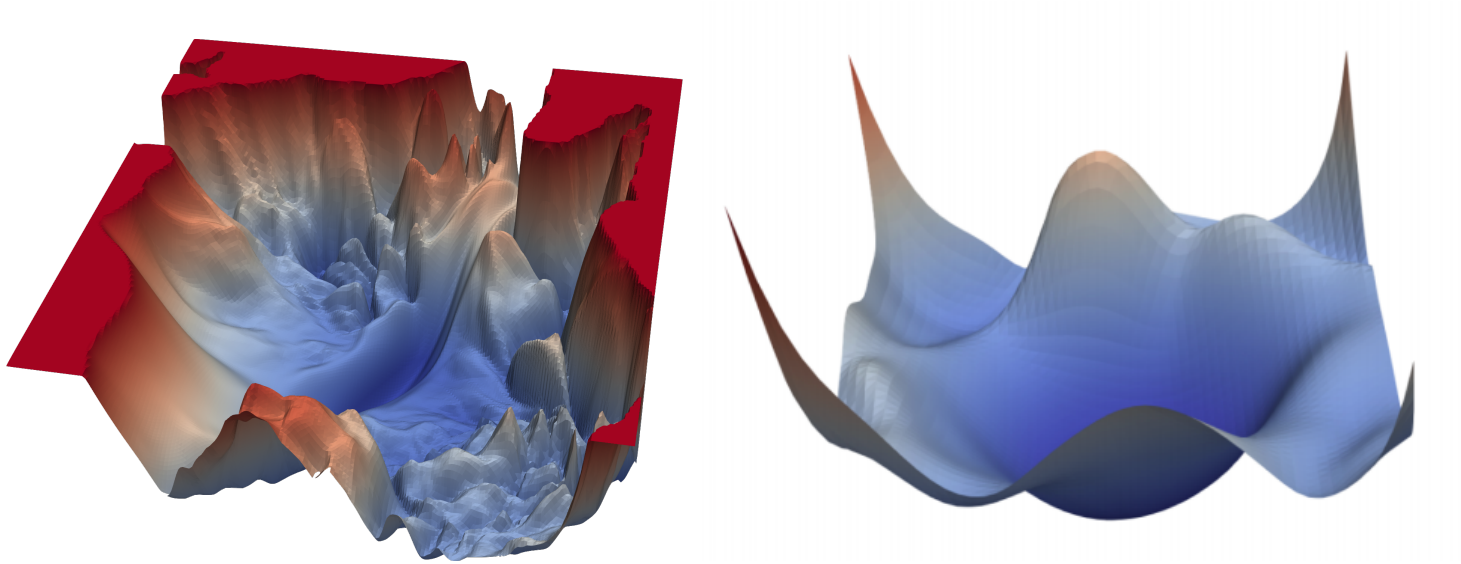


Figure: From [Li et al., 2017]

Fighting against Vanishing/Exploding Gradient

4 Avoid "stuck states" induced by activation function:

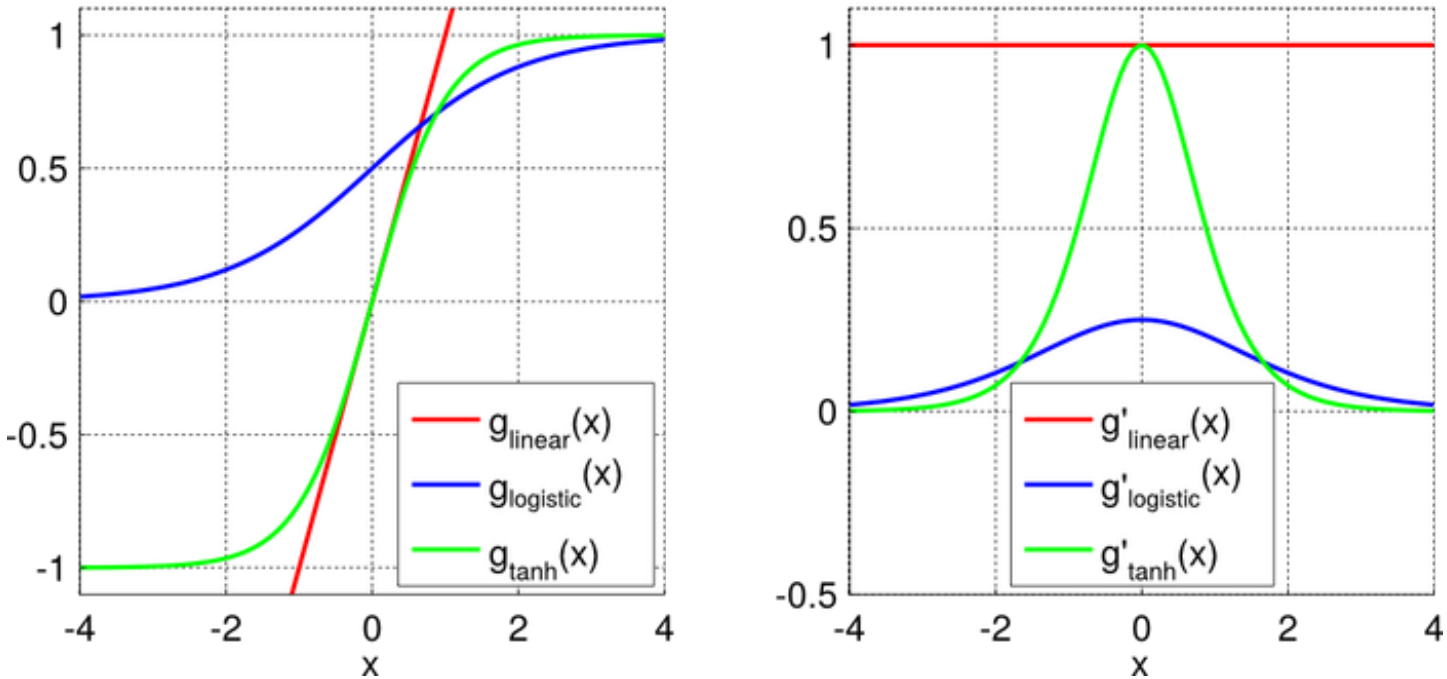


Figure: Left: Three activation function Right: Derivative of activation function.

Batch Normalization [Ioffe and Szegedy, 2015] [Mishkin and Matas, 2016]

BatchNorm (BN) is a transformation applied before the applying activation in a given layer, i.e. if x_i are the output of a layer for a mini-batch, the BN is defined by

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2}}, \qquad BN_{\gamma,\beta}(x_i) := \gamma \hat{x}_i + \beta$$

where μ_B and σ_B^2 are respectively the mini-batch mean and variance. γ scale and β shift a learned parameters.

- Moving values to zero (activation works better!)
- Large learning rates can scale the parameters which could amplify the gradients, thus leading to an explosion. In Batch normalization small changes in parameter to one layer do not get propagated to other layers.
- This makes it possible to use higher learning rates for the optimizers, which otherwise would not have been possible.
- It also makes gradient propagation in the network more stable.

Batch Normalization

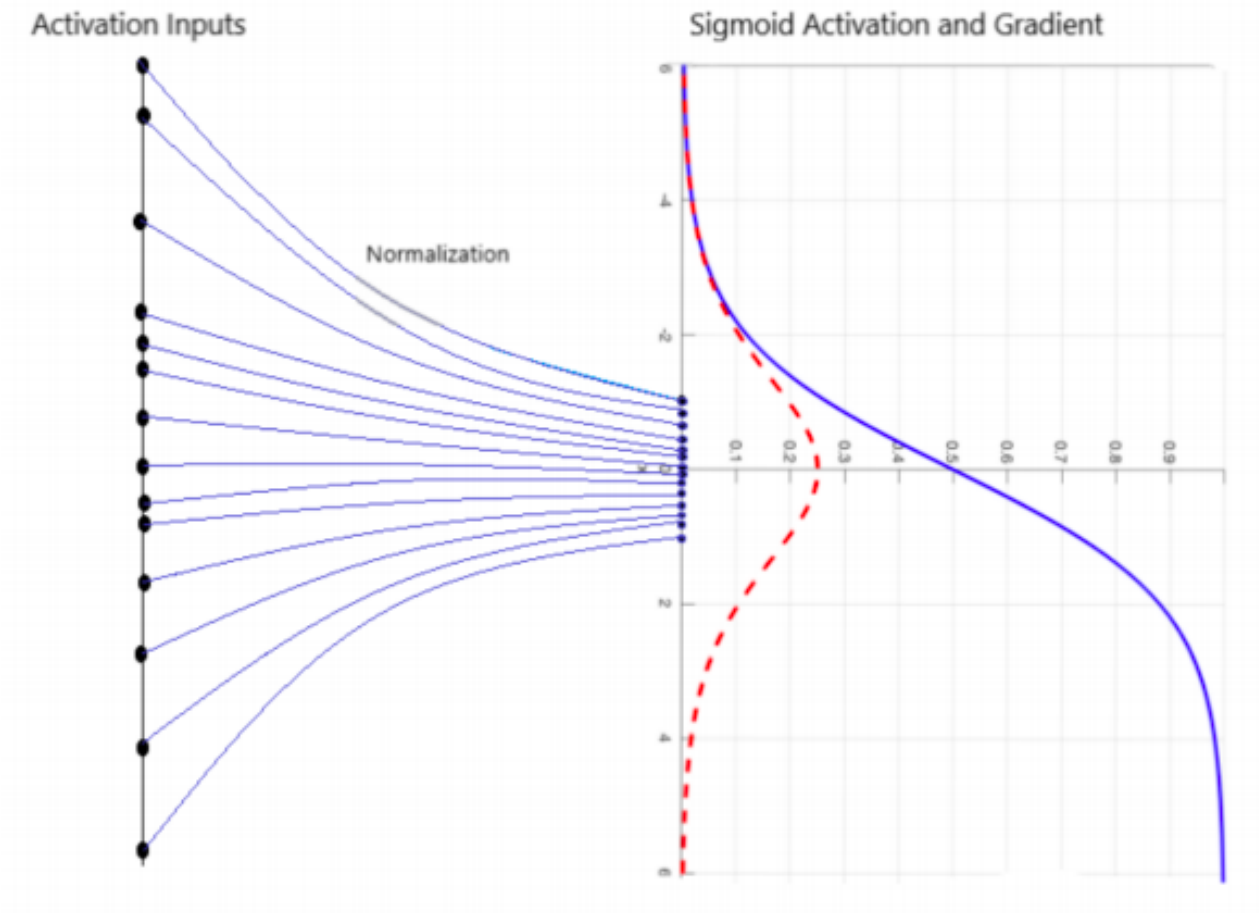


Figure: Image from "Intuit and Implement: Batch Normalization"

Other Normalizations

- Layer normalization (LN) : normalizes each feature of the activations to zero mean and unit variance.
- Group normalization (GN) : is also applied along the feature direction but unlike LN, it divides the features into certain groups and normalizes each group separately.
- Instance normalization: normalizes along each combination of feature and sample.
- Weight Standardization (WS): is transforming the weights of any layer to have zero mean and unit variance.

Other Normalizations

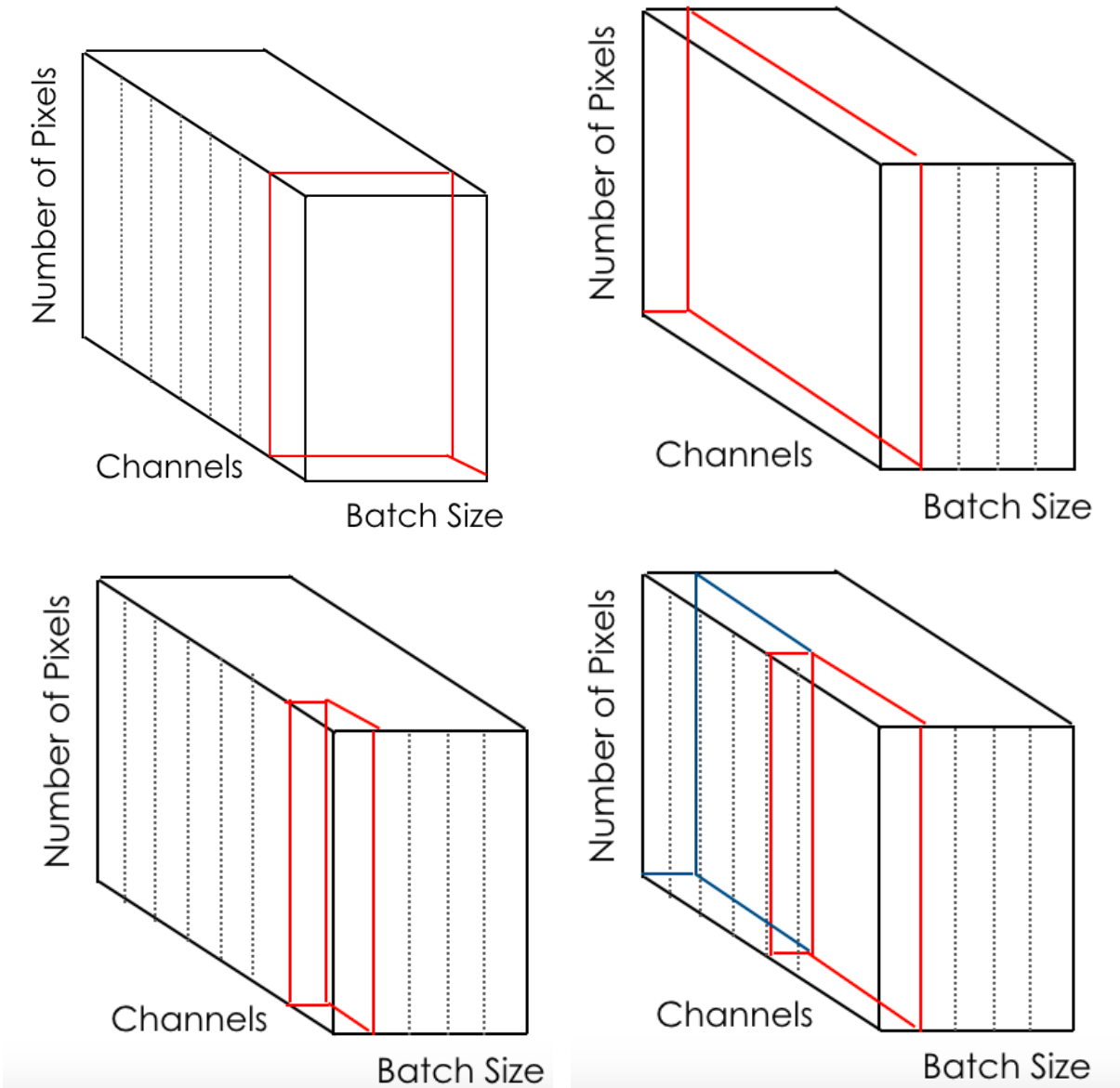
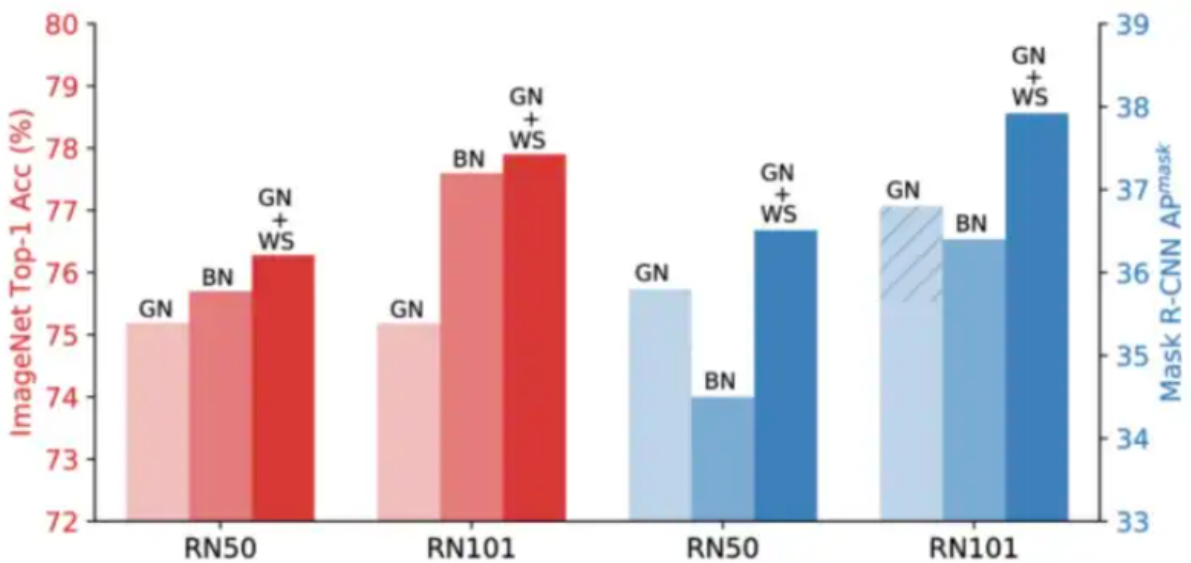


Figure: a) Batch Normalization, b) Layer Normalization, c) Instance Normalization and d) Group Normalization

Normalization effect

- All the normalization help to avoid vanishing gradient.
- Usually accelerate the convergence in number of epochs.
- Models with normalization overfits easily. It should be combine with strong regularizations methods.



Siyuan Qiao et al. Weight Standardization. GN+WS effect on classification and object detection task

Conclusions

1. More and more real Data + Data augmentation + Regularization on Training -¿ Use of large models!
2. Avoid Vanishing Gradient -¿ Normalization
3. Too many "cooking recipes" ... (Boring part of training DL models).

Contents

References I

[Amari, 1998] Amari, S.-I. (1998). Natural gradient works efficiently in learning. Neural computation, 10(2):251–276.

[Amari et al., 2000] Amari, S.-I., Park, H., and Fukumizu, K. (2000). Adaptive method of realizing natural gradient learning for multilayer perceptrons. Neural Computation, 12(6):1399–1409.

[Cauchy, 1847] Cauchy, A. (1847). Méthode générale pour la résolution des systèmes d?équations simultanées. arXiv preprint arXiv:1608.03983.

[Chapelle et al., 2001] Chapelle, O., Weston, J., Bottou, L., and Vapnik, V. (2001). Vicinal risk minimization. In Advances in neural information processing systems, pages 416–422.

References II

[DeVries and Taylor, 2017] DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121–2159.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 249–256.

References III

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778.

[Hendrycks et al., 2019] Hendrycks, D., Mu, N., Cubuk, E. D., Zoph, B., Gilmer, J., and Lakshminarayanan, B. (2019). Augmix: A simple data processing method to improve robustness and uncertainty. arXiv preprint arXiv:1912.02781.

[Hinton et al., 2012] Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[Huang et al., 2017] Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. (2017). Snapshot ensembles: Train 1, get m for free. ICLR.

References IV

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. ICLR.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105.

[LeCun et al., 2012] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In Neural networks: Tricks of the trade, pages 9–48. Springer.

References V

[Leshno et al., 1993] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural networks, 6(6):861–867.

[Li et al., 2017] Li, H., Xu, Z., Taylor, G., and Goldstein, T. (2017). Visualizing the loss landscape of neural nets. arXiv preprint arXiv:1712.09913.

[Loshchilov and Hutter, 2016] Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983.

[Mishkin and Matas, 2016] Mishkin, D. and Matas, J. (2016). All you need is a good init. ICML.

References VI

[Nesterov, 1983] Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $o(1/k^2)$. Dokl. Akad. Nauk SSSR, 269:543–547.

[Qian, 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural networks, 12(1):145–151.

[Robbins and Monro, 1985] Robbins, H. and Monro, S. (1985). A stochastic approximation method. In Herbert Robbins Selected Papers, pages 102–109. Springer.

[Saxe et al., 2014] Saxe, A. M., McClelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. ICLR.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

References VII

[Smith, 2017] Smith, L. N. (2017). Cyclical learning rates for training neural networks. In Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on, pages 464–472. IEEE.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1):1929–1958.

[Vapnik, 1999] Vapnik, V. N. (1999). An overview of statistical learning theory. IEEE transactions on neural networks, 10(5):988–999.

References VIII

[Yun et al., 2019] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 6023–6032.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[Zhang et al., 2017] Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412.