Optimization on Deep Learning
Course: Deep Learning for Image Analysis
19 - 24 November, Paris

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

MINES ParisTech
PSL Research University
Center for Mathematical Morphology

Contents

Contents

Intuition: Gradient Descent [Cauchy, 1847]

Goal: Minimize $f(x)$

Intuition: Gradient Descent [Cauchy, 1847]

Goal: Minimize $f(x)$ (evaluate as few times as possible the function f)

Intuition: Gradient Descent [Cauchy, 1847]

Goal: Minimize $f(x)$ (evaluate as few times as possible the function f)

The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number $x+\epsilon$ is the power series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{f'(x)}{2}\epsilon^2 + \cdots \tag{1}$$

Intuition: Gradient Descent [Cauchy, 1847]

Goal: Minimize $f(x)$ (evaluate as few times as possible the function f)

The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number $x + \epsilon$ is the power series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{f'(x)}{2}\epsilon^2 + \cdots \tag{1}$$

from that we have: $f(x+\epsilon) \approx f(x) + f'(x)\epsilon$,

Intuition: Gradient Descent [Cauchy, 1847]

Goal: Minimize $f(x)$ (evaluate as few times as possible the function f)

The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number $x+\epsilon$ is the power series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{f'(x)}{2}\epsilon^2 + \cdots \tag{1}$$

from that we have: $f(x+\epsilon) \approx f(x) + f'(x)\epsilon$,

Substituting ϵ with $-\eta f'(x)$ where η is a constant, we have

$$f(x - \eta f'(x)) \approx f(x) - f'(x)\eta f'(x) = f(x) - \eta f'(x)^2 \tag{2}$$

Intuition: Gradient Descent [Cauchy, 1847]

Goal: Minimize $f(x)$ (evaluate as few times as possible the function f)

The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number $x + \epsilon$ is the power series

$$f(x + \epsilon) = f(x) + f'(x)\epsilon + \frac{f'(x)}{2}\epsilon^2 + \cdots \quad (1)$$

from that we have: $f(x + \epsilon) \approx f(x) + f'(x)\epsilon$,

Substituting ϵ with $-\eta f'(x)$ where η is a constant, we have

$$f(x - \eta f'(x)) \approx f(x) - f'(x)\eta f'(x) = f(x) - \eta f'(x)^2 \quad (2)$$

If η is set as a small positive value, we obtain

$$f(x - \eta f'(x)) \leq f(x) \quad (3)$$

Intuition: Gradient Descent [Cauchy, 1847]

Goal: Minimize $f(x)$ (evaluate as few times as possible the function f)

The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number $x+\epsilon$ is the power series

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \frac{f'(x)}{2}\epsilon^2 + \cdots \tag{1}$$

from that we have: $f(x+\epsilon) \approx f(x) + f'(x)\epsilon$,

Substituting ϵ with $-\eta f'(x)$ where η is a constant, we have

$$f(x-\eta f'(x)) \approx f(x) - f'(x)\eta f'(x) = f(x) - \eta f'(x)^2 \tag{2}$$

If η is set as a small positive value, we obtain

$$f(x-\eta f'(x)) \leq f(x) \tag{3}$$

Thus, if we would like to minimize f, we need to update

$$x_{i+1} := x_i - \eta f'(x_i) \tag{4}$$

where $\eta \in \mathbb{R}$ is called *learning rate*.

Gradient Descent [Cauchy, 1847]

Algorithm 1 pseudocode gradient descent

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

Who is f in deep neural networks?

Gradient Descent [Cauchy, 1847]

Algorithm 2 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
    $\mathbf{g}_t = \nabla f_i(\mathbf{X}, \boldsymbol{\theta}_{t-1})$ `Return parameter gradient`
    $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_i$

Who is f in deep neural networks? $\mathbb{E}_{(\mathcal{X},\mathcal{Y})}[L_{\boldsymbol{\theta}}(\mathbf{x}, y)]$, for a given loss function L

SGD [Robbins and Monro, 1951]

Algorithm 3 pseudocode for stochastic gradient descent

```
given initial learning rate η ∈ ℝ and dataset X
initialize time step t = 0, parameter vector θ_{t=0} ∈ ℝ^P,
repeat
    t=t+1
    X_t = SelectBatch(X)  Select a batch from data, whole data, only one, ...
    g_t = ∇f_i(X_t, θ_{t−1})  Return parameter gradient
    θ_t = θ_{t−1} − η g_t
until stopping criterion is met
return  optimized parameters θ_i
```

$\texttt{SelectBatch}(\mathbf{X})$

1. Vanilla gradient descent, a.k.a. *batch gradient descent*, computes the gradient of the cost function w.r.t. to the parameters $\boldsymbol{\theta}$ for the entire training dataset.
2. *Stochastic gradient descent* (SGD) in contrast performs a parameter update for each training example.
3. *Mini-batch gradient descent* performs an update for every mini-batch of N training examples.

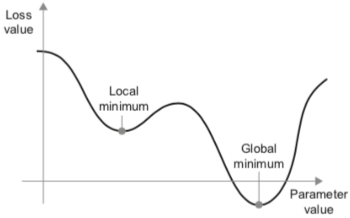
Some difficulties:

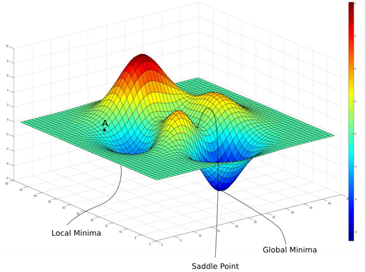
1. Choosing a proper learning rate can be difficult.
2. Same learning rate applies to all parameter updates
3. Highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima.

Difficulties in Deep Network Optimisation

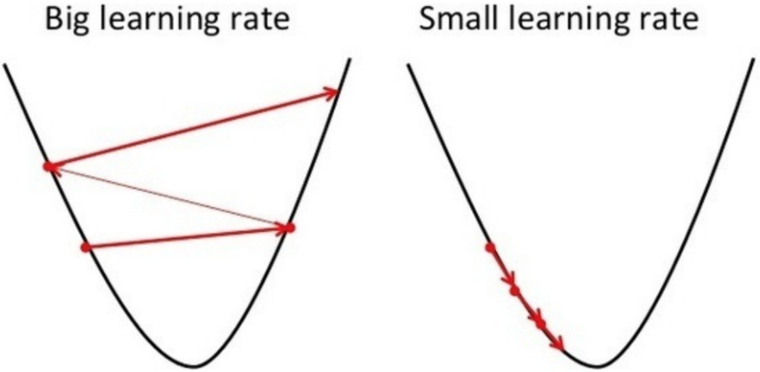
A Local minima / Global minima

B Saddle Point (Plateaus or Flat Regions)





Learning Rate



Learning Rate in Training Process

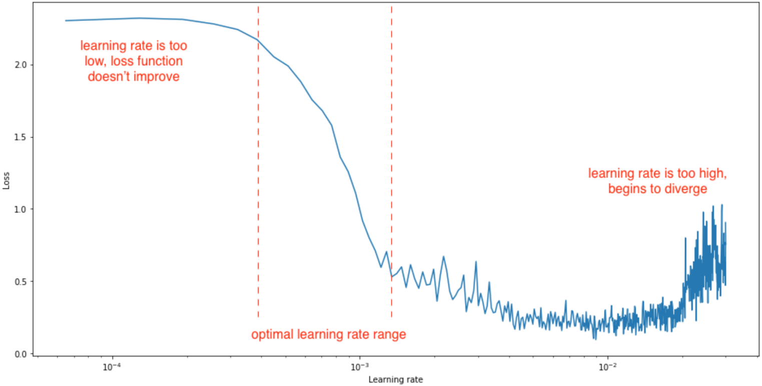


Image by Jeremy Jordan via https://www.jeremyjordan.me/nn-learning-rate/

SGD with Learning Rate Schedule

Algorithm 4 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, ...`
    $\mathbf{g}_t = \nabla f_i(\mathbf{X}_t, \boldsymbol{\theta}_{t-1})$ `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 $\boldsymbol{\theta}_i$

Visualizing the Loss Landscape of Neural Net

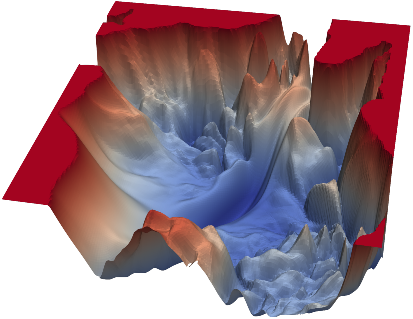


Figure: Loss Function for VGG56 [Li et al., 2017]

SGD with Learning Rate Schedule: New problems.... new ideas!

- Exponential Moving Decay [Dozat and Manning, 2017]
- Cyclical Learning Rates [Smith, 2017]
- Decay with restarts [Loshchilov and Hutter, 2016]

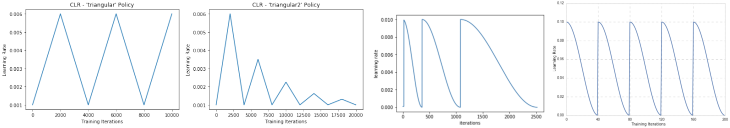


Figure: Schedules: Triangular / Triangular with exponential decay / Exponential Decay with Restarts / Snapshot

- Snapshot Ensembles: Train 1, Get M for free [Huang et al., 2017]
- Don't Decay the Learning Rate, Increase the batch size [Smith et al., 2018]

Optimizer in Keras

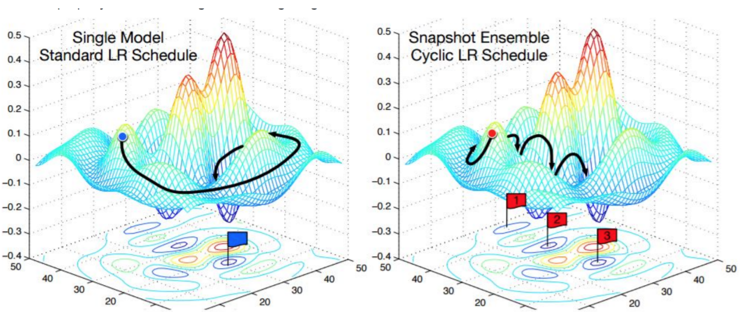


Figure: [Huang et al., 2017]

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

Practical Work 1: Cyclic LR

SGD with momentum [Qian, 1999]

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`

  $\mathbf{g}_t = \nabla f_t(\mathbf{X}_t, \boldsymbol{\theta}_{t-1})$ `Return parameter gradient`

  $\eta_t = \text{SetScheduleMultiplier}(t)$ Can be fixed,decay, warm restarts, ...

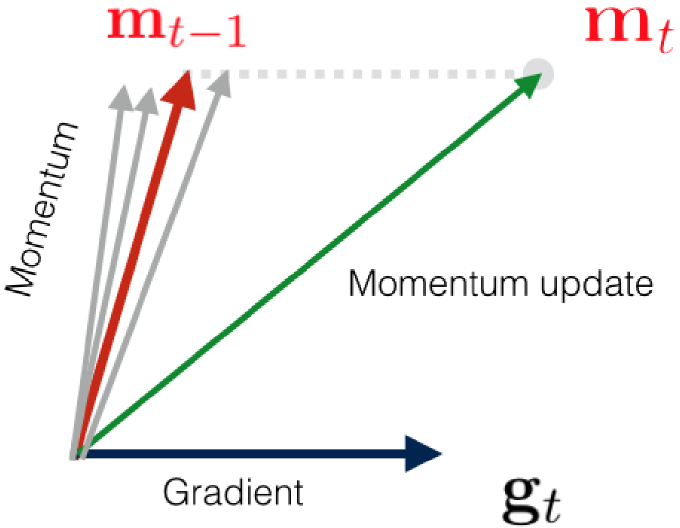
  $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + \eta_t \eta \mathbf{g}_t$

  $\boldsymbol{\theta}_i = \boldsymbol{\theta}_{t-1} - \mathbf{m}_t$

until stopping criterion is met

return optimized parameters $\boldsymbol{\theta}_i$

Momentum



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

SGD with Nesterov's momentum [Nesterov, 1983]

Algorithm 6 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`
    $\mathbf{g}_t = \nabla f_t(\mathbf{X}_t, \boldsymbol{\theta}_{t-1} + \mathbf{m}_{t-1})$ `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

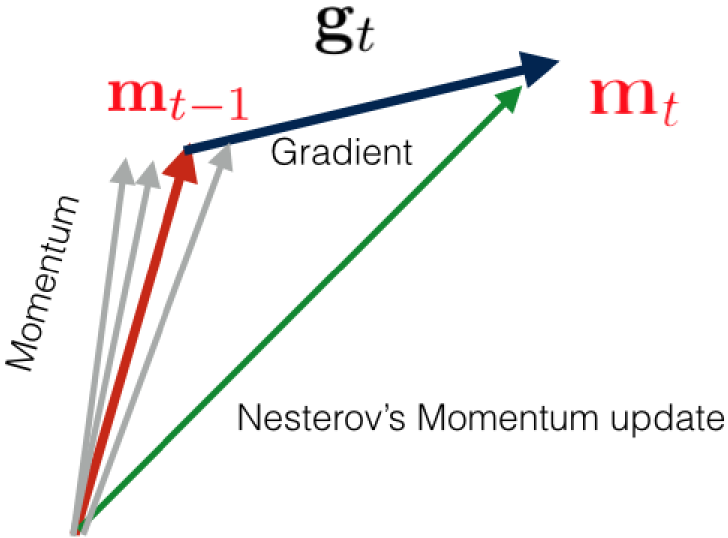


Table: Summary: Some Optimizers I

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.

Figure: Gradient Descent has poor performance on loss functions having contours that are very long "ellipsoids

Motivation: Invariance

Suppose we would like to fit a linear regression on y by using $\mathbf{x} = [x_1, x_2]$, i.e. $\hat{y} = \theta_1 x_1 + \theta_2 x_2$:

Table: default

x_1	x_2	y
134.2	0.01234	1.2
124.4	0.0294	2.2
92.2	0.0194	1.56
⋮	⋮	⋮
98.2	0.0214	1.76

1. This can happen since the inputs have arbitrary units.
2. Which weight, θ_1 or θ_2, should receive a larger gradient descent update?

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.

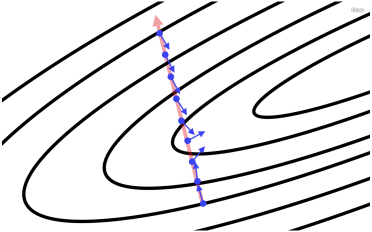


Figure: SGD

Motivation

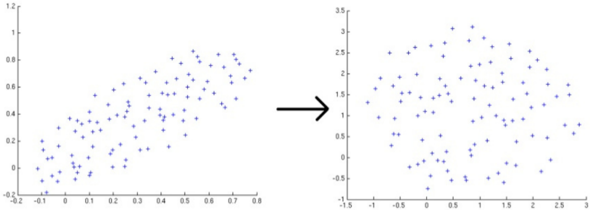


Figure: Which transformation can "decorrelate data? (Statistical Whitening Transform).

Motivation

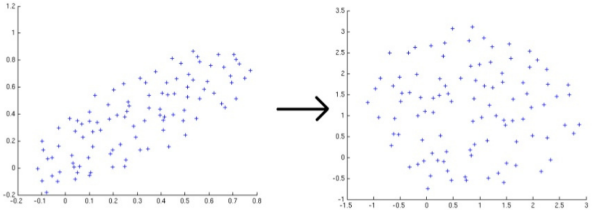


Figure: Which transformation can "decorrelate data? (Statistical Whitening Transform).

Inverse Covariance matrix do the work!

Motivation

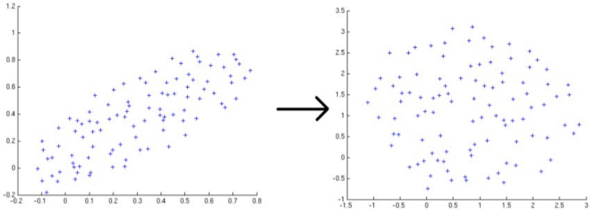


Figure: Which transformation can "decorrelate data? (Statistical Whitening Transform).

Inverse Covariance matrix do the work! Note that one can decompose $\mathbf{\Sigma}^{-1}$ into $\mathbf{V}\mathbf{\Lambda}^{T}\mathbf{V}^{-1}$ (Singular value decomposition), then with $\mathbf{\Omega} = \{1/\sqrt{\lambda_i}\}_{i=1}^{P}$

$$\begin{aligned} \mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} = & \quad \mathbf{x}^T\mathbf{V}\mathbf{\Omega}\mathbf{\Omega}\mathbf{V}^T\mathbf{x} \\ = & \quad (\mathbf{\Omega}\mathbf{V}^T\mathbf{x})^T\mathbf{\Omega}\mathbf{V}^T\mathbf{x} \\ = & \quad ||\mathbf{\Omega}\mathbf{V}^T\mathbf{x}||_2^2 \end{aligned} \tag{5}$$

Euclidean distance in the projected space

From SGD to Natural Gradient Descend

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	$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{\Sigma}_{\boldsymbol{\theta}_t}^{-1} \mathbf{g}(\boldsymbol{\theta}_t)$

Two interpretations for the same method:

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

There are many other methods ... What is the problem with $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}$?

Adagrad

In the Adagrad formulation [Duchi et al., 2011],

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

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 $\mathbf{g}$.
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

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

where $\mathtt{RMS}(G_{ii,t})$ 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 $\mathtt{RMS}(G_{ii,t}) = E(\mathbf{g}^2)_t + \epsilon$.

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:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\texttt{RMS}(\partial\boldsymbol{\theta})_{t-1}}{\sqrt{\texttt{RMS}(G_{ii,t} + \epsilon)}}\mathbf{g}(\boldsymbol{\theta}_t) \tag{8}$$

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

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

Difficulties in Deep Network Optimisation

A Local minima / Global minima
B Saddle Point (Plateaus or Flat Regions)
C Initialization issues
D Vanishing Gradient
E Overfitting

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 fornetworks with non-linearities.
4. Recommended lecture: All you need is a good init [Mishkin and Matas, 2016]

Difficulties in Deep Network Optimisation

D Vanishing Gradient: If feedback signal has to be propagated through a deep stack of layers, the signal may become tenuous or even be lost entirely, rendering the network untrainable. During training, it causes the model's parameter to grow so large so that even a very tiny amount change in the input can cause a great update in later layers' output. The value of layer weights sometimes it overflow and the value becomes NaN.

Fighting against Vanishing Gradient [Pascanu et al., 2013]

1. Initialization of Weights: Don't initialize to values that are too large.
2. 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.)
```

Fighting against Vanishing Gradient

3 Skip connections or Shortcuts (Residual Networks):

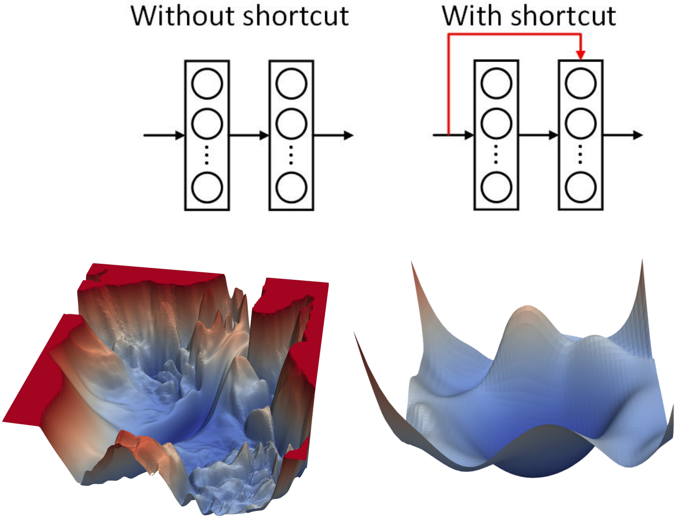


Figure: From [Li et al., 2017]

Fighting against Vanishing Gradient

4 Avoid 'stuck states" induced by activation function:

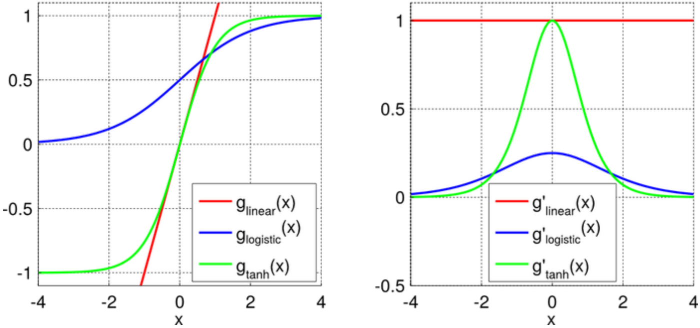


Figure: Left: Three activation function Right: Derivative of activation function.

Fighting against Vanishing Gradient

5 Regularization: L_2 or L_1 norm applies "weight decay" in the cost function of the network. Note that for many activation function, when the activation value is small, that will be almost linear.

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l2(0.01)))
```

6. 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.

6. Batch Normalization

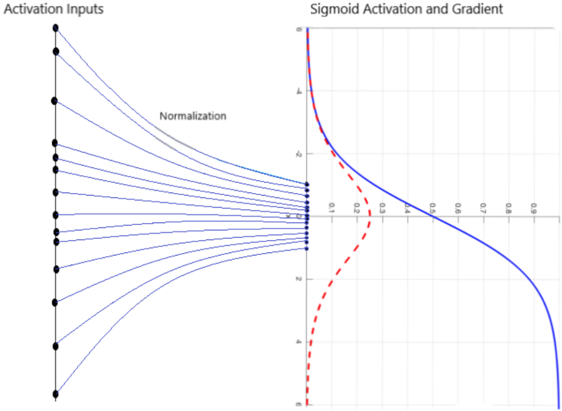


Figure: Image from "Intuit and Implement: Batch Normalization"

Contents

Supervised Learning (Machine Learning)

- **Data**: N observations $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, N$, **i.i.d.**
- **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
- **Minimization of Regularized Empirical Risk**: We would like to find $\boldsymbol{\theta}^*$ the solution of:

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

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

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

Other loss functions, other models

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

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \max\{1 - y\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}), 0\} \tag{9}$$

2. Logistic Regression:

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \log\left(1 + \exp\left(-y\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})\right)\right) \tag{10}$$

3. Mean Squared Regression:

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \frac{1}{2}(y - \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))^2 \tag{11}$$

4. Adaboost

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \exp^{-(y - \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))} \tag{12}$$

5. 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

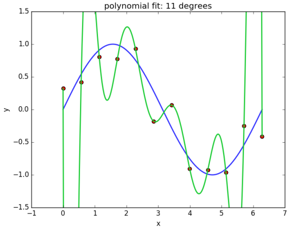


Figure: There are infinity minimizer of the empirical risk!

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

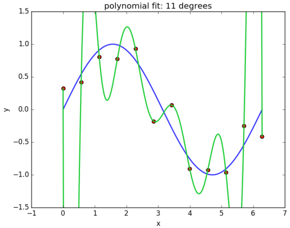


Figure: There are infinity minimizer of the empirical risk!

- Expected Risk : $f(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},y)} L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))$

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

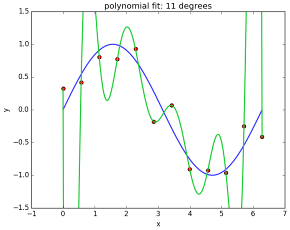


Figure: There are infinity minimizer of the empirical risk!

- 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 $\mathbf{x}$

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

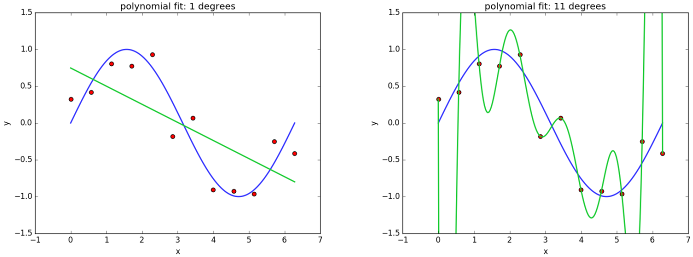
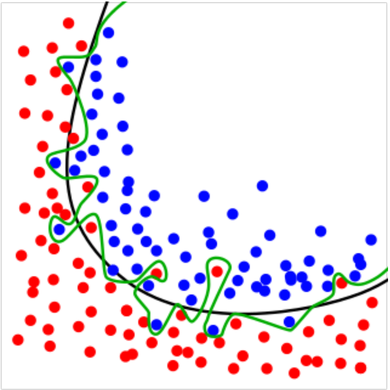


Figure: Underfitting / Overfitting

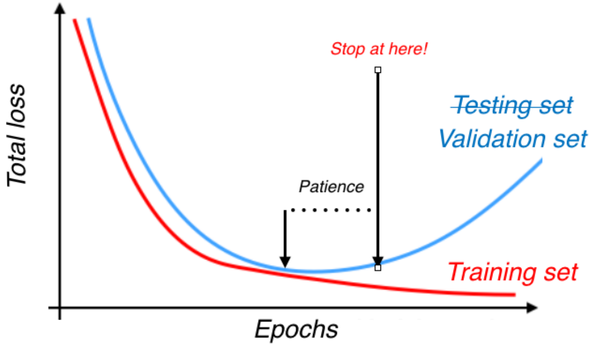
Underfitting : Prediction with less variance but more bias
Overfitting : Prediction with more variance but less bias

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



- Training Set / Testing Set
- Cross-Validation
- $||\boldsymbol{\theta}||_p$ is large

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!

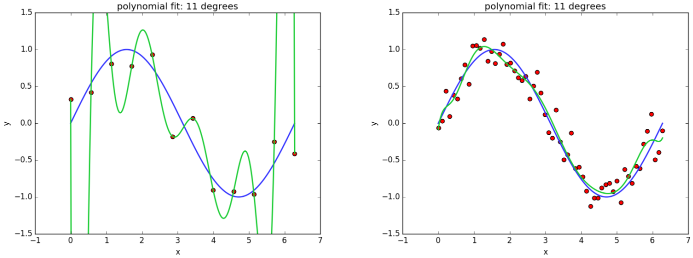


Figure: Same model complexity but more data

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*

$$R_\delta(\texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x})) dP_\delta(\mathbf{x}, y) = \frac{1}{N}\sum_{i=1}^{N} L(\texttt{Model}(\mathbf{x}_i), y_i) \tag{13}$$

Learning the function f by minimizing (13) is known as the Empirical Risk Minimization (ERM) principle [Vapnik, 1999]. If the number of parameters are comparable to N, one trivial way to minimize (13) 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 Noise Layer

```
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 use 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 transformations (translations, rotations, zooming, ...)

```python
training_generator = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

```python
training_generator = train_datagen.flow_from_directory(
        'data/train',  # this is the target directory
        target_size=(224, 224),  # all images will be resized to 150x150
        batch_size=batch_size)
```

```python
model.fit_generator(generator=train_gen.flow(x_train, y_train),
                    steps_per_epoch=steps_per_epoch,
                    epochs=epochs,
                    validation_data=valid_gen.flow(x_validation, y_validation),
                    workers=32)
```

Figure: Data Generator in Keras

Regularization vs Overfitting

1. "Weight Decay": Again?
2. Early Stopping
3. More data! : Data Augmentation
4. More data! Bad data! : Adversarial Examples
5. Summing-up: Dropout

How can we reduce the variance of a model?

- Remember: given N i.i.d. obsevations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ 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 a model?

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

Dropout: [Srivastava et al., 2014]

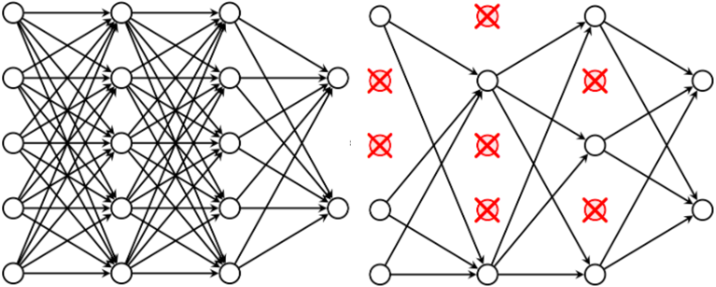


Figure: Left: No Dropout Right: Dropout

- 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

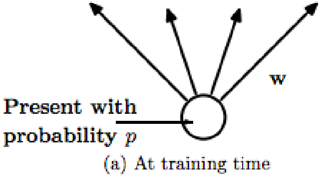


Figure: Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{W}$. Right: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Dropout

```python
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 $p = .5$

Practical Work 2: Dropout on Fashion MNIST

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.

[Dozat and Manning, 2017] Dozat, T. and Manning, C. D. (2017). Deep biaffine attention for neural dependency parsing. *ICLR*.

References II

[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.

[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.

References III

[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*.

[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*.

References IV

[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.

[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*.

References V

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

[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.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

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

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, (3):400–407.

References VI

[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*.

[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.

[Smith et al., 2018] Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2018). Don't decay the learning rate, increase the batch size. *ICML*.

References VII

[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.

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