# Lecture 09 Network Tuning II





# A Few Optimization Algorithms





## Nesterov Momentum

### Add a correction factor to the standard method of momentum

hm 8.2 Stochastic gradient descent (SGD) with momentum
e: Learning rate $\epsilon_i$ momentum parameter $\alpha$
e: Initial parameter $\boldsymbol{\theta}$ , initial velocity $\boldsymbol{v}$ .
$\epsilon$ stopping criterion not met <b>do</b>
nple a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
responding targets $\boldsymbol{y}^{(i)}$ .
mpute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
mpute velocity update: $oldsymbol{v} \leftarrow lpha oldsymbol{v} - \epsilon oldsymbol{g}$
ply update: $oldsymbol{ heta} \leftarrow oldsymbol{ heta} + oldsymbol{v}$
while
e: Initial parameter $\boldsymbol{\theta}$ , initial velocity $\boldsymbol{v}$ . $\boldsymbol{\theta}$ stopping criterion not met <b>do</b> nple a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ we responding targets $\boldsymbol{y}^{(i)}$ . mpute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$ mpute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$ ply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$ while

$$oldsymbol{v} \leftarrow lpha oldsymbol{v} - \epsilon 
abla_{oldsymbol{ heta}} \left( rac{1}{m} \sum_{i=1}^m L(oldsymbol{f}(oldsymbol{x}^{(i)};oldsymbol{ heta}),oldsymbol{y}^{(i)}) 
ight), \quad oldsymbol{ heta} \leftarrow oldsymbol{ heta} + oldsymbol{v}.$$

v: velocity or momentum with unit mass ( $\mathbf{p} = m\mathbf{v} = 1 \cdot \mathbf{v}$ )

• Provides a direction and speed at which the parameters move through parameter space

- $\alpha \in [0,1]$ : a hyperparameter about the momentum
- Determines how quickly the contributions of previous gradients exponentially decay.

#### AncoraSIR.com

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentumRequire: Learning rate  $\epsilon$ , momentum parameter  $\alpha$ .Require: Initial parameter  $\theta$ , initial velocity v.while stopping criterion not met doSample a minibatch of m examples from the training set  $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$  with corresponding labels  $\boldsymbol{y}^{(i)}$ .Apply interim update:  $\theta \leftarrow \theta + \alpha \boldsymbol{v}$ Compute gradient (at interim point):  $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \tilde{\theta}), \boldsymbol{y}^{(i)})$ Compute velocity update:  $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$ Apply update:  $\theta \leftarrow \theta + \boldsymbol{v}$ end while

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[ \frac{1}{m} \sum_{i=1}^{m} L \left( \boldsymbol{f}(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right], \quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

20

10 0

-10

-20

#### Lecture 09 Network Tuning II

## Algorithms with Adaptive Learning Rates

Incremental (or mini-batch-based) methods that adapt the learning rates of model parameters.

Algorithm 8.4 The AdaGrad algorithm

**Require:** Global learning rate  $\epsilon$ 

**Require:** Initial parameter  $\boldsymbol{\theta}$ 

**Require:** Small constant  $\delta$ , perhaps  $10^{-7}$ , for numerical stability

Initialize gradient accumulation variable  $\boldsymbol{r}=\boldsymbol{0}$ 

while stopping criterion not met do

Sample a minibatch of *m* examples from the training set  $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$  with corresponding targets  $\boldsymbol{y}^{(i)}$ . Compute gradient:  $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$ Accumulate squared gradient:  $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$ Compute update:  $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \boldsymbol{g}$ . (Division and square root applied

element-wise)

Apply update:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$ end while

For training deep neural network models—the accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate.

Adapts the learning rates of all model parameters Scaling them inversely

proportional to the square root of the sum of all of their historical squared values

## Algorithms with Adaptive Learning Rates

### Root Mean Squared Propagation

### **Root Mean Squared Propagation**

Algorithm 8.5 The RMSProp algorithm Require: Global learning rate  $\epsilon$ , decay rate  $\rho$ . Require: Initial parameter  $\theta$ Require: Small constant  $\delta$ , usually  $10^{-6}$ , used to stabilize division by small numbers. Initialize accumulation variables r = 0while stopping criterion not met do Sample a minibatch of m examples from the training set  $\{x^{(1)}, \ldots, x^{(m)}\}$  with corresponding targets  $y^{(i)}$ . Compute gradient:  $g \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i} L(f(x^{(i)}; \theta), y^{(i)})$ Accumulate squared gradient:  $r \leftarrow \rho r + (1 - \rho)g \odot g$ Compute parameter update:  $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot g$ .  $(\frac{1}{\sqrt{\delta + r}}$  applied element-wise) Apply update:  $\theta \leftarrow \theta + \Delta \theta$ end while

In AdaGrad: Accumulate squared gradient:  $r \leftarrow r + g \odot g$ 

 $v_t = \beta v_{t-1} + (1 - \beta)\theta_t$ 

### **RMSProp** with Nesterov Momentum

Algorithm 8.6 RMSProp algorithm with Nesterov momentum
<b>Require:</b> Global learning rate $\epsilon$ , decay rate $\rho$ , momentum coefficient $\alpha$ .
<b>Require:</b> Initial parameter $\boldsymbol{\theta}$ , initial velocity $\boldsymbol{v}$ .
Initialize accumulation variable $r = 0$
while stopping criterion not met do
Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
corresponding targets $\boldsymbol{y}^{(i)}$ .
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{r}} \odot \boldsymbol{g}$ . $(\frac{1}{\sqrt{r}} \text{ applied element-wise})$
Apply update: $\boldsymbol{ heta} \leftarrow \boldsymbol{ heta} + \boldsymbol{v}$
end while

It allows for individual adjustment of the learning rate for each parameter of the model



Apply Exponentially Weighted Averages

## Algorithms with Adaptive Learning Rates

### Adaptive Moment (Adam) Estimation

**Momentum** is incorporated directly as an estimate of the first order moment (with exponential weighting) of the gradient.

Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin.

Adam is generally regarded as being *fairly robust* to the choice of hyperparameters,

• Though the learning rate sometimes needs to be changed from the suggested default.

Algorithm 8.7 The Adam algorithm **Require:** Step size  $\epsilon$  (Suggested default: 0.001) **Require:** Exponential decay rates for moment estimates,  $\rho_1$  and  $\rho_2$  in [0,1). (Suggested defaults: 0.9 and 0.999 respectively) **Require:** Small constant  $\delta$  used for numerical stabilization. (Suggested default:  $10^{-8}$ ) **Require:** Initial parameters  $\boldsymbol{\theta}$ Initialize 1st and 2nd moment variables s = 0, r = 0Initialize time step t = 0while stopping criterion not met do Sample a minibatch of m examples from the training set  $\{x^{(1)}, \ldots, x^{(m)}\}$  with corresponding targets  $\boldsymbol{y}^{(i)}$ . Compute gradient:  $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$  $t \leftarrow t + 1$ Update biased first moment estimate:  $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1) \mathbf{g}$ Update biased second moment estimate:  $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$ Correct bias in first moment:  $\hat{s} \leftarrow \frac{s}{1-o_{i}^{t}}$ Correct bias in second moment:  $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-o_1^{t_2}}$ Compute update:  $\Delta \theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$  (operations applied element-wise) Apply update:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$ end while

ME336 Collaborative Robot Learning

### Choosing the Right Optimization Algorithm

Unfortunately, no consensus on this point now

Algorithms with *adaptive learning rates* performed fairly robustly,

• But no single best algorithm has emerged

Currently, the most popular optimization algorithms actively in use

- SGD
- SGD with momentum
- RMSProp
- RMSProp with momentum
- AdaDelta
- Adam

The choice depends largely on the user's familiarity with the algorithm

• For ease of hyperparameter tuning

#### AncoraSIR.com



ME336 Collaborative Robot Learning

#### Lecture 09 Network Tuning II

# Train a Network of Trash Sorting





## Recyclable Waste Sorting Dataset

• Four classes of waste for sorting

Glass						N	let	al					P	ap	er					Pl	ast	tic					
alare 21	alar 22	alars 22	also 24	also 25	alse 26	elar 27	140		estal 142	etal 1/2	©	antal 146	Carlor 148	namer 19	Daner 20	paper 22	paper 24	paper 26	paper 27	paper 28	plastic 180	Sec.	plastic 182	nlastic 183	plastic 184	nlastic 185	plastic 186
glass_28	glass_22	glass_23	glass_24	glass_25	glass_20	glass_27	metal_150	metal 152	metal 153	metal 154	metal 155	metal 156	metal_145	paper 31	paper_20	paper 33	paper_24	paper 36	paper 37	paper_20	plastic_187	plastic 188	plastic_182	plastic 190	plastic_104	plastic 193	plastic_194
glass_38	glass_39	glass_40	glass_41	glass_42	glass_43	glass_44	metal_158	metal 159	metal_161	metal_162	metal_163	metal_164	metal_165	paper_39	paper_40	paper_41	paper_43	paper_45	paper_47	paper_48	plastic_195	plastic_196	plastic_198	plastic_200	plastic_201	plastic_202	plastic_203
glass_46	glass_47	glass_48	glass_49	glass_50	glass_51	glass_53	metal_166	(Contraction) metal_168	metal_169	metal_170	metal_171	metal_174	metal_175	paper_50	paper_52	paper_53	paper_54	paper_55	paper_56	paper_57	plastic_204	Plastic_205	plastic_206	plastic_207	plastic_208	plastic_209	plastic_210
glass_54	glass_55	glass_56	glass_58	glass_59	glass_60	glass_61	metal_176	metal_178	metal_179	() metal_181	metal_182	metal_184	metal_186	paper_58	paper_59	paper_60	paper_61	paper_62	paper_64	paper_65	plastic_211	plastic_212	plastic_213	plastic_214	plastic_215	plastic_216	plastic_217
glass_62	glass_63	glass_64	glass_65	glass_66	glass_67	glass_68	metal_187	metal_188	metal_189	metal_191	metal_192	metal_193	metal_194	paper_66	paper_67	paper_68	paper_69	paper_70	paper_71	paper_72	plastic_219	plastic_220	plastic_222	plastic_223	plastic_224	plastic_229	plastic_230
glass_69	glass_70	glass_72	glass_73	glass_74	glass_75	glass_76	metal_196	metal_197	metal_198	metal_199	metal_200	metal_201	metal_202	paper_73	paper_74	paper_76	paper_77	paper_78	paper_79	paper_80	plastic_231	plastic_232	plastic_233	plastic_234	plastic_236	plastic_237	plastic_238
glass_77	glass_78	glass_79	glass_80	glass_81	glass_83	glass_85	metal_204	metal_205	metal_206	metal_207	metal_208	metal_209	metal_210	paper_81	paper_82	paper_83	paper_84	paper_87	paper_89	paper_90	plastic_239	plastic_240	plastic_241	plastic_242	plastic_243	plastic_245	plastic_246
glass_86	glass_87	glass_90	glass_92	glass_93	glass_94	glass_95	metal_211	metal_212	metal_215	metal_216	metal_218	metal_219	metal_220	paper_91	paper_92	paper_93	paper_94	paper_95	paper_96	paper_98	plastic_247	plastic_249	plastic_250	plastic_251	plastic_253	plastic_254	plastic_255

#### AncoraSIR.com

**SUST**ech

Southern University of Science and Technology

## Download Dataset

- Download the dataset from website:
  - <u>https://pan.baidu.com/s/1IQuPEPhxA6N97AZrJU4Hmw</u>
  - with code: 69cs
- Extract the files to have the following folder structure:
  - BionicDL
    - \_train\_: 80% of the total data
      - glass, metal, paper, plastic
    - \_test\_: 20% of the total data
      - glass, metal, paper, plastic

	Train (80%)	Test (20%)	Total
Glass	204	50	254
Metal	396	98	494
Paper	1291	322	1613
Plastic	1202	300	1502
Total	3093	770	3863



### Data Generator & Augmentation

With TensorFlow and Keras

• The data generator class in *tensorfow.keras* can conveniently generate data flow and add data augmentation for GPU training.

from tensorflow.keras.preprocessing.image import ImageDataGenerator
BATCH\_SIZE = 32
# Data generators, data augmentation
# You can uncomment the data augmentation parameters
train = ImageDataGenerator(
 samplewise\_center=True,
# rotation\_range = 90.0,
# width\_shift\_range=0.2,
# height\_shift\_range=0.2,
# horizontal\_flip = True,
vertical\_flip = False)

valid = ImageDataGenerator(samplewise\_center=True)

```
# Target directories
trainGenerator = train.flow_from_directory(
    '../data/BionicDL/_train_',
    target_size = (224, 224),
    batch_size=BATCH_SIZE,
    class_mode = "categorical",
    color_mode = "rgb", shuffle=True, seed=42)
validationGenerator = valid.flow_from_directory(
    '../data/BionicDL/_test_',
    target_size = (224, 224),
    batch_size=1,
    class_mode = "categorical",
    color_mode = "rgb", shuffle=False)
```

Found 3093 images belonging to 4 classes. Found 770 images belonging to 4 classes.

#### AncoraSIR.com

SUSTech

them University

## AlexNet Model

- Architecture: five convolutional layers and three fully-connected layers.
- Contributions:
  - ReLU instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
  - Dropout instead of regularization to deal with overfitting
  - Overlay pooling to reduce the size of network



### Build the Model in TensorFlow

#### model = Sequential(name="AlexNet")

# 1st Convolutional Layer

model.add(Conv2D(filters=96, input\_shape=(224,224,3), kernel\_size=(11,11),strides=(4,4), padding='valid', name='conv\_1')) model.add(Activation('relu')) # Pooling model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='valid')) # Batch Normalisation before passing it to the next Layer model.add(BatchNormalization())

# 2nd Convolutional Layer
model.add(Conv2D(filters=256, kernel\_size=(11,11), strides=(1,1), padding='valid', name='conv\_2'))
model.add(Activation('relu'))
# Pooling
model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='valid'))
# Batch Normalisation
model.add(BatchNormalization())

# 3rd Convolutional Layer
model.add(Conv2D(filters=384, kernel\_size=(3,3), strides=(1,1), padding='valid', name='conv\_3'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())

# 4th Convolutional Layer
model.add(Conv2D(filters=384, kernel\_size=(3,3), strides=(1,1), padding='valid', name='conv\_4'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())

# 5th Convolutional Layer
model.add(Conv2D(filters=256, kernel\_size=(3,3), strides=(1,1), padding='valid', name='conv\_5'))
model.add(Activation('relu'))
# Pooling
model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='valid'))
# Batch Normalisation
model.add(BatchNormalization())

# Passing it to a dense Layer
model.add(Flatten())
# 1st Dense Layer
model.add(Dense(4096, input\_shape=(224\*224\*3,), name='dense\_1'))
model.add(Activation('relu'))
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())

# 2nd Dense Layer
model.add(Dense(4096, name='dense\_2'))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())

# Output Layer
model.add(Dense(4, name='dense\_3\_new'))
model.add(Activation('softmax'))



## Model Summary

### 24 million parameters with Default random weights initialization

#### Model: "AlexNet"

Layer (type)	Output	Shape	Param #
conv_1 (Conv2D)	(None,	54, 54, 96)	34944
activation (Activation)	(None,	54, 54, 96)	0
<pre>max_pooling2d (MaxPooling2D)</pre>	(None,	27, 27, 96)	0
batch_normalization (BatchNo	(None,	27, 27, 96)	384
conv_2 (Conv2D)	(None,	17, 17, 256)	2973952
activation_1 (Activation)	(None,	17, 17, 256)	0
<pre>max_pooling2d_1 (MaxPooling2</pre>	(None,	8, 8, 256)	0
batch_normalization_1 (Batch	(None,	8, 8, 256)	1024
conv_3 (Conv2D)	(None,	6, 6, 384)	885120
activation_2 (Activation)	(None,	6, 6, 384)	0
batch_normalization_2 (Batch	(None,	6, 6, 384)	1536
conv_4 (Conv2D)	(None,	4, 4, 384)	1327488
activation_3 (Activation)	(None,	4, 4, 384)	0
batch_normalization_3 (Batch	(None,	4, 4, 384)	1536

conv_4 (Conv2D)	(None, 4, 4, 384)	1327488
activation_3 (Activation)	(None, 4, 4, 384)	0
batch_normalization_3 (Batch	(None, 4, 4, 384)	1536
conv_5 (Conv2D)	(None, 2, 2, 256)	884992
activation_4 (Activation)	(None, 2, 2, 256)	0
<pre>max_pooling2d_2 (MaxPooling2</pre>	(None, 1, 1, 256)	0
batch_normalization_4 (Batch	(None, 1, 1, 256)	1024
flatten (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 4096)	1052672
activation_5 (Activation)	(None, 4096)	0
dropout (Dropout)	(None, 4096)	0
batch_normalization_5 (Batch	(None, 4096)	16384
dense_2 (Dense)	(None, 4096)	16781312
activation_6 (Activation)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0
batch_normalization_6 (Batch	(None, 4096)	16384

dense_3_new (Dense)	(None,	4)	16388
activation_7 (Activation)	(None,	4)	0
Total params: 23,995,140 Trainable params: 23,976,004 Non-trainable params: 19,136			



## Training the Model

- Training options
  - Epochs:
  - Optimizer and learning rate: SGD:
  - Callbacks: functions to be executed during the training process:
  - Starting training:

# compute steps per epoch EPOCHS = 30 steps\_per\_epoch = 3093//BATCH\_SIZE # 3093 items in the training dataset

# you can test different optimizer settings here
# opt = tf.keras.optimizers.SGD(lr=0.01)
opt = tf.keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

```
# utility callback that displays training curves
plot_training = PlotTraining(sample_rate=1, zoom=1)
checkpoint = tf.keras.callbacks.ModelCheckpoint(
    'BionicDL-bs32-weights.{epoch:02d}-{val_acc:.3f}-DenseNet169.hdf5',
    verbose=1, save_best_only=True,
    save_weights_only=False,
    mode='max', period=1)
```

# Compile

model.compile(loss='categorical\_crossentropy', optimizer=opt, metrics=['accuracy'])



## Training the Model

- Training VS Validation:
  - Validation accuracy is saturated at 85% while training accuracy is approaching 100%.





SUSTech

### Results

### Check the prediction accuracy of each category

• Confusion matrix





of the of the office of the of





Room 606 7 Innovation Park 南科创园7栋606室

# Thank you~

songcy@sustech.edu.cn



