Friedrich-Alexander-Universität Erlangen-Nürnberg

Chair of Multimedia Communications and Signal Processing

Prof. Dr.-Ing. André Kaup

Master Thesis

Automatic Determination of Model Parameters for the Estimation of the Processing Energy of Video Decoders

Junyue Wu

14.10.2015

Supervisor: Dipl.-Ing. Christian Herglotz

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt und von dieser als Teil einer Prüfungsleistung angenommen. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Ort, Datum	Unterschrift

Lehrstuhl für Multimediakommunikation
und Signalverarbeitung
Prof. Dr.-Ing. A. Kaup

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Master Thesis

for

Ms. Junyue Wu

Automatic Determination of Model Parameters for the Estimation of the Processing Energy of Video Decoders

(Automatisierte Bestimmung von Modellparametern zur Schätzung der
Prozessenergie von Videodecodierern)

In recent years, the use of portable devices like smartphones or tablet PCs has increased rapidly. To preserve battery power and thus extend operating times, energy saving algorithms are highly desirable. As video decoding requires a significant part of the complete power consumption of a mobile device, it shall be further investigated in this thesis. The goal is to estimate the energy consumption of a given video decoding solution as accurately as possible.

To this end, Ms. Junyue Wu shall investigate an existing energy consumption model that is capable of estimating this energy. It is her task to develop a suitable training method to derive the parameter values of the model. Therefore, a high number of measurement data will be provided that can be used for training and testing. For the training part, methods like regression, least-squares fits, or cross-validation can be used. As a result, the method that performs best shall be implemented into an automated Matlab-Script that calculates trained parameter sets and that returns the overall estimation accuracy. The script shall be implemented in such a way that external users may easily apply this model to their own needs. The thesis shall describe the methodology and the different tested approaches and explain the obtained results in a clear and understandable manner.

Start:	15.04.2015
End:	14.10.2015

(Prof. Dr.-Ing. A. Kaup)

Contents

Kurzfassung

In den letzten Jahren haben sich die tragbare Geräte wie Smartphones oder Tablet-PCs schnell entwickelt, für die ein niedriger Energieverbrauch ist eine wichtige Eigenschaft. Um einen energieeffizienten Algorithmus für Decodierung zu finden, ist die Decodierenergie erforderlich zu messen. In einem bestehenden Energieverbrauchsmodell werden die spezifische Energien mit deren entsprechenden Anzahlen des Eintrittes multipliziert, in denen die spezifische Energien im Voraus bekannt sein. Das Ziel dieser Arbeit ist es, eine geeignete Trainingsmethode zur Bestimmung der Parameterwerte für dieses Modell zu erstellen, mit deren der Energieverbrauch von Videodecodieren geschätzt werden kann.

Abstract

In recent years, the portable devices like smartphones and tablet PCs have developed rapidly, for which a low energy consumption is an important property. In order to find an energy-efficient decoding algorithm, the measurement of the decoding energy is required. An energy consumption model using only the specific energies and their corresponding occurrence numbers has been developed, in which the specific energies should be known in advance. The goal of this thesis is to develop a suitable training method to derive the specific energies, and thus the energy consumption of a given video sequence can be estimated.

Chapter 1

Introduction

Since two decades ago, the demand for video services such as streaming, recording and coding increased so rapidly that video services accounted for a large part of the global internet traffic. How to reduce bit-rates and keep the visual quality at the same time was the issue that was mainly dealt with in the early days, while the decoding energy has obtained growing attention in recent years [8]. The development of portable devices like smartphones and tablet PCs contributes to this change, as preserving battery power and thus extending operating times are essential demands for portable devices. A significant part of the complete energy consumption is used for decoding, and thus the search for an energy-efficient video decoding technique has become a desirable task.

In order to develop an energy-efficient algorithm for video decoding, the measurement of the decoding energy is necessary, which is normally complex and costly. To solve this problem, an energy consumption model has been developed, in which a video sequence is considered as a combination of different routines [7] [9]. The energy consumption of each routine is a fixed value, and thus the complete energy consumption of the video sequence can be derived by multiplying the specific energies of different routines with their corresponding occurrence numbers in this video sequence. In this way, this energy consumption model can reduce the complexity of measuring the decoding energy. However, a problem for this energy model is that the specific energies of different routines should be known in advance. The goal of this thesis is to develop a suitable training method to derive the specific energies (parameters). In this work, a high number of measurement data is given for training. The number of parameters and their corresponding occurrence numbers in all the measurement data are also known in advance.

The HEVC standard that was finalized several years ago has become the successor to H.264/MPEG-4 AVC. This thesis deals with the HEVC video sequences. At first, the HEVC standard and the existing energy consumption model will be introduced in Chapter 2. Then several training methods and a validating method will be explained in Chapter 3. Finally, the evaluation results of the training methods will be presented in Chapter 4, and thus a relatively suitable method can be selected to derive the specific energies of the energy consumption model.

Chapter 2

HEVC and Energy Consumption

HEVC, which is short for High Efficiency Video Coding, is a new video compression standard. The goal of HEVC is to improve compression performance relative to existing standards [1]. In this section, the HEVC standard and a model for its energy consumption will be introduced.

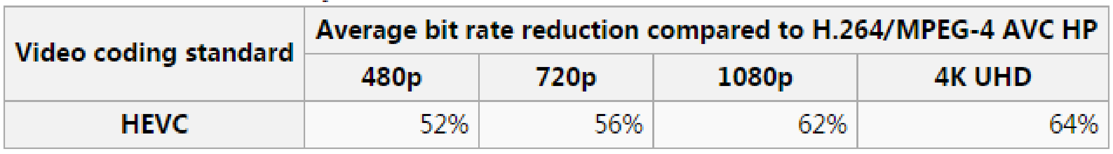
2.1 HEVC

2.1.1 Development History of HEVC

HEVC was developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) together. These two organizations are known as the Joint Collaborative Team on Video Coding (JCT-VC) [2]. The first version of HEVC was finalized in 2013 and the second version, which includes format range extensions (RExt), scalable coding extensions (SHVC), and multi-view extensions (MV-HEVC), was completed in 2014 and published in early 2015 [3].

HEVC is considered to be the successor to H.264/MPEG-4 Advanced Video Coding (AVC), which has led to an evolution in the area of digital video processing in the years between 1999 and 2003. H.264/MPEG-4 AVC is applied in many kinds of areas, such as broadcast of high definition (HD) TV signals, video content acquisition and editing systems, Internet and mobile network video and Blu-ray Discs [1].

However, due to the development of mobile devices and tablet PCs, higher quality and resolutions of video coding are required. HEVC, as the improvement of H.264/MPEG-4 AVC, is designed for all the above mentioned applications and focuses especially on higher video resolutions and parallel processing architectures [1]. Compared to H.264/MPEG-4 AVC at the same level of video quality, HEVC is able to double the data compression ratio. Fig. 2.1 [4] shows the average bit rate reduction with HEVC.



Video coding standard	Average bit rate reduction compared to H.264/MPEG-4 AVC HP			
	480p	720p	1080p	4K UHD
HEVC	52%	56%	62%	64%

Figure 2.1 Video performance comparison [4]

In HEVC only the bitstream structure and syntax is standardized, which is similar to the past video coding standards. Although it allows free implementations of the standard in this way, the end-to-end reproduction quality is not guaranteed, as crude encoding techniques will be allowed. [1]

In order to explain this new standard more clearly, not only a text specification document but also reference software source code is developed. During the design of HEVC, the reference software has been used for research and it can be used for general research as well [1].

2.1.2 HEVC Design

The standard HEVC uses the same basic hybrid coding architecture as the prior video coding standards, such as inter or intra picture prediction and 2-D transform coding. Fig. 2.2 [1] shows the block diagram of a hybrid video encoder and decoder.

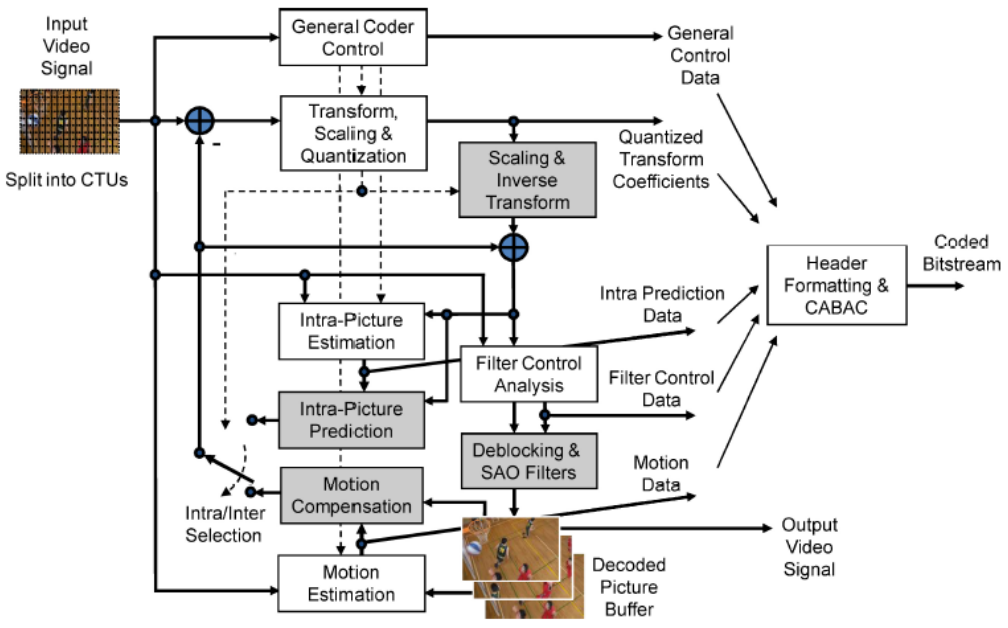


Figure 2.2 Typical HEVC video encoder and decoder shaded in gray [1]

The process of producing an HEVC compliant bitstream is represented in Fig. 2.2. Every frame of the input video signal is divided into blocks. With intra picture prediction, which only uses the information spatially within the same frame and does not depend on other frames, the first frame of the input sequence in each iteration is coded, that is to say, the first frame is refreshed after each iteration. Inter picture prediction, which is considered to be a temporal prediction, is then applied for all the remaining frames of the input sequence. During the inter picture prediction a reference frame and motion vector are determined, which are transmitted together with the difference between the original frame and its prediction, which is the residual error. This residual error is transformed by a linear spatial transform, of which the transform coefficients are scaled, quantized and entropy coded. The decoder and encoder generate identical prediction signals, as a result, in the decoding process the quantized transform coefficients can be obtained by inverse scaling, with which the residual error is reconstructed. In order to get the final frame representation, which is used to predict the subsequent frame, the residual error carrying the difference information between the original frame and its prediction is added to the prediction [1].

Although the HEVC standard retains the basic hybrid coding architecture of H.264/MPEG-4 AVC, including inter picture prediction and intra picture prediction, several significant differences were introduced as follows.

1) Coding tree unit (CTU): Compared to the prior encoding process, HEVC uses coding tree units and coding tree blocks (CTB) structure instead of a macroblock, which contains a 16×16 block of luma samples and two corresponding 8×8 blocks of chroma samples [1]. However, the size of CTU is selected by the encoder and therefore larger resolutions than traditional macroblock can be obtained, for example 64×64, which is the largest size of CTU. Fig. 2.3 shows all prediction block partitioning modes that specify how a coding block is divided into prediction blocks. All the modes in Fig. 2.3 can be applied in inter picture coded coding units, while only the first two modes in intra picture coded coding units. [1][5]

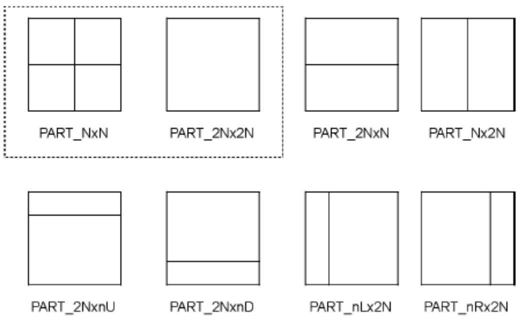


Figure 2.3 All prediction block partitioning modes [5]

2) Transforms and quantization: H.264/MPEG-4 AVC uses 4-point and 8-point transforms which are not suited to larger transform sizes. As a result, the HEVC standard applies a different method that simply defines transforms as straightforward fixed-point matrix multiplications. The matrix multiplications are shown in the following formulae (2.1) and (2.2) [5], where Y and R are respectively the vertical and horizontal component of the inverse transform. The values of Y use 16 bit that is guaranteed by a scaling and saturating function s() and the transform matrix T uses signed 8-bit numbers. C represents 16-bit signed coefficients and is multiplied with T. The transform order of HEVC, namely column-row order, is also different from H.264/MPEG-4 AVC. This approach is preferred in software, as rows are transformed, which is easier realized in registers and therefore requires fewer registers and software operations [5].

$$\mathrm{Y} = \mathrm{s}(C^T \cdot T), \tag{2.1}$$

$$\mathrm{R} = Y^T \cdot T. \tag{2.2}$$

3) Entropy coding: In HEVC, CABAC is defined as the single entropy coding method, while H.264/AVC features CAVLC and CABAC [6] entropy coders. Binarization of syntax elements, context modeling, and binary arithmetic coding are three stages of CABAC. Compared to H.264/AVC, there are several differences in context modeling and binarization. First of all, in order to reduce the amount of memory required during the entropy decoding, much effort has been made to reduce the number of contexts. The version 8.0 of HEVC has only 154 contexts, while H.264/AVC has 299 contexts. Secondly, spatial neighborhood relationships, that means using the values around the current block to derive the context, are used in H.264/AVC, while this method is avoided in HEVC to reduce the line buffers [5].

4) Sample-adaptive offset filter: In H.264/AVC only a deblocking filter is applied, while an additional sample-adaptive offset (SAO) filter is applied in HEVC. The SAO filter adds

an additional offset value to each sample. Two modes determined by indexing a small lookup table are used. One of the modes is that the samples in the same value range use the same offset. The other mode is that the offset is calculated based on differences between the current and two neighboring samples. Because of this additional offset value, the complexity is increased [5].

For some important modules, such as transforms, intra picture prediction, and motion compensation, the complexity of HEVC is higher than H.264/AVC. However, for some other modules such as entropy coding and deblocking, the complexity is reduced. As the main complexity is considered to be in motion compensation, entropy coding, and in-loop filtering, the HEVC decoder is not much more complex than the H.264/AVC decoder. For encoding, an HEVC encoder is much more complex than an H.264/AVC encoder. However, the advantages of higher complexity are obvious, for example, the higher complexity of HEVC can guarantee a great improvement in rate-distortion performance [5].

2.2 Model of the Energy Consumption for the HEVC Decoder

The HEVC standard is developed for the demand of high-resolution, high-quality, and 3D content. However, minimizing the system requirements and energy consumption is always a main problem. As the decoding process consumes a large amount of energy, this thesis focuses on the decoding energy consumption. A prior method is to use a decoder built on a hardware chip, which is complex but efficient. Another method is to choose a coding method with which the decoding energy is minimized. With this method the decoding time can be used to estimate the decoding energy, which is an obvious disadvantage to this method, as the energy consumption cannot be estimated precisely only by decoding time [7].

In order to estimate the decoding energy without decoding time, a model presented in [7] is applied. At first, this model was used for the intra frame coding. Then this idea is expanded to deal with inter frame coded P- and B-frames with disabled in-loop filters. [7] This model multiplies the occurrence number of routines, with experimentally derived specific energies. Such a model can be described as

$$\hat{E}_{dec} = \sum\nolimits_{\forall\, routines} e_r \cdot n_r \ . \tag{2.3}$$

This model is a linear model. In this formula, $\hat{E}_{dec}$ represents the estimated decoding energy, and e_r, n_r represent respectively routine specific energy and number of routine executions. For a given decoding system, with this model the energy consumption of different hardware and software can be observed and therefore analyzed. One obvious advantage of this energy model is that the estimation of the energy consumption can be derived without details about the implementation, such as processor, peripheral configuration, and source code, that is to say, the decoder can be seen as a black box. As a result, a set of videos for training is required to determine an accurate energy estimation model [7].

In the following three subsections, the test setup, the prior intra frame decoding model, and the later inter frame decoding model will be introduced respectively.

2.2.1 Test Setup

The test setup presented in Fig. 2.4 is used to measure the energy during the decoding process. The power meter (LMG95) internally calculates the energy in a defined time interval. The DUT, that is short for device under test, is a Pandaboard. The Pandaboard, which is similar with the architecture of smartphone features an OMAP4430 SoC with an ARMv7 dual core processor.

The energy consumption includes not only the decoding energy, but also the energy from other consumers, such as LEDs or background processes [8]. To get the pure decoding energy, two measurements are performed. During the first measurement the energy is measured without user processes, which is recorded as E_{idle}. Then the energy when the decoding process is active is measured again, which is recorded as E_{all}. Finally the pure decoding energy can be calculated with

$$E_{dec} = E_{all} - E_{idle}. \tag{2.4}$$

In Figure 2.5, the green line represents the idle mode, and the blue line represents the sum of the pure decoding energy and other background energy. The area between the green and blue line represents the pure decoding energy. In this case, the decoder starts at about 0.5s and ends at about 1.5s. [8]

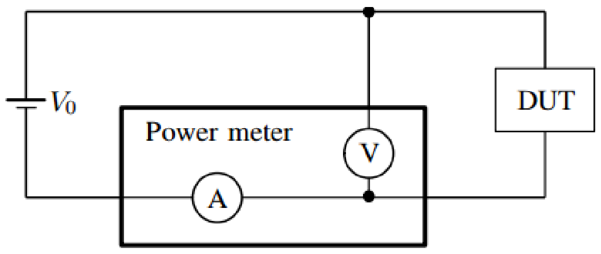


Figure 2.4 Test setup [8]

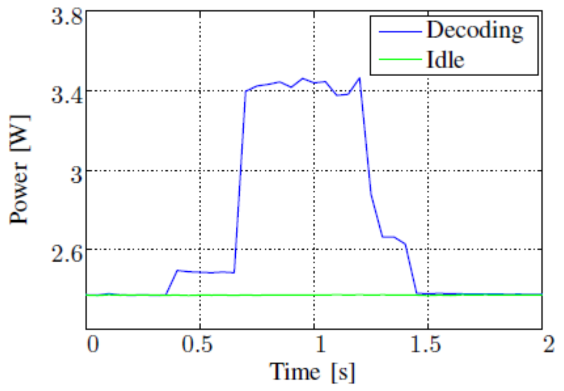


Figure 2.5 Energy consumption of the decoder [8]

However, the measured energy is not constant, as some factors like temperature have effects on the measurement. To minimize the errors caused in this situation, multiple measurements are performed until the real mean value lies near to the measured mean value, for example, inside a small region around the measured mean value. When the energy is considered as a random variable and M measurements are performed for each bit stream, the mean energy can be calculated

$$\hat{E}_{dec,n} = \frac{1}{M}\sum_{m=1}^{M} E_{dec,n,m}, \tag{2.5}$$

where $E_{dec,n,m}$ is the measured energy of the m^{th} measurement, and n is the index of the bit stream. When the real mean value lies within probability α, the size of the confidence interval is

$$\hat{E}_{dec,n} - \frac{\sigma}{\sqrt{M}} \cdot t_\alpha(M-1) < \mu < \hat{E}_{dec,n} + \frac{\sigma}{\sqrt{M}} \cdot t_\alpha(M-1), \tag{2.6}$$

where σ is the standard deviation of the measurement and t_α is the critical t-value of the student t distribution. The measurement is repeated until the following criterion

$$\frac{\sigma}{\sqrt{M}} \cdot t_\alpha(M-1) < 0.01 \cdot \hat{E}_{dec,n} \tag{2.7}$$

is fulfilled. In this way, the real mean value will be located within probability $\alpha = 99\%$ inside the small region around the measured mean value [8].

2.2.2 Intra Frame Decoding Energy Model

As mentioned before, this energy model is at first applied for intra frame decoding. For the HEVC, each frame is decomposed into CTUs, which are later divided into CUs. A CU includes a prediction unit, that carries the information of the intra prediction mode like planar, DC, or angular, and a transform unit, that provides transform coefficients for compensation of the prediction error. This mode is called intra frame, as it depends on the spatial neighborhood [9].

Two models of estimating the energy consumption for intra frame decoding were introduced in [9]. One is an accurate model and the other is a simple model. The accurate model is depicted in the following formula [9]:

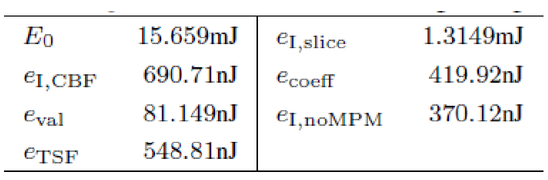
$$\hat{E}_{I,a} = E_0 + e_{I,slice} \cdot n_{I,slice} + \sum\nolimits_{\forall size}\left(\sum\nolimits_{\forall mode} e_{I,mode,size} \cdot n_{I,mode,size}\right) + e_{I,CBF} \cdot n_{I,CBF} + e_{coeff} \cdot n_{coeff} + e_{val} \cdot \sum\nolimits_{\forall c \neq 0} \log_2|c| + e_{I,noMPM} \cdot n_{I,noMPM} - e_{TSF} \cdot n_{TSF}, \quad (2.8)$$

and the simple model [9] is

$$\hat{E}_{I,s} = E_0 + e_{I,slice} \cdot n_{I,slice} + \sum\nolimits_{\forall size} e_{I,avg,size} \cdot n_{I,size} + e_{I,CBF} \cdot n_{I,CBF} + e_{coeff} \cdot n_{coeff}. \quad (2.9)$$

Tables 1 and 2 [7] define the values of the specific energies E_0 and e, which are derived by the above training method. The values and meaning of the energy constants are clearly explained in [9]. The variable n represents quantities and $n_{I,slice}$ the number of I-slices. $n_{I,mode,size}$ is the number of blocks of a specific size which are predicted by a specific intra mode, and $n_{I,size}$ is the complete number of intra frame predicted blocks of a specific size. $n_{I,CBF}$ is the number of CBF that is short for coded block flags. The number of non-zero residual coefficients is n_{coeff}, and their values c. $n_{I,noMPM}$ is the number of intra frame coded CUs which are not most probable intra modes, and n_{TSF} is the number of transform skips. All the variables above describe the properties of a bit stream, and E_0 is regarded as a constant offset energy. n_{coeff}, c, and n_{TSF} are also taken into account in the next subsection in the inter frame decoding model. [7]

Table 1 Specific intra frame decoding energies [7]



E_0	15.659mJ	$e_{\mathrm{I,slice}}$	1.3149mJ
$e_{\mathrm{I,CBF}}$	690.71nJ	e_{coeff}	419.92nJ
e_{val}	81.149nJ	$e_{\mathrm{I,noMPM}}$	370.12nJ
e_{TSF}	548.81nJ		

Table 2 Specific decoding energies per intra frame coded block [μJ]. The horizontal, vertical, and diagonal modes are summarized to hvd; angular comprises the remaining modes [7]

$e_{\mathrm{I,mode,size}}$	32×32	16×16	8×8	4×4
planar	270.54	66.297	23.469	8.0700
DC	266.59	65.429	23.834	8.2082
hvd	267.92	68.456	24.191	8.0954
angular	272.57	69.646	24.672	8.1389
$e_{\mathrm{I,avg,size}}$	271.68	69.260	24.545	8.1327

2.2.3 Inter Frame Decoding Energy Model

The model introduced above is now applied for P- and B-frame coded videos. Similar to the model of intra frame decoding, the model of inter frame decoding has also two types. One is an accurate model and the other is a simple model. The accurate model of inter frame decoding has 73 parameters, while the simple model has only 20 parameters, that provides a more convenient representation. [7]

The model of inter frame decoding energy is explained in detail in [7]. The accurate model is described by the following formula [7]:

$$\hat{E}_{PB,a} = e_{PB,slice} \cdot n_{PB,slice} + \hat{E}_{pred,a} + \hat{E}_{frac,a} + \hat{E}_{trans,a} + e_{bi} \cdot n_{bi} + e_{IMV} \cdot n_{IMV} + e_{MVD} \cdot \sum_{\forall MVD \neq 0} \log_2 |MVD| + e_{CSBF} \cdot n_{CSBF}, \tag{2.10}$$

and the simple model [7] is

$$\hat{E}_{PB,s} = e_{PB,slice} \cdot n_{PB,slice} + \hat{E}_{pred,s} + \hat{E}_{trans,s} + e_{frac\_hor\_depth2} \cdot n_{fracpel} + e_{bi} \cdot n_{bi}, \tag{2.11}$$

where fracType is short for the fractional pel filtering type, IMV for identical motion vector, MVD for motion vector difference, and CSBF for the coded sub block flag [7]. Compared to the accurate model, many parameters are simplified in the simple model. The specific energies are shown in Table 3.

For the accurate model, $e_{PB,slice} \cdot n_{PB,slice}$ is the energy used to initialize the P- or B-slice and decode its slice header, where $n_{PB,slice}$ is the number of P- and B-slices.

$\hat{E}_{pred,s}$ is the energy used to decompress the inter frame prediction modes and is defined by

$$\hat{E}_{pred,s} = \sum_{\forall PUtype} \left(\sum_{depth=0}^{3} e_{PUtype,depth} \cdot n_{PUtype,depth} \right), \tag{2.12}$$

where PU is the prediction unit and $n_{PUtype,depth}$ is the number of PUs. The specific energies $e_{PUtype,depth}$ are shown in Table 4.

$\hat{E}_{frac,a}$ is the energy used for prediction with fractional pel motion vectors and is defined by

$$\hat{E}_{frac,a} = \sum_{\forall fracType}\left(\sum_{depth=0}^{3} e_{fracType,depth} \cdot n_{fracType,depth}\right), \tag{2.13}$$

where the specific energies $e_{fracType,depth}$ are shown in Table 5. Three types of the fractional pel filtering (fracType) are horizontal or vertical filtering for luma fractional pels, and chroma half pel filtering [7]. For the luma case, every fractionally predicted luma pixel is counted to obtain $n_{fracType,depth}$. When the pixel is fractional in both horizontal and vertical direction, where the vertical filtering requires horizontal fractional pixels outside of the current PU, additional numbers of horizontal filtering should be added [1].

$\hat{E}_{trans,a}$ is the energy used for residual coefficient transformations and is defined by

$$\hat{E}_{trans,a} = \sum_{\forall comp}\left(\sum_{\forall size} e_{comp,size} \cdot n_{comp,size}\right), \tag{2.14}$$

where the specific energies $e_{comp,size}$ are shown in Table 6.

Table 3 Specific inter frame decoding energies [7]

$e_{\mathrm{PB,slice}}$	1.6486mJ	e_{bi}	411.57nJ
e_{IMV}	−703.50nJ	e_{MVD}	140.79nJ
e_{CSBF}	672.17nJ		

Table 4 Specific decoding energies for PUs in the inter frame prediction mode [µJ] [7]

$e_{\mathrm{PUtype,depth}}$	depth 0	depth 1	depth 2	depth 3
Skip	405.67	110.97	33.434	13.139
Merge 2N × 2N	413.74	200.61	50.333	16.606
Merge SMP	207.72	56.458	18.132	7.5813
Merge AMP	208.09	57.575	18.333	–
Inter 2N × 2N	428.68	115.89	35.625	14.828
Inter SMP	216.12	59.874	19.345	8.8222
Inter AMP	217.11	60.080	19.630	–
$e_{\mathrm{CU,depth}}$	420.85	115.21	36.125	14.790

Table 5 Specific fractional pel filtering energy [nJ] [7]

$e_{\mathrm{fracType,depth}}$	depth 0	depth 1	depth 2	depth 3
luma horizontal (hor)	26.214	27.728	32.206	44.351
luma vertical (ver)	33.803	35.997	41.160	54.272
chroma half pel (chr)	25.279	29.666	47.633	98.962

Table 6 Specific transformation energies per TU [μJ] [7]

$e_{\mathrm{comp,size}}$	32×32	16×16	8×8	4×4
luma	113.23	22.203	5.4291	2.2481
chroma	–	20.506	4.1150	1.4389

2.2.4 Model Evaluation

In order to evaluate the intra and inter frame decoding energy model, the estimated energy is compared to the measured energy. Table 7 [7] shows 19 differently coded videos to be tested. From the table, we can see that the range of OPs is from 0 to 45 and the resolution from 416 × 240 to 1920 × 1080. The videos Kimono (Ki), Bubbles (Bu), Basketball Pass (BP) and Race Horses (RH) are selected for testing, that are marked with bold configurations. The estimated energies of these six videos are compared the measured energies, whose results are shown in Fig. 2.6. [7]

For all the evaluated videos, the average of relative error $e = \frac{|E_{meas} - \hat{E}_{PB}|}{E_{meas}}$ is 2.34% for the accurate model and 3.63% for the simple model. The maximum error is not larger than 5.04% for the accurate model and 7.6% for the simple model. [7]

Table 7 Evaluated video parameters [7]

Index, name, size, slices, QP	Index, name, size, slices, QP
1. BP (LDB), 416×240, 24, 32	**11. Bu (RA)**, 416×240, 12 ,10
2. BP (LDB), 416×240, 24, 45	12. RH (LDB), 832×480, 12, 45
3. BP (LDP), 416×240, 24, 32	**13. RH (LDP)**, 832×480, 12, 45
4. BP (LDP), 416×240, 24, 45	**14. RH (LDB)**, 832×480, 6, 0
5. BP (RA), 416×240, 24, 32	15. RH (LDB), 832×480, 6, 2
6. BP (RA), 416×240, 24, 45	16. RH (LDB), 832×480, 6, 4
7. Bu (LDB), 416×240, 6, 0	17. RH (LDB), 832×480, 6, 6
8. Bu (LDB), 416×240, 6, 8	18. RH (LDB), 832×480, 6, 8
9. Bu (LDB), 416×240, 12, 10	**19. Ki (LDB)**, 1920×1080, 2, 45
10. Bu (LDB), 416×240, 6, 12	

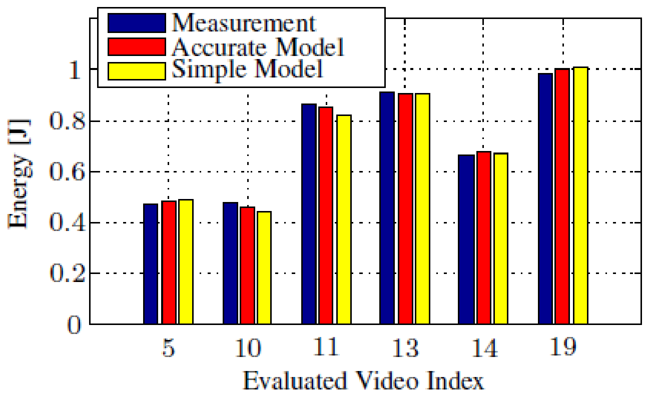


Figure 2.6 A comparison between the estimated and measured energy consumption for the videos marked in Table 5 with bold configurations [7]

In this thesis, the energy consumption model with in-loop filters is dealt with. For different hardware and software implementations, the specific energies are required to be known. In order to automatic determine the specific energies in different hardware and software, a new training and validating method that was only reported in prior work is performed, where we can only deal with the properties of this energy model, but not the types or properties of the hardware and software.

Chapter 3

Training and Validating

The statistical process to estimate the relationships among variables is known as training, which can be used to analyze the relationship between a dependent variable and one or more independent variables, for example, to observe and analyze how the value of the dependent variable changes when any one of the independent variables is varied and the other independent variables are fixed. A regression model involves unknown parameters, dependent and independent variables. In this thesis, the specific energies are the unknown parameters of the energy consumption model mentioned above, which are the final results to derive, and the occurrence numbers of the specific energies and the measured energies are the independent and dependent variables respectively. Since the energy consumption model here is a linear model, we focus on the linear regression in this thesis.

In linear regression, unknown model parameters are estimated from the given data that are modeled using linear predictor functions, where training and validating methods are required. In this section, three training methods, cross-validation method and their implementations on the energy model mentioned above will be introduced.

3.1 Training Methods

3.1.1 Least-Squares

For parameter estimation in linear regression, that is the aim of this thesis, a large number of procedures have been developed. These methods differ in computational simplicity of algorithms, robustness with respect to heavy-tailed distributions, presence of a closed-form solution, and theoretical assumptions needed to validate desirable statistical properties. Among these methods, least-squares is the simplest and most common estimator.

OLS is short for ordinary least squares. There are also some other kinds of least squares such as generalized least squares (GLS) and iteratively reweighted least squares (IRLS). Here, OLS is to be explained, as other kinds of least squares are normally an extension or variation of it.

A simple linear model with two parameters can be expressed by

$$y_i = ax_i + b; \quad i = 1, \dots, p \quad , \tag{3.1}$$

where x_i represents the independent variables, y_i is the dependent variables, p is the number of measurements, and $\boldsymbol{\alpha} = [a \quad b]^T$ is the unknown parameters of this model [10]. With least squares method, $\boldsymbol{\alpha}$ can be estimated when $\mathbf{X} = \left[X_1 \dots X_p\right]^T$ and $\mathbf{Y} = \left[Y_1 \dots Y_p\right]^T$ are given as input and output measurements. For this model, least squares can be distinguished into three types [10]:

- OLS1: Ordinary LS based on only output errors Δy_i,
- OLS2: Ordinary LS based on only input errors Δx_i,
- TLS: Total LS based on both input and output errors.

The fitting errors of these three types are defined as $\mathbf{e} = \left[e_1 \dots e_p\right]^T$,

$$e_i = \begin{cases} \Delta y_i & for\ OLS1 \\ \Delta x_i & for\ OLS2 \\ \sqrt{\left(\Delta {x_i}^2 + \Delta {y_i}^2\right)} & for\ TLS \end{cases} , \tag{3.2}$$

where $\Delta x_i = X_i - x_i$ and $\Delta y_i = Y_i - y_i$ [10]. Fig. 3.1 depicts fitting errors of these three types of least squares. (a) presents fitting errors for OLS based on only output errors, and thus e_i is vertical. (b) presents fitting errors for OLS based on only input errors, and thus e_i is horizon-tal. (c) presents fitting errors for Total LS based on both input and output errors, and thus e_i represents the distance from (X_i, Y_i) to the fitting line. Finally the unknown model parameters $\boldsymbol{\alpha}$ can be determined by minimizing the square of the fitting errors.

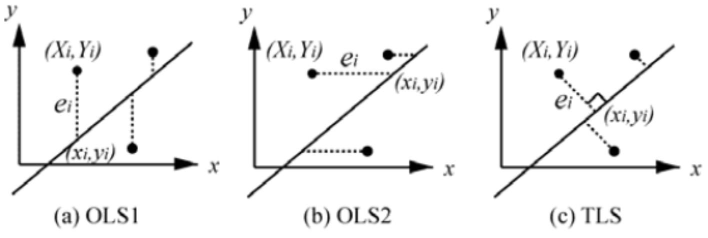


Figure 3.1 Fitting errors for least squares [10]

In the following, least squares is described in the form of formula. Here we define: $\bar{x} = \frac{1}{p}\sum_{i=1}^{p} x_i$, $\bar{y} = \frac{1}{p}\sum_{i=1}^{p} y_i$, $L_{xy} = \sum_{i=1}^{p}(x_i - \bar{x})(y_i - \bar{y}) = p(\overline{xy} - \bar{x}\cdot\bar{y})$, $L_{xx} = \sum_{i=1}^{p}(x_i - \bar{x})^2 = p\left(\overline{x^2} - \bar{x}^2\right)$, $L_{yy} = \sum_{i=1}^{p}(y_i - \bar{y})^2 = p\left(\overline{y^2} - \bar{y}^2\right)$, $r = \frac{L_{xy}}{\sqrt{L_{xx}L_{yy}}}$.

In the direction of y, the regression function is $y = a_y x + b_y$, and then the estimated $a_y = \frac{L_{xy}}{L_{xx}}$, $b_y = \bar{y} - a_y\bar{x}$. In the direction of x, the regression function is $y = a_x x + b_x$, and then the estimated $a_x = \frac{L_{yy}}{L_{xy}}$, $b_x = \bar{y} - a_x\bar{x}$.

If a linear system has more than one independent variables, the system is described as

$$y = a_0 + \sum_{j=1}^{m} a_j x_j. \tag{3.3}$$

In practical situation, the observation samples may be larger than the number of parameters. Here, the number of observations is defined as p, and x_{ij}, y_i are respectively the value of x, y in the i^{th} observation. The square of error is defined as

$$e = \sum_{i=1}^{p}(y_i - y)^2 = \sum_{i=1}^{p}\left(y_i - a_0 - \sum_{j=1}^{m} a_j x_{ij}\right)^2. \tag{3.4}$$

In order to minimize the error e, the following equation should be fulfilled:

$$\frac{\partial e}{\partial a_m} = -2\sum_{i=1}^{p}\left(y_i - a_0 - \sum_{j=1}^{m} a_j x_{ij}\right)x_{im} = 0, \tag{3.5}$$

and when $\left(x_{ij}, y_i\right)$ is used as input signal, the parameters can be then computed.

In this thesis, the parameters (the specific energies) of the energy consumption model mentioned above are considered to be known in advance. The linear model for least squares is described by

$$f(n_r, e_r) = e_1 n_1 + e_2 n_2 + \cdots, \tag{3.6}$$

and $$error_r = \left(y_r - f(n_r, e_r)\right)^2, \tag{3.7}$$

where e_r is the specific energy that is the parameter, n_r is the occurrence number of specific energy, y_r is the measured energy, and $f(n_r, e_r)$ is the estimated energy.

The MATLAB function *lsqcurvefit()* is at first directly applied. Two matrices containing the information of the measured energies and the occurrence number of parameters are used as input signals. The code is:

```
vecEnergiesOptimized = lsqcurvefit( @evaluateModel, basicVectors,
```

sequence_numbers, sequence_energies),

where *basicVectors* is the initial values of the parameters, *evaluateModel* is described by $\hat{E}_{dec} = \sum_{\forall\, routines} e_r \cdot n_r$, *sequence_numbers* and *sequence_energies* are respectively the occurrence numbers of the parameters and the measured energies.

The methods introduced in the following subsections are based on least squares. The following iterative methods are applied to minimize the error $(y_r - f(n_r, e_r))^2$ in the sense of least squares.

3.1.2 Newton's Method

Newton's method is also called Newton-Raphson method, which is used for deriving the better approximations to the solutions of a function. Each iteration in the Newton's method with one dimension is described as

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \tag{3.8}$$

where x_k and $f'(x_k)$ are respectively the value of x and its derivative in the k^{th} iteration. The procedure of Newton's method is as follows: a value is chosen at first as the initial value x_0, then the tangent line to the function f at the point $(x_k, f(x_k))$ is used for approximation, and the x-intercept of this tangent line is considered as a better approximation than the initial value. Here, the tangent line at the point $x = x_m$ is defined by $y_{tangent} = f'(x_m)(x - x_m) - f(x_m)$.

For the energy consumption model $\hat{E}_{dec} = \sum_{\forall\, routines} e_r \cdot n_r$ that has more than one dimension, the direction of approximation for the r^{th} parameter e_r in the k^{th} iteration is described as

$$e_{r,k+1} = e_{r,k} - \frac{f(x_{r,k})}{f'(x_{r,k})}, \tag{3.9}$$

where $f(x_{r,k}) = \left(E_{meas} - \hat{E}_{dec,k}\right)^2$, $f'(x_{r,k}) = \frac{\partial\left(E_{meas} - \hat{E}_{dec,k}\right)^2}{\partial e_r} = 2n_r \cdot \left(E_{meas} - \hat{E}_{dec,k}\right)$. The approximation process is repeated until the difference for e_r in two neighboring iterations is smaller than a given threshold, which is significantly small. This means that e_r has reached the point, where $f(x_{r,k})$ is the minimum.

3.1.3 Gradient Descent

Gradient descent is also called steepest descent, which is an optimization algorithm for finding the local minimum of a function. At every iteration, the current point moves one step proportional to the negative of the gradient of the function from the current position. If the step is proportional to the positive of the gradient, a local maximum of that function can be approached, that is called gradient ascent.

Every iteration of gradient descent can be defined as $x_{n+1} = x_n - \gamma_n \nabla F(x_n)$, where $F(x)$ is the model function, $-\nabla F(x_n)$ is the negative gradient at the point x_n, and γ_n is the step size. Fig. 3.2 depicts the procedure of the function shown above. The blue circles that are bowl shape are considered as $F(x)$. $F(x)$ decreases fastest if x goes from the current position in the direction of negative gradient of F, that is shown by the red arrows in the figure. The value in the middle of the blue circles is the minimum value. As Fig. 3.2 shows, $F(x_0) \geq F(x_1) \geq F(x_2) \geq \cdots$ is fulfilled, which means that after every iteration $F(x)$ approaches to the minimum value [11].

When gradient descent is expanded to a model with more variables, the procedure can be shown in Fig. 3.3. The function depicted in this figure has many local minima. If Q_1 is chosen as the initial value, the global minimum can be achieved easily. If other points around which several local minima exist such as Q_2 and Q_3, the global minimum cannot be achieved, as gradient descent is an optimization algorithm to find the local minimum. As a result, the position of initial value is important for gradient descent. In order to overcome this drawback, variable step sizes, conjugate directions, and convergence checking are applied [12]. However, in this thesis we do not take this drawback into account, as the energy model dealt with here is a linear model. For the above introduced energy model $f(n_r, e_r) = e_1 n_1 + e_2 n_2 + \cdots$, the value of e_r when the estimation error $F(e_r) = (y_r - f(n_r, e_r))^2$ is minimum is required. In this case, each parameter e_r should fulfill the equation $\nabla F(e_r) = 0$, which has only one solution. As a result, we assume that this energy model has only one minimum that is considered as a global minimum.

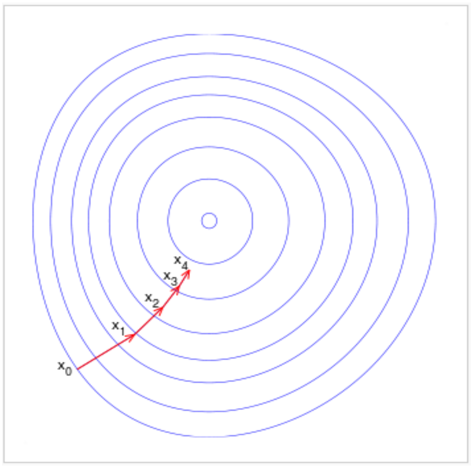


Figure 3.2 The process of gradient descent [11]

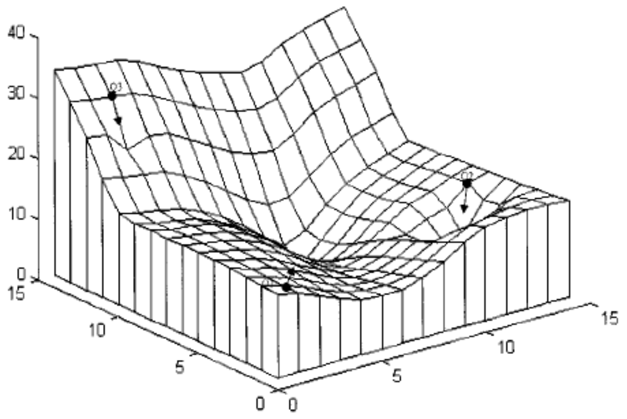


Figure 3.3 Gradient descent of a multi-model function [12]

For the energy consumption model dealt with in this thesis, that is described by $\hat{E}_{dec} = \sum_{\forall\, routines} e_r \cdot n_r$, the gradient in every iteration is

$$\frac{\partial F}{\partial e_r} = \frac{\partial (E_{meas} - \hat{E}_{dec})^2}{\partial e_r} = -2n_r\left(E_{meas} - \hat{E}_{dec}\right). \tag{3.10}$$

The MATLAB codes of this process are shown as follows:

```
for  k = 1 : training_size
        error  =  measured_energy  -  occurrence_num*initial_parameter;
        df(k) = sum(error.* occurrence_num(:,k));
end,
```

where *df(k)* is the gradient of the k^{th} parameter. After running the above codes, we can get a vector containing the gradients of all the parameters in this iteration. Then this gradient matrix is used to update the matrix of initial parameters, which can be used in the next iteration. The codes will repeat until the error is smaller than a defined small value, that can be seen as a threshold.

Gradient descent can also be used to deal with a system of linear equations. If the parameters of the system are unknown, it is parameter estimation. For example, a linear system with unknown parameters is defined as $\mathbf{Xa} = \mathbf{y}$, where $\mathbf{a}$ represents the parameters. The fitting error is defined as $E = ||\mathbf{Xa} - \mathbf{y}||^2$ in the sense of least squares. The parameters can be determined by minimizing E with gradient descent. In the i^{th} iteration, the gradient is described by $\nabla E\left(\mathbf{a}^{(i)}\right) = 2\mathbf{X}^T\left(\mathbf{Xa}^{(i)} - \mathbf{y}\right)$, and the parameters are refreshed by $\mathbf{a}^{(i+1)} = \mathbf{a}^{(i)} - \gamma_i \nabla E\left(\mathbf{a}^{(i)}\right)$, where γ_i is the step size.

However, for a linear model like the energy model used in this thesis, gradient descent is not a common method, as conjugate gradient method is a popular alternative.

3.1.4 Conjugate Gradient Method

Conjugate Gradient Method (CG) is an algorithm combining the Newton's method and gradient descent to solve the linear equations. Normally it is applied using an iterative algorithm. The linear system $\mathbf{Xa} = \mathbf{y}$, that is defined above, is still taken as an example here. In the Conjugate Gradient Method, in order to get the parameters $\mathbf{a}$ we should minimize the following function [13]:

$$f(a) = \frac{1}{2}(\mathbf{a}_{exa} - \mathbf{a})^T\mathbf{X}(\mathbf{a}_{exa} - \mathbf{a}), \tag{3.11}$$

where $\mathbf{a}_{exa}$ is the exact solution. The gradient is $-\nabla f(a) = \mathbf{X}(\mathbf{a}_{exa} - \mathbf{a}) = \mathbf{y} - \mathbf{Xa}$. We define p_0 as the negative gradient of f at $a = a_0$, where a_0 is the initial value of a. In the Conjugate Gradient Method, the first iteration is the same as the Gradient Descent, as a result, we can get $\mathbf{p}_0 = \mathbf{y} - \mathbf{Xa}_0$, where $\mathbf{p}$ is the moving direction of the approximation to $\mathbf{a}_{exa}$. At the kth iteration, we define $\mathbf{r}_k = \mathbf{y} - \mathbf{Xa}_k$, where $\mathbf{r}_k$ is the negative gradient of f at $\mathbf{a} = \mathbf{a}_k$. Since the approximation direction is conjugate to the gradient, based on Gram–Schmidt orthonormalization we have [13]:

$$\mathbf{p}_k = \mathbf{r}_k - \sum_{i<k} \frac{\mathbf{p}_i^T\mathbf{X}\mathbf{r}_k}{\mathbf{p}_i^T\mathbf{X}\mathbf{p}_i}\mathbf{p}_i, \tag{3.12}$$

and the next position is $\mathbf{a}_{k+1} = \mathbf{a}_k + \alpha_k\mathbf{p}_k$, where $\alpha_k = \frac{\mathbf{p}_k^T\mathbf{y}}{\mathbf{p}_k^T\mathbf{X}\mathbf{p}_k} = \frac{\mathbf{p}_k^T(\mathbf{r}_{k-1}+\mathbf{X}\mathbf{a}_{k-1})}{\mathbf{p}_k^T\mathbf{X}\mathbf{p}_k} = \frac{\mathbf{p}_k^T\mathbf{r}_{k-1}}{\mathbf{p}_k^T\mathbf{X}\mathbf{p}_k}$, which is the step size [13]. The approximation direction of $\mathbf{a}$ is conjugate to the gradient, and thus this method is named as conjugate gradient method [13].

The procedure of conjugate gradient method is shown in the following steps [13], where β_k is the coefficient that guarantees the moving direction **p** and the gradient **r** are conjugate.

Algorithm 1:

1) $\mathbf{r}_0 = \mathbf{y} - \mathbf{X}\mathbf{a}_0$, $\mathbf{p}_0 = \mathbf{r}_0$, $\mathrm{k} = 0$
2) $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{X} \mathbf{p}_k}$
3) $\mathbf{a}_{k+1} = \mathbf{a}_k + \alpha_k \mathbf{p}_k$
4) $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{X} \mathbf{p}_k$
5) If r_{k+1} is smaller than a defined small value then exit the loop, otherwise,
6) $\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
7) $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
8) *$k = k + 1$, Shift 2).*

The final result of this procedure is $\mathbf{a}_{k+1}$.

3.1.5 Gauss-Seidel Method

For a linear system with equations, the Gauss–Seidel method, which is similar to the Jacobi method, can also be used to solve the linear system. Compared to the Jacobi method that can only be applied to solve a diagonally dominant system of linear equations, the Gauss–Seidel method can be applied to any matrix with non-zero elements on the diagonals, which can be easier fulfilled in this work. [14]

Since the Gauss-Seidel method is similar to the Jacobi method, the Jacobi method is also introduced here. The Jacobi method can be described in the form of matrices. A system with *n* linear equations is defined as $\mathbf{X}\mathbf{a} = \mathbf{y}$, where:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix}, \qquad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Then the matrix **X** is divided into two matrices. One is a diagonal matrix **D**, and the other is the remainder **R**:

$$\mathbf{A} = \mathbf{D} + \mathbf{R}, \text{ where } \mathbf{D} = \begin{bmatrix} x_{11} & 0 & \dots & 0 \\ 0 & x_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & x_{nn} \end{bmatrix} \text{ and } \mathbf{R} = \begin{bmatrix} 0 & x_{12} & \cdots & x_{1n} \\ x_{21} & 0 & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & 0 \end{bmatrix}.$$

The solution of this system is achieved by iteration, which is defined as [15]:

$$\mathbf{a}^{(k+1)} = \mathbf{D}^{-1}\left(\mathbf{y} - \mathbf{R}\mathbf{a}^{(k)}\right), \tag{3.13}$$

where $\mathbf{a}^{(k)}$ is the value of $\mathbf{a}$ at the k^{th} iteration, and $\mathbf{a}^{(k+1)}$ is the value of $\mathbf{a}$ at the $(k+1)^{th}$ iteration.

This process can also be described in the form of elements as follows:

$$a_i^{(k+1)} = \frac{1}{x_{ii}}\left(y_i - \sum_{j\neq i} x_{ij} a_j^{(k)}\right), \quad i = 1,2,\dots,\text{n}. \tag{3.14}$$

As the above formula shows, the computation in every iteration of the Jacobi method requires not only the value of a computed at the previous iteration, but also all of the other values in a. As a result, the value of a cannot be overwritten in each iteration, which can be however performed in the Gauss–Seidel method.

An important limitation of the Jacobi method is that the matrix *X* must be strictly diagonally dominant. The matrix *X* is diagonally dominant if

$$|x_{ii}| \geq \sum_{j\neq i} |x_{ij}|, \tag{3.15}$$

which means that the magnitude of the diagonal element of each row is not smaller than the sum of the magnitudes of all the other elements in that row.

Unlike the Jacobi method, such limitation does not apply to the Gauss–Seidel method. The Gauss–Seidel method can be applied to any matrix in which the elements on the diagonals are non-zero.

The Gauss–Seidel method will be introduced in the following parts. Here, the model defined above with linear equations $\mathbf{X}\mathbf{a} = \mathbf{y}$ is taken as an example. In this model, **a** represents the parameters to be solved, and **X**, **y** are respectively the independent and dependent variables. **X** is an (n×n) sparse matrix, and **a**, **y** are vectors of length *n*. To solve this system of linear equations an iterative method is used, which attempts to find a solution approximating the real parameters. The Gauss–Seidel method can be described as [16]:

$$a_i^{(k+1)} = \frac{1}{x_{ii}}\left(y_i - \sum_{j<i} x_{ij} a_j^{(k+1)} - \sum_{j>i} x_{ij} a_j^{(k)}\right), \tag{3.16}$$

where $a_i^{(k)}$ is the i^{th} element of **a** at the k^{th} iteration, $a_i^{(0)}$ is the initial value of i^{th} element in **a**, and x_{ij}, y_i are respectively the elements of **X**, **y**. At the $(k+1)^{th}$ iteration of the Jacobi method, all the elements of $\mathbf{a}^{(k+1)}$ are computed from $\mathbf{a}^{(k)}$, while in the Gauss-Seidel Method the i^{th} element of $\mathbf{a}^{(k+1)}$ is computed from the already renewed elements of $\mathbf{a}^{(k+1)}$ before the i^{th} element and the remaining elements of $\mathbf{a}^{(k)}$ after the i^{th} element.

The Gauss–Seidel method can also be described in the form of matrices [16]:

$$\mathbf{a}^{(k+1)} = (\mathbf{D}+\mathbf{L})^{-1}\left[\mathbf{y} - \mathbf{U}\mathbf{a}^{(k)}\right], \tag{3.17}$$

where $\mathbf{a}^{(k+1)}$ is the values of **a** at the $(k+1)^{th}$ iteration, **D** is the diagonal of **X**, **L** and **U** are respectively the strictly lower and upper triangle component of **X**, which are shown as follows:

$$\mathbf{L} = \begin{bmatrix} x_{11} & 0 & \dots & 0 \\ x_{21} & x_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ 0 & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & x_{nn} \end{bmatrix}.$$

The procedure of solving the linear equations by the Gauss–Seidel method is presented in Algorithm 2, where **I** is the identity matrix [17].

Algorithm 2:

1) Compute the strictly lower and upper component **L**, **U** of **X**.
2) Compute the Gauss–Seidel iteration matrix $(\mathbf{I}-\mathbf{L})^{-1}\mathbf{U}$.
3) Compute $\rho((\mathbf{I}-\mathbf{L})^{-1}\mathbf{U}) = \max\left\{\left|\frac{x_{12}x_{21}}{x_{11}x_{22}}\right|, \left|\frac{x_{32}x_{23}}{x_{22}x_{33}}\right|, \dots, \left|\frac{x_{n-1,n}x_{n,n-1}}{x_{n-1,n-1}x_{nn}}\right|\right\}$. If $\rho((\mathbf{I}-\mathbf{L})^{-1}\mathbf{U}) < 1$, turn back to 3); otherwise, continue to 4).
4) Compute the approximate solutions of $\mathbf{Xa} = \mathbf{y}$ iteratively until the precision requirement is met. Firstly, compute $\mathbf{C} = \mathbf{D}^{-1}\mathbf{y} = \left(\frac{y_1}{x_{11}}, \frac{y_2}{x_{22}}, \dots, \frac{y_n}{x_{nn}}\right)^T$. Secondly, compute $\mathbf{G} = (\mathbf{I}-\mathbf{L})^{-1}\mathbf{U}$, and $\mathbf{H} = (\mathbf{I}-\mathbf{L})^{-1}\mathbf{C}$. At last, compute the vectors $\mathbf{a}^{(0)}, \mathbf{a}^{(1)}, \dots, \mathbf{a}^{(n)}, \dots$ based on the formula $\mathbf{a}^{(k+1)} = \mathbf{G}\mathbf{a}^{(k)} + \mathbf{H}$. If $\left\|\mathbf{a}^{(k+1)} - \mathbf{a}^{(k)}\right\|$ meets the precision requirement, that is to say, is smaller than a given small value, $\mathbf{a}^{(k+1)}$ is the final approximate solution.

3.2 Validating Methods

3.2.1 Purpose of Cross-Validation

Validating methods are used to assess how the estimated results of a statistical analysis generalize to a set of independent data. A phenomenon named overfitting will occur, when a statistical model describes random error or noise instead of the accurate relationship between the dependent and independent variables. This phenomenon generally occurs when a model is relatively complex, e.g. provides many parameters relative to the observations. The energy model dealt with in this thesis is in such situation. To overcome overfitting, validating methods are required. In a validating method, a dataset from the original data to be trained is defined to test the estimated model derived by the training methods, and thus we can analyze how the estimated model will generalize to an input dataset.

However, in practical operation, only one set of validating data cannot evaluate the estimated model accurately, as the condition such as irradiances, temperatures, and load or grid impedance changes all the time [18]. In order to evaluate the model more precisely, the cross validation is applied. The goal of cross-validation is to estimate the model derived by training methods to a dataset that is independent of the training data [18]. The cross-validation is also called rotation estimation, as it is generally performed by using a different dataset as validating data, and the results of validation will be averaged over the rounds [18]. Many methods of cross-validation have been developed such as leave-one-out cross-validation and k-fold validation. In the next subsection, several types of cross-validation will be introduced.

3.2.2 Types of Cross-Validation

The cross-validation can be distinguished into two main types: exhaustive and non-exhaustive cross-validation. In the cross-validation, when all the possible ways of dividing the original signal into training and validating data are taken into account, it is called exhaustive cross-validation, while in non-exhaustive cross-validation, only parts of the possible ways of dividing the original signal are computed.

Exhaustive cross-validation

One of the exhaustive cross-validation methods is leave-*p*-out cross-validation, in which *p* observations are used in each iteration as validating data and the remaining observations as training data. This will be repeated until all the ways to divide the original sample into a validating dataset of p observations and a training dataset are computed. If n is the number of observations, leave-p-out cross-validation will validate $C_p^n = \frac{n!}{p!(n-p)!}$ times.

Another one is leave-one-out cross-validation. For leave-p-out cross-validation, if $p = 1$, it is called leave-one-out cross-validation. As a result, leave-one-out cross-validation is a particular case of leave-p-out cross-validation. Obviously leave-one-out cross-validation will validate $C_1^n = n$ times with n observations.

Non-exhaustive cross-validation

One of the non-exhaustive cross-validation is k-fold cross-validation. In k-fold cross validation, a sample set is defined as **S**, which is randomly divided into k parts with equal size. Here,

$$\mathbf{S} = \{S_1, S_2, \dots, S_k\}. \tag{3.18}$$

In each iteration, one part is selected out of the training process as validating dataset and the remaining *k-1* parts are regarded as training dataset. This process is repeated k times, as k-fold cross-validation. In this way, every subsample is used once as the validating data. In k-fold cross-validation, the value of k is commonly 5 or 10. When $k = n$, the k-fold cross-validation is then the leave-one-out cross-validation introduced above.

Another method of non-exhaustive cross-validation is repeated random sub-sampling validation, in which the original dataset is split into training and validating data randomly. Compared to the k-fold cross-validation, the sample number of training or validating dataset is not dependent on the number of iterations, that is considered as an advantage. However, some original samples may never be selected as validating data, while some others may be selected several times, that is to say, the validating data may overlap, that is considered as a disadvantage.

In this thesis, the k-fold cross-validation is used to evaluate the estimated model. Here, $k = 10$. All the measured energies and the occurrence numbers of the specific energies are divided randomly into ten subsets, and one subset is chosen as the validating dataset at every iteration. This process is repeated ten times, and thus all the samples are computed once as the validating data.

3.2.3 Applications of Cross-Validation

The cross-validation can be applied in various areas. For example, the cross-validation is considered as one of the most popular approaches to estimate the classification error rate in gene expression data classification [19]. Another application is to compare the performances of different predictive modeling procedures, for example, the performances of support vector machines (SVM) and k nearest neighbors (KNN) for deriving the true character from a handwritten image. Furthermore, the cross validation can be applied in variable selection. In practical operation, we do not want to use all the original samples for training, but only parts of the samples, where cross-validation is used to find the subset of original samples with best performance. [20]

Chapter 4

Evaluation

In Chapter 2, the energy consumption model for HEVC has been introduced, in which the parameters (the specific energies) are considered to be known in advance. However, deriving the specific energies in advance is generally not efficient. As a result, the goal of this thesis is to automatically determine the model parameters of the energy consumption model introduced in Chapter 2, which is described by $\hat{E}_{dec} = \sum_{\forall\, routines} e_r \cdot n_r$, where e_r is the specific energy of each routine, n_r is the occurrence number of each routine, and $\hat{E}_{dec}$ is the estimated energy.

Here, 2400 bit streams, whose exact energies have been measured and thus are already known, are used as the original samples. In the measurement of the decoding energy, a Pandaboard, which is similar with the architecture of smartphone, is used. The Pandaboard used here features an OMAP4430 SoC with an ARMv7 dual core processor [8].

$\mathbf{n}_\mathrm{r}$ is a (2400×90) matrix, which is also already known and provides the information that the energy model of these samples has 90 parameters. Each row of the matrix $\mathbf{n}_\mathrm{r}$ represents the occurrence number of all the 90 parameters in a certain bit stream. With $\mathbf{n}_\mathrm{r}$ and the measured energies of the samples $\mathbf{E}_\mathrm{meas}$, the specific energies of each routine can be estimated by training methods introduced in Chapter 3. Since the specific energies are regarded as the parameters of this energy model, this procedure is called parameter estimation. In order to select an appropriate dataset as the training data, in other words, to select a dataset with the minimized fitting error, cross-validation introduced in Chapter 3 is applied.

In this section, three types of the estimation error used for evaluation will be firstly introduced. Then the evaluation results of several training methods already introduced in Chapter 3 will be presented and analyzed.

4.1 Types of Estimation Error

In the energy consumption model introduced in the above sections, the specific energy e_r is regarded as the parameter which is unknown, and the occurrence number n_r and the measured energy E_{meas} are the original data for training which are already known. In this case, three types of the estimation error are used in this thesis, which are defined by

$$Error_{absolute} = \frac{1}{M}\sum_{m=1}^{M}|E_{est} - E_{meas}|, \tag{4.1}$$

$$Error_{relative} = \frac{1}{M}\sum_{m=1}^{M}\frac{|E_{est}-E_{meas}|}{E_{meas}}, \tag{4.2}$$

$$Error_{MSE} = \frac{1}{M}\sum_{m=1}^{M}(E_{est} - E_{meas})^2, \tag{4.3}$$

where M is the number of samples for evaluation. For each sample, the absolute error $Error_{absolute}$ is the magnitude of the absolute difference between the estimated energy and the measured energy, and the relative error $Error_{relative}$ is the absolute error divided by the magnitude of the measured energy. The MSE is short for the mean squared error, which is the square of the absolute error.

4.2 Evaluation Results

At first, all the 2400 samples are used for training. Table 8 and Figure 4.1, 4.2 present the results with the method of the MATLAB function *lsqcurvefit()* and the gradient descent. The initial values of the parameters are defined by $e_{r,initial} = \frac{\sum_{m=1}^{M} E_{meas}}{N\sum_{m=1}^{M} n_r}$, where M is the number of training samples and N is the number of parameters. As we now use all the 2400 samples with 90 parameters for training, $M = 2400$ and $N = 90$. These initial parameters are defined with the assumption that the energy consumption of each routine in one sample is equal. In these two methods, different initial values do not affect the estimation error, but the processing time, as the same estimation error can be achieved with different initial parameters. When the initial values are all zero, 10^4 more iterations are performed with the method of gradient descent to achieve the same estimation error. And in the method of *lsqcurvefit()*, the processing time is 0.2837, that is a little longer than the processing time with $e_{r,initial}$. In Figure 4.1 and 4.2, the estimation error is reduced with increasing iteration numbers.

Table 8 Estimation error with 2400 training samples

lsqcurvefit			
$Error_{absolute}$	$Error_{relative}$	$Error_{MSE}$	Time
0.0284	2.3839%	0.0025	0.2483
Gradient Descent			
$Error_{absolute}$	$Error_{relative}$	$Error_{MSE}$	Iterations
0.2894	6.3124%	0.2508	0.5×10^4
0.2770	6.0432%	0.2290	1×10^4
0.2694	5.8834%	0.2160	1.5×10^4
0.2646	5.7761%	0.2074	2×10^4
0.2613	5.7012%	0.2013	2.5×10^4
0.2589	5.6583%	0.1957	3×10^4
0.2574	5.6123%	0.1919	3.5×10^4
0.2561	5.5921%	0.1888	4×10^4

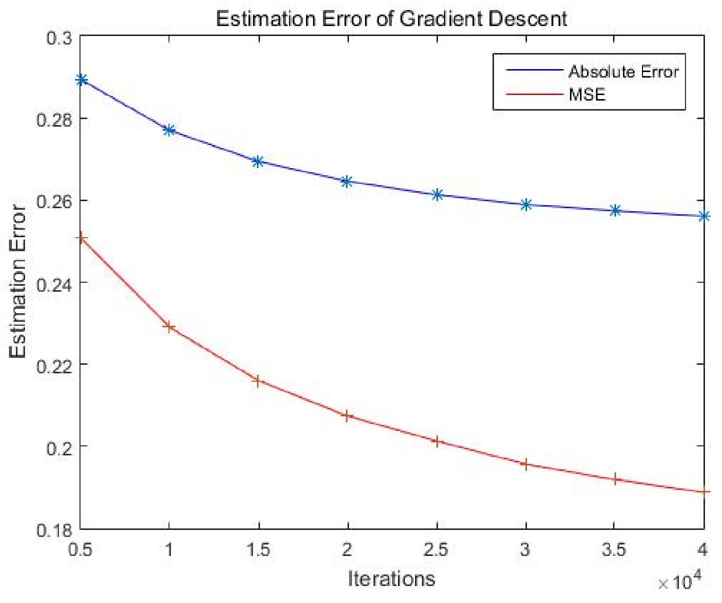


Figure 4.1 Estimation errors with 2400 training samples(1)

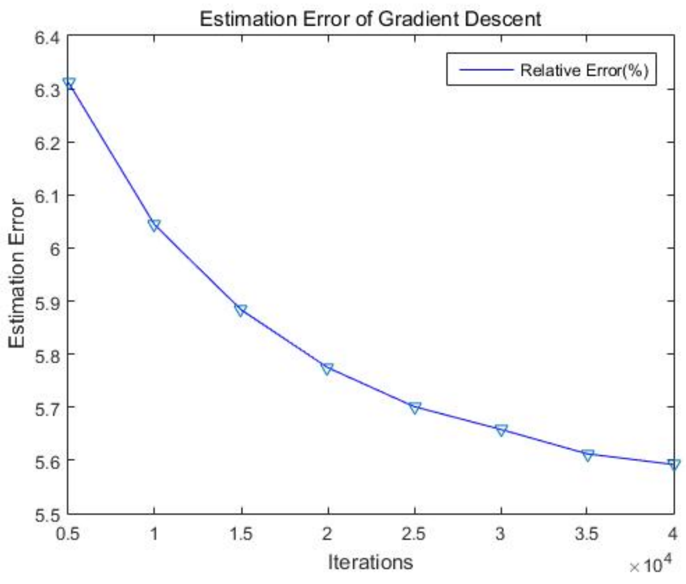
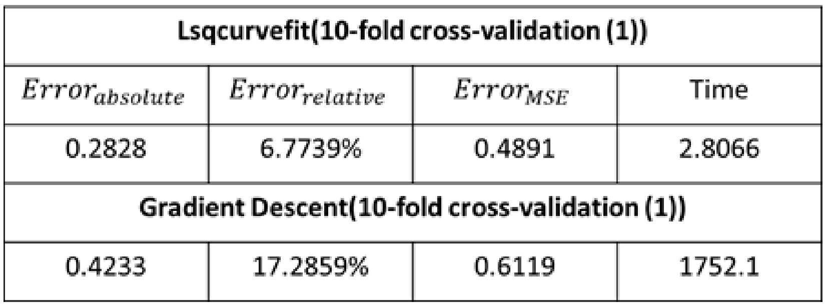


Figure 4.2 Estimation errors with 2400 training samples(2)

Then the cross-validation is applied to evaluate the estimated model. In order to compare these two training methods, we do not divide the original samples randomly at first. As 10-fold cross-validation is used, each part has $\frac{2400}{10} = 240$ original samples. Here, the first 240 original samples are chosen as the first part, the next 240 samples as the second part and so on. This partition method is recorded as *10-fold cross-validation (1)*. In this case, the evaluation results are shown in Table 9. Here, the iteration number of the gradient descent is 4×10^4. In the method of gradient descent, the relative error is much higher than *lsqcurvefit*. When we divide the original samples in other ways, for example, the original samples are divided into ten sample sets orderly, and then the first sample of every set is selected as the first part for the cross-validation, the second sample of every set is selected as the second part for the cross-validation and so on. This partition method is recorded as *10-fold cross-validation (2)*. In this case, the evaluation results are shown in Table 10. The results are much better than the results with *10-fold cross-validation (1)*. Figure 4.3 and 4.4 present the results with different iteration numbers. With increasing iteration number, the results will be better.

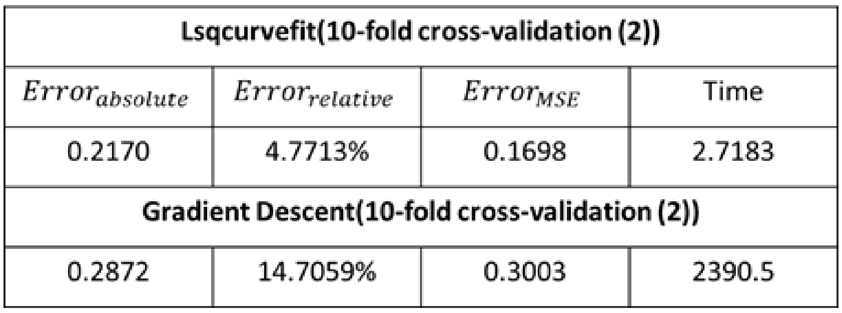
However, the relative error of gradient descent is about 14%, which is still considered as a large error. One of the possible reasons may be too many zeros in the occurrence numbers of the parameters. 2400 original samples are used as the training and validating data, of which the occurrence numbers of some parameters are often zero. For example, the occurrence number of the 28th parameter is zero in 1635 original samples, while the total number of the original samples is 2400, in other words, the 28th parameter does not exist in about 68% of the original samples. If all the 240 validating samples are chosen from these 1635 samples, where the 28th parameter does not exist, the evaluation error will be very large. Another reason may be the large difference between the occurrence numbers of the same parameter in different original samples. For example, in the occurrence numbers of the 28th parameter, 1635 values are zero, but the average of the other 765 values is about 1770 and the maximum is 41455. In this case, the evaluation error will be very sensitive to different partition methods in the cross-validation. When the validating samples are selected randomly, the evaluation error varies. Finally, the training method with 10-fold cross-validation is performed 20 times, and each time the validating samples are chosen randomly. The relative error of the MATLAB function *lsqcurvefit()* and the gradient descent lies respectively from 2.41% to 8.15% and from 11.06% to 19.72%. In order to solve this problem, pre-processing of the training data may be necessary, that is to say, the training samples are analyzed and processed before training, which is not solved in this thesis.

Table 9 Evaluation errors with cross-validation (1)



Lsqcurvefit(10-fold cross-validation (1))			
$Error_{absolute}$	$Error_{relative}$	$Error_{MSE}$	Time
0.2828	6.7739%	0.4891	2.8066
Gradient Descent(10-fold cross-validation (1))			
0.4233	17.2859%	0.6119	1752.1

Table 10 Evaluation errors with cross-validation (2)



Lsqcurvefit(10-fold cross-validation (2))			
$Error_{absolute}$	$Error_{relative}$	$Error_{MSE}$	Time
0.2170	4.7713%	0.1698	2.7183
Gradient Descent(10-fold cross-validation (2))			
0.2872	14.7059%	0.3003	2390.5

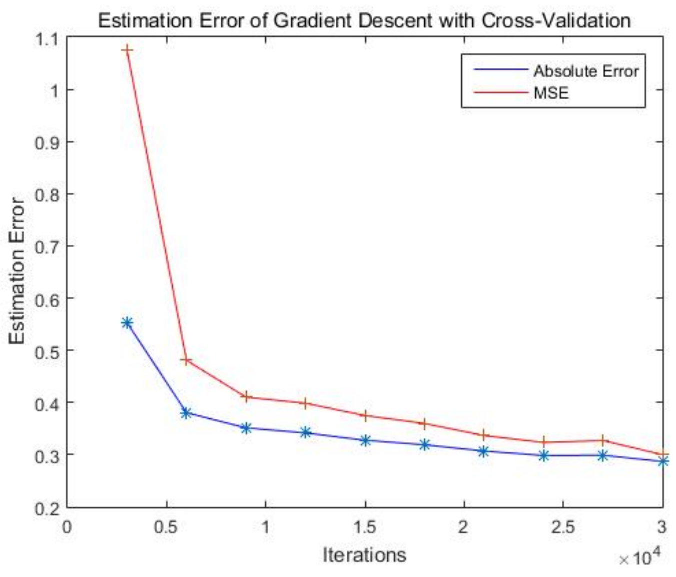


Figure 4.3 Estimation error of gradient descent with cross-validation (1)

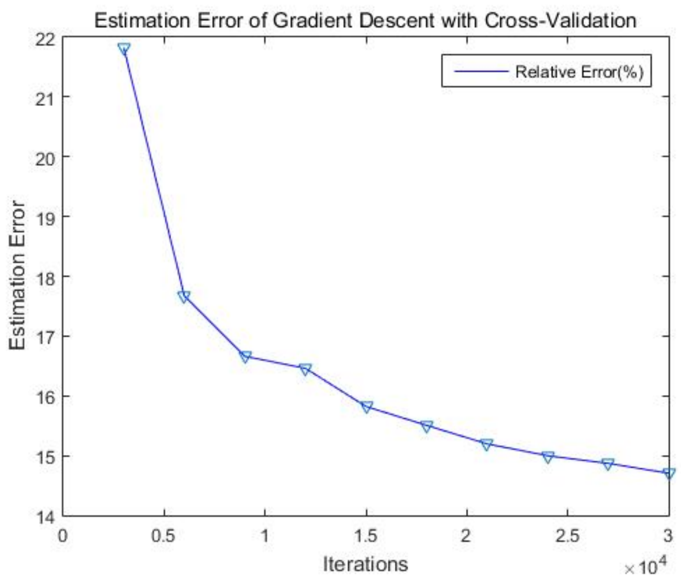


Figure 4.4 Estimation error of gradient descent with cross-validation (2)

For the conjugate gradient method and the Gauss-Seidel method, we use different ideas from the two methods presented above. As these two methods are used to solve a system of linear equations, at each iteration 90 original samples with the occurrence numbers of the 90 parameters are selected randomly to form a 90×1 vector $\mathbf{E}_{meas}$ and a 90×90 matrix $\mathbf{n}$ carrying the information of occurrence numbers. After each iteration, a vector considered as the parameters can be derived. At last, the average value of the parameters computed from each iteration is the estimated parameter of this model, but the parameters far away from the average value will not be taken into account. When 90 original samples are chosen for training, evaluation results computed with the same samples are shown in Table 11.

Table 11 Estimation error of 90 samples

Conjugate Gradient		
$Error_{absolute}$	$Error_{relative}$	$Error_{MSE}$
0.2845	7.8713%	0.1115
Gauss-Seidel		
0.2393	6.7059%	0.1013

However, the estimation results are large, when these two methods are used for training more samples. This idea can be seen as a special case of cross-validation. Each iteration of this idea is leave-*p*-out cross-validation. The estimation errors are large and do not converge in these two methods. As only 90 samples are selected as the training data in each iteration, the estimation error is very sensitive to different selections. Furthermore, a training set of 90 samples is such a small set that the influence of noise cannot be avoided. As a result, the estimation errors may be very large and do not converge.

Another problem for the Gauss-Seidel method is that the 90 samples at each iteration cannot be selected randomly. In the Gauss-Seidel method, the elements on the diagonals are non-zero. As a result, the occurrence numbers of different routines are checked and the non-zero values are chosen to form the matrix **n**. This is a limitation to the Gauss-Seidel method, as some original samples whose occurrence numbers in a certain video sequence are zero cannot be chosen for training. This is also a reason for the high estimation error.

Chapter 5

Conclusion

In order to develop an energy-efficient algorithm for video decoding, the measurement of the decoding energy is required to be efficient and less costly. An existing energy consumption model was developed to solve this problem, as this model computes the decoding energy by simply multiplying the specific energies with their corresponding occurrence numbers. However, the specific energies should be known in advance. In this thesis, several training methods are implemented on this energy model to determine the specific energies of different routines automatically. Then cross-validation is applied to evaluate the estimated model. The gradient descent is selected as the training method. Compared to the absolute error and MSE, the relative error of the estimated model is relatively high. In order to solve this problem, the original samples can be pre-processed, for example, the original samples are tested before training, so that the samples with too much noise should not be taken into account. Furthermore, the situation where the occurrence numbers have too many zero values should also be avoided.

List of Figures

List of Tables

References

[1] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, “Overview of the high efficiency video coding (HEVC) standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1649 –1668, Dec. 2012.

[2] B. Bross, W.-J. Han, G. J. Sullivan, J.-R. Ohm, and T. Wiegand, High Efficiency Video Coding (HEVC) Text Specification Draft 9, document JCTVC-K1003, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), Oct. 2012.

[3] G. J. Sullivan, J. M. Boyce, Y. Chen, J.-R. Ohm, C. A. Segall, and A. Vetro (December 2013), “Standardized Extensions of High Efficiency Video Coding” (PDF). IEEE Journal on Selected Topics in Signal Processing (IEEE) 7 (6). Retrieved 2015-02-21.

[4] TK Tan, M. Mrak, V. Baroncini, and N. Ramzan (2014-05-18), “Report on HEVC compression performance verification testing,” JCT-VC. Retrieved 2014-05-25.

[5] F. Bossen, B. Bross, K. Sühring, and D. Flynn, “HEVC Complexity and Implementation Analysis,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, Dec. 2012.

[6] D. Marpe, H. Schwarz, and T. Wiegand, “Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 620–636, Jul. 2003.

[7] C. Herglotz, D. Springer, and A. Kaup, “Modeling the Energy Consumption of HEVC P- and B-Frame Decoding”.

[8] C. Herglotz; Elisabeth Walencik; A. Kaup. "Estimating the HEVC Decoding Energy Using the Decoder Processing Time“. IEEE Int. Symp. on Circuits and Systems (ISCAS), 05-2015.

[9] C. Herglotz, D. Springer, A. Eichenseer, and A. Kaup, “Modeling the Energy Consumption of HEVC Intra Decoding,” IEEE, 2013.

[10] Y. Kim, and J. Kim, “Linear Kinematic Model-Based Least Squares Methods for Parameter Estimation of a Car-Trailer System Considering Sensor Noises,” IEEE Information Science and Applications (ICISA), pp. 1-3, 2014.

[11] L. Liu and E. Feig, “A block-based gradient descent search algorithm for block motion estimation in video coding,” IEEE Trans. Circuits Syst. Video Technol., vol. 6, pp. 419–422, Aug. 1996.

[12] Oscal T., and C. Chen, “Motion Estimation Using a One-Dimensional Gradient Descent Search,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, no. 4, June 2000.

[13] Smith, S.T., “Linear and nonlinear conjugate gradient methods for adaptive processing,” IEEE International Conference, vol. 3, pp. 1834 – 1837, 1996.

[14] Golub, Gene H.; Van Loan, Charles F. (1996), Matrix Computations (3rd ed.), Baltimore: Johns Hopkins, ISBN 978-0-8018-5414-9.

[15] Gimenez, D.; van de Geijn, R.; Hernandez, V.; Vidal, A.M., “Exploiting the symmetry on the Jacobi method on a mesh of processors,” IEEE Parallel and Distributed Processing, 1996. PDP '96. Proceedings of the Fourth Euromicro Workshop, pp. 377-384, 1996.

[16] D. P. Koester, S. Ranka, and G. C. Fox, “A Parallel Gauss-Seidel Algorithm for Sparse Power System Matrices,” Supercomputing 94., Proceedings, pp. 184-193, 1994.

[17] F. Tianxiang, H. Dingxiu, and L. Hongxia, “Computer Realization of Solving Tri-diagonal Equations by Gauss-Seidel Method,” International Conference on Computational Intelligence and Security, pp. 667-670, 2010.

[18] N. Patcharaprakiti, K. Kirtikara, C.Jivacate, A.Sangswang, K.Tunlasakun, and B. Muenpinij, “System Identification with Cross Validation Technique for Modeling Inverter of PhotovoItaic System,” Mechanical and Electrical Technology (ICMET), pp. 594-598, 2010.

[19] K. Yang, H. Wang, G. Dai, S. Hu, Y. Zhang, and J. Xu, “Determining the repeat number of cross-validation,” 4th International Conference on Biomedical Engineering and Informatics (BMEI), vol. 3, pp. 1706-1710, 2011.

[20] P. Richard; C. Dennis (1984). “Cross-Validation of Regression Models,” Journal of the American Statistical Association 79 (387): 575–583. doi:10.2307/2288403. JSTOR 2288403.