LifeStep Internship Project4
Real-Time EMG Regression Pipeline
(TD + CNN + TCN)

Prepared by
Jungmyung Lee, Sangmin Park

모델 설계 방향 1. TD

TD

- RMS, MAV, WL, ZC, SSC (5개)

CNN

- 2-layer
- Filters: 16 → 32
- Kernel size: 3
- Activation: ReLU
- Output: compact representation (32–48 dims)

TCN

- 1~2 layer
- Hidden channels: 32~64
- Dilations: 1, 2
- Kernel size: 3

Output

- FC layer (input = TCN hidden channels) → Joint angle/torque (DOF)

Windowing(윈도우 기반 추론)

- 150ms window
- 10ms hop (100Hz control loop)

Sampling Rate: 1000 Hz

150 ms → **150 samples**

Windowing(윈도우 기반 추론)

- 150ms window
- 10ms hop (100Hz control loop)

🔸 처음 **1**번 예측하기까지 **(**시스템이 실행되고 나서**) 0.15**초가 필요하다**.**

🔸 그 이후에는 **0.01**초마다 **(10ms**마다**)** 새로운 관절각도를 계속 예측한다**.**

1️⃣ **Raw EMG 수집 (1000 Hz)**

⬇️

2️⃣ **Pre-processing (전처리)**

- Band-pass (20–450 Hz)
- Notch (50/60 Hz)

⬇️

3️⃣ **Normalization (캘리브레이션 & 채널 스케일 맞추기)**

⬇️

4️⃣ **Windowing (150ms / 10ms step)**

⬇️

5️⃣ **Segmentation (3 segments 등)**

⬇️

6️⃣ **TD feature 계산(MAV, RMS, WL, ZC, SSC)**

⬇️

7️⃣ **CNN/TCN 입력**

- **Band-pass filter:**
  - **20–450 Hz** (또는 20–500 Hz)
  - 저주파(움직임, baseline drift) 제거 + 고주파 잡음 컷
- **Notch filter:**
  - **50 Hz 또는 60 Hz notch** (실험 환경에 맞게)
  - 전원 라인 노이즈 제거

3) normalization을 robust하게 만들어줌

채널별 보정값(ref_gain)을 잡을 때:

- 100% MVC에서 RMS/MAV 구하면
  → 각 채널의 최대 사용 범위를 직접 측정한 값이 됨

→ 이후 실시간 EMG를

```ini
x_norm = x(t) / ref_gain
```

으로 나누면

- 사람마다 근전도 크기가 달라도
- 모델에는 거의 동일한 스케일로 들어감

TD

- RMS, MAV, WL, ZC, SSC (5개)

실제 상용 환경에서 검증된,
정확도·속도·안정성 밸런스 최적 세트라서
선택했다
-> 모두 연산량이 낮음. (곱셈/덧셈 수준)

특징	담당하는 정보
MAV	평균적인 근육 힘(amplitude)
RMS	근육 에너지(power)
WL	신호 변동량/복잡도(complexity)
ZC	신호의 주파수 특성(frequency)
SSC	파형 모양(shape)·변곡점

1.역사적·실험적 근거

- **Hudgins(1993)** 이후, MAV/WL/ZC/SSC 조합은 의수 제어·손동작 인식에서 **대표적인 TD feature set**으로 사용됨.

->RMS는 3단계 연산이 필요했기에 이때는 없었으나 MAV, MAVS, WL, ZC, SSC를 효율적인 계산을 하기 위해 사용했다고 함.

2. 실시간·임베디드 제약

FFT, 웨이블릿, 복잡한 엔트로피 계열 특징은 정확도는 약간 올라가도 **연산량과 지연시간이 커짐**

3. CNN과 TCN에서의 연산 최소화
-FDF, TFDF는 연산이 무거움.
-TD를 사용함으로써 연산 최소화.

4. 다른 TD들은 중복되는 내용이 많고
노이즈가 많은 경우가 많다.

모든 TD 특징을 비교하면 다음 결론이 나옴:

특징	정보량	노이즈 안정성	중복도	실시간 적합성	이유
VAR	낮음	보통	**높음(RMS)**	좋음	RMS 제공과 중복
IEMG	낮음	좋음	**높음(MAV)**	좋음	MAV와 동일 정보
WAMP	중간	**매우 약함**	낮음	나쁨	threshold 민감
Kurtosis	낮음	**매우 약함**	중간	나쁨	고차 통계량 취약
AR 계열	높음	약함	중간	**나쁨**	연산량 큼
MAV	높음	강함	낮음	**최고**	amplitude 핵심
RMS	높음	강함	낮음	**최고**	energy 핵심
WL	높음	강함	낮음	**최고**	복잡도 핵심
ZC	중간	보통	낮음	**최고**	주파수 핵심
SSC	중간	보통	낮음	**최고**	shape 핵심

A New Strategy for Multifunction Myoelectric Control

Bernard Hudgins, Philip Parker, *Senior Member, IEEE*, and Robert N. Scott, *Senior Member, IEEE*

Abstract—**This paper describes a novel approach to the control of a multifunction prosthesis based on the classification of myoelectric patterns. It is shown that the myoelectric signal exhibits a deterministic structure during the initial phase of a muscle contraction. Features are extracted from several time segments of the myoelectric signal to preserve pattern structure. These features are then classified using an artificial neural network. The control signals are derived from natural contraction patterns which can be produced reliably with little subject training. The new control scheme increases the number of functions which can be controlled by a single channel of myoelectric signal but does so in a way which does not increase the effort required by the amputee. Results are presented to support this approach.**

Background

MYOELECTRIC systems have received widespread use as controls of prosthetic devices for individuals with amputations or congenitally deficient upper limbs [1], [2]. Many systems are now available commerically to control a single device (hand, elbow, wrist). These systems extract a control signal based on an estimate of the amplitude [3], or on the rate of change [4] of the myoelectric signal (MES). This control signal is either derived from a single myoelectric control inputs or channels. The number of functions per control channel of a level coded or rate coded system is limited to at most two [6]. An attempt to increse the number of states per channel by using state feedback has been unsuccessful [7]. Other multifunction prostheses have been developed using several channels of amplitude coding [8], [9]. These require the existence of several electrode sites which are usually difficult if not possible to locate on high level amputees. The Boston elbow [5] and Utah arm [10] have been used with some success in combination with an electric hand but this has required the use of a mechanical switching arrangement or a switch based on a quick co-contraction to select which of the two devices is to be controlled. More elaborate multifunction prostheses have been attempted but the result is that training the user to isolate the required number of control muscles is impractical if not impossible [11].

The myoelectric signal is essentially a one-dimensional pattern and the methods and algorithms developed for patten recognition can be applied to its analysis. The information extracted from the myoelectric signal, represented in a feature vector, is chosen to minimize the control error. In order to

myoelectric patterns:

1) Mean Absolute Value —An estimate of the mean absolute value of the signal, $\overline{X}_i$, in segment i which is N samples in length is given by

$$\overline{X}_i = \frac{1}{N} \sum_{k=1}^{N} |x_k| \quad \text{for } i = 1, \cdots, I \tag{1}$$

where x_k is the kth sample in segment i and I is the total number of segments over the entire sampled signal.

2) Mean Absolute Value Slope —This is simply the difference between sums in adjacent segments, i and $i+1$, as defined by

$$\Delta\overline{X}_i = \overline{X}_{i+1} - \overline{X}_i \quad \text{for } i = 1, \cdots, I-1. \tag{2}$$

3) Zero Crossings —A simple frequency measure can be obtained by counting the number of times the waveform crosses zero. A threshold must be included in the zero crossing calculation to reduce the noise induced zero crossings. Assuming a system noise of 4 μV peak to peak and a system gain of 5000, this dead zone can be calculated to be ± 10 mV measured at the input to the A/D converter. Given two consecutive samples x_k and x_{k+1}, increment the zero crossing count, ZC, if

- EMG 채널 수: **8 channels**
- 윈도우 길이: **150 ms (150 samples)**
- segment(구간): **3개**
- TD 특징 5개: **MAV, RMS, WL, ZC, SSC**

📌 예시 표 (PPT 그대로 사용 가능)

Segment	Channel	MAV	RMS	WL	ZC	SSC
S1	C1	0.12	0.21	15.4	6	5
S1	C2	0.15	0.24	18.2	4	6
...	...	...	...	...	...	...
S3	C8	0.11	0.19	14.1	5	4

Raw EMG shape = [Channels, Samples]
= [8, 150]

EMG 150 samples
------S1------|------S2------|------S3------
50 ms 50 ms 50 ms

TD Feature Tensor 형태

TD Feature shape = [3 segments, 8 channels, 5 features]

🔷 CNN 입력을 위한 Flatten 과정

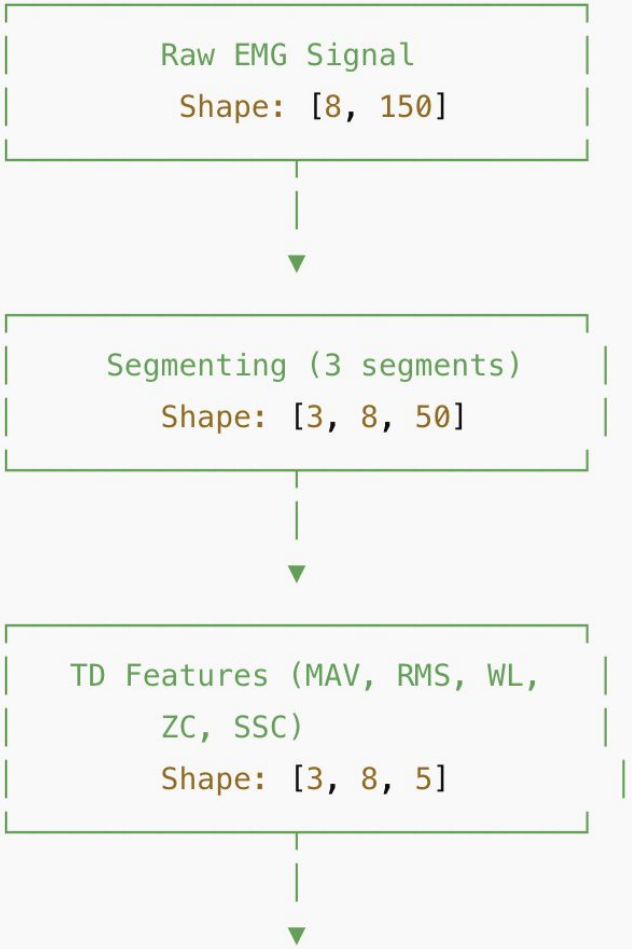
`segment × feature = 3 × 5 = 15`

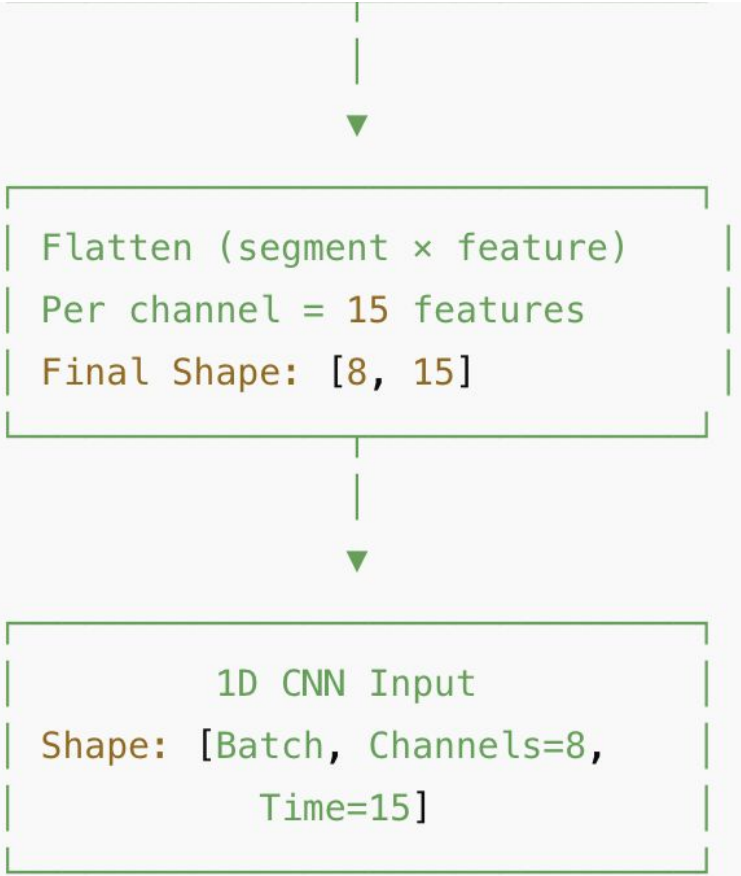
→ 각 채널당 15차원의 시퀀스가 만들어짐.

java 코드 복사

```
Flattened shape per channel = 15
CNN Input shape = [Batch, Channels=8, Time=15]
```

Time = [
  seg1_feat1, seg1_feat2, ..., seg1_feat5,
  seg2_feat1, seg2_feat2, ..., seg2_feat5,
  seg3_feat1, seg3_feat2, ..., seg3_feat5
]





**학습(train)할 때는 Batch>1
실시간에서 할때는 Batch=1 로
놓고 0.01초로 진행한다.

Time axis (15 TD features) -> 1Batch일 때 1sample(150ms window 1개의 예시)

<--->

seg1-MAV seg1-RMS seg1-WL seg1-ZC seg1-SSC
seg2-MAV seg2-RMS seg2-WL seg2-ZC seg2-SSC
seg3-MAV seg3-RMS seg3-WL seg3-ZC seg3-SSC

Channel 1 → [MAV1-1] [RMS1-1] [WL1-1] [ZC1-1] [SSC1-1]
[MAV1-2] [RMS1-2] [WL1-2] [ZC1-2] [SSC1-2]
[MAV1-3] [RMS1-3] [WL1-3] [ZC1-3] [SSC1-3]

Channel 2 → [MAV2-1] [RMS2-1] [WL2-1] [ZC2-1] [SSC2-1]
[MAV2-2] [RMS2-2] [WL2-2] [ZC2-2] [SSC2-2]
[MAV2-3] [RMS2-3] [WL2-3] [ZC2-3] [SSC2-3]

Channel 3 → [MAV3-1] [RMS3-1] [WL3-1] [ZC3-1] [SSC3-1]
[MAV3-2] [RMS3-2] [WL3-2] [ZC3-2] [SSC3-2]
[MAV3-3] [RMS3-3] [WL3-3] [ZC3-3] [SSC3-3]

...

Channel 8 → [MAV8-1] [RMS8-1] [WL8-1] [ZC8-1] [SSC8-1]
[MAV8-2] [RMS8-2] [WL8-2] [ZC8-2] [SSC8-2]
[MAV8-3] [RMS8-3] [WL8-3] [ZC8-3] [SSC8-3]

python-repl

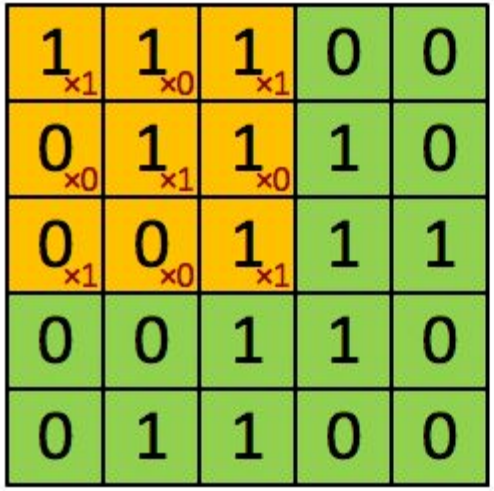
```
         time → t    t+1   t+2
ch1      w11   w12   w13
ch2      w21   w22   w23
ch3      w31   w32   w33
...
ch8      w81   w82   w83
```

여기서 중요한 점:

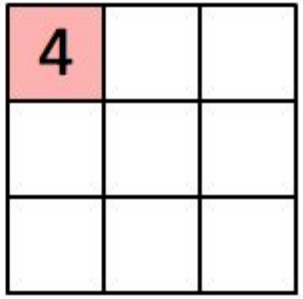
⭐ CNN은 이 전체 8×3 필터를

"한 번에 EMG patch 위에 올려놓고" 계산한다.

슬라이딩은 시간축으로만 이동한다.



Image



Convolved Feature

단어	V1	V2	V3	V4	…	Vp-2	Vp-1	Vp
W1								
W2								
W3								
…								
Wn-2								
Wn-1								
Wn								

모델 설계 방향 2. CNN, TCN

TD

- RMS, MAV, WL, ZC, SSC (5개)

CNN

- 2-layer
- Filters: 16 → 32
- Kernel size: 3
- Activation: ReLU
- Output: compact representation (32–48 dims)

TCN

- 1~2 layer
- Hidden channels: 32~64
- Dilations: 1, 2
- Kernel size: 3

Output

- FC layer (input = TCN hidden channels) → Joint angle/torque (DOF)

Windowing(윈도우 기반 추론)

- 150ms window
- 10ms hop (100Hz control loop)

CNN (Convolutional Neural Network) - 1D CNN

개념

- 원래는 이미지용으로 유명하지만
  여기서는 시간축에만 합성곱을 적용하는 **1차원 CNN**
- "슬라이딩 필터"로 EMG의 짧은 구간 패턴을 찾아내는 구조

왜 쓰는가 (EMG 기준)

- MUAP 형태, 짧은 burst, 급격한 변화 같은 **local temporal pattern**을 잘 잡는다
- TD로는 표현하기 어려운 복잡한 파형 모양을 자동으로 학습
- 전극 위치 변화나 개인차가 있어도 유용한 공통 패턴을 뽑아줄 수 있다

이 모델에서 역할

- CNN은 정제된 EMG와 TD 정보를 입력으로 받아
  근육 시너지 패턴과 국소적인 활성 패턴을 **압축된 특징 벡터(32~48차원)**로 바꾸는 "특징 인코더(feature encoder)" 역할을 한다.

1. CNN을 2층으로 둔 이유

1) 1층으로는 “근육 패턴 + 노이즈 분리”가 부족함

- 1-layer CNN은 사실상 **“조금 더 똑똑한 FIR 필터 뭉치”** 에 가깝다.
- 아주 로컬한 패턴(예: spike, 짧은 burst)만 보고
  “이게 진짜 근육 활성인지, 그냥 노이즈인지”를 충분히 구분하기 어려워.
- 특히 EMG는
  - 근육 발화(MUAP)
  - 전극 움직임 노이즈
  - 전력선 간섭
    이런 것들이 섞여 있어서, **한 번의 convolution만으로는 구조를 정리하기 힘듦.**

→ **2층으로 쌓으면**
1층: raw에 가까운 low-level 특징 추출 (에지, 작은 burst, 기본 waveform)
2층: 그 패턴들을 조합해서 근육 시그니처 / 잡음 패턴을 분리

2) 3층 이상은 EMG에서 “득보다 실이 많음”

- EMG 데이터는 길이와 피험자 수가 제한적이어서, 너무 깊은 CNN을 사용하면 과적합으로 가기 쉽다.
- 레이어가 늘어날수록
  - 파라미터 수↑
  - 연산량↑
  - latency↑ → 실시간성↓
  - 전력↑ → 배터리·발열 문제

특히 우리 목표는 **의수 실시간 제어 + 상업화 고려**라서
굳이 ResNet 같은 깊은 구조는 오버킬이다.

2. 필터 수를 **16 → 32**로 둔 이유

1) 첫 층 16 필터: “기본적인 근육 패턴 세트”

- 첫 CNN layer는 실제로 보면
  - 특정 주파수 대역 강조
  - 특정 waveform 모양 (뾰족한 MUAP, 완만한 파형 등)
  - 채널 간 비슷하게 반복되는 패턴 을 잡는 필터들이 생겨.
- EMG 채널 수가 예를 들어 8개 정도라고 하면, 16개 필터는 “채널 수의 2배 정도”라서 각 채널을 두세 개 스타일로 보는 정도의 표현력을 준다.
- 이 정도면:
  - 너무 단순하지도 않고
  - 그렇다고 noise까지 과하게 모델링하지도 않음.

2) 두 번째 층 32 필터: “조합 패턴 (시너지) 표현”

- 두 번째 층에서는
  - 여러 근육이 동시에 활성되는 패턴
  - 특정 task에서만 나타나는 시너지 패턴
  - noise와 true activation의 조합 형태 를 학습하게 된다.
- 이때 필터 수를 늘려 주면 1층에서 나온 “기본 패턴”들을 **여러 방식으로 recombination** 할 수 있음.

3) 왜 16→32지, 32→64나 64→128이 아닌가?

- EMG 데이터 양 + 상업용 의수라는 제약 때문에 **필터 수를 키울수록:**
  - 파라미터 수 폭증
  - 오버피팅 위험↑
  - 연산량↑(= latency, 전력↑)
- 16→32는
  - 첫 층: 가벼운 기본 필터
  - 둘째 층: 표현력을 한 번만 확장 하는 형태라 **정확도/연산/전력 사이의 밸런스가 좋다**고 볼 수 있다.

3. **Kernel size = 3**을 선택한 이유

1) k=3은 “로컬 패턴 + 계산량” 모두 잡는 최적값

- k=3이면, 한 번의 convolution에서
  현재 시점 + 바로 이전/다음까지 총 3개 샘플을 봄.
- 이걸 여러 층에 쌓으면
  - receptive field(실제로 보는 시간 길이)가 점점 커짐
  - 하지만 각 층은 여전히 “짧은 패턴”에 집중해서 학습 가능

EMG에서 중요한 건

- 아주 긴 추세(수 초 단위)가 아니라
- **수 ms ~ 수십 ms 정도의 근 활성 구간**이기 때문에
  k=3은 이걸 표현하기에 충분하다.

2) k=1, k=5,7과 비교해보면

- **k=1**
  - 그냥 point-wise linear transform 느낌
  - 시간 구조를 거의 못 씀 → EMG 시계열엔 별로
- **k=5,7**
  - 한 번에 보는 길이가 길어짐
  - but 파라미터 수 증가, 과적합 위험↑
  - 너무 긴 패턴을 한 번에 보려다 보니 “정교함”이 떨어질 수도 있음
  - TCN이 이미 긴 temporal dependency를 담당하고 있어서
    CNN에서 또 길게 볼 필요가 없음

→ CNN은 “짧은 패턴(근육 local pattern)”에 집중,
→ TCN이 “긴 패턴(동작 시퀀스)”을 담당하는 설계이기 때문에
k=3이 역할 분담 상 가장 자연스러운 선택.

4. Activation을 **ReLU**로 둔 이유

1) 계산이 가볍고, 임베디드에 적합

- ReLU(x) = max(0, x)
- 곱셈/나눗셈, exp 같은 것 없이
  비교 한 번, 분기 한 번이면 끝
  → ARM, 라즈베리파이, 마이크로컨트롤러에서 매우 빠름.

2) EMG 특성과도 잘 맞음

- rectified EMG, TD feature들은 원래 **비음수 (non-negative)** 스케일이 많음.
- ReLU는 음수는 잘라버리고, 양수만 전달해서
  - 중요 패턴만 살아남고
  - 잡음이 어느 정도 걸러지는 효과

3) 학습 안정성

- Sigmoid / tanh는 gradient vanishing 문제
- ReLU는 깊이가 조금만 있어도 gradient가 잘 흘러서 학습이 안정적

→ 전체적으로,
"경량 디바이스에서, 노이즈 많은 EMG를 다루기 좋은 기본 선택"이라 ReLU가 가장 무난하고 합리적이야.

5. Output을 **compact representation (32–48차원)**으로 만든 이유

1) 너무 많은 차원 → TCN이 과부하 + 오버피팅

- CNN output이 128, 256차원씩 나오면
  - TCN 입력이 매우 고차원
  - 파라미터 수 폭증
  - 오버피팅↑
  - 연산량↑ → 실시간성↓, 전력↑
- 특히 EMG는 데이터 수가 적어서
  고차원 feature는 그대로 overfit으로 연결됨.

2) 너무 적은 차원 → 정보 손실·표현력 부족

- 8~16차원 정도는
  - noise는 줄어들겠지만
  - 근육 간 시너지, task-specific 패턴을 표현하기엔 부족할 수 있음.

3) 그래서 32–48 차원이 딱 좋은 "보틀넥" 역할

- 이 정도 크기의 feature는:
  - **근육 activation의 주요 패턴은 충분히 담고**,
  - noise/불필요한 세부사항은 자연스럽게 날려준다.
- 또한 TCN 입장에서 볼 때도
  - 입력 차원이 너무 크지 않아서
  - 파라미터 효율이 좋고
  - 실시간 추론에 부담이 적음.

✅ 2) 그럼 "32~48차원 벡터"의 실제 생김새는?

예를 들어 CNN이 **32차원**을 뽑는다고 해보자.

TCN이 받는 input은 이런 "한 줄짜리 숫자 리스트"야:

```csharp
[0.12, 1.03, -0.22, 0.89, 0.44, 0.01, -0.55,
 0.92, 1.11, -0.33, 0.08, 0.27, 0.77, -0.41,
 ...
 (총 32개)]
```

코드 복사

이 32개의 숫자가 의미하는 것:

- 특정 근육 활성 패턴
- 채널 간 상관관계
- 주파수/shape 기반 변화
- TD feature 조합에서 학습된 latent 정보

즉, **EMG의 중요한 정보를 압축한 "요약본"**이야.

6. CNN 블록을 2개로 제한한 이유

- EMG는 short-term pattern이 중요 → deep CNN 필요 없음
- 과적합을 피하면서 표현력을 확보하는 최적 지점
- ARM·Raspberry Pi에서 실시간 처리 가능
- CNN 3개부터는 성능 대비 연산 부담↑ → wearable에 부적합

요약

2-layer: 표현력은 확보하면서도, EMG 데이터 크기/실시간성/전력 고려했을 때 가장 무난한

Filters 16→32: 1층에서 기본 패턴, 2층에서 조합 패턴을 학습하기에 충분한 용량이면서 과

Kernel size 3: 근육의 짧은 temporal pattern을 잘 잡으면서, 긴 쪽은 TCN에 맡기는 역할 분

ReLU: 노이즈 많은 EMG + 임베디드 환경에 적합한, 빠르고 안정적인 기본 활성함수

Output 32–48 dims: 정보는 충분히 남기면서도, TCN과 실시간 제어에 부담이 없는 "압축 표

🔥 CNN 2-layer 전체 흐름도

```scss
Input [B, 8, 15]
     ↓
Conv1D(16 filters, k=3)
     ↓
ReLU
     ↓
Conv1D(32 filters, k=3)
     ↓
ReLU
     ↓
Output dims 32–48 (compact feature)
```

TCN (Temporal Convolutional Network)

개념

- 시계열 전용 CNN 구조
- 과거 여러 시점의 정보를 **합성곱과 dilation**을 사용해 넓게 보면서, RNN이나 LSTM처럼 순차 계산을 하지 않고 **병렬로 처리**하는 모델
- 항상 과거 데이터만 사용하도록 설계된 **인과적 (causal) 구조**라 실시간 제어에 적합

왜 쓰는가 (EMG 기준)

- 손목·손 움직임은 **수십에서 백 ms 정도의 시간 흐름**을 보고 판단해야 한다
- TCN은 이 시간 의존성을
  - LSTM처럼 잘 학습하면서도
  - CNN처럼 빠르게 계산할 수 있다
- 연산량이 적고, 지연 시간이 짧아서 의수 같은 장치에 넣기 좋다

이 모델에서 역할

- TCN은 CNN이 만들어낸 특징 시퀀스를 받아 시간에 따른 변화 흐름을 학습하고 최종적으로 "지금 이 시점에서의 관절 각도나 토크"를 연속적으로 예측하는 "시계열 해석기" 역할을 한다.

1. TCN을 1~2 Layer만 사용하는 이유

1) EMG 제어에 필요한 temporal range는 “100–200ms”

- 사람 손/손목의 근 활성 패턴은 주로 **수 ms ~ 150ms 정도의 temporal context**에서 결정됨.
- TCN이 이만큼만 커버하면 joint angle/torque estimation에는 충분함.

TCN layer가 많아지면
→ receptive field가 불필요하게 커짐
→ 오버피팅 + 연산량 증가

2) TCN을 깊게 만들면 오히려 과적합 + 실시간성 악화

- dilation=1,2,4… 이런 식으로 쌓을수록
  연산량 증가
  latency 증가
  과적합 증가
- wearable 의수에서 “딜레이 증가”는 바로 체감됨
  (40ms 넘기면 이미 부자연스럽다고 평가됨)

→ **1~2 layer가 “정확도 + 실시간성”의 최적 지점**

3) BiLSTM 수준 표현력 확보

연구적으로 확인된 사실:
TCN 1~2 layer = LSTM 1~2 layer와 비슷한 temporal modeling 성능

그런데
TCN은 CNN처럼 병렬 계산 → 훨씬 빠름.

→ **실시간 wearable 디바이스에서는 깊은 TCN이 의미 없음**

2. **Hidden Channels를 32~64로 제한한 이유**

1) Hidden channel = “시간 패턴을 표현하는 용량(capacity)”

Hidden channel이 크면
더 복잡한 시계열을 표현할 수 있음.

2) 32~64가 EMG 제어에 필요한 적정치

- EMG는 고차원 시계열이지만
  근 활성 시그니처 자체는 압축된 형태임
  (MUAP 폼도 제한적, synergy 패턴도 제한적)
- CNN에서 이미 compact representation(32~48dim)을 만들어서
  TCN에는 너무 큰 용량이 필요하지 않음.

3) 64 이상은 과적합·전력 소모·속도 저하

Hidden channel 128, 256 수준으로 가면:

- parameter 폭증
- 오버 피팅
- 연산량 증가 → ARM보드 속도 저하
- 실시간성 악화
- 배터리 소모 증가
- 발열 증가 → 착용감 악화

3. Dilation = 1, 2만 사용하는 이유

1) EMG는 “짧은 temporal gap 구조”

근육의 시계열 특징은

- MUAP 사이 간격
- activation rise time
- relaxation 과정
- 50~150ms 근육 burst pattern
- EMG의 짧은 패턴을 건너뛰어 버림
- 의미 있는 정보 손실
- noisy한 distant dependency를 과하게 배우면서 과적합 유발

✔ 2) dilation 1,2 조합이면 receptive field 충분

TCN receptive field 공식:

```ini
R = (kernel_size - 1) * sum(dilations) + 1
```

kernel=3 기준:

```nginx
Layer1 dilation=1 → RF = 2
Layer2 dilation=2 → RF = 2 + 4 = 6
```

여기서

CNN window(100ms) × TCN receptive field(6 patch)

→ **약 100~200ms의 효과적 temporal range**

즉, EMG에 필요한 temporal context를 정확히 커버함.

3) dilations 4, 8을 쓰면 “말도 안되게 멀리 있는 신호”를 보게 됨

전극 위치가 미묘하게 바뀌거나
근피로가 돼도
이런 큰 dilation은 noise에 과하게 예민해지고 성능이 떨어짐.

4. Kernel size = 3을 선택한 이유

1) k=3은 “근육의 가장 기본적인 변화 패턴”을 포착

- spike 모양
- MUAP waveform peak
- 급격한 활성 변화
- 작은 activation shift

2) k=3은 연산량 대비 성능 효율이 가장 좋음

- k=1: temporal structure 못 봄
- k=5,7: 연산량 증가 + 과적합↑
- k=3:
  - 충분히 local
  - receptive field 증가에 유리
  - 연산량 적음
  - 실시간 제어에 우수

3) 역할 분담의 관점에서도 최적

- “long temporal dependency”는 TCN dilation이 담당
- CNN은 local pattern 담당
- TCN kernel은 short temporal aggregation 담당

따라서 TCN kernel을 크게 만들 필요 없음.

✅ 1) FC 레이어는 어떻게 작동하는가?

Input → TD → CNN → TCN → **FC** → Output

관절각도 예측이 1개라고 가정하면:

FC는 이렇게 생김:

```ini
angle = w1*h1 + w2*h2 + ... + w32*h32 + b
```

즉,

- **각 hidden feature(h1~h32)**에
- **학습된 가중치(w1~w32)**를 곱해서 더하고
- **bias(b)**를 더한 후
- 그 합을 "관절각도"로 해석함.

이건 선형 변환이며, 신경망의 마지막 단계에서 항상 사용됨.

✅ 1) 그럼 FC 가중치는 어디서, 어떻게 학습되는가?

훈련 루프(Training loop)에서 발생한다:

Step 1) CNN/TCN이 EMG 데이터를 통과시켜서

```ini
hidden_feature = [h1, h2, ..., h32]
```

이런 feature vector를 만들어냄.

Step 2) FC layer는 처음에는 random weight를 가진 상태임

```css
w1, w2, ..., w32  ← 처음엔 랜덤 값
bias b ← 랜덤 값
```

Step 3) FC 출력 angle_pred를 계산

```ini
angle_pred = w1*h1 + w2*h2 + ... + w32*h32 + b
```

Step 4) 실제 관절각도(angle_gt)와 비교해 Loss 계산

예: MSELoss()

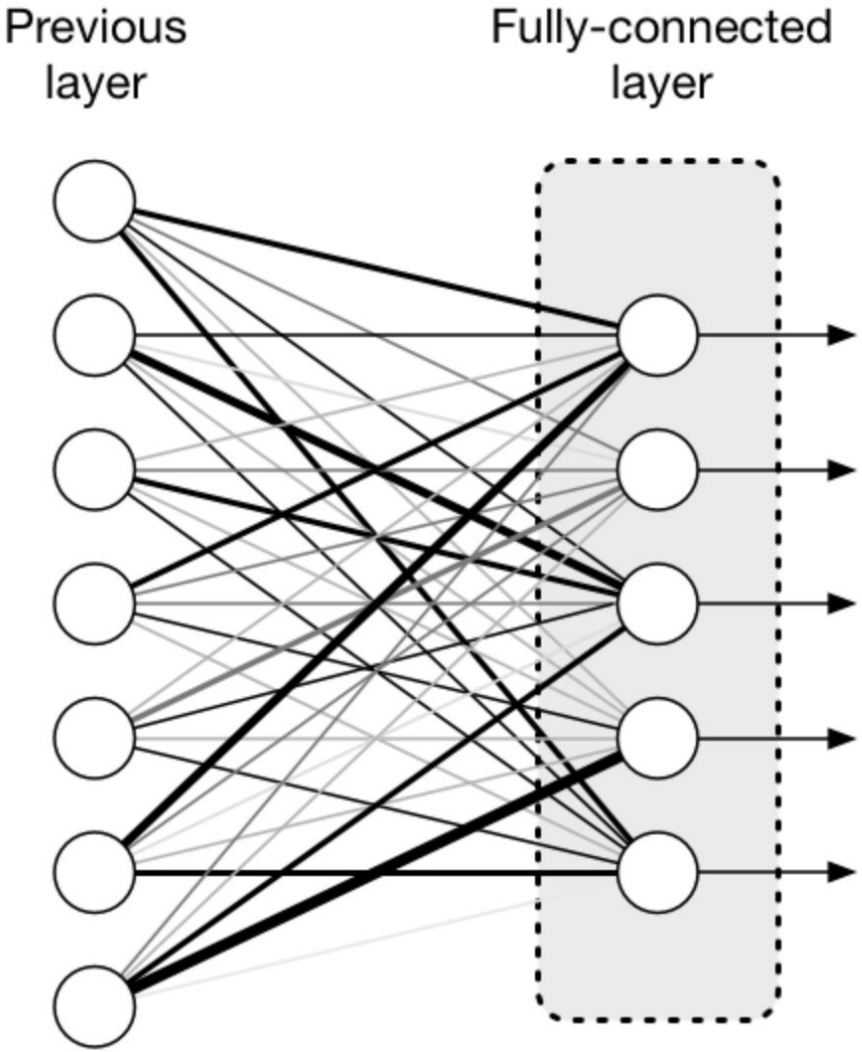
```ini
loss = (angle_pred - angle_gt)^2
```

Step 5) Backpropagation이 시작됨

이때 optimizer(Adam, SGD 등)가 다음을 수행한다:

```css
w1 := w1 - lr * d(loss)/d(w1)
w2 := w2 - lr * d(loss)/d(w2)
...
w32 := w32 - lr * d(loss)/d(w32)
b  := b  - lr * d(loss)/d(b)
```

즉,



h1 h2 h3 ... h32

\ \ \ \

\ \ \ \

\ \ \ \

w1 w2 w3 ... w32 ← weight

(학습됨)

\ | /

\ | /

(+ b)

↓

Output (angle)

```
y = [
    W_flex,     # Wrist flexion/extension
    W_dev,      # Wrist radial/ulnar deviation

    T_CMC_flex, # Thumb CMC flexion
    T_MCP_flex, # Thumb MCP flexion
    T_IP_flex,  # Thumb IP flexion

    I_MCP_flex, # Index MCP flexion
    I_PIP_flex, # Index PIP flexion
    I_DIP_flex, # Index DIP flexion

    M_MCP_flex, # Middle MCP flexion
    M_PIP_flex, # Middle PIP flexion
    M_DIP_flex, # Middle DIP flexion

    R_MCP_flex, # Ring MCP flexion
    R_PIP_flex, # Ring PIP flexion
    R_DIP_flex, # Ring DIP flexion

    L_MCP_flex, # Little MCP flexion
    L_PIP_flex, # Little PIP flexion
    L_DIP_flex  # Little DIP flexion
]
```

손목 2개: 굽힘/폄, 편위

엄지 3개: CMC, MCP, IP 굽힘

손가락 4개 × 3 DOF = 12개: MCP, PIP, DIP 굽힘

총 17개 각도