**Algoritmos y modelos utilizados para el reconocimiento y traducción de lenguaje de Señas (ASL fingerspelling)**

1. **Naturaleza del problema**

Cada registro de los archivos Parquet (ver docmento de exploración) representa:

* Secuencia temporal de frames pertenecientes a una secuencia ID (palabra o conjunto de letras).
* En cada frame: coordenadas (x,y,z) de puntos clave del cuerpo, rostro y manos.
* El objetivo es predecir una letra o secuencia de letras (fingerspelling) a partir de esos puntos.

El problema es especialmente de **aprendizaje supervisado secuencial[[1]](#footnote-1) (sequence models)**, similar a problemas de:

* Reconocimiento de escritura a mano en línea (trayectorias de lapicero).
* Reconocimiento de gestos humanos a partir. De sensores.
* Modelado de series de tiempo multivariadas.

1. **Estructura general del pipeline**
2. Lectura y preprocesamiento de las secuencias.
   * Agrupar por sequence\_id.
   * Normalizar coordenadas
   * Padding/truncation de secuencias a longitud fija.
3. Extracción de características (feature extraction): algunas investigaciones usan PCA o Autoencoders para reducir dimensionalidad.

**Autoencoders[[2]](#footnote-2)**

* + Es un tipo de arquitectura de red neuronal diseñada para comprimir (encode) data de input a sus componentes principales y luego reconstruir (decode) el input original de su representación comprimida.
  + Usa machine learning no supervisado.
  + Están entrenados para descubrir *latent variables* en la data de input: variables ocultas o random que, aunque no son directamente observables, afectan fundamentalmente la manera en que los datos están distribuidos.
  + La mayoría de itpos de autoencoders se usan en tareas de IA, relacionadas con feature extraction como compresión de datos, image denoising, detección de anomalías y reconocimiento facial.
  + **Diferencia con encoder-decoder**: los encoder-decoder son usados en modelos como CNN, RNN, entre otros. Da diferencia principal es que en los encoder-decoder, el input es distinto al output. En los autoencoders, el output describe la misam data.

🡪 Los autoencoders son un subset específico de arquitecturas encoder-decoder que son entrenadas con aprendizaje no supervisado para *reconstruir su propia data de input*.

* + **Ventaja sobre PCA:** los autoencoder capturan correlacions *no lineales*. Las funciones de acitivación normalmente no son lineales.
  + Los hiperparámetros del modelo: tamaño del código
  + **Tipos de autoencoder[[3]](#footnote-3)**
    - Denoising autoencoder: para inputs noisy o corruptos. Aprende a quitar el ruido y ayuda a reconstruir data línea.
    - Sparse autoencoder: contiene más unidades ocultas que features de input, solo permite que algunas neuronas estén activas simultáneamente.
    - Variational autoencoder: asume cosas sobre la distribución de probabilidad de la data y trae de basarse en eso para hacer mejores predicciones.
    - Autoencoder correlacional: usa CNNs para extraer features.

**VER CÓDIGO EN LINK DE GEEKSFORGEEKS (footnote 3)**

1. Modelado secuencial (core del modelo):
   * Modelos recurrentes (LSTM, GRU).
   * Transformers temporales
   * Modelos 1D-CNN o híbridos CNN + RNN
   * Sequence-to-Sequence (Seq2Seq)
2. Decodificación: softmax para clasificación de letras, CTC o attention si es secuencia a secuencia.
3. **Algoritmos principales usados en este tipo de datos**
   1. RNN, LSTM y GRU
   * Son los modelos base más utilizados en tareas de secuencias de keypoints.
   * Cada timestep contiene un vector de features concatenando las coordenadas de todos los puntos (por ejemplo, 543 puntos x 3 coordenadas = 1629 dimensiones).
   * Las LSTM pueden capturar dependencias de movimiento en el tiempo.
   * Algunas variantes:
     + **BiLSTM** (Bidirectional LSTM): ve la secuencia completa para aprender dependencias hacia adelante y atrás.
     + **Stacked LSTM:** múltiples capas para capturar distintos niveles de abstracción.
   1. Temporal Convolutional Networks (TCN)
   * Alternativa a LSTM. Usan convoluciones 1D con dilatación para capturar dependencias a largo plazo sin recurrencias.
   * Ventaja: paralelización más eficiente.
   * Útiles cuando las secuencias son muy largas.
   1. Transformers para series temporales[[4]](#footnote-4)
   * Cada frame (vector de coordenadas) se trata como un “token”.
   * El modelo aprende atención temporal, descubriendo qué frames son más relevantes para cada letra.
   * Modelos destacados:
     + **TimeSformer** (Bertasius et al., 2021) – pero aplicado a features numéricos en lugar de imágenes.
     + **Temporal Transformer Encoder (TTE)** – simplificación de ViT para secuencias numéricas.
     + **Sign Language Transformers (SLT)** – versión adaptada para datos de poses.
   * En datasets de ASL con keypoints, suelen superar a las LSTM en precisión cuando el dataset es grande.
   1. Graph Neural Networks (GNN) o Spatial-Temporal Graph Convolutional Networks (ST-GCN)
   * Como los puntos de la mano y cuerpo tienen estructura de grafo (conexiones entre articulaciones), algunos modelos aprovechan: cada punto clave es un nodo, las conexiones anatómicas son aristas.
   * El modelo aprende cómo se mueven las articulaciones relativas en el tiempo.
   * Modelos: **ST-GCN** (Yan et al., 2018), **MSG3D, Pose-GCN.**
   * Son especialmente potentes para capturar la forma de la mano sin perder información estructural.

**Arquitectura[[5]](#footnote-5)**

* + Capa convolucional: hace la convolución para cada nodo para aprender sus conexiones. Usa la información de los vecinos de los nodo, hace una agregación para actualizar los pesos.
  + Activación no lineal
  + Capa de output lineal.

**Casos de uso[[6]](#footnote-6)**

* + NLP: para modelar relaciones entre las palabras u oraciones en documentos.
  + Computer vision: segmentacoines, de imágenes (modela relaciones entre pixeles), detección de objetos.
  + Bioinformática: análisis de secuencia de genes, descubrimiento de medicinas, …

**MÁS INFORMACIÓN:**

* + <https://distill.pub/2021/gnn-intro/>,
  + <https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial>
  + <https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial>

1. Híbridos (CNN 1D + RNN/Transformer/GCN)
   * Muchos modelos ganadores de competiciones (como la de ASL Fingerspelling) combinan: **1D CNN** para traducir ruido y extraer patrones locales y **LSTM o Transformer** para modelar la temporalidad global.

Comparación[[7]](#footnote-7):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parámetro** | **RNN** | **LSTM** | **GRU** | **Transformers** |
| **Arquitectura** | Estructura simple con loops | Células de memoria con gates de input, forget y output. | Combina input y forget gates en el update gate, menos parámetros | Usa mecanismos de atención sin recurrencia. |
| **Secuencias largas** | No mantiene dependencias a largo plazo por vanishing gradients. | Bueno para capturar dependencias de largo plazo. | Mejor que las RNN pero un poco peor que LSTM para dependencias de largo plazo. | Maneja secuencias largas efectivamente, usando mecanismos de autoatención. |
| **Tiempo de entrenamiento** | Rápido pero menos acertado con data compleja. | Lento por la cantidad de operaciones de memoria. | Más rápido que las LSTM pero más lento que las RNN. | Necesita poder de computación pero permite entrenamiento paralelo. |
| **Uso de memoria** | Bajos requerimientos de memoria. | Consumo alto de memoria por complejidad de arquitectura. | Bajo consumo de memoria comparado a LSTM pero alto comparado con RNN. | Alto consumo de memoria por el multi-head attention y feed-forward layers. |
| **Cantidad de parámetros** | Baja cantidad de parámetros en general. | Más parámetros que las RNN por la cantidad de gates y células de memoria. | Menos parámetros que las LSTMs por la estructura simplificada. | Gran cantidad de parámetros por las capas de multi-head attention. |
| **Facilidad de entrenamiento** | Suceptible al vanishing gradient problem, no bueno para secuencias largas. | Más fácil de entrenar para secuencias largas por manejo bueno de gradiente. | Más simple que las LSTMs y más fáciles de entrenar que las RNN. | Necesita poder computacional alto y GPUs. |
| **Casos de uso** | Útil para secuencias simples como precios. | Ideal para series temporales, generación de texto y tareas que necesitan dependencias a largo plazo. | Aplicaciones similares a las LSTM pero preferido cuando la eficiencia computacional es importante. | Se usa para tareas de NLP como traducción, sumarización, visión por computadora y procesamiento de lenguaje de voz. |
| **Paralelismo** | Limitado. | Mismas limitaciones que las RNNs; el procesamiento secuencial restringe el paralelismo. | Mismas limitaciones que las RNNs; el procesamiento secuencial restringe el paralelismo. | Alto paralelismo permitido por el mecanismo de atención y el diseño no secuencial. |
| **Performance con secuencias largas** | Bajo | Bueno | Moderado a bueno | Excelente |

1. **Entrenamiento y técnicas complementarias** 
   1. Data augmentation: añadir ruido gaussiano a las coordenadas.
   2. Normalización: estandarizar por z-score, centrar en la muñeca, en el torso o en el punto central.
   3. Reducir dimensiones con autoencoders.
   4. Batching: entrenar por secuencia (sequence\_id) con generadores que carguen cada Parquet secuencialmente.
   5. Modelo a utilizar: Transformers (si no funciona bien, el accuracy es muy malo, evaluar la integración de GNN al modelo para relacionar los puentos entre ellos[[8]](#footnote-8)).
   6. Loss functions: Categoriacl Crossentropy (si se predice una letra), CTC Loss (si se predice toda la palabra).
2. A screenshot of a white table with black text

   AI-generated content may be incorrect.**Modelos de referencia en papers y competiciones**
3. **Conclusión**

Para nuestros Parquets con coordenadas MediaPipe

* Base recomendada: CNN1D + BiLSTM o Transformer Encoder.
* Alternativa avanzada: ST-GCN o PoseFormer (si se tiene suficiente poder de cómputo)
* Entrenamiento en batch: se puede procesar Parquet por Parquet, ya que cada archivo contiene miles de secuencias (no se necesita todo en RAM).

**Modelo pre hecho: SignBERT – transformers con preentrenamiento multimodal**.

1. <https://medium.com/machine-learning-basics/sequence-modelling-b2cdf244c233> , <https://analyticsindiamag.com/ai-trends/a-tutorial-on-sequential-machine-learning/> [↑](#footnote-ref-1)
2. <https://www.ibm.com/think/topics/autoencoder> [↑](#footnote-ref-2)
3. <https://www.geeksforgeeks.org/machine-learning/auto-encoders/> [↑](#footnote-ref-3)
4. <https://arxiv.org/pdf/2003.13830> [↑](#footnote-ref-4)
5. <https://www.ibm.com/think/topics/graph-neural-network> [↑](#footnote-ref-5)
6. <https://www.ibm.com/think/topics/graph-neural-network> [↑](#footnote-ref-6)
7. <https://www.geeksforgeeks.org/deep-learning/rnn-vs-lstm-vs-gru-vs-transformers/> [↑](#footnote-ref-7)
8. <https://www.signitysolutions.com/tech-insights/combining-transformers-and-graph-neural-networks> [↑](#footnote-ref-8)