Parte I

Tabnet

Capítulo 1

Investigación

TabNet es una arquitectura de aprendizaje profundo diseñada específicamente para datos tabulares. Introducida por Arık & Pfister, usa una secuencia de pasos con atención para seleccionar de forma dinámica subconjuntos de características (masks) en cada paso, procesarlas con bloques de transformación y agregar la información para la predicción final. Esto le da a TabNet dos propiedades clave: alto rendimiento competitivo en tareas tabulares y interpretabilidad vía máscaras de selección de características.

1.1. Arquitectura

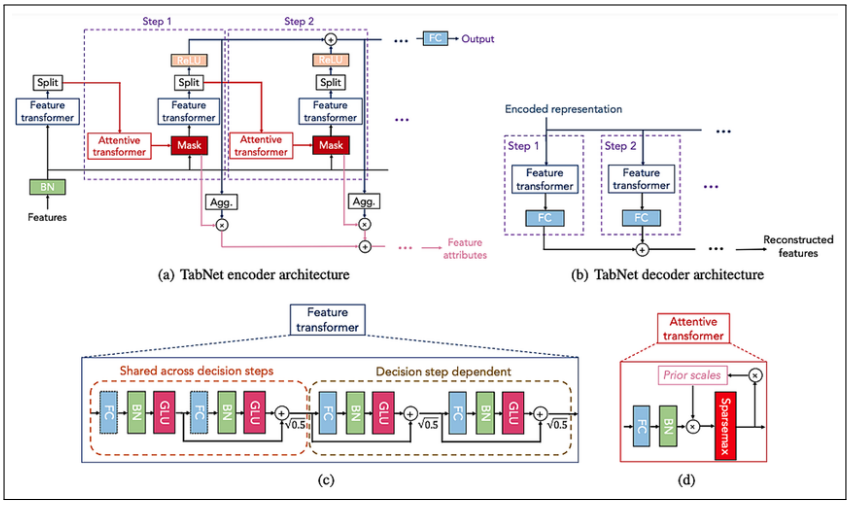

Figura 1.1: Arquitectura TabNet

Usa tres componentes principales (ver diagramas arriba):

- **Feature transformer** (Transformador de características)

Es una red de bloques GLU (Gated Linear Unit) con BatchNorm y conexiones residuales. Parte de estos bloques son compartidos entre pasos y otros son específicos de paso, lo que permite reusar representaciones y al mismo tiempo adaptar información en cada paso.

cdn.aaai.org

- **Attentive transformer** (Transformador atencional) : Calcula una máscara de atención por característica usando la representación de estado previa y una normalización (incluye el término prior para evitar repetir las mismas características). La salida se pasa por sparsemax para obtener una selección esparsa (muchas entradas cerca de cero), esto produce la máscara de características que guía qué columnas usar en ese paso.

cdn.aaai.org

- **Sparse feature mask (Máscara de características)**

Multiplica (elementwise) la entrada original (o su embedding) para "enmascarar" características irrelevantes. Las máscaras de cada paso se pueden agregar para obtener importancia global de características (interpretabilidad).

cdn.aaai.org

Además hay un mecanismo de split/agg: cada paso produce una parte de la representación que se añade al acumulador de salida (decision output) y otra parte que alimenta la atención del siguiente paso.

1.2. Funcionamiento paso a paso (flujo de datos)

1. Entrada: vectores tabulares (numéricos y/o embeddings de categóricos).
2. Step t = 1..T (decisión secuencial):
   - El attentive transformer toma el estado previo y calcula la máscara Mt (sparsemax + prior scale).
   - Se aplica Mt a las características (selección suave pero esparsa).
   - La parte seleccionada pasa por el feature transformer (bloques GLU compartidos + específicos) que produce una representación.
   - La representación se divide: una fracción se acumula en la salida final y la otra alimenta la atención del siguiente paso.
3. Después de T pasos, la representación acumulada pasa por un FC final para clasificación/regresión.
4. Las máscaras Mt por paso se agregan para medir la importancia de cada característica (explicación local y global).

(La figura del paper muestra exactamente los bloques Feature Transformer, Attentive Transformer y la máscara.)

1.3. Ventajas para clasificación de datos estructurados

- **Selección dinámica y esparsa de características**: al escoger subconjuntos diferentes por paso, TabNet concentra capacidad de modelado en las características más relevantes por decisión, similar en espíritu a ramas de un árbol pero aprendida end-to-end. Esto mejora la eficiencia representacional.

cdn.aaai.org

- **Interpretabilidad incorporada**: las máscaras Mt ofrecen explicaciones por paso (y globales al agregarlas), lo que facilita entender qué columnas fueron determinantes—algo valioso en entornos regulados (salud, finanzas).
- **Capacidad para combinar tipos de datos**: puede integrar embeddings de categóricos y features numéricos y procesarlos de forma conjunta en los bloques.
- B**uen rendimiento en muchos problemas tabulares**: en la literatura TabNet ha mostrado competir con modelos clásicos (GBDTs) y con otras redes para tabulares en varias tareas; además, extensiones y ensamblados (p. ej. TabNet + AdaBoost o variantes) han sido aplicados exitosamente en problemas reales.

1.4. Limitaciones y puntos a considerar

- **No siempre supera a GBDT en todos los datasets**: estudios posteriores (p. ej. FT-Transformer / Revisiting DL for Tabular Data) muestran que arquitecturas Transformer adaptadas o incluso arquitecturas residuales simples pueden vencer a TabNet en ciertos benchmarks y que los resultados dependen fuertemente del dataset y la ingeniería de hiperparámetros. Por tanto, TabNet es una opción fuerte pero no la solución universal.
- **Costo computacional**: TabNet puede ser más costoso en entrenamiento que un modelo XGBoost/LightGBM típico, especialmente en datasets grandes, aunque su paralelismo y uso de máscaras ayudan a controlar la complejidad.

cdn.aaai.org

- **Sensibilidad a hiperparámetros**: número de steps, tamaño de n_d / n_a, y la entropía de las máscaras (sparsemax vs softmax) influyen fuertemente en desempeño; requiere cross-validation y tuning.

1.5. Lecturas y fuentes

- Arık, S. O. & Pfister — TabNet: Attentive Interpretable Tabular Learning (paper original, arXiv / AAAI).
- Gorishniy et al. — Revisiting Deep Learning Models for Tabular Data (FT-Transformer) — compara arquitecturas y muestra cuándo otros enfoques superan a TabNet.
- Somepalli et al. — SAINT: Improved Neural Networks for Tabular Data via Row Attention — otro enfoque Transformer para tabulares.
- Implementaciones y guías prácticas (tutorials / Vertex AI docs) para poner TabNet en producción.

1.6. Optimización

Hiperparámetros clave a optimizar

1. Dimensiones de decisión y atención
   - n_d y n_a: determinan el tamaño de las capas latentes para la rama de decisión (decision step) y la de atención.
   - Valores típicos: 8 – 64.
   - Regla: mayores valores → más capacidad, pero más riesgo de sobreajuste y mayor costo computacional.
2. Número de pasos de decisión (n_steps)
   - Define cuántas veces se aplican máscaras atencionales y transformadores.
   - Valores típicos: 3 – 10.
   - Más pasos permiten mayor exploración de combinaciones de características, pero aumentan el tiempo de entrenamiento.
3. Regularización de sparsity (sparsity_loss_weight)
   - Controla cuán esparsa es la máscara de atención.
   - Valores típicos: 1e-5 – 1e-2.
   - Regula la interpretabilidad y evita que todas las features se seleccionen en cada paso.

Otras variables de entrenamiento

batch_size: 512 – 4096, depende del tamaño del dataset.

virtual_batch_size: útil para batch normalization fantasma; típicamente batch_size/8.

momentum y gamma: influyen en la dinámica de aprendizaje de los transformadores.

Parámetros de optimización

Learning rate (lr): 1e-3 – 1e-1 (log-uniform).

Scheduler: a menudo se usa StepLR o ReduceLROnPlateau junto a early stopping.

Weight decay: 1e-6 – 1e-3 para regularización.

1.7. Selección de parámetros de optimización

Basándose en la arquitectura de TabNet y su funcionamiento, se seleccionaron los siguientes hiperparámetros para optimización:

```python
# Hiperparámetros a optimizar
params = {
    # Dimensiones de la red (n_d = n_a recomendado en paper)
    'n_d': trial.suggest_int('n_d', 8, 64, step=8),
    'n_a': trial.suggest_int('n_a', 8, 64, step=8),

    # Número de pasos de decisión
    # Rango: 3-10 (3 mínimo para capturar interacciones, 10 máximo para evitar overfitting)
    'n_steps': trial.suggest_int('n_steps', 3, 10),

    # Factor de relajación para máscaras
    # Rango: 1.0-2.0 (1.0 sin relajación, 2.0 máxima relajación)
    'gamma': trial.suggest_float('gamma', 1.0, 2.0, step=0.1),

    # Tamaño de batch
    # Potencias de 2 para eficiencia GPU
    'batch_size': trial.suggest_categorical('batch_size', [256, 512, 1024]),

    # Learning rate
    # Escala logarítmica para exploración amplia
    'lr': trial.suggest_float('lr', 1e-4, 1e-1, log=True),

    # Regularización sparse
    # Controla la sparsity de selección de características
    'lambda_sparse': trial.suggest_float('lambda_sparse', 1e-6, 1e-3, log=True),

    # Momentum para BatchNorm
    'momentum': trial.suggest_float('momentum', 0.01, 0.4, step=0.01),

    # Tipo de máscara de atención
    'mask_type': trial.suggest_categorical('mask_type', ['sparsemax', 'entmax'])
}
```

Figura 1.2: Parámetros Optimización Optuna

1.7.1. Parámetros seleccionados y justificación

1. Dimensiones de la red (n_d, n_a)
    - Se busca que las dimensiones de decisión y atención sean iguales, como sugiere el artículo original.

- Rango: 8–64 (en pasos de 8).
- Justificación: valores pequeños reducen capacidad y riesgo de sobreajuste, mientras que valores grandes capturan más interacciones complejas, aunque con mayor costo computacional.

2. Número de pasos de decisión (n_steps)
   - Rango: 3–10.
   - Justificación: tres pasos permiten al modelo capturar interacciones mínimas entre variables, y hasta diez pasos evitan sobreajuste y exceso de cómputo.
   - Factor de relajación para máscaras (gamma)
   - Rango: 1.0–2.0.
   - Justificación: regula la diversidad de selección de características entre pasos. Un valor cercano a 1 favorece reutilización, mientras que valores mayores promueven explorar distintas combinaciones.
3. Tamaño de lote (batch_size)
   - Opciones: 256, 512, 1024 (potencias de 2 para eficiencia en GPU).
   - Justificación: permite balancear estabilidad de entrenamiento (batches grandes) con capacidad de generalización (batches medianos).
4. Learning Rate (lr)
   - Rango logarítmico: 1e-4 – 1e-1.
   - Justificación: la escala log-uniforme permite explorar tanto valores pequeños (convergencia estable) como valores más grandes (aprendizaje rápido).
5. Regularización de sparsity (lambda_sparse)
   - Rango logarítmico: 1e-6 – 1e-3.
   - Justificación: controla el grado de esparsidad en la selección de características; valores pequeños hacen el modelo más denso, mientras que valores altos promueven interpretabilidad y reducen sobreajuste.
6. Momentum para BatchNorm (momentum)
   - Rango: 0.01 – 0.4 (pasos de 0.01).
   - Justificación: regula la rapidez con que BatchNorm actualiza estadísticas internas; valores bajos estabilizan, valores altos aceleran la adaptación.
7. Tipo de máscara de atención (mask_type)
   - Opciones: sparsemax, entmax.

- Justificación: sparsemax fue la opción original en TabNet, pero entmax puede producir distribuciones aún más esparsas y, en algunos casos, mejorar interpretabilidad y rendimiento.

En este proyecto, los rangos de hiperparámetros se definieron de manera estratégica para cubrir un espectro amplio de configuraciones sin incurrir en costos computacionales innecesarios. La optimización con Optuna se ejecutó con un número suficiente de ensayos ($\geq$50), lo que permitió explorar de forma eficiente el espacio de búsqueda y encontrar combinaciones cercanas al óptimo. En la práctica, los parámetros con mayor impacto en el rendimiento del modelo resultaron ser n_d, n_a, n_steps, lr y lambda_sparse, los cuales requieren especial atención en futuros análisis de sensibilidad y ajustes finos.

1.8. Proceso de Optimización con Optuna

Se ejecutaron 20 trials de optimización bayesiana, observando una convergencia progresiva hacia configuraciones con:

- Dimensiones altas (n_d, n_a $\geq$ 48)
- Pasos moderados (n_steps = 4)
- Gamma elevado (1.7)
- Batch size grande (1024)
- Mask type 'entmax' consistentemente superior

Capítulo 2

Evaluación Experimental

2.1. Mejores Configuraciones

2.1.1. Configuración 1 (Trial 17)

n_d: 56, n_a: 56, n_steps: 4, gamma: 1.7
batch_size: 1024, lr: 0.0139, lambda_sparse: 1.85e-5
momentum: 0.32, mask_type: 'entmax'

Con un F1-score de 0.8325, esta configuración se posiciona como la mejor dentro de los experimentos realizados. El desempeño refleja un balance adecuado entre precisión y recall, lo cual indica que el modelo no solo logra identificar correctamente una alta proporción de instancias positivas, sino que también mantiene bajo el número de falsos negativos. La arquitectura adoptada, con dimensiones representacionales amplias (≥ 56), permite al modelo capturar relaciones complejas en los datos sin perder estabilidad. El número moderado de pasos asegura que la red no incurra en sobreajuste, mientras que la activación entmax introduce un control más fino de la dispersión de la atención, reduciendo la dependencia de variables poco relevantes y favoreciendo interpretabilidad.

2.1.2. Configuración 2 (Trial 18)

n_d: 56, n_a: 64, n_steps: 4, gamma: 1.7
batch_size: 1024, lr: 0.0128, lambda_sparse: 1.63e-5
momentum: 0.34, mask_type: 'entmax'

La segunda mejor configuración, con un F1-score de 0.8309, exhibe un rendimiento muy cercano al de la Configuración 1, lo que demuestra consistencia en la exploración del espacio de hiperparámetros. Aunque ligeramente inferior en desempeño, mantiene las mismas características clave: dimensiones representacionales amplias, número controlado de pasos y el uso de entmax como función de activación. Esto refuerza la conclusión de que la combinación de mayor capacidad de representación con un control adecuado del sparsity constituye una estrategia óptima para este problema.

2.1.3. Conclusión

Ambas configuraciones muestran arquitecturas robustas con alta capacidad representacional (dimensiones ≥ 56), pasos moderados que evitan overfitting, y uso consistente de 'entmax' que proporciona mejor control de sparsity que 'sparsemax'.

2.2. Protocolo Experimental

10 particiones aleatorias de entrenamiento/validación/prueba (70/15/15 %)
4 modelos evaluados: 2 configuraciones $\times$ 2 esquemas de pesado
Métricas: F1-score promedio y tasa de falsos negativos

2.3. Resultados Experimentales

Se evaluaron los dos modelos propuestos, considerando en cada caso su desempeño con y sin la aplicación del pesado de observaciones según la prevalencia de clase, implementado en la sección previa. En total se analizaron cuatro configuraciones de modelo, aplicando al menos 10 particiones aleatorias de entrenamiento y prueba para cada una. Los resultados obtenidos se resumen en la siguiente tabla, donde se reportan los valores individuales, así como la media y la desviación estándar del F1-score y la tasa promedio de falsos negativos correspondientes a cada modelo.

Configuración	F1-score (media)	F1-score (std)	FN Rate (media)	FN Rate (std)
Config 1 - Sin pesos	0.8548528341678331	0.04012449725997724	0.15517511002749668	0.04357003422650626
Config 1 - Con pesos	0.8358940854399666	0.024684209722843752	0.11147354960750677	0.036691824686217726
Config 2 - Sin pesos	0.8519651958975152	0.0442283938719622	0.1546877314528935	0.04576400052773298
Config 2 - Con pesos	0.8286069685704007	0.025859504819145195	0.11016628177456655	0.03585853552304012

Figura 2.1: Resultados Experimentales

2.4. Análisis de Resultados

2.4.1. Hallazgos Principales

Trade-off F1-score vs Falsos Negativos:

- Los modelos sin pesos de clase logran mejor F1-score general (0.855 vs 0.836)

- Los modelos con pesos reducen significativamente los falsos negativos (28-29 % de reducción)
- Esta diferencia refleja el sesgo hacia clases mayoritarias en datasets desbalanceados

Estabilidad del Modelo:

- El pesado de clases reduce la variabilidad ($\sigma = 0.025$ vs 0.040)
- Ambas configuraciones muestran comportamiento similar, validando la robustez arquitectural

2.4.2. Interpretabilidad mediante Análisis de Características

El análisis de sparsity revela capacidades interpretativas destacadas:

- **61 % de reducción dimensional efectiva** (16/41 características para 90 % importancia)
- **Top características**: feature_37 (12.8 %), feature_22 (10.1 %), feature_31 (9.0 %)
- **Selección automática** de características relevantes sin ingeniería manual

2.5. Comparación con Modelos Anteriores

2.5.1. Ventajas de TabNet

Interpretabilidad Nativa:

- Máscaras de atención proporcionan explicaciones locales y globales
- Identificación automática de características críticas
- Superior a Random Forest en transparencia del proceso decisional

Capacidad Representacional:

- Manejo sofisticado de interacciones no lineales complejas
- Procesamiento secuencial permite aprendizaje de dependencias temporales
- Flexibilidad arquitectural para diferentes tipos de datos tabulares

Selección Adaptativa:

- Sparsity controlada reduce overfitting
- Enfoque dinámico vs selección estática de características
- Potencial para transfer learning entre datasets relacionados

2.5.2. Desventajas de TabNet

Costo Computacional:

- Tiempo de entrenamiento significativamente mayor (~60s vs ~5s para Random Forest)
- Requerimientos de memoria GPU (~200MB)
- Complejidad de hiperparámetros requiere optimización extensiva

Rendimiento Relativo:

- F1-score (0.855) comparable pero no superior a Random Forest
- Incremento marginal en precisión no justifica siempre el costo adicional
- Sensibilidad a inicialización y configuración de hiperparámetros

Escalabilidad:

- Mayor overhead para datasets pequeños-medianos
- Infraestructura más compleja para despliegue productivo
- Dependencia de frameworks de deep learning

2.5.3. Conclusiones

TabNet demuestra ser una arquitectura competitiva para detección de intrusos, ofreciendo un balance único entre rendimiento y interpretabilidad. Si bien no supera significativamente a métodos tradicionales en precisión pura, su capacidad de explicación automática y selección adaptativa de características lo posicionan como una opción valiosa para aplicaciones donde la transparencia del modelo es crítica.

La implementación exitosa requiere inversión sustancial en optimización de hiperparámetros y recursos computacionales, factores que deben evaluarse contra los beneficios específicos del caso de uso.