Instructivo Preparado por:
Paola Andrea Sánchez-Sanchez, PhD.

Procesamiento de Lenguaje Natural – PLN en Deep Learning

Introducción

El Procesamiento del Lenguaje Natural (PLN) es el área de la Inteligencia Artificial (IA) dedicada a dotar a las máquinas de la capacidad de comprender, interpretar y generar lenguaje humano. Su historia es una trayectoria marcada por cambios de paradigma. Los primeros sistemas de PLN, desde mediados del siglo XX hasta los años 80, se basaban predominantemente en reglas lingüísticas diseñadas manualmente y complejos diccionarios, un enfoque que demostró ser frágil e ineficaz ante la vasta complejidad y ambigüedad del lenguaje real.

A partir de finales de los 80, una revolución estadística comenzó a tomar forma, utilizando modelos probabilísticos y de Machine Learning tradicional para manejar la incertidumbre y la variabilidad lingüística. Sin embargo, el verdadero punto de inflexión llegó con el auge del Deep Learning en la década de 2010. La capacidad de las redes neuronales profundas para aprender representaciones jerárquicas y no lineales a partir de datos masivos superó rápidamente a los enfoques anteriores.

El desarrollo de arquitecturas como las Long Short-Term Memory (LSTM) por Hochreiter y Schmidhuber (1997) fue crucial para manejar las dependencias de largo alcance en el lenguaje, sentando las bases para el rendimiento que vemos hoy. El Deep Learning permitió la transición de embeddings estáticos a representaciones contextuales dinámicas, marcando la era actual.

El presente documento tiene como objetivo proporcionar una guía integral sobre los principios, arquitecturas y aplicaciones del Procesamiento del Lenguaje Natural con Deep Learning. Se busca que los lectores comprendan tanto los fundamentos teóricos de la representación del lenguaje, las limitaciones superadas por los mecanismos de gating y atención, como el paradigma actual dominado por la arquitectura Transformer.

Para lograr esto, la guía se estructura en cuatro secciones clave: Fundamentos de PLN para Deep Learning, que cubre la preparación del texto y el concepto de la representación vectorial; Modelos Fundamentales de Secuencia, analizando RNNs y cómo el mecanismo de gating y de atención permitió superar las limitaciones secuenciales; La Arquitectura Transformer, detallando la autoatención como su núcleo operativo; y finalmente, Aplicaciones Avanzadas del PLN con Deep Learning, ilustrando cómo estas técnicas resuelven problemas del mundo real.

1. Fundamentos de PLN para Deep Learning

Para que las redes neuronales profundas puedan interactuar con el lenguaje, este debe transformarse de una secuencia de símbolos discretos (caracteres o palabras) a una matriz de valores numéricos continuos. Este proceso comienza con la preparación del texto y culmina con la creación de representaciones vectoriales densas.

1.1. Preprocesamiento de Texto

El Preprocesamiento de Texto es la primera etapa y es crucial para asegurar que los datos de entrada sean consistentes y manejables por los modelos. El texto crudo (o *raw text*) requiere varios pasos de limpieza y estructuración.

El primer paso es la tokenización, el proceso de dividir el texto en unidades significativas llamadas *tokens*. Tradicionalmente, la tokenización se realizaba a nivel de palabra (separando el texto por espacios y puntuación), aunque esto presenta desafíos para idiomas aglutinantes o al encontrar palabras fuera del vocabulario (OOV). En el contexto del Deep Learning moderno y los *Transformers*, la tokenización a nivel de subpalabra ha ganado preponderancia.

Este método, ejemplificado por técnicas como WordPiece (utilizado por BERT) o Byte-Pair Encoding (BPE) (utilizado por GPT), descompone palabras raras en subunidades más pequeñas y frecuentes. Por ejemplo, la palabra "tokenization" podría dividirse en *tokens* como "token", "iz", y "ation". Esto asegura que, incluso las palabras nunca vistas en el entrenamiento, puedan ser representadas a partir de *tokens* constituyentes conocidos.

Tras la tokenización, se realiza la normalización. Esto incluye la conversión a minúsculas y la gestión de la puntuación y caracteres especiales. Una técnica importante en esta fase es la lematización, que reduce las palabras flexionadas a su forma base o lema canónico, utilizando un diccionario morfológico. Por ejemplo, las formas verbales "fui", "eras" y "somos" se lematizarían a "ser".

Esto es preferible al stemming (o derivación), que simplemente trunca la palabra heurísticamente (p. ej., reduciendo "computadora" y "computación" a "comput"), lo que puede dar lugar a *tokens* que no son palabras válidas. La correcta normalización y lematización permiten que el modelo trate diferentes formas de la misma palabra como una sola unidad semántica, mejorando la eficiencia del aprendizaje.

1.2. Representación Vectorial (Embeddings)

La Representación Vectorial del Texto o *Embeddings* es el pilar que conecta el lenguaje humano con el ámbito matemático de las redes neuronales. Consiste en mapear *tokens* a vectores densos y de baja dimensión ($d \ll |V|$, donde d es la dimensión del vector y $|V|$ es el tamaño del vocabulario) en un espacio euclidiano. En este espacio, la distancia geométrica (comúnmente la similitud del coseno) entre dos vectores se correlaciona con la similitud semántica y sintáctica entre las palabras que representan.

Históricamente, los primeros modelos de *embeddings* distribucionales como Word2Vec (Mikolov et al., 2013) y GloVe (Pennington et al., 2014) aprendieron estas representaciones estáticas (como se explicó en la introducción), basándose en la hipótesis distribucional: las palabras que aparecen en contextos similares tienden a tener significados similares. Estos *embeddings* lograban capturar

sorprendentes relaciones analógicas. Por ejemplo, al realizar aritmética vectorial en el espacio de *embeddings*, se observaba que:

$$Vector("Rey") - Vector("Hombre") + Vector("Mujer") \approx Vector("Reina")$$

Sin embargo, el verdadero avance del Deep Learning fue la creación de Embeddings Contextuales. A diferencia de los modelos anteriores, que asignaban un único vector a "banco", los modelos contextuales (como los generados por las capas de los *Transformers*) generan un vector diferente para la palabra "banco" en cada oración, reflejando fielmente el sentido particular que la palabra tiene en ese contexto específico.

Este cambio de representaciones estáticas a representaciones dinámicas y contextuales es la base del poder de los modelos actuales de PLN. La explicación detallada de las arquitecturas Word2Vec y GloVe, junto con el análisis de los *embeddings* contextuales, será profundizada en los siguientes materiales de la unidad.

2. Modelos Fundamentales de Secuencia

Las arquitecturas iniciales de *Deep Learning* diseñadas para procesar secuencias de texto se basaron en las Redes Neuronales Recurrentes (RNN). Estos modelos fueron esenciales para establecer el manejo de la dependencia temporal en el lenguaje, aunque con limitaciones que impulsaron el desarrollo de mecanismos más sofisticados.

2.1. Redes Neuronales Recurrentes (RNN)

Las Redes Neuronales Recurrentes (RNN) son un tipo de red neuronal artificial diseñada específicamente para operar sobre datos secuenciales (texto, audio o series de tiempo). La característica distintiva de una RNN es su capacidad para mantener un estado oculto (h_t) que se propaga a lo largo de la secuencia, capturando la información de los pasos temporales anteriores. En cada paso t, el estado oculto se calcula a partir de la entrada actual (x_t) y el estado oculto anterior (h_{t-1}), utilizando funciones de activación como la tangente hiperbólica ($tanh$):

$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t + b_h\right)$$

Donde W_{hh} y W_{xh} son las matrices de pesos que se comparten a lo largo de toda la secuencia. Esta compartición de pesos es lo que permite a la red aprender patrones a través de la longitud de la secuencia, sin aumentar el número de parámetros con cada *token*.

El principal desafío de las RNNs simples es el problema del gradiente evanescente (*vanishing gradient*). Durante el entrenamiento, que implica la propagación hacia atrás a través del tiempo (BPTT), los gradientes que contienen la información de error tienden a disminuir exponencialmente a medida que se propagan a través de pasos de tiempo distantes. Esto significa que la red pierde la

capacidad de aprender o recordar dependencias que abarcan largos tramos de texto (la dependencia a largo plazo).

2.2. Variantes con Mecanismos de Compuertas (*Gating*) (LSTM y GRU)

Para contrarrestar el problema del gradiente evanescente y mejorar la captura de dependencias a largo plazo, se desarrollaron arquitecturas recurrentes más complejas, dotadas de mecanismos de compuertas (*gating*).

Long Short-Term Memory (LSTM)

Las LSTM (Long Short-Term Memory), propuestas por Hochreiter y Schmidhuber (1997), introducen una célula de estado (C_t) o "autopista de información" que fluye linealmente con pocas interacciones no lineales, facilitando que la información importante se mantenga o se omita. El flujo de información en la célula es regulado por tres compuertas principales, que utilizan la función sigmoide (σ) para producir valores entre 0 (cerrado) y 1 (abierto):

1. Compuerta de Olvido (f_t): Decide qué información debe ser descartada del estado de célula anterior (C_{t-1}).
$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

2. Compuerta de Entrada (i y $\tilde{C}_t$): Decide qué nueva información de la entrada actual (x_t) es relevante para actualizar el estado de célula.
$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$

3. Compuerta de Salida (o_t): Decide qué partes del estado de célula actualizado (C_t) se utilizarán para calcular el estado oculto final (h_t).
$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

El nuevo estado de célula se calcula como: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$. El uso de la multiplicación por elementos ($*$) y la adición asegura un mejor flujo del gradiente.

Gated Recurrent Unit (GRU)

La GRU (Gated Recurrent Unit) es una variante más simple de la LSTM, que combina las compuertas de olvido y entrada en una única compuerta de actualización (z_t) y fusiona el estado oculto y el estado de célula en un solo vector de estado (h_t). Las GRUs son generalmente más rápidas de entrenar debido a su menor número de parámetros, ofreciendo un rendimiento comparable a las LSTMs en muchas tareas de PLN.

2.3. El Mecanismo de Atención como Precursor

Para tareas de mapeo de secuencias de longitud variable a secuencias de longitud variable (Seq2Seq), como la Traducción Automática Neuronal (NMT), se popularizó la arquitectura Encoder-

Decoder. Un *Encoder* (típicamente una LSTM o GRU) lee la secuencia de entrada y la condensa en un único vector de contexto de estado fijo. El *Decoder* utiliza este vector para generar la secuencia de salida *token* por *token*.

El principal defecto de este modelo básico es que el vector de contexto único se convierte en un cuello de botella de información. Para secuencias largas, el *Encoder* es incapaz de comprimir de manera efectiva toda la información relevante en un único vector de tamaño fijo.

El Mecanismo de Atención, introducido por Bahdanau et al. (2014), resolvió este problema de manera radical. En lugar de forzar al *Decoder* a depender únicamente del estado final del *Encoder*, la atención permite que el *Decoder* acceda directamente a todos los estados ocultos generados por el *Encoder* en cada paso de tiempo.

El proceso opera en tres etapas conceptuales:

1. Cálculo de Puntuaciones de Alineación: En el paso i de la decodificación, se calcula una puntuación de compatibilidad (o *score*) entre el estado oculto actual del *Decoder* (s_i) y cada uno de los estados ocultos del *Encoder* (h_j).
2. Cálculo de Pesos de Atención: Estas puntuaciones se normalizan utilizando la función *Softmax* para obtener un conjunto de pesos de atención (α_{ij}). La suma de todos los pesos es igual a uno ($\sum_j \alpha_{ij} = 1$). Estos pesos representan la importancia relativa de cada *token* de la entrada para generar el *token* de salida actual.
3. Cálculo del Vector de Contexto: El vector de contexto (c_i) utilizado por el Decoder en el paso i se calcula como una suma ponderada de los estados ocultos del Encoder, donde los pesos son los coeficientes de atención calculados:

$$c_i = \sum_{j=1}^{T_z} \alpha_{ij} h_j$$

Este mecanismo permitió a los modelos de NMT "enfocar" su atención de manera selectiva y dinámica en las partes más relevantes de la secuencia de entrada al generar cada *token* de la salida, logrando avances significativos en la calidad de la traducción. La atención no solo mejoró el rendimiento, sino que también proporcionó un nivel de interpretabilidad a los modelos recurrentes, al permitir visualizar qué partes de la entrada contribuyen más a la salida.

3. La Arquitectura Transformer

La arquitectura Transformer, presentada por Vaswani et al. (2017) en su influyente artículo "Attention Is All You Need", representó el cambio de paradigma más significativo en el PLN moderno. Su innovación clave fue demostrar que la recurrencia (propia de las RNNs, LSTMs y GRUs) era innecesaria para capturar dependencias de largo alcance, basándose por completo en el mecanismo

de atención. Esto permitió una paralelización masiva del entrenamiento, superando la limitación secuencial de las arquitecturas anteriores.

3.1. Autoatención (*Self-Attention*)

El corazón del Transformer es el mecanismo de Autoatención (*Self-Attention*) o Atención Escalada de Producto Punto (*Scaled Dot-Product Attention*). A diferencia de la atención Seq2Seq (donde el *Decoder* atiende al *Encoder*), la Autoatención permite que un *token* en la secuencia pondere la relevancia de todos los demás *tokens* en esa misma secuencia para entender su propio significado contextual.

El mecanismo requiere que cada *token* de entrada x_i sea proyectado linealmente en tres vectores distintos:

1. Query (Q): El vector de consulta, que representa lo que el *token* está buscando o pidiendo.
2. Key (K): El vector clave, que representa lo que el *token* ofrece.
3. Value (V): El vector de valor, que transporta el contenido informativo del *token*.

La puntuación de atención se calcula mediante los siguientes pasos, conocidos como la función de atención:

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Donde Q, K, y V son matrices que contienen las consultas, claves y valores de toda la secuencia. El término $\sqrt{d_k}$ (raíz cuadrada de la dimensión de las claves) se utiliza para escalar el producto punto, evitando que los valores sean demasiado grandes y saturen el gradiente de la función *Softmax*.

El resultado de esta operación es una suma ponderada de los vectores V, donde los pesos son determinados por la similitud entre el vector Q del *token* actual y todos los vectores K de la secuencia. Este proceso permite al modelo crear representaciones contextuales ricas, por ejemplo, resolviendo ambigüedades como el pronombre "ello" en la frase "La célula perdió el núcleo porque ello murió", donde la autoatención asignará un peso alto al *token* "célula" para determinar el referente correcto.

3.2. Atención Multi-Head (*Multi-Head Attention*)

El Transformer extiende la Autoatención mediante la Atención Multi-Head (MHA). En lugar de realizar una única proyección de la Autoatención, la MHA divide los vectores Q, K, y V en múltiples subespacios (o "cabezas", H), y realiza la atención en paralelo en cada uno.

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_H)W^O$$

Donde $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$.

Cada cabeza aprende a enfocarse en diferentes aspectos o relaciones dentro de la secuencia (p. ej., una cabeza podría enfocarse en la sintaxis, otra en la semántica, y otra en las dependencias de largo alcance). Las salidas de todas las cabezas se concatenan y se proyectan linealmente (W^O) para producir el vector de salida final. Esto dota al modelo de una capacidad de modelado paralela y diversificada.

3.3. Arquitectura General y Positional Encoding

La arquitectura completa del Transformer consta de un Encoder y un Decoder, ambos apilados con múltiples bloques idénticos. Cada bloque incluye:

1. Una capa de MHA.
2. Una capa de *Feed-Forward Network* (FFN) o *Position-wise Feed-Forward Network*, aplicada de forma idéntica e independiente a cada posición.

Dado que el Transformer carece de recurrencia, no procesa la información secuencialmente y, por lo tanto, pierde la información del orden de las palabras. Para reintroducir esta información crucial, el modelo utiliza el Positional Encoding (PE). El PE es un vector que se añade al *embedding* de entrada, codificando la posición absoluta de cada *token* en la secuencia mediante funciones seno y coseno de diferentes frecuencias, permitiendo al modelo inferir las relaciones de orden.

3.4. Modelos Preentrenados y el Transfer Learning

La arquitectura Transformer se convirtió en la base de los Modelos de Lenguaje a Gran Escala (LLMs), estableciendo el paradigma Preentrenamiento y Ajuste Fino (*Pre-training and Fine-tuning*).

Durante el Preentrenamiento, los modelos se entrenan en corpus de texto masivos (p. ej., Common Crawl, Wikipedia) en tareas de auto-supervisión que no requieren etiquetado manual:

- Modelado de Lenguaje Enmascarado (*Masked Language Modeling - MLM*): Utilizado por modelos *Solo-Encoder* como BERT (Devlin et al., 2018). El modelo aprende a predecir *tokens* que han sido intencionalmente enmascarados en la entrada, forzándolo a utilizar el contexto bidireccional completo (izquierda y derecha) para inferir la palabra correcta. Esto genera representaciones altamente contextualizadas, ideales para tareas de comprensión.
- Modelado de Lenguaje Causal (*Causal Language Modeling - CLM*): Utilizado por modelos *Solo-Decoder* como GPT (Radford et al., 2018). El modelo aprende a predecir el siguiente *token* en la secuencia basándose únicamente en el contexto unidireccional (a la izquierda), lo que lo hace particularmente hábil en tareas de generación de texto coherente.

El conocimiento lingüístico profundo adquirido se transfiere a tareas específicas (*Downstream Tasks*) (como la clasificación de texto o la respuesta a preguntas) mediante el proceso de Ajuste Fino (*Fine-tuning*), donde los parámetros del modelo preentrenado se ajustan ligeramente utilizando un

conjunto de datos etiquetado y mucho más pequeño. Este proceso es la clave de la eficiencia y el alto rendimiento de los modelos modernos en el PLN aplicado.

4. Aplicaciones Avanzadas del PLN con Deep Learning

La madurez de los modelos basados en *Transformers* y el paradigma de preentrenamiento han catapultado el rendimiento en tareas aplicadas del PLN. Hoy en día, los sistemas de *Deep Learning* no solo clasifican o traducen, sino que también razonan contextualmente y generan contenido coherente.

4.1. Análisis de Sentimiento y Clasificación de Texto

El Análisis de Sentimiento (o minería de opinión) es una tarea fundamental de la Clasificación de Texto, cuyo objetivo es determinar la actitud (positiva, negativa, neutra) del hablante o escritor hacia un tema o entidad.

Mientras que los métodos tradicionales utilizaban diccionarios léxicos y *features* estadísticas, los modelos basados en *Deep Learning* (particularmente los *Solo-Encoder* como BERT) logran una precisión muy superior gracias a su capacidad de comprensión contextual. Un modelo preentrenado puede diferenciar matices, gestionar negaciones (e.g., "La película no es mala") e incluso identificar sarcasmo. Para tareas de clasificación más generales, como la detección de *spam*, la categorización de documentos o el enrutamiento de consultas de clientes, la etapa de ajuste fino (*fine-tuning*) sobre un modelo Transformer permite adaptar el conocimiento lingüístico general a dominios específicos con un alto rendimiento.

4.2. Reconocimiento de Entidades Nombradas (NER)

El Reconocimiento de Entidades Nombradas (NER) es una tarea de etiquetado de secuencias (*sequence labeling*) cuyo objetivo es localizar y clasificar elementos de información atómica y predefinida, como nombres de personas, organizaciones, ubicaciones, expresiones temporales o valores monetarios, dentro de un texto.

En el contexto del *Deep Learning*, el NER se resuelve comúnmente mediante modelos basados en RNN bidireccionales (Bi-LSTMs) o, más eficientemente, mediante la salida de la capa *Encoder* de un *Transformer*. La tarea se formula como una clasificación para cada *token* en la secuencia utilizando el esquema de etiquetado BIO (Begin, Inside, Outside). Por ejemplo, en la frase: "Elon Musk fundó Tesla en Palo Alto en 2003", el modelo asignaría:

$$\text{Elon(B-PER) Musk(I-PER) fundó(O) Tesla(B-ORG) ...}$$

El BERT es especialmente eficaz en esta tarea porque sus *embeddings* contextuales bidireccionales permiten al modelo tomar decisiones de clasificación para un *token* (t_i) basándose en el contexto completo de la oración (antes y después de t_i), algo crucial para distinguir, por ejemplo, "Washington" como persona vs. como ubicación.

4.3. Traducción Automática Neuronal (NMT)

La Traducción Automática Neuronal (NMT) es el estándar de oro en la traducción automática desde que los modelos Seq2Seq con atención (Bahdanau et al., 2014) demostraron una calidad superior a los métodos estadísticos. La introducción del Transformer (Vaswani et al., 2017), con su estructura pura de Encoder-Decoder sin recurrencia, consolidó este avance, permitiendo que los sistemas de NMT escalaran a múltiples idiomas y corpus masivos (p. ej., Google Translate).

La NMT moderna utiliza la atención cruzada (*Cross-Attention*) en el *Decoder* para alinear la frase objetivo con la frase de origen, asegurando que se mantenga el significado y la coherencia estructural. El resultado es una traducción que es no solo literal, sino también fluida y contextualizada, capturando el significado general del texto en lugar de solo la traducción directa de palabras.

4.4. Respuesta a Preguntas (Question Answering - QA)

La capacidad de un modelo de *Deep Learning* para responder a preguntas requiere una comprensión profunda del lenguaje, no solo a nivel de palabra sino también a nivel de documento. Existen dos categorías principales:

1. QA Extractiva: El modelo debe identificar y extraer un tramo de texto exacto (el *span*) del documento de referencia que contiene la respuesta a la pregunta. Modelos como BERT (Devlin et al., 2018) fueron pioneros en esta tarea, formulándola como un problema de predicción de la posición de inicio y fin de la respuesta en el contexto. El modelo predice dos vectores de probabilidad: uno para el inicio y uno para el final del *span* de respuesta.
2. QA Generativa: El modelo debe generar una respuesta completamente nueva y coherente, que puede requerir sintetizar información de múltiples fuentes o basarse en el conocimiento interno del modelo (Modelos de Conocimiento, *Knowledge-Grounded Models*). Esta tarea es el dominio de los modelos *Solo-Decoder* grandes, y es la base de los asistentes virtuales y los *chatbots* avanzados.

4.5. Resumen de Texto (*Text Summarization*)

El resumen de texto busca condensar un documento extenso en una versión más corta, preservando la información crucial. Los modelos *Transformer* (especialmente los *Encoder-Decoder* como BART o T5) son los más utilizados. Se distingue entre:

- Resumen Extractivo: El modelo selecciona las oraciones más importantes del texto original para formar el resumen. Es menos complejo y más fiel a la fuente.
- Resumen Abstractivo: El modelo genera frases nuevas que condensan la información, utilizando vocabulario y sintaxis que no estaban presentes en el documento original. Esta es una tarea de *generación de lenguaje* mucho más desafiante, ya que requiere un

entendimiento profundo y la capacidad de parafrasear y sintetizar sin caer en la alucinación (generación de información falsa).

Estos avances demuestran cómo los principios de atención y las arquitecturas *Transformer* han permitido al PLN pasar de ser una herramienta de procesamiento lingüístico superficial a una herramienta de comprensión y generación de lenguaje sofisticada, fundamental para la IA moderna.

Referencias Bibliográficas

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. OpenAI.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.