Investigación preliminar sobre celda LSTM iterativa

Leandro Palma

5 de septiembre de 2016

1. Introducción

El uso de modelos de secuencias ha tenido un auge a partir de recientes investigaciones que prueban su versatilidad y precisión en contextos complejos y de longitud indefinida.

Sin embargo la estructura fundamental detrás de dichas implementaciones sólo sufrió ligeras variaciones desde su concepción hace aproximadamente 10 años. Fue entonces, cuando se modificó la estructura básica de una red neuronal recurrente para que considere el contexto en su evaluación y la propagación de los errores.

La estructura resultante de dicha modificación fue la celda LSTM, la cual mantuvo su estructura básica hasta la actualidad.

Esta investigación tiene como objetivo explorar la aplicación de sucesivas evaluaciones de una celda LSTM dentro de una red neuronal. Dichas evaluaciones estarán condicionadas por la activación de una compuerta basada en una regresión logística sobre variables de la celda y su contexto.

2. Trabajo previo

La idea de evaluar en forma iterativa una porción de una red neuronal no es nueva y recientemente ha recibido más atención gracias a la semejanza con el funcionamiento del cerebro fundamentada en [1].

[2] analiza el potencial de un red recurrente iterando de forma indefinida y su semejanza con un aproximador universal de Turing.

Adicionalmente, conjetura que el avance en el desempeño de redes neuronales extensas visto en [3] están justificados en la propiedad de dichas redes neuronales en imitar redes iterativas.

Dicha investigación expone evidencia de la mejoría en el desempeño de la red neuronal en un rango de cantidad de iteraciones, encontrando que hay un valor óptimo de iteraciones a realizar para el modelo entrenado. Este análisis fue la principal motivación de esta investigación.

Análogamente, [4] entrena una red neuronal convolucional para que realice iterativamente evaluaciones sobre una imágenes reduciendo su dimensionalidad hasta el conjunto de clases a identificar.

[5] analiza el desempeño de distintas topologías de redes iterativas mostrando resultados que superan al estado del arte en variadas disciplinas.

Otro tipo de modificaciones en la estructura de las celdas LSTM, como la propuesta en [6]; donde se conectan los estados de celdas en capas adyacentes, exponen resultados favorables a la modificación de la estructura básica de una celda LSTM.

[8] investigan la capacidad de inferir información concreta por medio de regresiones softmax que aporte un comportamiento definido como reglas ante los mismos casos de variables de entrada.

Finalmente, en [7] los autores proponen por primera vez la celda LSTM argumentando que en el enfoque básico los parámetros de la red neuronales son actualizados en forma conflictiva, además de no preservar las dependencias por secuencias de extensa longitud.

Particularmente los autores destacan que una estructura más adecuada para el computo evitaría los conflictos mencionados. Aun después de tanto tiempo esta idea de proveer una estructura más adecuada y sensible al contexto para el cómputo es lo que motiva investigaciones como esta a explorar variantes de la estructura básica de la celda LSTM.

3. LSTM iterativa

Según lo anticipado anteriormente, se investiga el desempeño de una estructura análoga a LSTM pero con la capacidad de realizar sucesivas evaluaciones sobre los datos de entrada.

La cantidad de iteraciones a realizar por la celda estará regulada por una compuerta de activación. La misma aplica una regresión logística sobre los datos del contexto actual de la red neuronal sobre la que se itera, tales como los datos de entrada y el estado de la celda.

La figura 3 muestra la estructura en cuestión, en este caso para una red iterativa de una celda.

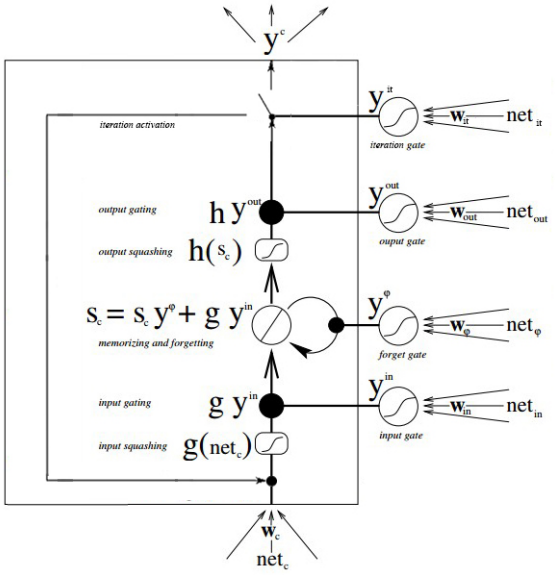

Figura 1: Esquema de la estructura de la celda LSTM iterativa propuesta.

Como puede verse, la estructura coincide con la de una celda LSTM básica retro alimentada t veces con una composición de los datos de entrada y la última evaluación de dicha celda.

Siguiendo el enfoque de construcción de una estructura adecuada comentado en la introducción se propone el mecanismo de retroalimentación de la celda. Este consiste en la composición de la entrada a una celda LSTM normal, proveniente de la salida de la capa anterior, con la salida de la última iteración realizada por la celda para este instante de la secuencia. En caso de que sea la primera iteración, sólo se considera el valor de entrada a la celda.

Se preserva el funcionamiento interno de una celda LSTM básica, tanto en el cálculo como en la transición de estados entre evaluaciones.

La retroalimentación de la celda para el caso en el que esta representa una sub red neuronal de longitud mayor a una única capa, la iteración se aplica a

todas las capas en forma secuencial y la realimentación se da desde la última capa a la primera.

Este enfoque es el usado tanto en [2] como en [5] argumentando que la transformación realizada por múltiples capas es capaz de adaptarse a una simple identidad si se usa residual learning en el caso de que no fuese necesaria dicha capa.

Finalmente, el último componente agregado a la estructura es la compuerta de activación de la iteración.

Este componente consta de una regresión logística realizada sobre un conjunto de variables de la celda y su contexto. Esto significa que aplica una transformación afín sobre las variables y sobre ese resultado aplica una función de activación.

La justificación de que un componente tan simple pueda determinar en forma efectiva la correlación de las variables con la activación o no de la compuerta está basada en el análisis realizado en [8] sobre el potencial de estas estructuras para aprender a replicar reglas simples. En este caso las reglas a aprender son las que determinan bajo qué valores de las variables es conveniente producir una iteración adicional de la celda.

La función de activación usada en todos los casos es la función sigmoidea. Su resultado es usado para determinar la correspondencia con alguna de las dos clases objetivo, en este caso la realización de otra iteración o la exposición del resultado obtenido a la siguiente capa.
Este resultado pertenece al intervalo $x \in [0, 1]$, por lo tanto es interpretado como la probabilidad de las clases objetivo.

Por otra parte, para finalmente determinar la activación o no de la compuerta se discretiza el valor en 0 o 1, según el resultado pertenece o no pertenece al intervalo de longitud definida por la probabilidad de cada clase objetivo.

Este valor discretizado es calculado para cada unidad de cada celda y es usado no solo para conocer la condición de seguir iterando de la celda como un todo, sino que también como función indicadora del valor de la salida de cada unidad a usar, siendo el resultado de la última iteración o el resultado de la iteración actual los que se corresponden a que la unidad de la celda realice una nueva iteración o no, respectivamente.

La condición de seguir iterando de la celda dependerá finalmente de que al menos una de las compuertas de iteración de las unidades que la componen indique la activación de una iteración adicional.

En particular la operación lógica para determinar la condición de iteración

de la celda completa es un OR sobre la salida de la compuerta de activación de las unidades que la componen.

4. Variaciones

Los componentes de la estructura presentada admiten múltiples implementaciones, todas en principio son coherentes. Esto motivó la ejecución de numerosas ejecuciones del experimento para evaluar el desempeño y viabilidad de cada configuración en particular, y de la idea como un todo. Los resultados se detallan en la sección 6.

4.1. Datos de entrada

En la estructura de una red neuronal recurrente convencionales el estado de la celda es una función de las salidas de las capas previas a la celda en un instante t y de los estados previos de dicha celda. A partir de este nuevo estado se computa la salida de la celda.

Por este motivo la elección de los datos de entrada es fundamental en el flujo de información dentro de la red neuronal, no sólo para el cálculo del valor actual de la secuencia más que para la secuencia como un todo.

En la versión iterativa estudiada, este componente también define cómo se retro alimenta a la celda; por lo tanto el cálculo de su estado entre iteraciones y el cálculo de la activación de una siguiente iteración.

Las principales configuraciones estudiadas para el componente son:

- Salida de capa previa constante entre iteraciones, como entrada. Estado iteración previa, como estado (vectores h y c).

  En esta configuración los únicos vectores variables entre iteraciones son los que conforman el estado.

- Salida de capa previa compuesta con salida iteración previa, como entrada. Estado iteración previa, como estado.

  En esta configuración la entrada a la celda es calculada en función de la salida de la capa anterior y la salida de la celda en la iteración anterior.

  Dos variantes de este enfoque fueron estudiadas, la primera como el cálculo residual (salida de la capa anterior sumado elemento a elemento a la salida de la celda en la iteración anterior) y la segunda análoga a la primera pero como un cálculo ponderado (salida de la capa anterior multiplicado elemento a elemento a la salida de la celda en la iteración anterior). Estas

variantes fueron las estudiadas en [5].

El estado para ambas variantes de la celda también es actualizado al obtenido en la iteración previa.

Otras configuraciones de este componente también fueron estudiadas, tales como actualización parcial del estado(únicamente actualizar h o c) y únicamente actualizar la entrada, dejando constante el estado. El desempeño de la red neuronal no mejora para estas últimas configuraciones.

4.2. Datos de entrada de la compuerta de iteración

Según lo visto en la sección 3, es la compuerta de iteración que determina si la celda será evaluada una vez más.

Las variables que considera este componente definirán la cantidad de iteraciones a realizar para el cómputo de cada instante de la secuencia.

Las principales configuraciones estudiadas para este componente son:

- Entrada a la celda y estado resultante de la última iteración, como datos de entrada.
- Misma configuración que la anterior, agregando el estado previo a la última iteración de la celda.

Ambas configuraciones resultan similares, siendo la primera una versión reducida de la segunda.

La razón para proponer una reducción en los parámetros usados para el cálculo de la activación radica en que se reducen los parámetros del modelo. Un modelo con mayor cantidad de parámetros suele tener un peor desempeño que su versión reducida.

Otras configuraciones con variantes menores fueron consideradas, tales como solo considerar parcialmente el estado (seleccionando el vector h o c), considerar el número de iteración actual y reducir los datos de entrada(seleccionando sólo la entrada a la celda o sólo el estado).
Nuevamente, el desempeño de la red no mejora para estas últimas configuraciones mencionadas.

4.3. Probabilidad de iteración

La estructura propuesta determina si realizar una iteración adicional o exponer el resultado de la iteración a las capas superiores en base a una regresión logística sobre datos seleccionados.

La selección del valor de probabilidad mínima para el cual concebir una nueva iteración no sólo impacta en la cantidad de iteraciones a realizar, sino también en la cantidad de veces que se propagan los errores por la celda. El algoritmo de entrenamiento elegido considera a la red con tantas capas como iteraciones realizadas.

El valor de este parámetro considerado fue de 0.5 para la iteración inicial. Su elección se justifica en que no condiciona el cálculo a ningún resultado.

Este valor es reducido en forma cuadrática en cada iteración para evitar una cantidad excesiva de iteraciones.

Otros valores del parámetro fueron considerados sin resultar en mejoras en el desempeño de la red. Asimismo determinados experimentos no reducían el valor inicial, resultando en excesivas iteraciones o en el desempeño regular de la red.

4.4. Máxima cantidad de iteraciones posibles

El planteo teórico de la estructura propuesta no especifica un límite para la cantidad de iteraciones a realizar, sin embargo en la práctica encontramos que no es viable dejar este parámetro libre.

La ejecución del experimento permitiendo una cantidad indefinida de iteraciones produce un bucle ininterrumpido de ejecuciones de la celda, evitando que esta pueda retornar finalmente su valor en un tiempo razonable.

Una posible causa de esto es la inicialización aleatoria de los parámetros de la compuerta de activación de la iteración. Los mismos podrían tomar valores que permitan con una gran frecuencia las iteraciones, produciendo este bucle de indeterminada duración.

La solución práctica a este problema consiste en limitar la cantidad de iteraciones realizadas a un valor determinado. Este valor es un parámetro del comportamiento de la celda.

El valor adoptado es de un máximo de 50 iteraciones, considerando este valor más que suficiente para permitir a la celda resolver en forma iterativa la evaluación de los valores de entrada. Esto se hace evidente al observar que el valor promedio del número máximo de iteraciones realizadas es de 3 iteraciones.

4.5. Longitud de la red a iterar

En [5] y [2] se ejecutan experimentos que realizan iteraciones sobre redes reducidas en longitud. Este enfoque fue considerado en una implementación se-

parada de celda iterativa básica.

Los resultados en esta configuración tienden a ser tan buenos como los de la configuración básica por lo que los mismos experimentos se ejecutan en ambas configuraciones y sus desempeños son comparados.

En principio la configuración básica debería ser capaz de concebir una estructura similar a su versión amplia dado que la elección de la cantidad de iteraciones a realizar permite realizar cálculos sobre una longitud mayor de capas. Aunque considerando que los parámetros entre dichas capas son compartidos puede argumentarse que la red no tendrá el mismo poder de adaptación o aprendizaje al error propagado.

Por otra parte, la configuración básica reduce la cantidad de parámetros del modelo, sugiriendo que debería ser más rápida su convergencia a un mejor desempeño.

4.6. Reactivación de iteraciones por unidad de la celda

Dada que la condición de seguir iterando de la celda dependerá de cada una de las compuertas de activación de la iteración de las unidades que la componen, entonces es posible que conjuntos de estas celdas permanezcan inactivas durante la iteración de la celda completa.

Al finalizar la iteración se recalcula la salida de cada una de las compuertas de activación, incluso la de aquellas unidades para las cuales ya se determinó que no debería aplicarse una iteración adicional.

Esto sucede ya que si bien el resultado de la evaluación de la unidad no cambio y permanece el de la última iteración realizada, es posible que, dependiendo de los datos usados para la determinación de la activación de la iteración, se determine que dicha unidad debería realizar una iteración adicional.

En particular un caso donde esto sucede es aquel en el que la activación de una iteración adicional depende de los datos de entrada a la celda que varían por la iteración de las otras unidades de la misma.

Sobre la definición de este comportamiento se optó por dos alternativas:

- Permitir que las unidades de la celda para las cuales se determinó que no producción nuevas iteración anteriormente se reactiven y produzcan nuevas iteraciones.

  En particular esto se logra considerando únicamente la última salida de la compuerta de activación de la iteración de dicha unidad.

- No permitir que se produzcan nuevas iteraciones para las unidades de la celda para las que anteriormente se determinó que no deberían producirse nuevas iteraciones.

  En particular esto se logra considerando como salida de la compuerta de iteración el resultado de la operación lógica AND entre la salida de la compuerta en la iteración actual y el resultado de la compuerta en la iteración anterior.

No se fue posible concluir cuál de las configuraciones provee un mejor desempeño de la celda y la estructura como un todo. Una posible causa de esto es la directa dependencia con el cálculo de los datos de entrada a la celda y los datos de entrada a las compuertas de activación de la iteración de cada unidad.

5. Implementación

La implementación de la estructura de una celda iterativa propuesta se basa en la implementación de una celda perteneciente a una red neuronal recurrente existente en la librería de código abierto Tensorflow.

Esta librería permite usar el lenguaje de Python para estructurar una serie de operaciones a realizar sobre los datos de entrada.

La ejecución de la estructura resultante sucede en una etapa posterior en la que no se ejecuta código Python, sino que se ejecutan instrucciones optimizadas para dispositivos específico.

Por default y en la implementación provista se ejecutan instrucciones de CPU pero nada impide el uso de otros dispositivos como GPU.

En particular se desarrolló una extensión de la clase `RNNCell`, definiendo un bucle en la evaluación del método `call`.

Determinadas clases provistas por Tensorflow también sufrieron cambios, particularmente para obtener una métrica del número de iteraciones realizadas por las celdas LSTM iterativas en la red neuronal. Todas las modificaciones sobre la implementación original de una celda LSTM se ubicaron en el archivo `rnn_cell`.

La métrica empleada es el promedio, sobre las celdas iterativas, de la máxima cantidad de iteraciones realizada en los *'batchs'* en paralelo.

Cabe destacar que incluso cuando en un *'batch'* la compuerta de activación expuso un valor que evita la evaluación de la celda por más iteraciones, estas se siguen calculando mientras que la celda esté habilitada a seguir realizando iteraciones sobre los datos en algún *'batch'*.

Para los *'batchs'* en los que la celda no está realizando nuevas iteraciones se expone en forma repetida la salida de la última iteración. Por lo tanto el efecto de las nuevas iteraciones no repercute en los valores de salida ni en la propagación de los errores por la celda.

La implementación de Tensorflow de un bucle `tf.while_loop` requiere la definición de dos funciones evaluables, el primero de las cuales consta del cuerpo del bucle, y la segunda de la evaluación condicional.

Para reducir la complejidad se optó por una implementación simple en la cual la función que constituye el cuerpo del bucle calcula tanto los valores de la celda en la iteración actual, como la condición de seguir iterando.
El valor de verdad de esta condición es pasado por una variable booleana a la función que evalúa la condición, que simplemente retorna el valor de dicha variable.

El cuerpo del bucle es análogo para todas las variantes y lo mismo sucede para la evaluación del condicional de dicho bucle. Sus diferencias se aplican en el cálculo resultante de la llamada la función `IterativeLSTM_CellCalculation`.

Para ambos casos el indicador de la activación de la iteración es calculado en el método que resuelve el cálculo de la celda.

El segmento de código 1 muestra el cuerpo del bucle de iteración y la evaluación del condicional usado mayormente sin cambios en todas las implementaciones de las variables.

```
def iterativeLSTM_Iteration(inputs, state, num_units, forget_bias,
    iteration_number, max_iterations,
                            iteration_prob, iteration_prob_decay,
                                iteration_activation, keep_looping)
                                :

    new_output, new_state, new_iteration_activation =
        iterativeLSTM_CellCalculation(inputs, state, num_units,
                                                                    for
                                                                    ite
    iteration_flag = tf.reduce_max(new_iteration_activation)

    new_iteration_number = iteration_number + iteration_flag

    do_keep_looping = tf.logical_and(tf.less(new_iteration_number,
        max_iterations), tf.equal(iteration_flag, tf.constant(1.0))
        )

    new_iteration_prob = iteration_prob * iteration_prob_decay

    # Here the current output is selected. If there will be another
         iteration, then the inputs remain. Otherwise, the last
        output will be used.
```

```
    new_output = tf.cond(do_keep_looping, lambda:  inputs, lambda:
        new_output)

    return new_output, new_state, num_units, forget_bias,
        new_iteration_number, max_iterations, new_iteration_prob,
        iteration_prob_decay, new_iteration_activation,
        do_keep_looping


def iterativeLSTM_LoopCondition(inputs, state, num_units,
    forget_bias, iteration_number, max_iterations,
                                iteration_prob,
                                    iteration_prob_decay,
                                    iteration_activation,
                                    keep_looping):
    return keep_looping
```

Listing 1: Implementación del cuerpo y evaluación condicional del bucle de iteración.

El comportamiento de una celda LSTM y el cálculo de sus variables y salidas se encapsula en el método LSTM. Las implementaciones del mismo varían según la variante de la longitud de la celda elegida.

El objetivo de esta implementación diferenciada es el de hacer un cálculo eficiente de la salida de la compuerta de activación de la iteración en el caso de una longitud de una única capa.

Para ambos casos el cálculo de la nueva probabilidad de iteración y el cálculo del número de iteración actual se calculan en el cuerpo del bucle de iteración, como puede verse en los segmentos de código 2 y 3 respectivamente.

```
new_iteration_prob = iteration_prob * iteration_prob_decay
```

Listing 2: Implementación del cálculo de la nueva probabilidad de iteración.

```
iteration_flag = tf.reduce_max(new_iteration_activation)

new_iteration_number = iteration_number + iteration_flag
```

Listing 3: Implementación del cálculo del número de iteración actual.

La parametrización del comportamiento de la celda se realiza en su instanciación, recibiendo por default valores predefinidos.

Para implementar las variaciones en los parámetros de entrada a la celda se especifica la composición que se usará y se pasa este parámetro por el argumento correspondiente a la entrada a la celda.

El bucle considerará esta composición como el nuevo valor de la entrada. La implementación de este comportamiento puede verse en los segmentos de código 4.

```
new_output = tf.cond(do_keep_looping, lambda: inputs, lambda:
    new_output)

return new_output, new_state, num_units, forget_bias,
    new_iteration_number, max_iterations, new_iteration_prob,
    iteration_prob_decay, new_iteration_activation,
    do_keep_looping
```

Listing 4: Implementación del cálculo del valor de entrada a la celda.

La forma de implementar las variaciones en los datos de entrada a la compuerta de activación de la iteración consiste en seleccionar los datos deseados como elementos del arreglo que recibe la función `linear`.

Dicha función calcula la sumatoria de los productos de los elementos seleccionados con los parámetros del modelo.

Asimismo, la actualización en el estado de la celda dependerá del tensor elegido para que conforme el estado de la celda al final de la iteración.

La implementación de la variación en la longitud de la celda deseada se obtiene a partir de ejecutar el script del lenguaje python correspondiente al modelo que hace uso de la implementación deseada.

A continuación se analizan los detalles relevantes en la implementación de las variantes expuestas con anterioridad.

Si bien la variación de la celda LSTM iterativa de mayor longitud expone los mismos valores que la implementación original, sus implementaciones internas varían.

La justificación de esta variación; considerando que la implementación original es un caso particular de la variación extendida en longitud, radica en que la implementación propuesta para la celda de longitud igual a una única capa reduce el conjunto de operaciones realizadas al mínimo, resultando en una ejecución más rápida.

5.1. LSTM Iterativa básica

La construcción de la red en el caso de la celda iterativa básica se consigue a partir de instanciar tantas celdas como capas se especifiquen en el modelo. Estas instancias se usan para estructurar una instancia de MultiRNN. Siendo esta última una implementación provista por Tensorflow que concatena la evaluación de las celdas instanciadas, aplicando la normalización por Dropout en la conexión entre celdas.

La implementación del cálculo de la celda y su salida de la compuerta de activación de la iteración se calculan en una función que implementan en la

función iterativeLSTM para el caso de la celda de longitud reducida.

La implementación del cálculo de la celda y de la activación de la compuerta de iteración pueden verse en los segmentos de código 5 y 6, respectivamente.

```
def iterativeLSTM_CellCalculation(inputs, state, num_units,
    forget_bias, iteration_prob, iteration_activation):
    number_of_units = num_units.get_shape().dims[0].value

    new_output, new_state, p = iterativeLSTM(inputs, state,
        number_of_units, forget_bias, iteration_activation)

    new_iteration_activation = floor(tf.nn.sigmoid(p) +
        iteration_prob) * iteration_activation

    return new_output,  new_state, new_iteration_activation
```

Listing 5: Implementación del cálculo de la celda LSTM Iterativa básica.

```
def iterativeLSTM(inputs, state, num_units, forget_bias,
    iteration_activation):
    # This function aplies the standard LSTM calculation plus the
        calculation of the evidence to infer if another iteration
        is needed.

    # "BasicLSTM"
    # Parameters of gates are concatenated into one multiply for
        efficiency.
    c, h = array_ops.split(1, 2, state)
    concat = linear([inputs, h], 4 * num_units, True)

    # i = input_gate, j = new_input, f = forget_gate, o =
        output_gate
    i, j, f, o = array_ops.split(1, 4, concat)

    new_c = c * sigmoid(f + forget_bias) + sigmoid(i) * tanh(j)
    new_h = tanh(new_c) * sigmoid(o)

    # Only a new state is exposed if the iteration gate in this
        unit of this batch activated the extra iteration.
    new_h = new_h * iteration_activation + h * (1 -
        iteration_activation)
    new_c = new_c * iteration_activation + c * (1 -
        iteration_activation)

    new_state = array_ops.concat(1, [new_c, new_h])

    # In this approach the evidence of the iteration gate is based
        on the inputs that doesn't change over iterations and its
        state
    p = linear([ inputs, new_state], num_units, True,scope= "
        iteration_activation")
    return new_h, new_state,p
```

Listing 6: Implementación del cálculo de la activación de la compuerta de iteración en celda iterativa básica.

5.2. LSTM Iterativa extendida

Para el caso de la variante extendida en longitud el cálculo de cada capa de la celda se realiza en la función LSTM y se concatenan las distintas capas tantas veces como se halla especificado. La concatenación incorpora el mecanismo de normalización por Dropout.
Al finalizar la evaluación de todas las capas se calcula el valor de la compuerta de activación.

La implementación del cálculo de cada capa y de la activación de la compuerta de iteración pueden verse en los segmentos de código 7 y 8, respectivamente.

```
def LSTM(inputs, state, num_units, forget_bias, scope):
    # This function aplies the standard LSTM calculation.

    # "BasicLSTM"
    # Parameters of gates are concatenated into one multiply for
        efficiency.
    c, h = array_ops.split(1, 2, state)
    concat = linear([inputs, h], 4 * num_units, bias=True, scope=
        scope)

    # i = input_gate, j = new_input, f = forget_gate, o =
        output_gate
    i, j, f, o = array_ops.split(1, 4, concat)

    new_c = c * sigmoid(f + forget_bias) + sigmoid(i) * tanh(j)
    new_h = tanh(new_c) * sigmoid(o)
    new_state = array_ops.concat(1, [new_c, new_h])

    return new_h, new_state
```

Listing 7: Implementación del cálculo de la celda LSTM Iterativa extendida.

```
def iterativeLSTM_CellCalculation(inputs, state, num_units,
    num_layers, forget_bias, iteration_prob, iteration_activation,
    keep_prob):
    # This function calculates a single iteration over the entire
        network.

    number_of_units = num_units.get_shape().dims[0].value
    number_of_layers = num_layers.get_shape().dims[0].value
    cur_state_pos = 0
    cur_inp = inputs
    new_states = []
    cs =[]
    new_cs =[]
    for i in range(number_of_layers):
        with vs.variable_scope("Cell%d" %i):
          cur_state = array_ops.slice(state, [0, cur_state_pos],
              [-1, number_of_units * 2])
          c, h = array_ops.split(1, 2, cur_state)
          cs.append(c)
          cur_state_pos += number_of_units * 2
          cur_inp, new_state = LSTM(cur_inp, cur_state,
              number_of_units, forget_bias, vs.get_variable_scope()
              )
```

```
            new_c, new_h = array_ops.split(1, 2, new_state)

            # Only a new state is exposed if the iteration gate in
                this unit of this batch activated an extra iteration.
            new_c = new_c * iteration_activation + c * (1 -
                iteration_activation)
            new_h = (new_h) * iteration_activation + h * (1 -
                iteration_activation)
            new_h = tf.nn.dropout(new_h, keep_prob)

            # Here the new state is the entirely update.
            new_state = array_ops.concat(1, [new_c, new_h])
            new_states.append(new_state)
            new_cs.append(new_c)
            cur_inp = new_h
        new_output = cur_inp
        new_state = array_ops.concat(1, new_states)
        new_c = array_ops.concat(1, new_cs)
        c = array_ops.concat(1, cs)

        p = linear([ inputs, new_state], number_of_units, True, scope="
            iteration_activation")

        new_iteration_activation = floor(tf.nn.sigmoid(p) +
            iteration_prob) * iteration_activation

        return new_output,  new_state, new_iteration_activation
```

Listing 8: Implementación del cálculo de la activación de la compuerta de iteración en celda iterativa extendida.

6. Experimento

El experimento realizados abordan la tarea de modelado del lenguaje. Parten de un set de datos, en particular el set Penn Treebank, constituido por información de numerosas oraciones.

El objetivo del modelo es determinar las palabras más adecuadas para completar una frase.

La naturaleza no determinista del problema implica que no es adecuado entrenar el modelo para reducir el error respecto de un resultado esperado dado que no existe un resultado correcto más que un resultado más probable.

Siguiendo este principio se elige la perplejidad del modelo como medida de su desempeño. Una perplejidad menor implica que las predicciones del modelo aproximan mejor a las soluciones más probables.

Finalmente, los experimentos realizados se basan en la implementación provista como ejemplo en Tensorflow del experimento propuesto en [9].

Los mismos consideran diversos tamaños para los modelos entrenados, los mismos varían en cantidad de celdas por capa y cantidad de capas.

Los modelos sobre los que se ejecutó el experimento son los de tamaño pequeño; compuesto por 2 capas de 200 unidades, tamaño mediano; compuesto por 2 capas de 650 unidades, y tamaño grande; compuesto por 2 capas de 1500 unidades.

Para los modelos estudiados se usaron los métodos de normalización *Batch Normalization*; con un tamaño de *batch* igual a 20 y *Dropout* (solo usado en el modelo de tamaño mediano); con una probabilidad de retener la información de 0.8 . Otros detalles de los parámetros pueden verse en el código fuente de la implementación y en la documentación de Tensorlow.

La estructura elegida como control es la celda LSTM básica comúnmente usada, usualmente denominada *Vanilla LSTM*. Su implementación se detalla en el relevamiento realizado en **??**.

Los resultados del desempeño visibles en la **??**de los modelos en el experimento realizado indican que existe una mejora en la estructura iterativa propuesta respecto de la estructura básica de una celda LSTM. Mostrando que un modelo que emplea la estructura propuesta puede obtener mejor desempeño que un modelo entrenado con la estructura básica.

Una posible explicación alternativa a la que proveen los argumentos presentados podría consistir en que la ejecución de múltiples iteraciones sobre la celda tiene un efecto de normalización sobre el aprendizaje, logrando que el modelo perciba en menor medida del fenómeno de *overfitting*.

Esto es consistente con los valores de perplejidad resultantes del experimento donde se observa que el enfoque tradicional tiene un mejor desempeño en el set de entrenamiento y un peor desempeño en el set de test.

Es oportuno comentar que el efecto en el tiempo de ejecución que se observa al incorporar las estructuras iterativas estudiadas se corresponde con un aumento de aproximadamente 3 veces el tiempo de ejecución de un modelo con celdas LSTM básicas.

Este valor que indica una correlación lineal entre el tiempo de ejecución de los enfoques estudiados. En particular el valor del aumento en el tiempo es aproximadamente el mismo al valor medido de la cantidad de iteraciones máxima promedio.

La correlación detectada es fácilmente argumentable por el hecho de que en cada iteración la celda ejecuta aproximadamente la misma cantidad de operaciones que la celda básica. Por otra parte la celda LSTM básica consideramos que realiza una única iteración frente al promedio de 3 iteraciones observado en los modelos que hacen uso de la estructura iterativa propuesta.

7. Conclusión

La investigación preliminar sobre la estructura propuesta y su desempeño respecto de la versión actualmente usada muestra que es posible obtener una mejora apreciable.
Más investigación sobre la estructura propuesta, sus variaciones y su efecto sobre modelos de mayor tamaño y periodos de entrenamiento más largos permitirían definir concluyente los beneficios de su uso.

Referencias

[1] M. A. Goodale and A. D. Milner. Separate visual pathways for perception and action. *Trends in neurosciences*,15(1):20–25, 1992.

[2] Qianli Liao and Tomaso Poggio. Bridging the Gaps Between Residual Learning,

Recurrent Neural Networks and Visual Cortex. *arXiv preprint arXiv:1604.03640*, 2016

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXivpreprint arXiv:1512.03385*, 2015.

[4] PHO Pinheiro and R Collobert. Recurrent Convolutional Neural Networks for Scene Labeling. *Idiap-RR-22-2013*, 2013.

[5] Isaac Caswell, Chuanqi Shen, and Lisa Wang. Loopy Neural Nets: Imitating Feedback Loops in the Human Brain. *CS231n Report, Stanford0*, $http://cs231n.stanford.edu/reports2016/110_Report.pdf$, Google Scholar timestamp: March 25th, 2016.

[6] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh and Chris Dyer. Depth-Gated LSTM. *arXiv preprint arXiv:1508.03790v4*, 2015.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, vol. 9,pp. 1735–1780, 1997.

[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra,Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil,Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu and Hemal Shah. Wide & Deep Learning for Recommender Systems. *arXiv:1606.07792v1*, 2016.

[9] Wojciech Zaremba, Ilya Sutskever and Oriol Vinyals. Recurrent Neural Network Regularization. `arXiv:1409.2329`, 2014.

[10] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *arXiv:1503.04069*, 2015.

Celda LSTM	Variante	Modelo	Perplejidad
LSTM	Vanilla LSTM	pequeño	Train: 36.862 Valid: 122.413 Test: 117.247
LSTM Iterativa	Longitud: una capa. Probabilidad de iteración: 0.5 . Decaimiento de probabilidad: cuadrático. Máxima cantidad de iteraciones: 50. Reactivación de iteraciones: no permitido. Datos de entrada: entrada constante, estado de la última iteración. Datos de compuerta de iteración: entrada constante, estado de la última iteración. Función de activación de la compuerta de iteración: función sigmoidea	pequeño	Train: 39.805 Valid: 116.698 Test: 110.835
LSTM Iterativa	Longitud: dos capas. Probabilidad de iteración: 0.75 . Decaimiento de probabilidad: cuadrático. Máxima cantidad de iteraciones: 50. Reactivación de iteraciones: no permitido. Datos de entrada: entrada compuesta(por multiplicación) con última salida, estado de la última iteración. Datos de compuerta de iteración: entrada compuesta(por multiplicación) con última salida, estado de la última iteración. Función de activación de la compuerta de iteración: función sigmoidea	pequeño	Train: 44.116 Valid: 87.712 Test: **109.072**
LSTM	Vanilla LSTM	mediano	Train: 44.116 Valid: 87.712 Test: 83.823
LSTM Iterativa	Longitud: una capa. Probabilidad de iteración: 0.5 . Decaimiento de probabilidad: cuadrático. Máxima cantidad de iteraciones: 50. Reactivación de iteraciones: no permitido. Datos de entrada: entrada constante, estado de la última iteración. Datos de compuerta de iteración: entrada constante, estado de la última iteración. Función de activación de la compuerta de iteración: función sigmoidea	mediano	Train: 45.862 Valid: 85.499 Test: **81.601**
LSTM	Vanilla LSTM	grande	Train: 37.87 Valid: 82.62 Test: **78.29**

Cuadro 1: Desempeños del modelo en función de la implementación de celda LSTM usada.