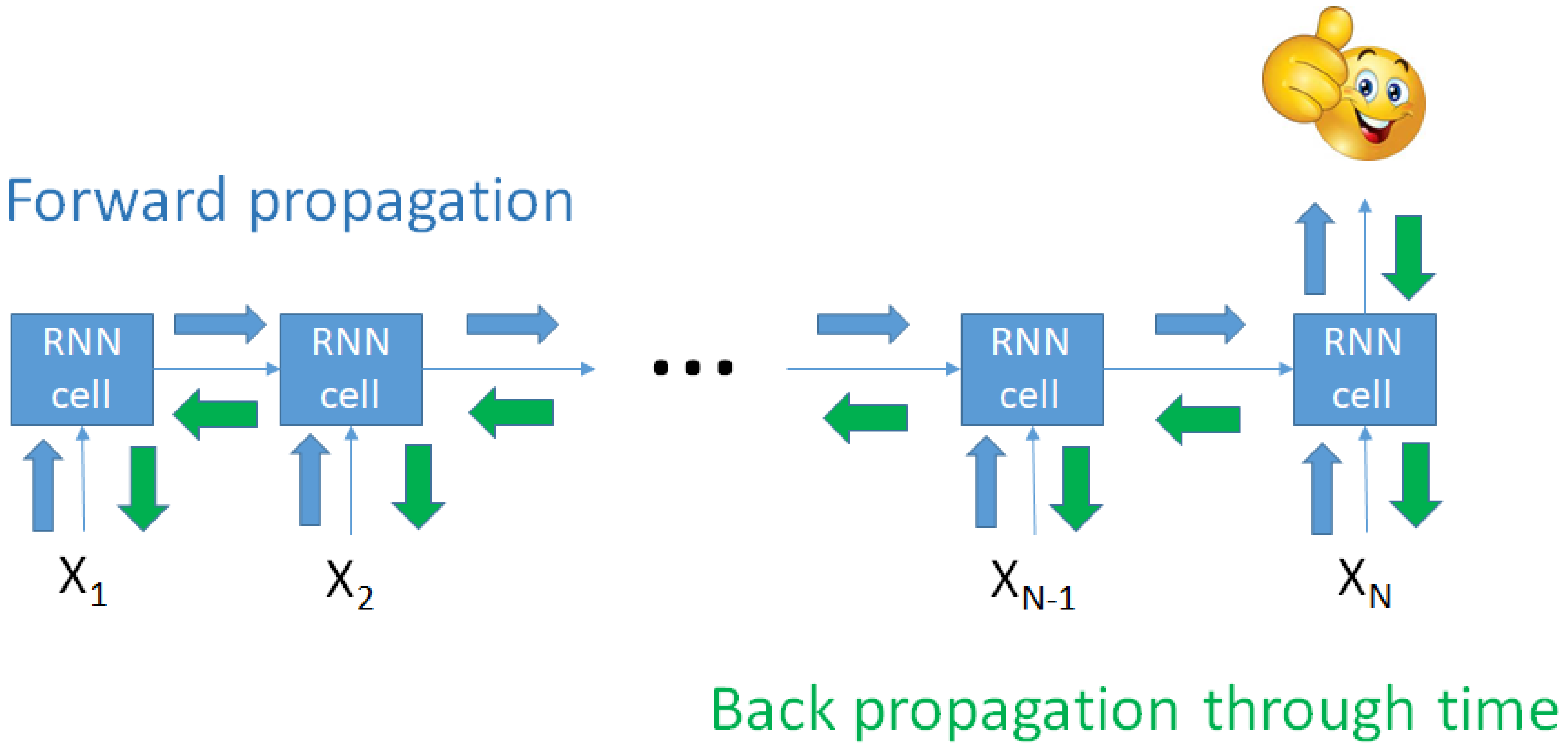
Vanishing and exploding gradients

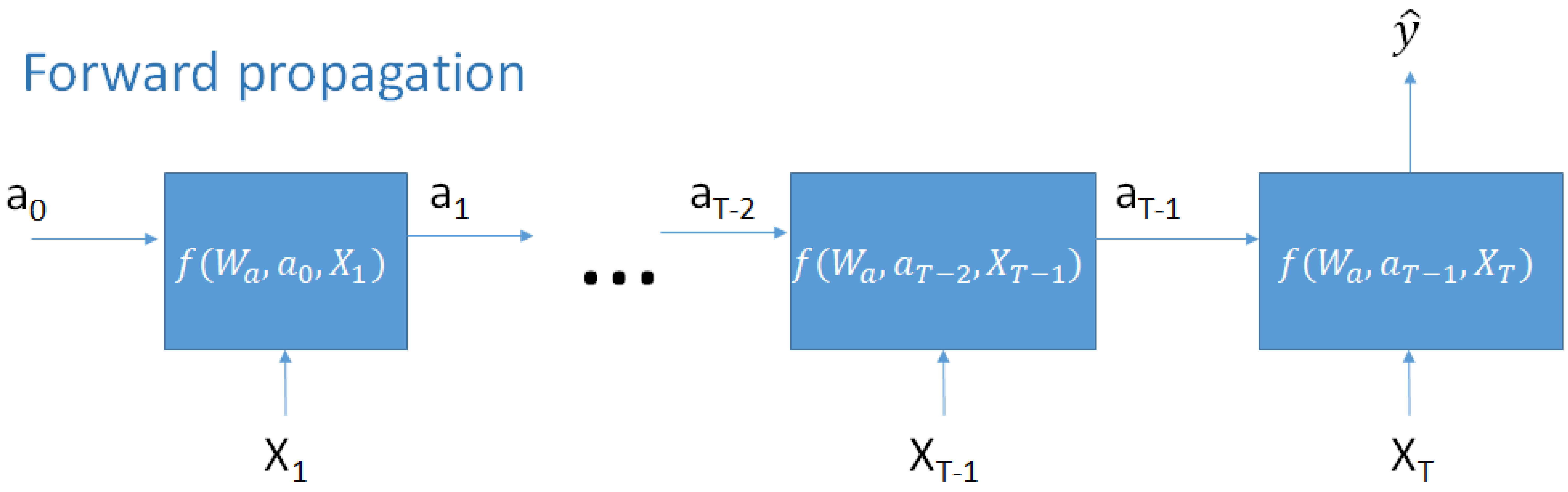
RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

David Cecchini
Data Scientist

Training RNN models



Forward propagation

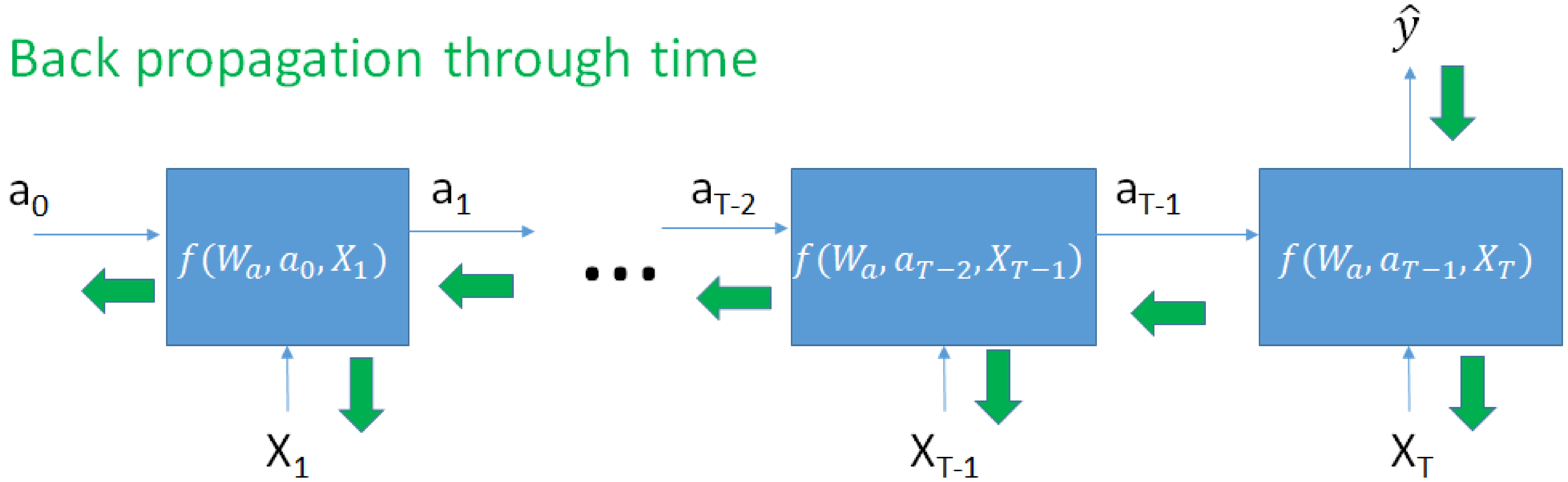


Example:

$$a_2 = f(W_a, a_1, x_2)$$

$$= f(W_a, f(W_a, a_0, x_1), x_2)$$

Back propagation through time



Remember that:

$$a_T = f(W_a, a_{T-1}, x_T)$$

a_T also depends on a_{T-1} which depends on a_{T-2} and W_a, and so on !

BPTT continuation

Computing derivatives leads to

$$\frac{\partial a_t}{\partial W_a} = (W_a)^{t-1} g(X)$$

- $(W_a)^{t-1}$ **can converge to 0**
- **or diverge to** $+\infty$**!**

Solutions to the gradient problems

Some solutions are known:

Exploding gradients

- Gradient clipping / scaling

Vanishing gradients

- Better initialize the matrix W
- Use regularization
- Use ReLU instead of tanh / sigmoid / softmax
- **Use LSTM or GRU cells!**

Let's practice!

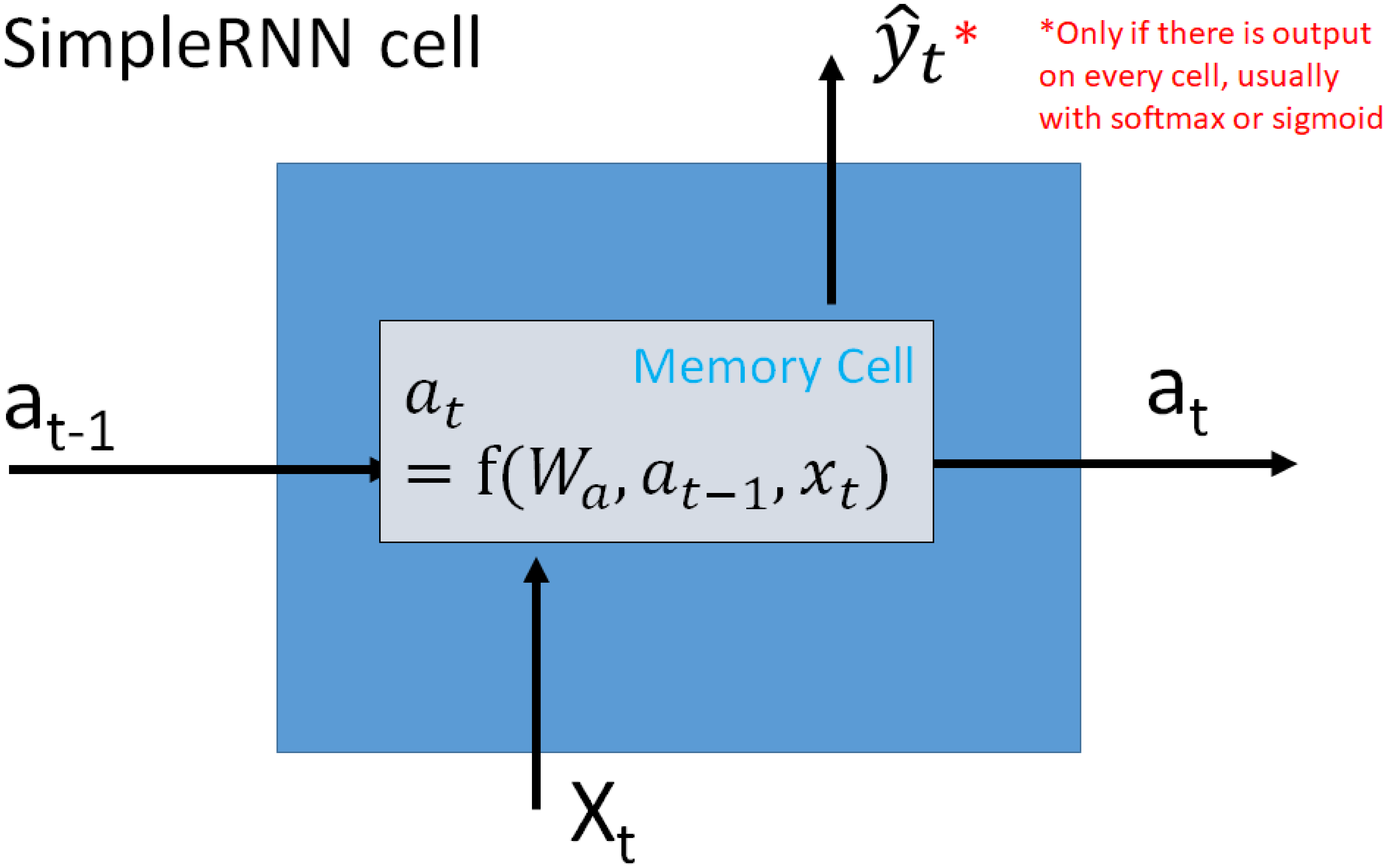
RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

GRU and LSTM cells

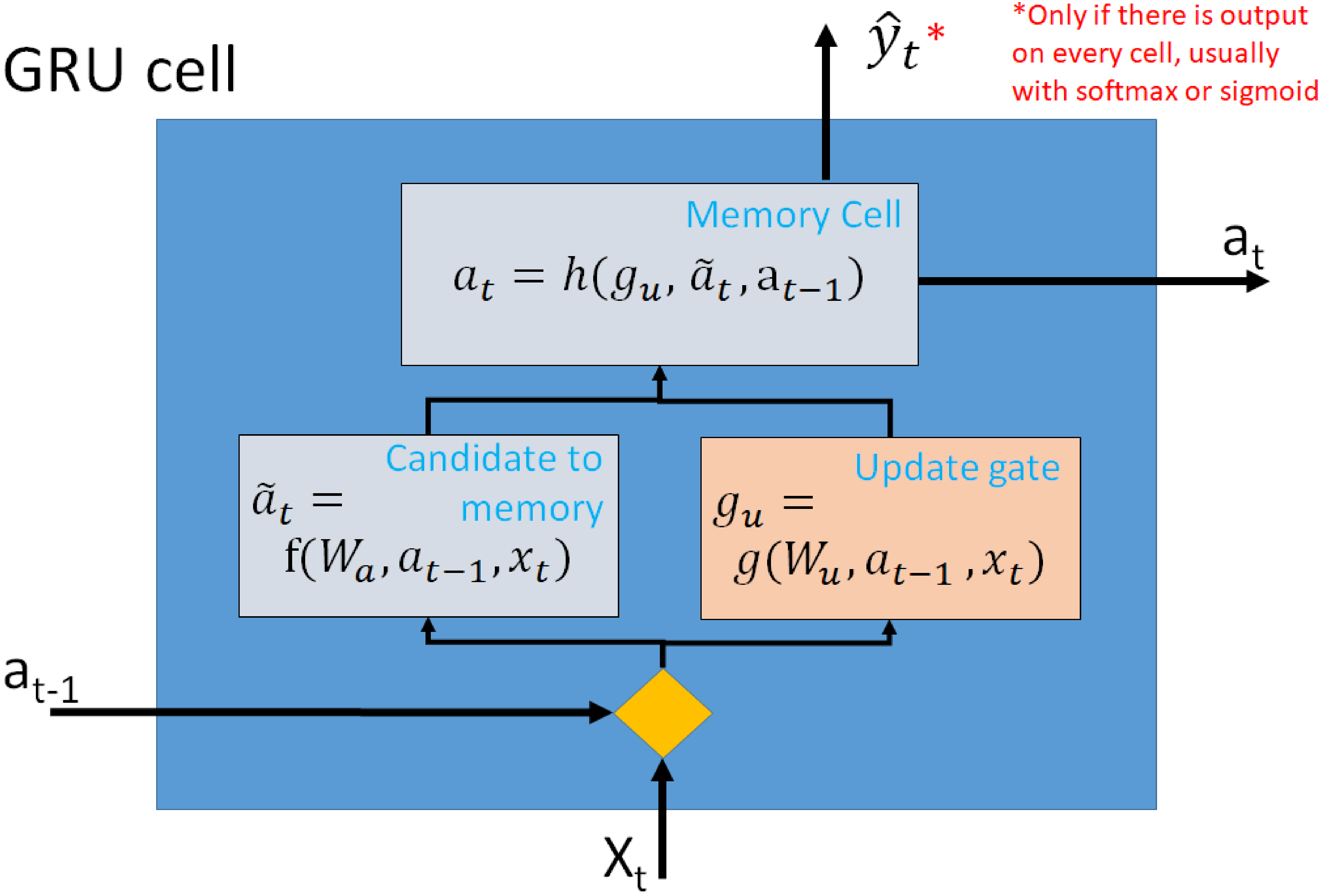
RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

David Cecchini
Data Scientist

SimpleRNN cell

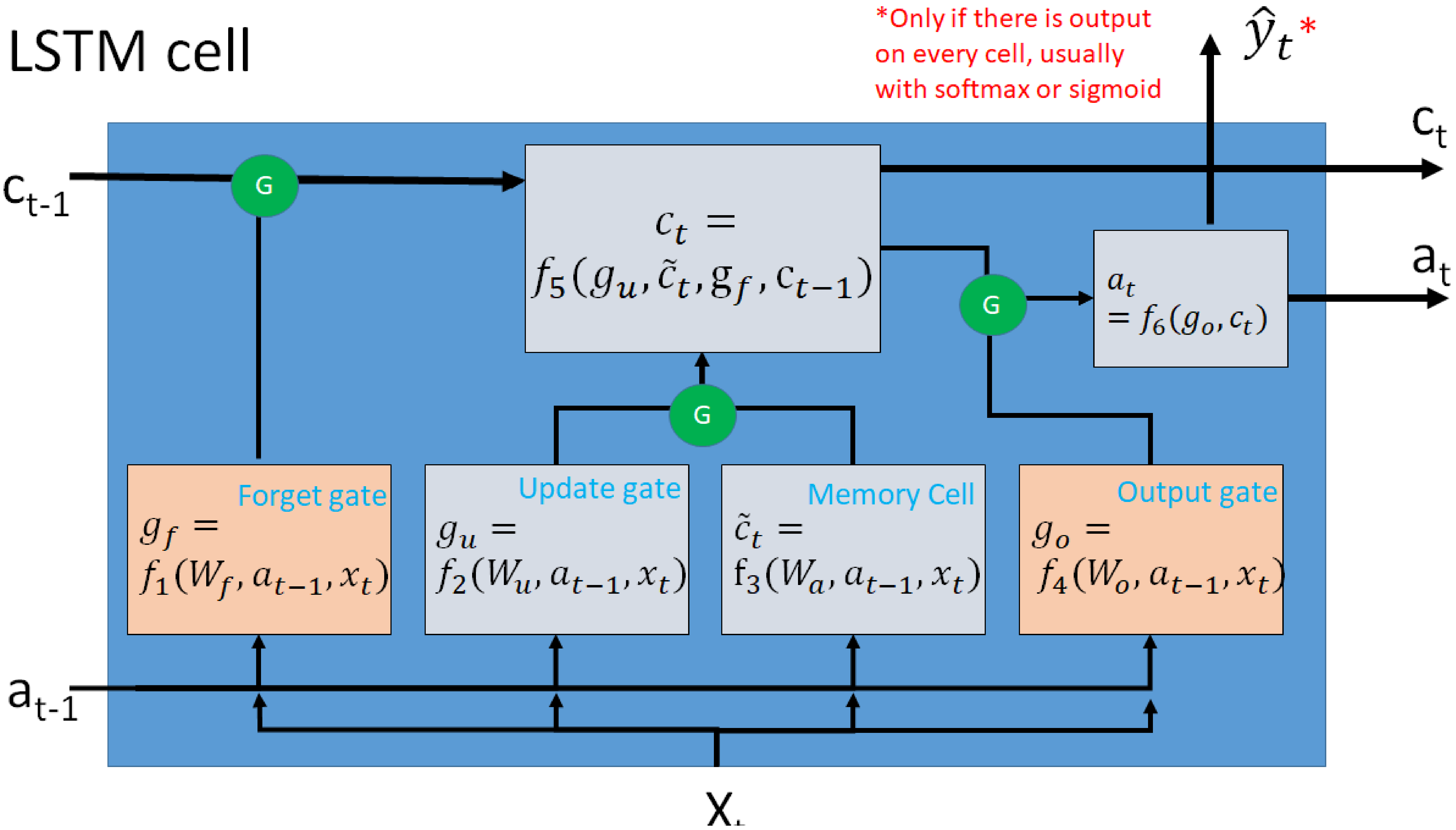


GRU cell



$\hat{y}_t$*
*Only if there is output on every cell, usually with softmax or sigmoid
Memory Cell
$a_t = h(g_u, \tilde{a}_t, a_{t-1})$
a_t
Candidate to memory
$\tilde{a}_t = f(W_a, a_{t-1}, x_t)$
Update gate
$g_u = g(W_u, a_{t-1}, x_t)$
a_{t-1}
X_t

LSTM cell



*Only if there is output on every cell, usually with softmax or sigmoid
$\hat{y}_t$*
c_t
a_t
c_{t-1}
G
$c_t = f_5(g_u, \tilde{c}_t, g_f, c_{t-1})$
G
$a_t = f_6(g_o, c_t)$
G
G
Forget gate
$g_f = f_1(W_f, a_{t-1}, x_t)$
Update gate
$g_u = f_2(W_u, a_{t-1}, x_t)$
Memory Cell
$\tilde{c}_t = f_3(W_a, a_{t-1}, x_t)$
Output gate
$g_o = f_4(W_o, a_{t-1}, x_t)$
a_{t-1}
x_t

No more vanishing gradients

- The `simpleRNN` cell can have gradient problems.
  - The weight matrix power t multiplies the other terms
- `GRU` and `LSTM` cells don't have vanishing gradient problems
  - Because of their gates
  - Don't have the weight matrices terms multiplying the rest
  - Exploding gradient problems are easier to solve

Usage in keras

```
# Import the layers
from tensorflow import keras
from tensorflow.keras.layers import GRU, LSTM
```

```
# Add the layers to a model
model.add(GRU(units=128, return_sequences=True, name='GRU layer'))
model.add(LSTM(units=64, return_sequences=False, name='LSTM layer'))
```

Let's practice!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

The Embedding layer

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

David Cecchini
Data Scientist

Why embeddings

Advantages:

- Reduce the dimension

```
one_hot = np.array((N, 100000))
embedd = np.array((N, 300))
```

- Dense representation
  - `king - man + woman = queen`
- Transfer learning

Disadvantages:

- Lots of parameters to train: training takes longer

How to use in keras

In keras:

```
from tensorflow.keras.layers import Embedding
model = Sequential()

# Use as the first layer
model.add(Embedding(input_dim=100000,
                    output_dim=300,
                    trainable=True,
                    embeddings_initializer=None,
                    input_length=120))
```

Transfer learning

Transfer learning for language models

- GloVE
- word2vec
- BERT

In keras:

```
from tensorflow.keras.initializers import Constant
model.add(Embedding(input_dim=vocabulary_size,
                    output_dim=embedding_dim,
                    embeddings_initializer=Constant(pre_trained_vectors))
```

Using GloVE pre-trained vectors

Official site: `https://nlp.stanford.edu/projects/glove/`

```
# Get hte GloVE vectors
def get_glove_vectors(filename="glove.6B.300d.txt"):
    # Get all word vectors from pre-trained model
    glove_vector_dict = {}
    with open(filename) as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = values[1:]
            glove_vector_dict[word] = np.asarray(coefs, dtype='float32')

    return glove_vector_dict
```

Using the GloVE on a specific task

```
# Filter GloVE vectors to specific task
def filter_glove(vocabulary_dict, glove_dict, wordvec_dim=300):
    # Create a matrix to store the vectors
    embedding_matrix = np.zeros((len(vocabulary_dict) + 1, wordvec_dim))
    for word, i in vocabulary_dict.items():
        embedding_vector = glove_dict.get(word)
        if embedding_vector is not None:
            # words not found in the glove_dict will be all-zeros.
            embedding_matrix[i] = embedding_vector

    return embedding_matrix
```

Let's practice!

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

Sentiment classification revisited

RECURRENT NEURAL NETWORKS (RNNS) FOR LANGUAGE MODELING WITH KERAS

David Cecchini
Data Scientist

Previous results

We had bad results with our initial model.

```
model = Sequential()
model.add(SimpleRNN(units=16, input_shape=(None, 1)))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.evaluate(x_test, y_test)
```

```
$[0.6991182165145874, 0.495]
```

Improving the model

To improve the model's performance, we can:

- Add the embedding layer
- Increase the number of layers
- Tune the parameters
- Increase vocabulary size
- Accept longer sentences with more memory cells

Avoiding overfitting

RNN models can overfit

- Test different batch sizes.
- Add `Dropout` layers.
- Add `dropout` and `recurrent_dropout` parameters on RNN layers.

```
# removes 20% of input to add noise
model.add(Dropout(rate=0.2))
# Removes 10% of input and memory cells respectively
model.add(LSTM(128, dropout=0.1, recurrent_dropout=0.1))
```

Extra: Convolution Layer

Not in the scope:

```
model.add(Embedding(vocabulary_size, wordvec_dim, ...))
model.add(Conv1D(num_filters=32, kernel_size=3, padding='same'))
model.add(MaxPooling1D(pool_size=2))
```

- Convolution layer do feature selection on the embedding vector
- Achieves state-of-the-art results in many NLP problems

One example model

```
model = Sequential()
model.add(Embedding(  vocabulary_size, wordvec_dim, trainable=True,
                      embeddings_initializer=Constant(glove_matrix),
                      input_length=max_text_len, name="Embedding"))
model.add(Dense(wordvec_dim, activation='relu', name="Dense1"))
model.add(Dropout(rate=0.25))
model.add(LSTM(64, return_sequences=True, dropout=0.15, name="LSTM"))
model.add(GRU(64, return_sequences=False, dropout=0.15, name="GRU"))
model.add(Dense(64, name="Dense2"))
model.add(Dropout(rate=0.25))
model.add(Dense(32, name="Dense3"))
model.add(Dense(1, activation='sigmoid', name="Output"))
```

Let's practice!