

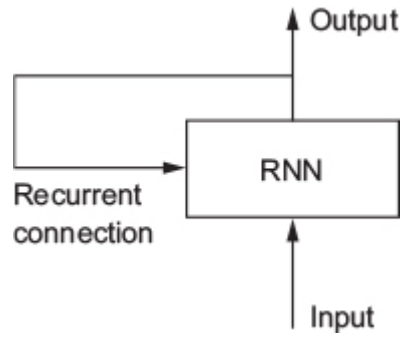
Recurrent NNs (RNNs)

PUBH 8475/Stat 8056

Based on “Deep Learning with R”

<https://www.manning.com/books/deep-learning-with-r>

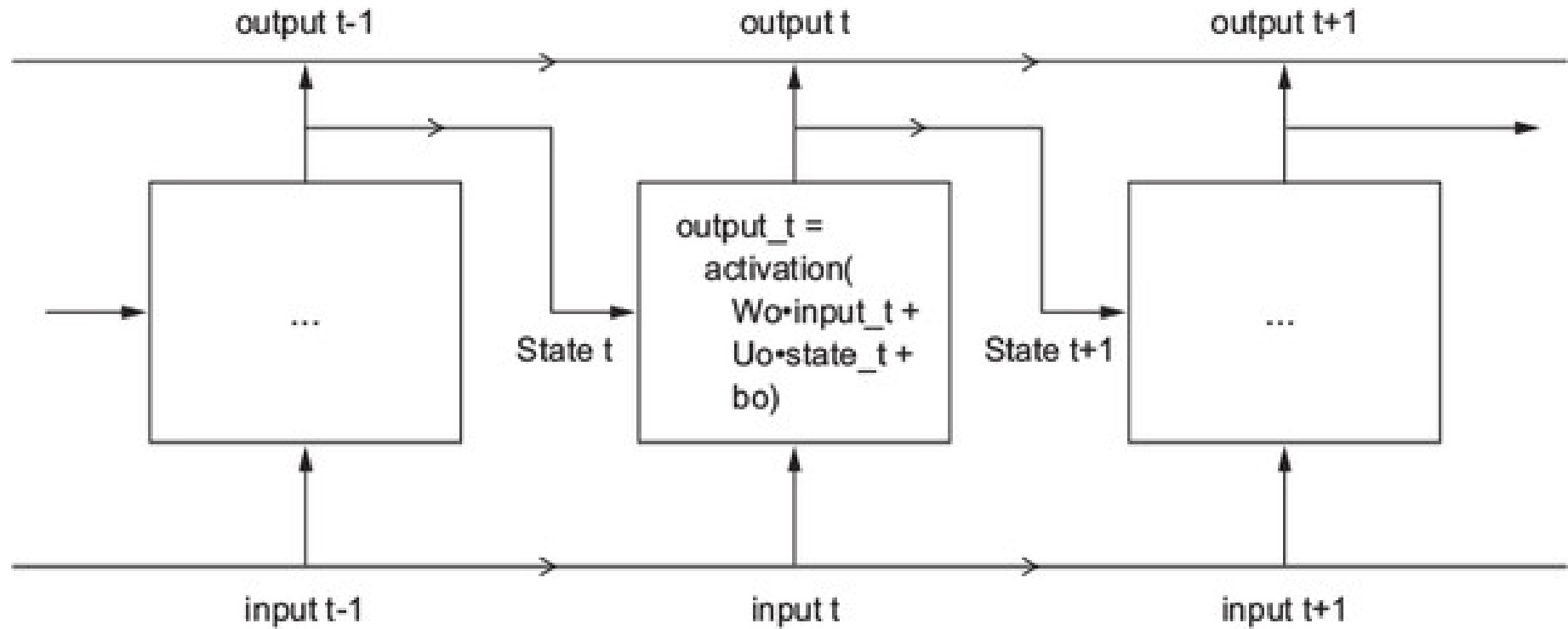
(including source code)



```
state_t <- 0
for (input_t in input_sequence) {
  output_t <- activation(dot(W, input_t) + dot(U, state_t) + b)
  state_t <- output_t
}
```

```
timesteps <- 100
input_features <- 32
output_features <- 64
random_array <- function(dim) {
  array(runif(prod(dim)), dim = dim)
}
inputs <- random_array(dim = c(timesteps, input_features))
state_t <- rep_len(0, length = c(output_features))
W <- random_array(dim = c(output_features, input_features))
U <- random_array(dim = c(output_features, output_features))
b <- random_array(dim = c(output_features, 1))
output_sequence <- array(0, dim = c(timesteps, output_features))
for (i in 1:nrow(inputs)) {
  input_t <- inputs[i,]
  output_t <- tanh(as.numeric((W %*% input_t) + (U %*% state_t) + b))
  output_sequence[i,] <- as.numeric(output_t)
  state_t <- output_t
}
```

Figure 6.10. The starting point of an LSTM layer: a simple RNN



Simple RNN:

```
output_t <- activation(dot(W, input_t) + dot(U, state_t) + b)
state_t+1 <- output_t
```

Does not have memory of long-range dependence → Long Short-Term Memory (**LSTM**):

```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + bo)
```

#forget gate:

```
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
```

#input gate:

```
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
```

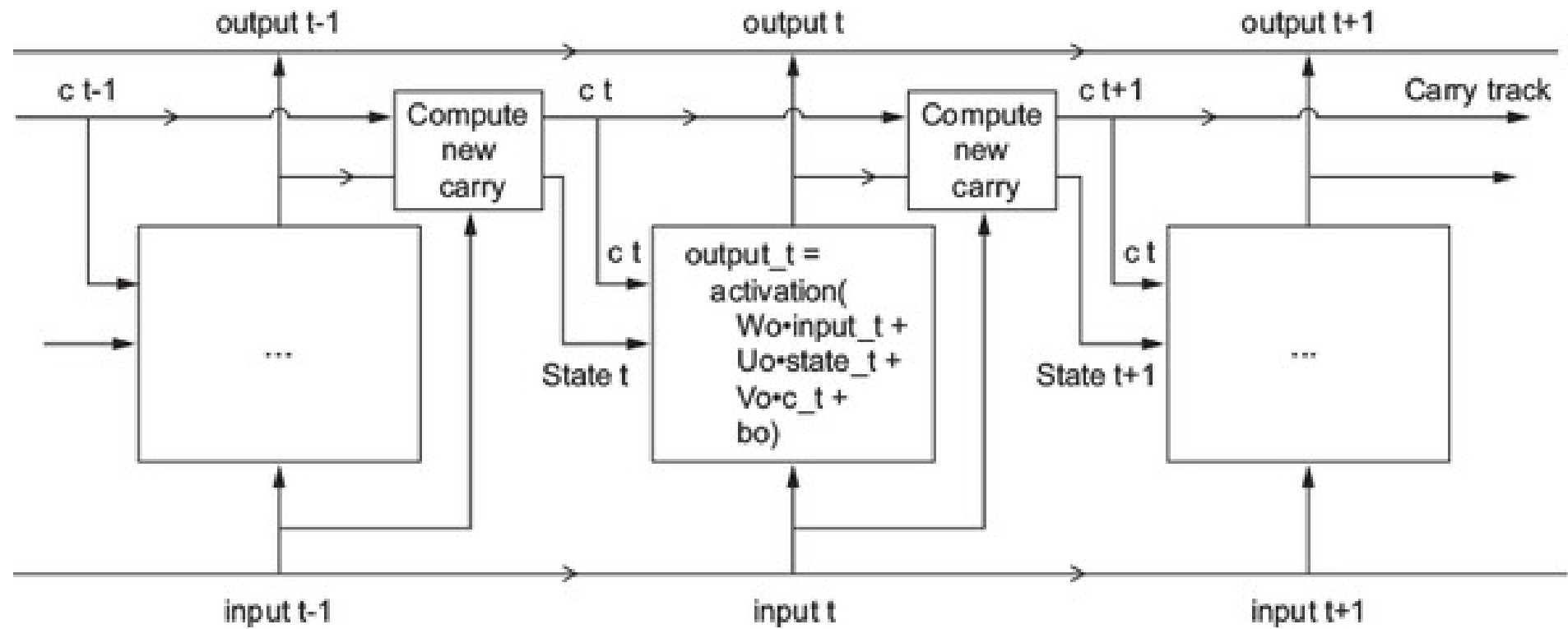
```
cc_t = tanh(dot(state_t, Uk) + dot(input_t, Wk) + bc)
```

```
c_t+1 = i_t * cc_t + c_t * f_t
```

```
state_t+1=tanh(c_t+1)*output_t
```

activation: usually sigmoid

Figure 6.12. Anatomy of an LSTM



```
library(keras)
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 32) %>%
  layer_simple_rnn(units = 32)
```

```
> summary(model)
```

Layer (type)	Output Shape	Param #
embedding_22 (Embedding)	(None, None, 32)	320000
simplernn_10 (SimpleRNN)	(None, 32)	2080

Total params: 322,080
 Trainable params: 322,080
 Non-trainable params: 0

320000 = 10000 * 32

#Q: why 2080? = (32+32+1)*32

```
library(keras)
max_features <- 10000
maxlen <- 500

cat("Loading data...\n")
imdb <- dataset_imdb(num_words = max_features)
c(c(input_train, y_train), c(input_test, y_test)) %<-% imdb
cat(length(input_train), "train sequences\n")
#25000 train sequences
cat(length(input_test), "test sequences")
#25000 test sequences
cat("Pad sequences (samples x time)\n")
input_train <- pad_sequences(input_train, maxlen = maxlen)
input_test <- pad_sequences(input_test, maxlen = maxlen)
cat("input_train shape:", dim(input_train), "\n")
#input_train shape: 25000 500
cat("input_test shape:", dim(input_test), "\n")
#input_test shape: 25000 500
```

```
model <- keras_model_sequential() %>%  
  layer_embedding(input_dim = max_features, output_dim = 32) %>%  
  layer_simple_rnn(units = 32) %>%  
  layer_dense(units = 1, activation = "sigmoid")
```

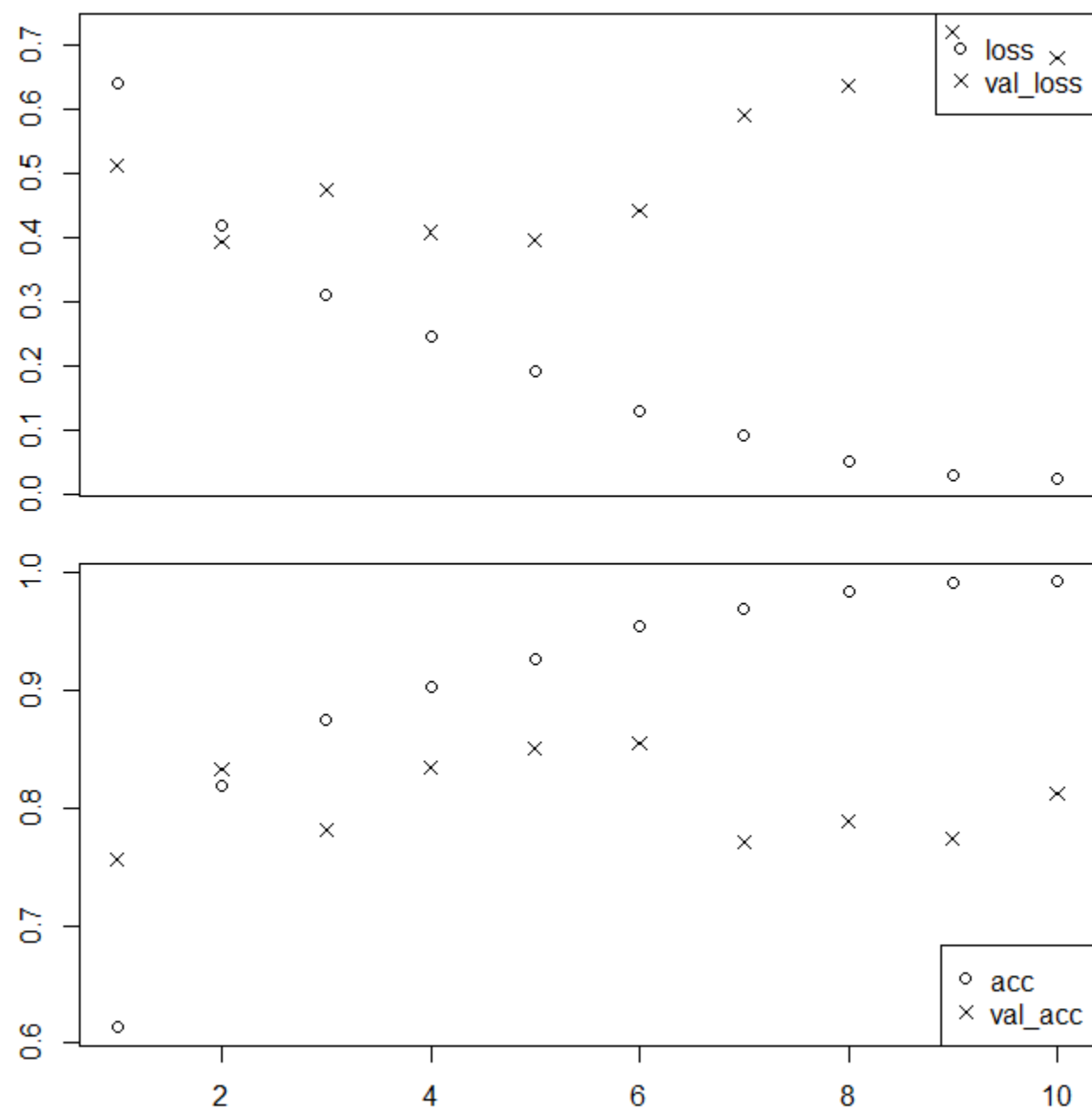
```
model %>% compile(  
  optimizer = "rmsprop",  
  loss = "binary_crossentropy",  
  metrics = c("acc")  
)
```

```
history <- model %>% fit(  
  input_train, y_train,  
  epochs = 10,  
  batch_size = 128,  
  validation_split = 0.2  
)
```

> model
Model
Model: "sequential_5"

Layer (type)	Output Shape	Param #
=====		
embedding_2 (Embedding)	(None, None, 32)	320000
simple_rnn_1 (SimpleRNN)	(None, 32)	2080
dense_9 (Dense)	(None, 1)	33
=====		

Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0



```
model <- keras_model_sequential() %>%  
  layer_embedding(input_dim = max_features, output_dim = 32) %>%  
  layer_lstm(units = 32) %>%  
  layer_dense(units = 1, activation = "sigmoid")
```

```
model %>% compile(  
  optimizer = "rmsprop",  
  loss = "binary_crossentropy",  
  metrics = c("acc")  
)
```

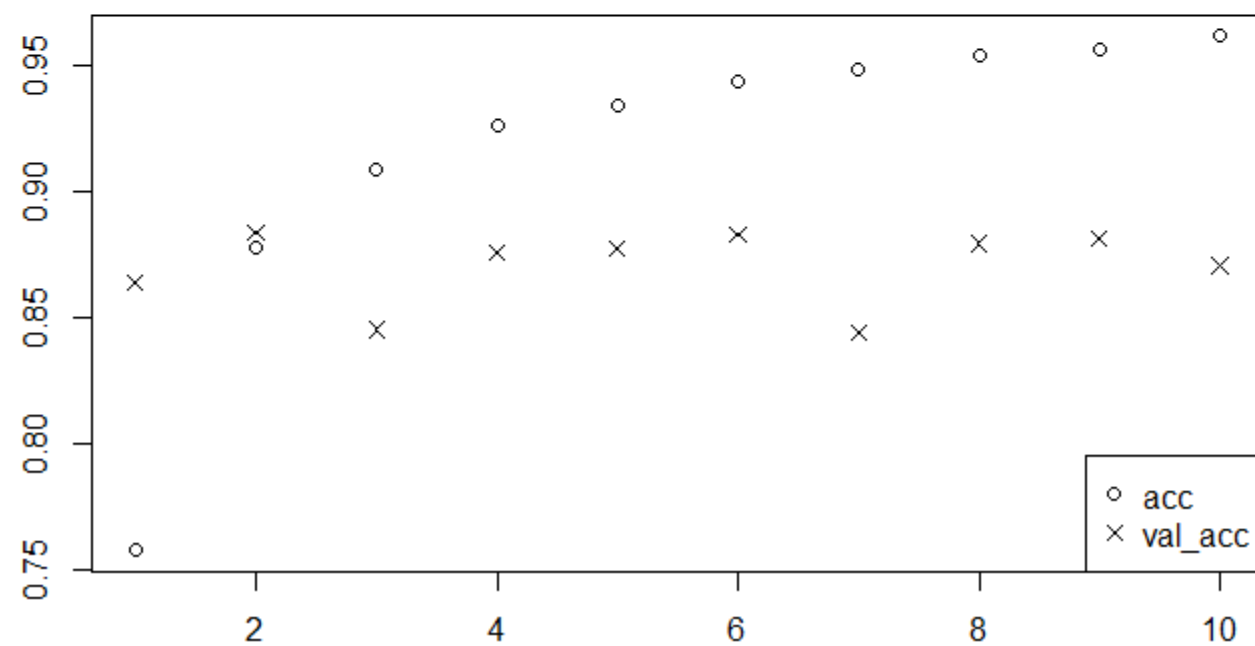
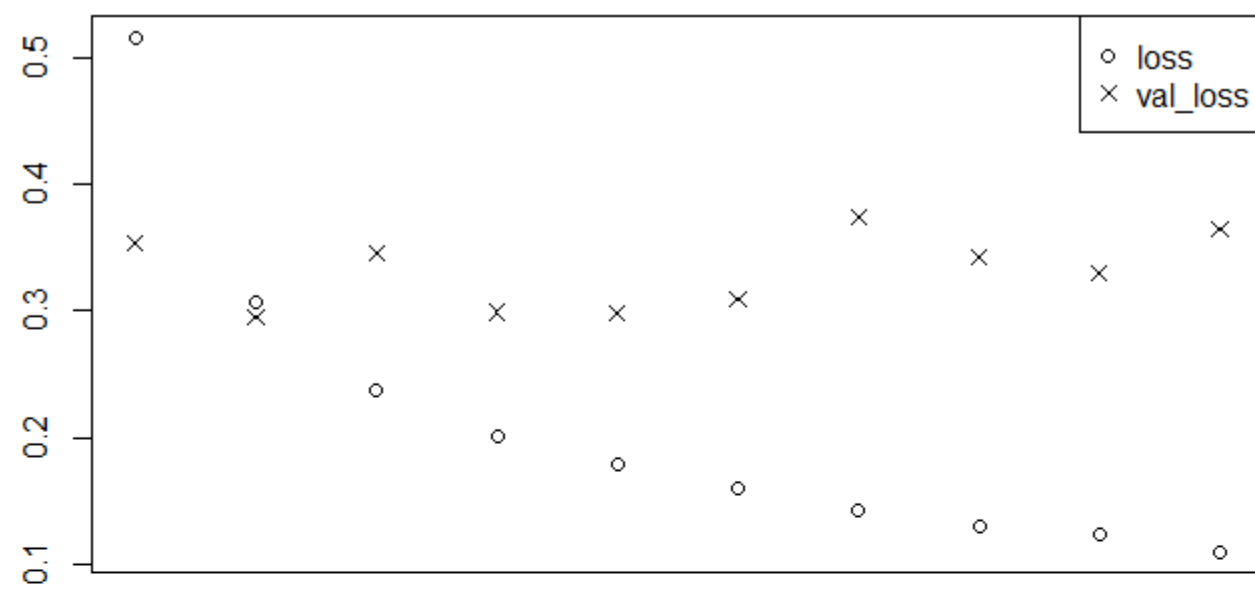
```
history <- model %>% fit(  
  input_train, y_train,  
  epochs = 10,  
  batch_size = 128,  
  validation_split = 0.2  
)
```

```
> model
Model
Model: "sequential_6"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, None, 32)	320000
lstm_1 (LSTM)	(None, 32)	8320
dense_10 (Dense)	(None, 1)	33

Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0

#Q: why 8320? $= (32+32+1) * 4 * 32$



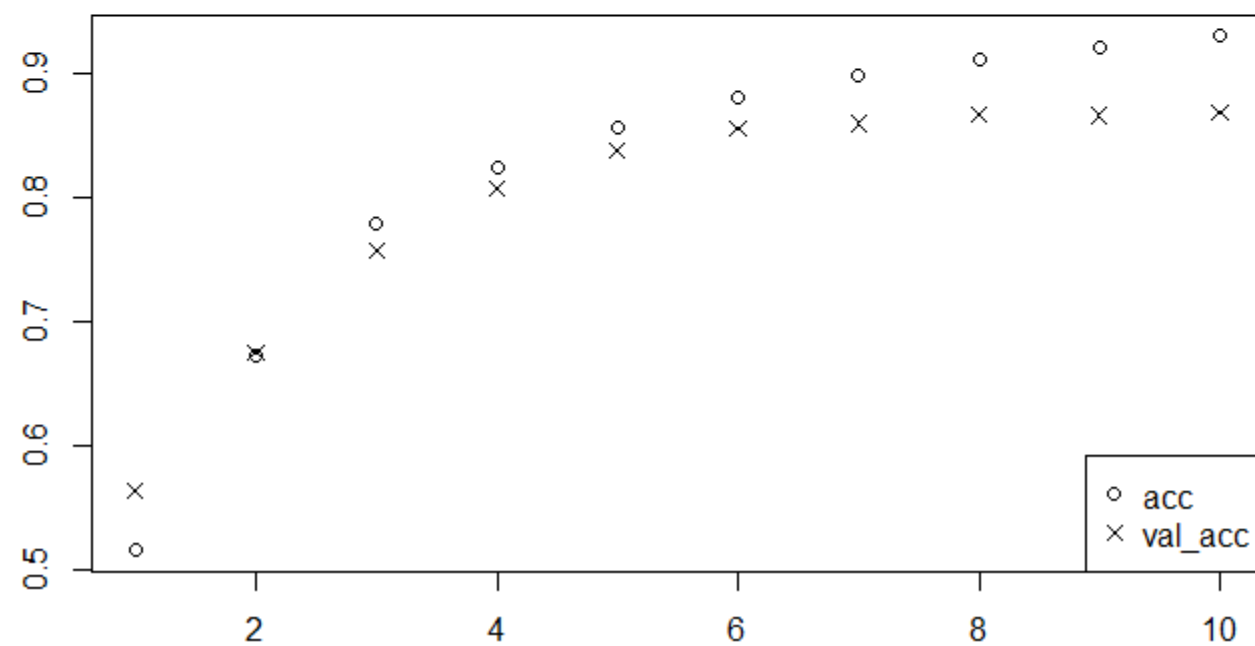
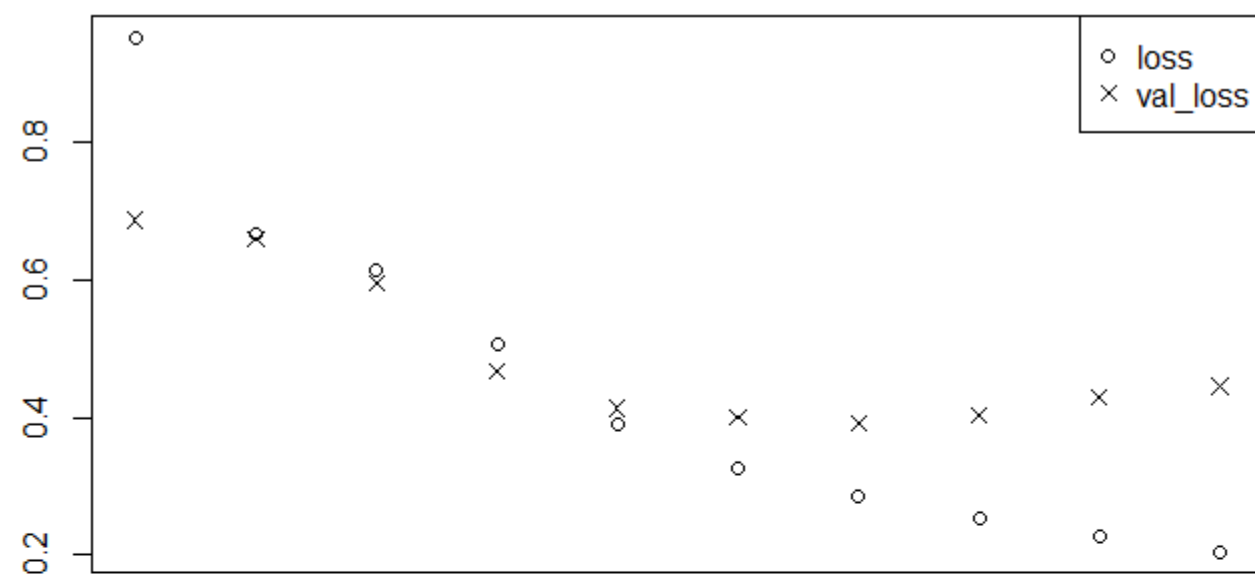
##1D CNN:

```
model <- keras_model_sequential() %>%  
  layer_embedding(input_dim = max_features, output_dim = 128,  
                  input_length = maxlen) %>%  
  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%  
  layer_max_pooling_1d(pool_size = 5) %>%  
  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%  
  layer_global_max_pooling_1d() %>%  
  layer_dense(units = 1)  
summary(model)  
model %>% compile(  
  optimizer = optimizer_rmsprop(lr = 1e-4),  
  loss = "binary_crossentropy",  
  metrics = c("acc")  
)  
history <- model %>% fit(  
  input_train, y_train,  
  epochs = 10,  
  batch_size = 128,  
  validation_split = 0.2  
)
```

> summary(model)
Model: "sequential_6"

Layer (type)	Output Shape	Param #
=====		
embedding_6 (Embedding)	(None, 500, 128)	1280000
conv1d_3 (Conv1D)	(None, 494, 32)	28704
max_pooling1d_1 (MaxPooling1D)	(None, 98, 32)	0
conv1d_2 (Conv1D)	(None, 92, 32)	7200
global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 32)	0
dense_15 (Dense)	(None, 1)	33
=====		

Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0



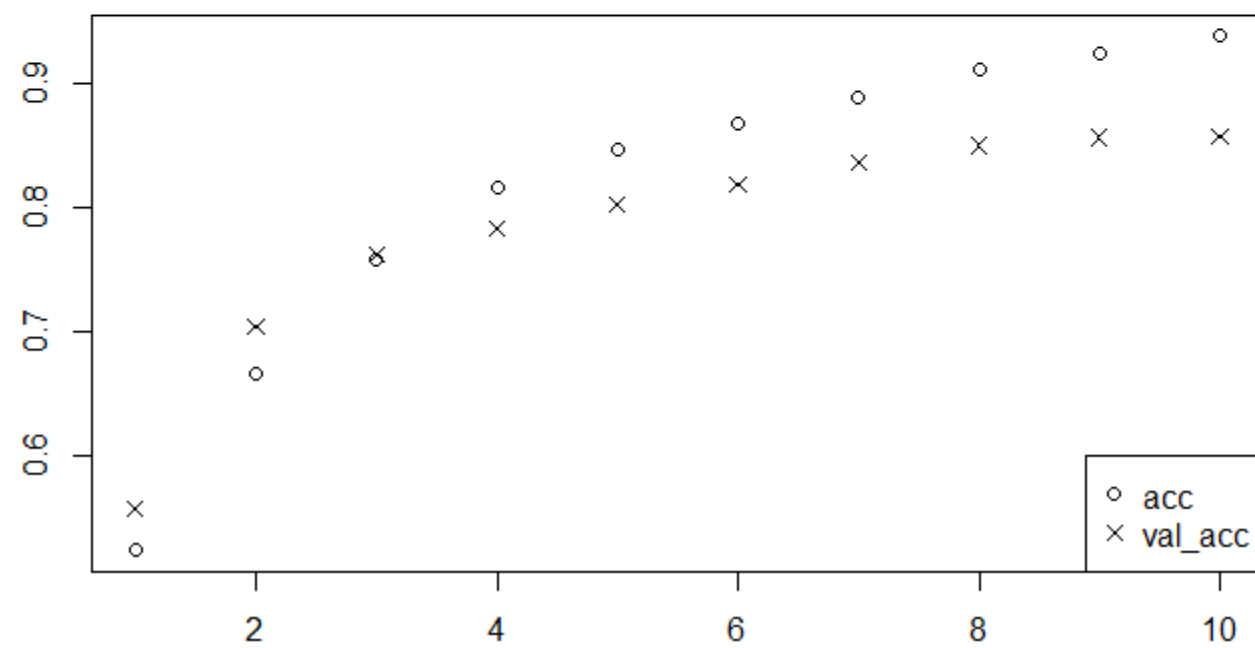
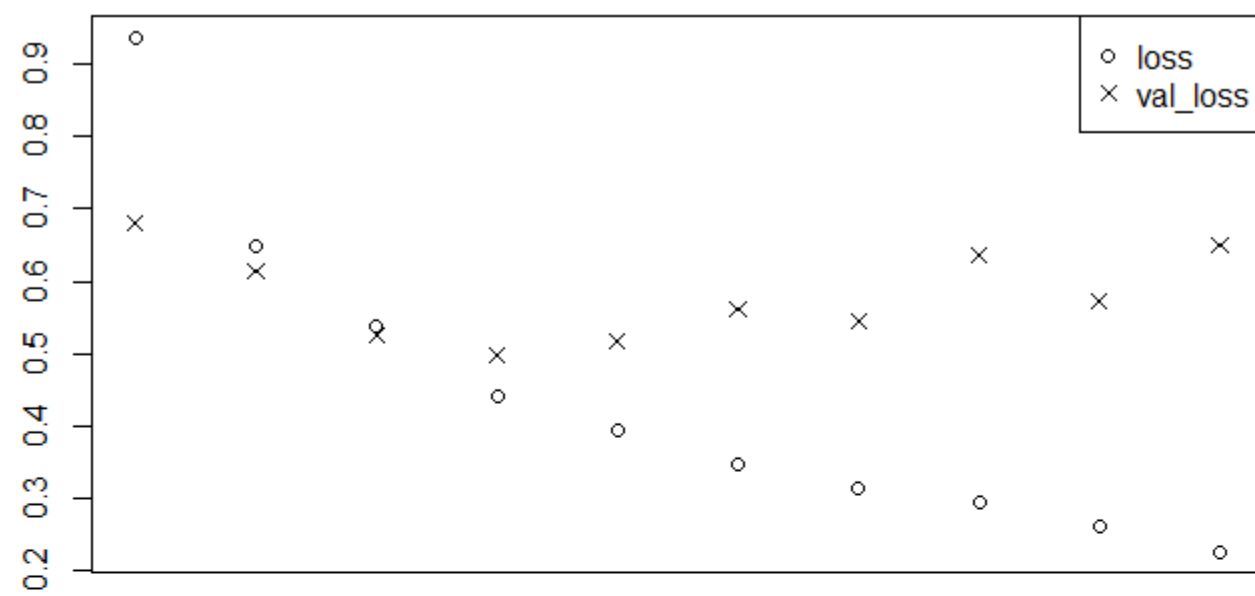
##1D CNN + RNN:

```
model <- keras_model_sequential() %>%  
  layer_embedding(input_dim = max_features, output_dim = 128,  
    input_length = maxlen) %>%  
  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%  
  layer_max_pooling_1d(pool_size = 5) %>%  
  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%  
  layer_gru(units=32) %>%  
  layer_dense(units = 1)  
summary(model)  
model %>% compile(  
  optimizer = optimizer_rmsprop(lr = 1e-4),  
  loss = "binary_crossentropy",  
  metrics = c("acc")  
)  
history <- model %>% fit(  
  input_train, y_train,  
  epochs = 10,  
  batch_size = 128,  
  validation_split = 0.2  
)
```

> summary(model)
Model: "sequential_7"

Layer (type)	Output Shape	Param #
=====		
embedding_7 (Embedding)	(None, 500, 128)	1280000
conv1d_5 (Conv1D)	(None, 494, 32)	28704
max_pooling1d_2 (MaxPooling1D)	(None, 98, 32)	0
conv1d_4 (Conv1D)	(None, 92, 32)	7200
gru (GRU)	(None, 32)	6240
dense_16 (Dense)	(None, 1)	33
=====		

Total params: 1,322,177
Trainable params: 1,322,177
Non-trainable params: 0



- GRU (Gated Recurrent Unit): similar to LSTM; less computationally demanding and less powerful?
 - R: `layer_gru(units = 32, ...)`
- Can use dropout
 - `layer_gru(units = 32, dropout = 0.1, recurrent_dropout = 0.5)`
- Using bidirectional RNNs:
 - `bidirectional(
 layer_lstm(units = 32)
)`
- RNNs: expensive/difficult to train
 - Vanishing/exploding gradients for simple RNNs
- For short seqs, CNNs or FFNs may work better (and being cheaper)!
- Applications:
 - article/music/image generation,
 - Google Neural Machine Translation (GNMT),
Microsoft: Awadalla et al (2018). Achieving Human Parity on Automatic Chinese to English News Translation. arXiv:1803.05567
 - image captioning, ...
- YouTube: “LSTM is dead. Long Live Transformers!”
ChatGPT,....

RESEARCH ARTICLE

Predicting enhancer-promoter interaction from genomic sequence with deep neural networks

Shashank Singh¹, Yang Yang², Barnabás Póczos¹, Jian Ma^{2,*}

¹ Machine Learning Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

² Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

* Correspondence: jianma@cs.cmu.edu

Predicting enhancer-promoter interaction

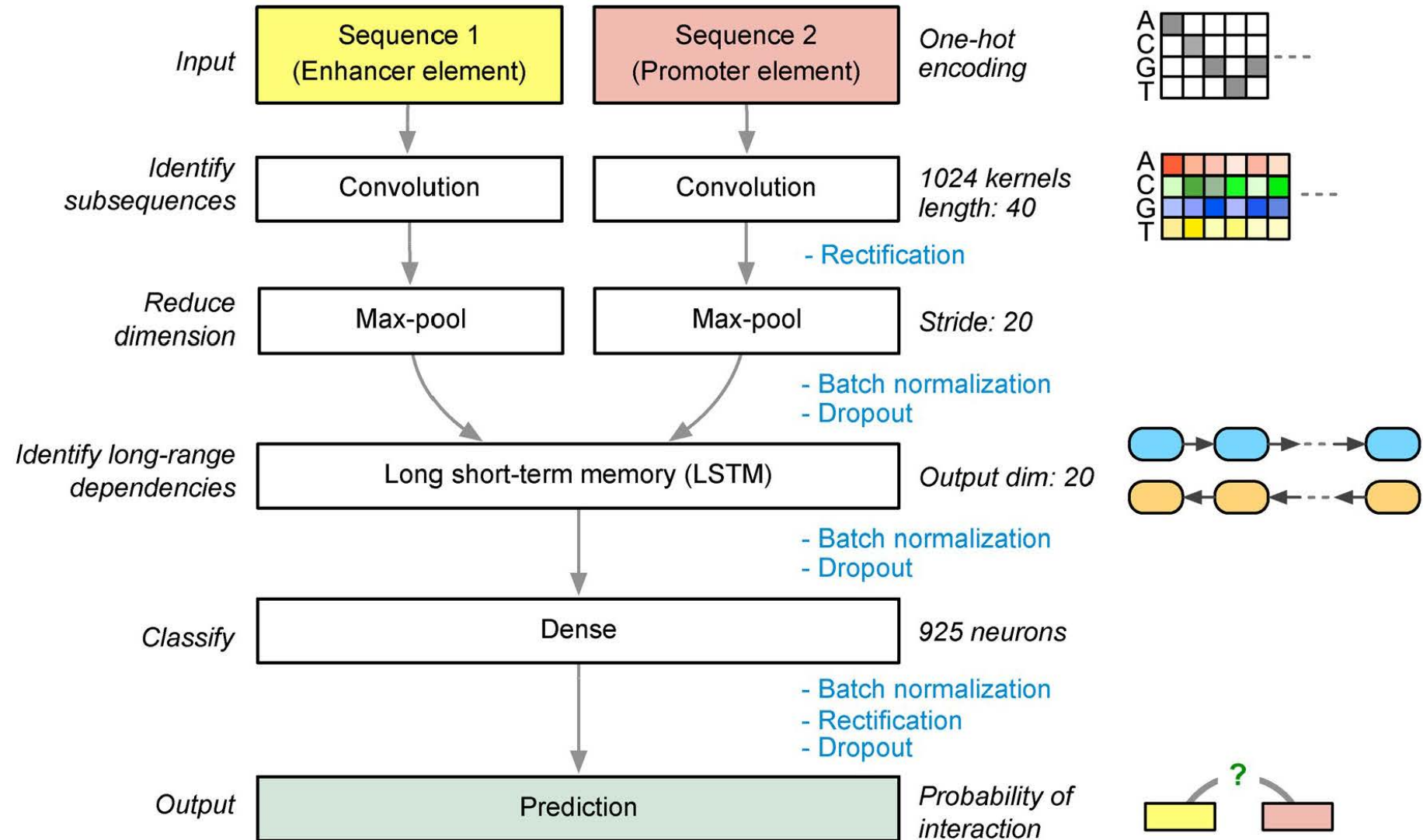


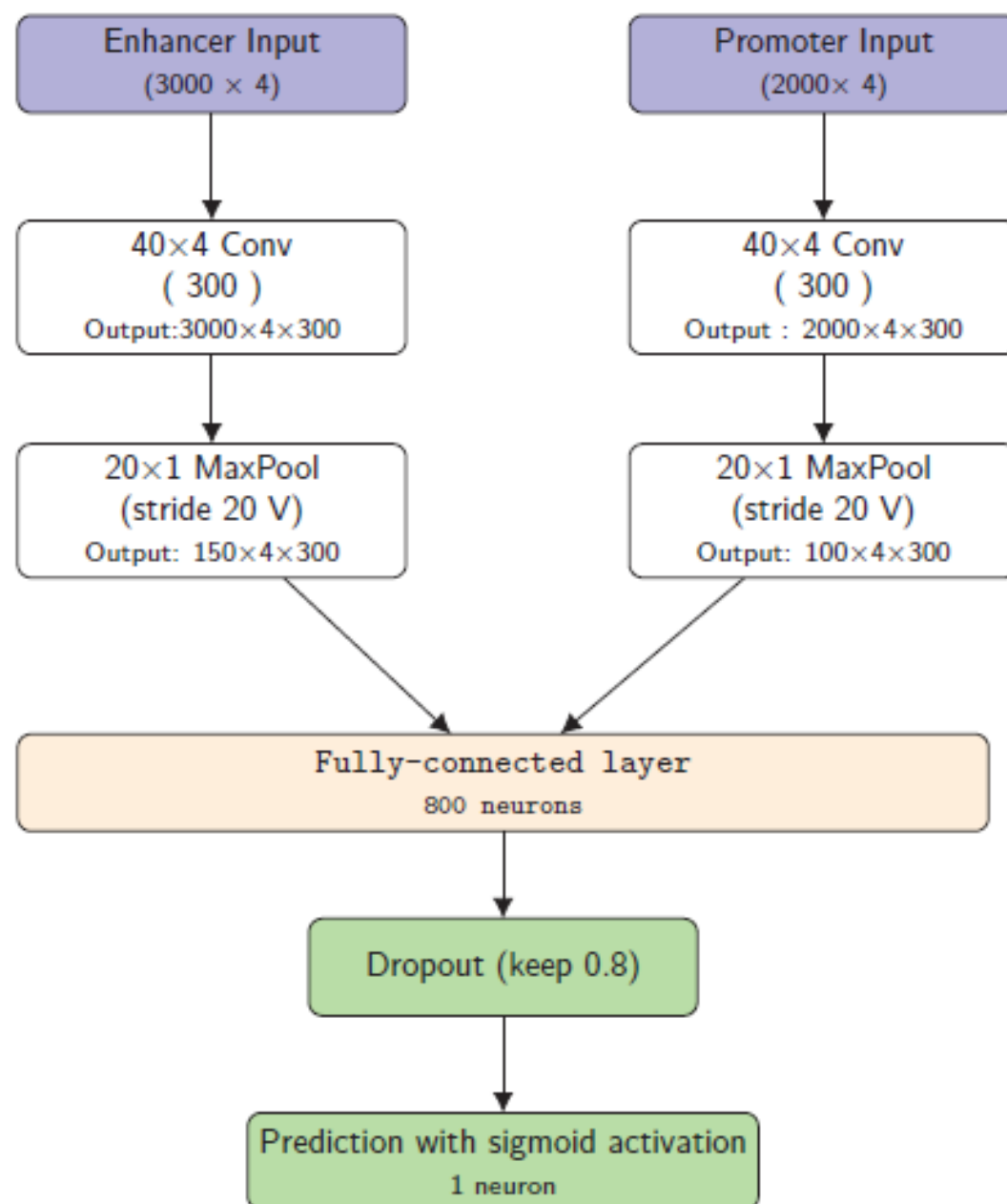
Figure 1. Diagram of our deep learning model SPEID to predict enhancer-promoter interactions based on sequences only. Key steps involving rectification, batch normalization, and dropout are annotated. Note that the final output step is essentially a logistic regression in SPEID which provides a probability to indicate whether the input enhancer element and promoter element would interact.

Genome analysis

A simple convolutional neural network for prediction of enhancer–promoter interactions with DNA sequence data

Zhong Zhuang¹, Xiaotong Shen² and Wei Pan^{3,*}

¹Department of Electrical and Computer Engineering, ²School of Statistics and ³Division of Biostatistics, University of Minnesota, Minneapolis, MN 55455, USA





Article

Local Epigenomic Data are more Informative than Local Genome Sequence Data in Predicting Enhancer-Promoter Interactions Using Neural Networks



Mengli Xiao ¹, Zhong Zhuang ² and Wei Pan ^{1,*}

¹ Division of Biostatistics, University of Minnesota, Minneapolis, MN 55455, USA; xiaox345@umn.edu

² Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA; zhuan143@umn.edu

* Correspondence: panxx014@umn.edu; Tel.: +01-612-626-2705



updates

Received: 29 November 2019; Accepted: 26 December 2019; Published: 29 December 2019

check for