CSI5180. Machine Learning for Bioinformatics Applications

Deep learning — practical issues

by

Marcel Turcotte

Version **November 19, 2019**

Preamble

Preamble

Deep learning — practical issues

In this last lecture deep learning, we consider practical issues when using existing tools and libraries.

General objective :

- **Discuss** the pitfalls, limitations, and practical considerations when using deep learning algorithms.

Learning objectives

- **Discuss** the pitfalls, limitations, and practical considerations when using deep learning algorithms.
- **Explain** what is a dropout layer
- **Discuss** further mechanisms to regularize deep networks

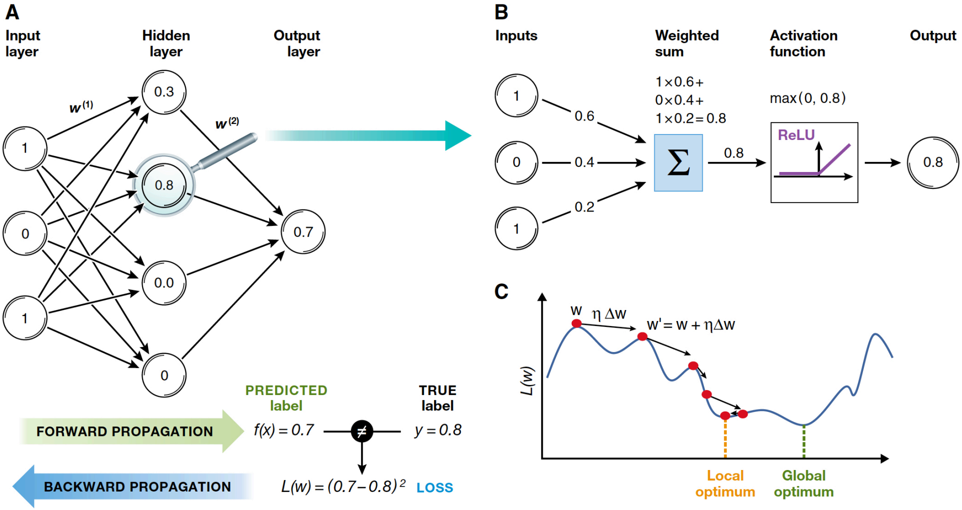
Reading:

- Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. Deep learning for computational biology. *Mol Syst Biol* **12**(7):878, 07 2016.

Plan

As mentioned previously

Overview



Source: [1] Box 1

Summary

- In a **dense** layer, **all** the neurons are connected to **all** the neurons from the previous layer.

Summary

- In a **dense** layer, **all** the neurons are connected to **all** the neurons from the previous layer.
  - The number of parameters grows exponentially with each additional layer, making it nearly impossible to create deep networks.

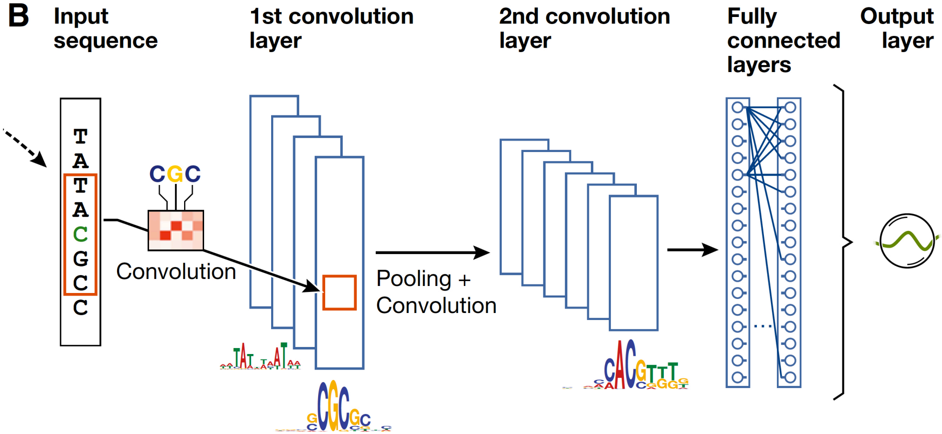
Summary

- In a **dense** layer, **all** the neurons are connected to **all** the neurons from the previous layer.
  - The number of parameters grows exponentially with each additional layer, making it nearly impossible to create deep networks.
- **Local connectivity**. In a **convolutional layer** each neuron is connected to a small number of neurons from the previous layer. This small rectangular region is called the **receptive field**.

Summary

- In a **dense** layer, **all** the neurons are connected to **all** the neurons from the previous layer.
  - The number of parameters grows exponentially with each additional layer, making it nearly impossible to create deep networks.
- **Local connectivity**. In a **convolutional layer** each neuron is connected to a small number of neurons from the previous layer. This small rectangular region is called the **receptive field**.
- **Parameter sharing**. All the neurons in a given **feature map** of a **convolutional layer** share the same **kernel** (**filter**).

Convolutional layer (Conv1D)



Source: [1] Figure 2B

Convolutional layer

- Contrary to **Dense layers**, **Conv1D layers** preserve the identity of the monomers (nucleotides or amino acids), which are seen as channels.

Convolutional layer

- Contrary to **Dense layers**, **Conv1D layers** preserve the identity of the monomers (nucleotides or amino acids), which are seen as channels.
- **Convolutional Neural Networks** are able to detect patterns **irrespective** of their location in the input.

Convolutional layer

- Contrary to **Dense layers**, **Conv1D layers** preserve the identity of the monomers (nucleotides or amino acids), which are seen as channels.
- **Convolutional Neural Networks** are able to detect patterns **irrespective** of their location in the input.
  - **Pooling** makes the network less sensitive to small translations.

Convolutional layer

- Contrary to **Dense layers**, **Conv1D layers** preserve the identity of the monomers (nucleotides or amino acids), which are seen as channels.
- **Convolutional Neural Networks** are able to detect patterns **irrespective** of their location in the input.
  - **Pooling** makes the network less sensitive to small translations.
  - In bioinformatics, **CNN** networks are ideally suited to detect local (sequence) motifs, independent of their position within the input (sequence). They are also the most prevalent architecture.

Summary

- **Recurrent networks (RNN)** and **Long Short-Term Memory** (**LSTM**) can process input sequences of varying length.

Summary

- **Recurrent networks (RNN)** and **Long Short-Term Memory** (**LSTM**) can process input sequences of varying length.
  - Literature suggests that RNNs are more difficult to train than other architectures.

Regularization

Dropout

- Hinton and colleagues say that dropout layers are "**preventing co-adaptation**".

```
model = keras.models.Sequential([
    ...
    Dropout(0.5),
    ...
])
```

Dropout

- Hinton and colleagues say that dropout layers are "**preventing co-adaptation**".
- During **training**, each input unit in a dropout layer has probability p of being ignored (set to 0).

```
model = keras.models.Sequential([
    ...
    Dropout(0.5),
    ...
])
```

Dropout

- Hinton and colleagues say that dropout layers are "**preventing co-adaptation**".
- During **training**, each input unit in a dropout layer has probability p of being ignored (set to 0).
  - According to [3] §11:

```
model = keras.models.Sequential([
    ...
    Dropout(0.5),
    ...
])
```

Dropout

- Hinton and colleagues say that dropout layers are "**preventing co-adaptation**".
- During **training**, each input unit in a dropout layer has probability p of being ignored (set to 0).
  - According to [3] §11:
    - 20-30% is a typical value of p **convolution networks**;

```
model = keras.models.Sequential([
    ...
    Dropout(0.5),
    ...
])
```

Dropout

- Hinton and colleagues say that dropout layers are "**preventing co-adaptation**".
- During **training**, each input unit in a dropout layer has probability p of being ignored (set to 0).
  - According to [3] §11:
    - 20-30% is a typical value of p **convolution networks**;
    - whereas, 40-50% is a typical of p for **recurrent networks**.

```
model = keras.models.Sequential([
    ...
    Dropout(0.5),
    ...
])
```

Dropout

- Hinton and colleagues say that dropout layers are "**preventing co-adaptation**".
- During **training**, each input unit in a dropout layer has probability p of being ignored (set to 0).
  - According to [3] §11:
    - 20-30% is a typical value of p **convolution networks**;
    - whereas, 40-50% is a typical of p for **recurrent networks**.
- **Dropout layers** can make the network converging more slowly. However, the resulting network is expected to make **fewer generalization errors**.

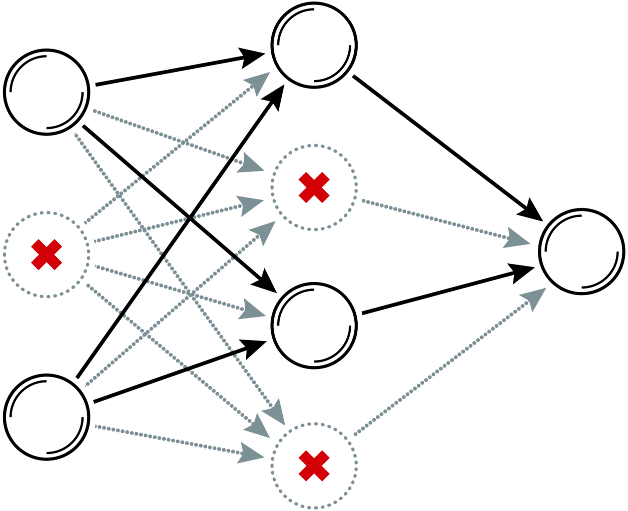
```
model = keras.models.Sequential([
    ...
    Dropout(0.5),
    ...
])
```

Dropout

- Hinton and colleagues say that dropout layers are "**preventing co-adaptation**".
- During **training**, each input unit in a dropout layer has probability p of being ignored (set to 0).
  - According to [3] §11:
    - 20-30% is a typical value of p **convolution networks**;
    - whereas, 40-50% is a typical of p for **recurrent networks**.
- **Dropout layers** can make the network converging more slowly. However, the resulting network is expected to make **fewer generalization errors**.
- https://keras.io/layers/core/

```
model = keras.models.Sequential([
    ...
    Dropout(0.5),
    ...
])
```

Dropout



Source: [1] Figure 5F

Regularizers

- Applying penalties on layer parameters
- https://keras.io/regularizers/

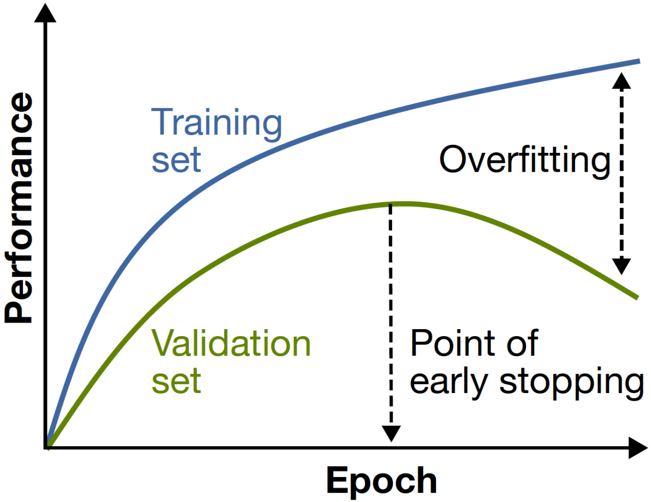
```
# other import directives are here
from keras import regularizers

model = Sequential()
model.add(Dense(32, input_shape=(16,)))
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01)))
```

Available penalties

```
keras.regularizers.l1(0.)
keras.regularizers.l2(0.)
keras.regularizers.l1_l2(l1=0.01, l2=0.01)
```

Early stopping



Source: [1] Figure 5E

Hyperparameters

Optimizers

- An **optimizer** should be **fast** and should ideally guide the solution towards a **“good” local optimum** (or better, a global optimum).

Optimizers

- An **optimizer** should be **fast** and should ideally guide the solution towards a **“good” local optimum** (or better, a global optimum).
- **Momentum**

Optimizers

- An **optimizer** should be **fast** and should ideally guide the solution towards a **“good” local optimum** (or better, a global optimum).
- **Momentum**
  - Momentum methods keep track of the previous gradients and this information is used to update the weights.

$$m = \beta m - \eta \nabla_\theta J(\theta)$$

$$\theta = \theta + m$$

Optimizers

- An **optimizer** should be **fast** and should ideally guide the solution towards a **“good” local optimum** (or better, a global optimum).
- **Momentum**
  - Momentum methods keep track of the previous gradients and this information is used to update the weights.

$$m = \beta m - \eta \nabla_\theta J(\theta)$$

$$\theta = \theta + m$$

  - Momentum methods can **escape plateau** more effectively.

Optimizers

- An **optimizer** should be **fast** and should ideally guide the solution towards a **"good" local optimum** (or better, a global optimum).
- **Momentum**
  - Momentum methods keep track of the previous gradients and this information is used to update the weights.

$$m = \beta m - \eta \nabla_\theta J(\theta)$$

$$\theta = \theta + m$$

  - Momentum methods can **escape plateau** more effectively.
  - **Nesterov Accelerated Gradient**, **AdaGrad**, **RMSProp**, **Adam** and **Nadam**.

Optimizers

- An **optimizer** should be **fast** and should ideally guide the solution towards a **"good" local optimum** (or better, a global optimum).
- **Momentum**
  - Momentum methods keep track of the previous gradients and this information is used to update the weights.

$$m = \beta m - \eta \nabla_\theta J(\theta)$$

$$\theta = \theta + m$$

  - Momentum methods can **escape plateau** more effectively.
  - **Nesterov Accelerated Gradient**, **AdaGrad**, **RMSProp**, **Adam** and **Nadam**.
  - **Adam** is a good default choice.

Loss function

- **Regression**
  - **mean_squared_error** (MSE) or **mean_absolute_error** (MAE)
- **Classification**
  - **Binary classification** : binary_crossentropy
  - **Multiclass classification** : categorical_crossentropy
- https://keras.io/losses/

```
from keras import losses

model.compile(loss=losses.mean_squared_error, optimizer='sgd')
```

Output layer activation

- **Regression** [3] Table 10.1:
  - ReLU/softplus (if positive outputs)
  - logistic/tanh (if bounded outputs)
- **Classification**
  - **Binary classification** : logistic
  - **Multiclass classification** : softmax
- https://keras.io/activations/

```
model = keras.models.Sequential([
    ...
    Dense(64, activation="relu"),
    ...
])
```

Hyperparameters

Name	Range	Default value
Learning rate	0.1, 0.01, 0.001, 0.0001	0.01
Batch size	64, 128, 256	128
Momentum rate	0.8, 0.9, 0.95	0.9
Weight initialization	Normal, Uniform, Glorot uniform	Glorot uniform
Per-parameter adaptive learning rate methods	RMSprop, Adagrad, Adadelta, Adam	Adam
Batch normalization	Yes, no	Yes
Learning rate decay	None, linear, exponential	Linear (rate 0.5)
Activation function	Sigmoid, Tanh, ReLU, Softmax	ReLU
Dropout rate	0.1, 0.25, 0.5, 0.75	0.5
L1, L2 regularization	0, 0.01, 0.001	

Source: [1] Table 2

Keras

Keras

```
model = keras.models.Sequential([
    Conv2D(64, 7, ..., input_shape=[28, 28, 1]),
    MaxPooling2D(2),
    Conv2D(128, 3, activation="relu", padding="same"),
    Conv2D(128, 3, activation="relu", padding="same"),
    MaxPooling2D(2),
    Conv2D(256, 3, activation="relu", padding="same"),
    Conv2D(256, 3, activation="relu", padding="same"),
    MaxPooling2D(2),
    Flatten(),
    Dense(128, activation="relu"),
    Dropout(0.5),
    Dense(64, activation="relu"),
    Dropout(0.5),
    Dense(10, activation="softmax")
])
```

[3] §14:

Further considerations

Further considerations

We obviously barely scratched the surface of deep learning. Here are some important concepts that we did not consider:

- The **vanishing** and **exploding** gradient, see BatchNormalization.

Further considerations

We obviously barely scratched the surface of deep learning. Here are some important concepts that we did not consider:

- The **vanishing** and **exploding** gradient, see BatchNormalization.
- Weights initialization.

Further considerations

We obviously barely scratched the surface of deep learning. Here are some important concepts that we did not consider:

- The **vanishing** and **exploding** gradient, see BatchNormalization.
- Weights initialization.
- **Data augmentation**.

Further considerations

We obviously barely scratched the surface of deep learning. Here are some important concepts that we did not consider:

- The **vanishing** and **exploding** gradient, see BatchNormalization.
- Weights initialization.
- **Data augmentation**.
- Understanding what the network has learnt:

Further considerations

We obviously barely scratched the surface of deep learning. Here are some important concepts that we did not consider:

- The **vanishing** and **exploding** gradient, see BatchNormalization.
- Weights initialization.
- **Data augmentation**.
- Understanding what the network has learnt:
  - Shrikumar, A., Greenside, P. & Kundaje, A. Learning Important Features Through Propagating Activation Differences. *arXiv.org cs.CV*, (2017). [DeepLIFT]

Further considerations

We obviously barely scratched the surface of deep learning. Here are some important concepts that we did not consider:

- The **vanishing** and **exploding** gradient, see BatchNormalization.
- Weights initialization.
- **Data augmentation**.
- Understanding what the network has learnt:
  - Shrikumar, A., Greenside, P. & Kundaje, A. Learning Important Features Through Propagating Activation Differences. *arXiv.org cs.CV*, (2017). [DeepLIFT]
- **Attention** layer

Further considerations

We obviously barely scratched the surface of deep learning. Here are some important concepts that we did not consider:

- The **vanishing** and **exploding** gradient, see BatchNormalization.
- Weights initialization.
- **Data augmentation**.
- Understanding what the network has learnt:
  - Shrikumar, A., Greenside, P. & Kundaje, A. Learning Important Features Through Propagating Activation Differences. *arXiv.org cs.CV*, (2017). [DeepLIFT]
- **Attention** layer
- Multi-tasks (not multi-class, not multi-labels)

Of deep neural networks

- The see the world as a **hierarchy of concepts**, effectively bypassing the need to create features (features engineering).
  - “Deep neural networks can help **circumventing the manual extraction of features** by **learning them** from data.” [1]
- **Transfer learning** is a possibly unique to deep learning.
- **Hundreds of papers** in bioinformatics alone.

Prologue

Summary

- Deep networks consisting only of **dense layers** become **computationally intractable** as the number of parameters grows exponentially with each additional layer.

Summary

- Deep networks consisting only of **dense layers** become **computationally intractable** as the number of parameters grows exponentially with each additional layer.
- **Convolutional layers** considerably reduce the number of parameters since each unit is connected to a limited number of neurons from the previous layer, its **receptive field**.

Summary

- Deep networks consisting only of **dense layers** become **computationally intractable** as the number of parameters grows exponentially with each additional layer.
- **Convolutional layers** considerably reduce the number of parameters since each unit is connected to a limited number of neurons from the previous layer, its **receptive field**.
- **CNN** is able to detect patterns in a positon independent manner.

Summary

- Deep networks consisting only of **dense layers** become **computationally intractable** as the number of parameters grows exponentially with each additional layer.
- **Convolutional layers** considerably reduce the number of parameters since each unit is connected to a limited number of neurons from the previous layer, its **receptive field**.
- **CNN** is able to detect patterns in a positon independent manner.
- **RNN** and **LSTM** handle sequence information, where the input sequences can be of different lengths. They can detect patterns along the sequence.

Summary

- Deep networks consisting only of **dense layers** become **computationally intractable** as the number of parameters grows exponentially with each additional layer.
- **Convolutional layers** considerably reduce the number of parameters since each unit is connected to a limited number of neurons from the previous layer, its **receptive field**.
- **CNN** is able to detect patterns in a positon independent manner.
- **RNN** and **LSTM** handle sequence information, where the input sequences can be of different lengths. They can detect patterns along the sequence.
- **Dropout** layers are an effective regularization mechanism.

Next module

- **Concept**- and **rule**-based

References

Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle.
Deep learning for computational biology.
Mol Syst Biol, 12(7):878, 07 2016.

François Chollet.
Deep learning with Python.
Manning Publications, 2017.

Aurélien Géron.
Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.
O'Reilly Media, 2nd edition, 2019.

Andriy Burkov.
The Hundred-Page Machine Learning Book.
Andriy Burkov, 2019.

Marcel **Turcotte**
Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EECS)
University of Ottawa