

Recitation 2

Neural Networks

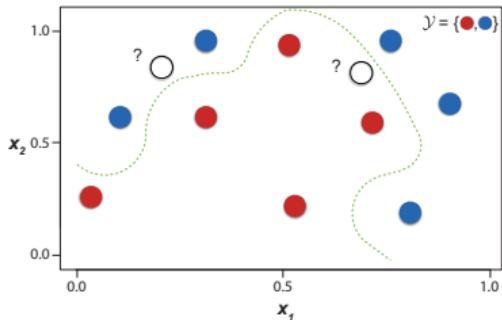
KONSTANTIN KRISMER

MIT - 6.802 / 6.874 / 20.390 / 20.490 / HST.506 - Spring 2019

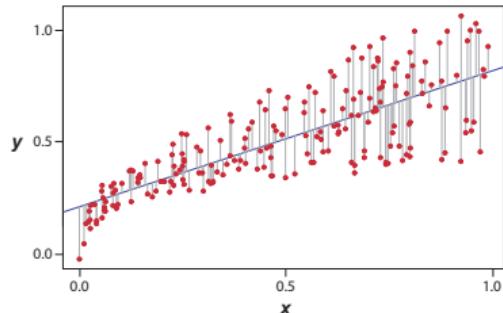
2019-02-14 / 2019-02-15

Recap of Recitation 1

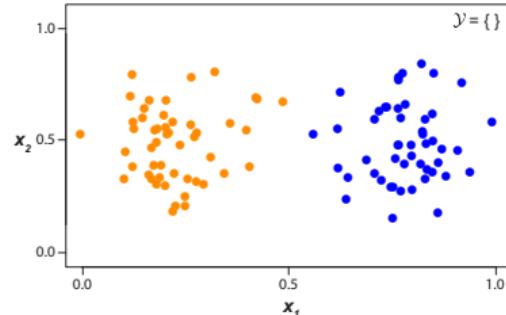
Classification



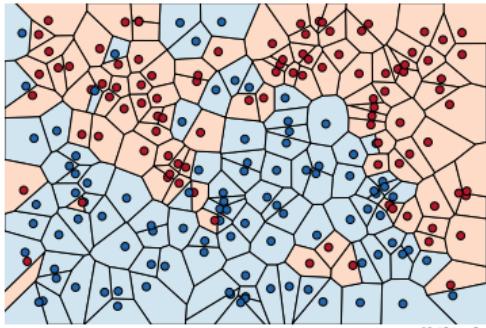
Regression



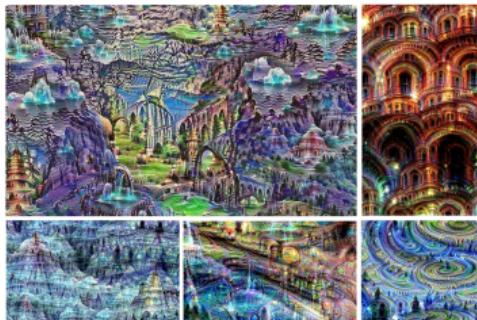
Unsupervised learning



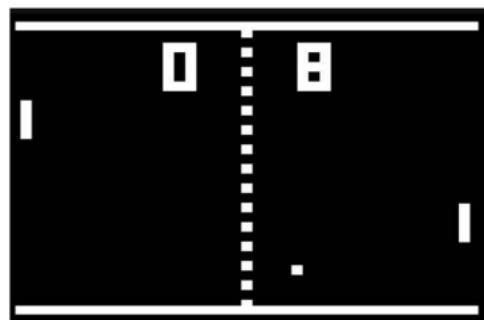
Non-parametric models



Generative models



Reinforcement learning



Recap of Recitation 1

Empirical risk associated with hypothesis $h(\mathbf{x})$:

$$\mathcal{R}_{\text{emp}}(h) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(h(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

Minimize $\mathcal{R}_{\text{emp}}(h)$ to find \hat{h} :

$$\hat{h} = \operatorname{argmin}_{h \in \mathcal{H}} \mathcal{R}_{\text{emp}}(h)$$

Recitation 1:

$\mathbf{x}^{(i)} \in \mathcal{X}, \mathbf{y}^{(i)} \in \mathcal{Y}$ (input space, label space)

$\mathcal{L}(h(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$ (loss functions)

$\operatorname{argmin}_{h \in \mathcal{H}}$ (optimizers)

Recitation 2:

$$\mathcal{H}$$

ML: Define \mathcal{H} , obtain \hat{h}

$\mathcal{H} \subset \mathcal{F}$, where \mathcal{F} is the set of all functions mapping \mathcal{X} onto \mathcal{Y} .

ML: Define \mathcal{H} , obtain \hat{h}

$\mathcal{H} \subset \mathcal{F}$, where \mathcal{F} is the set of all functions mapping \mathcal{X} onto \mathcal{Y} .

Goal: find $\hat{h} \in \mathcal{H}$

ML: Define \mathcal{H} , obtain \hat{h}

$\mathcal{H} \subset \mathcal{F}$, where \mathcal{F} is the set of all functions mapping \mathcal{X} onto \mathcal{Y} .

Goal: find $\hat{h} \in \mathcal{H}$

Step 1: define suitable hypothesis space \mathcal{H}

ML: Define \mathcal{H} , obtain \hat{h}

$\mathcal{H} \subset \mathcal{F}$, where \mathcal{F} is the set of all functions mapping \mathcal{X} onto \mathcal{Y} .

Goal: find $\hat{h} \in \mathcal{H}$

Step 1: define suitable hypothesis space \mathcal{H}

Problem Set 1

$$\mathbf{x} \in [0, 1]^{784}$$

$$\hat{\mathbf{y}} \in [0, 1]^{10}$$

$$\mathbf{W} \in \mathbb{R}^{784 \times 10}$$

$$\mathbf{b} \in \mathbb{R}^{10}$$

$$h(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \phi_{\text{softmax}}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$$

$$\mathcal{H} = \{h(\mathbf{x}; \mathbf{W}, \mathbf{b}) | \mathbf{W} \in \mathbb{R}^{784 \times 10}, \mathbf{b} \in \mathbb{R}^{10}\}$$

ML: Define \mathcal{H} , obtain \hat{h}

$\mathcal{H} \subset \mathcal{F}$, where \mathcal{F} is the set of all functions mapping \mathcal{X} onto \mathcal{Y} .

Goal: find $\hat{h} \in \mathcal{H}$

Step 1: define suitable hypothesis space \mathcal{H}

Problem Set 1

$$\mathbf{x} \in [0, 1]^{784}$$

$$\hat{\mathbf{y}} \in [0, 1]^{10}$$

$$\mathbf{W} \in \mathbb{R}^{784 \times 10}$$

$$\mathbf{b} \in \mathbb{R}^{10}$$

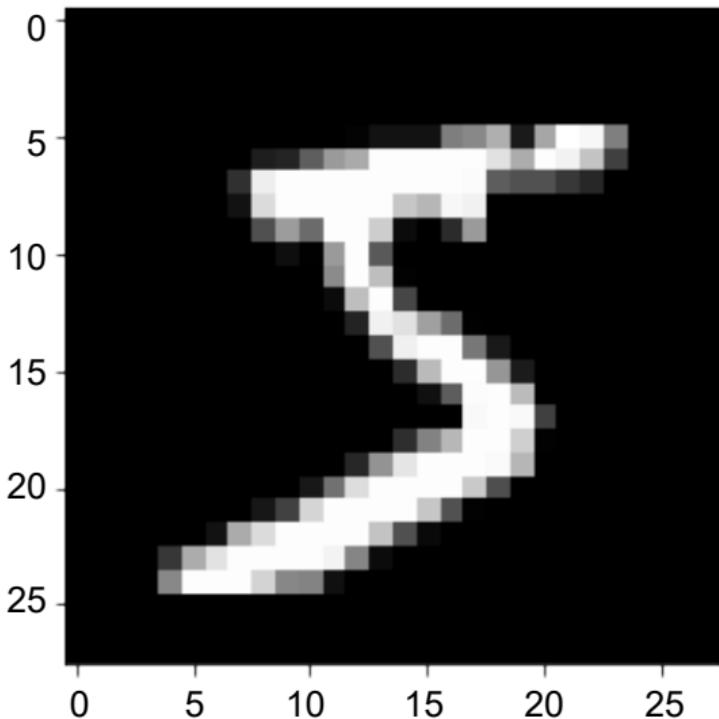
$$h(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \phi_{\text{softmax}}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$$

$$\mathcal{H} = \{h(\mathbf{x}; \mathbf{W}, \mathbf{b}) | \mathbf{W} \in \mathbb{R}^{784 \times 10}, \mathbf{b} \in \mathbb{R}^{10}\}$$

Step 2: use gradient-based optimization methods to obtain \hat{h}

Smartly constrain \mathcal{H}

What is lost when treating the pixels of an image like independent features?



Problem Set 1

$$\mathbf{x} \in [0, 1]^{784}$$

$$\hat{\mathbf{y}} \in [0, 1]^{10}$$

$$\mathbf{W} \in \mathbb{R}^{784 \times 10}$$

$$\mathbf{b} \in \mathbb{R}^{10}$$

$$h(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \phi_{\text{softmax}}(\mathbf{W}^T \mathbf{x} + \mathbf{b})$$

$$\mathcal{H} = \{h(\mathbf{x}; \mathbf{W}, \mathbf{b}) \mid \mathbf{W} \in \mathbb{R}^{784 \times 10}, \mathbf{b} \in \mathbb{R}^{10}\}$$

Using softmax regression, we treat the 28 by 28 2D image as a vector of 784 features, without any notion of topological distance between them.

Solution: Constrain the hypothesis space \mathcal{H} in a way to *exploit the structure of the input*.

Structured input

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Structured input

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Structured input

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Structured input

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Example 4 (1D structure): Natural language

Structured input

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Example 4 (1D structure): Natural language

Example 5 (2D structure): $\mathcal{X} = [0, 1]^{28 \times 28}$, MNIST images

Structured input

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Example 4 (1D structure): Natural language

Example 5 (2D structure): $\mathcal{X} = [0, 1]^{28 \times 28}$, MNIST images

Example 6 (3D structure): $\mathcal{X} = [0, 1]^{m \times n \times p}$, voxels (volumetric pixels) from imaging methods such as CT and MRI

Structured input

Example 1 (unstructured): $\mathcal{X} = \mathbb{R}^n$, gene expression values for n genes, order of features (genes) meaningless

Example 2 (1D structure): $\mathcal{X} = \mathbb{R}^{n \times t}$, gene expression values for n genes on t time points

Example 3 (1D structure): $\mathcal{X} = \{0, 1\}^{m \times n}$, biological sequences (DNA, RNA, protein sequence) of length m and alphabet size n , e.g., $n = 4$ (A, C, G, T)

Example 4 (1D structure): Natural language

Example 5 (2D structure): $\mathcal{X} = [0, 1]^{28 \times 28}$, MNIST images

Example 6 (3D structure): $\mathcal{X} = [0, 1]^{m \times n \times p}$, voxels (volumetric pixels) from imaging methods such as CT and MRI

Example 7 (4D structure): $\mathcal{X} = [0, 1]^{m \times n \times p \times t}$, doxels (dynamic voxels) a sequence of voxel data, e.g., time series of CT and MRI scans

Neural networks: flexibly constrain hypothesis space \mathcal{H}

- No structure:

Dense layer (also called fully connected layer)

Neural networks: flexibly constrain hypothesis space \mathcal{H}

- No structure:

Dense layer (also called fully connected layer)

- Spatial structure:

Convolutional layer (with 1D, 2D, 3D kernels): translationally equivariant through weight sharing

Input:



Filter output:



Pooling layer (with 1D, 2D, 3D kernels): (somewhat) translationally invariant through summaries

Input:



Filter output:



Neural networks: flexibly constrain hypothesis space \mathcal{H}

- No structure:

Dense layer (also called fully connected layer)

- Spatial structure:

Convolutional layer (with 1D, 2D, 3D kernels): translationally equivariant through weight sharing

Input:



Filter output:



Pooling layer (with 1D, 2D, 3D kernels): (somewhat) translationally invariant through summaries

Input:



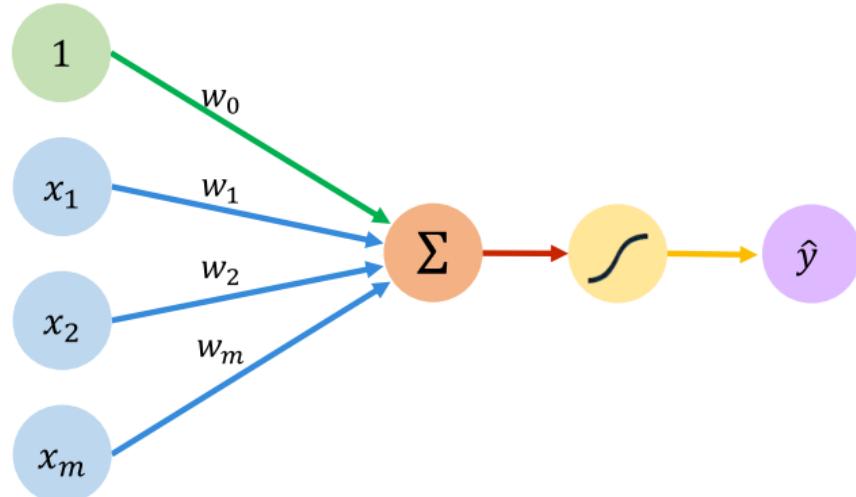
Filter output:



- Sequential / temporal structure:

Recurrent layer (and its variants - GRU, LSTM): shares weights between time points

Perceptron



Linear combination of inputs

Output

$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

Non-linear activation function

Bias

Illustrations by courtesy of [Amini and Soleimany, 2019].

© A. Amini, A. Soleimany

Activation functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Sigmoid)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016
<http://sebastianraschka.com>

only in output layer (why?):

- linear activation function (for regression)
- sigmoid activation function (for binary or multi-label classification)
- softmax activation function (for binary or multi-class classification)

- softmax (identical to sigmoid in binary classification)
- exponential linear unit (ELU)

$$\phi(z, \alpha)_{\text{ELU}} = \begin{cases} z, & \text{for } z \geq 0 \\ \alpha(e^z - 1), & \text{for } z < 0 \end{cases}$$

- scaled exponential linear unit (SELU)

$$\phi(z, \alpha, \beta) = \beta * \phi(z, \alpha)_{\text{ELU}}$$

- softsign

$$\phi(z) = \frac{z}{|z| + 1}$$

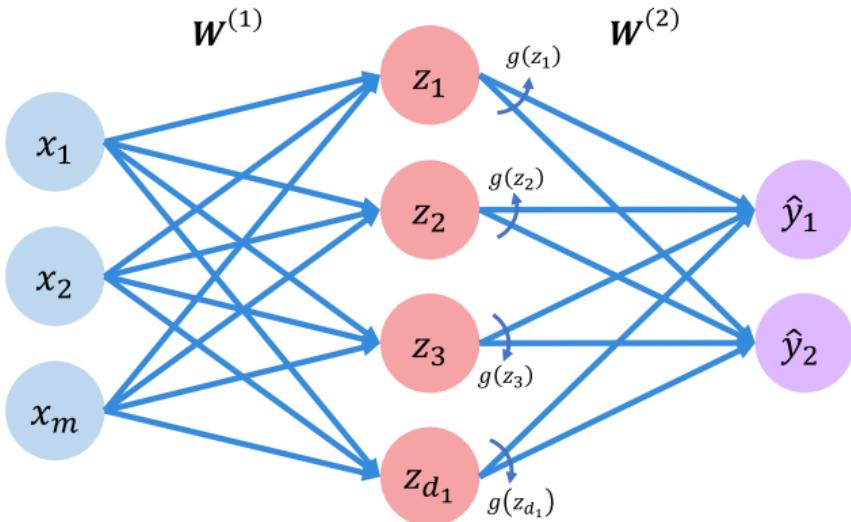
- leaky ReLU

$$\phi(z, \alpha) = \begin{cases} z, & \text{for } z \geq 0 \\ \alpha z, & \text{for } z < 0 \end{cases}$$

- parametric ReLU (PReLU, identical to leaky ReLU, but α is learned)

$$\phi(z; \alpha) = \begin{cases} z, & \text{for } z \geq 0 \\ \alpha z, & \text{for } z < 0 \end{cases}$$

Single-layer neural network



Inputs

Hidden

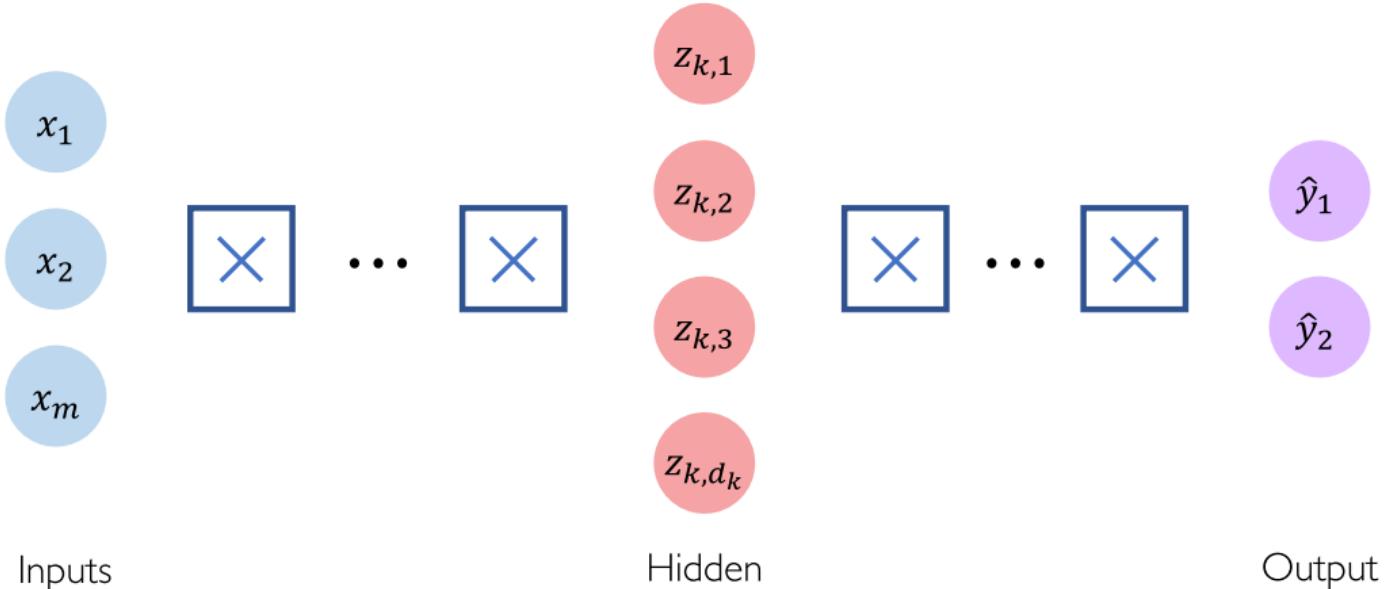
Final Output

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

© A. Amini, A. Soleimany

Multi-layer neural network

How deep is deep learning?



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$

© A. Amini, A. Soleimany

Dense layer in TF: low-level vs high-level

Listing 1: low-level TF (`tf.nn.*`)

```
def model(x_ph):
    ...
    w = tf.Variable(tf.truncated_normal([in_units, out_units]))
    b = tf.Variable(tf.constant(0.1, shape=[out_units]))
    dense_op = tf.matmul(x_ph, w)
    dense_op = tf.nn.bias_add(dense_op, b)
    dense_op = tf.nn.relu(dense_op)
    ...
    ...
```

PS2 will be the last problem set implemented in low-level TF.

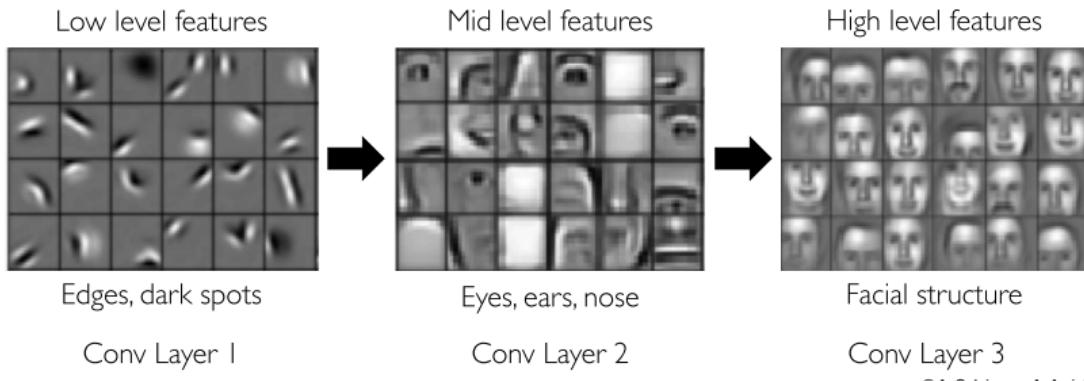
Listing 2: high-level TF (`tf.keras.*`)

```
model = tf.keras.Sequential()
# first layer
model.add(tf.keras.layers.Dense(out_units, activation='relu', input_shape=(in_channels,)))
...
model.add(tf.keras.layers.Dense(out_units, activation='relu'))
...
...
```

Convolutional neural networks

Idea: Learn a hierarchy of pattern detectors (representation or feature learning)

Example: Filters in 3-layer convolutional neural network for face recognition



© A. Soleimany, A. Amini

Example: In CNN for TF binding site classification, filters learn binding site motifs (position weight matrices)

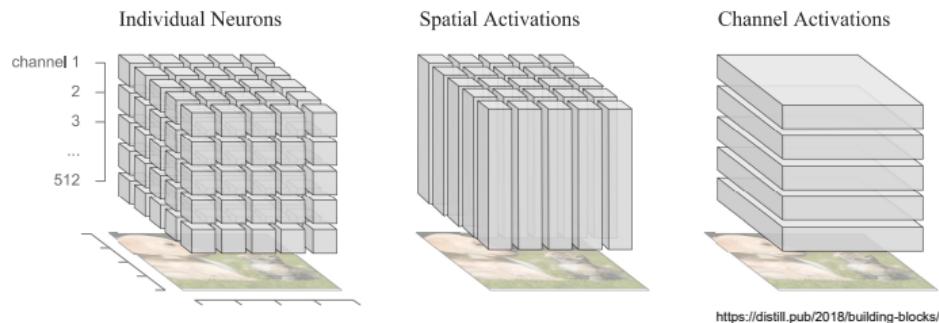
Filters and corresponding output channels

Examples of first layer input channels:

- RGB channels in color images
- ACGT channels in one-hot encoded DNA sequences

Examples of output channels of convolutional layers:

- horizontal edge channel, dark spot channel in low-level layer trained on images
- nose channel, eye channel, ear channel in higher-level layers trained on images of faces)
- CTCF binding site channel in low-level layer trained on DNA sequences of TF binding sites



Visualizations of filters and operations in latent space:

- Chris Olah's article on Distill: <https://distill.pub/2018/building-blocks/>
- OpenAI's Glow: <https://blog.openai.com/glow/>

Convolutional layer in TF: low-level vs high-level

Listing 3: low-level TF (`tf.nn.*`)

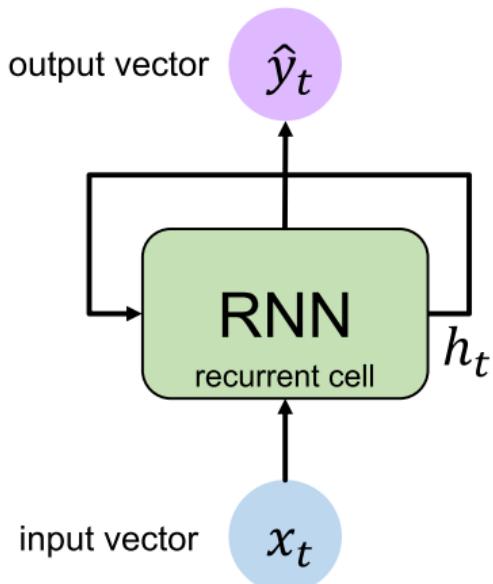
```
def model(x_ph):
    ...
    w = tf.Variable(tf.truncated_normal([filter_height, filter_width,
                                         in_channels, out_channels]))
    b = tf.Variable(tf.constant(0.1, shape=[out_channels]))
    conv_op = tf.nn.conv2d(x_ph, w, strides=[1, 1, 1, 1], padding="SAME")
    conv_op = tf.nn.bias_add(conv_op, b)
    conv_op = tf.nn.relu(conv_op)
    ...
    ...
```

Listing 4: high-level TF (`tf.keras.*`)

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(out_channels,
                               kernel_size=(filter_height, filter_width),
                               strides=1,
                               activation="relu", padding="SAME"))
...
...  
...
```

Recurrent neural networks

Idea: Learn sequential / temporal relationships



Apply a **recurrence relation** at every time step to process a sequence:

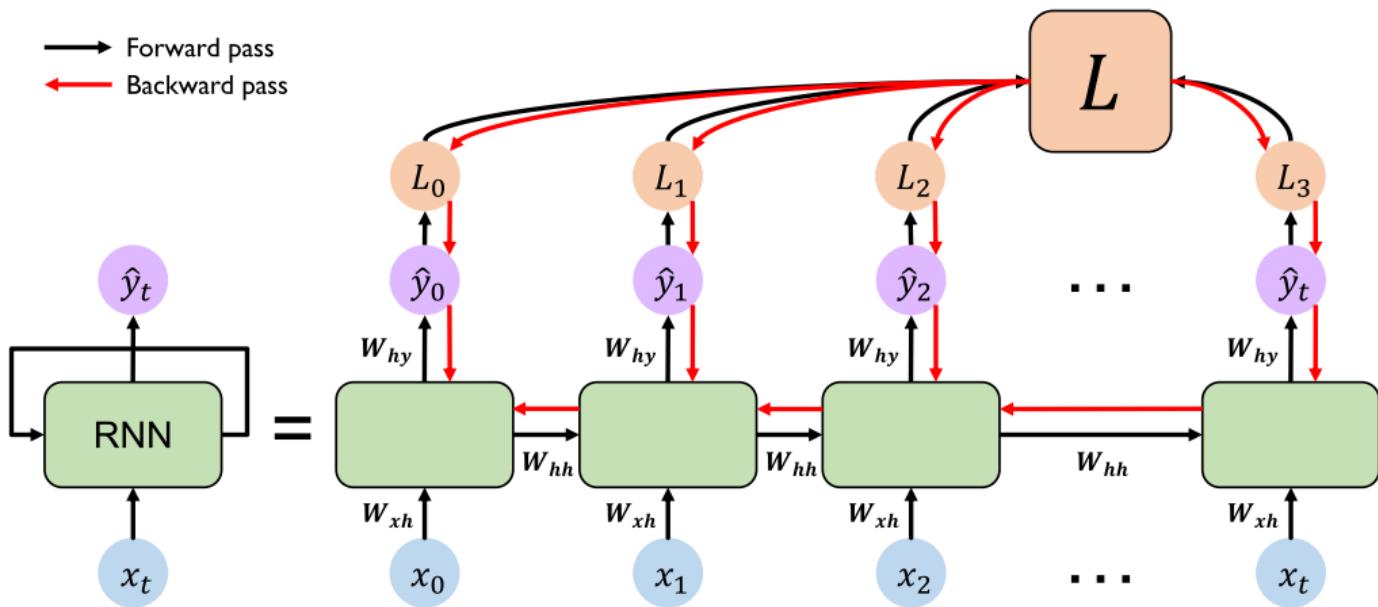
$$h_t = f_W(h_{t-1}, x_t)$$

new state function old state input vector at
 parameterized by W time step t

Note: the same function and set of parameters are used at every time step

© A. Soleimany, A. Amini

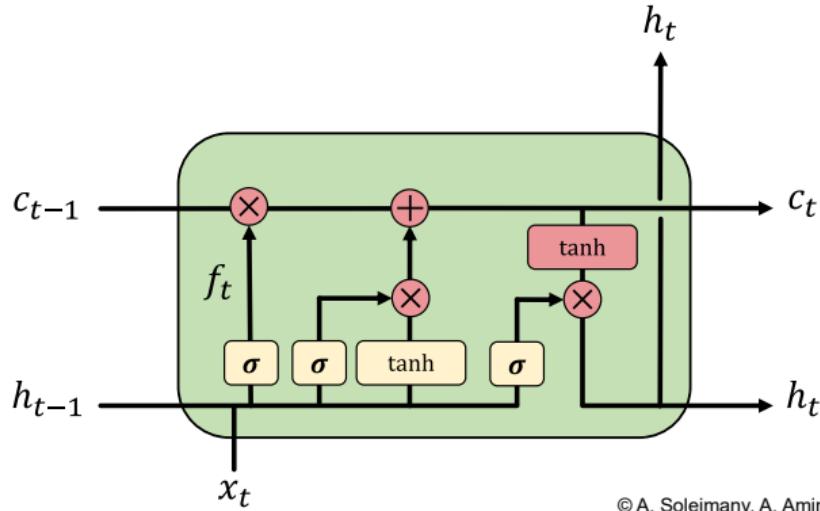
RNN backpropagation through time



© A. Soleimany, A. Amini

RNN variants GRU and LSTM

Idea: introduce memory in recurrent unit and control it via gates (forget gate, input / update gate, output gate)



© A. Soleimany, A. Amini

Useful links:

- 6.S191 lecture on RNN, GRU, LSTM:
http://introtodeeplearning.com/materials/2019_6S191_L2.pdf
- Chris Olah's article about LSTM:
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTM layer in TF: low-level vs high-level

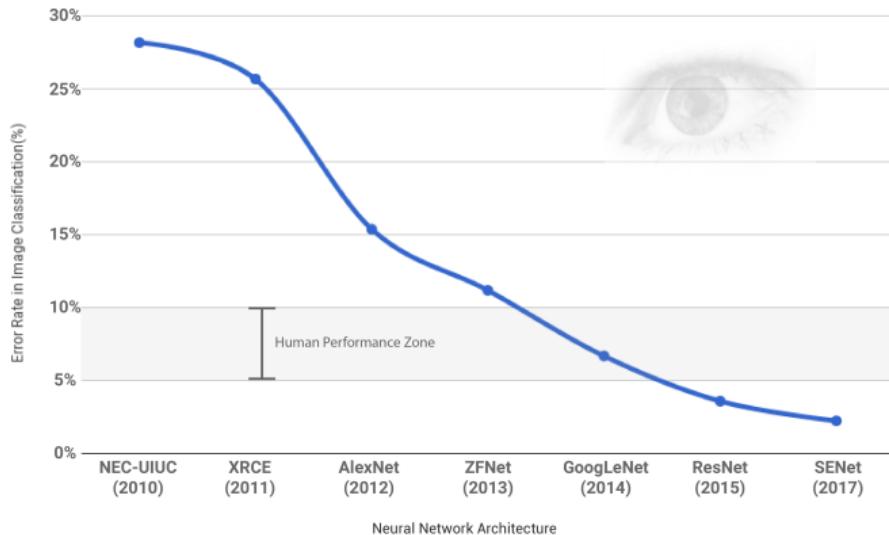
Listing 5: low-level TF (`tf.nn.*`)

```
def model(x_ph):
    # x_ph.get_shape() == [time_points * number_of_examples, in_channels]
    ...
    split_ops = tf.split(axis=0, num_or_size_splits=time_points, value=x_ph)
    lstm_cell = tf.nn.rnn_cell.LSTMCell(units)
    outputs, states = tf.nn.static_rnn(lstm_cell, split_ops, dtype=tf.float32)
    # outputs[-1] contains RNN prediction (prediction at last time point)
    ...
```

Listing 6: high-level TF (`tf.keras.*`)

```
model = tf.keras.Sequential()
...
model.add(tf.keras.layers.LSTM(units))
...  
...
```

Why Deep Residual Networks?



Last two ILSVRC winners are networks with residual modules:

- ILSVRC 2017: SENet [Hu et al., 2018] is an ensemble of different models, highest scoring model is modified ResNeXt [Xie et al., 2017]
- ILSVRC 2016: ResNet [He et al., 2016]

Evolution of Deep Residual Networks

- **ResNet:** to remedy vanishing gradients, introduce skip connections (bypass layers by adding identity mapping)

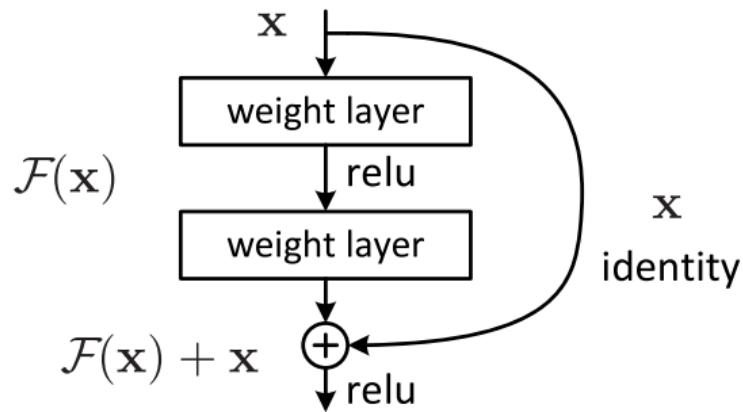


Figure 2. Residual learning: a building block.

Evolution of Deep Residual Networks

- **Wide Residual Networks:** decrease depth (number of layers), increase width (number of feature maps) in residual layers [Zagoruyko and Komodakis, 2016]
- **ResNeXt:** introduce cardinality (number of aggregated transformations), in addition to depth and width [Xie et al., 2017]

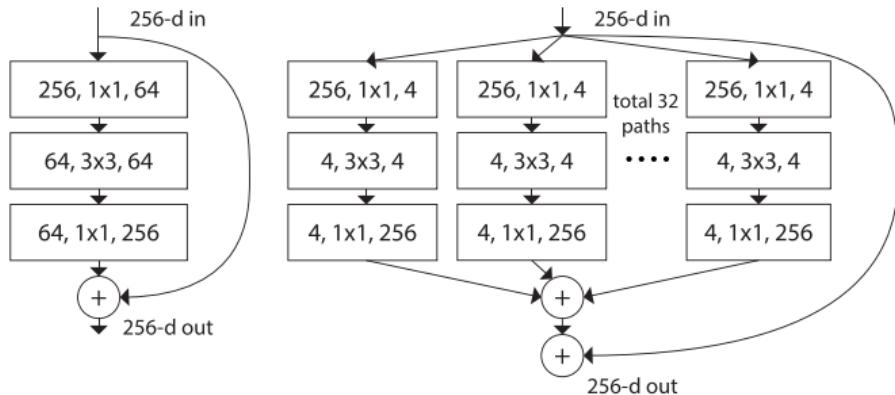


Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Evolution of Deep Residual Networks

- **DenseNet:** each layer has direct input connections to all previous feature maps [Huang et al., 2016]

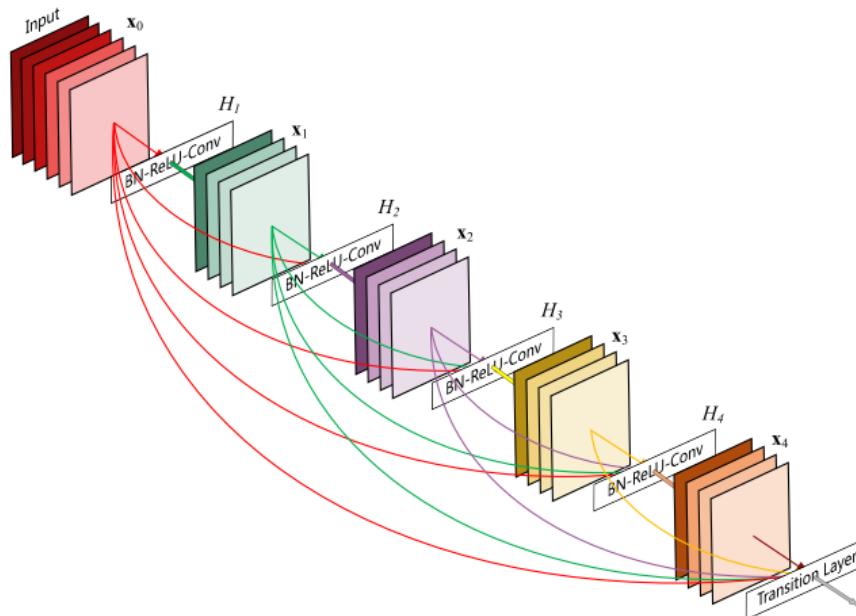


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

References

-  Amini, A. and Soleimany, A. (2019).
MIT 6.S191: Intro to Deep Learning.
<http://introtodeeplearning.com>.
-  He, K., Zhang, X., Ren, S., and Sun, J. (2016).
Deep residual learning for image recognition.
2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778.
-  Hu, J., Shen, L., and Sun, G. (2018).
Squeeze-and-excitation networks.
2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7132–7141.
-  Huang, G., Liu, Z., and Weinberger, K. Q. (2016).
Densely connected convolutional networks.
CoRR, abs/1608.06993.
-  Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017).
Aggregated residual transformations for deep neural networks.
In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995.
-  Zagoruyko, S. and Komodakis, N. (2016).
Wide residual networks.
CoRR, abs/1605.07146.