A Brief Overview of Recent Neural Network Methods and Architectures

UCLA CS161

Mohammad Kachuee

Dec 6, 2017

Overview

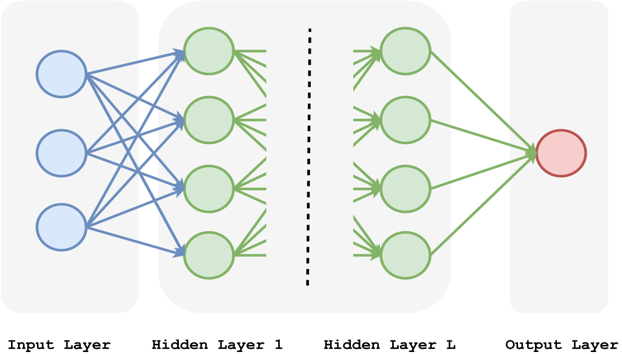
1. Fully Connected Neural Networks
   - Multi-layer Preceptron (MLP)
   - Limitations of MLP
   - Inspirations from Visual Cortex
2. Convolutional Neural Networks (CNNs)
   - Basic Ideas of CNNs
   - Famous CNN Architectures
   - Visualizing CNNs
3. Recurrent Neural Networks (RNNs)
   - Handling Sequences as Input
   - RNN Applications
   - Basic Ideas of RNNs
4. TensorFlow Tutorial

Outline

1. **Fully Connected Neural Networks**
   - **Multi-layer Preceptron (MLP)**
   - **Limitations of MLP**
   - **Inspirations from Visual Cortex**
2. Convolutional Neural Networks (CNNs)
   - Basic Ideas of CNNs
   - Famous CNN Architectures
   - Visualizing CNNs
3. Recurrent Neural Networks (RNNs)
   - Handling Sequences as Input
   - RNN Applications
   - Basic Ideas of RNNs
4. TensorFlow Tutorial

Multi-layer Preceptron (MLP) (1/2)

$$h_{i+1} = \sigma(h_i W_i + b_i)$$



Multi-layer Preceptron (MLP) (2/2)

- In a MLP, we have a few fully connected layers that are stacked with each other.
- There are three layer types: input layer, hidden layer, and output layer.
- Each layer takes the activation values of the previous layer as input, applies a linear transformation to it, and crates output activations by applying a non-linearity.
- The universal approximation theorem: theoretically, we can model any function using a neural network with a single hidden layer.

Limitations of MLP

- MLP discards any structure in the input space and creates a fully connected many to many mapping.

Limitations of MLP

- MLP discards any structure in the input space and creates a fully connected many to many mapping.
- Each layer in MLP has O(mn) weights (m input neurons and n output neurons).

Limitations of MLP

- MLP discards any structure in the input space and creates a fully connected many to many mapping.
- Each layer in MLP has O(mn) weights (m input neurons and n output neurons).
- Having many neurons in each layer and many layers makes the model very complex and hard to train.

Limitations of MLP

- MLP discards any structure in the input space and creates a fully connected many to many mapping.
- Each layer in MLP has O(mn) weights (m input neurons and n output neurons).
- Having many neurons in each layer and many layers makes the model very complex and hard to train.
- Practically, training networks with more than a few layers is not easy.

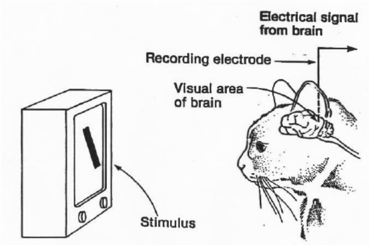
Limitations of MLP

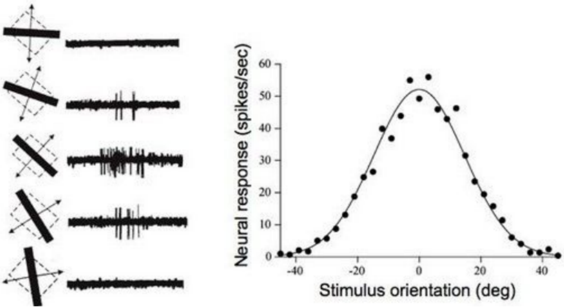
- MLP discards any structure in the input space and creates a fully connected many to many mapping.
- Each layer in MLP has O(mn) weights (m input neurons and n output neurons).
- Having many neurons in each layer and many layers makes the model very complex and hard to train.
- Practically, training networks with more than a few layers is not easy.
- Finding the optimal number of layers and neurons inside each one is an open problem.

Limitations of MLP

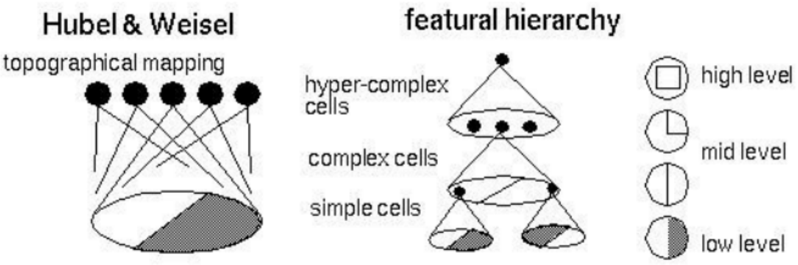
- MLP discards any structure in the input space and creates a fully connected many to many mapping.
- Each layer in MLP has O(mn) weights (m input neurons and n output neurons).
- Having many neurons in each layer and many layers makes the model very complex and hard to train.
- Practically, training networks with more than a few layers is not easy.
- Finding the optimal number of layers and neurons inside each one is an open problem.
- The universal approximation theorem is not helping us that much because a huge number of hidden neurons needed to approximate a function using a shallow network.

Hubel and Wiesel Experiments





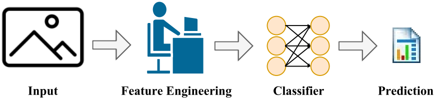
Hierarchical Organization of Visual Cortex



Outline

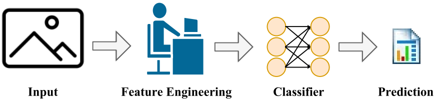
The Era of Deep Representations (1/2)

- Image recognition is a machine learning application in which the machine should predict objects that are present in the image.
- Before 2012, the major method in image recognition was using hand engineered features and classic machine learning classifiers.

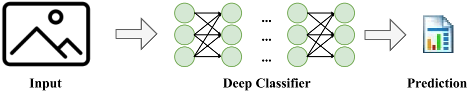


The Era of Deep Representations (1/2)

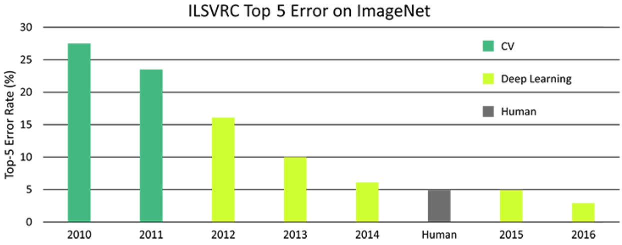
- Image recognition is a machine learning application in which the machine should predict objects that are present in the image.
- Before 2012, the major method in image recognition was using hand engineered features and classic machine learning classifiers.



- Around 2012, three major factors changed the future landscape of recognition tasks from engineered features and shallow architectures to using automated feature extraction through deep neural networks.
  - Significantly bigger datasets such as ImageNet (10,000,000 labeled images depicting 10,000+ object categories)
  - More powerful computers and GPU accelerated computation.
  - Development of techniques and tricks for training deeper networks.



The Era of Deep Representations (2/2)



Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

Natural Images as Input

- Natural images are smooth.



Natural Images as Input

- Natural images are smooth.
- Nearby pixels are coherent and highly correlated with each other.

Natural Images as Input

- Natural images are smooth.
- Nearby pixels are coherent and highly correlated with each other.
- An object inside an image can be at different coordinates.

Natural Images as Input

- Natural images are smooth.
- Nearby pixels are coherent and highly correlated with each other.
- An object inside an image can be at different coordinates.
- An object inside an image can be at different scales or distances from the view-point.

Natural Images as Input

- Natural images are smooth.
- Nearby pixels are coherent and highly correlated with each other.
- An object inside an image can be at different coordinates.
- An object inside an image can be at different scales or distances from the view-point.
- In real-world, we have lighting and noise affecting the image.

Natural Images as Input

- Natural images are smooth.
- Nearby pixels are coherent and highly correlated with each other.
- An object inside an image can be at different coordinates.
- An object inside an image can be at different scales or distances from the view-point.
- In real-world, we have lighting and noise affecting the image.

Is it possible to take the advantage of image characteristics in designing the network?

Basic Ideas of CNNs: Intuition

- The same object may be located anywhere inside an image.

Basic Ideas of CNNs: Intuition

- The same object may be located anywhere inside an image.
- Image patch characteristics are almost similar across different scenes and views.

Basic Ideas of CNNs: Intuition

- The same object may be located anywhere inside an image.
- Image patch characteristics are almost similar across different scenes and views.
- Use the same feature detector at different input locations.

Basic Ideas of CNNs: Intuition

- The same object may be located anywhere inside an image.
- Image patch characteristics are almost similar across different scenes and views.
- Use the same feature detector at different input locations.
- In other words, share network weights to prevent learning redundant mappings and reuse feature detectors.

Basic Ideas of CNNs: Intuition

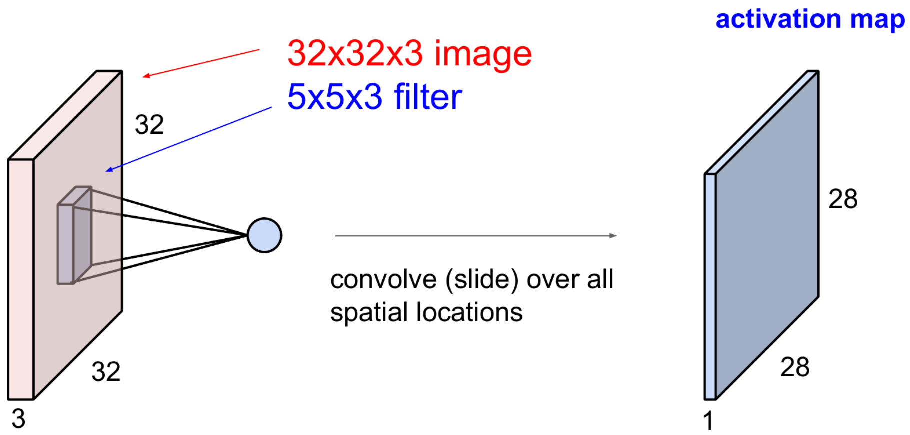
- The same object may be located anywhere inside an image.
- Image patch characteristics are almost similar across different scenes and views.
- Use the same feature detector at different input locations.
- In other words, share network weights to prevent learning redundant mappings and reuse feature detectors.
- It can be easily modeled as the 2D convolution operation.

Basic Ideas of CNNs: Intuition

- The same object may be located anywhere inside an image.
- Image patch characteristics are almost similar across different scenes and views.
- Use the same feature detector at different input locations.
- In other words, share network weights to prevent learning redundant mappings and reuse feature detectors.
- It can be easily modeled as the 2D convolution operation.
- It is, basically, learning a set of filters and applying them at different image locations.

Basic Ideas of CNNs: Conv Layers (1/3)

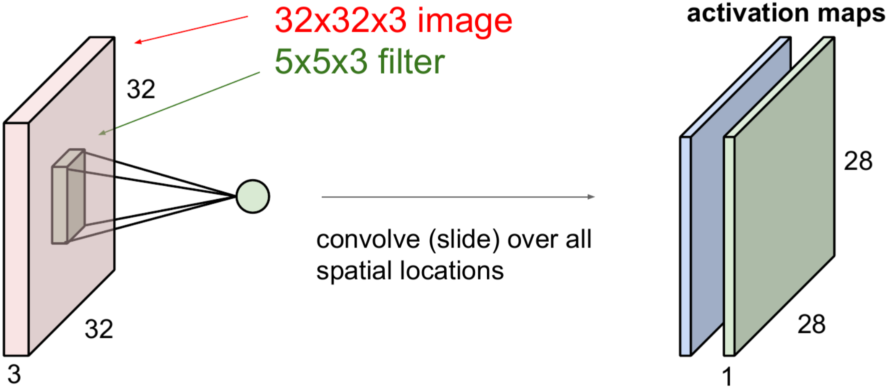
Applying a convolution filter on an image:



Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Basic Ideas of CNNs: Conv Layers (2/3)

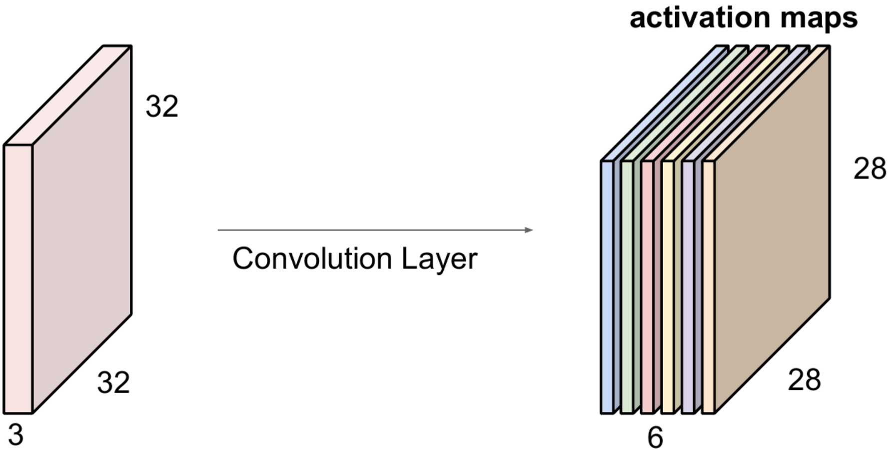
Applying two different convolution filters on an image:



Fei-Fei Li, Andrej Karpathy, and Justin Johnson

Basic Ideas of CNNs: Conv Layers (3/3)

Applying a set of different convolution filters on an image:



Fei-Fei Li, Andrej Karpathy, and Justin Johnson

- In the example above, we mapped from a $32 \times 32 \times 3$ input to a $28 \times 28 \times 1$ output using a $5 \times 5 \times 3$ filter.

Benefits of CNNs: Reducing model complexity

- In the example above, we mapped from a $32 \times 32 \times 3$ input to a $28 \times 28 \times 1$ output using a $5 \times 5 \times 3$ filter.
- If we had a fully connected maping, similar to what we had in MLP, we needed a $(32 \times 32 \times 3) \times (28 \times 28 \times 1)$ weight matrix.

Benefits of CNNs: Reducing model complexity

- In the example above, we mapped from a $32 \times 32 \times 3$ input to a $28 \times 28 \times 1$ output using a $5 \times 5 \times 3$ filter.
- If we had a fully connected maping, similar to what we had in MLP, we needed a $(32 \times 32 \times 3) \times (28 \times 28 \times 1)$ weight matrix.
- It means using 75 weights versus 2408448!

Benefits of CNNs: Reducing model complexity

- In the example above, we mapped from a $32 \times 32 \times 3$ input to a $28 \times 28 \times 1$ output using a $5 \times 5 \times 3$ filter.
- If we had a fully connected maping, similar to what we had in MLP, we needed a $(32 \times 32 \times 3) \times (28 \times 28 \times 1)$ weight matrix.
- It means using 75 weights versus 2408448!
- Usually there are more than one convolution filter.

Benefits of CNNs: Reducing model complexity

- In the example above, we mapped from a $32 \times 32 \times 3$ input to a $28 \times 28 \times 1$ output using a $5 \times 5 \times 3$ filter.
- If we had a fully connected maping, similar to what we had in MLP, we needed a $(32 \times 32 \times 3) \times (28 \times 28 \times 1)$ weight matrix.
- It means using 75 weights versus 2408448!
- Usually there are more than one convolution filter.
- However, the overall complexity is an order of magnitude lower!

Benefits of CNNs: Reducing model complexity

- In the example above, we mapped from a $32 \times 32 \times 3$ input to a $28 \times 28 \times 1$ output using a $5 \times 5 \times 3$ filter.
- If we had a fully connected maping, similar to what we had in MLP, we needed a $(32 \times 32 \times 3) \times (28 \times 28 \times 1)$ weight matrix.
- It means using 75 weights versus 2408448!
- Usually there are more than one convolution filter.
- However, the overall complexity is an order of magnitude lower!
- Lower complexity means ability to fit models faster, easier, and using fewer training samples.

Benefits of CNNs: Fast GPU Training

- Modern GPUs are optimized for fast and efficient convolution.

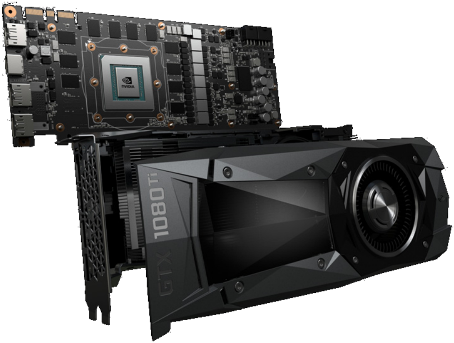


Figure: Nvidia GTX 1080Ti (3584 Cores, 11GB Memory). Source: nvidia.com

Benefits of CNNs: Fast GPU Training

- Modern GPUs are optimized for fast and efficient convolution.
- A modern GPU has about a few thousands of cores and tens of Gigabytes Ram that can work in parallel.

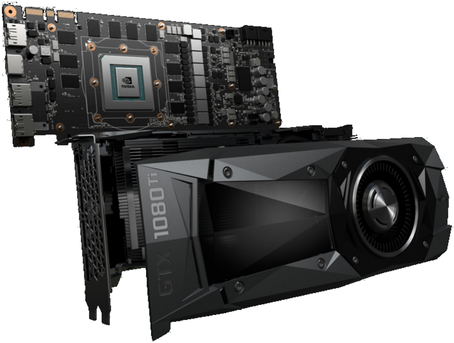


Figure: Nvidia GTX 1080Ti (3584 Cores, 11GB Memory). Source: nvidia.com

Benefits of CNNs: Fast GPU Training

- Modern GPUs are optimized for fast and efficient convolution.
- A modern GPU has about a few thousands of cores and tens of Gigabytes Ram that can work in parallel.
- Many GPUs can be coupled with each other, creating a GPU farm.

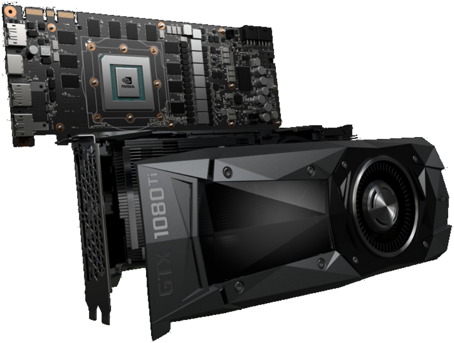
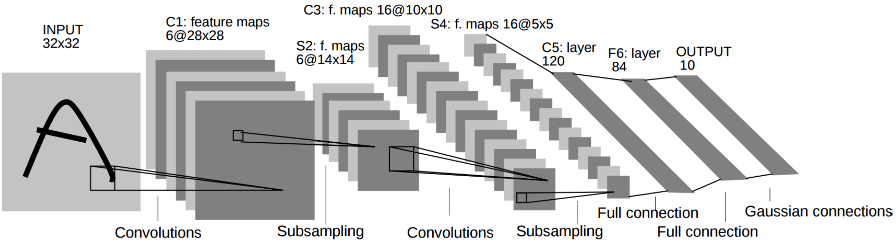


Figure: Nvidia GTX 1080Ti (3584 Cores, 11GB Memory). Source: nvidia.com

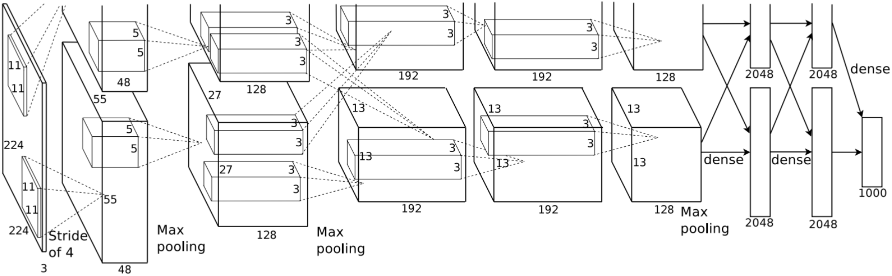
Famous CNN Architectures: LeNet 1998



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

Famous CNN Architectures: ImageNet 2012 (AlexNet) (1/2)

About 8 weight layers and 50 million parameters.

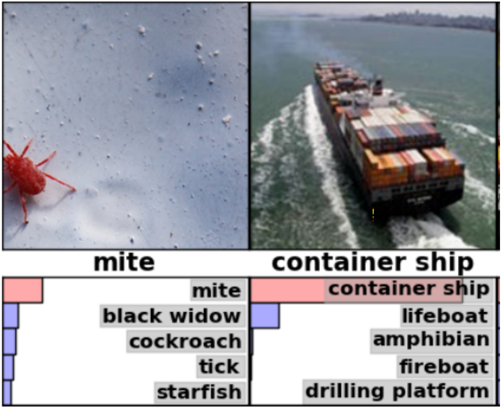


Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Famous CNN Architectures: ImageNet 2012 (AlexNet) (2/2)

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

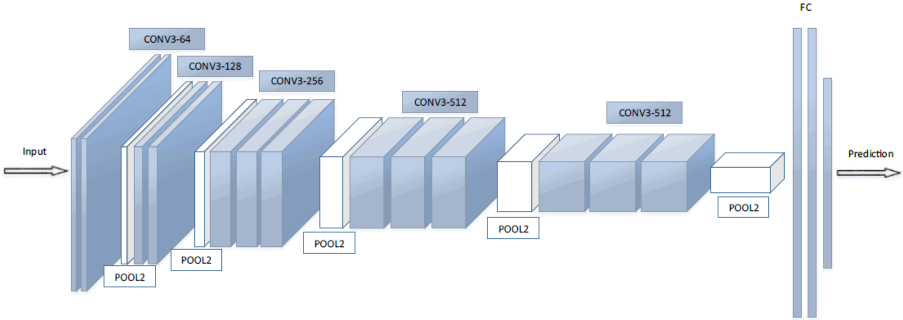
Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Famous CNN Architectures: VGG (1/2)

About 20 weight layers and 130 million parameters.



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

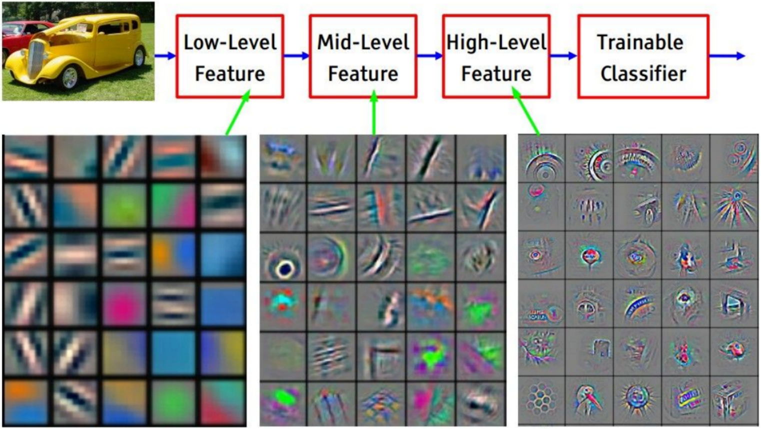
Famous CNN Architectures: VGG (2/2)

Table 7: **Comparison with the state of the art in ILSVRC classification.** Our method is denoted as "VGG". Only the results obtained without outside training data are reported.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

Visualizing CNNs



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.

Outline

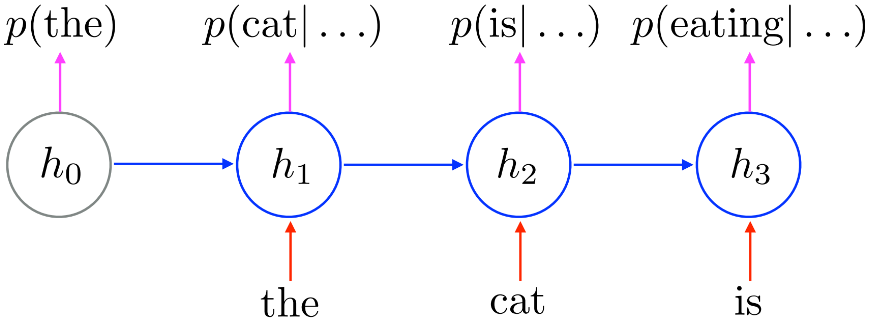
Handling Sequences as Input

- We have a wide range of applications in which:
  - We are dealing with a sequence of inputs (not a single feature vector!)
  - The sequence length is not necessarily fixed.
  - We may need to have multiple output values at different time instances.

Handling Sequences as Input

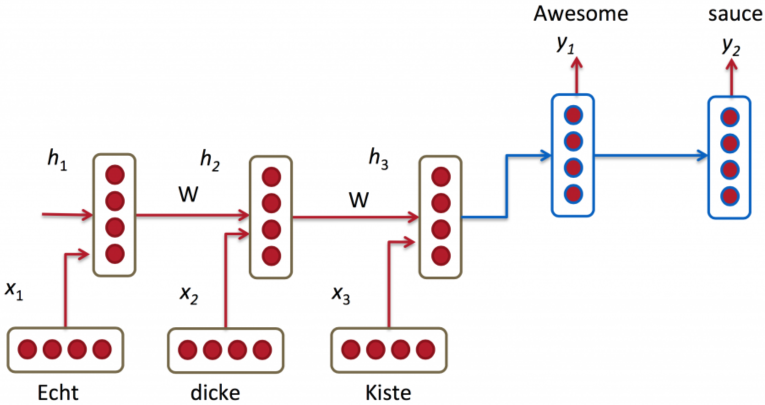
- We have a wide range of applications in which:
  - We are dealing with a sequence of inputs (not a single feature vector!)
  - The sequence length is not necessarily fixed.
  - We may need to have multiple output values at different time instances.
- For instance:
  - Predictive analysis in time series. e.g., predicting the expected future value of the stocks for a company.
  - Language modeling. e.g., designing an auto-complete engine which can predict and suggest the next work given the current incomplete sentence.
  - Sequence to sequence modeling. e.g., translating a speech from English to French in real-time.

RNN Applications: Language Modeling



Kyunghyun Cho, Natural Language Understanding

RNN Applications: Sentence Translation



Richard Socher, Recurrent Neural Networks

- RNNs are capable of having sequences as input or output.

Basic Ideas of RNNs

- RNNs are capable of having sequences as input or output.
- Recurrent connections in neural networks can be used to create a notion of memory.

Basic Ideas of RNNs

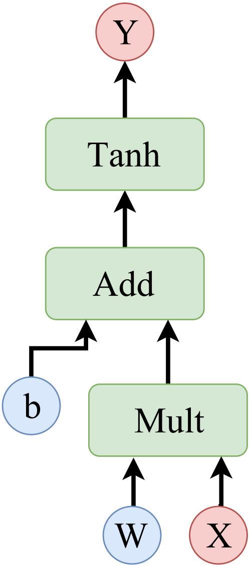
- RNNs are capable of having sequences as input or output.
- Recurrent connections in neural networks can be used to create a notion of memory.
- RNNs can be trained by using the same general techniques we had before (unrolling technique).

Basic Ideas of RNNs

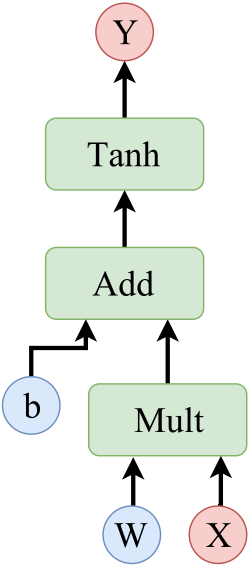
- RNNs are capable of having sequences as input or output.
- Recurrent connections in neural networks can be used to create a notion of memory.
- RNNs can be trained by using the same general techniques we had before (unrolling technique).
- A more detailed discussion of RNNs is out of the scope of this lecture.

Outline

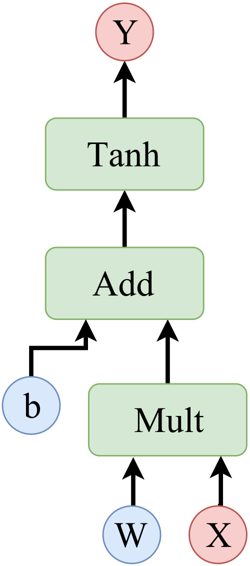
- Consider the computation of forward path in a simple one layer neural network:
  $Y = tanh(WX + b)$



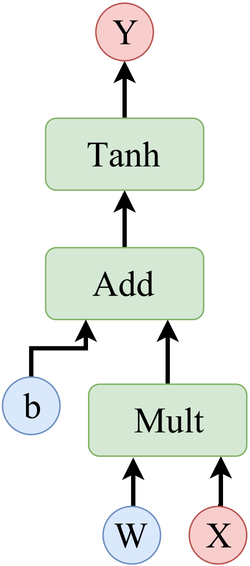
- Consider the computation of forward path in a simple one layer neural network: $Y = tanh(WX + b)$
- A data flow graph is a computational graph that explains the processing that is happening to the flow of data.



- Consider the computation of forward path in a simple one layer neural network: $Y = tanh(WX + b)$
- A data flow graph is a computational graph that explains the processing that is happening to the flow of data.
- Data flow model is in contrast to the control flow model.



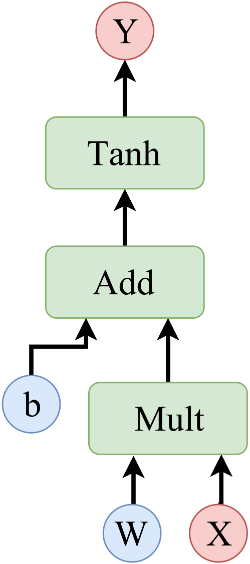
- Consider the computation of forward path in a simple one layer neural network: $Y = tanh(WX + b)$
- A data flow graph is a computational graph that explains the processing that is happening to the flow of data.
- Data flow model is in contrast to the control flow model.
- TensorFlow is a computational library by Google that is designed for fast and scalable computing based on data flow graphs.



TensorFlow: Introduction (2/2)

- In a typical computational library, we call functions on data sequentially.
  - It adds the overhead of function call, argument pass, and return to the actual processing time.
  - It prevents optimizing in higher-levels. E.g., multiplication result of two huge matrices is multiplied by a vector.

- In a typical computational library, we call functions on data sequentially.
  - It adds the overhead of function call, argument pass, and return to the actual processing time.
  - It prevents optimizing in higher-levels. E.g., multiplication result of two huge matrices is multiplied by a vector.
- In TensorFlow, we create a complete computational graph. Then we feed our data to the graph and every process takes place in background.



Basics: Variables, Constants, Operations

- We can define variable and constant tensors similar to numpy or any other computational library.

Basics: Variables, Constants, Operations

- We can define variable and constant tensors similar to numpy or any other computational library.
- We can define operations between tensors to create new tensors.

Basics: Variables, Constants, Operations

- We can define variable and constant tensors similar to numpy or any other computational library.
- We can define operations between tensors to create new tensors.
- Learning and optimization functions and operations are also can be defined in this fashion.

Basics: Variables, Constants, Operations

- We can define variable and constant tensors similar to numpy or any other computational library.
- We can define operations between tensors to create new tensors.
- Learning and optimization functions and operations are also can be defined in this fashion.
- Defined variables are by default trainable and need to be initialized before usage.

```
# import tensorflow package
import tensorflow as tf
# create a constant tensor of ones
C = tf.ones((2,2))
# create a tensor variable initialized with zeros, name it
    weights
W = tf.Variable(tf.zeros((2,2)), name="weights")
# define a multiplication tensor (Y) which is the result of
    multiplying C by W
Y = tf.matmul(C, W, name='op_matmul_CW')
```

Basics: Placeholders

- Placeholder variables are input ports to the TensorFlow graph.

Basics: Placeholders

- Placeholder variables are input ports to the TensorFlow graph.
- The Python program can feed data into and get information from the computational graph through placeholders.

Basics: Placeholders

- Placeholder variables are input ports to the TensorFlow graph.
- The Python program can feed data into and get information from the computational graph through placeholders.
- We define placeholders and use them as regular tensor variables inside our graph code.

Basics: Placeholders

- Placeholder variables are input ports to the TensorFlow graph.
- The Python program can feed data into and get information from the computational graph through placeholders.
- We define placeholders and use them as regular tensor variables inside our graph code.
- before running the graph, we assign values to the placeholders in our Python code.

```

# create a place holder of type float and shape 2x2
P = tf.placeholder(dtype=tf.float32, shape=(2,2), name='ph_input')
# create a tensor variable initialized with zeros, name it weights
W = tf.Variable(tf.zeros((2,2)), name="weights")
# define a multiplication tensor (Y) which is the result of multiplying P by W
Y = tf.matmul(P, W, name='op_matmul_CW')
```

Basics: Session and Graph

- After creating a graph by chaining TensorFlow operations, we have our computational graph specified.

Basics: Session and Graph

- After creating a graph by chaining TensorFlow operations, we have our computational graph specified.
- However, before using a graph, we need to create a session in which the graph evaluation will take place.

Basics: Session and Graph

- After creating a graph by chaining TensorFlow operations, we have our computational graph specified.
- However, before using a graph, we need to create a session in which the graph evaluation will take place.
- In TensorFlow we do this by creating a session object and running it while feeding the placeholders.

```
# the next three lines are similar to the previous example
P = tf.placeholder(dtype=tf.float32, shape=(2,2), name='ph_input')
W = tf.Variable(tf.zeros((2,2)), name="weights")
Y = tf.matmul(P, W, name='op_matmul_CW')
# create a session
sess = tf.Session()
# create a dummy feed data
feed_data = {P: np.ones((2,2), dtype=np.float32)}
# run the Y graph using the session and feed the dummy data
res = sess.run(Y, feed_dict=feed_data)
# print the result
print(res)
```

Training and Optimization

- TensorFlow supports automatic differentiation.
- For training a neural net, we need to:
  - Define forward path.
  - Define a cost function.
  - Create an optimizer.
  - Run the optimization operation to update network weights.

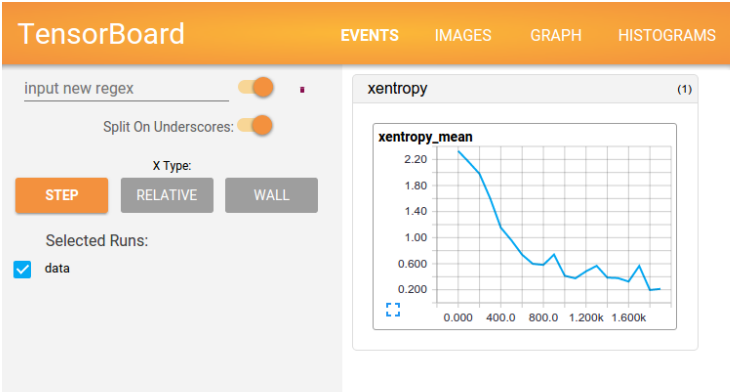
```
# define a train step
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(
    cross_entropy)
# training for 1000 iterations
for _ in range(1000):
  # load a dataset batch
  batch_xs, batch_ys = mnist.train.next_batch(100)
  # run the train step
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Linear Regression using TensorFlow

```python
# define placeholders
X = tf.placeholder(dtype=tf.float32, shape=(None,1), name='ph_input')
Y = tf.placeholder(dtype=tf.float32, shape=(None,1), name='ph_output')
# create variable tensors
W = tf.Variable(0.0, dtype=tf.float32, name="weight")
b = tf.Variable(0.0, dtype=tf.float32, name="bias")
# define a cost function
errors = Y - X * W + b
cost_mse = tf.reduce_mean(errors ** 2.0)
# define a train step
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cost_mse)
# training for 1000 iterations
for iter_trn in range(1000):
    # load a dataset batch
    batch_xs, batch_ys = train_dataset.next_batch(100)
    # run the train step
    sess.run(train_step, feed_dict={X: batch_xs, Y: batch_ys})
    # test the model
    batch_xs, batch_ys = test_dataset.next_batch(100)
    cost = sess.run(cost_mse, feed_dict={X: batch_xs, Y: batch_ys})
    print('Iter: ', iter_trn, ' Test cost:', cost)
```

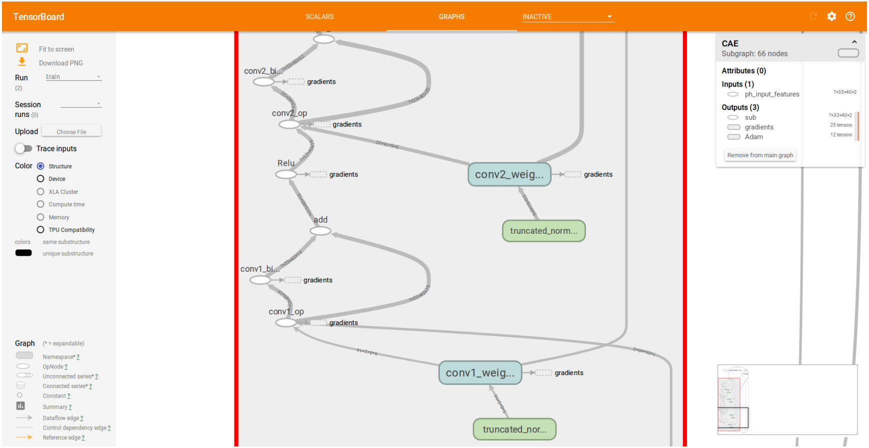
TensorBoard (1/2)

- TensorBoard is a web application by TensorFlow that offers visualization tools for graph visualization, drawing learning curves and histograms, etc.



tensorflow.org

TensorBoard (2/2)



The End