UiO : Department of Informatics
University of Oslo

INF 5860 Machine learning for image classification

Summary

June 5, 2018

Progress

- **TensorFlow**
- Convolutional neural networks
- Generalization
- Recurrent neural networks
- Deep reinforcement learning

Why do we need Deep learning frameworks?

- **Speed:**
  - Fast GPU/CPU implementation of matrix multiplication, convolutions and backpropagation
- **Automatic differentiations**:
  - Pre-implementation of the most common functions and it’s gradients.
- **Reuse:**
  - Easy to reuse other people’s models
- **Less error prone:**
  - The more code you write yourself, the more errors

TensorFlow

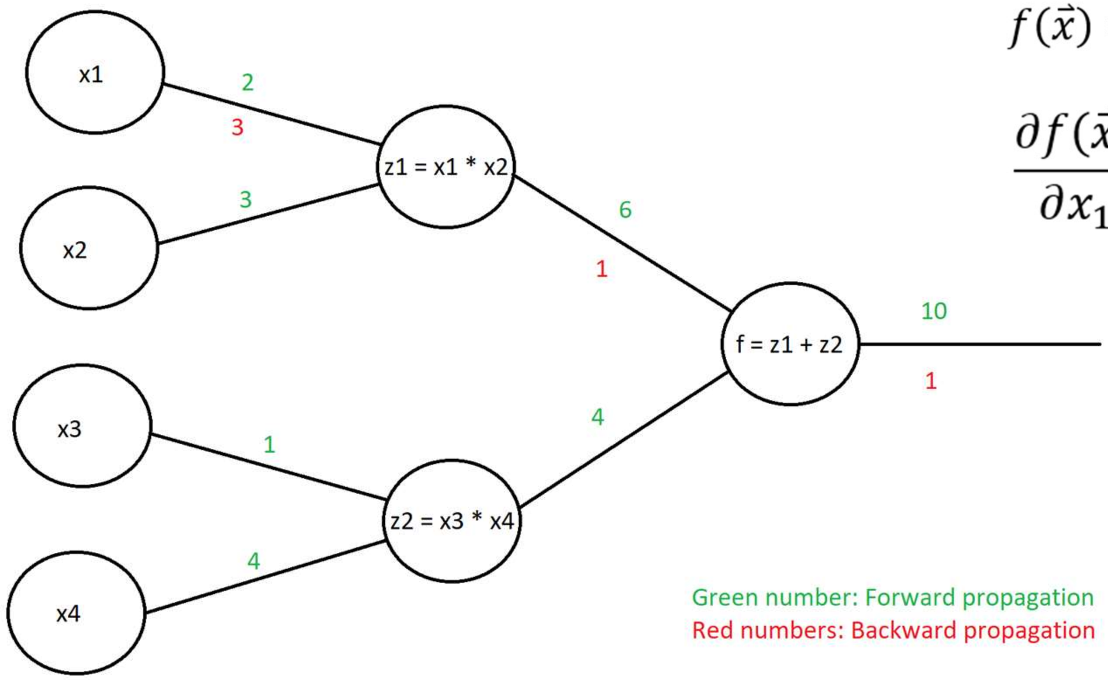
- TensorFlow graphs
- TensorFlow session
- TensorFlow constants
- TensorFlow variables
- TensorFlow feeding data to the graph
- Tensorboard
- TensorFlow save/restore models
- TensorFlow example

Graphs

$$f(\vec{x}) = x_1 * x_2 + x_3 * x_4$$

$$f(\vec{x}) = z_1 + z_2$$

$$\frac{\partial f(\vec{x})}{\partial x_1} = \frac{\partial f}{\partial z_1}\frac{\partial z_1}{\partial x_1} = x_2$$



Green number: Forward propagation
Red numbers: Backward propagation

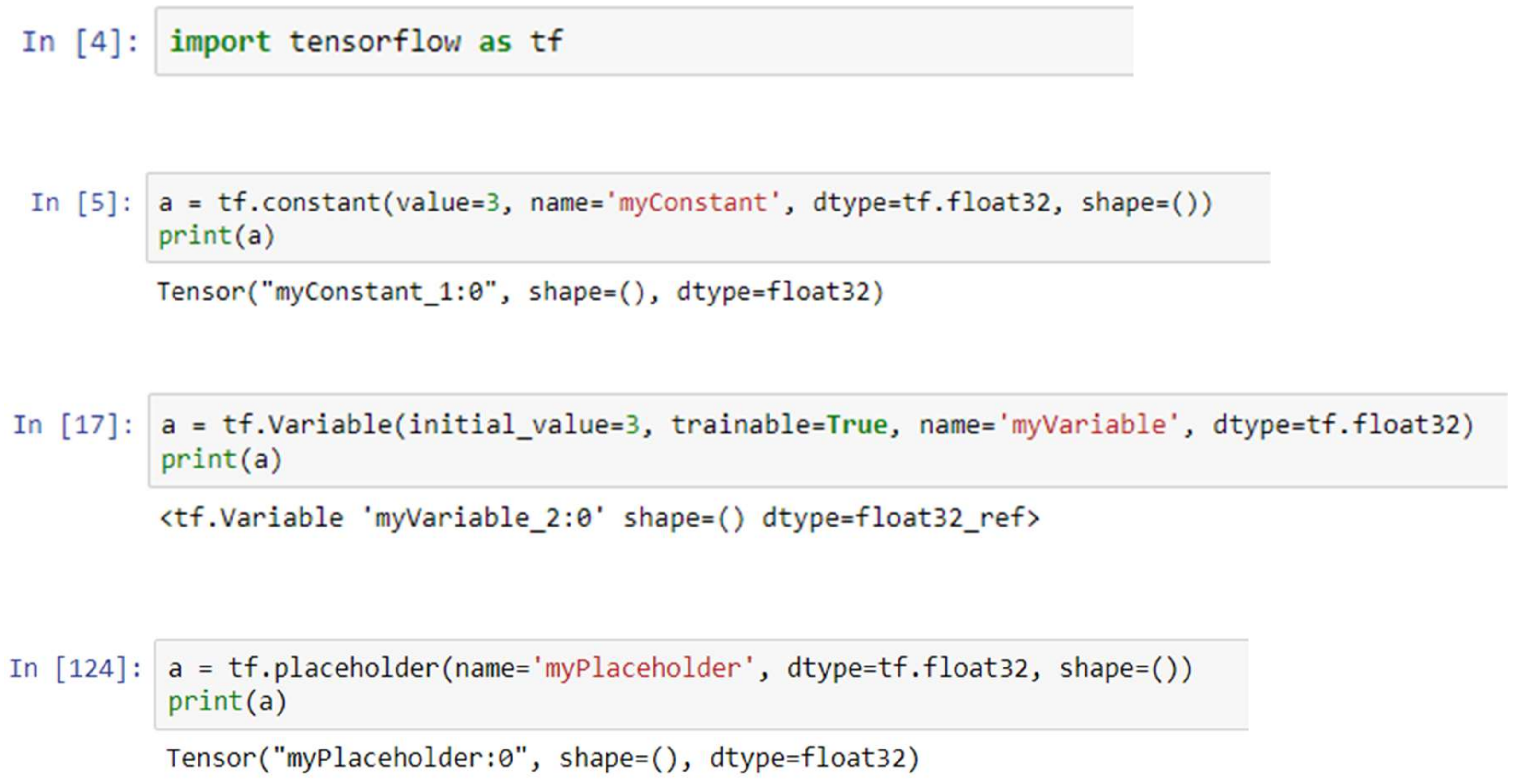
14.2.2018

Defining a tensor in TensorFlow

- The main types of tensors are:
  - tf.Variable / tf.get_variable
  - tf.constant
  - tf.placeholder

- Attributes (some of them):
  - Shape
  - dtype
  - name

Example of a tensors



```
In [4]: import tensorflow as tf
```

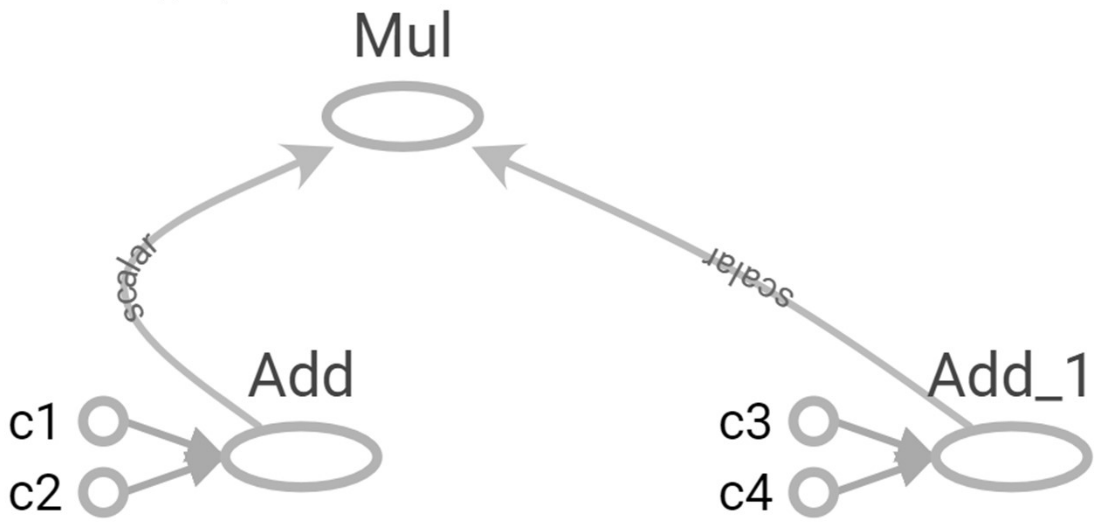
```
In [5]: a = tf.constant(value=3, name='myConstant', dtype=tf.float32, shape=())
        print(a)

        Tensor("myConstant_1:0", shape=(), dtype=float32)
```

```
In [17]: a = tf.Variable(initial_value=3, trainable=True, name='myVariable', dtype=tf.float32)
         print(a)

         <tf.Variable 'myVariable_2:0' shape=() dtype=float32_ref>
```

```
In [124]: a = tf.placeholder(name='myPlaceholder', dtype=tf.float32, shape=())
          print(a)

          Tensor("myPlaceholder:0", shape=(), dtype=float32)
```

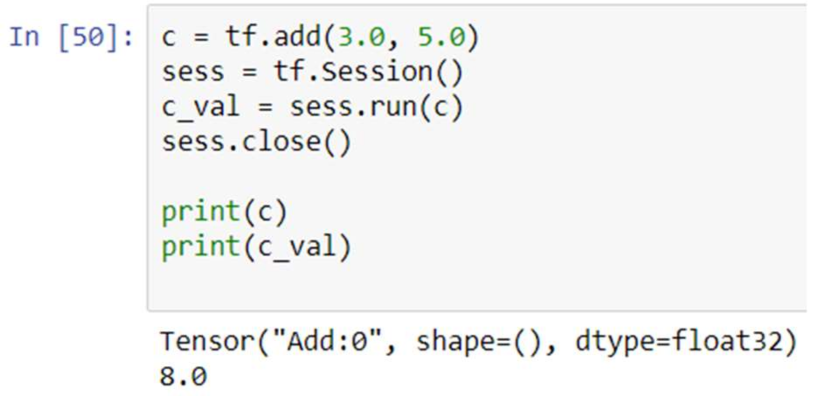
The graph

- The TensorFlow graph is a definition, not any computation.
- The computational graph is a series of TensorFlow operations arranged into a graph. The graph is composed of two types of objects:
  - Operation: Nodes in the graph
  - Tensors: The edges in the graph



Executing the tf.Graph: tf.Session

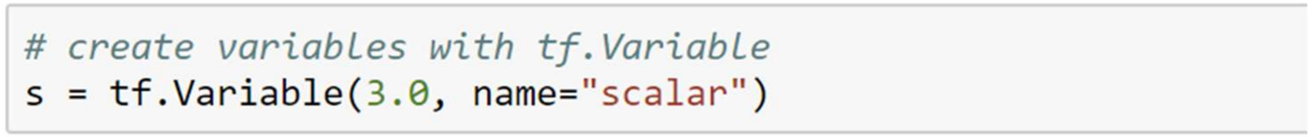
- We have seen that variables and constants are handles to elements in the computational graph only.
- We execute the graph using a tf.Session



```
In [50]: c = tf.add(3.0, 5.0)
         sess = tf.Session()
         c_val = sess.run(c)
         sess.close()

         print(c)
         print(c_val)

         Tensor("Add:0", shape=(), dtype=float32)
         8.0
```

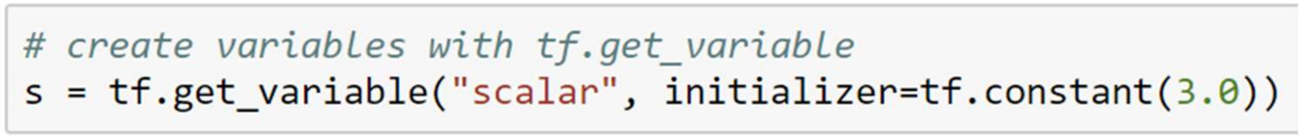
Creating variables

We can define variables two ways:



```
# create variables with tf.Variable
s = tf.Variable(3.0, name="scalar")
```

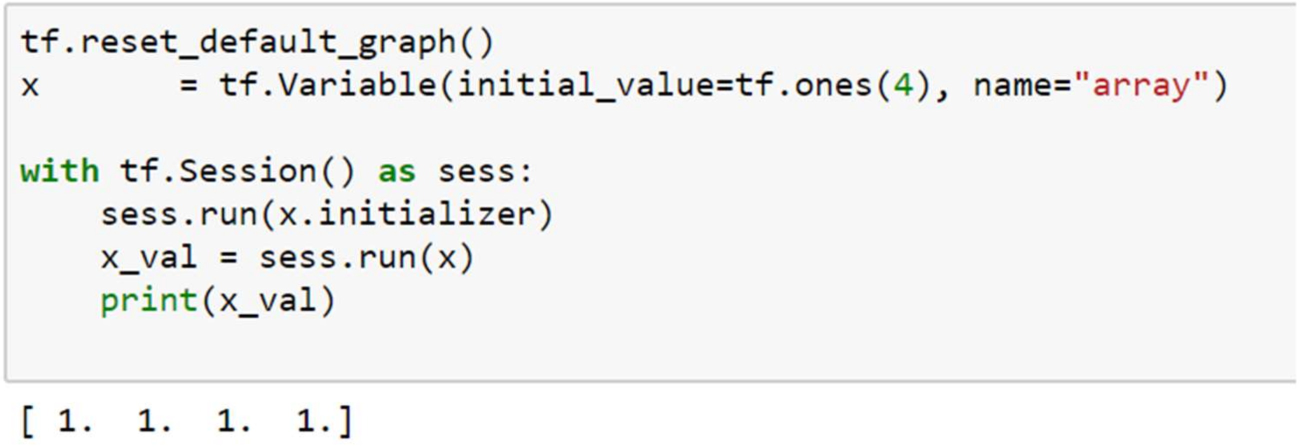
preferred



```
# create variables with tf.get_variable
s = tf.get_variable("scalar", initializer=tf.constant(3.0))
```

Initializing variables

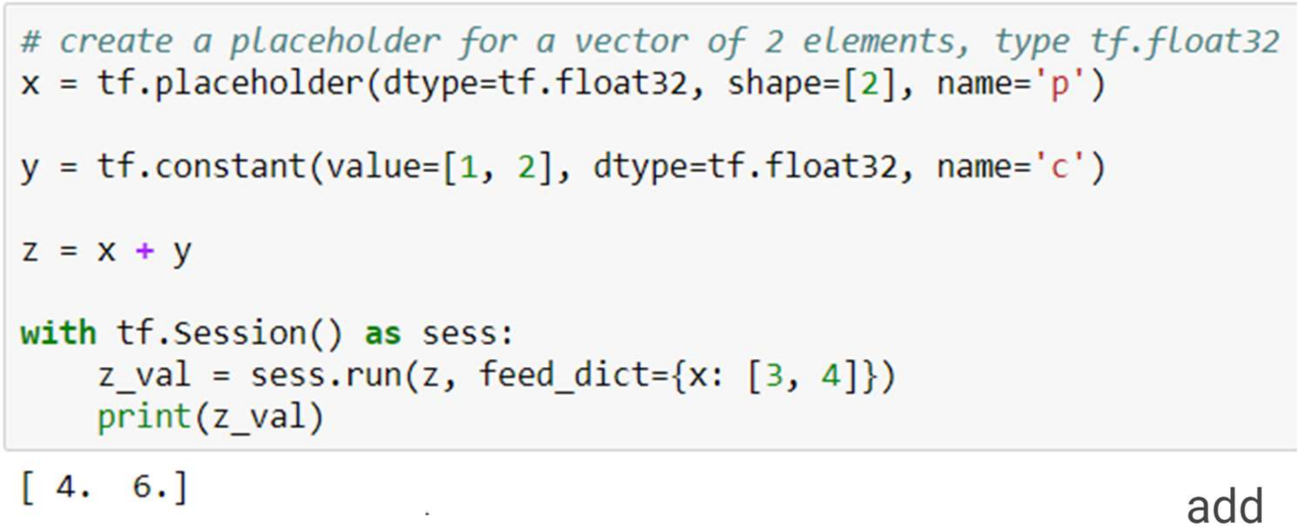
- The variables to be used in the graph have to be either:
  - Initialized
  - Restored



```
tf.reset_default_graph()
x       = tf.Variable(initial_value=tf.ones(4), name="array")

with tf.Session() as sess:
    sess.run(x.initializer)
    x_val = sess.run(x)
    print(x_val)
```

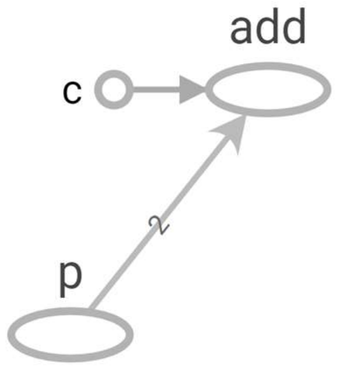
```
[ 1.  1.  1.  1.]
```

tf.placeholder

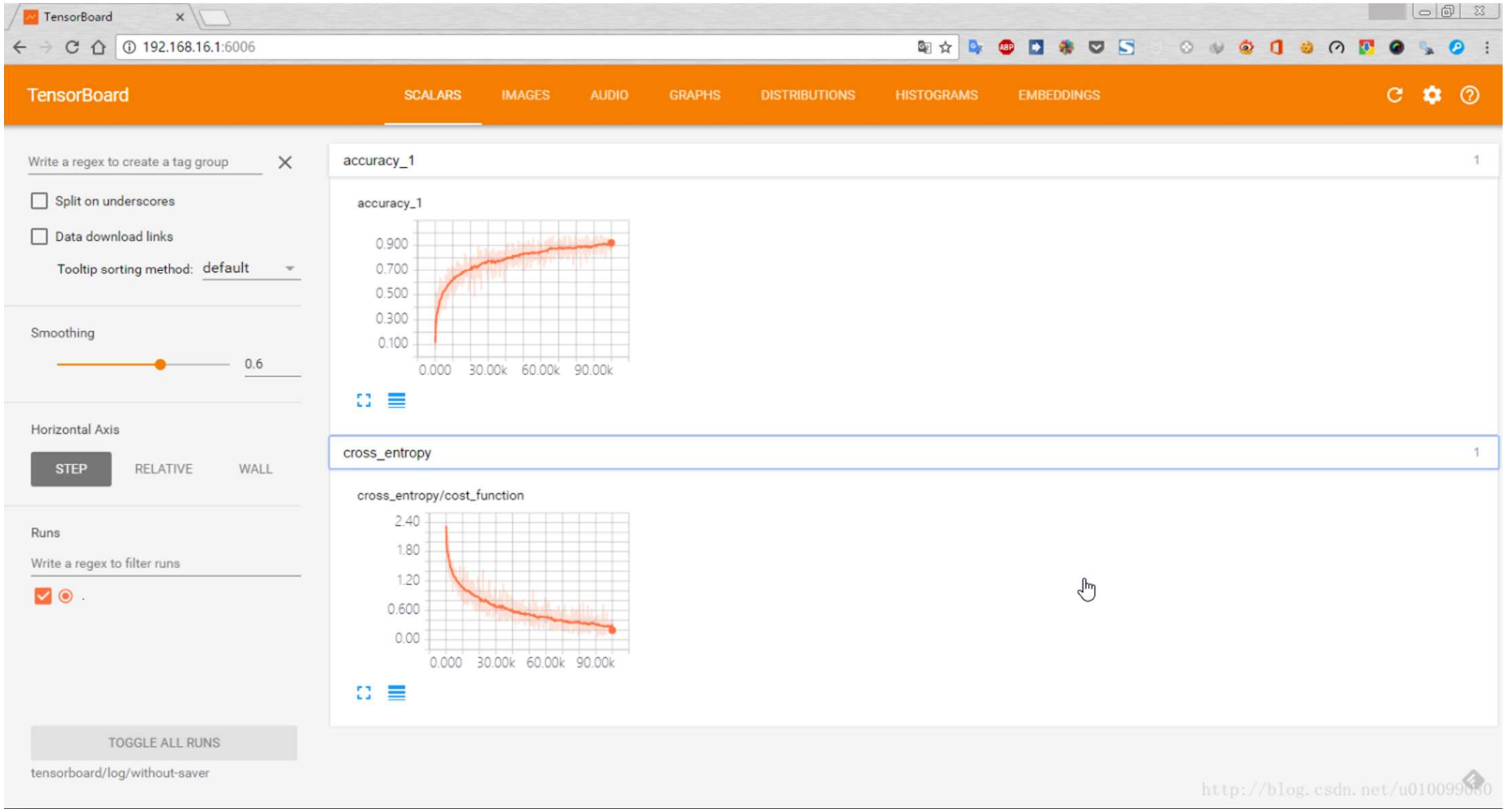


```
# create a placeholder for a vector of 2 elements, type tf.float32
x = tf.placeholder(dtype=tf.float32, shape=[2], name='p')

y = tf.constant(value=[1, 2], dtype=tf.float32, name='c')

z = x + y

with tf.Session() as sess:
    z_val = sess.run(z, feed_dict={x: [3, 4]})
    print(z_val)
```

```
[ 4.  6.]
```

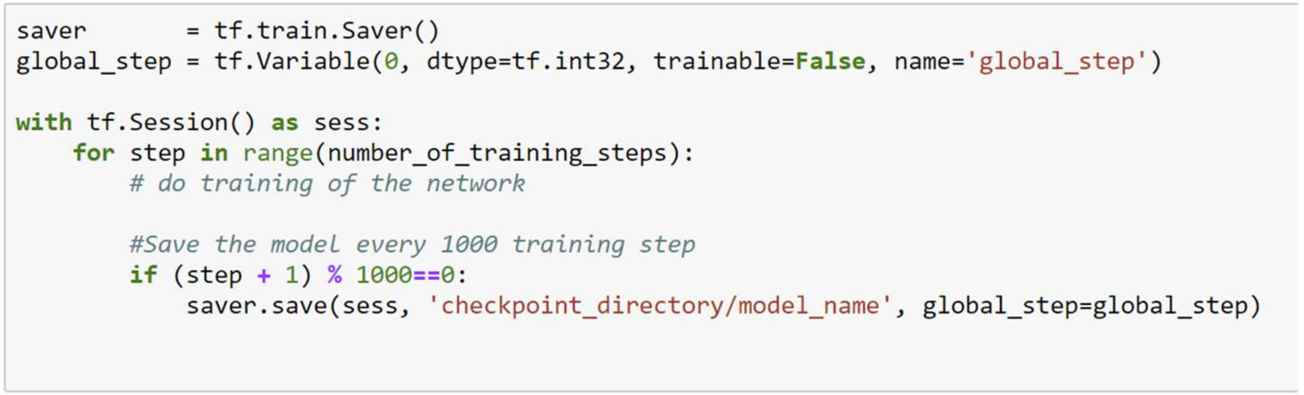


Tensorboard: Visualizing learning



tf.train.saver.save()

- How to save our model every 1000 iteration.

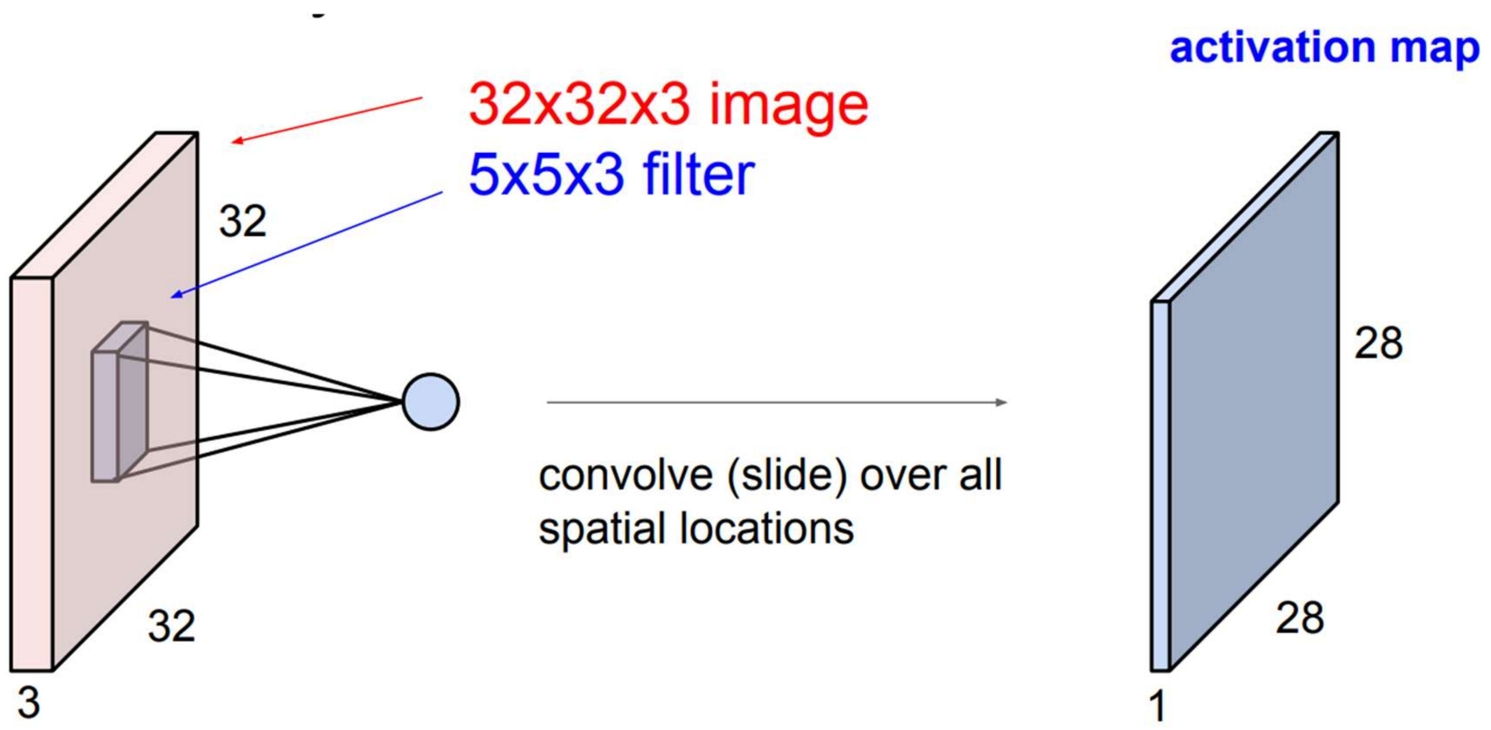


```
saver       = tf.train.Saver()
global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')

with tf.Session() as sess:
    for step in range(number_of_training_steps):
        # do training of the network

        #Save the model every 1000 training step
        if (step + 1) % 1000==0:
            saver.save(sess, 'checkpoint_directory/model_name', global_step=global_step)
```

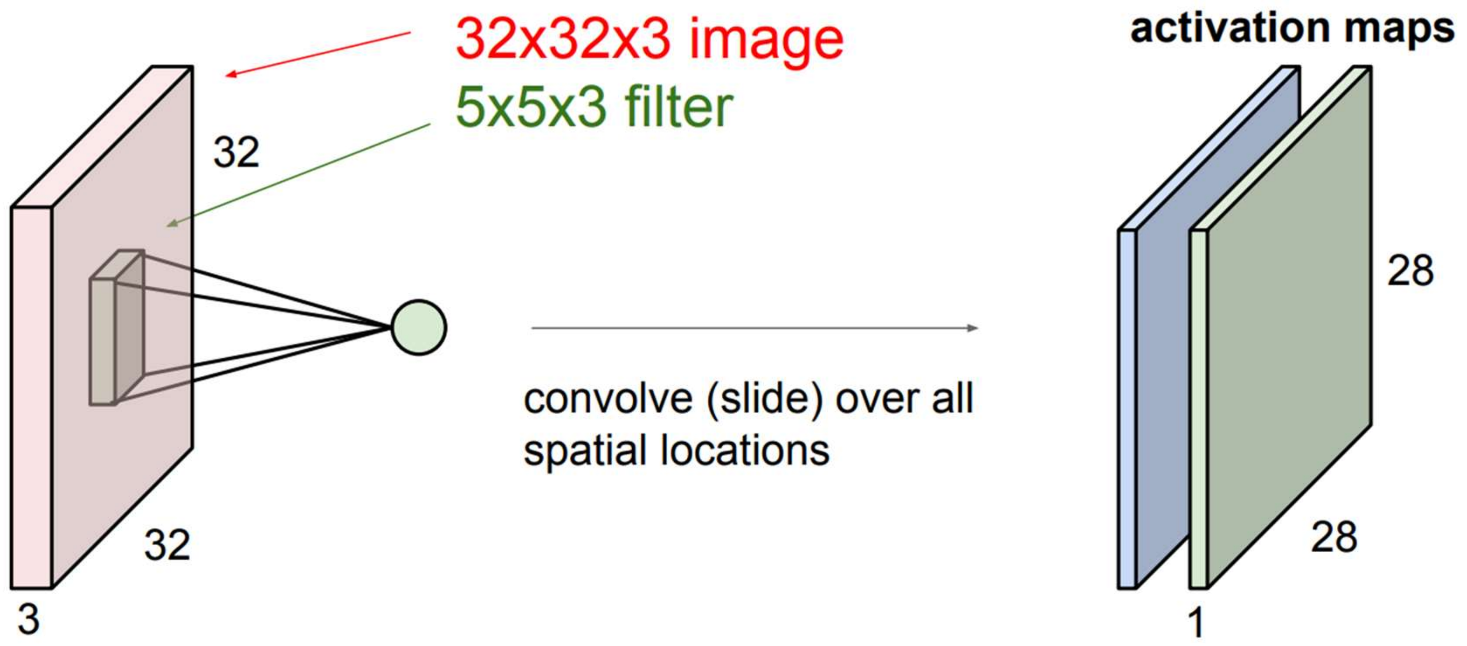
Progress

- TensorFlow
- **Convolutional neural networks**
- Generalization
- Recurrent neural networks
- Deep reinforcement learning

Convolutional layer

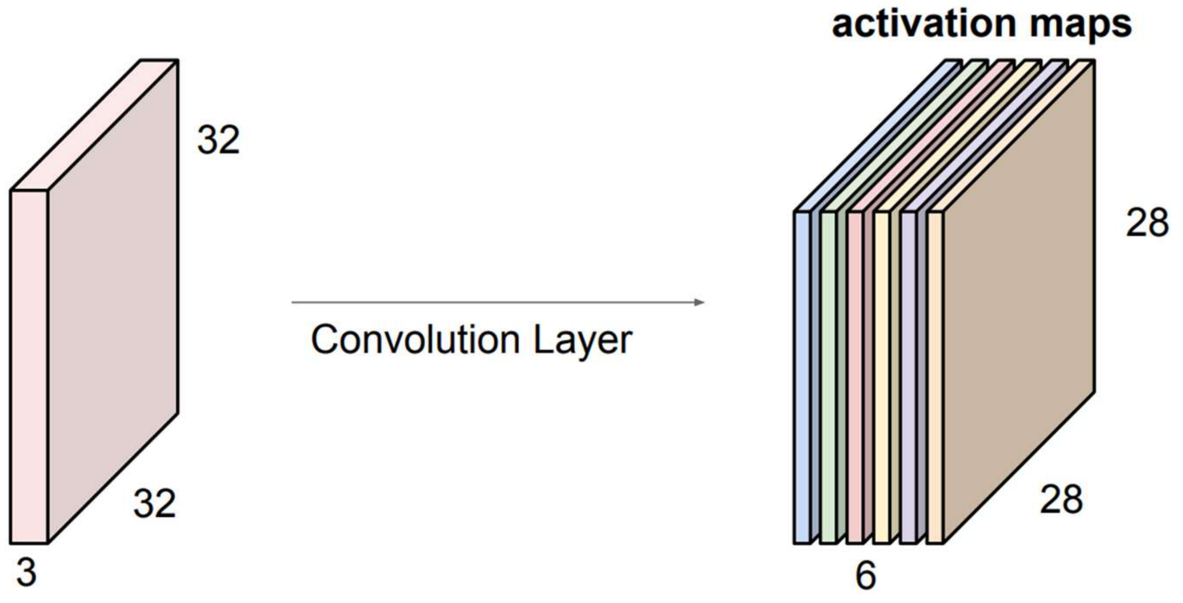


Convolutional layer



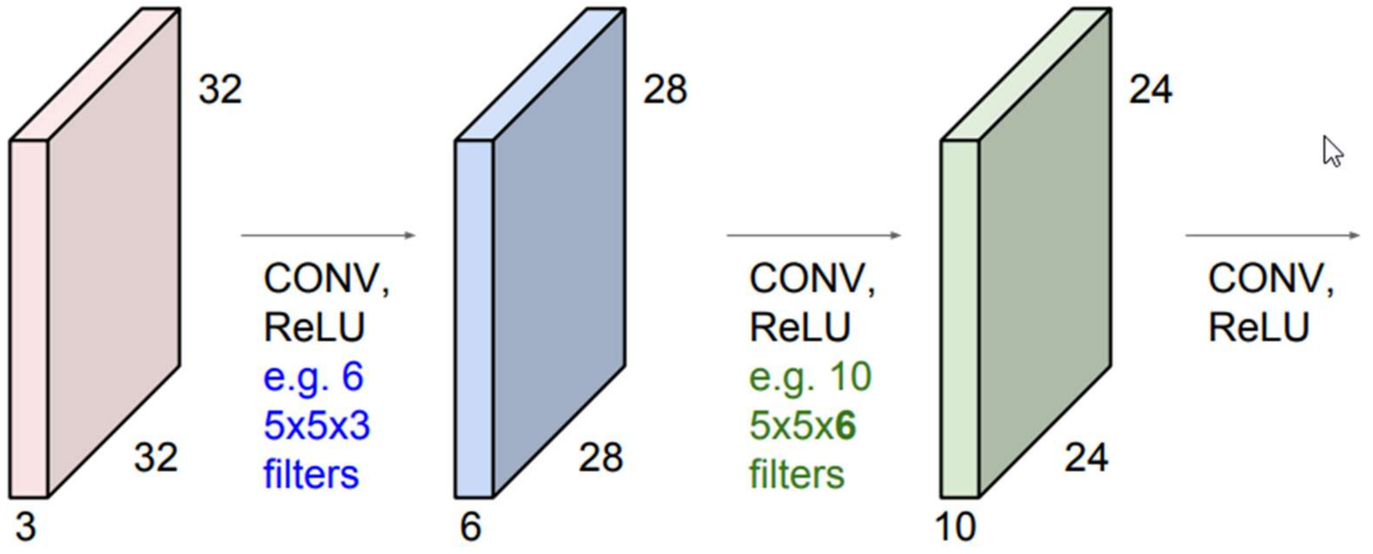
Convolutional layer

- If we filter the input volume 6 times using a 5x5x3 filter, we get a output volume with 6 channels (depth)



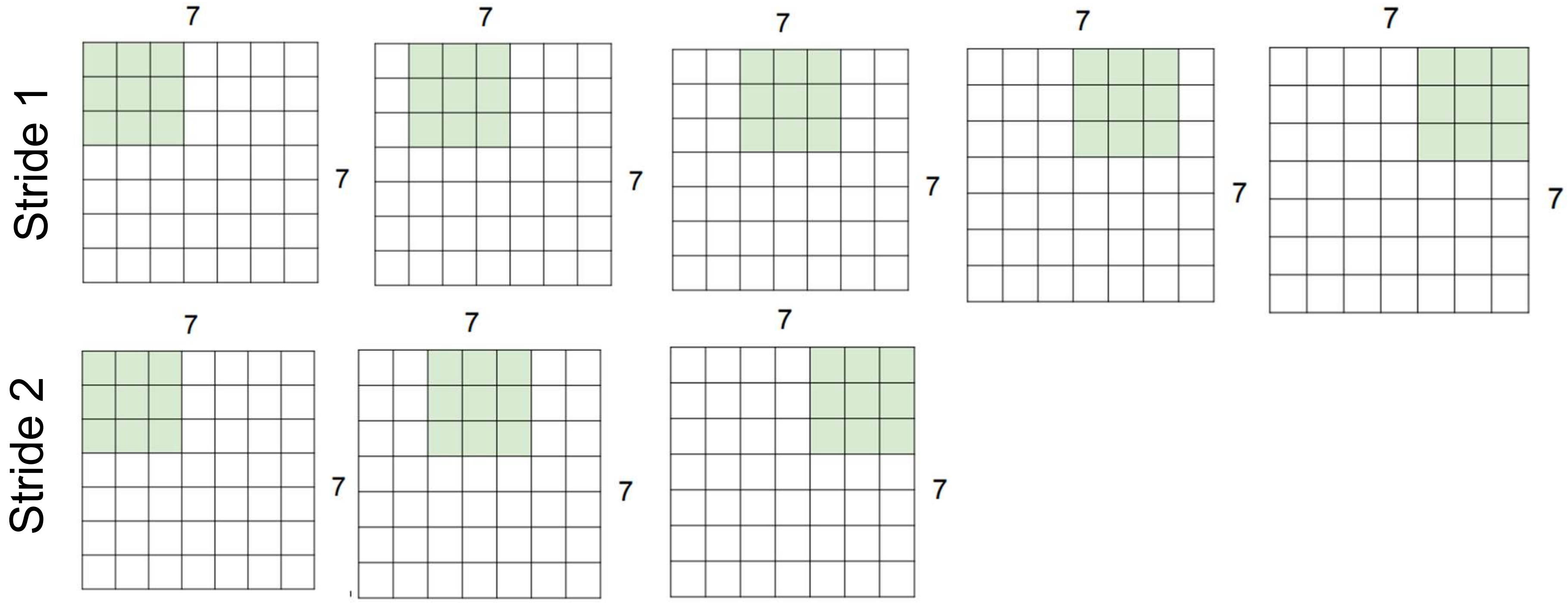
Activations

- We use an activation function separately on all elements of the output volume



Stride

- Stride is the spatial step length in the convolution operation.
- Example: Input volume 7x7x1, kernel (filter) size 3x3x1
- The stride is an important parameter for determining the spatial size of the output volume



Padding

- The output volume can get a lower spatial resolution compared to the input volume. We can solve this by padding the input volume. Common to use zero padding

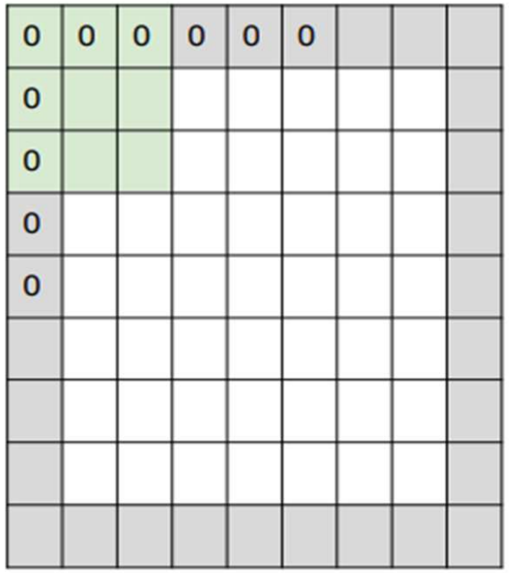
- Abbreviations: Stride (S), filter size (F), input size (N^0), output size (N^1) and padding (P)

- For $S = 1$, we can achieve $N^0 = N^1$ selecting P equal to:

$$P = \frac{(F-1)}{2}$$

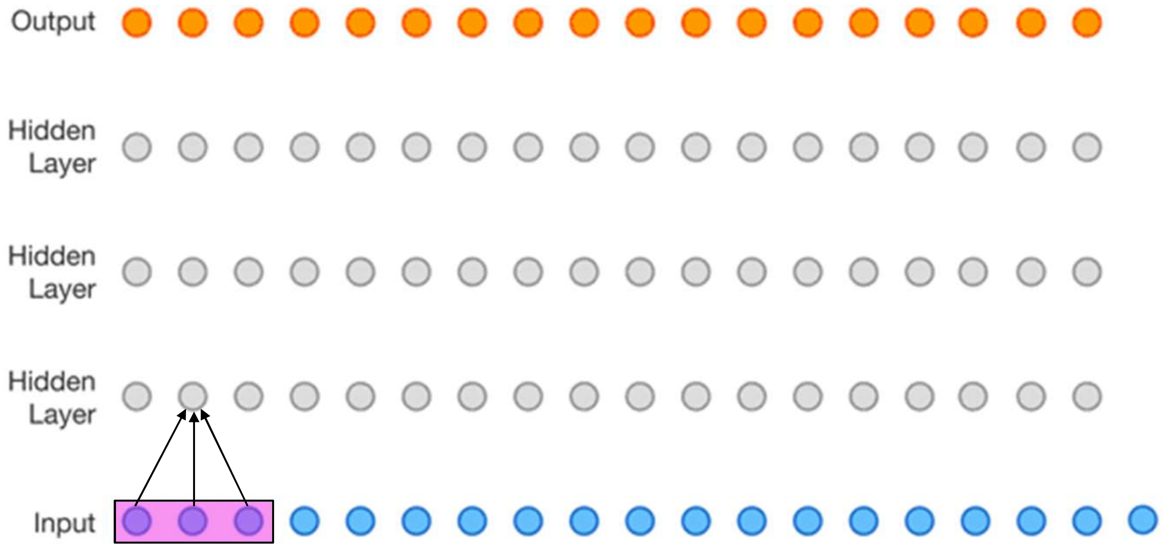
- Calculation of the spatial output size:

$$N^1 = \frac{N^0 - F + 2P}{S} + 1$$



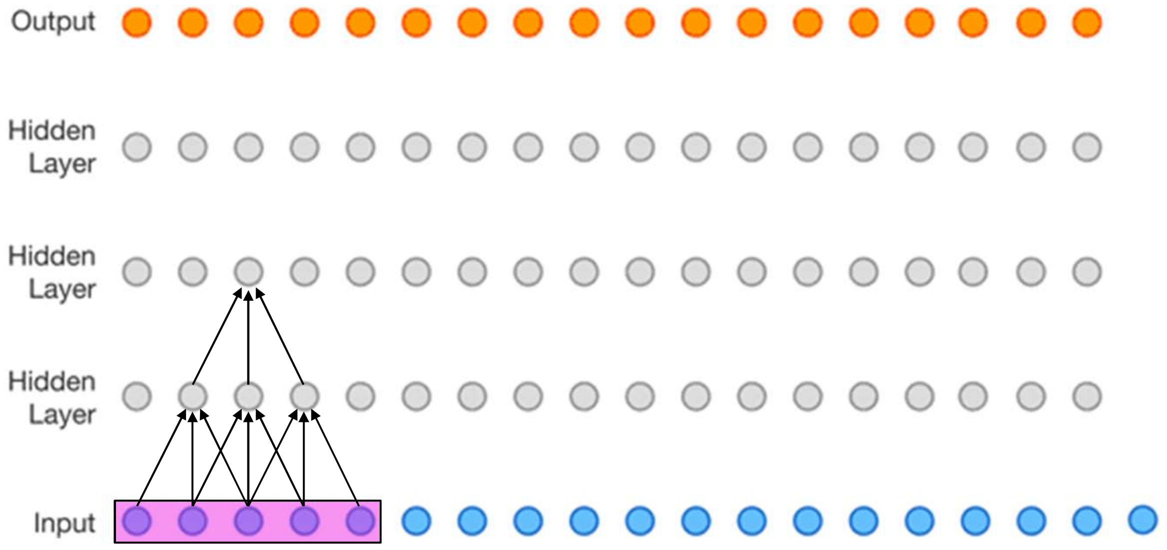
How large area influence the end result?

- With a convolutional network the receptive field increase with each layer
- 3 inputs influence each node in the first hidden layer



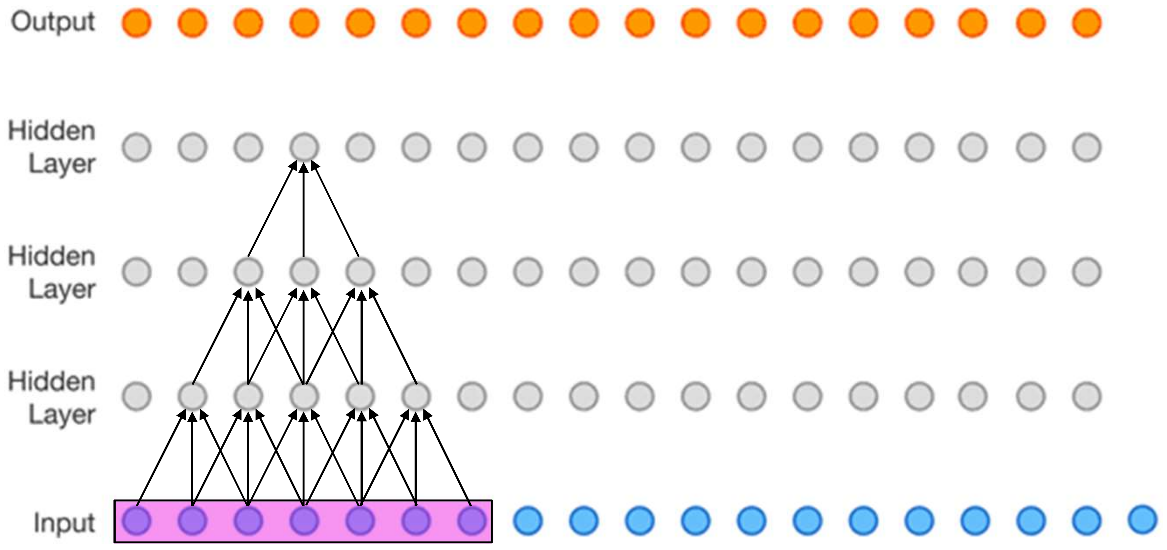
How large area influence the end result?

- With a convolutional network the receptive field increase with each layer
- 3 inputs influence each node in the first hidden layer
- 5 influence the next



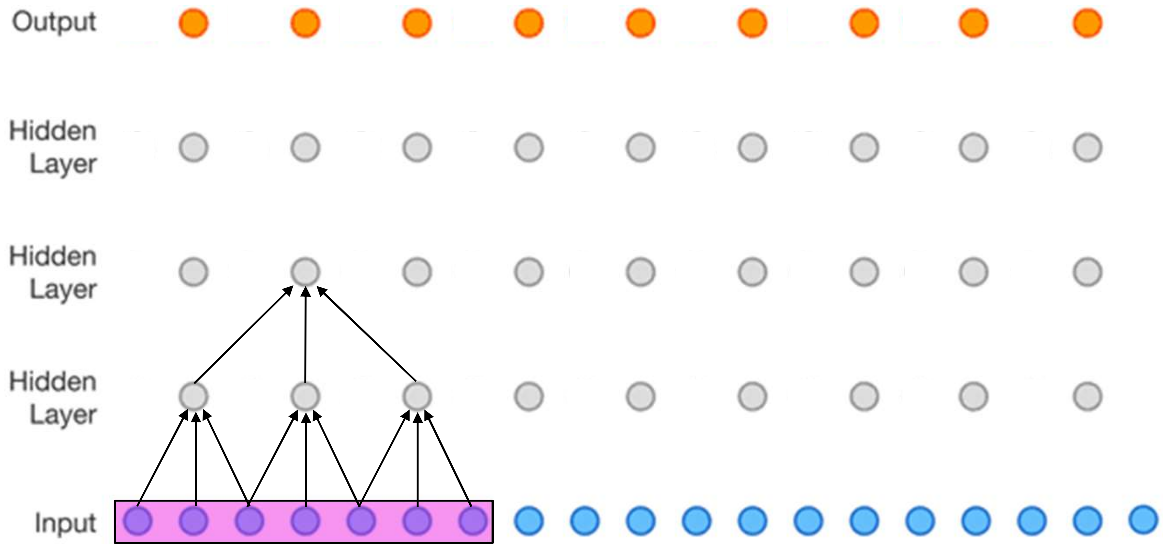
How large area influence the end result?

- With a convolutional network the receptive field increase with each layer
- 3 inputs influence each node in the first hidden layer
- 5 influence the next
- 7 influence the next



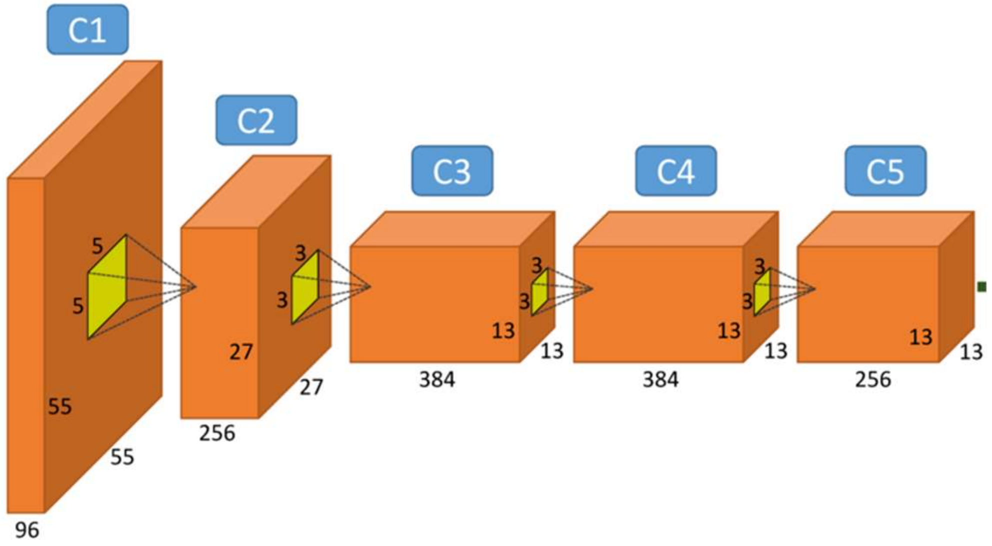
The effect of strided convolutions

- We still cover the whole input
- We have increased the receptive field from 5→7 in hidden layer 2



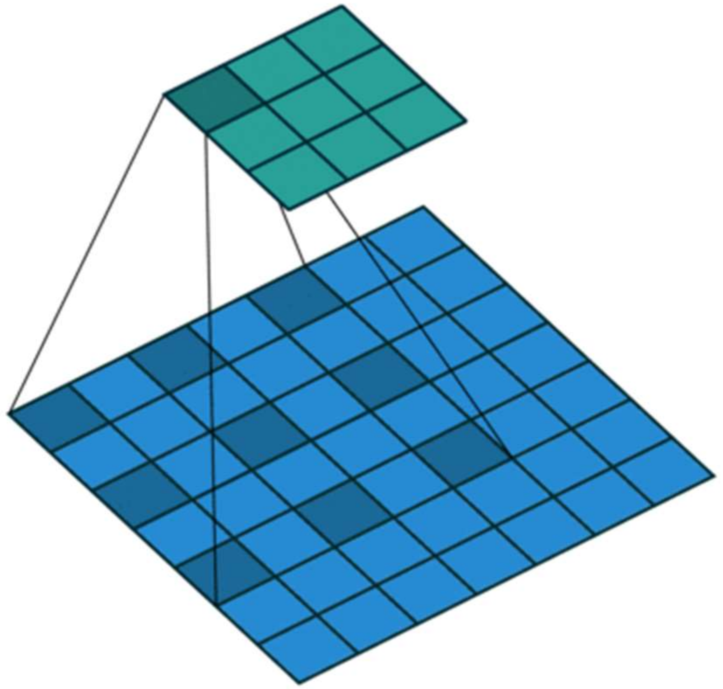
With strides, spatial dimensions will become smaller

- Usually some of the of the network capacity is preserved through an increasing number of channels



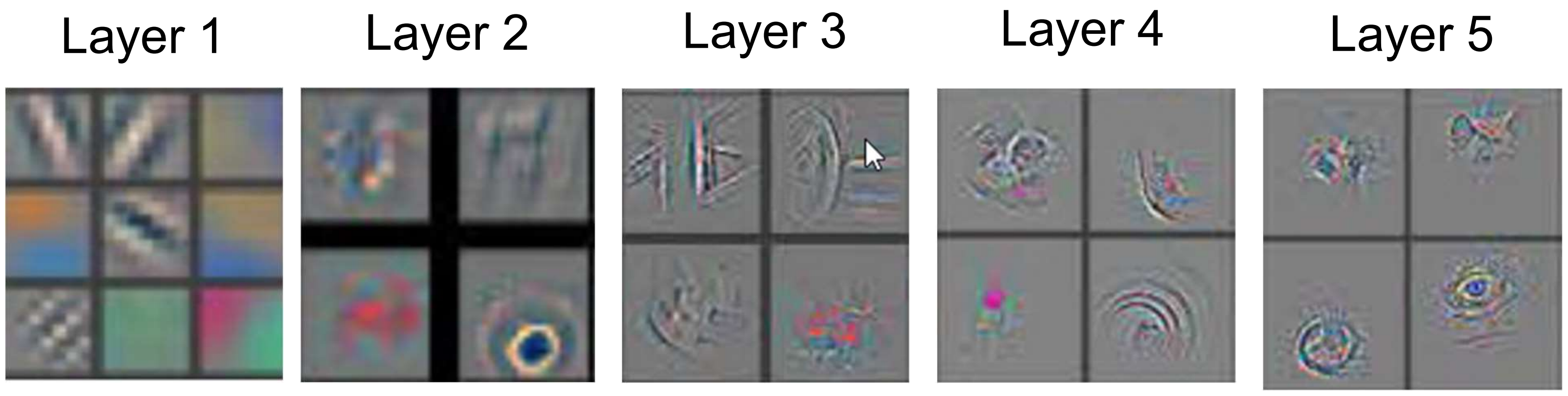
Dilated convolutions

- Skipping values in the kernel
- Same as filling the kernel with every other value as zero
- Still cover all inputs
- Larger kernel with no extra parameters



Visualizing and Understanding deeper layers

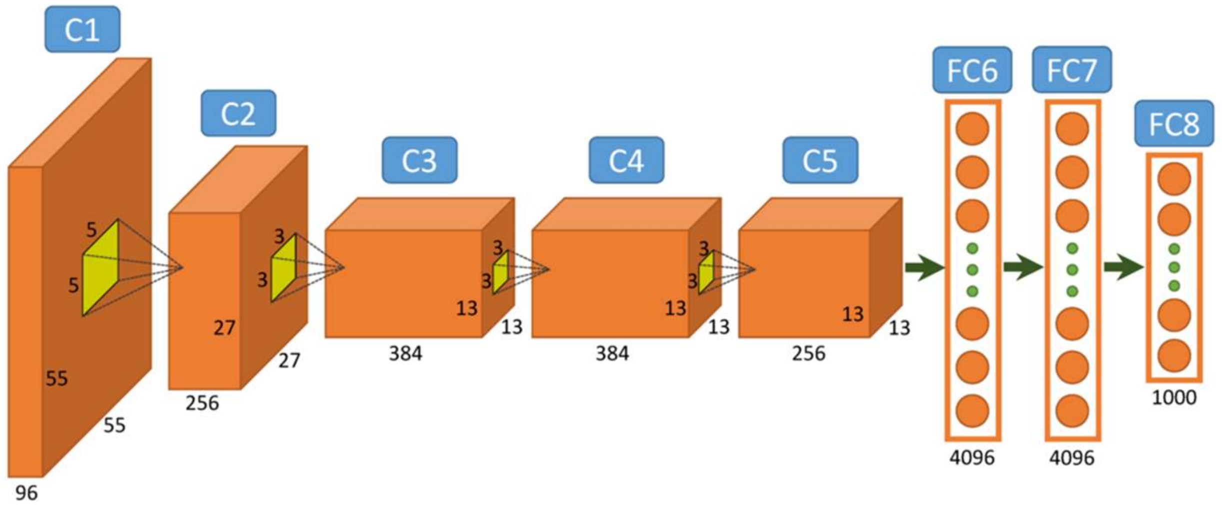
- Looking at the filer coefficient directly at deeper layer is not meaningful.
- Visualization with Deconvnet



Zeiler M.D., Fergus R. (2014) Visualizing and Understanding Convolutional Networks

Hierarchical learning

- A convolution neural network is built up as a hierarchy were the complexity (abstraction) is increased by depth.
- A hierarchical structure is parameter efficient



Progress

- TensorFlow
- Convolutional neural networks
- **Generalization**
- Recurrent neural networks
- Deep reinforcement learning

Notation

- **Formalization supervised learning:**

  - Input: $\boldsymbol{x}$
  - Output: $\boldsymbol{y}$
  - Target function: $f : \mathcal{X} \rightarrow \mathcal{Y}$
  - Data: $(\boldsymbol{x_1}, \boldsymbol{y_1}), (\boldsymbol{x_2}, \boldsymbol{y_2}) \cdots , (\boldsymbol{x_N}, \boldsymbol{y_N})$

    $\downarrow \quad \downarrow \quad \downarrow$

  - Hypothesis: $\mathrm{h} : \mathcal{X} \rightarrow \mathcal{Y}$

Example:

Hypothesis set: $y = w_1 x + w_0$

A hypothesis: $y = 2x + 1$

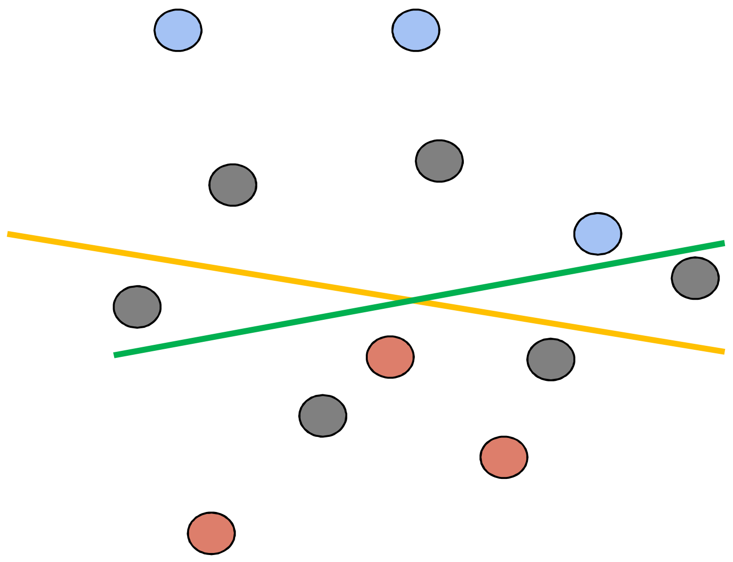
More notation

- **In-sample** (colored): Training data available to find your solution.

- **Out-of-sample** (gray): Data from the real world, the hypothesis will be used for.

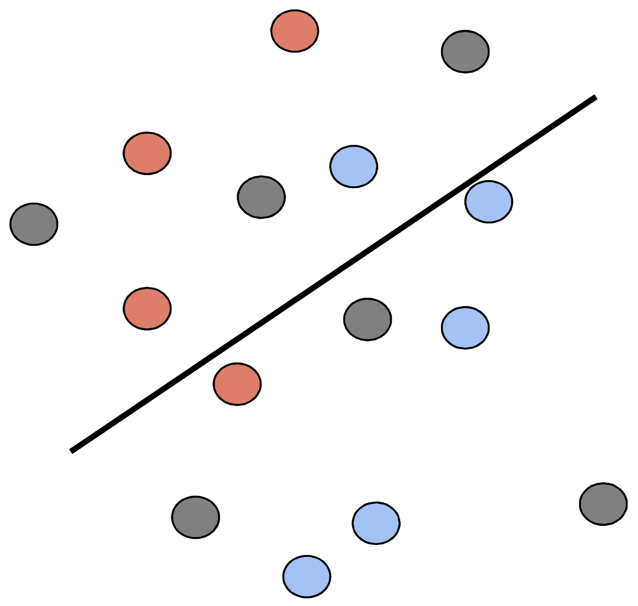
- **Final hypothesis:** 🟨
- **Target hypothesis:** 🟩

- **Generalization:** Difference between the in-sample error and the out-of-sample error



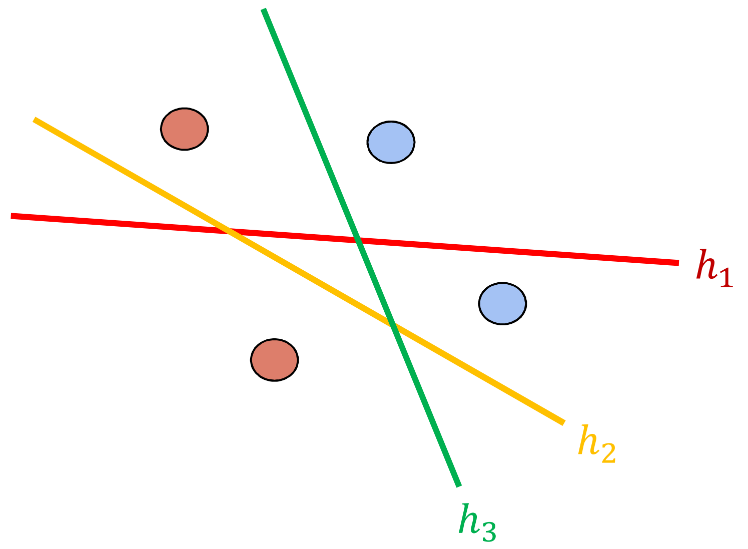
What is the expected out-of-sample error?

- For a randomly selected hypothesis
- The closest error approximation is the **in-sample** error



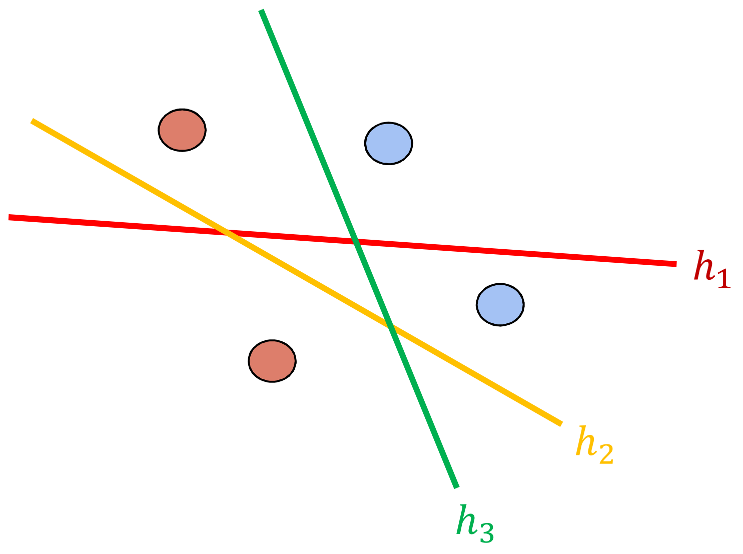
What is training?

- A general view of training:
  - Training is a search through possible hypothesis
  - Use in-sample data to find the best hypothesis



What is the effect of choosing the best hypothesis?

- Smaller **in-sample** error
- Increasing the probability that the result is a coincidence
- The expected **out-of-sample** error is greater or equal to the **in-sample** error

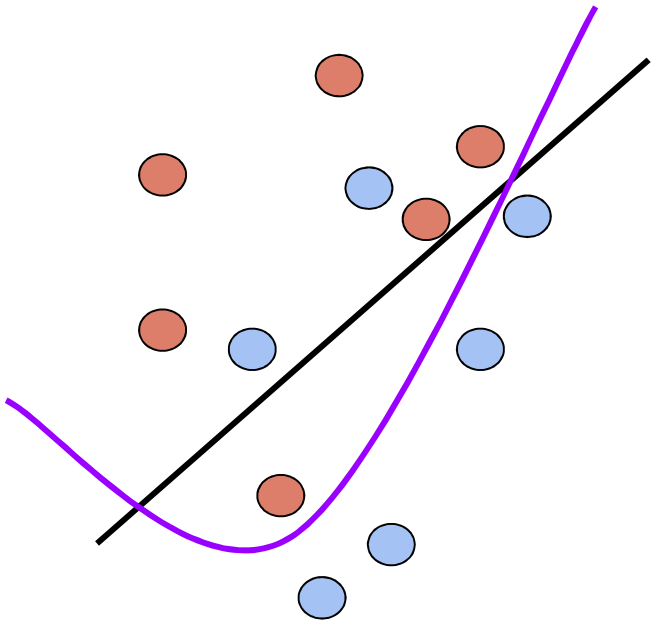


Capacity of the model (hypothesis set)

- The model restrict the number of hypothesis you can find
- Model capacity is a reference to how many possible hypothesis you have
- A linear model has a set of all linear functions as its hypothesis

$$\hat{y} = sign(\mathbf{w}^T\mathbf{x} + b)$$

$$\hat{y} = \mathbf{x}^T W\mathbf{x} + \mathbf{w}^T x + b$$

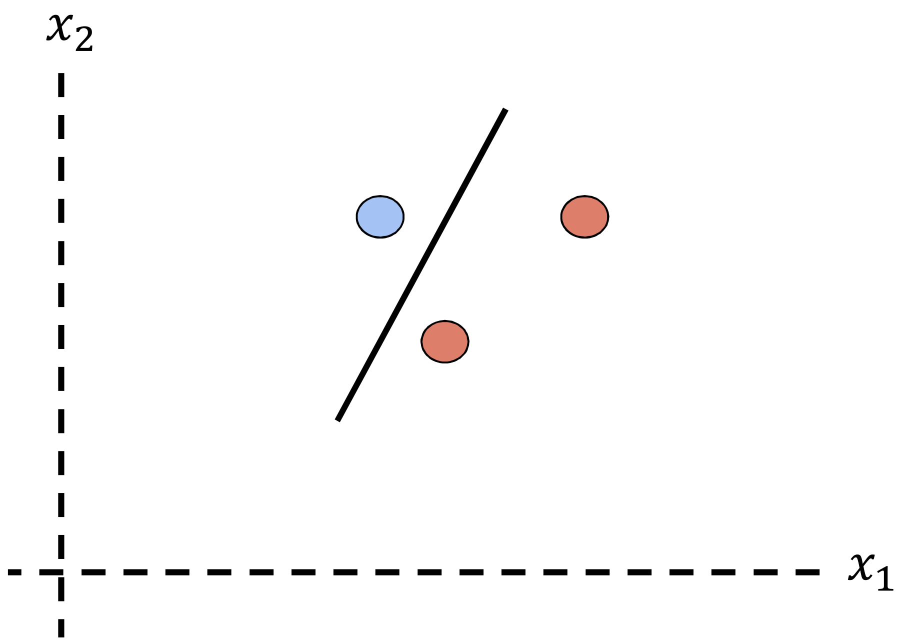


Measuring capacity

- **Vapnik-Chervonenkis (VC) dimension**
  - Denoted: $d_{VC}(\mathcal{H})$
  - Definition:
    - The maximum number of points that can be arrange such that $\mathcal{H}$ can shatter them.

Example VC dimension

- (2D) Linear model $\hat{y} = sign(\mathbf{w}^T\mathbf{x} + b)$
- Configuration (N =3)



Splitting of data

- Training set (60%)
  - Used to train our model
- Validation set (20%)
  - Used to select the best hypothesis
- Test set (20%)
  - Used to get a representative **out-of-sample** error

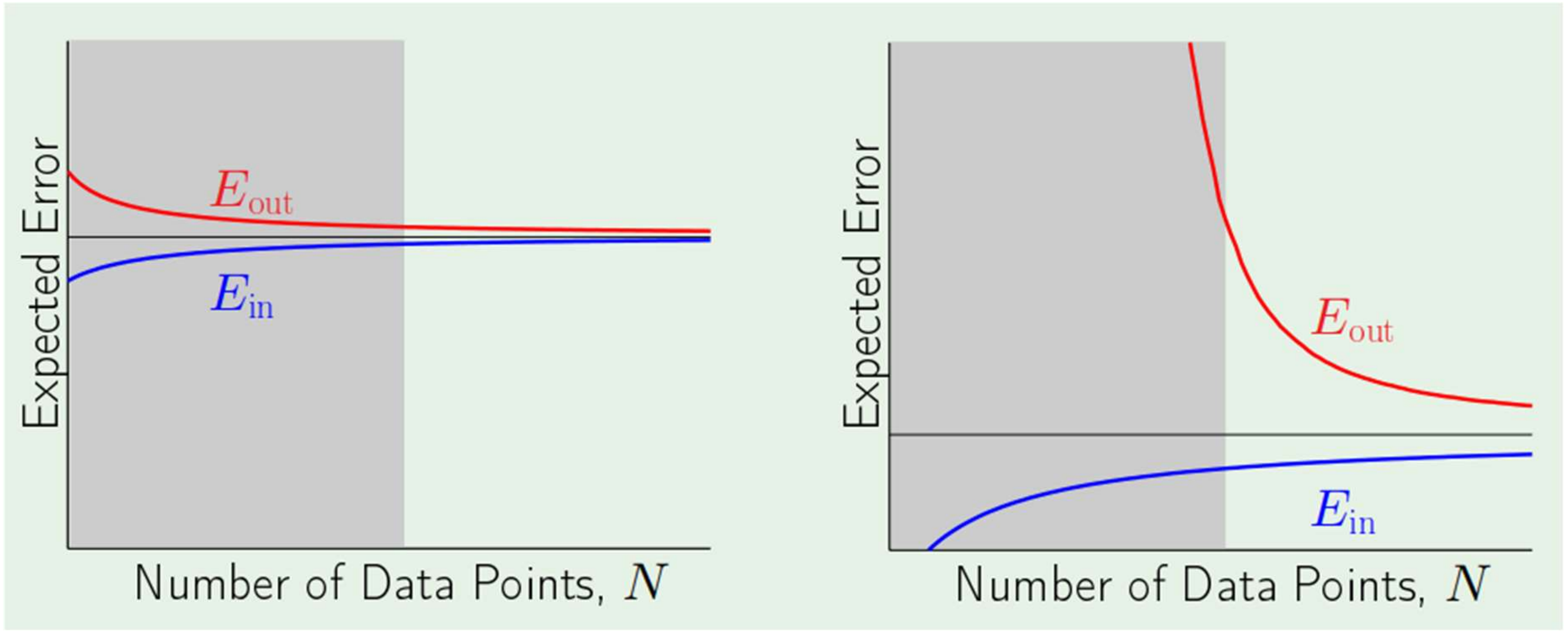
Important! No peeking

- Keep a dataset that you don’t look at until evaluation (**test set**)
- The test set should be as different from your **training set** as you expect the real world to be

Learning curves

Simple hypothesis

Complex hypothesis

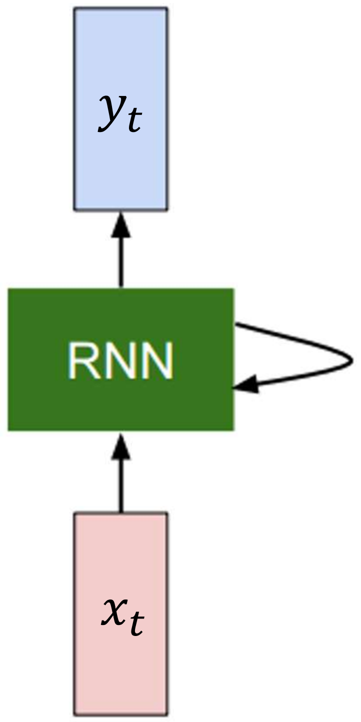


Progress

- TensorFlow
- Convolutional neural networks
- Generalization
- **Recurrent neural networks**
- Deep reinforcement learning

Recurrent Neural Network (RNN)

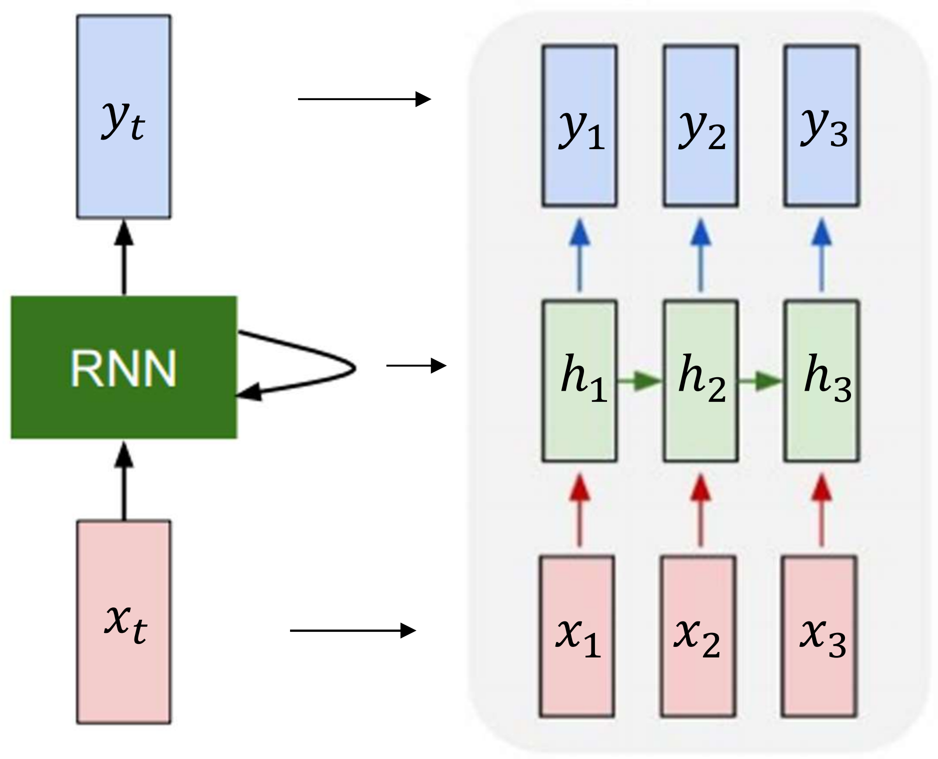
- Takes a new input
- Manipulate the state
- Reuse weights
- Gives a new output



Recurrent Neural Network (RNN)

- Unrolled view

- x_t : The input vector:
- h_t: The hidden state of the RNN
- y_t : The output vector:



(Vanilla) Recurrent Neural Network

- Input vector: x_t
- Hidden state vector: h_t
- Output vector: y_t
- Weight matrices: W_{hh}, W_{hx}, W_{hy}

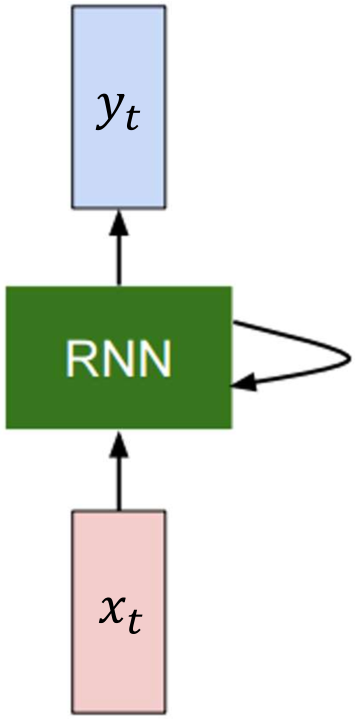
General form:

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

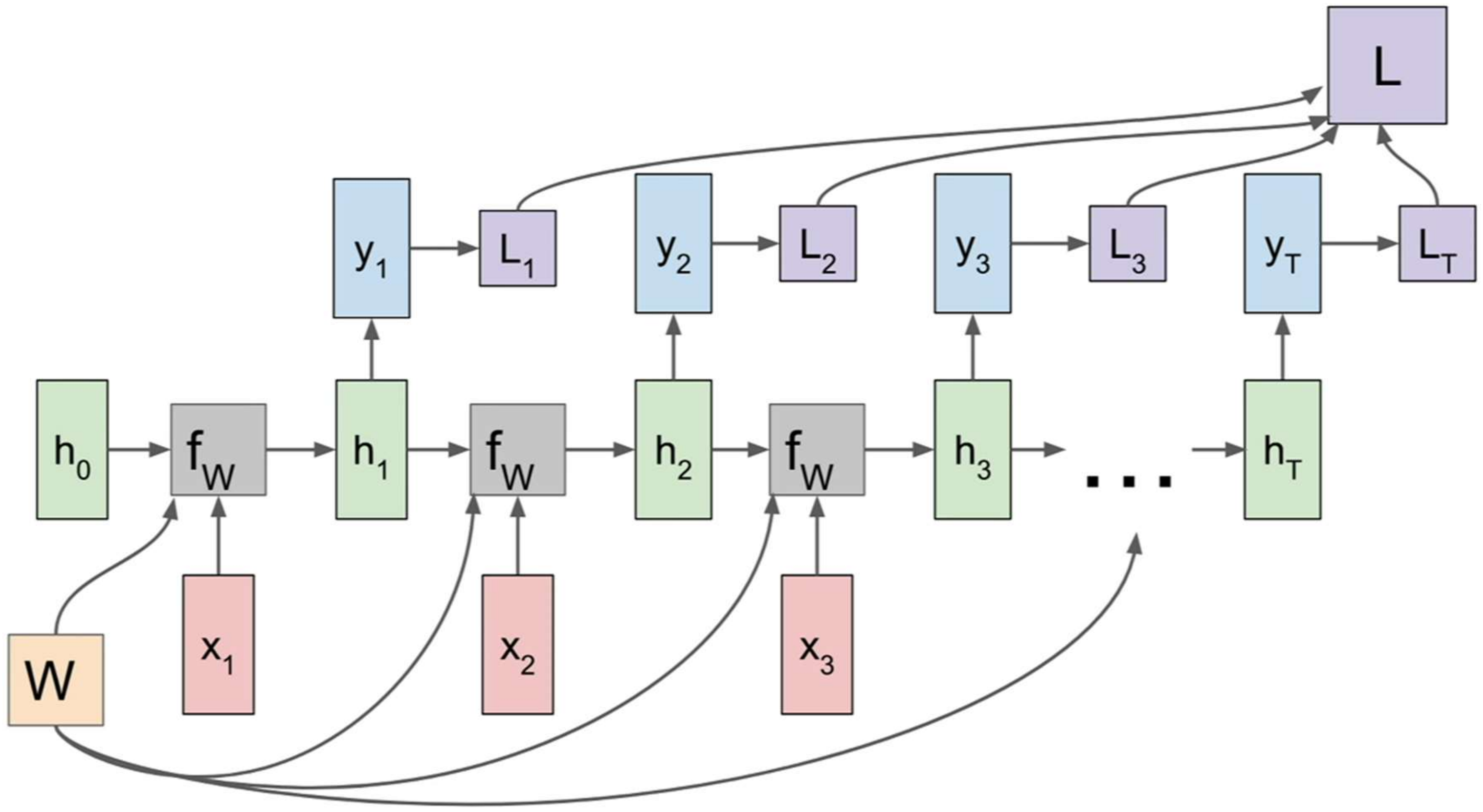
Vanilla RNN:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t + b)$$

$$y_t = W_{hy}h_t$$



RNN: Computational Graph

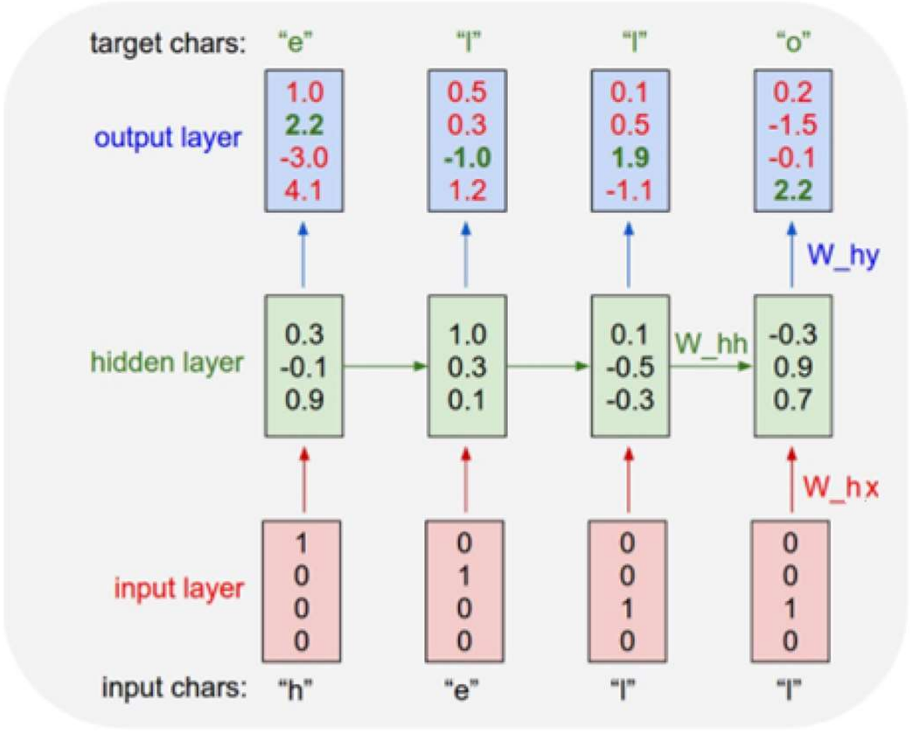


RNN: Predicting the next character

- Task: Predicting the next character
- Training sequence: “hello”
- Vocabulary:
  - [h, e, l, o]
- Encoding: Onehot
- Model:

$h_t = \tanh(W_{hh} h_{t-1} + W_{hx} x_t + b)$

$y_t = W_{hy} h_t$

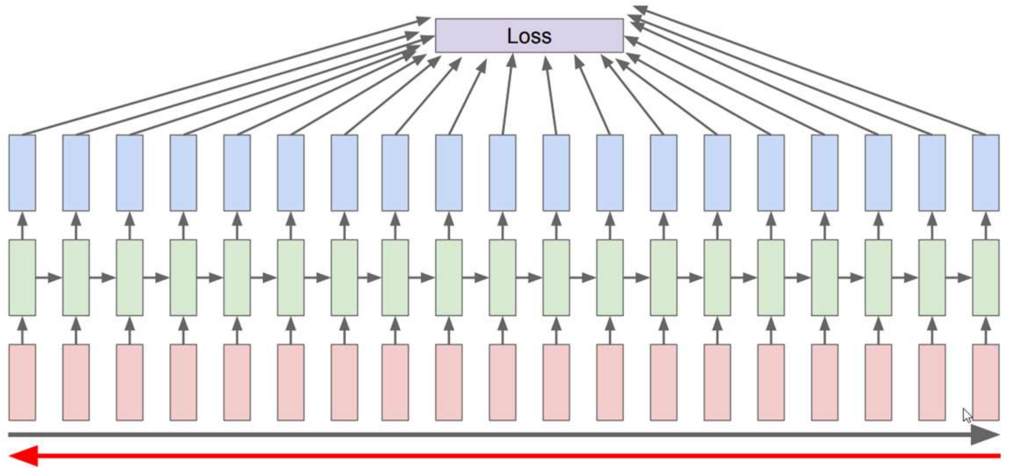


Input-output structure of RNN’s

- One-to-one
- one-to-many
- many-to-one
- Many-to-many
- many-to-many (encoder-decoder)

Vanishing gradients - “less of a problem”

- In contrast to feed-forward networks, RNNs will not stop learning in spite of vanishing gradients.
- The network gets “fresh” inputs each step, so the weights will be updated.
- The challenge is to learning long range dependencies. This can be improved using more advanced architectures.
- Outputs at time step t is mostly effected by the close previous state.



Exploding or vanishing gradients

- $\tanh()$ solves the exploding value problem
- $\tanh()$ does NOT solve the exploding gradient problem, think of a scalar input and a scalar hidden state.

$$h_t = tanh(W_{hh}h_{t-1} + W_{hx}x_t + b)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = [1 - \tanh^2(W_{hh}h_{t-1} + W_{hx}x_t + b)] \cdot W_{hh}$$

The gradient can explode/vanish exponentially in time (steps)

- If $|W_{hh}| < 1$, vanishing gradients
- If $|W_{hh}| > 1$, exploding gradients

Gated Recurrent Unit (GRU)

GRU is adding or removing to the state, not “transforming the state”

Vanilla RNN

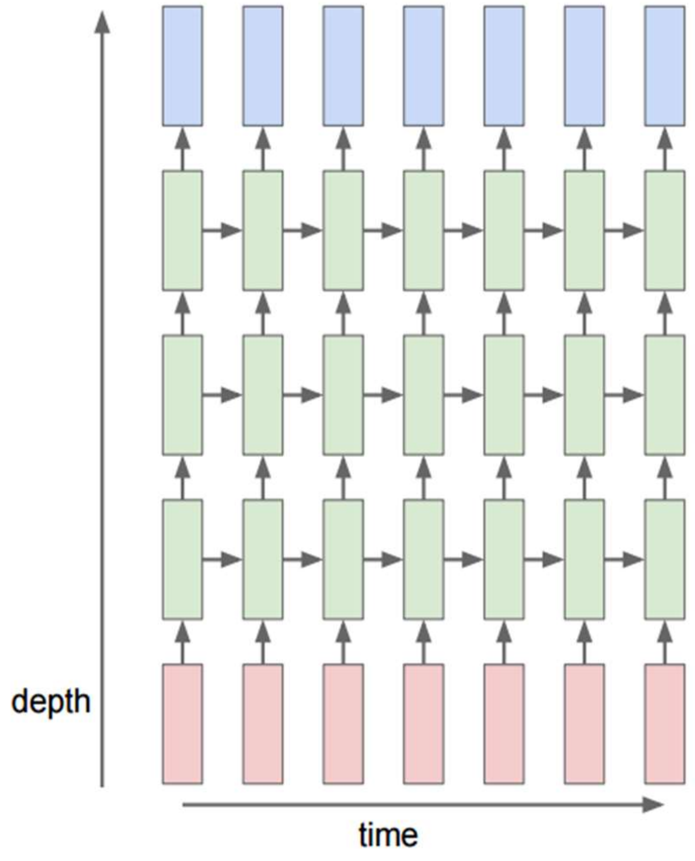
- Cell state $h_t = tanh(W^{hx}x_t + W^{hh}h_{t-1} + b)$

GRU

With Γ^r as ones and Γ^u as zeros, GRU → Vanilla RNN

Multi-layer Recurrent Neural Networks

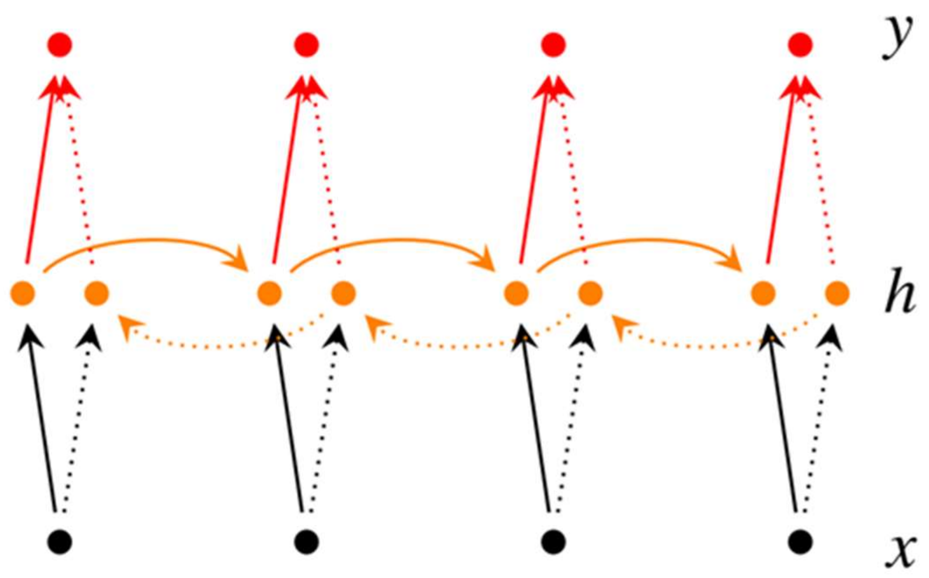
- Multi-layer RNN can be used to enhance model complexity
- Similar as for feed forward neural networks, stacking layers creates higher level feature representation
- Normally, 2 or 3 layer deep, not as deep as conv nets
- More complex relationships in time



Bidirectional recurrent neural network

- The blocks can be vanilla, LSTM and GRU recurrent units
- Real time vs post processing

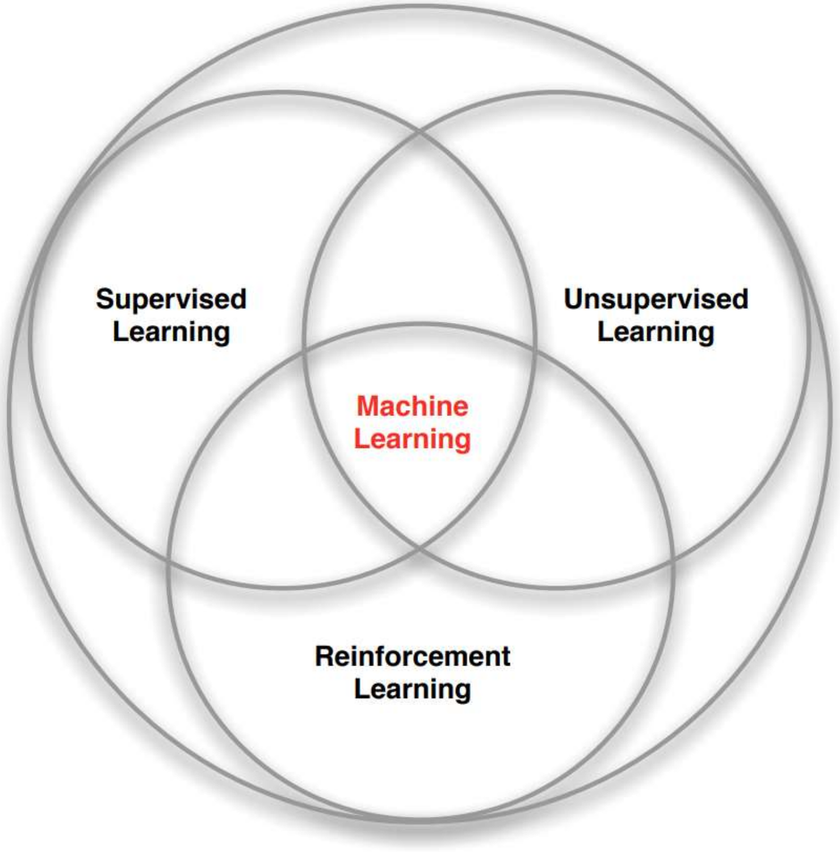
$$\overrightarrow{h}_t = f\left(\overrightarrow{W}_{hx}x_t + \overrightarrow{W}_{hh}\overrightarrow{h}_{t-1}\right)$$
$$\overleftarrow{h}_t = f\left(\overleftarrow{W}_{hx}x_t + \overleftarrow{W}_{hh}\overleftarrow{h}_{t+1}\right)$$
$$y_t = g(W_{hy}\left[\overrightarrow{h}_t, \overleftarrow{h}_t\right] + b)$$



Progress

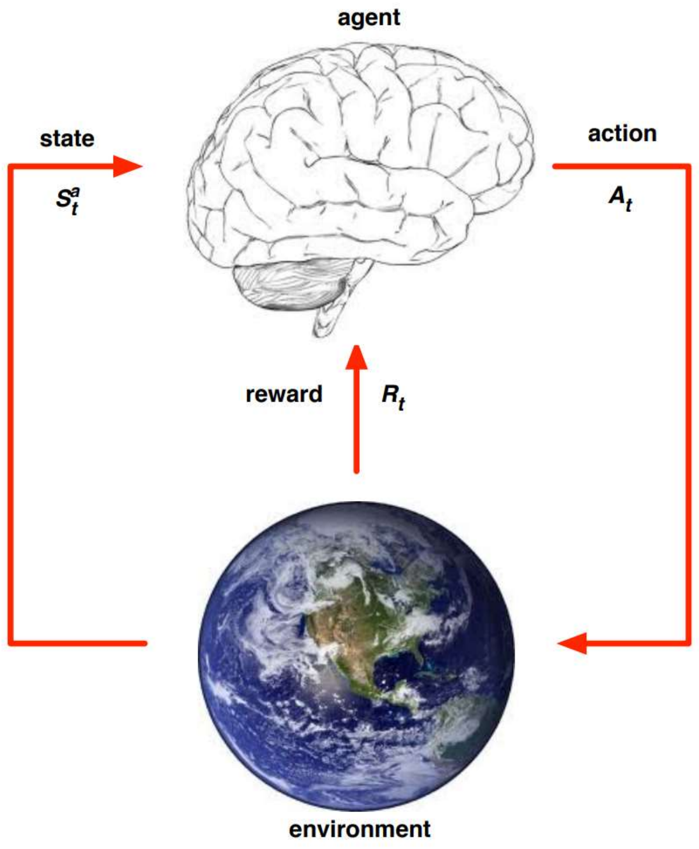
- TensorFlow
- Convolutional neural networks
- Generalization
- Recurrent neural networks
- **Deep reinforcement learning**

Branches of Machine Learning



Reinforcement learning

- Reinforcement Learning ~ Science of decision making
- In RL an agent learns from the experiences it gains by interacting with the environment.
- The goal is to maximize an accumulated reward given by the environment.
- An agent interacts with the environment via states, actions and rewards.



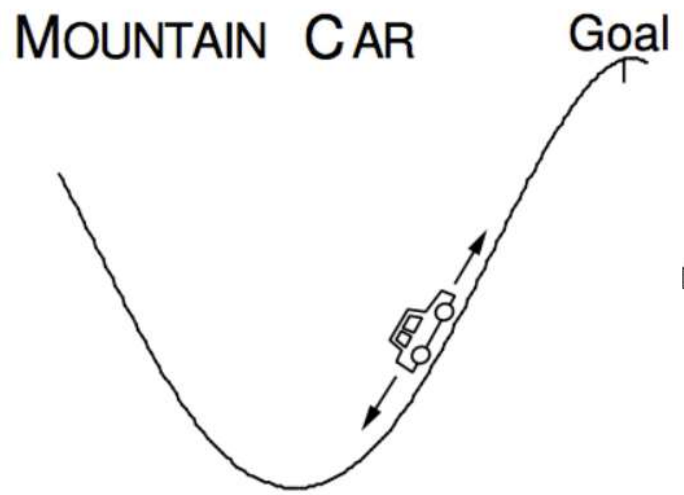
RL Course by David Silver

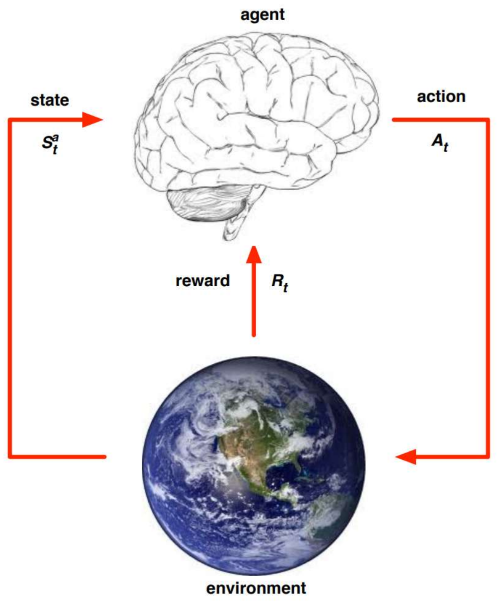
Reinforcement learning

- What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d data)
  - Agent's actions affect the subsequent data it receives

Mountain Car

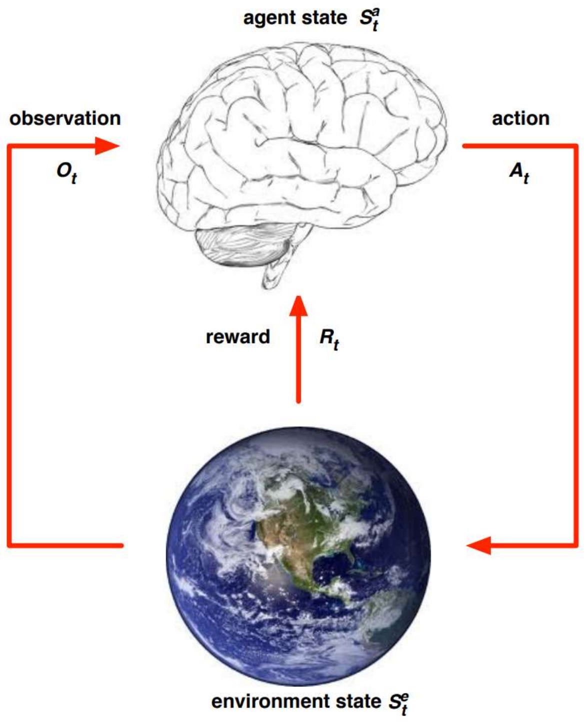
- **Objective:**
  - Get to the goal
- **State variables:**
  - Position and velocity
- **Actions:**
  - Motor: Left, Neutral, right
- **Reward:**
  - (-1) for each time step





History (trajectory) and State

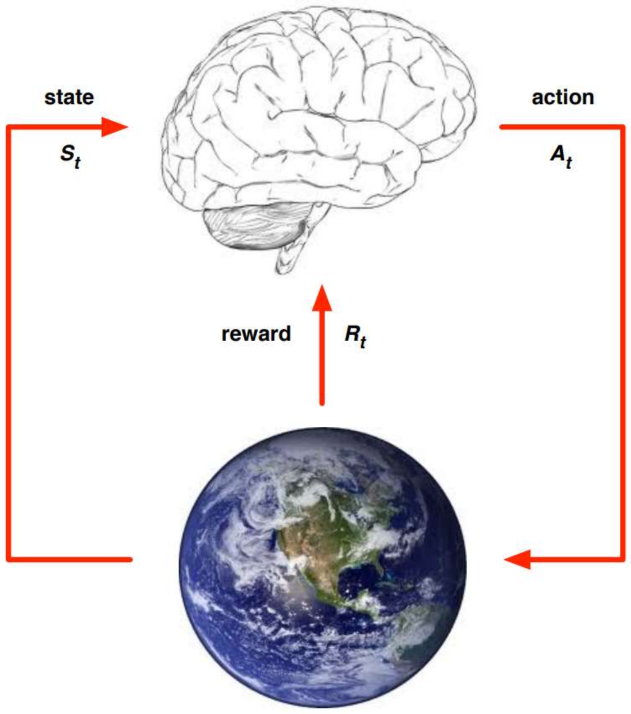
- **History / trajectory :**
  - $H_t = \tau_t = O_1, A_1, R_1, \; O_2, A_2, R_2, \dots, \; O_t, A_t, R_t$
- **Full observatory:**
  - Agent direct observe the environment state.
  - $O_t = S_t^e = S_t^a$
- **State:**
  - The state is a summary (of the actions and observations) that determines what happens next given an action.
  - $S_t = f(H_t)$
- **Partially observability:**
  - The agent indirectly observes the environment.
  - Robot with a camera



Markov Property

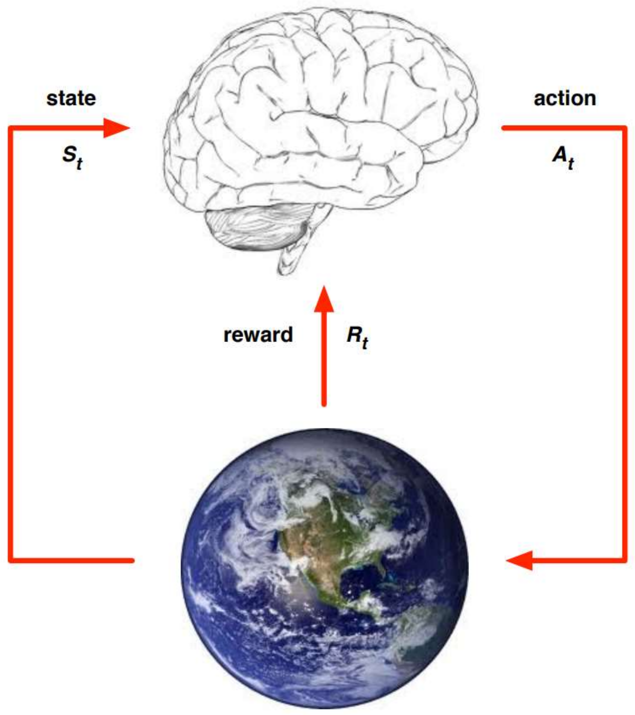
- **Definition**:
  - A state S_t is Markov if and only if:
    $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1} \mid S_1, S_2, \dots, S_t]$

- The state capture all relevant information from the history
- The state is sufficient to describe the statistics of the future.



Reward and Return

- The **reward**, R_t, is a scalar value the agent receives for each step t.
- The **return,** G_t, is the total discounted accumulated reward form a given time-step t.
  - $G_t = R_t + \gamma R_{t+1} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$
- **Discount factor**:
  - We can apply a discord factor, $\gamma \in [0,1]$, to weight how we evaluate return.
- The agent's goal is to maximize the **return**



Markov Decision Process (MDP)

- The mathematical formulation of the reinforcement learning (RL) problem.

- A **Markov Decision Process** is a tuple, $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$, where every state has the Markov property.

S: A finite set of states

A: A finite set of *actions*

P: The transition probability matrix

$$P^a_{s_t s_{t+1}} = \mathbb{P}[S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t]$$

R: Reward function:

$$R^a_s = \mathbb{E}[S_t = s_t,\ A_t = a_t]$$

γ: is a discount factor $\gamma \in [0,1]$

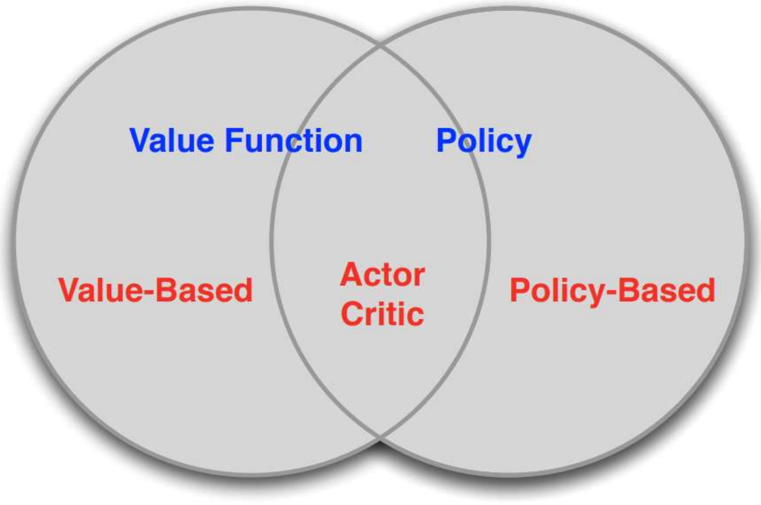
Objective

- Our goal is it find the policy which maximize the accumulated reward:

  $G_t = R_t + \gamma R_{t+1} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$

- Due to the randomness of the transition probability and the reward function, we use the expected value in the definition of the optimal policy.

$$\pi_* = \underset{\pi}{\arg\max}\, \mathbb{E}\,[G_t]$$



Bellman (optimality) equation

- Lets define the optimal Q-value (*action-value*) function, Q_*, to be the maximum expected reward given an state, action pair.

$$Q_*(s_t, a_t) = \max_{\pi} \mathbb{E}_{\pi}[\, G_t \mid A_t = a_t, S_t = s_t]$$

- The optimal Q-value function, Q_*, satisfy the following form of the bellman equation:

$$Q_*(s_t, a_t) = \mathbb{E}\left[\, R_t + \gamma \max_{a_{t+1}} Q_*(s_{t+1}, a_{t+1}) \mid A_t = a_t, S_t = s_t \right]$$

- Note: The optimal policy, π_*, is achieved by taking the action with the highest Q-value.
- Note: We still need the expectation, as the randomness of the environment is unknown.

Exploration vs Exploitation

- “The “max” property while sampling new episodes can lead to suboptimal policy”

- **Exploitation:**
  - By selecting the action with the highest q-value while sampling new episodes, we can refine our policy efficiently from an already promising region in the state action space.

- **Exploration:**
  - To find a new and maybe more promising region within the state action space, we do not want to limit our search in the state action space.
  - We introduce a randomness while sampling new episodes.
  - With a probability of ϵ lets choose a random action:

$$\pi(a|s) = \begin{cases} a_* = \underset{a \in A}{\operatorname{argmax}} Q(s,a), & with\ probabillity\ \ 1-\epsilon \\ \quad random\ action, & with\ \ probabillity\ \ \epsilon \end{cases}$$

Function approximation

- In the Gridworld example, we stored the state-values for each state. What if the state-action space is too large to be stored e.g. continuous?
- We approximate the Q-value using a parameterized function e.g. neural network.

$$\hat{Q}(s, a, \theta) \approx Q(s, a)$$

- We want the function to generalize:
  - Similar states should get similar action-values, $\hat{Q}(s, a, \theta)$ can also generalize to unseen states. A table version would just require to much data.
- In supervised learning:
  - Building a function approximation vs memorizing all images (table).

Solving for the optimal policy: Q-learning

- **Goal**: Find a Q-function satisfying the Bellman (optimality) equation.
- **Idea**: The Q-value at the last time step is bounded by the true Q-value, the correctness of the Q-value estimates increase with time-steps.
- **Init:** Initialize the weights in the neural network e.g. randomly.

$$Q_*(s_t, a_t, \theta_i) = \mathbb{E}\left[R_t + \gamma \max_{a_{t+1}} Q_*(s_{t+1}, a_{t+1}, \theta_{i-1}) \mid A_t = a_t, S_t = s_t \right]$$

- Reference:

$$y_i = \mathbb{E}\left[R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta_{i-1}) \mid A_t = a_t, S_t = s_t \right]$$

- Loss:

$$L_i(\theta_i) = \mathbb{E}_{s_t, s_{t+1}, a_t, r_t \sim D_i}\left[\left(y_i - Q(s_t, a_t, \theta_i) \right)^2 \right]$$

D_i is your dataset with state action pairs s_t, s_{t+1}, a_t, r_t

Policy based methods

- **Value function based methods**:
  - Learning the expected future reward for a given action.
  - The policy was to act greedily or epsilon-greedily on the estimated values.
- **Policy based methods**:
  - Learning the probability that an action is good directly.
- **Advantage of Policy based methods:**
  - We might need a less complex function for approximating the best action compared to estimate the final reward.
  - Example: Think of Pong

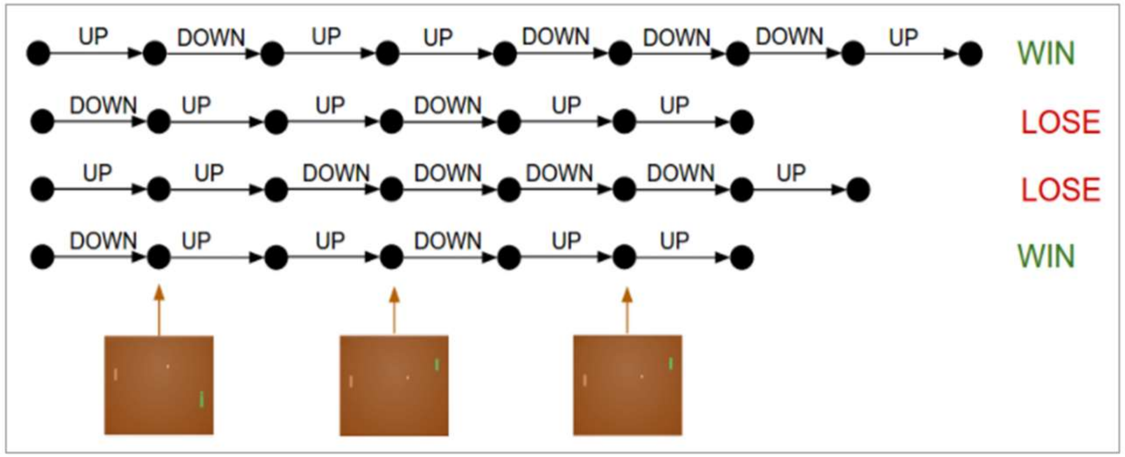
Policy based methods

- **Goal:**
  - The goal is to use experience/samples to try to make a policy better.

- **Idea:**
  - If a trajectory achieves a high reward, the actions were good
  - If a trajectory achieves a low reward, the actions were bad
  - We will use gradients to enforce more of the good actions and less of the bad actions. Hence the method is called Policy Gradients.

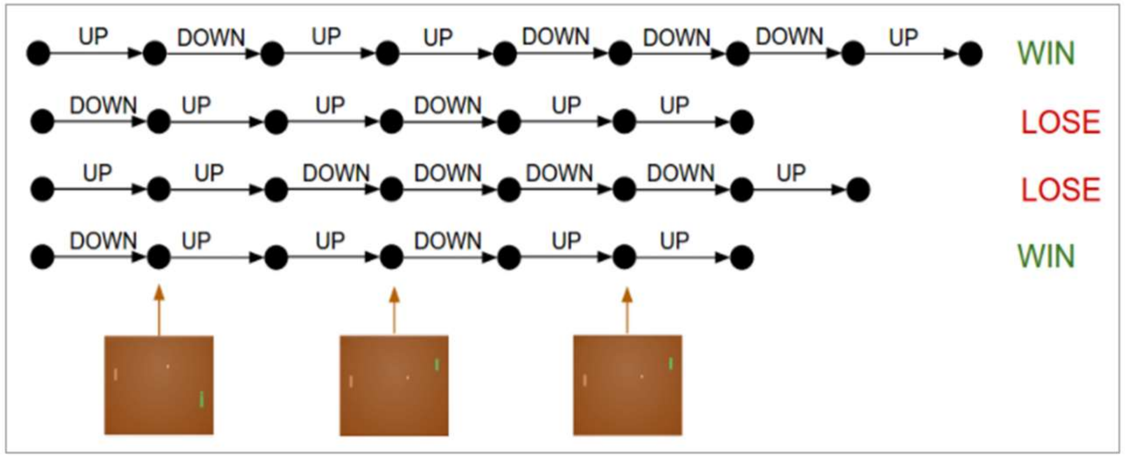
Playing games of Pong

- Examples of games/episodes
- You play a lot of actions and receive an reward at the end
- You get a result, WIN! Great, but how do you know which action, caused the victory?
  - Well… you don't

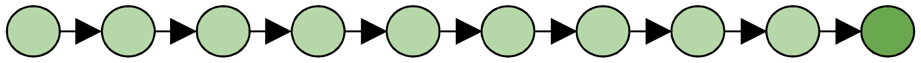


Which action caused the final results?

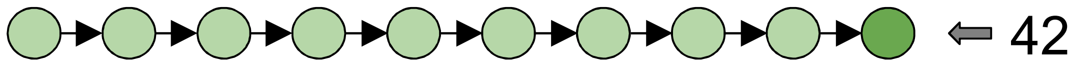
- In a winning series there may be many non-optimal actions
- In a losing series there may be good actions
- The **true** effect is found by averaging out the noise, as winnings series tend to have more good action and visa versa



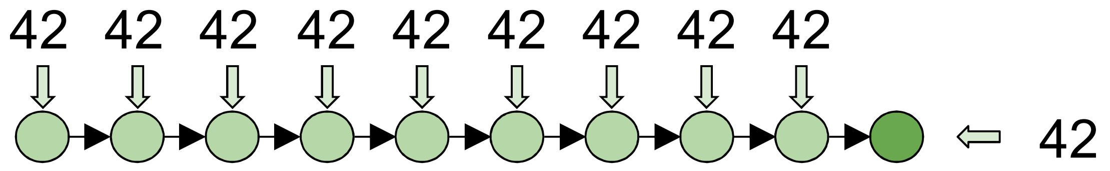
Policy gradients: High variance



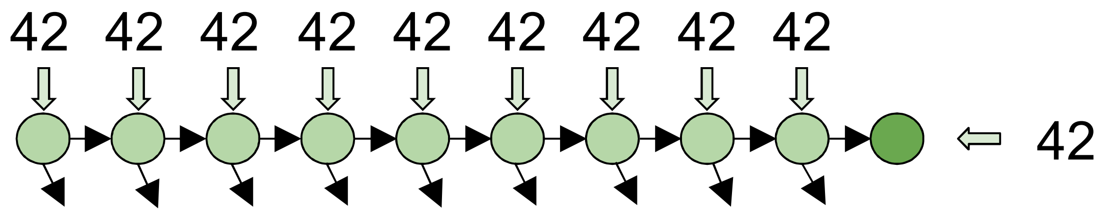
Policy gradients: High variance



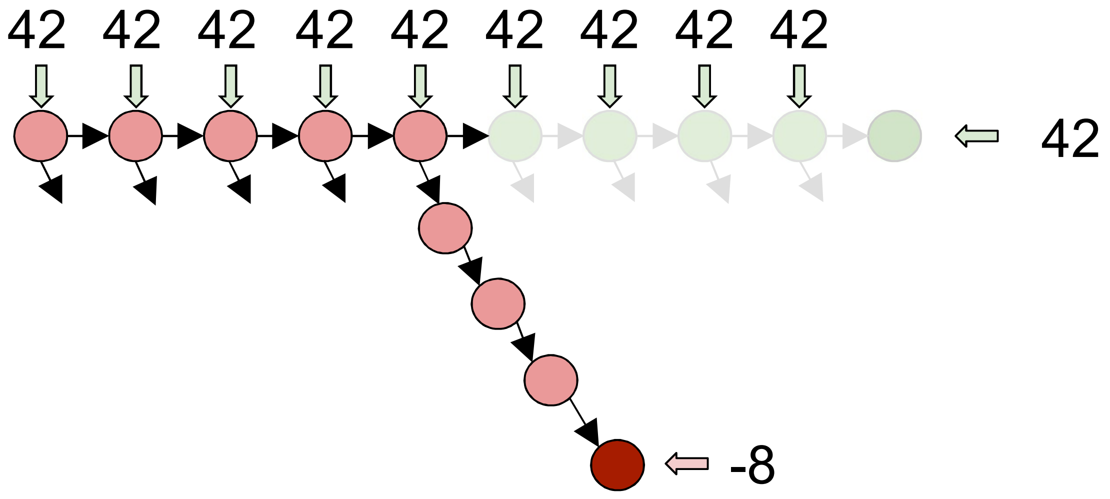
Variance - all choices get the reward



Variance - other possible paths



Variance - high probability to chose some other path



Variance - same actions for same state: now negative