2-INF-150: Machine Learning – Neural Nets

Marek Šuppa

October 24, 2018

(preliminary) plan

- 3. 10. – Python / Numpy
- 17. 10. – Regression / Learning Theory
- **24. 10. Neural Networks**
- 14. 11. – Support Vector Machines / Decision Trees / Forests
- 5. 12. – PCA / Clustering

Today

- (Almost) no coding necessary
- (Artificial) Neural Networks
- Convolutional Neural Networks
- Special Bonus

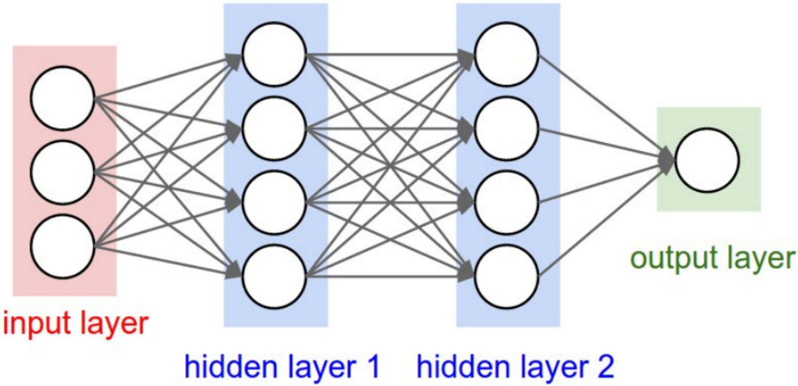
Setup

- Start Linux
- Open command line
- `pip3 install sklearn`
- `pip3 install tensorflow`
- `pip3 install keras`

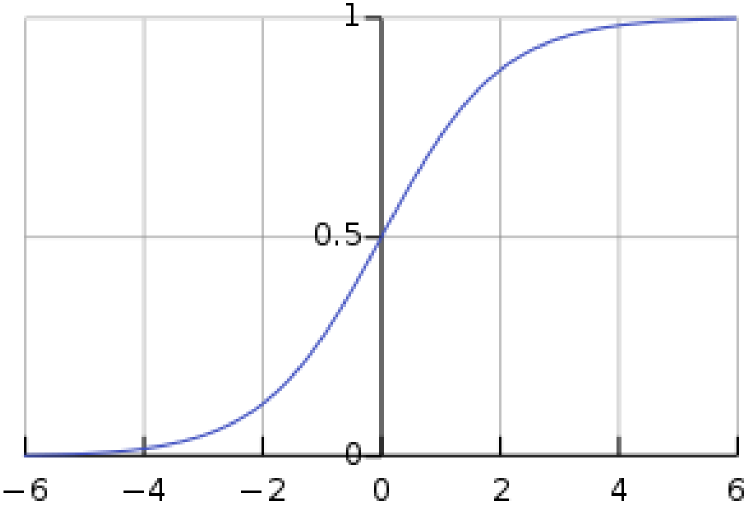
Setup II.

- `mkdir ml_exercise3`
- `cd ml_exercise3`
- `git clone https://github.com/NaiveNeuron/ml_exercises.git`
- or download directly https://github.com/NaiveNeuron/ml_exercises/archive/master.zip
- cd assignment2
- **Run jupyter-notebook (Please, do not run on shared disk)**

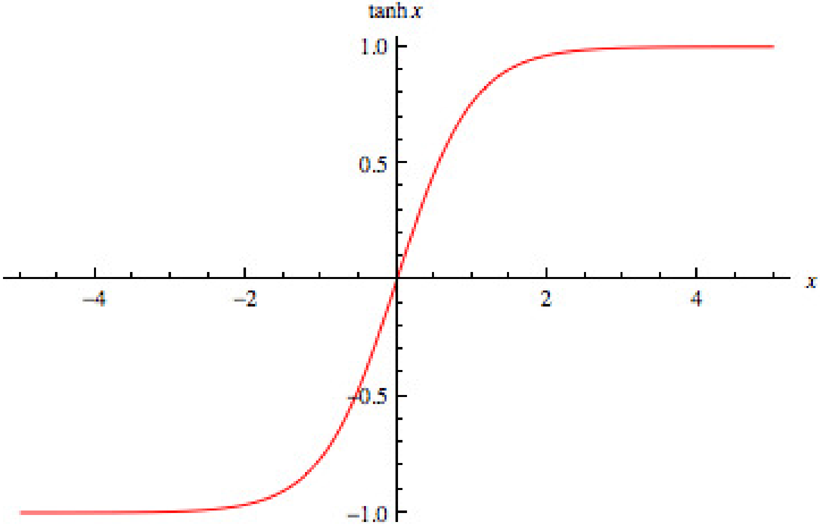
MLP



sigmoid

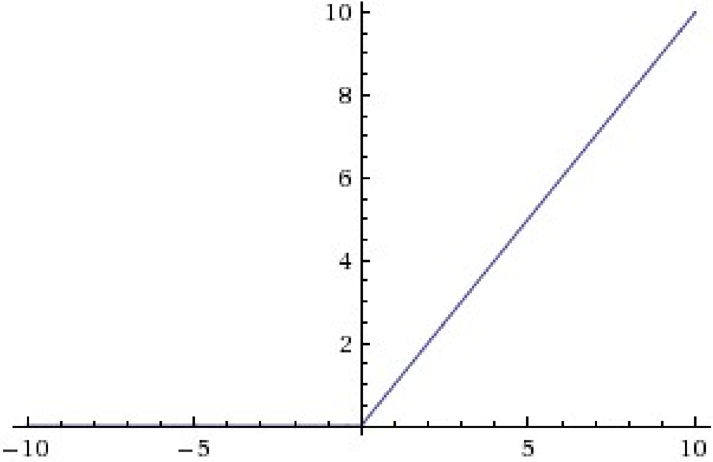


tanh



ReLU

Rectified Linear Unit



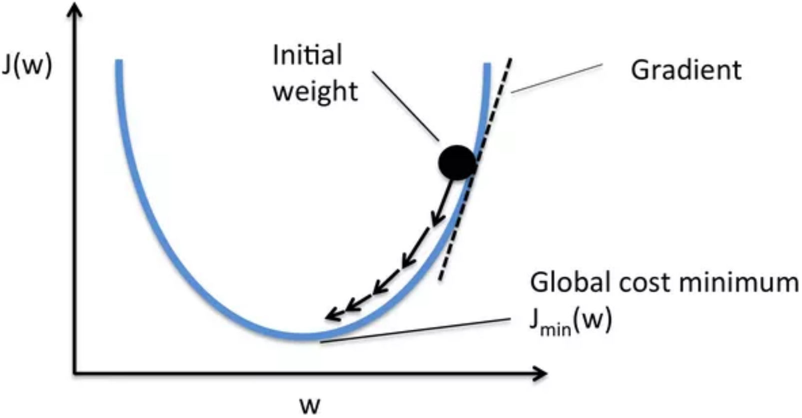
Gradient Descent Variants

Core idea of gradient descent

Minimize $J(\theta)$ parametrized by $\theta \in \mathbb{R}^d$ by updating θ in the opposite direction of the gradient $\nabla_\theta J(\theta)$.

Gradient Descent Variants

Core idea of gradient descent



Batch Gradient Descent

aka Vanilla Gradient Descent

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_{\theta_t} J(\theta_t)$$

- Might be very slow
- No-go for big datasets
- Impossible to update "online" (new examples on-the-fly)
- Guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

Stochastic Gradient Descent

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_{\theta_t} J(\theta_t; x^{(i)}; y^{(i)})$$

- Usually faster convergence
- Where batch gradient descent does redundant computation, SGD updates frequently and creates fluctuations.
- When slowly decreasing the learning rate, SGD shows the same convergence behaviour as batch gradient descent

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

Mini-batch Gradient Descent

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

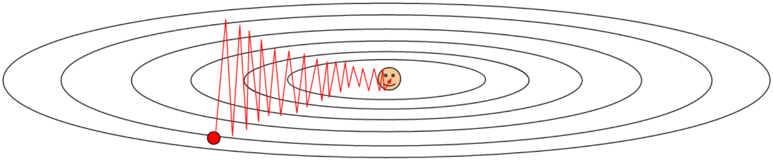
- Best of both worlds
- Reduced variance of parameter updates – more stable convergence
- SGD and Mini-batch Gradient Descent are used interchangeably

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Gradient Descent – Challenges

- Choosing a proper learning rate is difficult (too small, too large, too steady...)
- Learning rate schedules help, but still need to be pre-defined in advance
- Same learning rate for all parameter updates (larger updates to more infrequent features might be more desirable)
- Ending up trapped in suboptimal local optima

Stochastic Gradient Descent



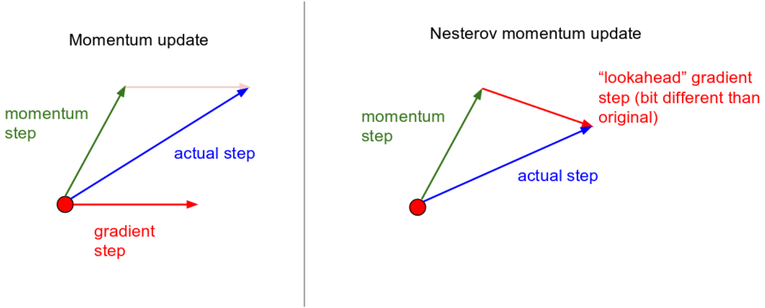
Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta_{t+1} = \theta_t - v_t$$

- Helps navigate SGD when one dimension curves more steeply than than the other (common around local optima)
- Basically fights against oscillations
- Momentum term γ is usually set to 0.9
- "Pushing a ball down a hill" metaphor

Nesterov Momentum



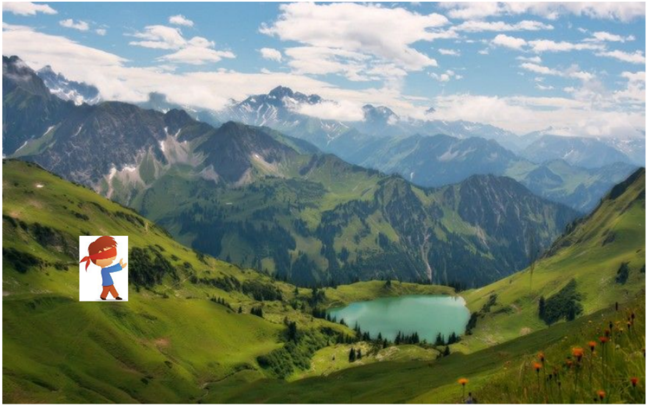
Nesterov Accelerated Gradient

Let's not blindly trust gravity

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta_{t+1} = \theta_t - v_t$$

- Give the moving ball some notion of where it is going
- $\theta - \gamma v_{t-1}$ approximates (gives a rough idea of) the next position of the parameters
- "Update with anticipation" prevents the ball from going too fast
- Is able to adapt updates to the slope – we'd like to also adapt updates to "parameter importance"

Parameter space



AdaGrad

$$g_{t,i} = \nabla_\theta J(\theta_i)$$
$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

- $G_t \in \mathbb{R}^{d \times d}$ – diagonal matrix where $G_{t,ii}$ is the sum of the squares of the gradients w.r.t θ_i up to time t.
- ϵ helps to avoid division-by-zero issues (usually on the order of $1e-8$)
- Main benefit: no need for manually decaying/tuning the learning rate
- Main weakness: accumulation of squared gradients in the denominator
- Learning rate will shrink (sometimes way too much)

RMSProp

RMSProp update

[Tieleman and Hinton, 2012]

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

RMSProp

rmsprop: A mini-batch version of rprop

- rprop is equivalent to using the gradient but also dividing by the size of the gradient.
  - The problem with mini-batch rprop is that we divide by a different number for each mini-batch. So why not force the number we divide by to be very similar for adjacent mini-batches?
- rmsprop: Keep a moving average of the squared gradient for each weight

$$MeanSquare(w,t) = 0.9\ MeanSquare(w,\ t{-}1) + 0.1\left(\partial E/_{\partial w}(t)\right)^2$$

- Dividing the gradient by $\sqrt{MeanSquare(w,\ t)}$ makes the learning work much better (Tijmen Tieleman, unpublished).

Introduced in a slide in Geoff Hinton's Coursera class, lecture 6

Cited by several papers as:

[52] T. Tieleman and G. E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude., 2012.

Adam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Both m_t and v_t are initialized as 0s, so they need to be bias-corrected.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Adam

Adam update

[Kingma and Ba, 2014]

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
  dx = # ... evaluate gradient
  m = beta1*m + (1-beta1)*dx # update first moment
  v = beta2*v + (1-beta2)*(dx**2) # update second moment
  mb = m/(1-beta1**t) # correct bias
  vb = v/(1-beta2**t) # correct bias
  x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

momentum

bias correction
(only relevant in first few iterations when t is small)

RMSProp-like

The bias correction compensates for the fact that m,v are initialized at zero and need some time to “warm up”.

Visual Demo

So what should one use?

- RMSProp and Adam are very similar
- Bias-correction in Adam has been shown to outpreform RMSProp slightly towards the end
- **Adam** is usually a good default choice for CNNs, RMSProp might be worth considering for big RNNs

ConvNets

ConvNets

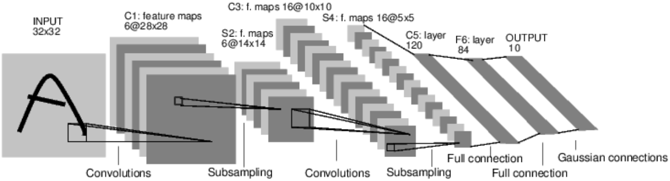
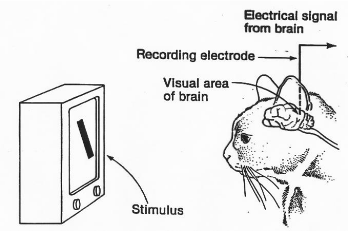
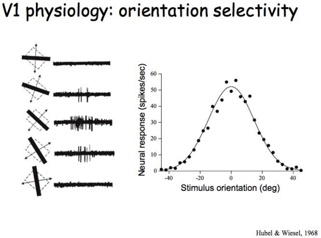


Figure: LeNet [LeCun et al., 1998]

Hubel & Wiesel

1959 - Receptive fields of single neurones in the cat's striate cortex
1962 - Receptive fields, binocular interaction and functional architecture in the cat's visual cortex





AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

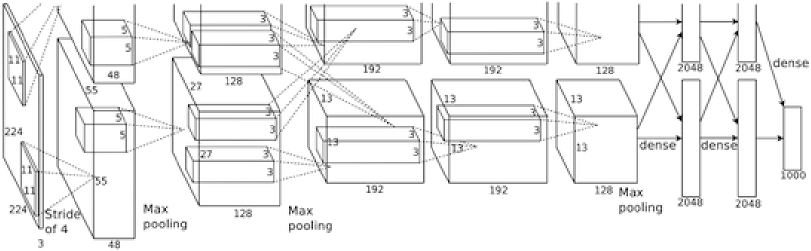


Figure: AlexNet [Krizhevsky, Sutskever, Hinton, 2012]

ConvNets today

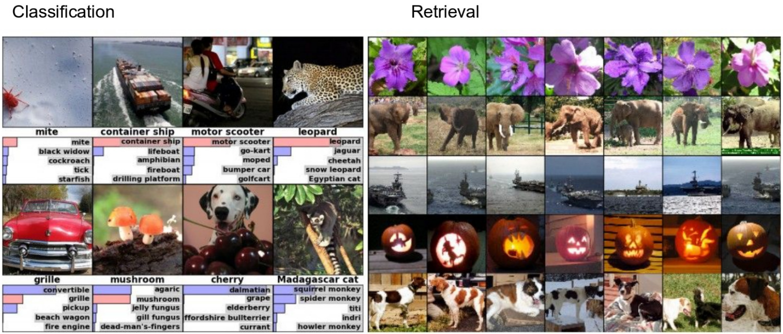


Figure: [Krizhevsky 2012]

ConvNets today

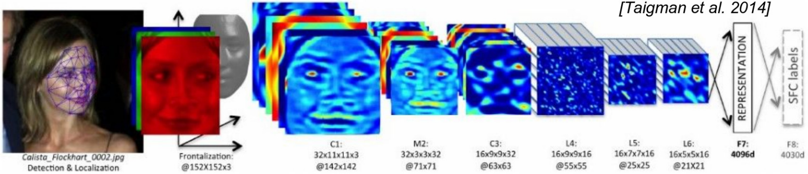
Figure: [Faster R-CNN: Ren, He, Girshick, Sun 2015] Detection Segmentation & [Farabet et al., 2012]

ConvNets today

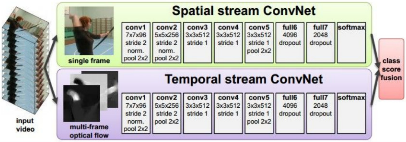
NVIDIA Tegra X1

Figure: Self driving cars

ConvNets today



[Taigman et al. 2014]



[Simonyan et al. 2014]

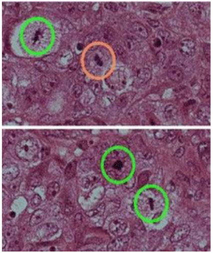
[Goodfellow 2014]

ConvNets today

[Toshev, Szegedy 2014]

[Mnih 2013]

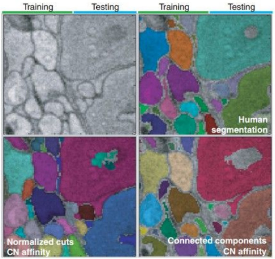
ConvNets today



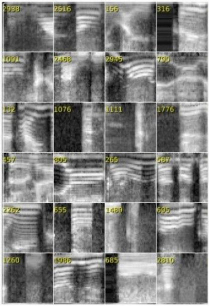
[Ciresan et al. 2013]

[Sermanet et al. 2011]
[Ciresan et al.]

ConvNets today



[Turaga et al., 2010]



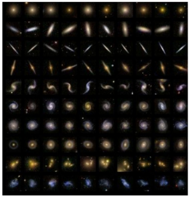
I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them.** Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horror/slasher buff. **Definitely worth checking out.**

I just saw this on a local independent station in the New York City area. **The cast showed promise but when I saw the director, George Cosmotos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmotos movie I ever saw.** He's like a stupid man's Michael Bey – with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

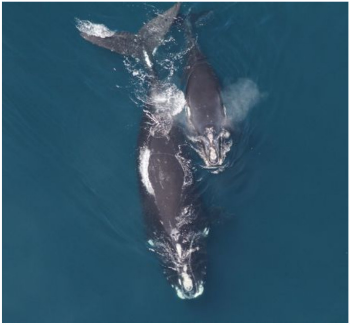
Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, thats doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it . What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. **It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.

[Denil et al. 2014]



ConvNets today



Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010

ConvNets today

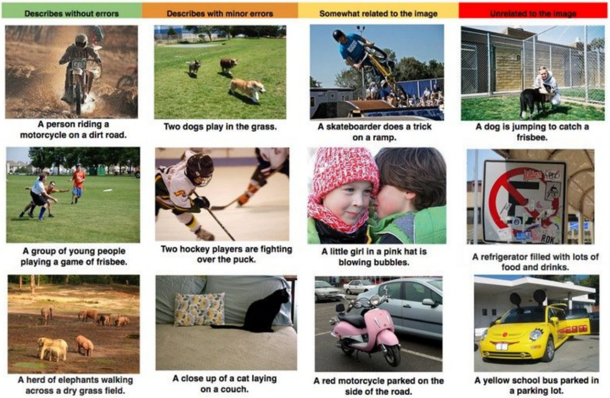


Image Captioning

[Vinyals et al., 2015]

ConvNets today

reddit.com/r/deepdream

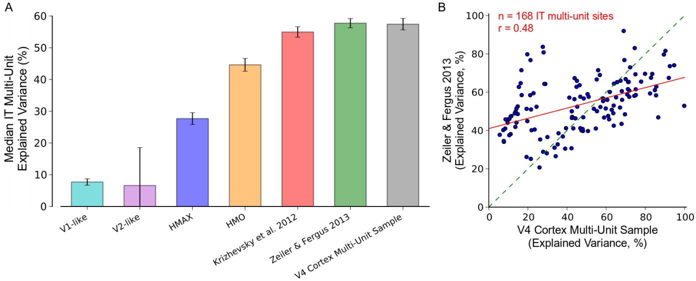
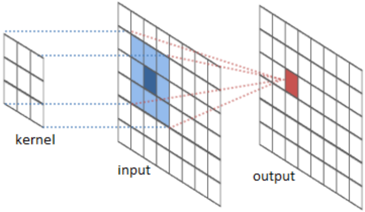
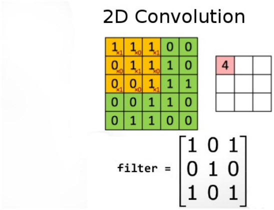


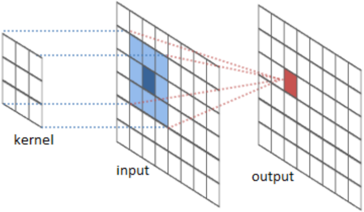
Figure: Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition [Cadieu et al., 2014]

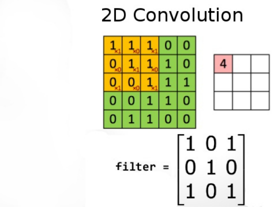
Convolution





Convolution



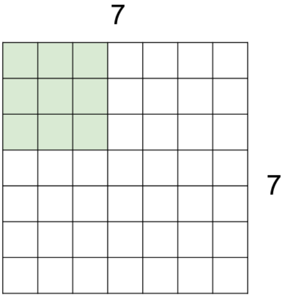


Convolution

We don't have to go with stride 1

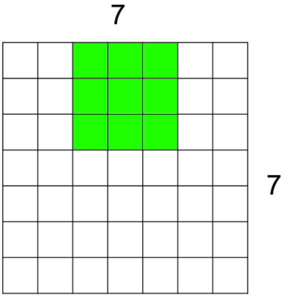
Convolution

We don't have to go with stride 1



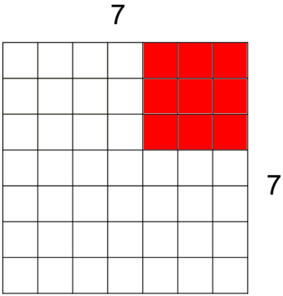
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution



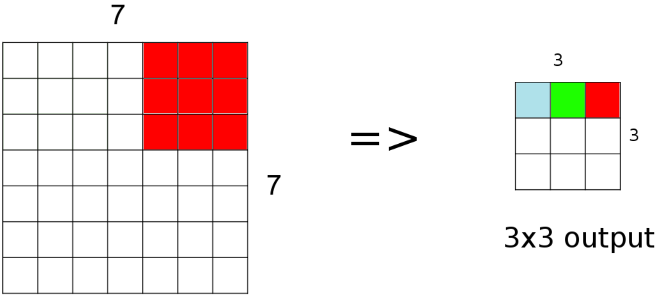
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution

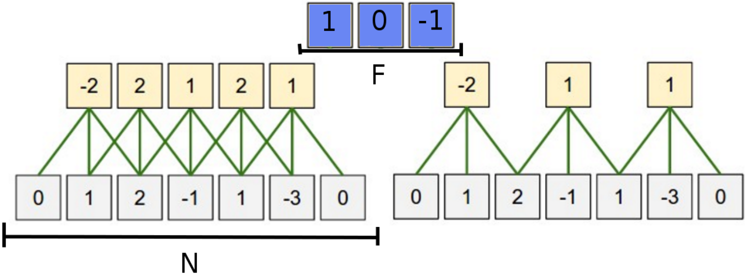


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution



Convolution



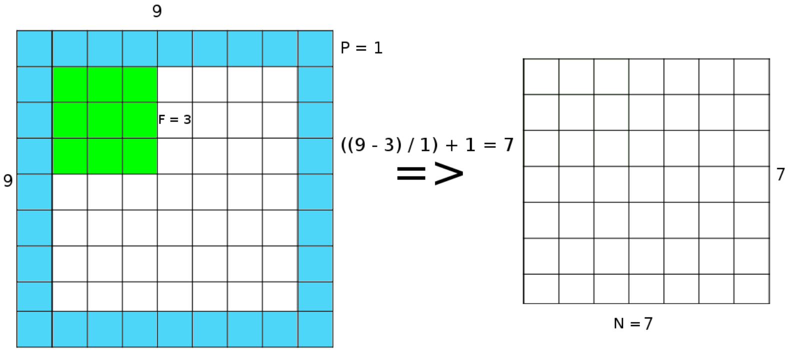
Output size: $\frac{N-F}{stride}+1$

Convolution

What if I want to keep spatial dimension?

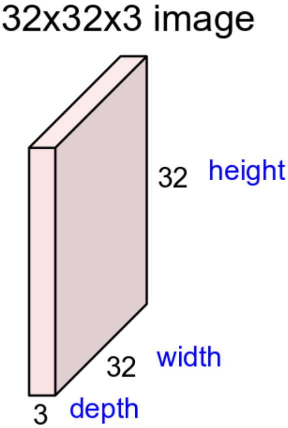
Convolution

What if I want to keep spatial dimension? Pad the input!

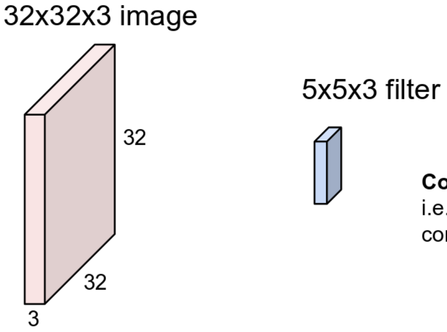


Output size: $\frac{N-F+2P}{stride} + 1$

Convolution layer

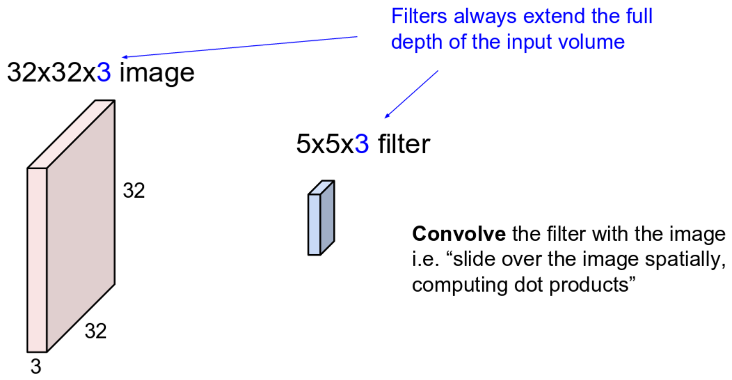


Convolution layer

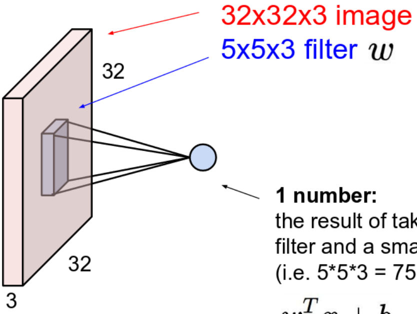


Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Convolution layer



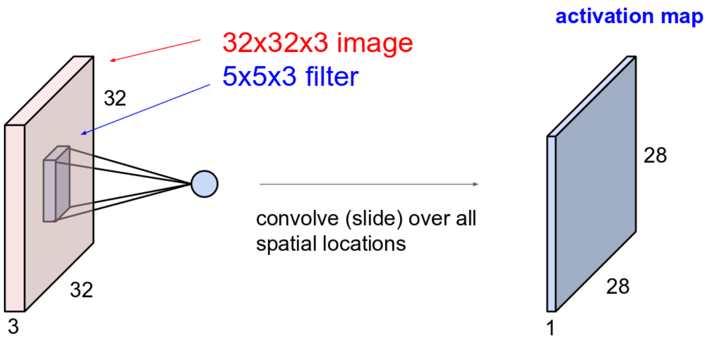
Convolution layer



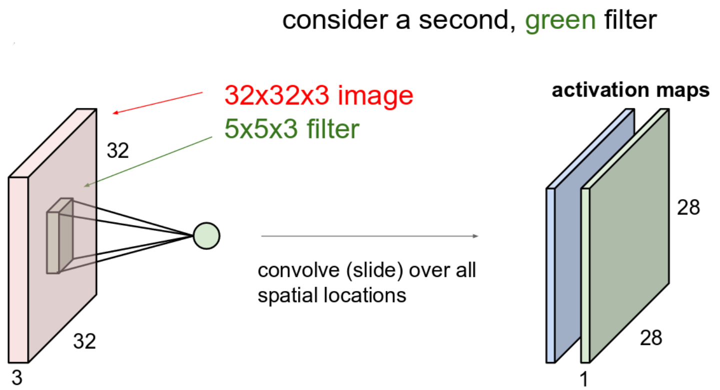
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution layer

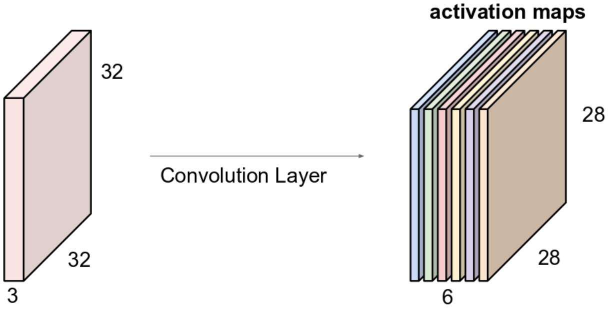


Convolution layer



Convolution layer

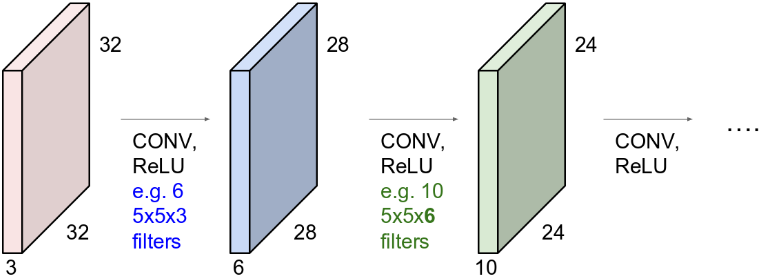
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolution layer

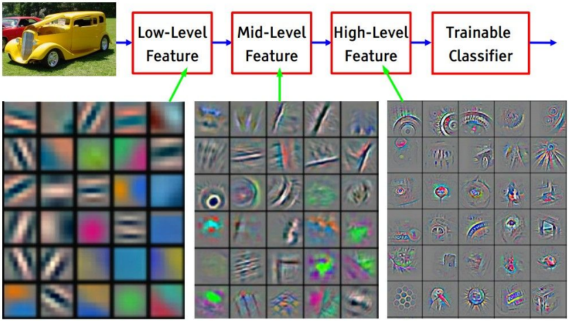
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Convolution layer

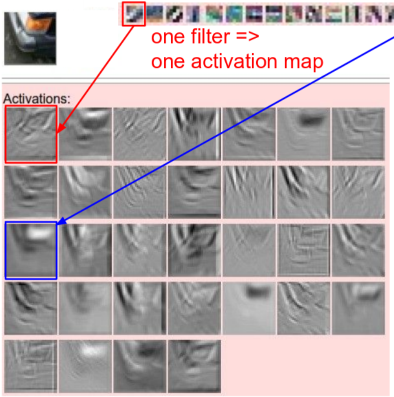
Preview

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolution layer



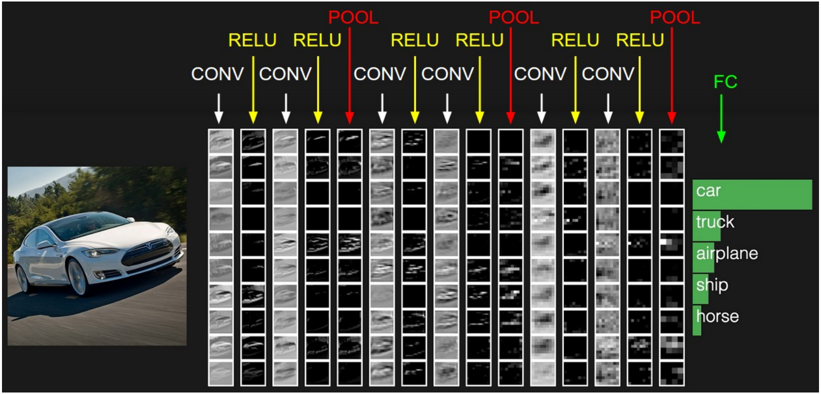
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

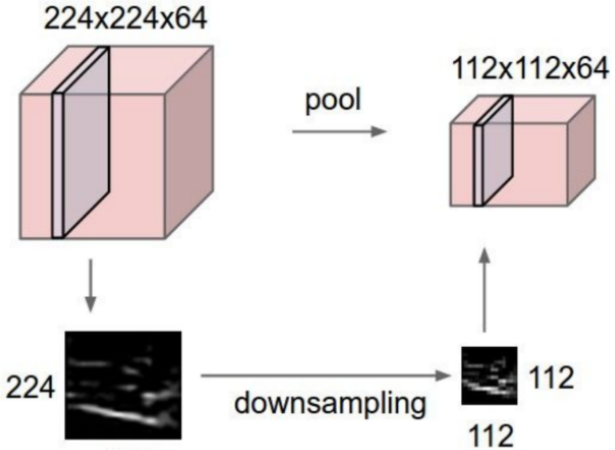
elementwise multiplication and sum of a filter and the signal (image)

Convolutional Neural Network

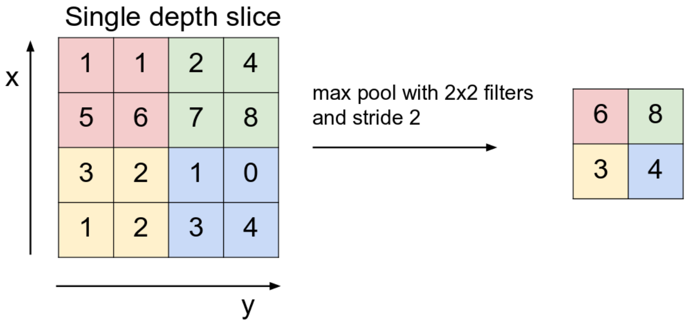


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

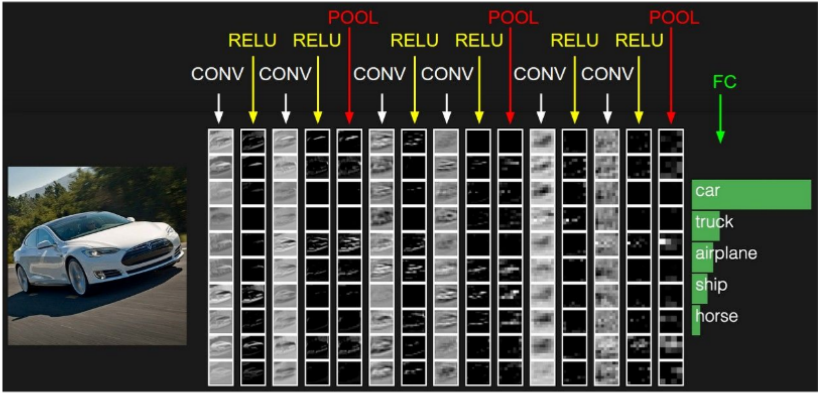


Max pool


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Fully Connected layers

Contains neurons that connect to the entire input volume, as in ordinary NN



Quite a lot for one short presentation

Do not worry, we realize that as well.

Quite a lot for one short presentation

Do not worry, we realize that as well.

But it's so cool we could not resist...

Data


T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot