Introduction to Deep Learning

- Basic Methods & Techniques
- Deep Learning Frameworks
- Physics Examples and further Applications

Martin Erdmann, Peter Fackeldey

(slide credits: Jonas Glombitza)

RWTH Aachen

Time Schedule

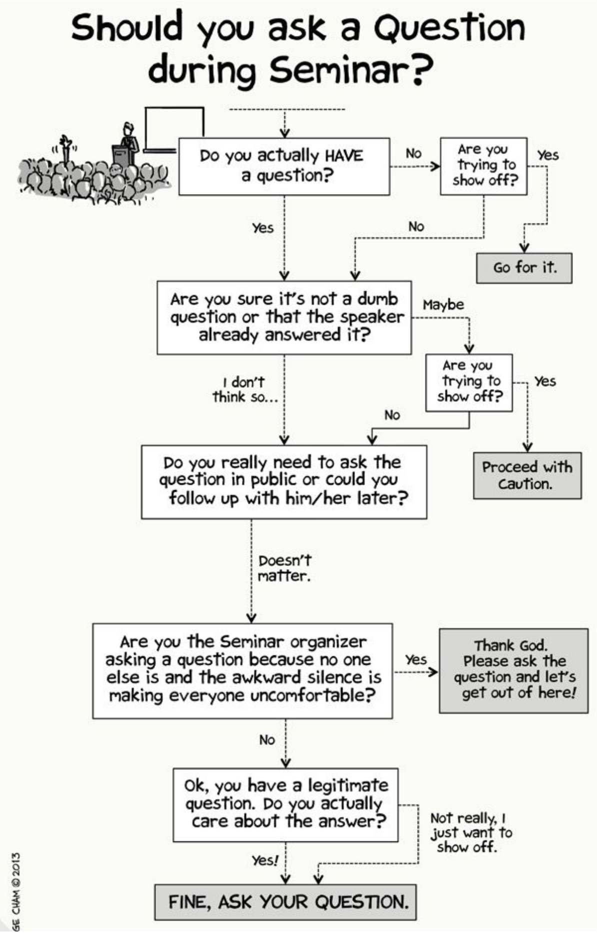
Deep Learning Basics, code examples + hands on session (VISPA GPU cluster)

Machine Learning and Neural Networks

- Training, Generalization and Regularization
- *Practice 1: CIFAR-10 Classification*

Convolutional Networks

- Pooling, Padding, Striding + basic architecture
- *Practice 2: CIFAR-10 Classification*

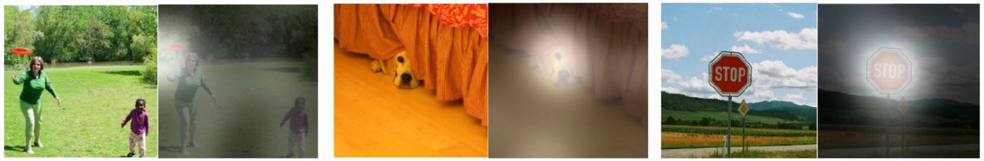

Should you ask a Question during Seminar?
Do you actually HAVE a question?
No
Are you trying to show off?
Yes
Go for it.
No
Yes
Are you sure it's not a dumb question or that the speaker already answered it?
Maybe
Are you trying to show off?
Yes
Proceed with Caution.
No
I don't think so...
Do you really need to ask the question in public or could you follow up with him/her later?
Doesn't matter.
Are you the Seminar organizer asking a question because no one else is and the awkward silence is making everyone uncomfortable?
Yes
Thank God. Please ask the question and let's get out of here!
No
Ok, you have a legitimate question. Do you actually care about the answer?
Not really, I just want to show off.
Yes!
FINE, ASK YOUR QUESTION.
GE CHAM ©2013

This is a tutorial → Please ask questions!

Deep Learning

- Machine Learning Basics
- Neural Networks
  - Backpropagation, Optimization
  - Activation, Initialization
  - Preprocessing

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

ArXiv: 1502:03044

KÜNSTLICHE INTELLIGENZ

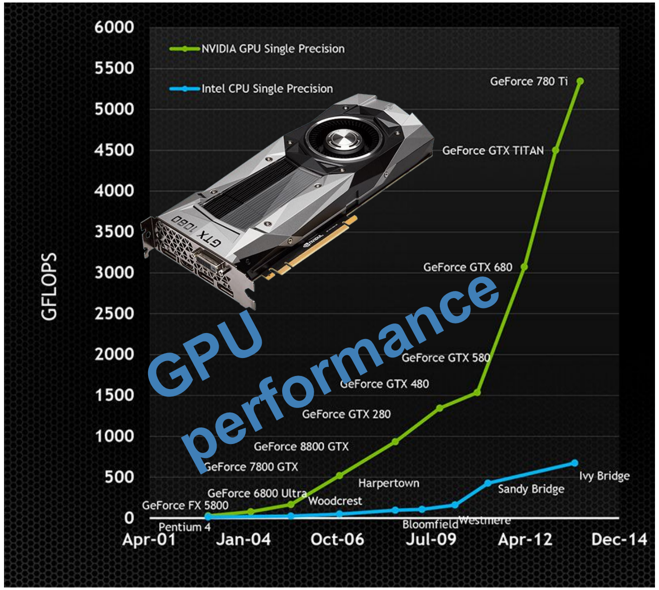
Schlau in zwei Stunden

VON ALEXANDER ARMBRUSTER - AKTUALISIERT AM 27.09.2017 - 11:41

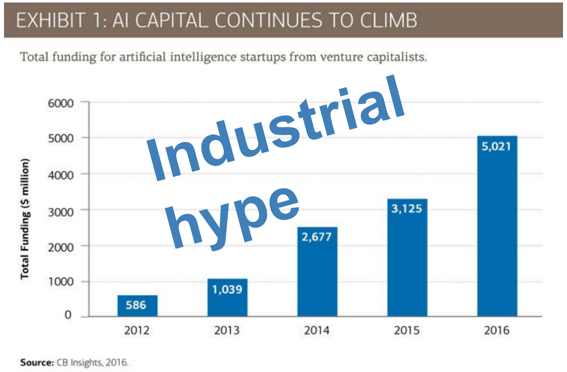
Artificial Intelligence - "The effort to automate intellectual tasks normally performed by humans"

Deep Learning

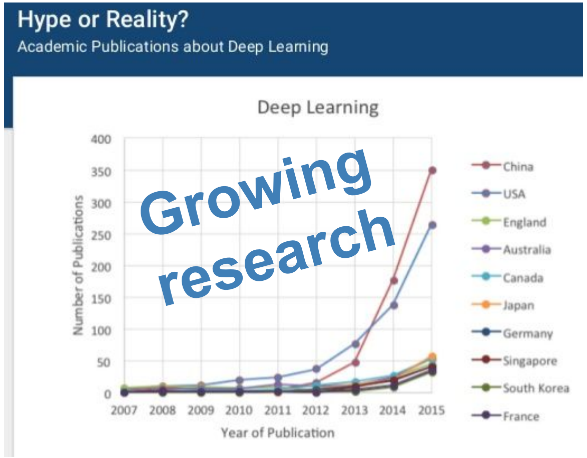
- Every minute:
  - Instagram users post 200,000 photos
  - Twitter users send 350,000 tweets
  - Data on billion scale every day



GPU performance



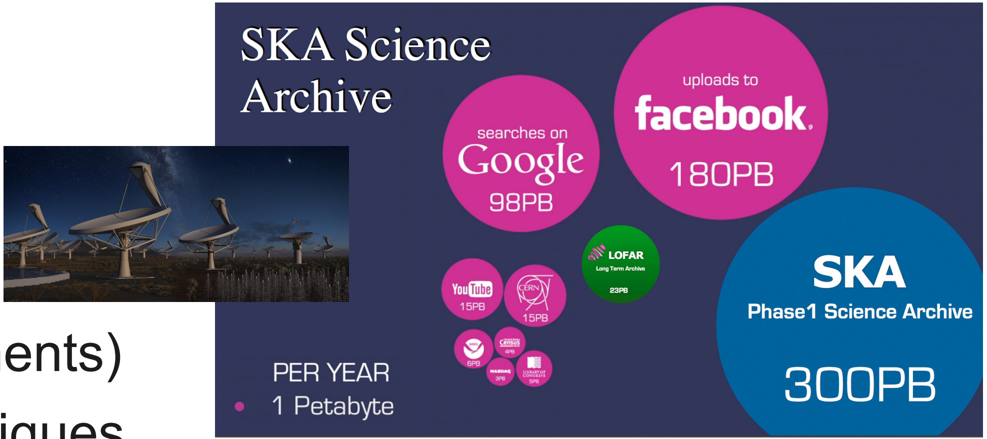
Industrial hype



Growing research

Deep Learning in Physics

Deep Learning is state-of-the-art machine learning approach for everything related to computer vision, speech and natural language processing and many artificial intelligence tasks in general.



What about physics?

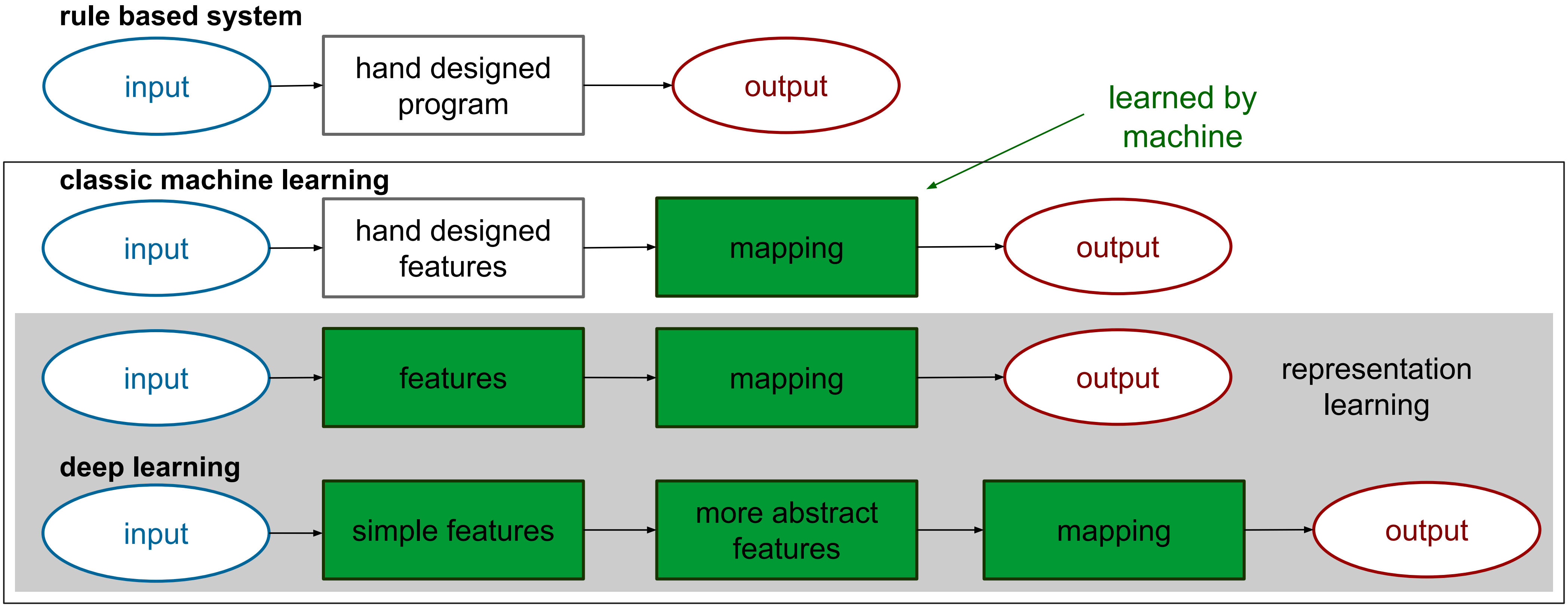
Ingredients:

- Complex problem (multivariate)
- Large amount of data (particle experiments)
- Stable software & computational techniques
- Sufficient computing resources

  ▯ **Now is a good time!**

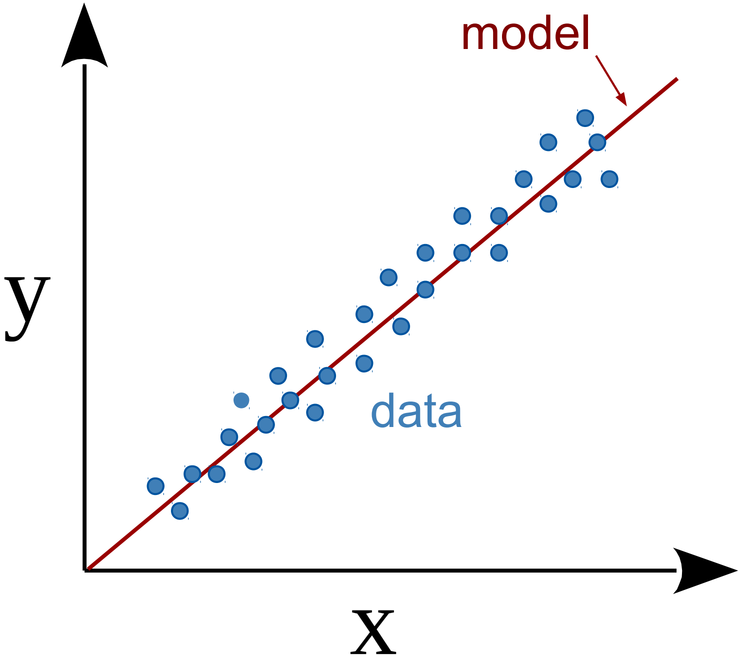
TensorFlow

When is it Deep?



"It's deep if it has more than one stage of non-linear feature transformation" - *Y. LeCun*

Machine Learning – Regression



- Data: $\{x_i, y_i\},\ i = 1, ..., N$

- Define model:
- $y_m(x, \theta) = Wx + b$ with free parameters $\theta = (W, b)$

- Define **objective function** (loss/cost)

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}[y_m(x_i, \theta) - y_i]^2$$

- Train model (minimize objective) $\hat{\theta} = argmin[J(\theta)]$
  - Optimize set of free parameters $\theta = (W, b)$
  - eg. use gradient descent

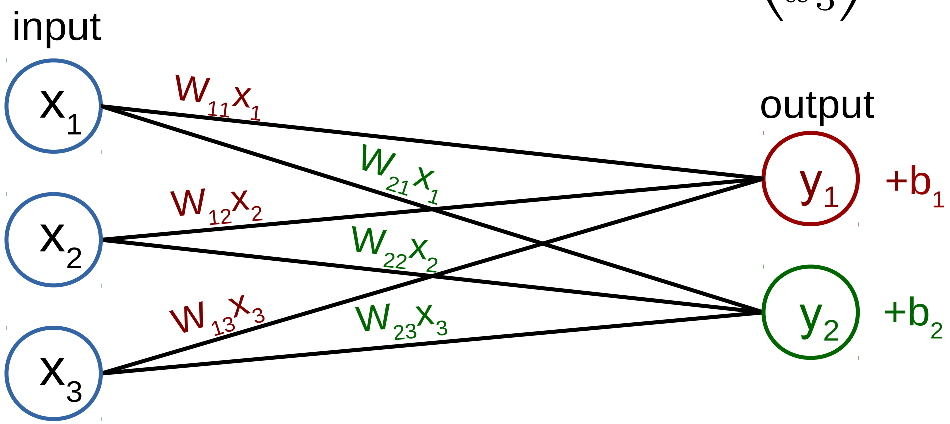
Multidimensional Linear Models

- Predict multiple outputs $\mathbf{y} = (y_1, ..., y_n)$ from multiple inputs $\mathbf{x} = (x_1, ..., x_n)$
- using linear function $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$

Note: We define linear = affine in this course

- Example: $x \in \mathbb{R}^3,\ y \in \mathbb{R}^2$

$$\begin{pmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$



Non-Linear Network Models

$\mathbf{W}\mathbf{x} + \mathbf{b}$ only describes linear models

- Use network with several linear layers:

$$h' = W^{(1)}x + b^{(1)}$$

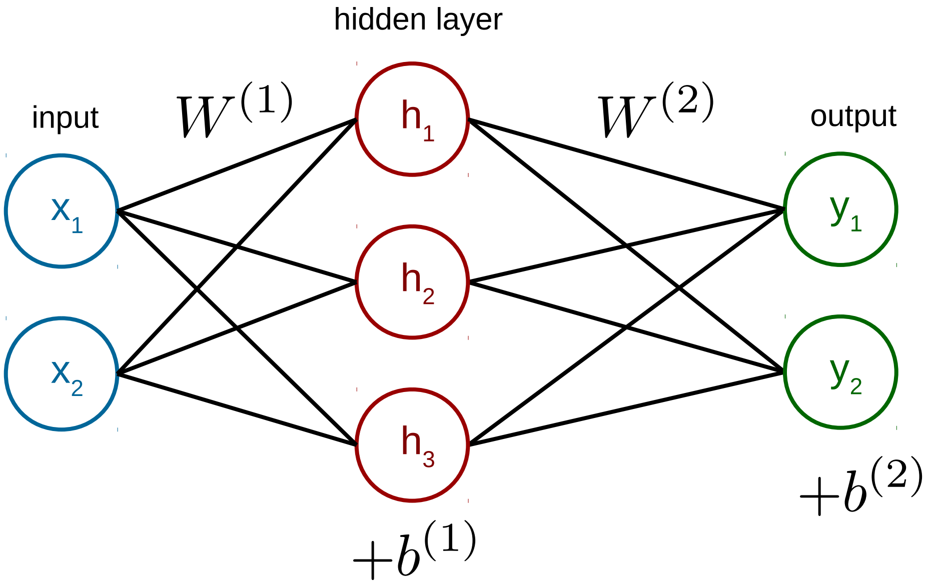
$$y = W^{(2)}h' + b^{(2)}$$

- Model is still linear!

$$y = W^{(2)}\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}$$

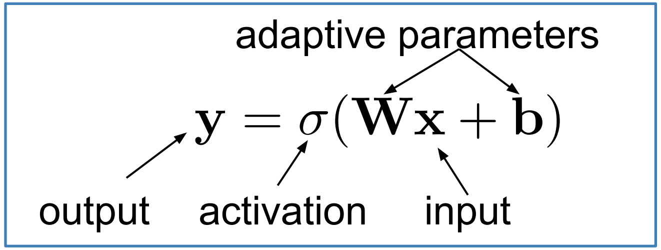
$$y = \underbrace{W^{(2)}W^{(1)}}_{W}x + \underbrace{W^{(2)}b^{(1)} + b^{(2)}}_{b}$$

- Solution: Apply non-linear activation σ to each element $\longrightarrow h = \sigma(h') = \sigma(Wx + b)$



Activation Functions

- Using an activation function the layer becomes a non linear mapping
  - Allows for stacking several layers



adaptive parameters

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

output activation input

Examples

- **Rectified Linear Unit**

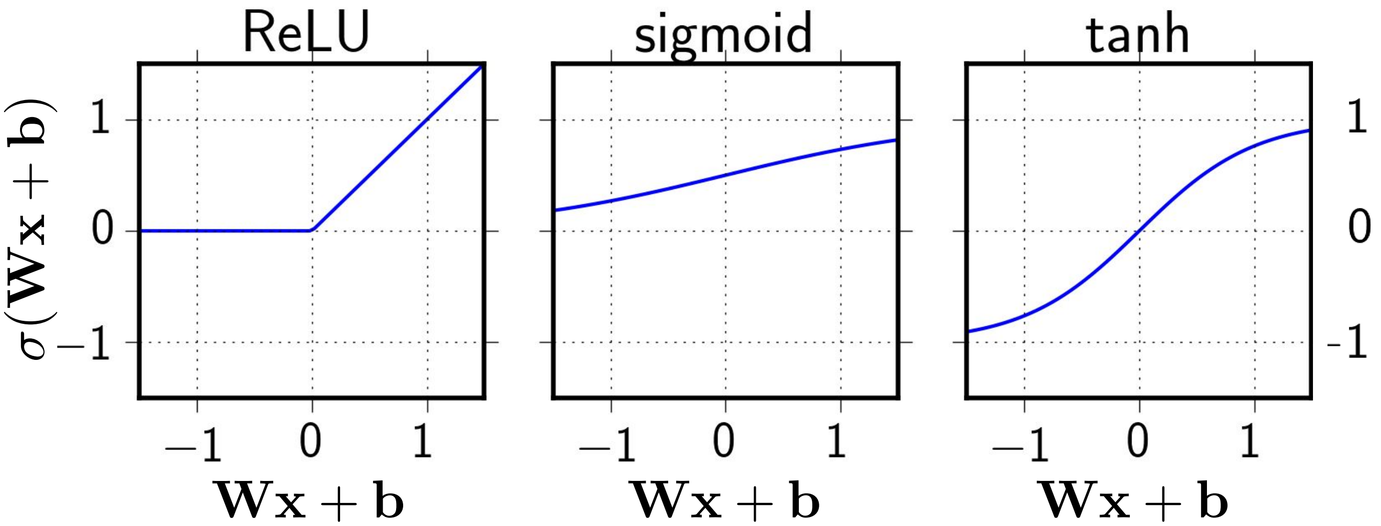
$$\sigma(x) = \max(0, x)$$

- **Sigmoid**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Hyperbolic tangent**

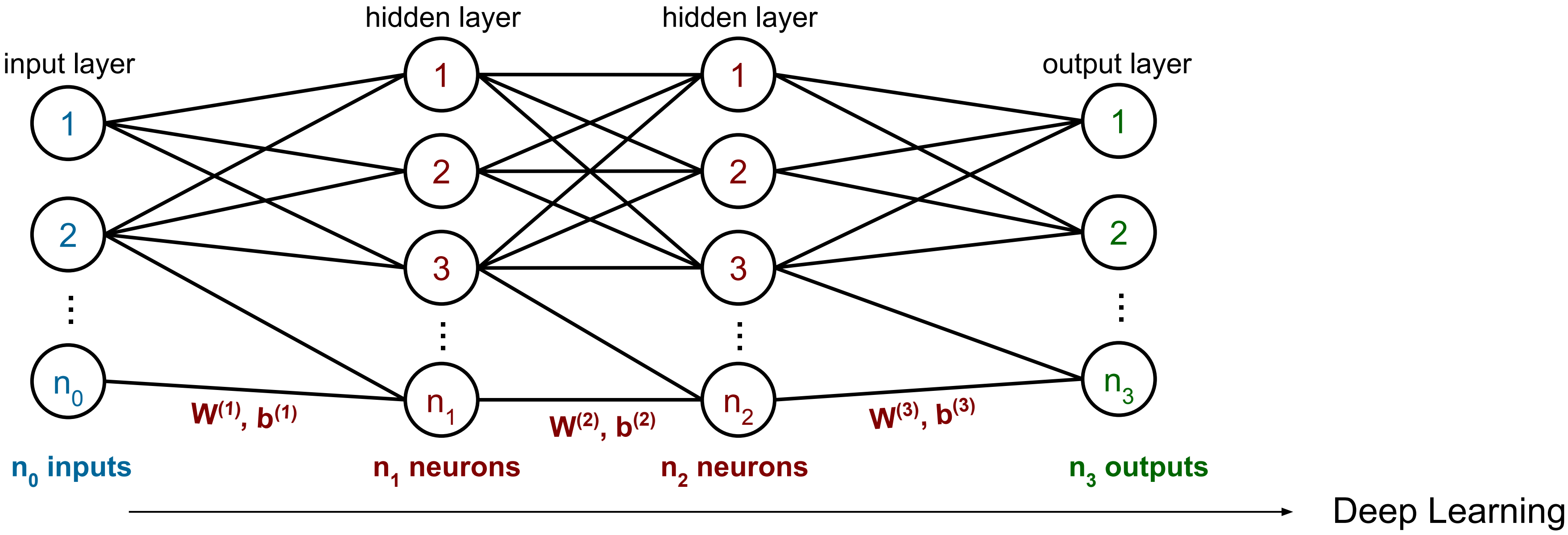
$$\sigma(x) = \frac{e^{+2x} - 1}{e^{-2x} + 1}$$



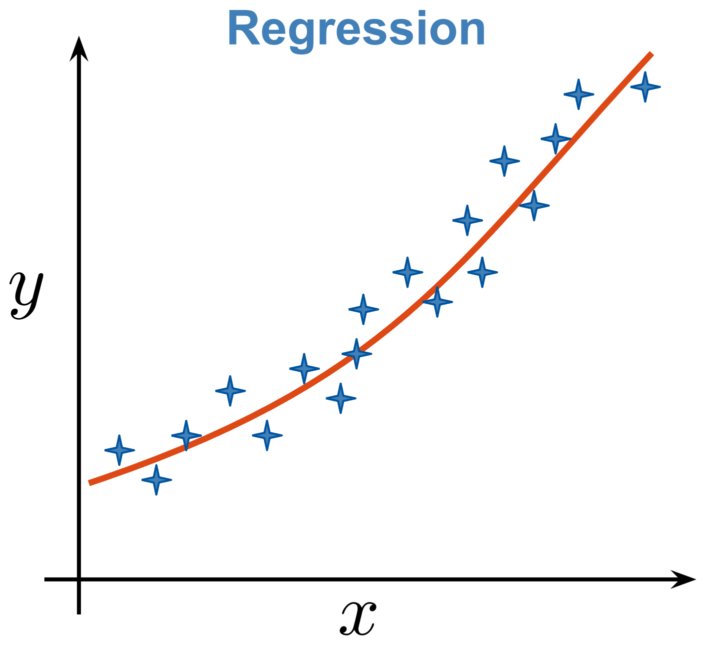
Neural Networks

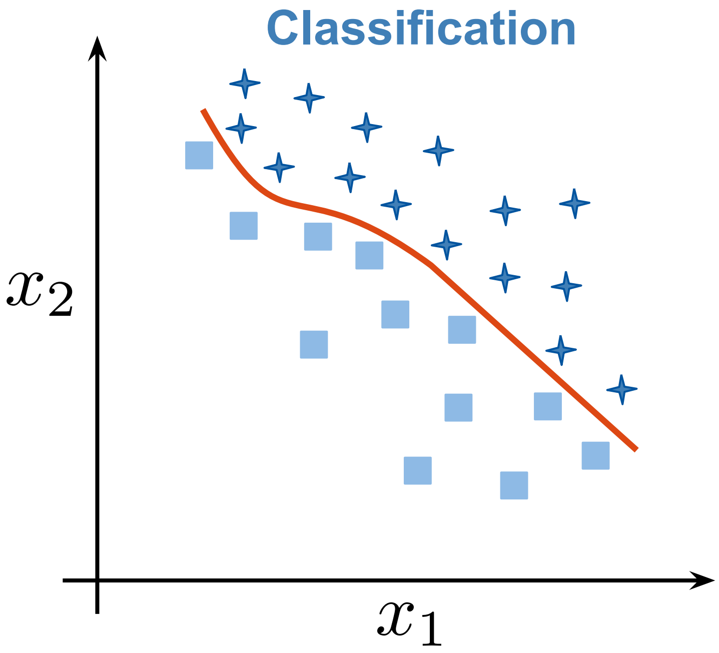
Basic unit $\sigma(Wx + b)$ is called **node/neuron** (analogy to neuroscience)

- Strength of connections between neurons is specified by **weight matrix** W
- **Width:** number of neurons per layer
- **Depth:** number of layers holding weights (do not count input layer)



Machine Learning Tasks

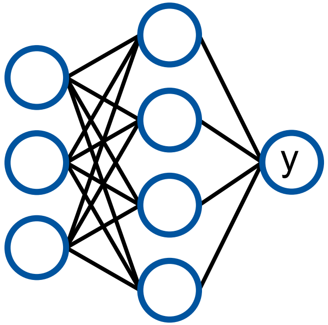




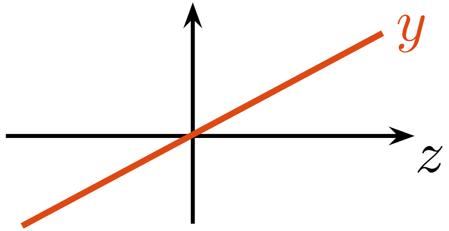
- Regression: Predict continuous label y
- Classification: Separate into different classes (cats, dogs, airplanes, …)
- Can sometimes convert to the other

Classification vs. Regression

Regression



Linear

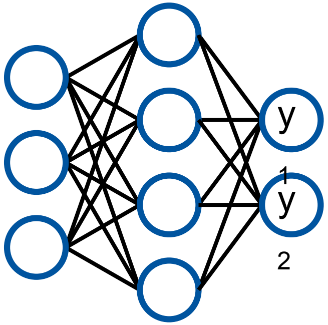


no activation function

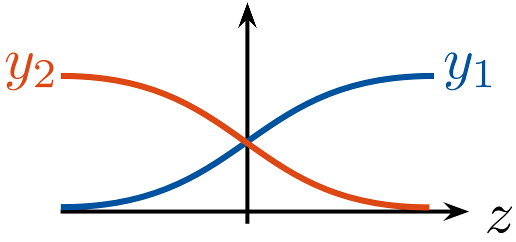
Minimize mean-squared-error

$$J(\theta) = \frac{1}{n}\sum_i [y_i - y_m(x_i)]^2$$

Classification



Softmax



$$y_j(z) = \frac{e^{z_j}}{\sum_i e^{z_i}}$$

Minimize cross entropy

$$J(\theta) = -\sum_i y_i \log[y_m(x_i)]$$

Gradient Descent

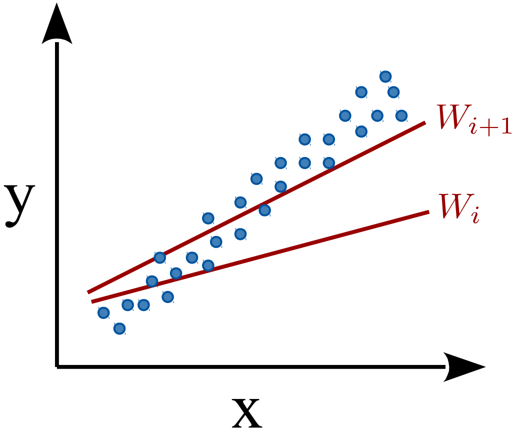
- Minimize objective function $J(\theta)$ by updating θ in **opposite** direction of gradient iteratively

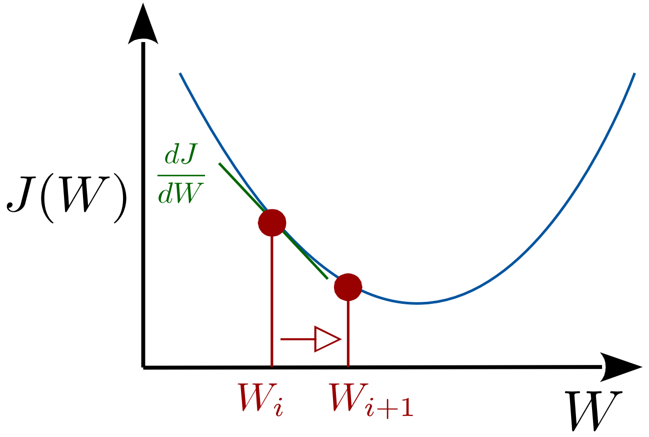
gradient: $dJ/d\theta$

stepsize: α

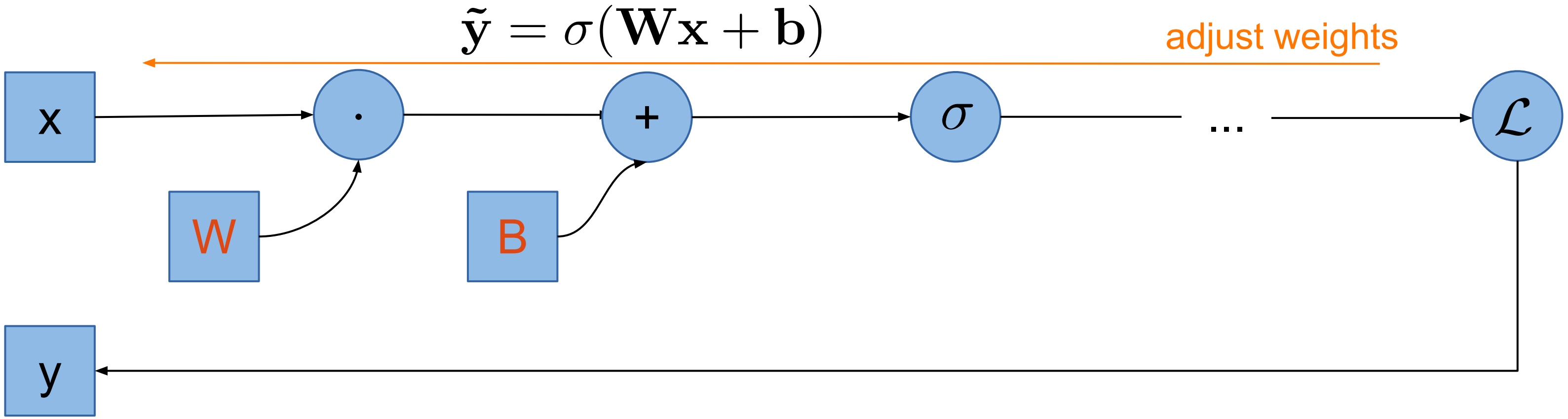
$$\tilde{\theta} \rightarrow \theta - \alpha \frac{dJ}{d\theta}$$

- Example: linear regression with mean squared error





Backpropagation



$\tilde{\mathbf{y}} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$
adjust weights
x
$\cdot$
+
σ
...
$\mathcal{L}$
W
B
y

- Network is series of simple operations (linear mappings/activations/loss ...)
- Each operations knows how to to calculate:
  - Its local output (forward pass)
  - Its derivative (backward pass)
- Use chain rule to evaluate gradient for each parameter
- Fast evaluation of the gradient → **Backpropagation**

Learning Rate

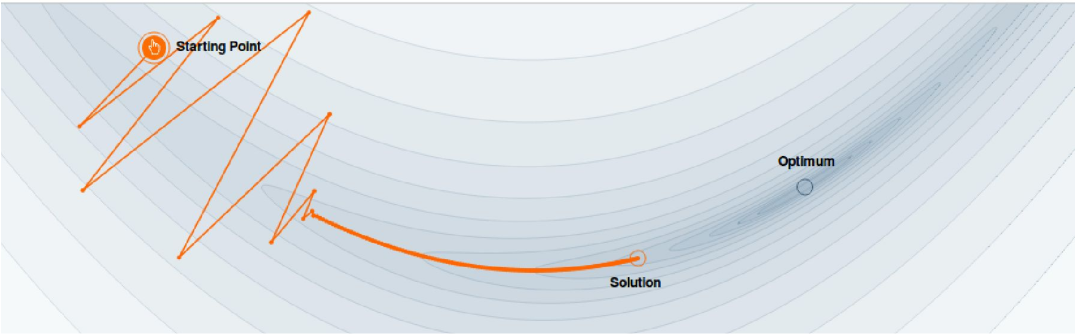
- Learning rate α determines speed of training
- High rate
  - poor convergence behavior or none at all
- Small rate
  - Very slow training or none at all
- Typical learning rate $\alpha = 10^{-3}$

- Advanced
- Reduce learning rate when loss stops decreasing
  - increase sensitivity to smaller scales

$$\boxed{\theta \rightarrow \theta - \alpha \frac{dJ}{d\theta}}$$

Learning rate

Stochastic Gradient Descent - SGD



Why Momentum Really Works, Distill

- Use small subset (mini batch) of dataset for calculating the gradient
  - 1 **epoch** = full pass through training data set
  - Reduces computational effort
  - More updates per epoch → speeds up convergence
  - Stochastic behavior → improve generalization performance
- **Batch size** is hyperparameter and mostly in order of ~32

“Friends don’t let friends use minibatches larger than 32” - Y. LeCun

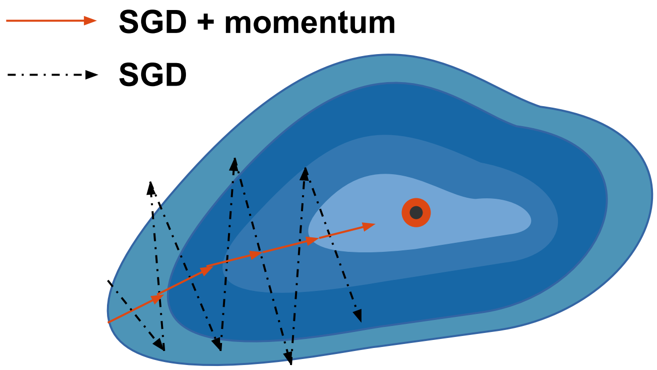
Advanced Optimizer

Momentum: Use past gradients (velocity)

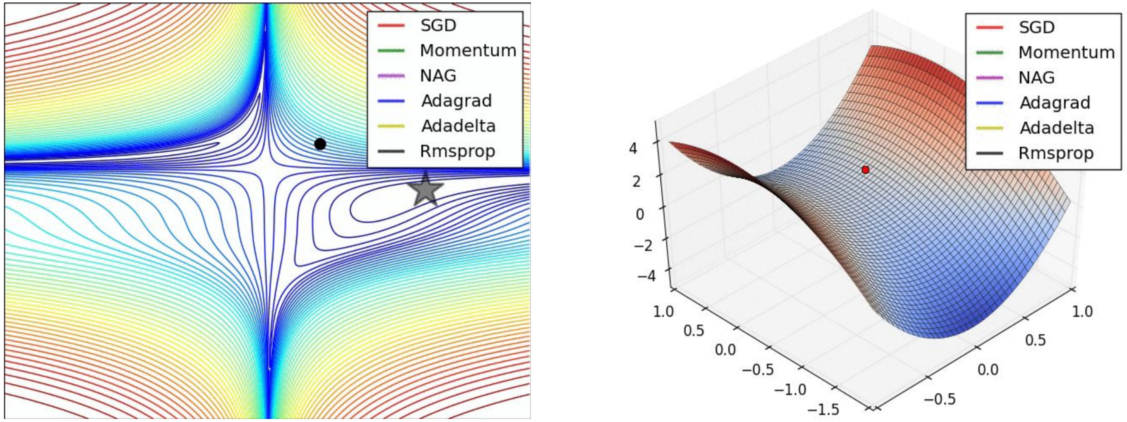
- Faster convergence by **damping oscillations** and increasing the step size for more informative gradients

Adaptive learning rate: Scaling using past gradients (Adagrad, **Adam**, Adadelta...)

- Use adaptive learning rates for each parameter



Convergence behavior of various optimizers

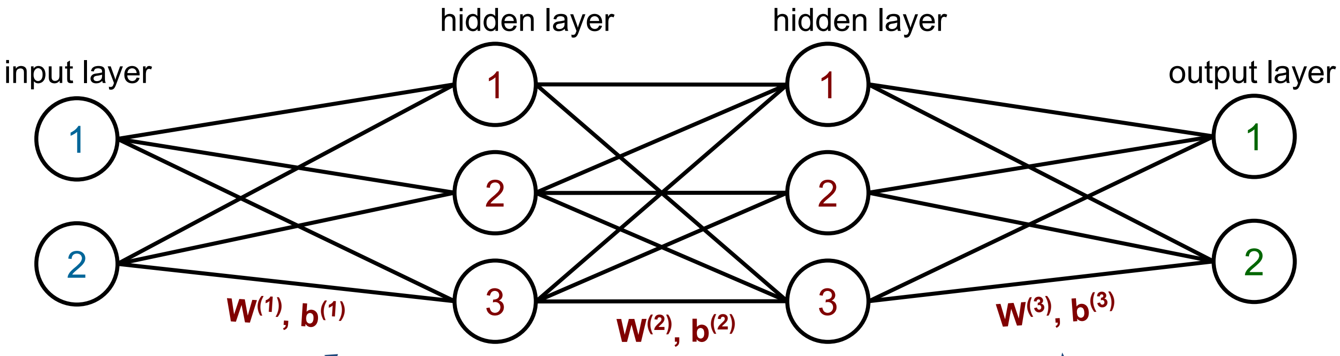


Sebastian Ruder: http://ruder.io/optimizing-gradient-descent/

Deep Neural Networks

Feature Hierarchy: each new layer extract more abstract information of the data.
Probabilistic Mapping: learns to combine the extracted features

Train model (to find $\theta = \{W_i, b_i\}$ that minimizes objective) is automatic process.



adaptive parameters

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

output activation input

$$\text{objective}: \quad J(\theta) = \sum_i \left[y_m(x_i, \theta) - y_i\right]^2$$

iterative update

$$\text{optimization}: \quad \frac{dJ}{d\theta} \to 0 \qquad \boxed{\tilde{\theta} \to \theta - \alpha \frac{dJ}{d\theta}}$$

Initialization

- Weights need different initial values → symmetry breaking
- Scale of weights very important
  - Too large → exploding signals & gradients
  - Too small → vanishing signals & gradients

No learning!

- For forward pass in each layer:

$$Var[x_l] = 1$$

- For backward pass in each layer:

$$Var[\Delta x_l] = 1$$

▯ Depends from activation function and number of in and outgoing nodes

$$Var[W] = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

→ For tanh

Glorot, Bengio

$$Var[W] = \frac{2}{n_{\text{in}}}$$

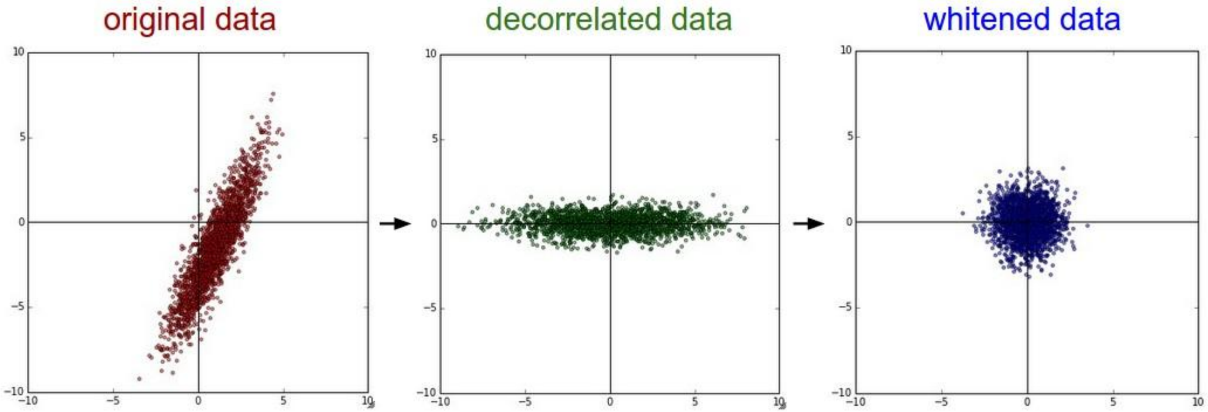
→ For ReLU

He et al.

- Can be sampled from Gaussian or uniform distribution (Var. scaled by factor of 3)

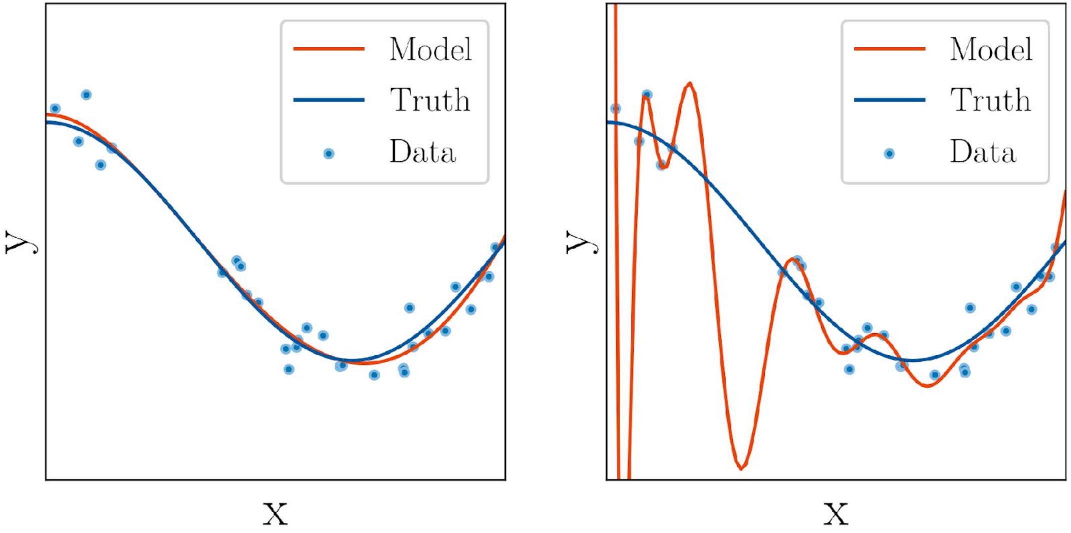
Data Preprocessing

- Input features of dataset should be on same scale
  - Prevent particular sensitivity to few features
- Common normalization strategies
  - Limit range between [0, 1] or [-1,1]
  - Standard normalization: $\mu(x_i) = 0 \,\&\, \sigma(x_i) = 1$
  - Whitening: standard normalization + decorrelation



Generalization

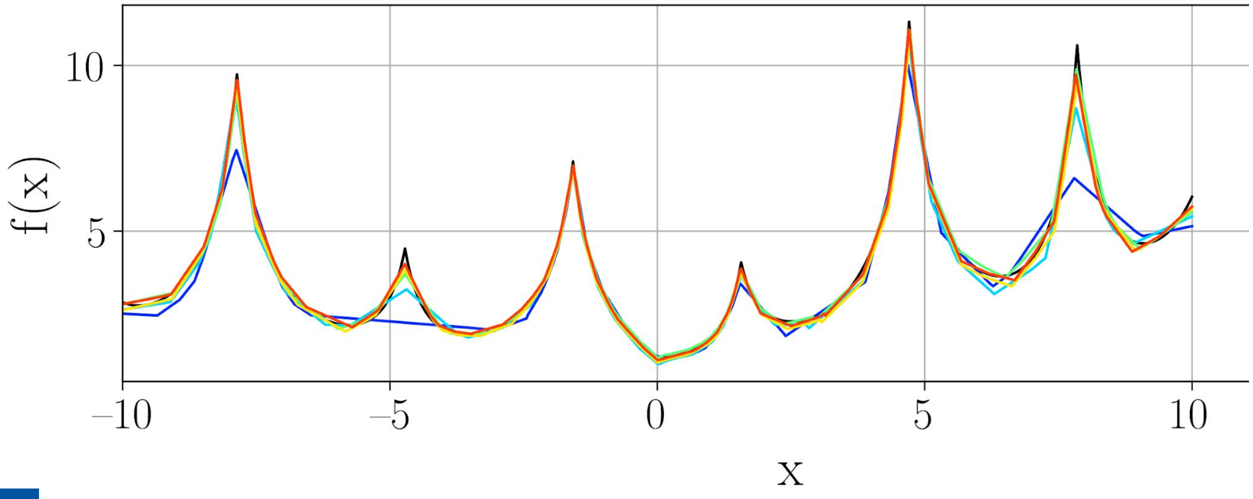
- Training, Validation, Test
- Under- and Overfitting
- Regularization



Universal Approximation Theorem

A feed-forward network with a linear output and at least ***one hidden layer*** *with a finite number of nodes can (in theory) approximate any reasonable function to arbitrary precision.*

- Network design considerations → feature engineering, network architecture
  - Shallow networks often show bad performance → train deep models!



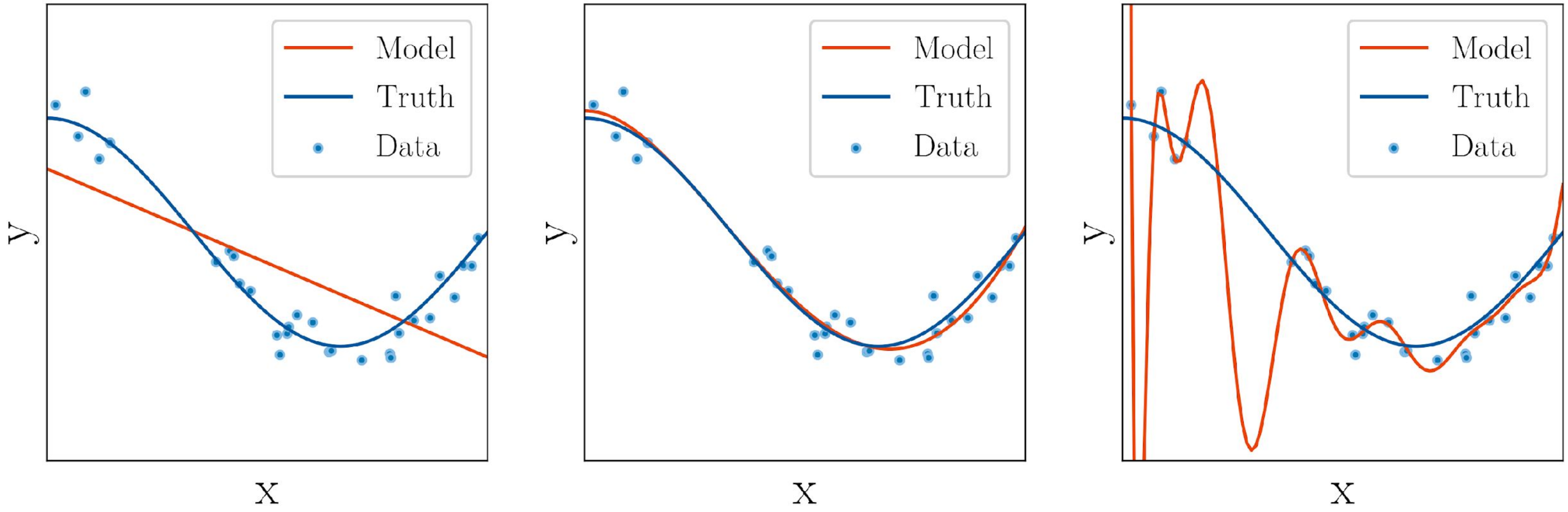
- Fit complicated function
- Use neural network
- 2 hidden layers a 30 nodes

Under- and Overfitting

- Challenging to find a good network design
- Under-complex models show bad performance
- complex models are prone to overfitting
  - Model memorizes training data under loss of generalization performance

underfitting

overfitting



Generalization & Validation

A complex network can learn any function, how can we monitor overfitting?

Generalization

Unknown true distribution $p_{true}(x, y)$ from which data is drawn.
Trained model $y_m(x)$ provides prediction based on this limited set

- How good is the model when faced with new data?

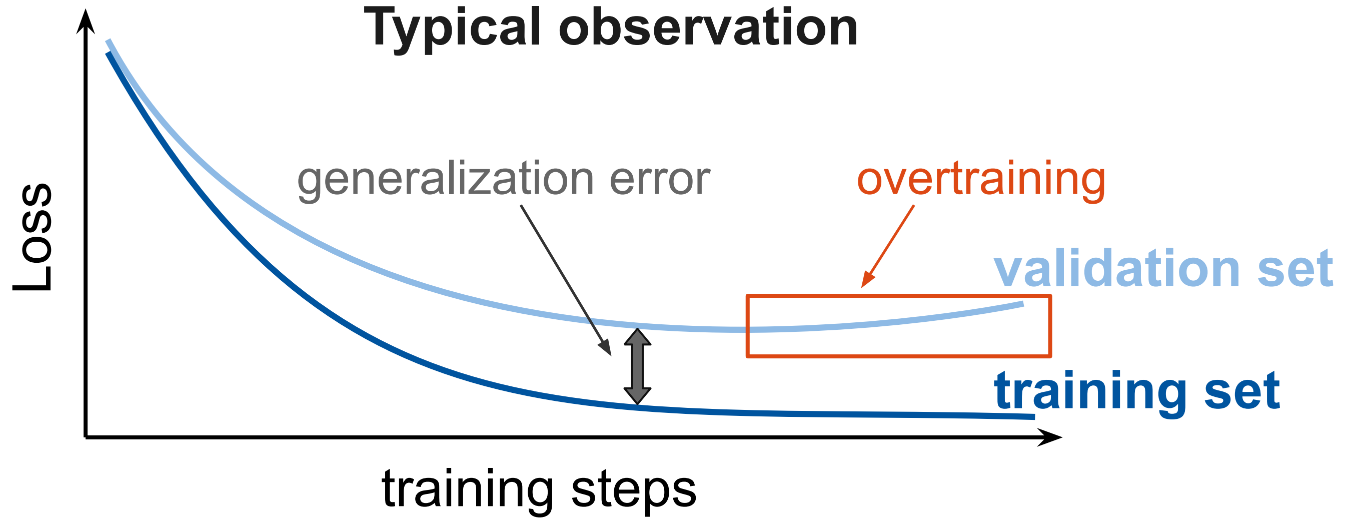
Validation

Estimate generalization error on data not used during training.
Split data into:

- **Training set:** to train the network
- **Validation set:** to monitor and tune the training (training of hyperparameter)
- **Test set:** to estimate final performance. Use only **once!**

Under- and Overtraining

- During training monitor the loss separately for training and validation set



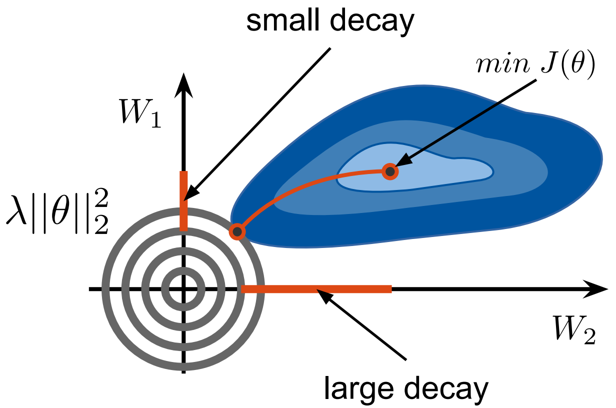
Training loss:
- decreases

Validation loss:
- is higher than training loss → **generalization gap**
- has a minimum → **overtraining**

Parameter Norm Penalties

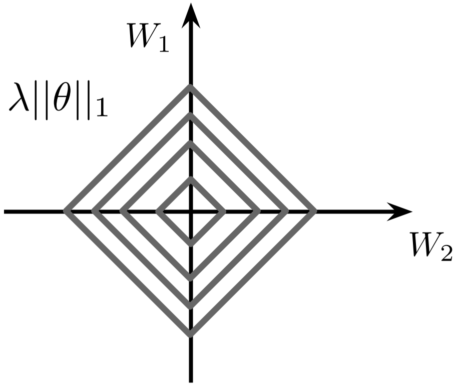
L^2 norm: (**weight decay**) $\lambda||\theta||_2^2 = \lambda(\theta_1^2 + \theta_2^2 + ...)$

- Contribution to loss dominated by largest weights
- Decay of weights which not contribute much to the reduction of the objective $J(\theta)$



L^1 norm: (**lasso**) $\lambda||\theta||_1 = \lambda(|\theta_1| + |\theta_2| + ...)$

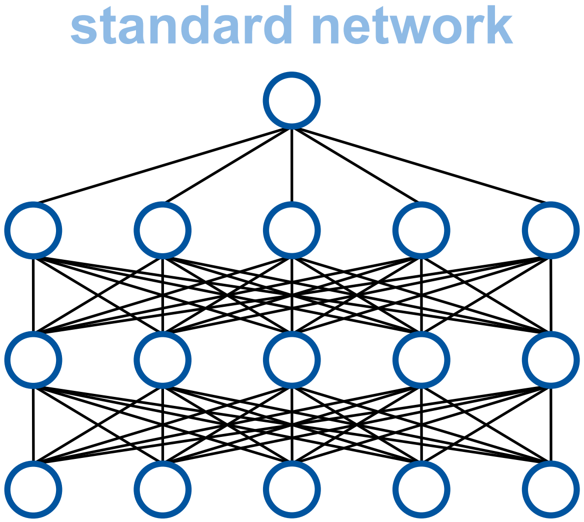
- Constant shrinking of parameters
- Allows for sparse network (feature selection mechanism)



ElasticNet: Combination of L^1 and L^2 norm

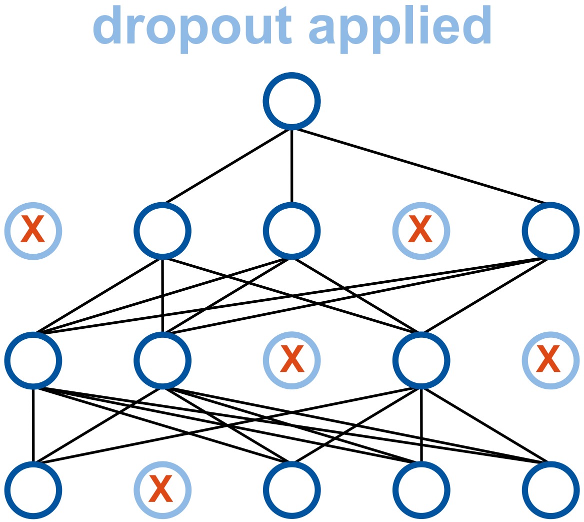
Dropout

Randomly turn of fraction p_{drop} of neurons in each training step



Typical fraction

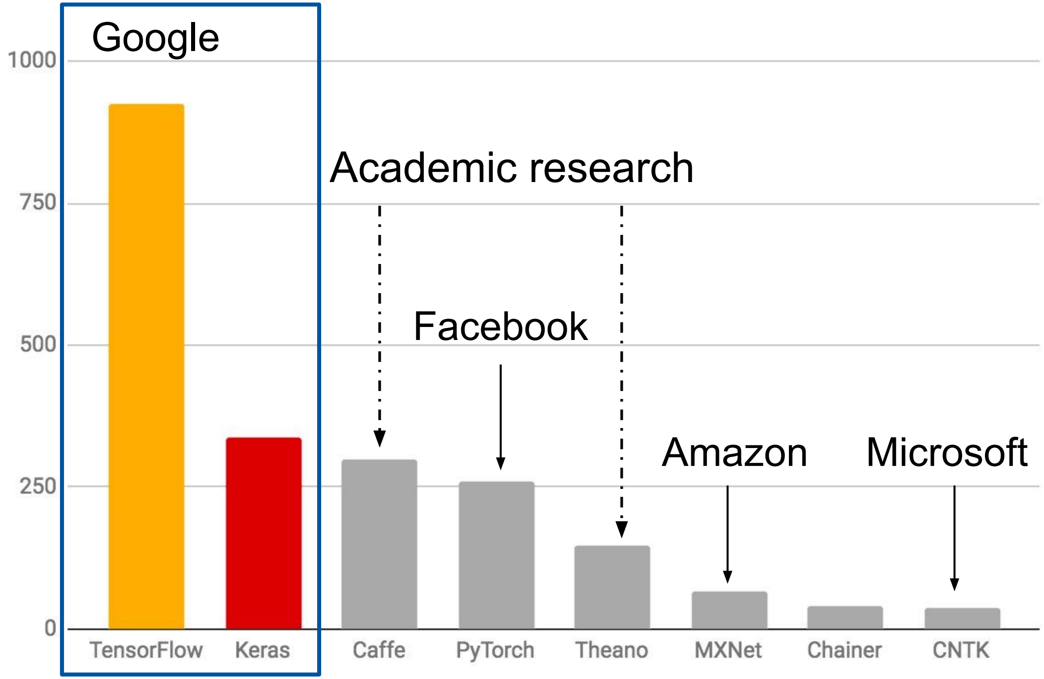
$0.2 < p_{drop} < 0.5$



- Adds noise to process of feature extraction
- Force network to train redundant representations
- During validation and test: no dropout applied → large ensemble of “submodels”

Practice I

- Software: TensorFlow 2.1 (using Keras)
- CIFAR Example



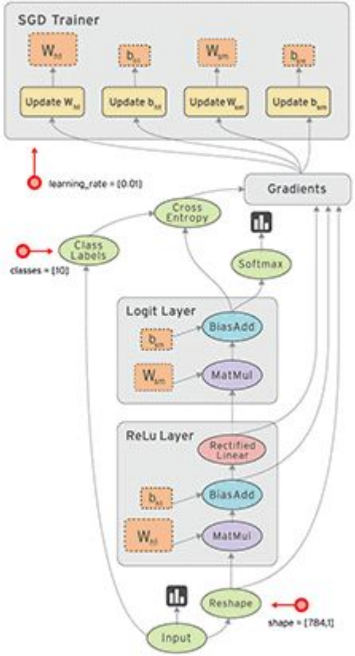
arXiv mentions as of 2018/03/07 (past 3 months)

TensorFlow

“Open source software library for numerical computation using data flowing graphs”

- **Nodes** represent mathematical operations
- **Graph edges** represent multi-dimensional data arrays (**tensors**) which **flow** through the graph

- Supports:
  - CPUs and **GPUs**
  - Desktops and mobile devices
- Released 2015, stable since Feb. 2017
- Developer: Google Brain

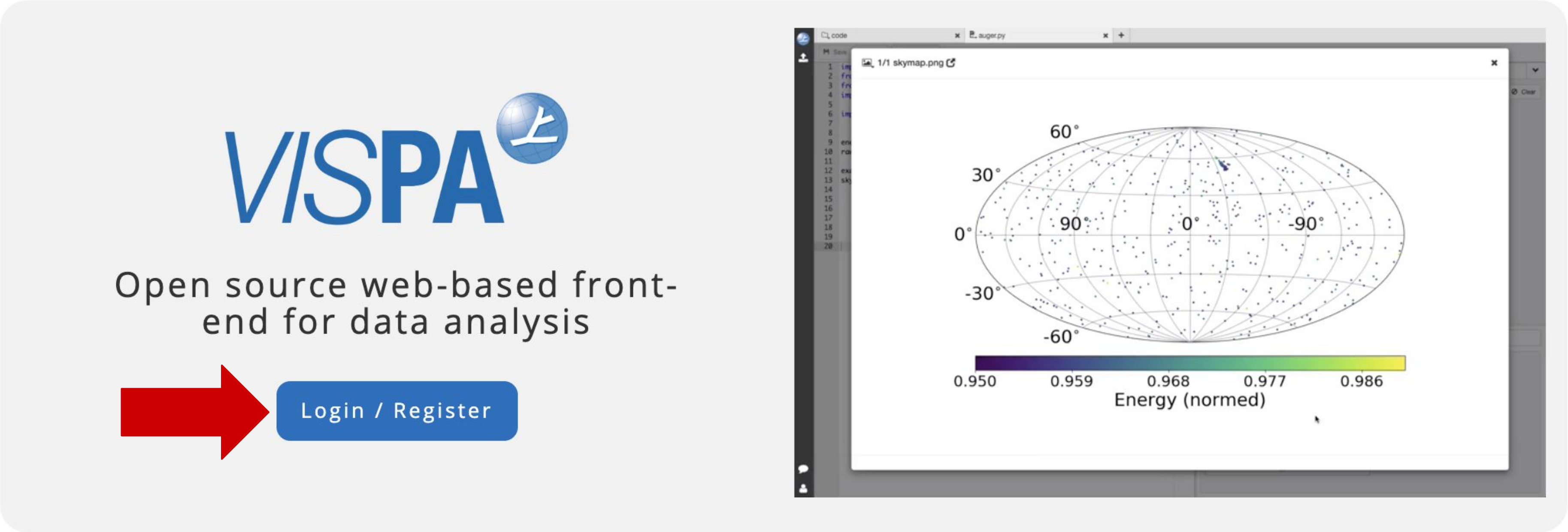


Keras

- Will use keras in this tutorial (TensorFlow backend) - https://keras.io
- High-level neural networks API, written in Python
- Concise syntax with many reasonable default settings
- Useful callbacks for monitoring the training procedure
- Nice Documentation & many examples and tutorials
- Can run on top of TensorFlow, Theano and CNTK
- Comes with TensorFlow

Create a VISPA account

Go here: https://vispa.physik.rwth-aachen.de/

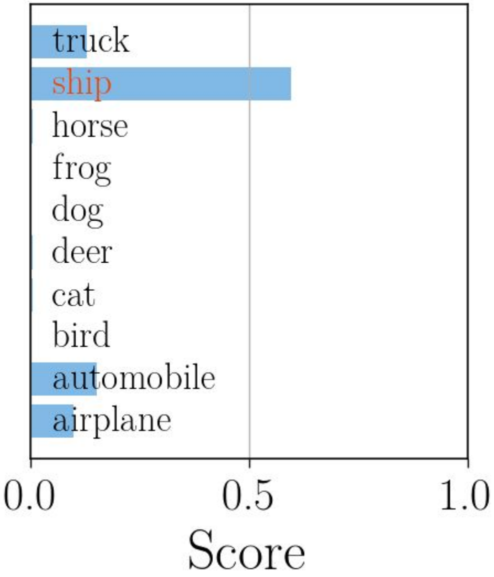


Open source web-based front-end for data analysis

Register now!!!

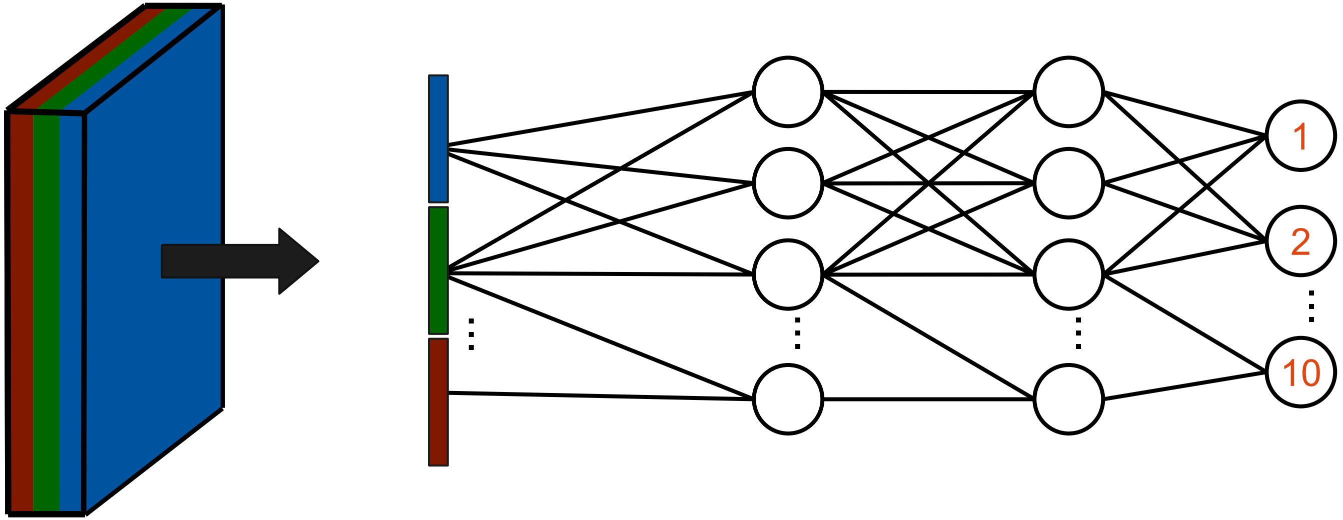
CIFAR-10 Classification Task

- 60,000 images with 10 classes
- Input $\mathbf{x} = (x_1, x_2, ..., x_{3072})$, for 32 x 32 x 3 = 3072 input features
- Output $\mathbf{y} = (y_1, y_2, ..., y_{10})$, one for each class (one-hot encoded)
  - frog, airplane, automobile, bird, cat, deer, dog, horse, ship, truck



- Model should learn to estimate the conditional probability distribution
- outputs probability for each class
- $\mathbf{y}_m(x_i|\theta) = (p_{\text{cat}}, p_{\text{dog}}, ...)$
- Take highest p_j as prediction
- Value of p_j states certainty

Fully Connected Network



- Input layer: Flatten image to 32 x 32 x 3 = 3072 vector
- Use fully connected network with some hidden layers, ReLU and dropout
- Output layer: 10 layer output with softmax
- Measure performance with independent **validation set**

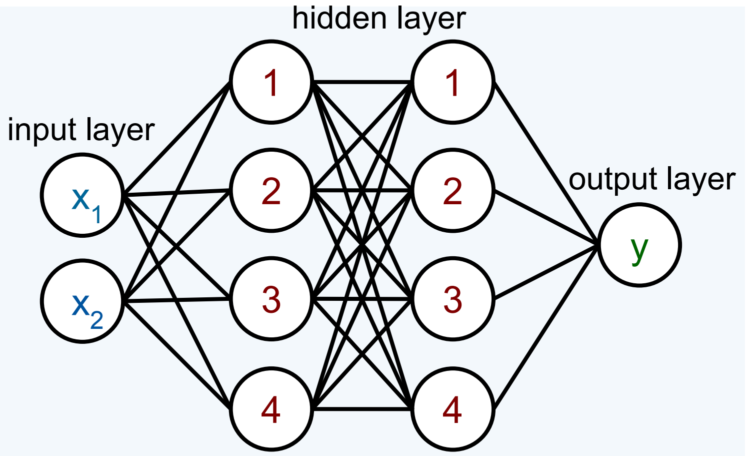
How to train your Model?

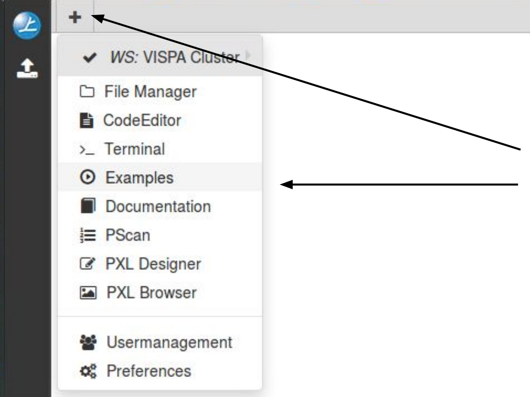
I. **Define** Model
  - Add layers, nodes, regularization, activation functions, ….)

II. **Compile** Model
  - Set Loss, optimizer settings and useful metrics

III. **Fit** Model
  - Set number of iterations and train model on given data

```
from tensorflow import keras
layers = keras.layers
models = keras.models

# setup and train a 3-layer regression network with Keras
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_dim=2))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='tanh'))
model.compile(loss='MSE', optimizer='SGD', metrics=['accuracy'])
model.fit(xdata, ydata, epochs=200)
```





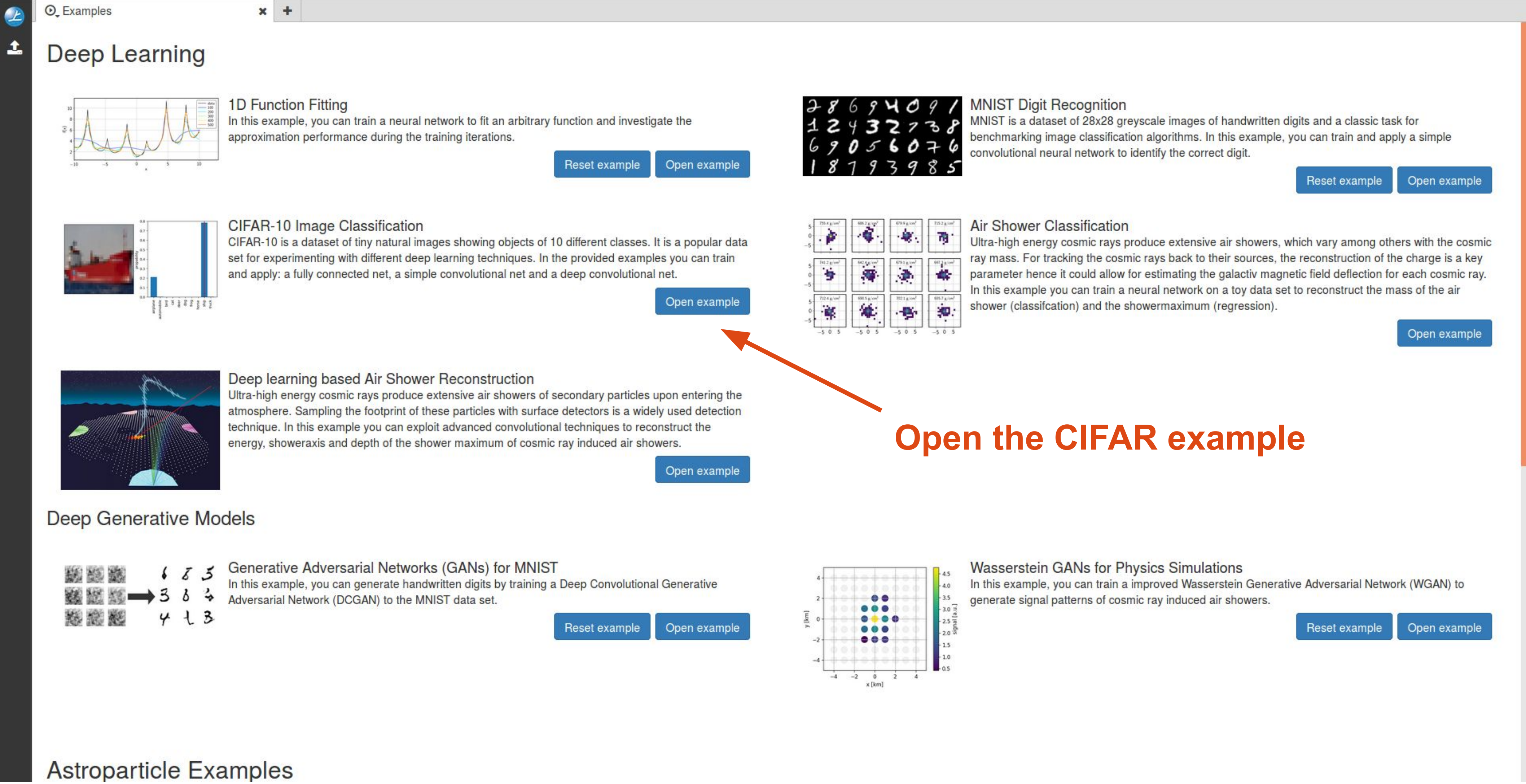
+

✓ WS: VISPA Cluster

- File Manager
- CodeEditor
- Terminal
- Examples
- Documentation
- PScan
- PXL Designer
- PXL Browser
- Usermanagement
- Preferences

Opens the example page

- Developed in Aachen (group of Prof. Erdmann)
- GPU extension
  - 29 NVIDIA GPUs
- Accessible via https://vispa.physik.rwth-aachen.de/



Deep Learning

1D Function Fitting

In this example, you can train a neural network to fit an arbitrary function and investigate the approximation performance during the training iterations.

MNIST Digit Recognition

MNIST is a dataset of 28x28 greyscale images of handwritten digits and a classic task for benchmarking image classification algorithms. In this example, you can train and apply a simple convolutional neural network to identify the correct digit.

CIFAR-10 Image Classification

CIFAR-10 is a dataset of tiny natural images showing objects of 10 different classes. It is a popular data set for experimenting with different deep learning techniques. In the provided examples you can train and apply: a fully connected net, a simple convolutional net and a deep convolutional net.

Air Shower Classification

Ultra-high energy cosmic rays produce extensive air showers, which vary among others with the cosmic ray mass. For tracking the cosmic rays back to their sources, the reconstruction of the charge is a key parameter hence it could allow for estimating the galactiv magnetic field deflection for each cosmic ray. In this example you can train a neural network on a toy data set to reconstruct the mass of the air shower (classifcation) and the showermaximum (regression).

Deep learning based Air Shower Reconstruction

Ultra-high energy cosmic rays produce extensive air showers of secondary particles upon entering the atmosphere. Sampling the footprint of these particles with surface detectors is a widely used detection technique. In this example you can exploit advanced convolutional techniques to reconstruct the energy, showeraxis and depth of the shower maximum of cosmic ray induced air showers.

Open the CIFAR example

Deep Generative Models

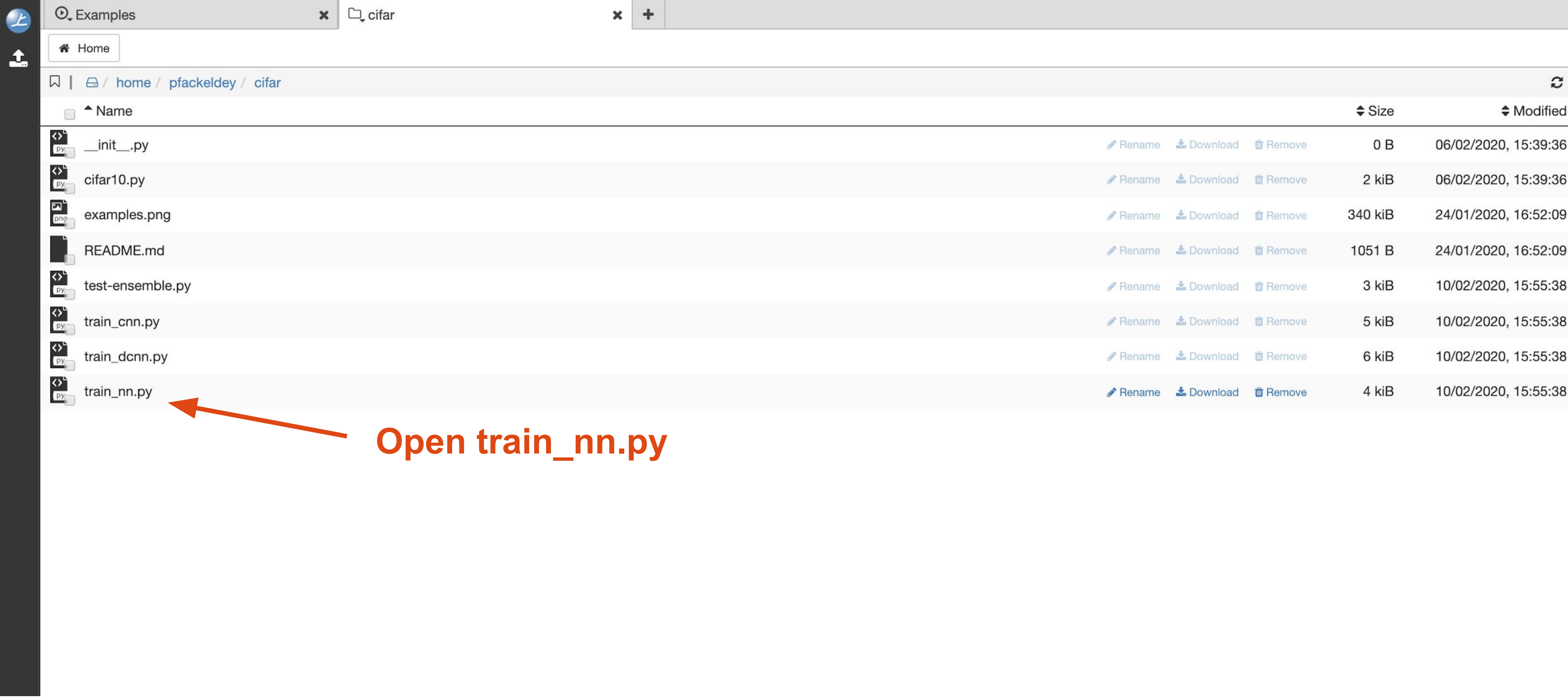
Generative Adversarial Networks (GANs) for MNIST

In this example, you can generate handwritten digits by training a Deep Convolutional Generative Adversarial Network (DCGAN) to the MNIST data set.

Wasserstein GANs for Physics Simulations

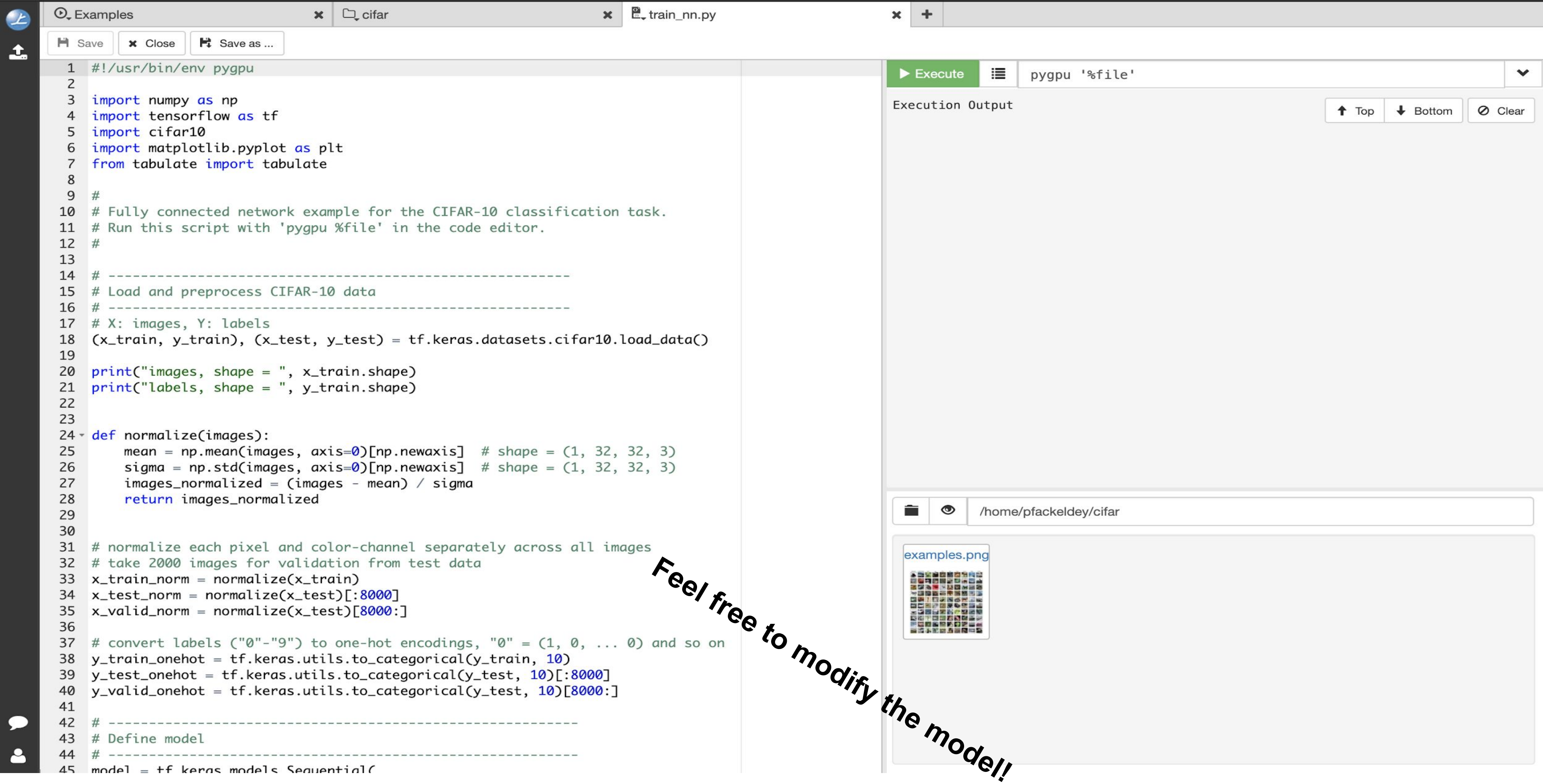
In this example, you can train a improved Wasserstein Generative Adversarial Network (WGAN) to generate signal patterns of cosmic ray induced air showers.

Astroparticle Examples



Name		Size	Modified
__init__.py	Rename Download Remove	0 B	06/02/2020, 15:39:36
cifar10.py	Rename Download Remove	2 kiB	06/02/2020, 15:39:36
examples.png	Rename Download Remove	340 kiB	24/01/2020, 16:52:09
README.md	Rename Download Remove	1051 B	24/01/2020, 16:52:09
test-ensemble.py	Rename Download Remove	3 kiB	10/02/2020, 15:55:38
train_cnn.py	Rename Download Remove	5 kiB	10/02/2020, 15:55:38
train_dcnn.py	Rename Download Remove	6 kiB	10/02/2020, 15:55:38
train_nn.py	Rename Download Remove	4 kiB	10/02/2020, 15:55:38

Open train_nn.py



```python
#!/usr/bin/env pygpu

import numpy as np
import tensorflow as tf
import cifar10
import matplotlib.pyplot as plt
from tabulate import tabulate

#
# Fully connected network example for the CIFAR-10 classification task.
# Run this script with 'pygpu %file' in the code editor.
#

# ---------------------------------------------------------
# Load and preprocess CIFAR-10 data
# ---------------------------------------------------------
# X: images, Y: labels
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

print("images, shape = ", x_train.shape)
print("labels, shape = ", y_train.shape)


def normalize(images):
    mean = np.mean(images, axis=0)[np.newaxis]  # shape = (1, 32, 32, 3)
    sigma = np.std(images, axis=0)[np.newaxis]  # shape = (1, 32, 32, 3)
    images_normalized = (images - mean) / sigma
    return images_normalized


# normalize each pixel and color-channel separately across all images
# take 2000 images for validation from test data
x_train_norm = normalize(x_train)
x_test_norm = normalize(x_test)[:8000]
x_valid_norm = normalize(x_test)[8000:]

# convert labels ("0"-"9") to one-hot encodings, "0" = (1, 0, ... 0) and so on
y_train_onehot = tf.keras.utils.to_categorical(y_train, 10)
y_test_onehot = tf.keras.utils.to_categorical(y_test, 10)[:8000]
y_valid_onehot = tf.keras.utils.to_categorical(y_test, 10)[8000:]

# ----------------------------------------------------------
# Define model
# ----------------------------------------------------------
model = tf.keras.models.Sequential(
```

Feel free to modify the model!

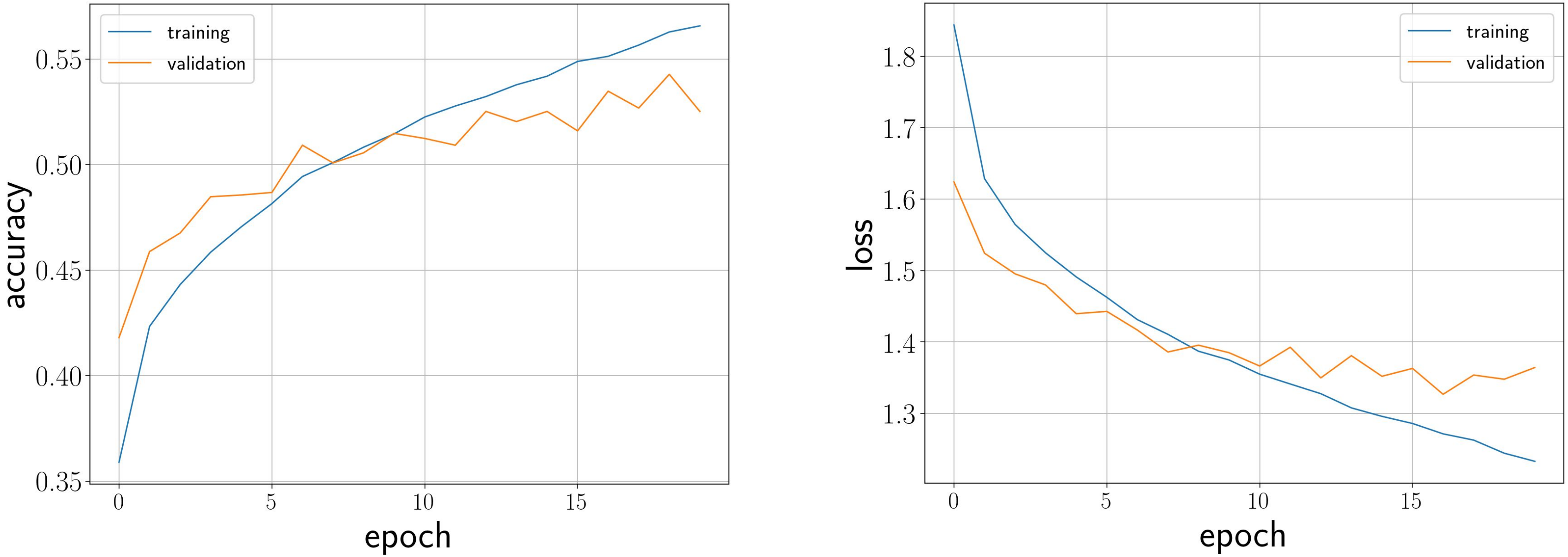
CIFAR 10: Exercise

- Model:
  - Add layers or nodes
  - Add regularization
    - Dropout, penalties
  - Activation functions

- Modify
  - Batch size, epochs
  - Optimizer, learning rate

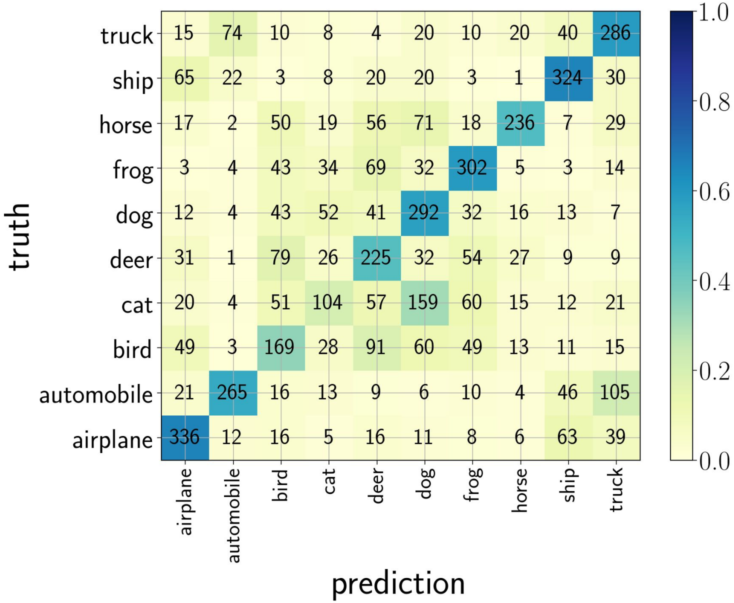
```
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(32, 32, 3)),
        tf.keras.layers.Dense(256, activation="relu"),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Dense(256, activation="relu"),
        tf.keras.layers.Dense(10, activation="softmax"),
    ],
    name="nn",
)

model.fit(
    x_train_norm,
    y_train_onehot,
    batch_size=32,
    epochs=20,
    verbose=2,
    validation_data=(x_valid_norm, y_valid_onehot),
)
```

Results

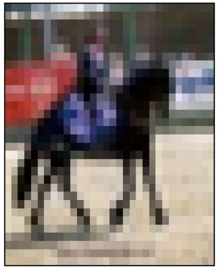


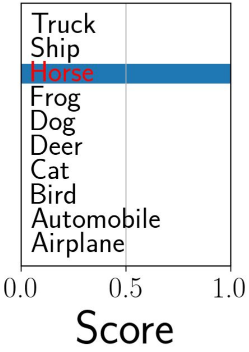
- Model roughly converged, accuracy ~ 50%
- Large generalization gap
- Fully connected network is prone to overtraining

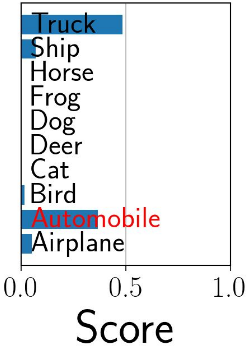
Feature Correlation

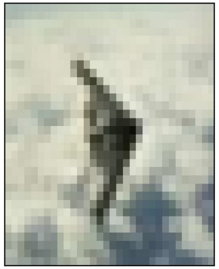


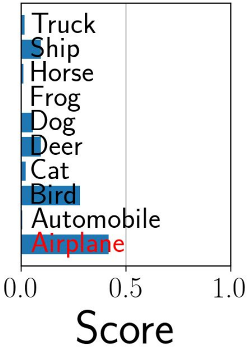
truth \ prediction	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
truck	15	74	10	8	4	20	10	20	40	286
ship	65	22	3	8	20	20	3	1	324	30
horse	17	2	50	19	56	71	18	236	7	29
frog	3	4	43	34	69	32	302	5	3	14
dog	12	4	43	52	41	292	32	16	13	7
deer	31	1	79	26	225	32	54	27	9	9
cat	20	4	51	104	57	159	60	15	12	21
bird	49	3	169	28	91	60	49	13	11	15
automobile	21	265	16	13	9	6	10	4	46	105
airplane	336	12	16	5	16	11	8	6	63	39







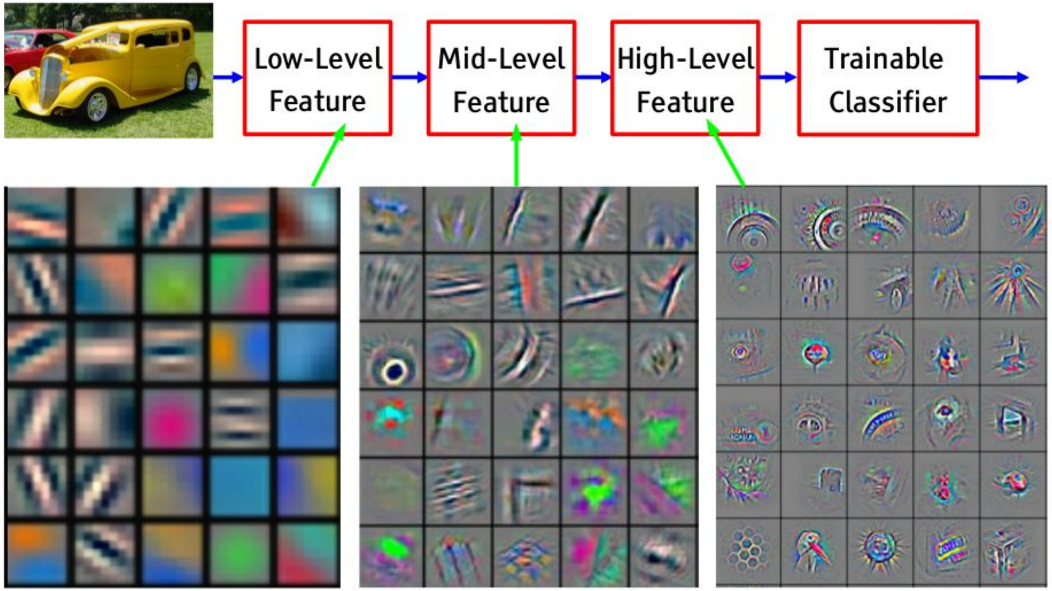




- Confusion matrix shows which classes have correlated features
  - Cat←→dog, truck←→automobile etc.

Convolutional Neural Networks

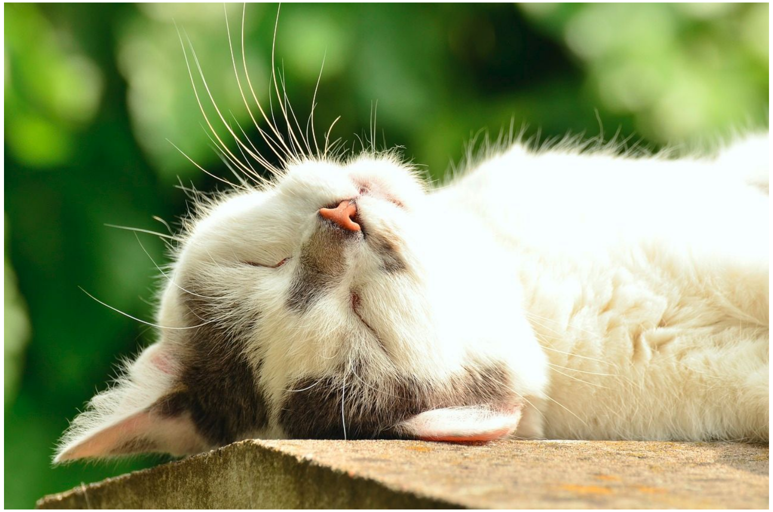
- Natural Images
- Convolutional Layers
  - Strides, Pooling, Padding



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

https://arxiv.org/abs/1311.2901

Natural Images



Automate task for humans, very challenging for machine learning models:

- High dimensional input (up to millions of pixels)
- Many possible classes depending on task
- Multiple variations
  - Viewing angle, light conditions, deformation, object variations, occlusions….

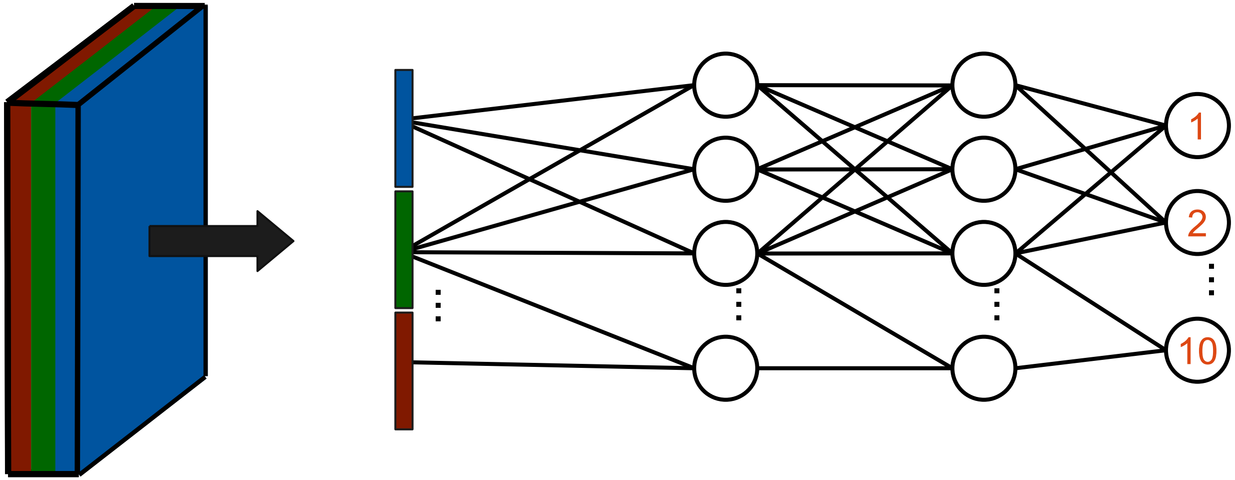
Fully Connected Network

- Input layer: Flatten image to 32 x 32 x 3 = 3072 vector
- Fully connected: every pixel connected with each other

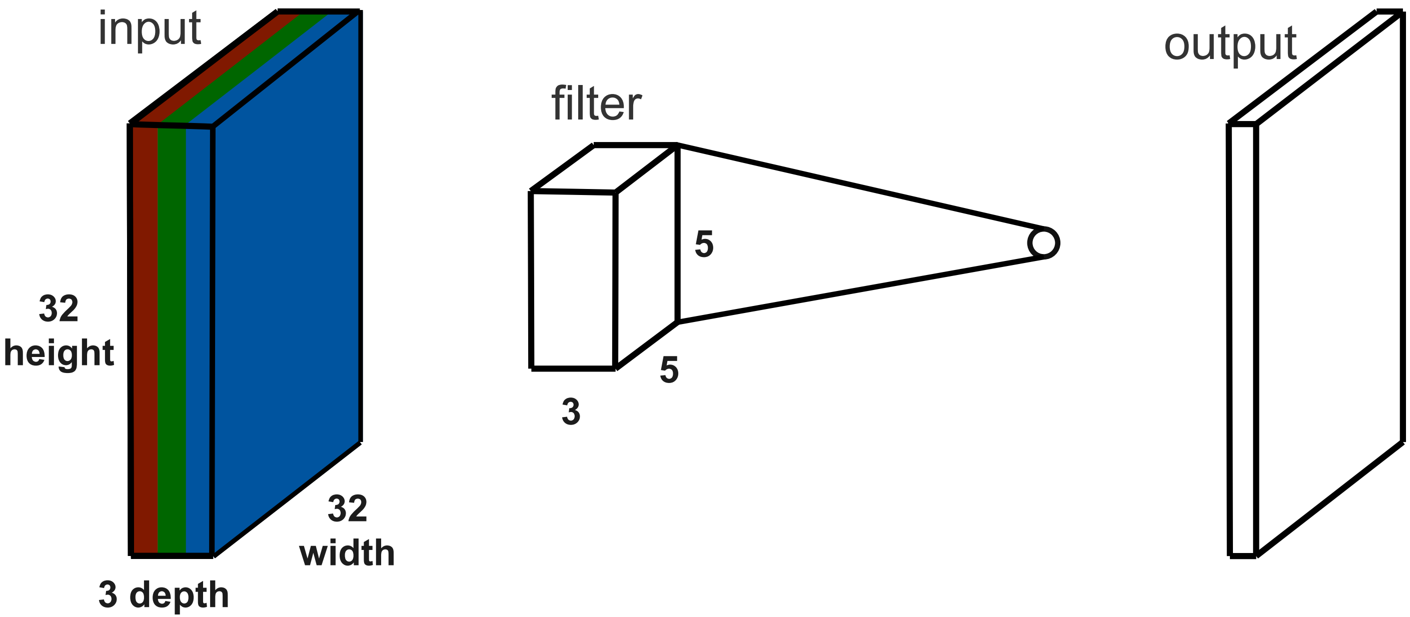
 Huge number of adaptive parameters per layer

 No use of translational variance

 No prior on local correlations

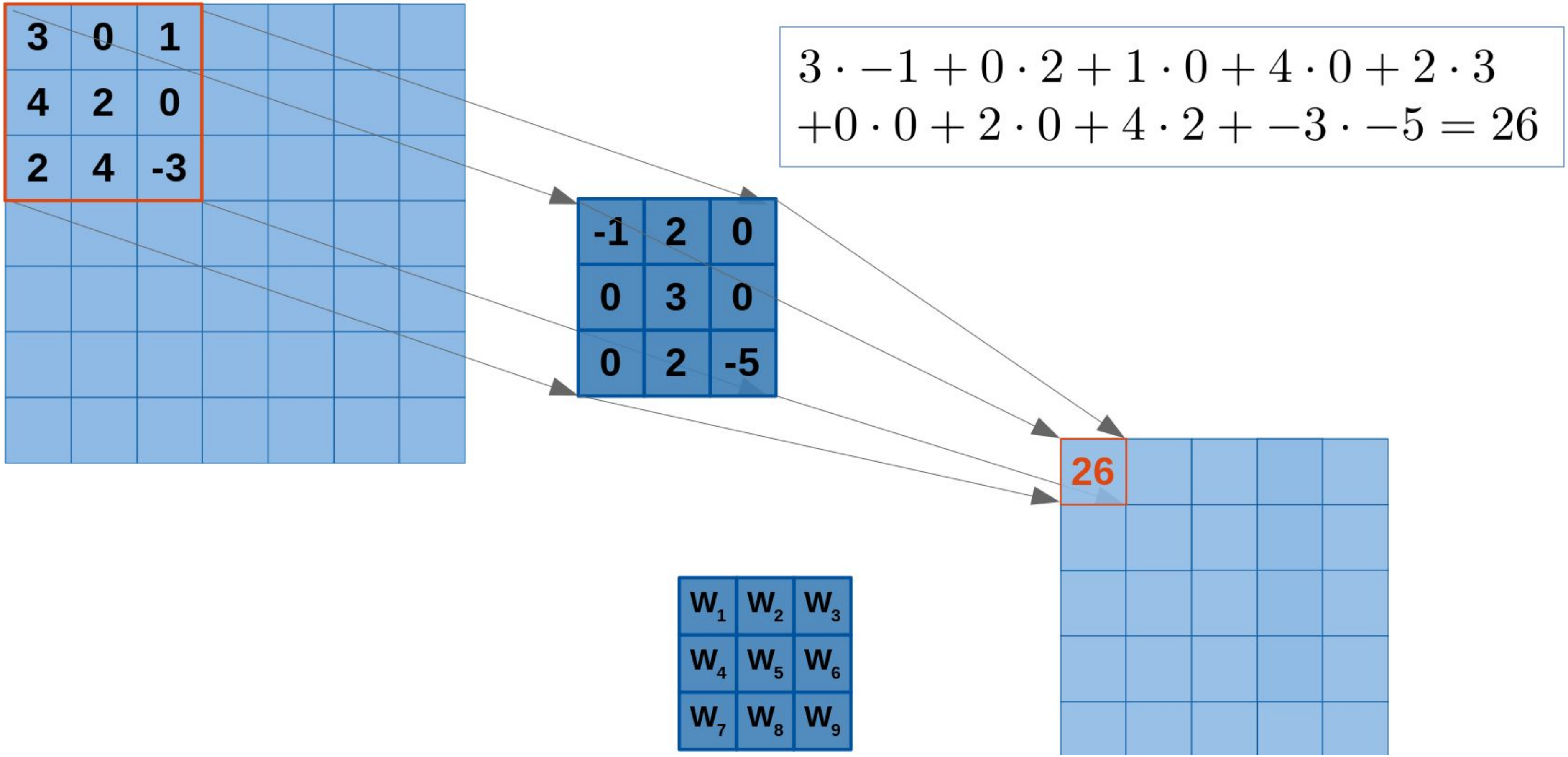


2D Convolutional Neural Networks



- Consider input volume (width x height x depth), eg. 3 color channels
- Use convolutional filter with smaller width and height but same depth
- Slide several filters over entire volume and calculate linear transformation to get **one** output value for each position

Convolutional Operation



$$3 \cdot -1 + 0 \cdot 2 + 1 \cdot 0 + 4 \cdot 0 + 2 \cdot 3$$
$$+ 0 \cdot 0 + 2 \cdot 0 + 4 \cdot 2 + -3 \cdot -5 = 26$$

2D Convolutional Neural Networks

- Filter scans input for the presence of one specific feature

Edge

-1	-1	-1
-1	8	-1
-1	-1	-1

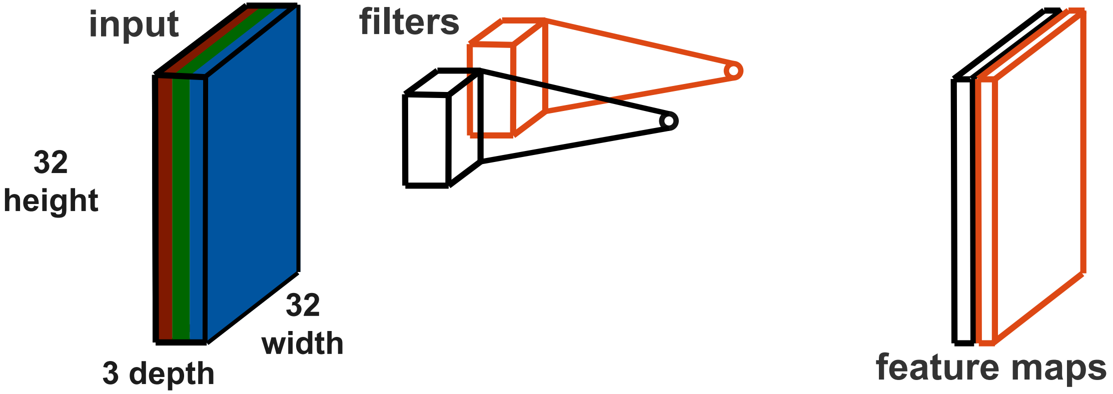
Horizontal edge

-1	-1	-1
2	2	2
-1	-1	-1

Diagonal edge

-1	-1	2
-1	2	-1
2	-1	-1

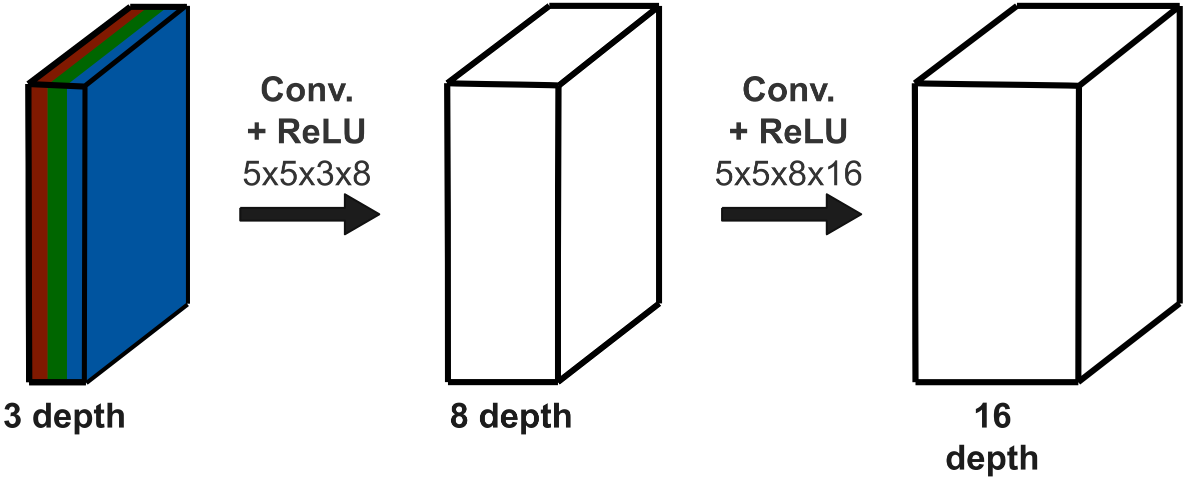
- Convolutional network learns task related filters **itself**
- Use multiple filters and stack the resulting **feature maps** depth-wise

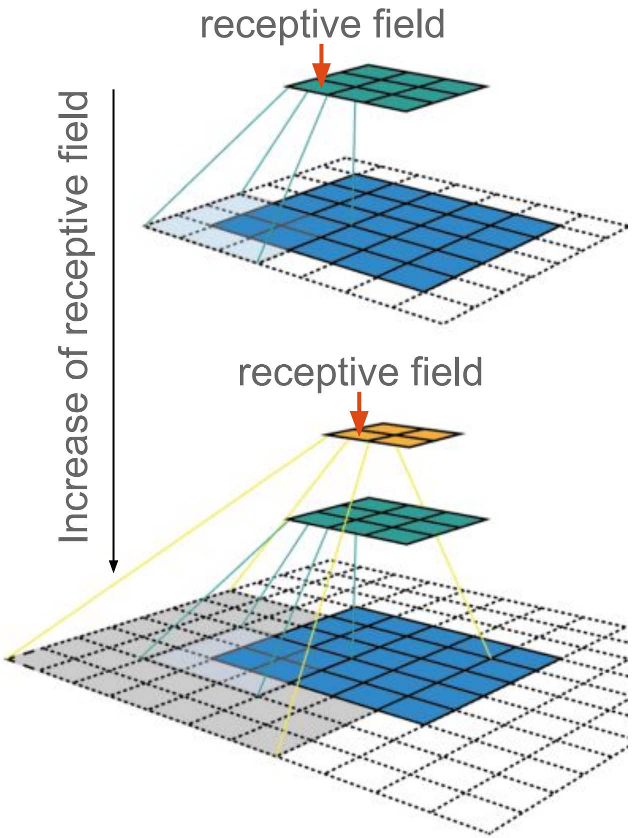


2D Convolutional Operation

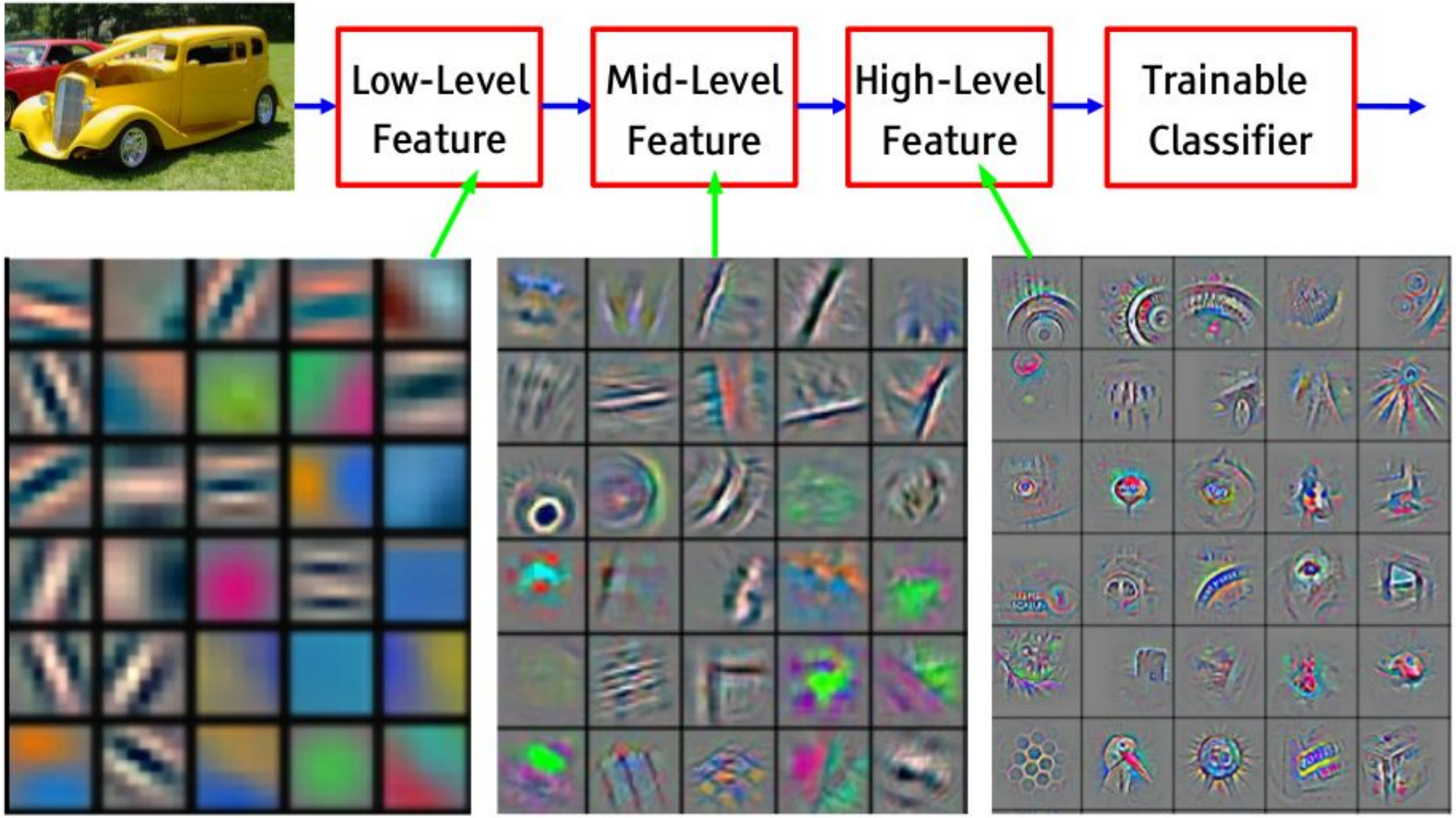
Stack multiple convolutional layers + activations

- Each convolution acts on feature map of previous layer
- Increasing feature hierarchy
- Increasing of receptive field





Feature Hierarchy



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

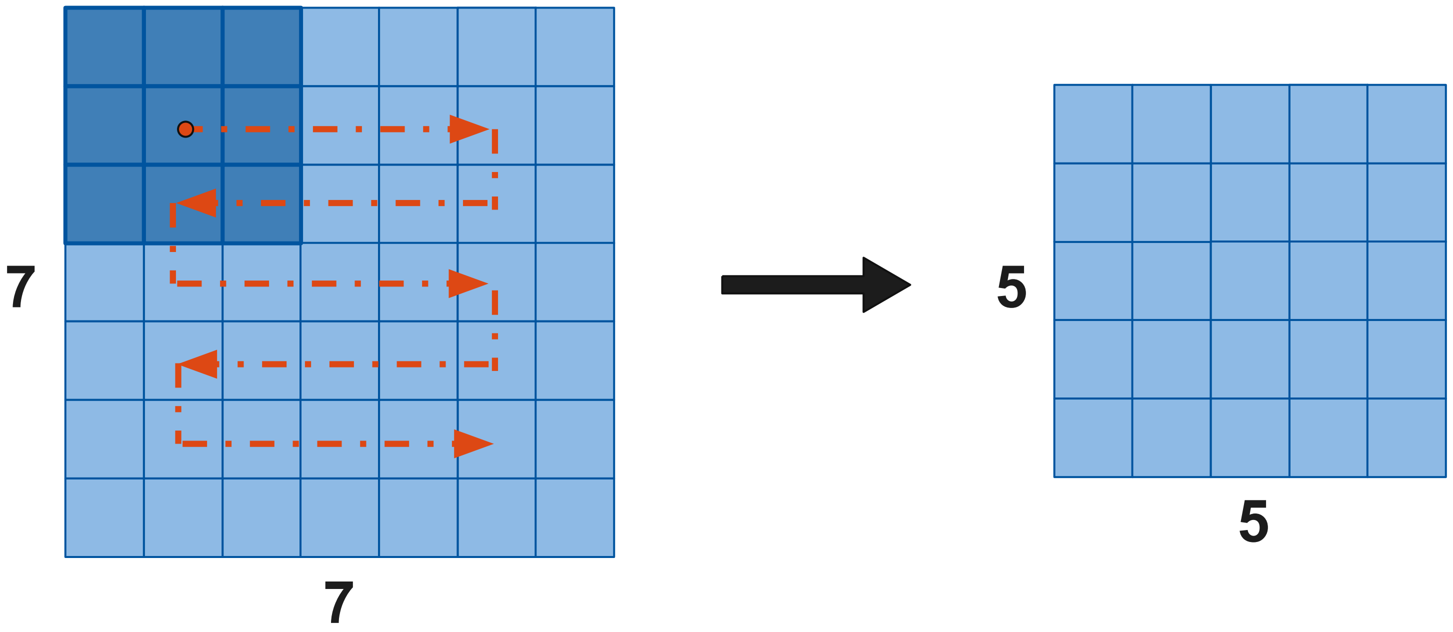
https://arxiv.org/abs/1311.2901

Spatial Output Size

Standard convolution reduces the output size due to extent of the filter

- Sets upper bound to the number of convolutional layers

- **Example:** Convolution with 3 x 3 filter

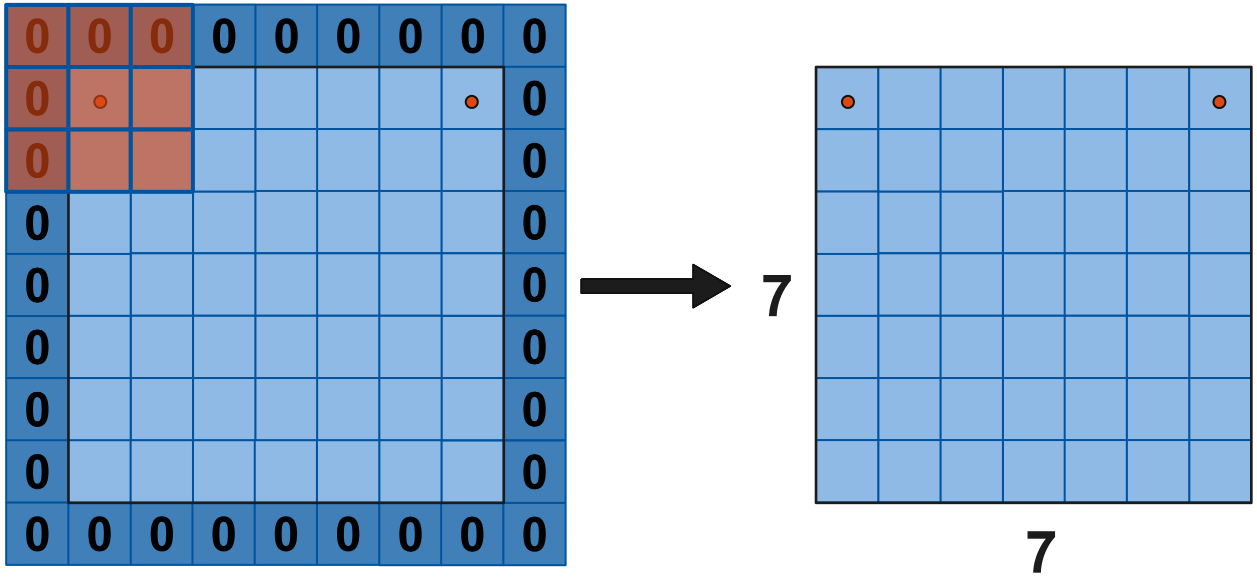


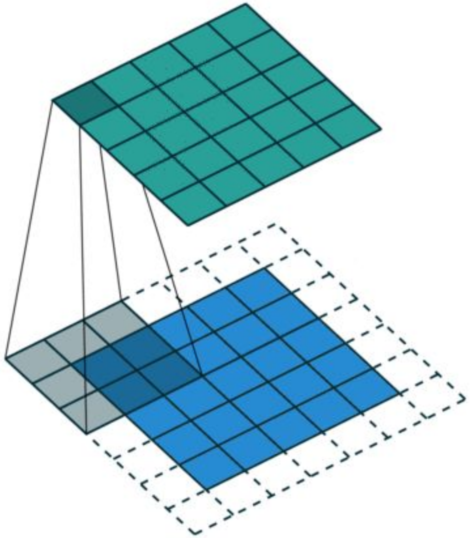
Padding

Add zeros around image borders to conserve the spatial extent of the input

- Prevents fast shrinking of the input data (image)

- **Example:** Convolution with 3 x 3 filter and padding





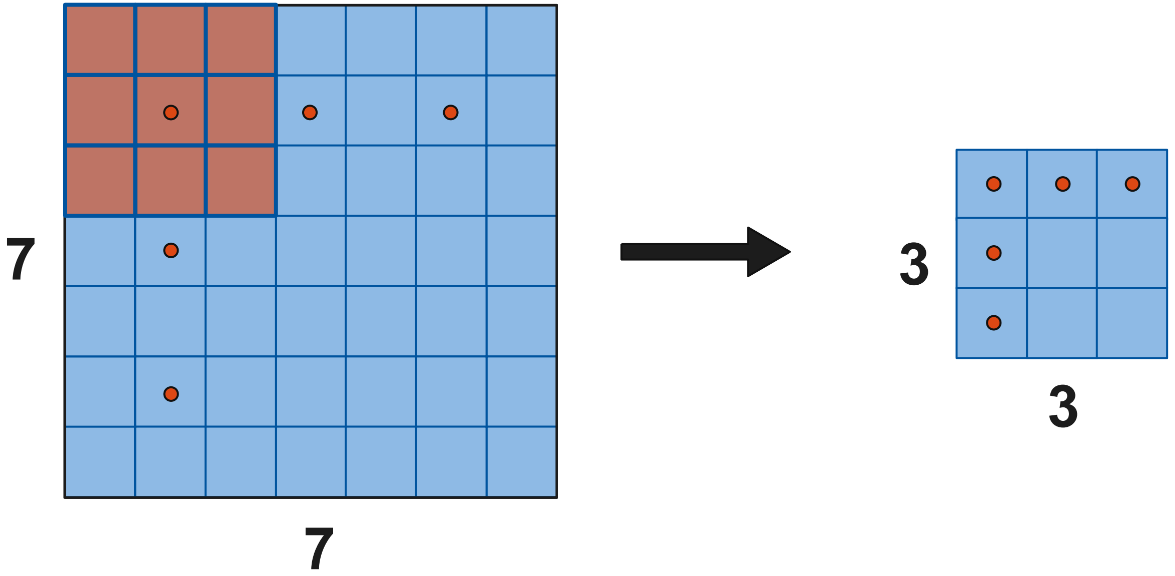
Paul-Louis Pröve, Towards Data Science

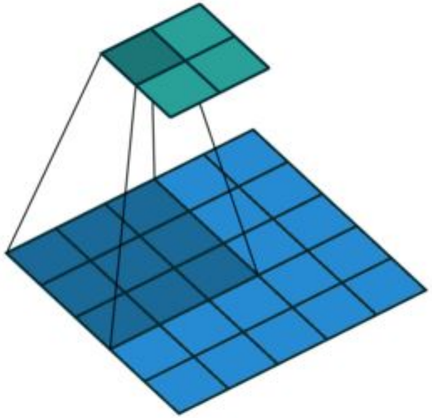
Striding

Using a larger stride when sliding over the input, reduces the output size

- Useful for switching to smaller image sizes / larger scales

- **Example:** Convolution with 3 x 3 filter and stride of 2





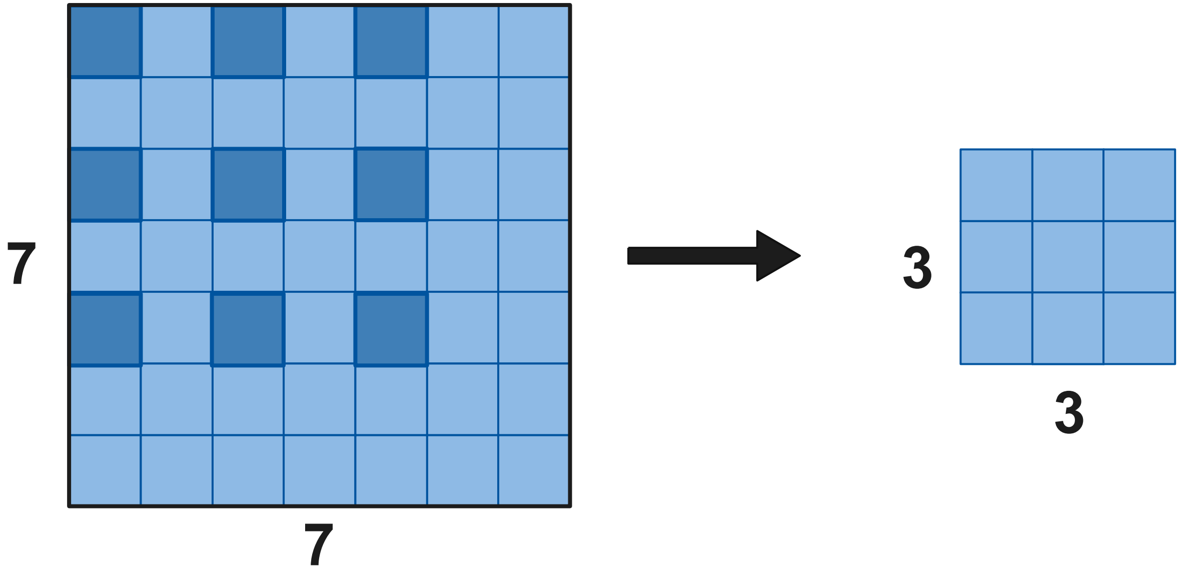
Paul-Louis Pröve,
Towards Data Science

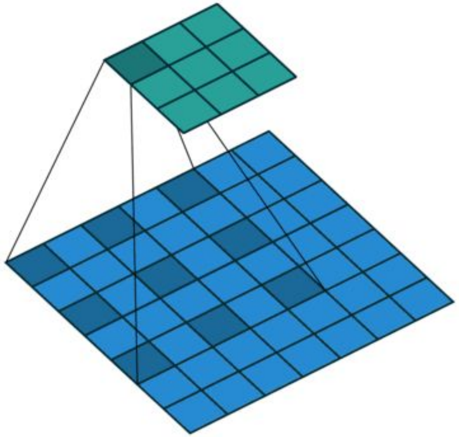
Dilating

Dilation leaves holes in where the filter is applied (also called **atrous convolution**)

- Useful for aggressively merging spatial information in large images
- Allows for a large field of view

- **Example:** Convolution with 3 x 3 filter and dilation 1





Paul-Louis Pröve,
Towards Data Science

Pooling

Sub-sample the input to reduce the output size

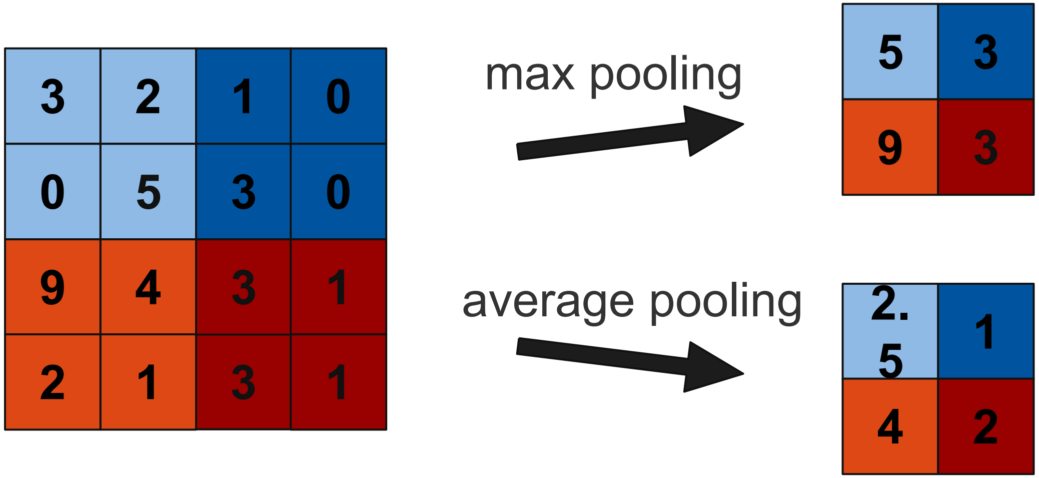
- Used to merge semantically similar features

Average pooling: Take the mean of each patch

Max pooling: Take the maximum of each patch

Global pooling: Take maximum/average over complete image

- **Example:**
  Pooling using 2 x 2 patches
  and a stride of 2



3	2	1	0
0	5	3	0
9	4	3	1
2	1	3	1

max pooling

5	3
9	3

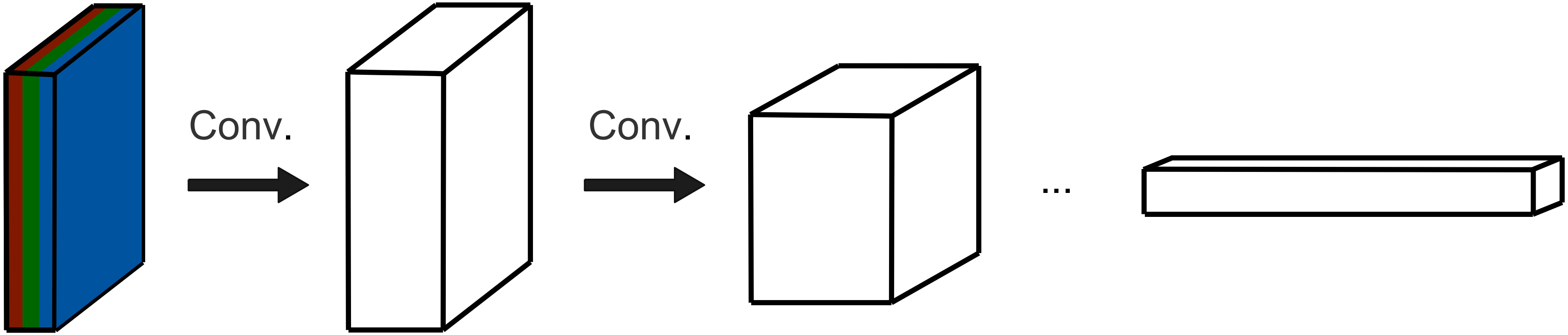
average pooling

2.5	1
4	2

Convolutional Pyramid

ConvNet architectures usually have a pyramidal shape. For deeper layers:

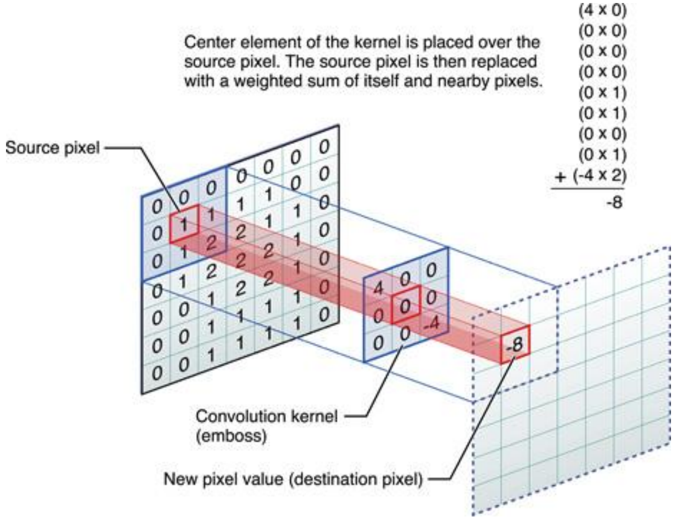
- Increasing of feature space
- Decreasing of spatial extent



- Spatial information is converted to representational features with increasing hierarchy

Summary

- 2D Convolution acts on 3D input (width x height x depth)
- Slide small filter over input and make linear transformation (dot product + bias)
- Hyperparameter:
  - Size of filter, typically (1 x 1), (3 x 3), (5 x 5) or (7 x 7)
  - Number of filters (feature maps)
  - **Padding** (maintain spatial extent)
  - **Striding** or **pooling** (reduce spatial extent)
  - **Dilation** to merge information over larger scales
- Reduction of parameters using symmetry in data:
  - Prior on **local correlations** (use small filters)
  - **Translational invariance** (weight sharing)



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

Convolutional Layers - Keras

- Same Syntax as for fully connected layers

```
layers.Convolution2D(32, kernel_size=(5, 5), padding='same', activation='relu', strides=(2, 2))
```

- Layer with 32 filters
- Size of filter 5x5 pixels
- Stride of 2 in both directions
- Use padding = ‘same’ to keep spatial dimension (else padding = ‘valid’)
- And ReLu activation

```
layers.MaxPooling2D((2,2), strides=(2, 2))          //          layers.AveragePooling2D((2,2), strides=(2, 2))
```

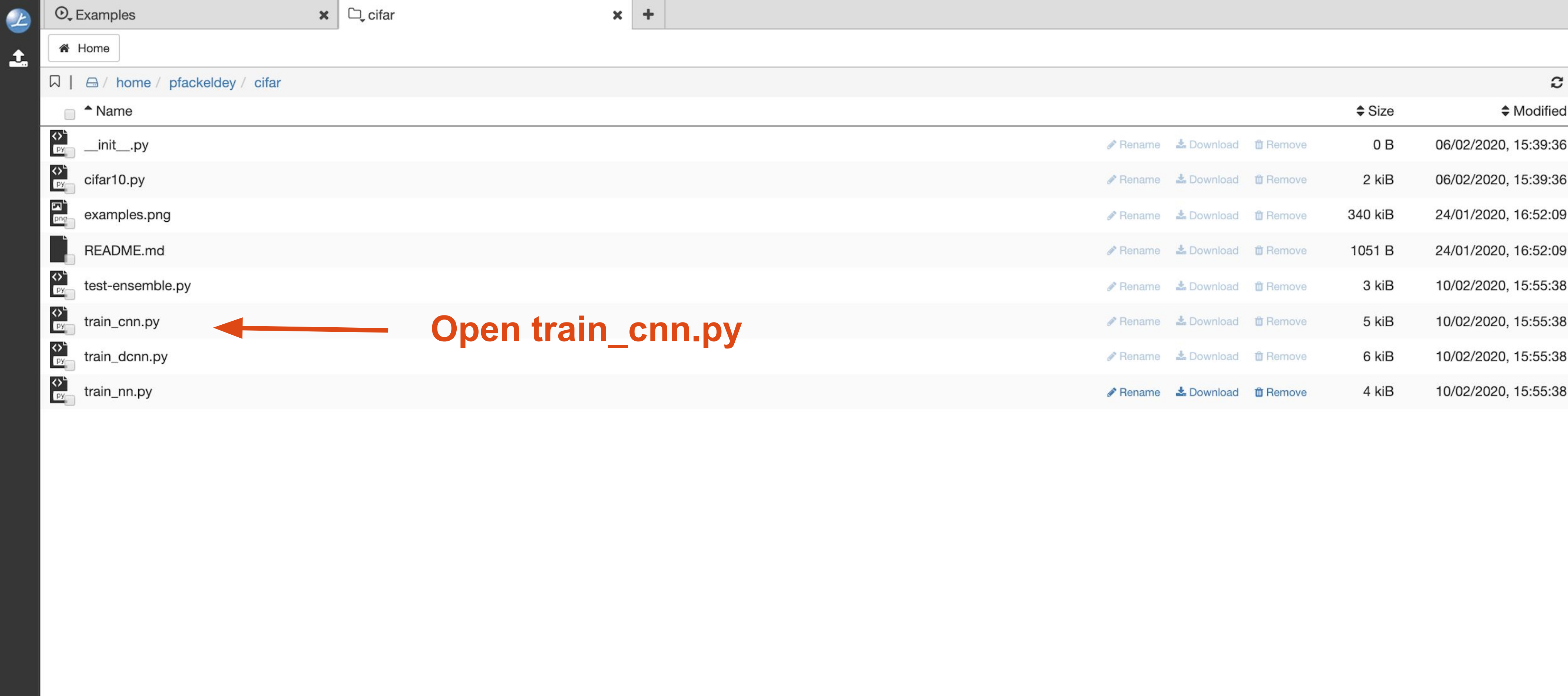
- Pooling layer with pooling size of 2x2 pixels and a stride of 2 in both dimensions

```
layers.Flatten()
```

- Layer flattens output to vector → allows use of Dense layers after Convolutions

```
layers.GlobalMaxPooling2D()                    //                    layers.GlobalAveragePooling2D()
```

- Pooling operation on complete feature map → (remove all spatial dimensions)



Name	Size	Modified
__init__.py	0 B	06/02/2020, 15:39:36
cifar10.py	2 kiB	06/02/2020, 15:39:36
examples.png	340 kiB	24/01/2020, 16:52:09
README.md	1051 B	24/01/2020, 16:52:09
test-ensemble.py	3 kiB	10/02/2020, 15:55:38
train_cnn.py	5 kiB	10/02/2020, 15:55:38
train_dcnn.py	6 kiB	10/02/2020, 15:55:38
train_nn.py	4 kiB	10/02/2020, 15:55:38

Open train_cnn.py

CIFAR 10 – CNN: Exercise

Model – add:

- Conv. layers and filters
- Pooling, Dense (FC) layers
- Regularization (after Flatten)

Model – modify:

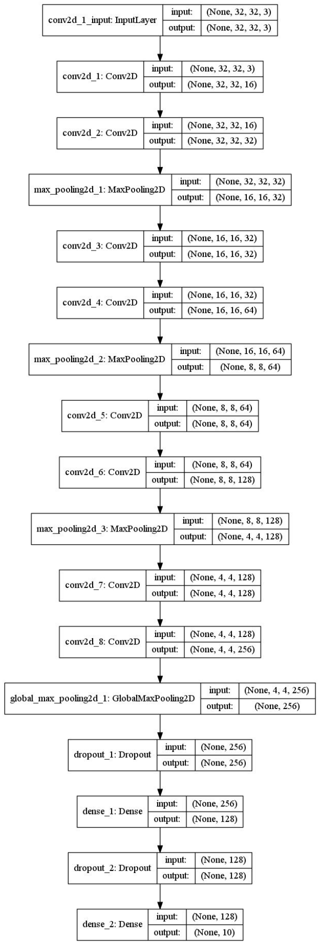
- Batch size, epochs
- Kernel size, strides
- Optimizer, learning rate

```
model = models.Sequential([
    tf.keras.layers.Convolution2D(32, kernel_size=(5, 5), strides=(2, 2),
activation="relu", input_shape=(32, 32, 3)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Convolution2D(64, kernel_size=(5, 5), strides=(2, 2),
activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Convolution2D(128, kernel_size=(5, 5), strides=(2,
2), activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(10, activation="softmax"),
])
```

- **Can you achieve >75% validation accuracy?**

CIFAR 10 – Deep Convolutional Network

```
model = models.Sequential([
    layers.Convolution2D(16, kernel_size=(3, 3), padding='same', activation='elu',
                                    input_shape=(32, 32, 3)),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(32, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(64, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(128, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.MaxPooling2D((2,2)),
    layers.Convolution2D(128, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.Convolution2D(256, kernel_size=(3, 3), padding='same', activation='elu'),
    layers.GlobalMaxPooling2D(),
    layers.Dropout(0.5),
    layers.Dense(128, kernel_regularizer=keras.regularizers.l1_l2(l1=0.025, l2=0.025),
                      activation='elu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')])
```



conv2d_1_input: InputLayer | input: (None, 32, 32, 3) | output: (None, 32, 32, 3)

conv2d_1: Conv2D | input: (None, 32, 32, 3) | output: (None, 32, 32, 16)

conv2d_2: Conv2D | input: (None, 32, 32, 16) | output: (None, 32, 32, 32)

max_pooling2d_1: MaxPooling2D | input: (None, 32, 32, 32) | output: (None, 16, 16, 32)

conv2d_3: Conv2D | input: (None, 16, 16, 32) | output: (None, 16, 16, 32)

conv2d_4: Conv2D | input: (None, 16, 16, 32) | output: (None, 16, 16, 64)

max_pooling2d_2: MaxPooling2D | input: (None, 16, 16, 64) | output: (None, 8, 8, 64)

conv2d_5: Conv2D | input: (None, 8, 8, 64) | output: (None, 8, 8, 64)

conv2d_6: Conv2D | input: (None, 8, 8, 64) | output: (None, 8, 8, 128)

max_pooling2d_3: MaxPooling2D | input: (None, 8, 8, 128) | output: (None, 4, 4, 128)

conv2d_7: Conv2D | input: (None, 4, 4, 128) | output: (None, 4, 4, 128)

conv2d_8: Conv2D | input: (None, 4, 4, 128) | output: (None, 4, 4, 256)

global_max_pooling2d_1: GlobalMaxPooling2D | input: (None, 4, 4, 256) | output: (None, 256)

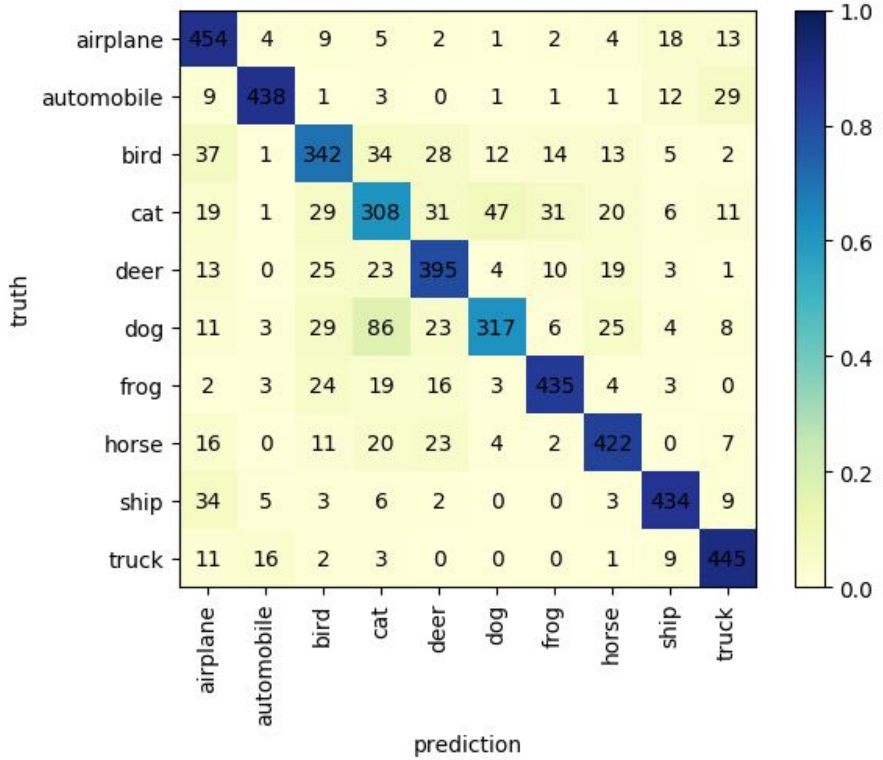
dropout_1: Dropout | input: (None, 256) | output: (None, 256)

dense_1: Dense | input: (None, 256) | output: (None, 128)

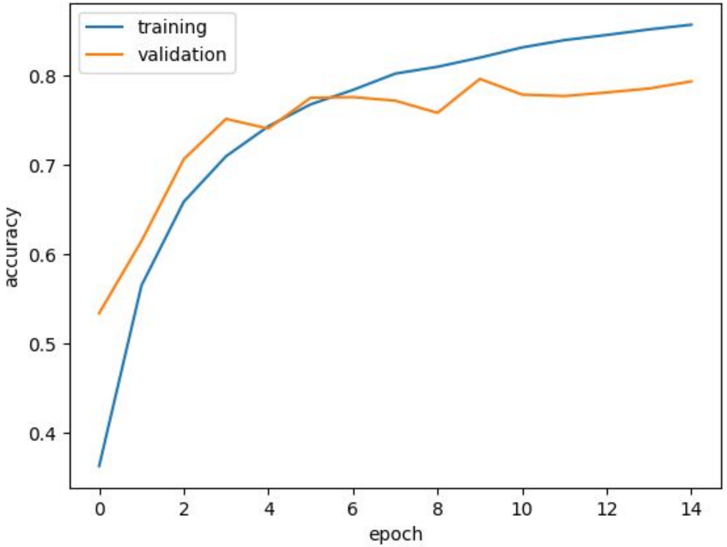
dropout_2: Dropout | input: (None, 128) | output: (None, 128)

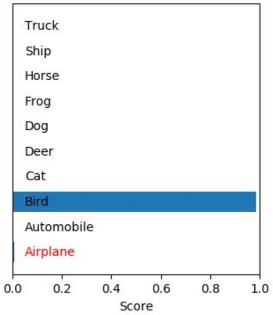
dense_2: Dense | input: (None, 128) | output: (None, 10)

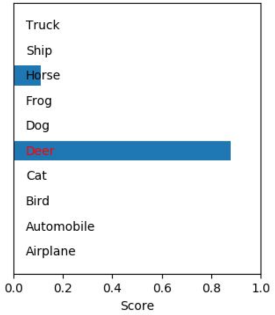
Results



truth \ prediction	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	454	4	9	5	2	1	2	4	18	13
automobile	9	438	1	3	0	1	1	1	12	29
bird	37	1	342	34	28	12	14	13	5	2
cat	19	1	29	308	31	47	31	20	6	11
deer	13	0	25	23	395	4	10	19	3	1
dog	11	3	29	86	23	317	6	25	4	8
frog	2	3	24	19	16	3	435	4	3	0
horse	16	0	11	20	23	4	2	422	0	7
ship	34	5	3	6	2	0	0	3	434	9
truck	11	16	2	3	0	0	0	1	9	445







Links & Resources

- Erdmann, Glombitza, Klemradt: Deep Learning in Physics Research, Summer Term Lecture Series RWTH
- TensorFlow Playground: https://playground.tensorflow.org
- Deep Learning (Goodfellow, Bengio, Courville), MIT Press, ISBN: 0262035618
- http://www.deeplearningbook.org/
- Neural Networks and Deep Learning (Nielson) - http://neuralnetworksanddeeplearning.com/
- CS231n - Convolutional Neural Networks for Visual Recognition (Kaparthy)
- http://cs231n.stanford.edu/syllabus.html
- Deep Learning by Google (Vanhoucke), Udacity https://www.udacity.com/course/deep-learning--ud730
- An Introduction to different Types of Convolutions in Deep Learning, Paul-Louis Pröve
- https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
- Deep Learning with Python, Francois Chollet
- The CIFAR-10 dataset - https://www.cs.toronto.edu/~kriz/cifar.html
- Deep Learning-based Reconstruction of Cosmic Ray-induced Air Showers - Erdmann, Glombitza, Walz https://doi.org/10.1016/j.astropartphys.2017.10.006

Introduction to Deep Learning

Additional Material

Martin Erdmann, Peter Fackeldey

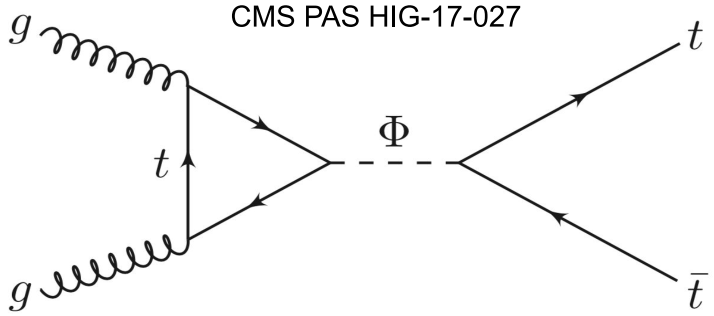
(slide credits: Jonas Glombitza)

RWTH Aachen

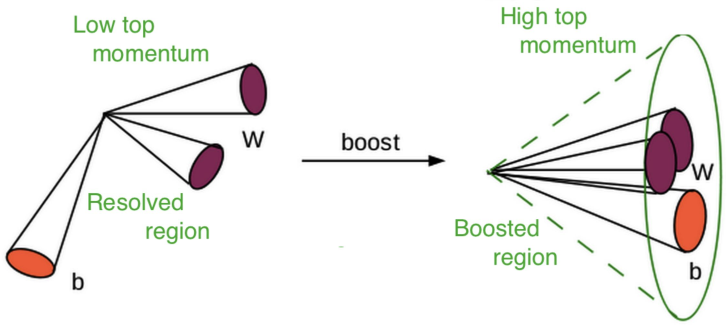
Toptagging in HEP: Motivation

- Many BSM searches with $t\bar{t}$ final state
- Especially interesting in BSM Higgs physics
  ➡ <u>strong</u> coupling to top quarks
- Topology: 2 fatjets in the final state
- We need a proper top tagging algorithm!
- Classic approach: use N-subjettiness and soft drop mass
- Our approach: use DNNs and CNNs

CMS PAS HIG-17-027



$$\mathcal{L}_{\mathrm{Yukawa,\,H}} = \frac{\boxed{m_{\mathrm{t}}}}{v} g_{\mathrm{Ht\bar{t}}}\, \mathrm{t\bar{t}H}$$



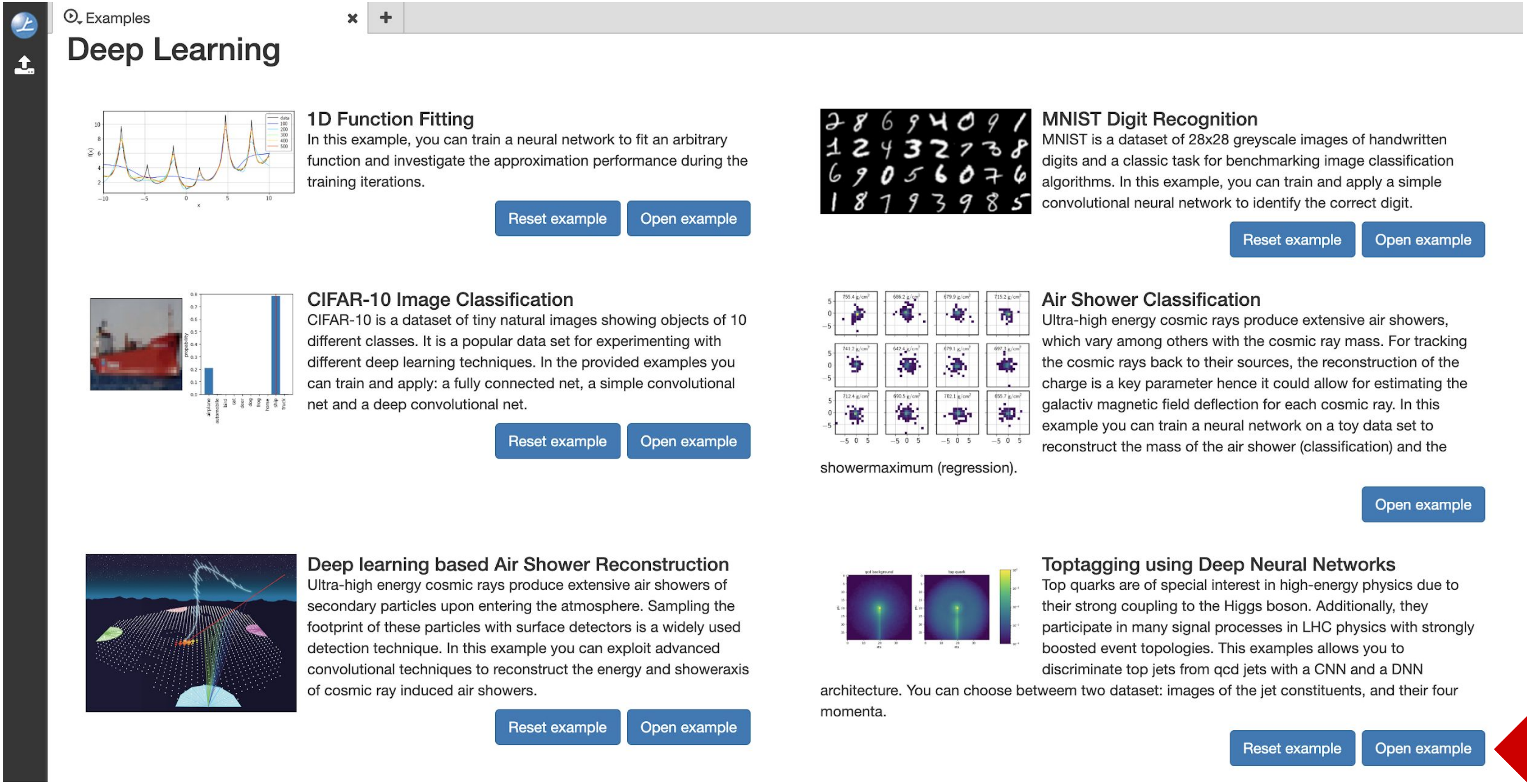
Top tagging at the LHC experiments with proton-proton collisions at √s = 13TeV

Toptagging in HEP: Dataset

- We have 2 classes: top jets and QCD jets (100000 jets per class)
- Each jet has 200 constituents (padded with zeros)
- Generated with pythia8, $\sqrt{s}$ = 14 TeV, no pileup
- Fatjets are reconstructed with anti-kt algorithm (delta R = 0.8)
- In each generated event only the leading jet is considered with:
  $550 < p_T < 650$ GeV and $|\eta| < 2$
- We prepared the dataset in 2 formats (numpy arrays):
  - eta, phi & four vector (E, px, py, pz) for each constituent in each jet,
    shape=(njets, 200, 6, 1)
  - Fatjet energy deposition as 2d-image in eta-phi plane with 40x40 pixel (already normalized to unity),
    shape=(njets, 40, 40, 1)

Link to dataset

Toptagging in HEP: Exercise



Deep Learning

1D Function Fitting

In this example, you can train a neural network to fit an arbitrary function and investigate the approximation performance during the training iterations.

MNIST Digit Recognition

MNIST is a dataset of 28x28 greyscale images of handwritten digits and a classic task for benchmarking image classification algorithms. In this example, you can train and apply a simple convolutional neural network to identify the correct digit.

CIFAR-10 Image Classification

CIFAR-10 is a dataset of tiny natural images showing objects of 10 different classes. It is a popular data set for experimenting with different deep learning techniques. In the provided examples you can train and apply: a fully connected net, a simple convolutional net and a deep convolutional net.

Air Shower Classification

Ultra-high energy cosmic rays produce extensive air showers, which vary among others with the cosmic ray mass. For tracking the cosmic rays back to their sources, the reconstruction of the charge is a key parameter hence it could allow for estimating the galactiv magnetic field deflection for each cosmic ray. In this example you can train a neural network on a toy data set to reconstruct the mass of the air shower (classification) and the showermaximum (regression).

Deep learning based Air Shower Reconstruction

Ultra-high energy cosmic rays produce extensive air showers of secondary particles upon entering the atmosphere. Sampling the footprint of these particles with surface detectors is a widely used detection technique. In this example you can exploit advanced convolutional techniques to reconstruct the energy and showeraxis of cosmic ray induced air showers.

Toptagging using Deep Neural Networks

Top quarks are of special interest in high-energy physics due to their strong coupling to the Higgs boson. Additionally, they participate in many signal processes in LHC physics with strongly boosted event topologies. This examples allows you to discriminate top jets from qcd jets with a CNN and a DNN architecture. You can choose betweem two dataset: images of the jet constituents, and their four momenta.

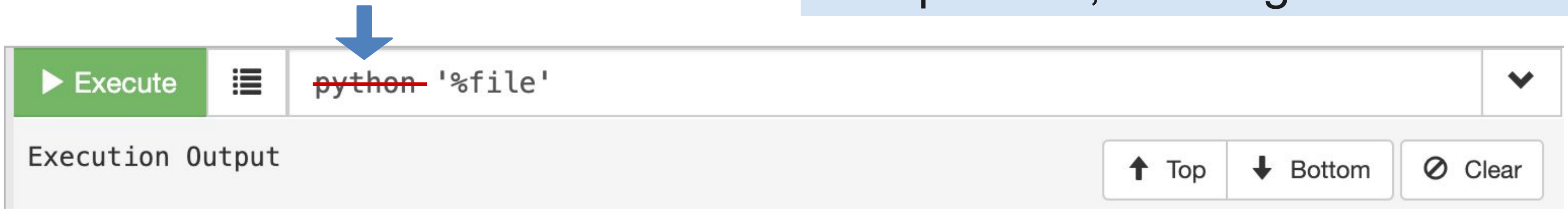
Toptagging in HEP: Exercise

- Two files:
  - jet_4vectors.py
  - jet_cnn.py
- Open results in dir:
  - /train-4vectors-XXX.X
  - /train-images-XXX.X

Model – add:
- Conv. layers and filters
- Pooling, Dense (FC) layers
- Regularization (after Flatten)

Model – modify:
- Batch size, epochs
- Kernel size, strides
- Optimizer, learning rate

pygpu



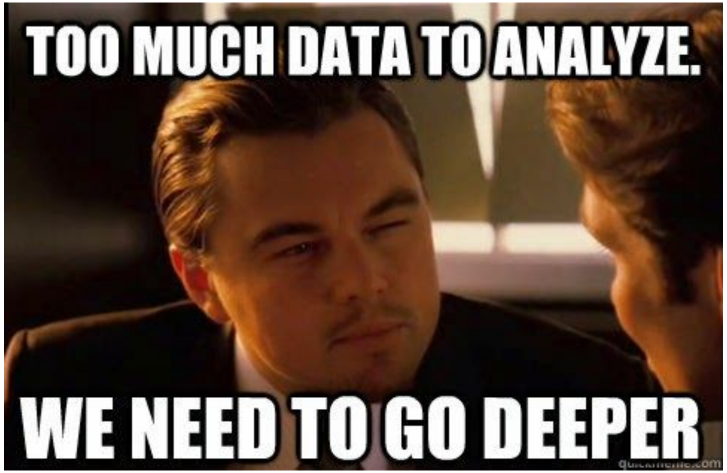
~~python~~ '%file'

Deep Learning in CMS (Software)

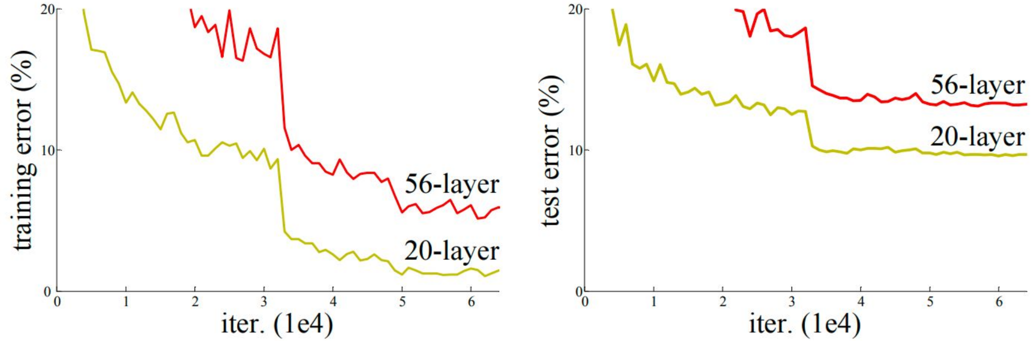
- Software in CMSSW 11X (https://github.com/cms-sw/cmsdist):
  - Many data-science tools: pandas, numpy, scikit-learn, scipy
  - Comes with tensorflow, theano, keras CMSSW > 94X
  - Evaluation of tf.graph in CMSSW: https://github.com/riga/CMSSW-DNN
- LCG software stacks (latest one “96”): http://lcginfo.cern.ch/
  - “source /cvmfs/sft.cern.ch/lcg/views/LCG_96/x86_64-centos7-gcc62-opt/setup.sh”
- One of the main problems: how to go from ROOT to numpy?
- 2 libraries:
  - uproot: https://github.com/scikit-hep/uproot (does not depend on ROOT!!!)
  - root_numpy: https://github.com/scikit-hep/root_numpy

Advanced Computer Vision Methods

- Deep Convolutional Networks
- Batch Normalization, Shortcuts
- Residual Networks, Inception



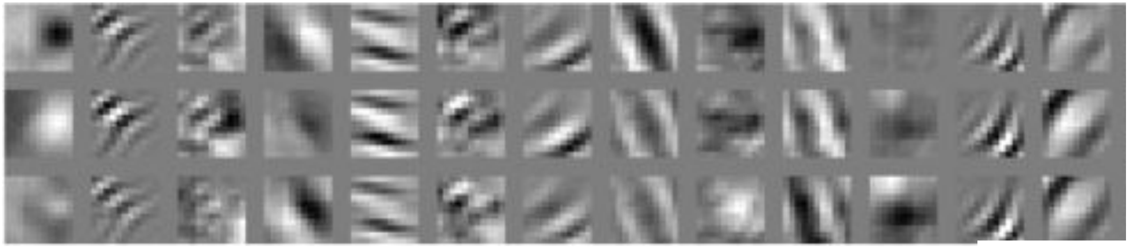
Obstacles when Going Deeper



- Neural networks should get monotonously better when adding more layers
- **Problems**
  - Convolutional filter show redundant behavior
  - Bad initialization
  - Internal covariate shift – need to constantly adapt changes in the earlier layer
  - Vanishing gradients – gradients become too small
  - Shattered gradients – gradients become white noise

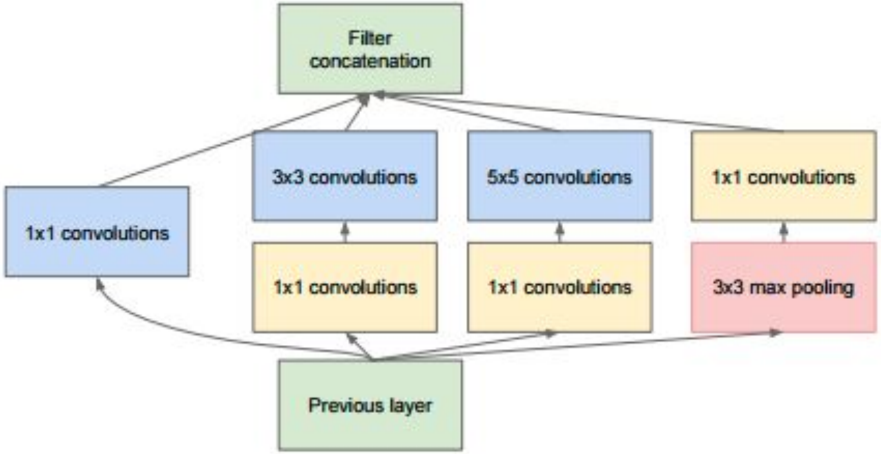
Inception Module

Key observation: Convolutional filters show redundant behavior



Idea: Factorize convolution operation

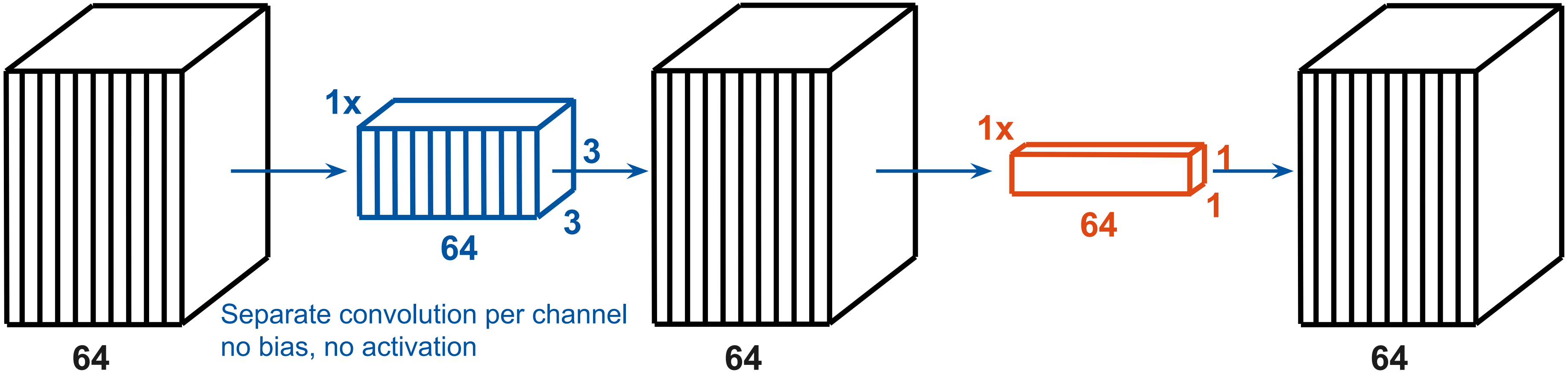
- Use different small convolutions in parallel and concatenate outputs
- Massive use of (1 x 1) convolutions



- Increase model complexity
- Make model sensitive to different scales

Xception ("Extreme Inception")

- **Idea:** If spatial correlations and cross-channel correlations are sufficiently decoupled it's better to compute them separately
- **Depthwise separable convolutions**
  - Perform depthwise separate convolution on each channel
  - Perform pointwise convolution (1 x 1) across channels



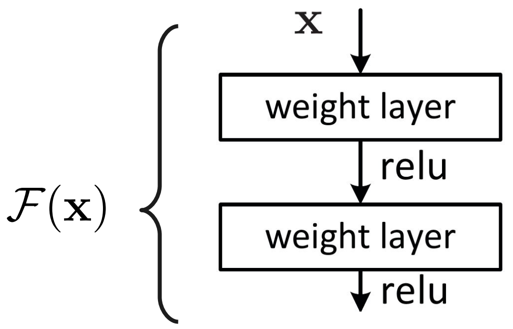
Separate convolution per channel
no bias, no activation

$$64(3 \cdot 3 \cdot 1) + 64(1 \cdot 1 \cdot 64 + 1) \approx 4,700$$

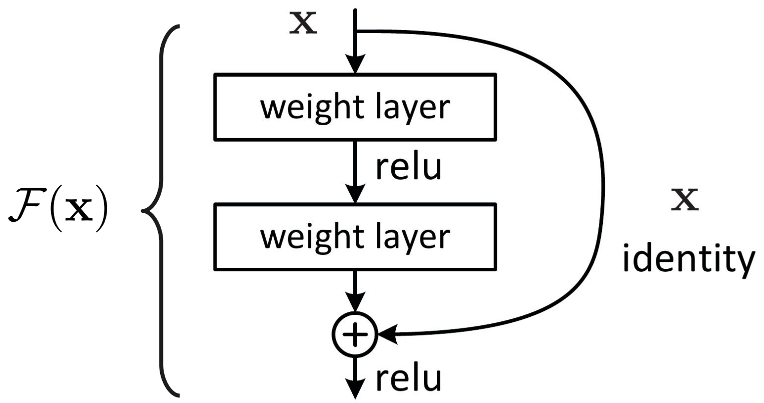
Standard convolution: $64(3 \cdot 3 \cdot 64 + 1) \approx 37,000$

Residual Unit

Idea: Residual unit consisting of small network and a shortcut (identity mapping)

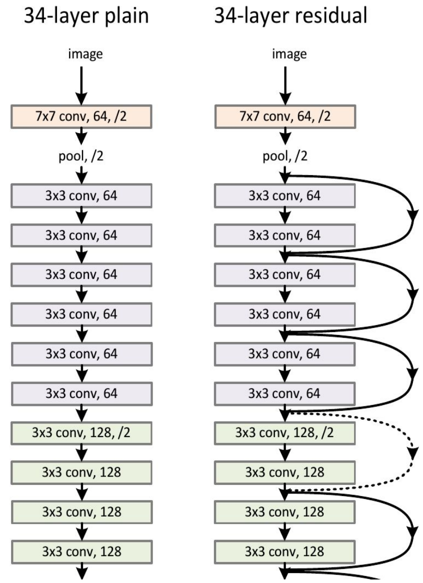


$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x})$$



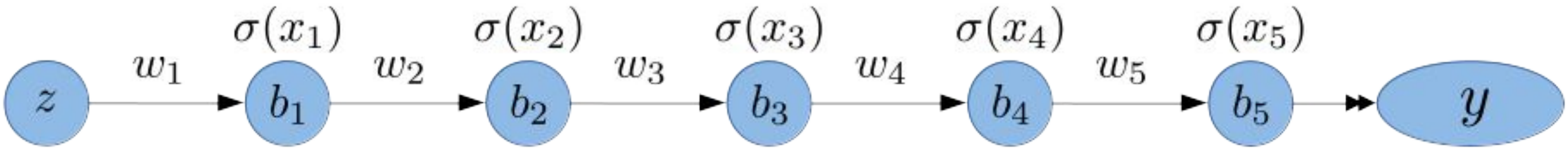
$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$$

- Weight block learns small residual $\mathcal{F}(\mathbf{x})$ on top of input $\mathbf{x}$
  - Output of residual unit $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$
- Shortcut let gradient propagate easily to earlier layers
- Later layers can easily turn weights to zero by $\mathcal{F}(\mathbf{x}) \to 0$



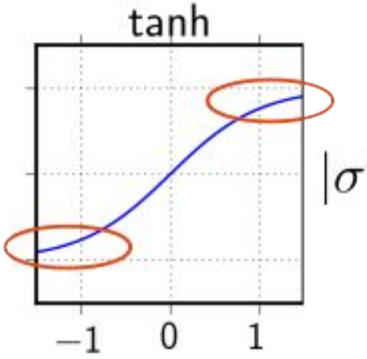
Up to several of hundreds layer deep!

Vanishing Gradient Problem



$$y = \sigma(x_5) = \sigma(w_5 \cdot \sigma(x_4) + b_5) = \sigma(w_5 \cdot \sigma(w_4 \cdot \sigma(x_3) + b_4) + b_5)....$$

$$\frac{\partial y}{\partial w_1} = \frac{\partial \sigma(x_5)}{\partial x_5}\frac{\partial x_5}{\partial \sigma(x_4)}\frac{\partial \sigma(x_4)}{\partial x_4}\frac{\partial x_4}{\partial w_1}.... = \underline{\sigma'(x_5)}w_5 \cdot \underline{\sigma'(x_4)}w_4.... \cdot \underline{\sigma'(x_1)}w_1$$



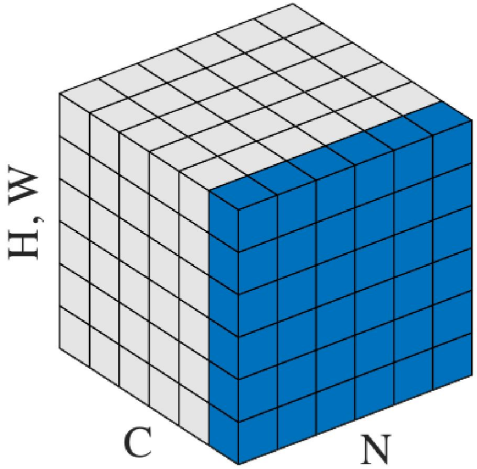
$$|\sigma'(x)| << 1$$

- Stacking many layers can lead to vanishing gradients
- Activation saturates
- Updates in early layers become very tiny
  - **No learning**
- Don't use sigmoids / tanh only rarely → use shortcuts

Batch Normalization

- Calculate batch-wise for each channel:
  - Mean: μ_B
  - Variance: σ_B^2
  - Add free parameters $\gamma,\ \beta$ to change scale and mean

→ $$y = \frac{x - \mu_B}{\sigma_B}\gamma + \beta$$



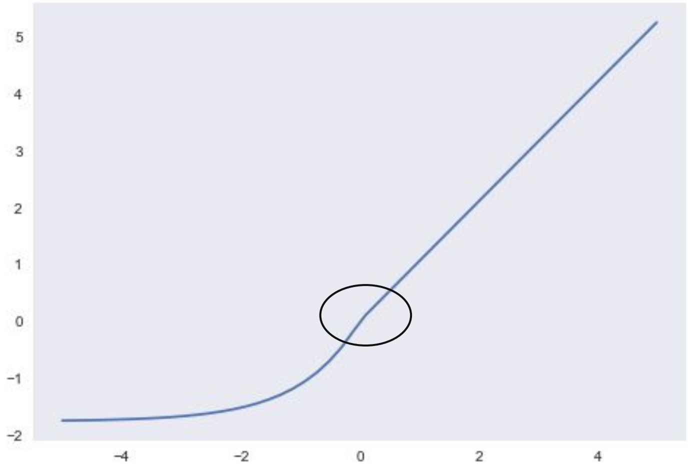
- Makes DNN robust against poor initializations
- Helps with vanishing gradient / less sensitive to high learning rates
- Has regularizing effect (no large weights, noise because of batch dependency)
- Reduce internal covariate shift

→ **Very successful for convolutional architectures**

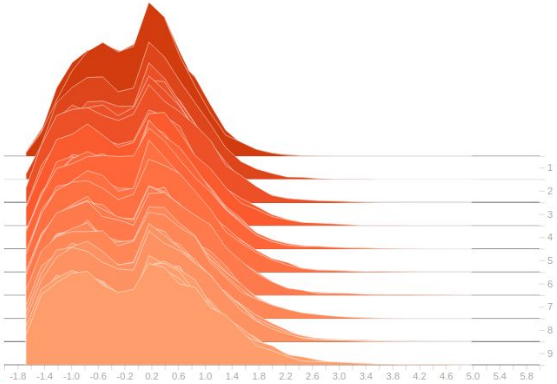
Self Normalizing Networks

- Batch normalization adds perturbations for training fully connected networks
- Use activation function which ensures standard normalized output: $\mu = 0,\ \sigma = 1$
- Stabilize the training
- Needs LeCun initialization & Alpha-dropout

▯ **Allow for very deep networks!**



$$\mathrm{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^{x} - \alpha & \text{if } x \leqslant 0 \end{cases}$$



Requirements:

- negative & positive values to control the mean
- Slope < 1 for damping the variance
- Slope > 1 to rise the variance

Normalization

Networks learn best when data is normalized

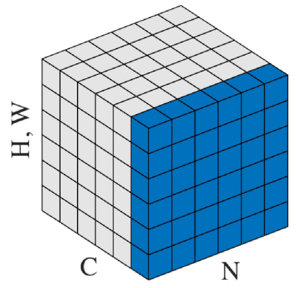
- Normalize data in between the layers

✓ Prevent very large activations (regularization) and very small gradients

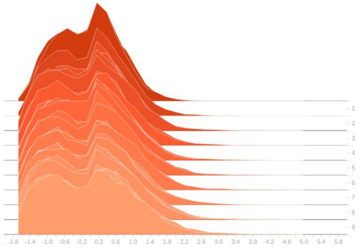
✓ Minimize covariate shift (perturbation due to simultaneous layer update)

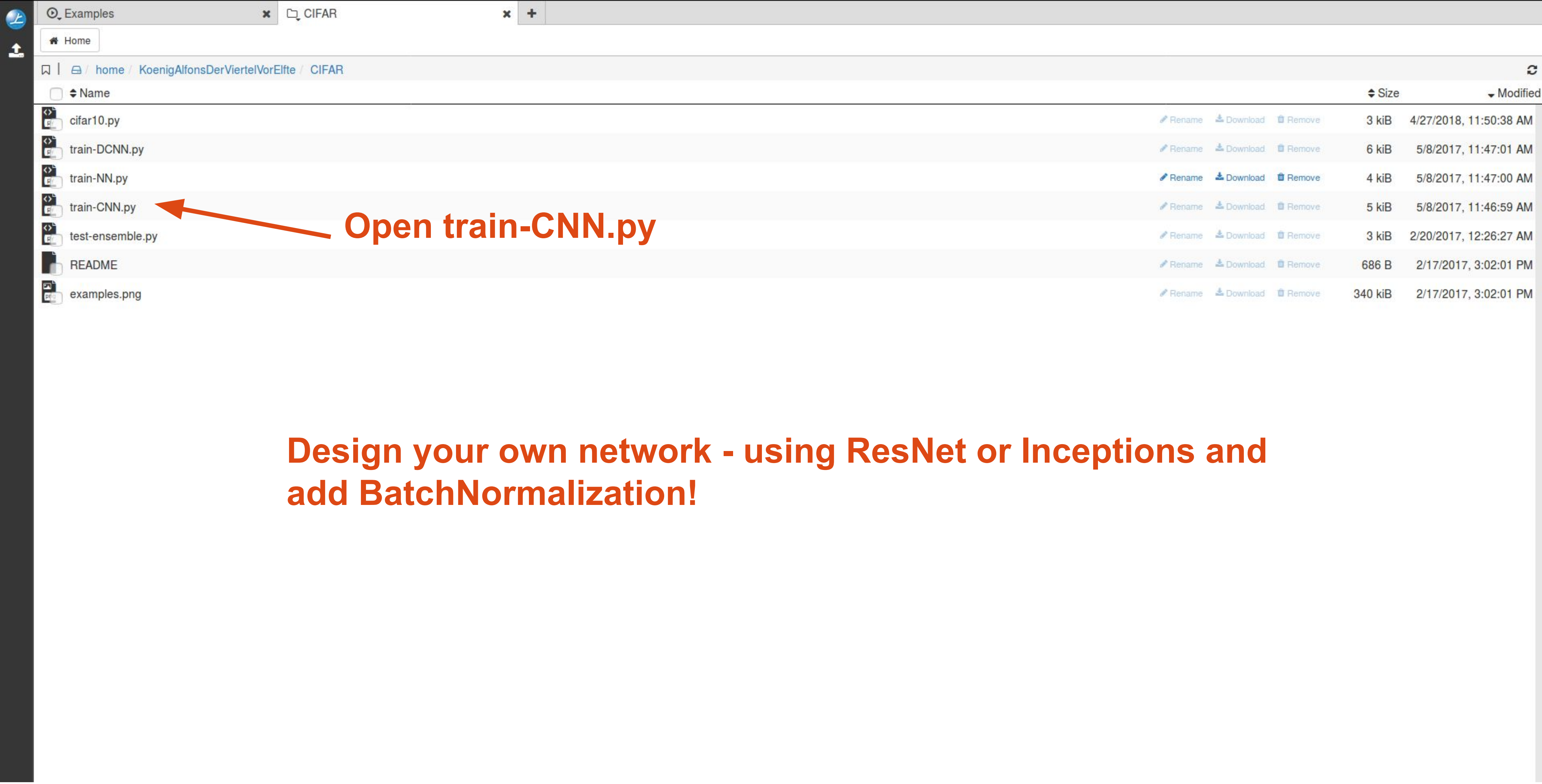
✓ Weights on same scale in each layer→ help against bad initializations

- Convolutional approach → Batchnormalization
  - Normalize feature activations over a batch if images



- Fully Connected approach → Selu
  - Use activation function which ensures standard normalized output





Name	Size	Modified
cifar10.py	3 kiB	4/27/2018, 11:50:38 AM
train-DCNN.py	6 kiB	5/8/2017, 11:47:01 AM
train-NN.py	4 kiB	5/8/2017, 11:47:00 AM
train-CNN.py	5 kiB	5/8/2017, 11:46:59 AM
test-ensemble.py	3 kiB	2/20/2017, 12:26:27 AM
README	686 B	2/17/2017, 3:02:01 PM
examples.png	340 kiB	2/17/2017, 3:02:01 PM

Open train-CNN.py

Design your own network - using ResNet or Inceptions and add BatchNormalization!

Code Examples – Advanced API

Inception Module

```python
from tensorflow import keras
layers = keras.layers

def inception_unit(x0):
  x1 = layers.Conv2D(16, (1, 1), padding='same', activation='relu')(x0)
  x1 = layers.Conv2D(16, (3, 3), padding='same', activation='relu')(x1)

  x2 = layers.Conv2D(16, (1, 1), padding='same', activation='relu')(x0)
  x2 = layers.Conv2D(16, (5, 5), padding='same', activation='relu')(x2)

  x3 = layers.Conv2D(16, (1, 1), padding='same', activation='relu')(x0)

  x4 = layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x0)
  x4 = layers.Conv2D(64, (1, 1), padding='same', activation='relu')(x4)
  return layers.concatenate([x1, x2, x3, x4], axis=-1)


x = ... # some tensor of shape say (n, nx, ny, 64)
x = inception_unit(x)
x = inception_unit(x)
```

Residual Module

```python
from tensorflow import keras
layers = keras.layers

def residual_unit(x0):
  x = layers.Conv2D(64, (1, 1), padding="same")(x0)

  x = layers.Activation("relu")(x)
  x = layers.Conv2D(64, (3, 3), padding="same")(x)

  return layers.add([x, x0])


x = ... # some tensor of shape say (n, nx, ny, 64)
x = residual_unit(x)
x = residual_unit(x)
```