CSCI 566: Deep Learning and its Applications

Yue Zhao
Thomas Lord Department of Computer Science
University of Southern California

Credits to previous versions of USC CSCI566, Stanford CS 229, 231n

Teaching Team

TAs and CPs:

Logistics

Take a screenshot then – location and time

Logistics

Take a screenshot then – website and communication

Please use Piazza for any course-related communication
piazza.com/usc/spring2024/csci566

Any non-necessary e-mail will be ignored

Logistics

Take a screenshot then – Exams and Quizzes

Logistics

Take a screenshot then – project

Logistics

Take a screenshot then – Grading, makeup, and late days

Logistics

Take a screenshot then – Course Schedule

Logistics

Take a screenshot then – Books

Important Interview Questions

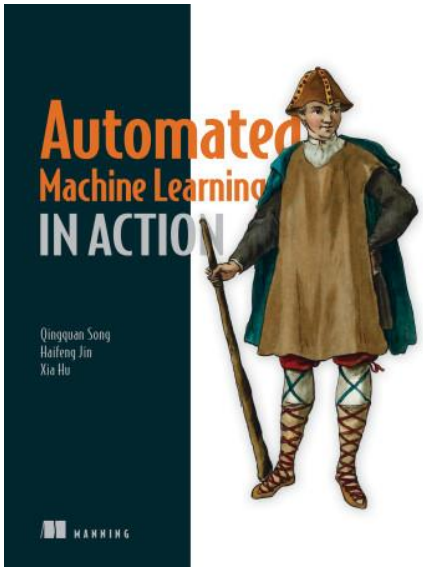
Question: Can

Answer: Overfitting.

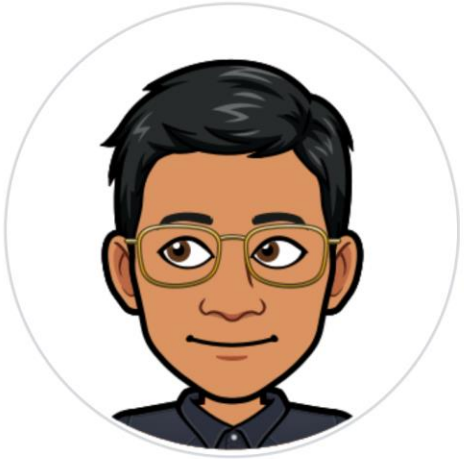
How to Start Building ML Open-Source?

Guest lecture & Presenter: Dr. Haifeng Jin

March 1st



Haifeng Jin is a software engineer on the Keras team at Google. He is the creator of **AutoKeras**, coauthor of **Keras Tuner**, and a contributor to **Keras** and **TensorFlow**. Haifeng got his Ph.D. in computer science at Texas A&M University. His research interest is automated machine learning (AutoML).



Haifeng Jin

haifeng-jin · he/him

Unfollow

Software engineer on Keras | Project lead of AutoKeras & KerasTuner

690 followers · 143 following

Followed by yifanjiang19

@keras-team

https://haifengjin.com/about

Pinned

keras-team/autokeras Public

AutoML library for deep learning

Python 9k 1.4k

keras-team/keras-tuner Public

A Hyperparameter Tuning Library for Keras

Python 2.8k 385

datamllab/automl-in-action-notebooks Public

Jupyter notebooks for the code samples of the book "Automated Machine Learning in Action"

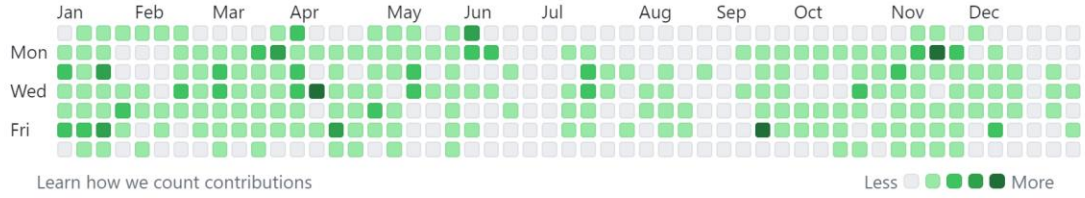
Jupyter Notebook 77 37

keras-team/keras Public

Deep Learning for humans

Python 60.2k 19.5k

956 contributions in the last year



Learn how we count contributions

Less More

2024

2023

2022

2021

2020

2019

2018

2017

@keras-team @tensorflow @conda-forge More

Activity overview

Contributed to keras-team/keras-tuner, keras-team/keras, keras-team/keras-core

15%
Code review

Recap

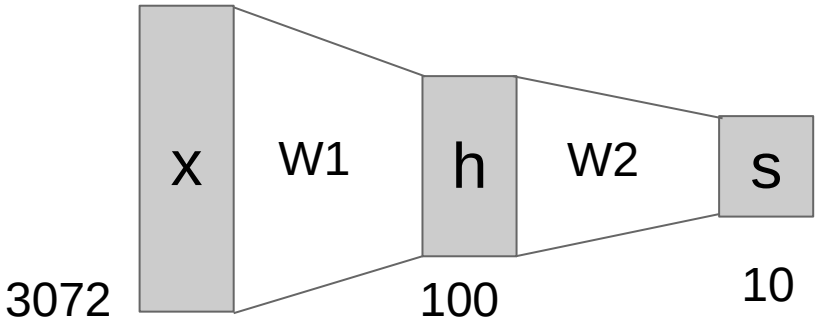
Neural Networks

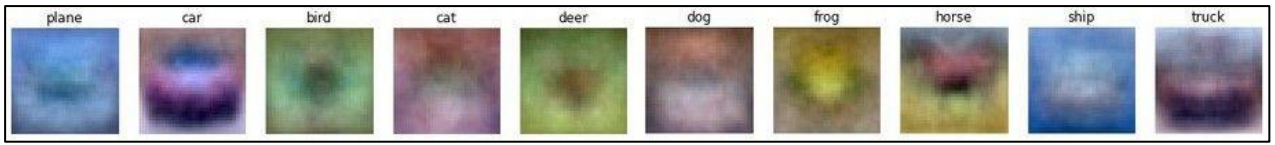
Linear score function:

$$f = Wx$$

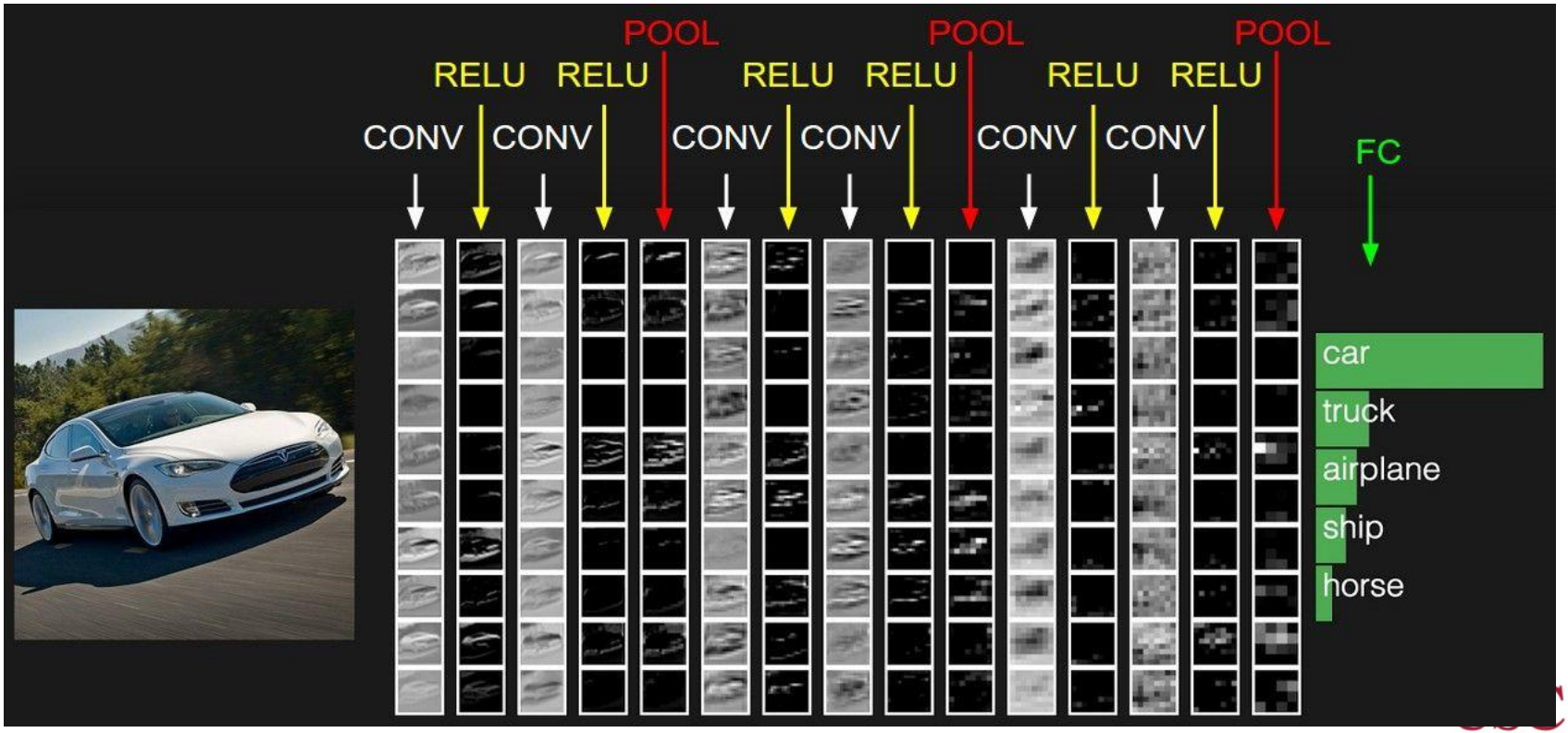
2-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$

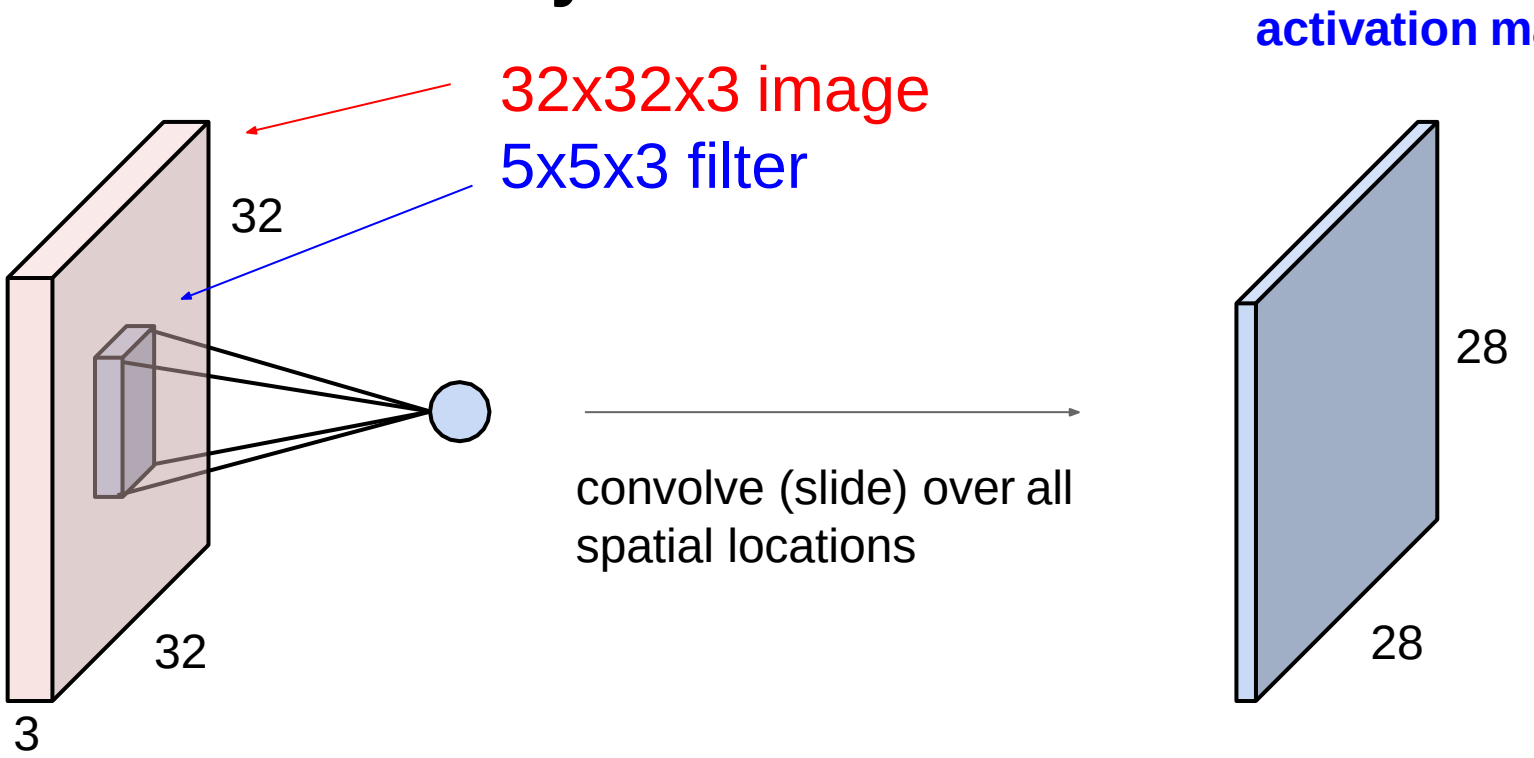




Convolutional Neural Networks

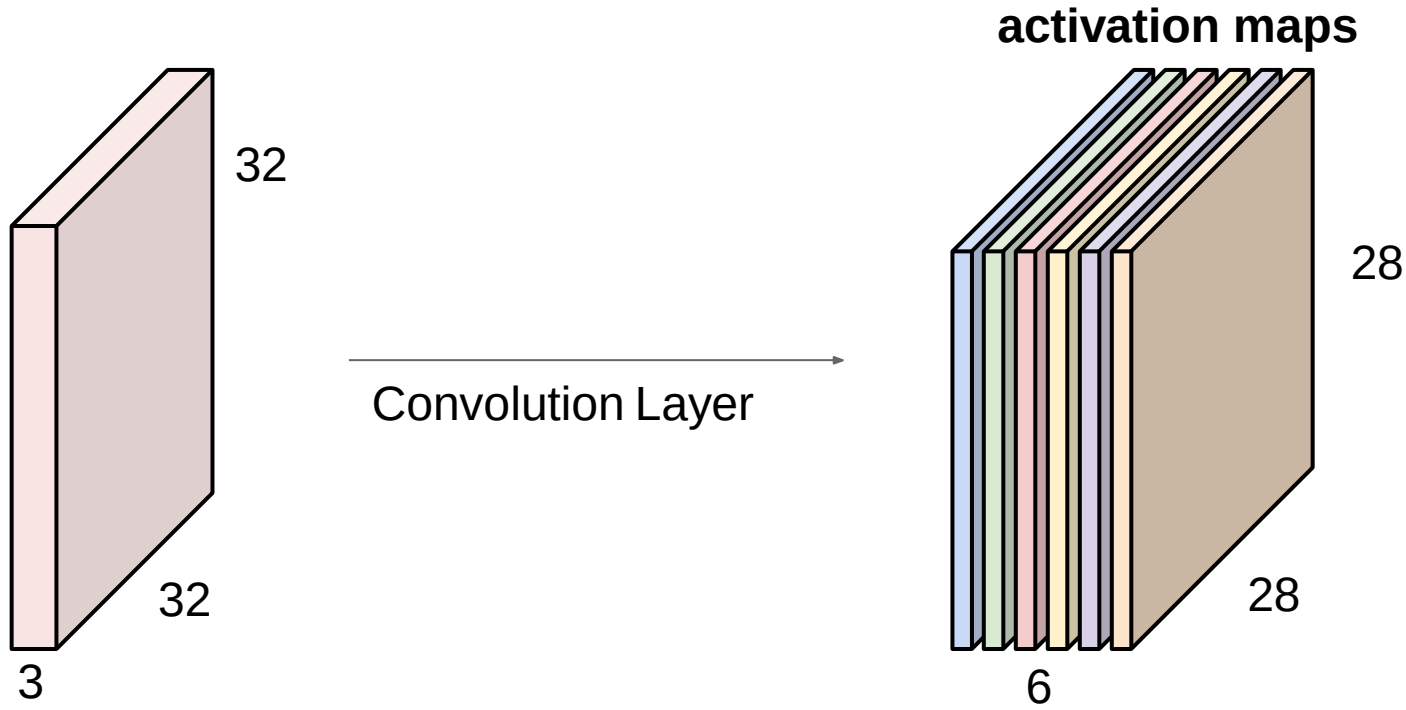


Convolutional Layer



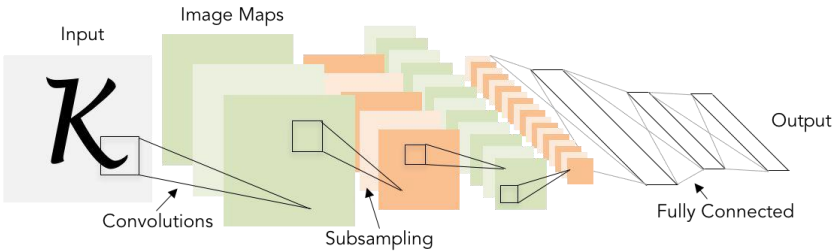
Convolutional Layer

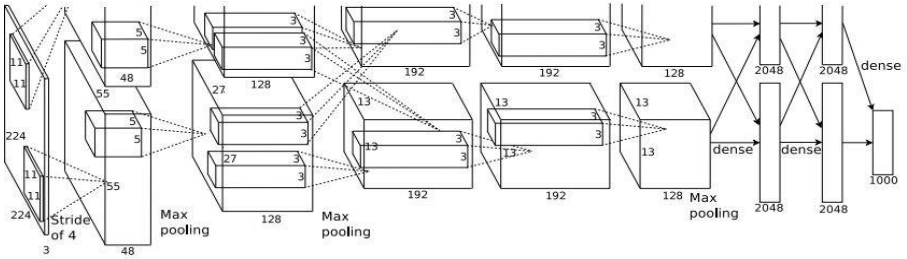
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

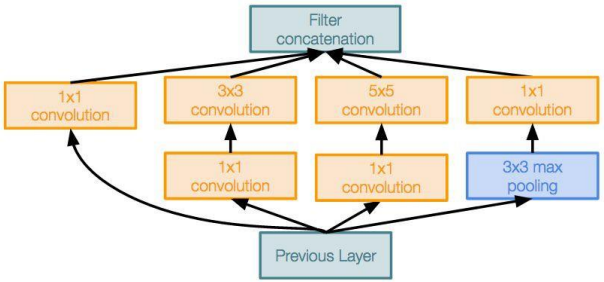


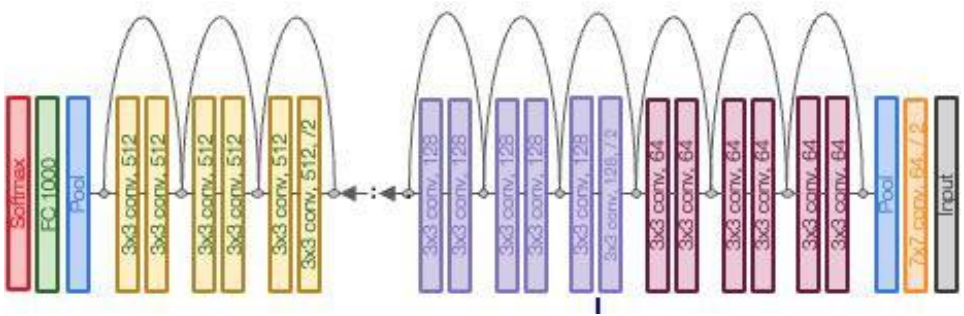
We stack these up to get a "new image" of size 28x28x6!

CNN Architectures

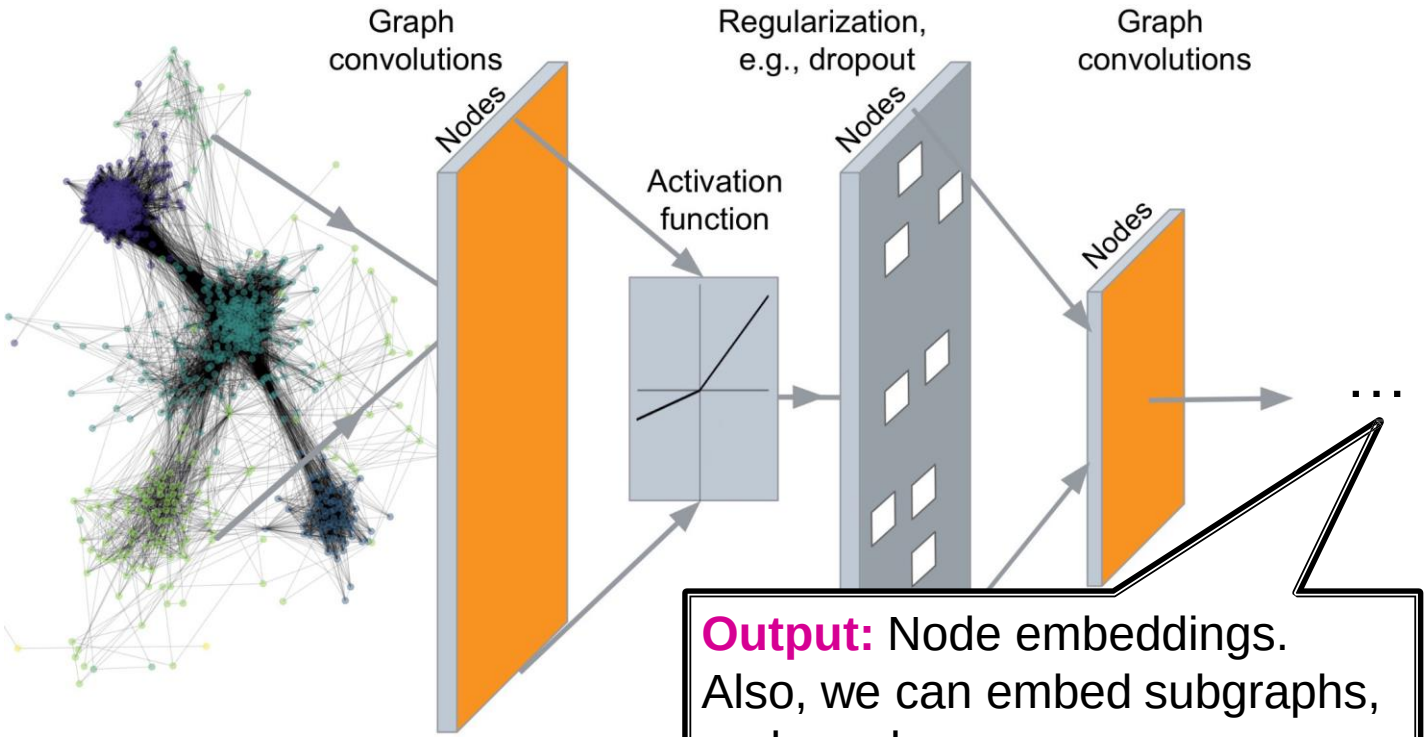








Deep Graph Encoders



Output: Node embeddings. Also, we can embed subgraphs, and graphs

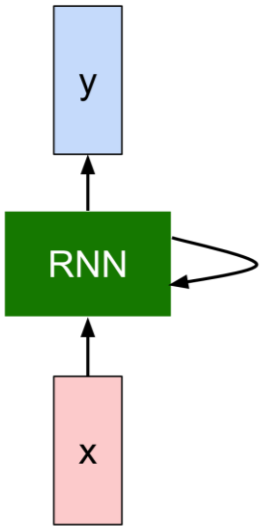
(Vanilla) Recurrent Neural Network

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

$$\downarrow$$

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

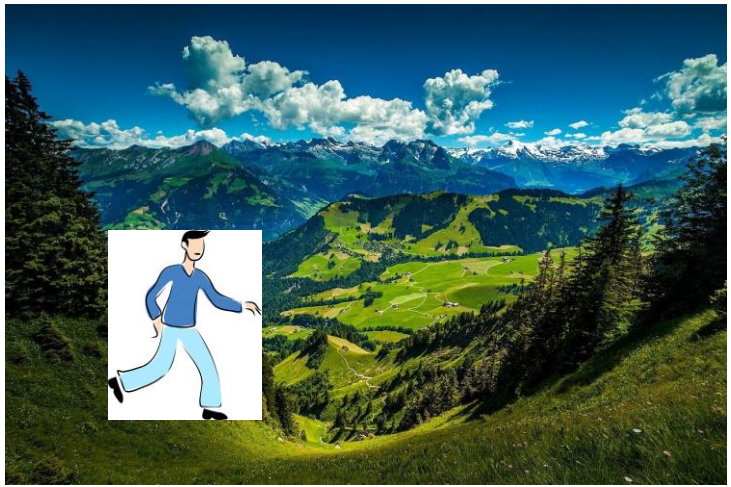
$$y_t = W_{hy}h_t$$

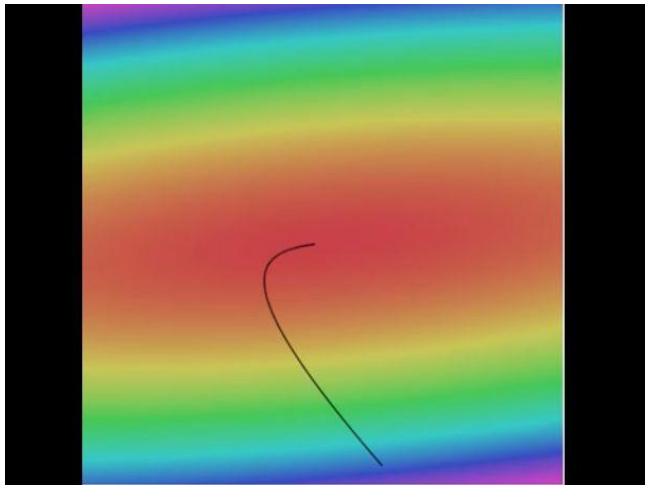


The state has a single “hidden” vector **h**

Slide credit: Stanford CS231n

Learning network parameters through optimization





```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Training Neural Networks

Basics

Data preprocessing

- Zero-center data
- Normalize data
- PCA Whitening

Data preprocessing

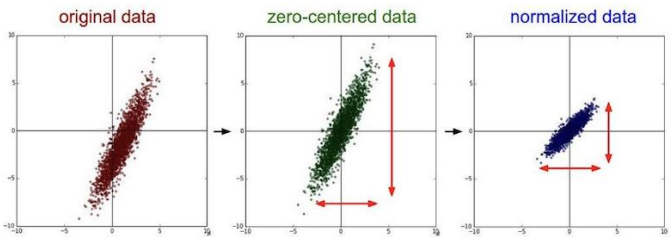
Why do we need preprocessing?

- Zero-center data
- Normalize data
- PCA Whitening

Data preprocessing

Why do we need preprocessing?

- Zero-center data
- Normalize data
- PCA Whitening



Activation functions (ReLU) works around **zero**.

If input data is biased toward positive or negative, randomly initialized layers produce biased output.

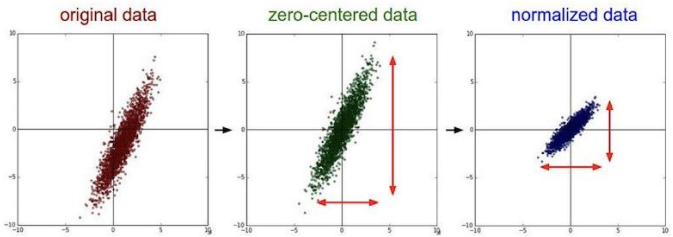
Figure from Stanford cs231n lecture slides

Data preprocessing

Why do we need preprocessing?

- Zero-center data
- Normalize data
- PCA Whitening

A feature with small scale has a negligible effect on backprop.



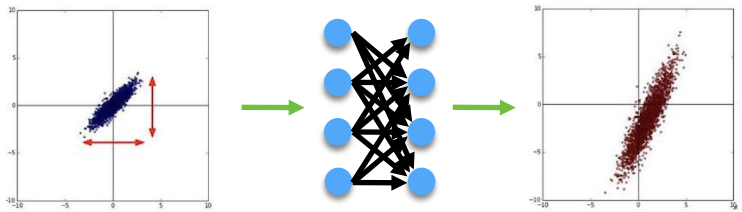

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Figure from Stanford cs231n lecture slides

Normalization

- After each layer, the distribution of activation signals changes (Internal Covariate Shift)

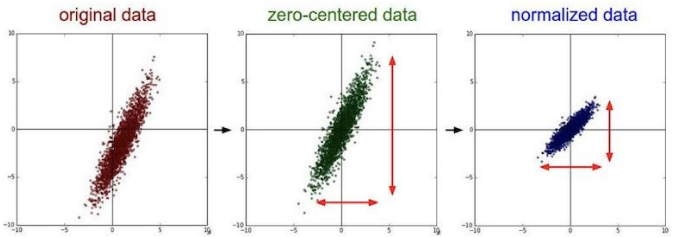


- As a network becomes deeper, distribution shifts more.
- Recall why we need data preprocessing

Figure from Stanford cs231n lecture slides

Normalization

- Normalize the distribution of activations to have zero mean and unit variance



Same as data preprocessing

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

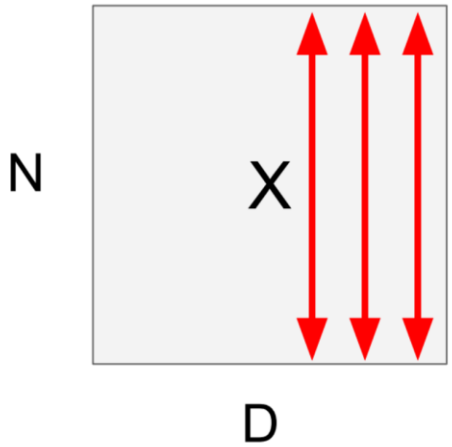
Figure from Stanford cs231n lecture slides

Batch normalization

- Why do we need normalization?
  - Internal Covariate Shift
  - To train deeper networks on larger data
- Make each dimension Unit Gaussian

Batch normalization

- Compute the empirical mean and variance independently for each dimension and normalize.



Batch normalization

- Vanilla Normalization:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

[Ioffe and Szegedy, 2015]

Batch normalization

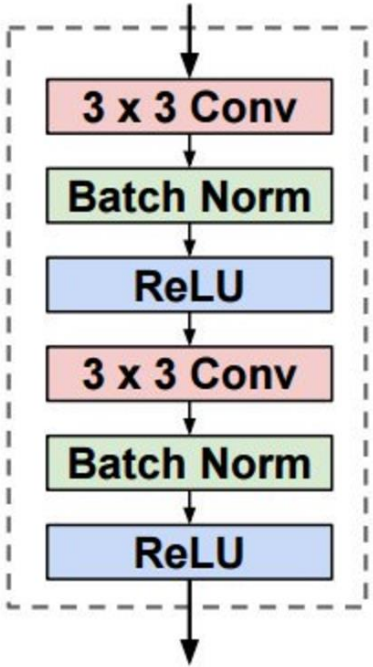
- Why is it good?
  - Improves gradient flow
  - Allows higher learning rates
  - Reduces strong dependence on initialization
  - Regularization

Batch normalization

- At test time
  - The mean and variance are not computed based on the batch.
  - Empirical mean of activations during training is used.

Batch normalization

- Usually added after convolutional layers or fully connected layers
- before non-linearities

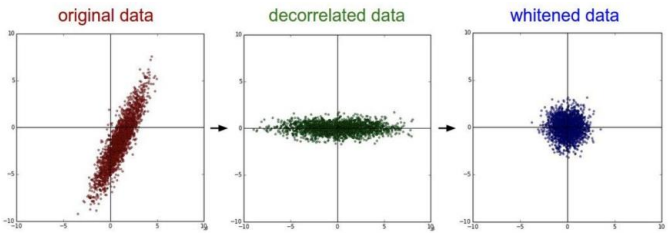


Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." *European Conference on Computer Vision*. Springer International Publishing, 2016.

Data preprocessing

Why do we need preprocessing?

- Zero-center data
- Normalize data
- PCA Whitening



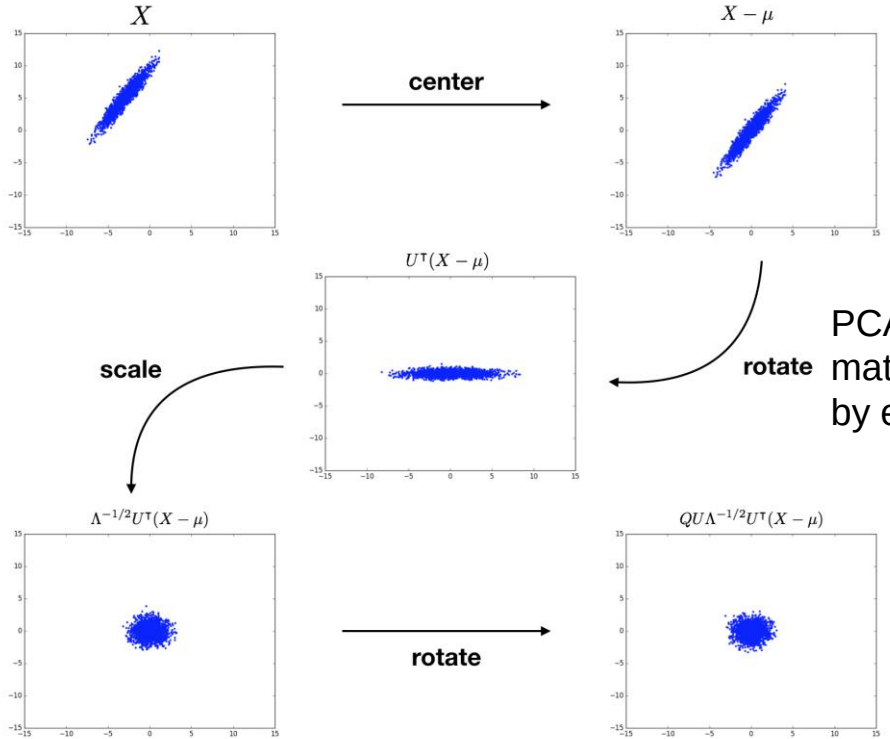
Remove correlation between input features

Figure from Stanford cs231n lecture slides

Data preprocessing

Whitening transforms the data into a space where the features are uncorrelated, and all have the same variance

Whitening: inverse square root of the eigenvalue matrix. normalizes the variance across all dimensions



Not common to use PCA whitening

TLDR: In practice for Images: center only

e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
  (mean along each channel = 3 numbers)
- Subtract per-channel mean and
  Divide by per-channel std (e.g. ResNet)
  (mean along each channel = 3 numbers)

Not common to do PCA or whitening

Important Interview Questions

Question: How would you handle missing data in a dataset?

Answer: Missing data can be handled in several ways:

1. **Imputation**: Replace missing values with a statistic like mean, median, or mode, or use a model to predict missing values.
2. **Removal**: If the missing values are not random or comprise a large portion of the data, it might be better to remove those points.
3. **Indicator Variable**: For some algorithms, it can be useful to create a new variable indicating the presence of missing data along with imputation.

Important Interview Questions

Question: How do you ensure that preprocessing steps applied to the training data are also correctly applied to the test data?

Answer: To ensure consistency, preprocessing steps should be defined and fitted on the training data, then applied to the test data.

For example, if scaling is applied, the scale parameters (like mean and standard deviation for standardization) should be derived from the training data and then used to scale the test data. This prevents data leakage and ensures that the model's performance is evaluated on the same terms as it was trained.

Important Interview Questions

The `preprocessing` module provides the `StandardScaler` utility class, which is a quick and easy way to perform the following operation on an array-like dataset:

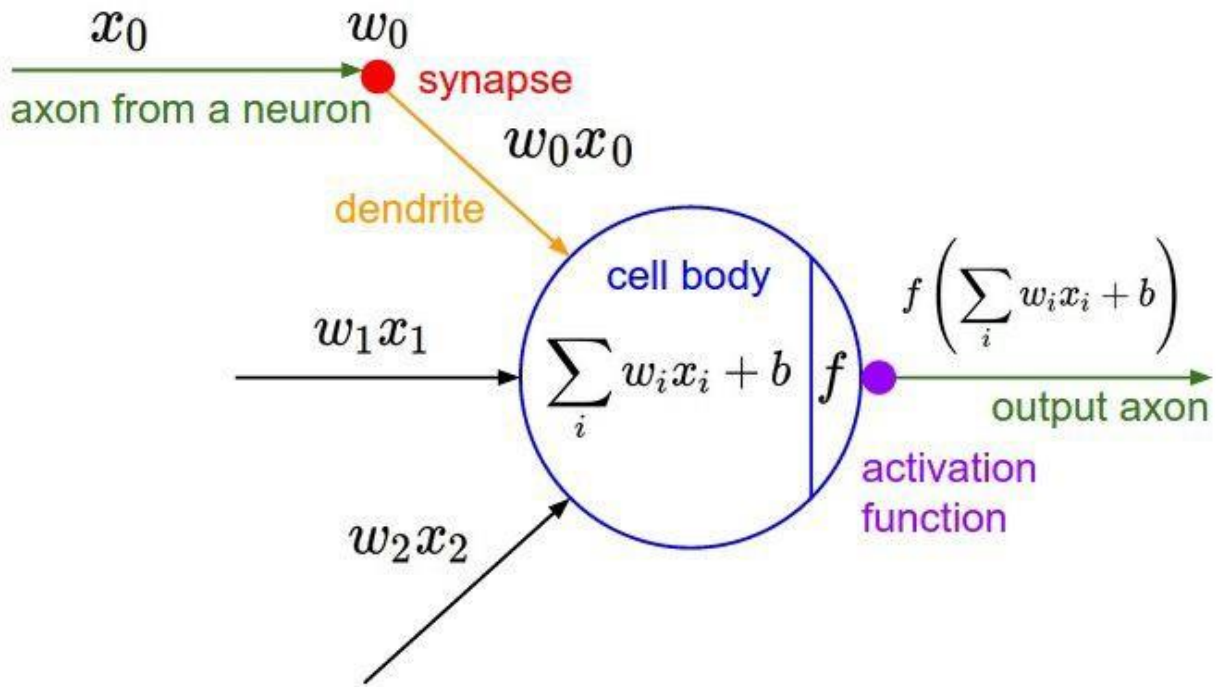
```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1.,  1.]])
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> scaler
StandardScaler()

>>> scaler.mean_
array([1. ..., 0. ..., 0.33...])

>>> scaler.scale_
array([0.81..., 0.81..., 1.24...])

>>> X_scaled = scaler.transform(X_train)
>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Scaled data has zero mean and unit variance:

Training Neural Networks

Activation Functions (Recap Note 3)

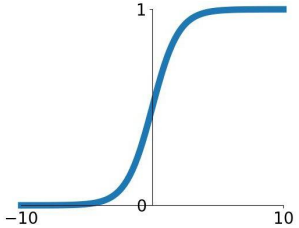
Activation Functions



Activation Functions

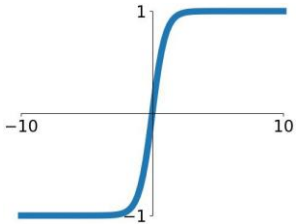
Sigmoid

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



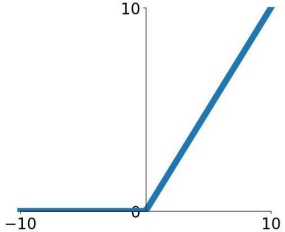
tanh

$$\tanh(x)$$



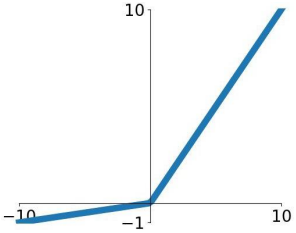
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

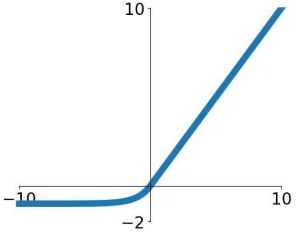


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

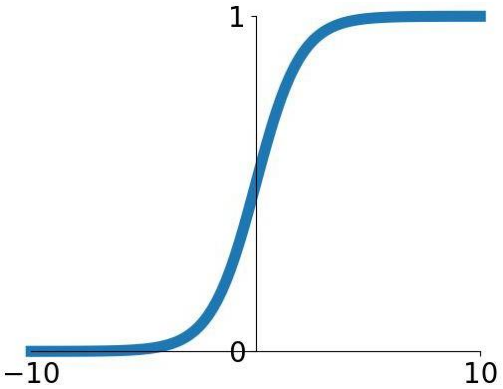
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



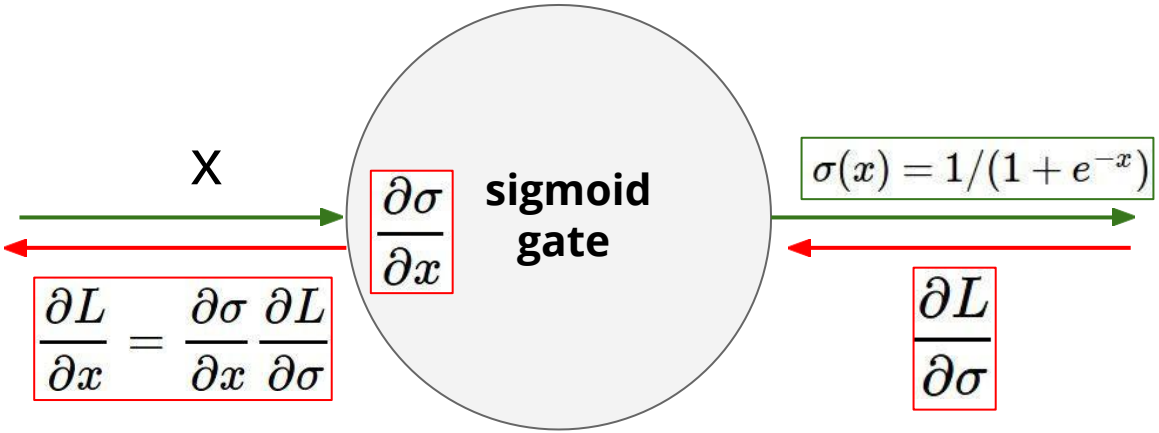
Activation Functions

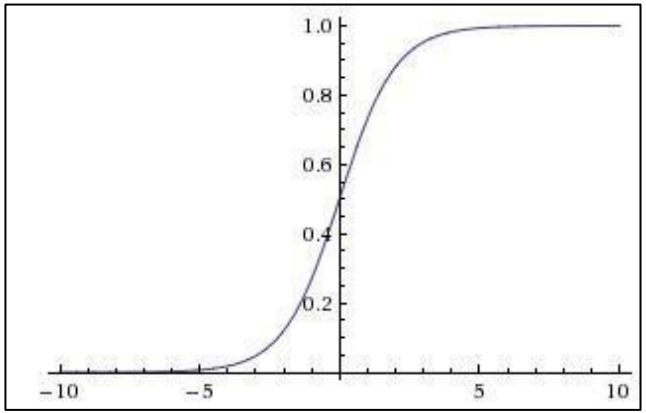
$$\sigma(x) = 1/(1 + e^{-x})$$



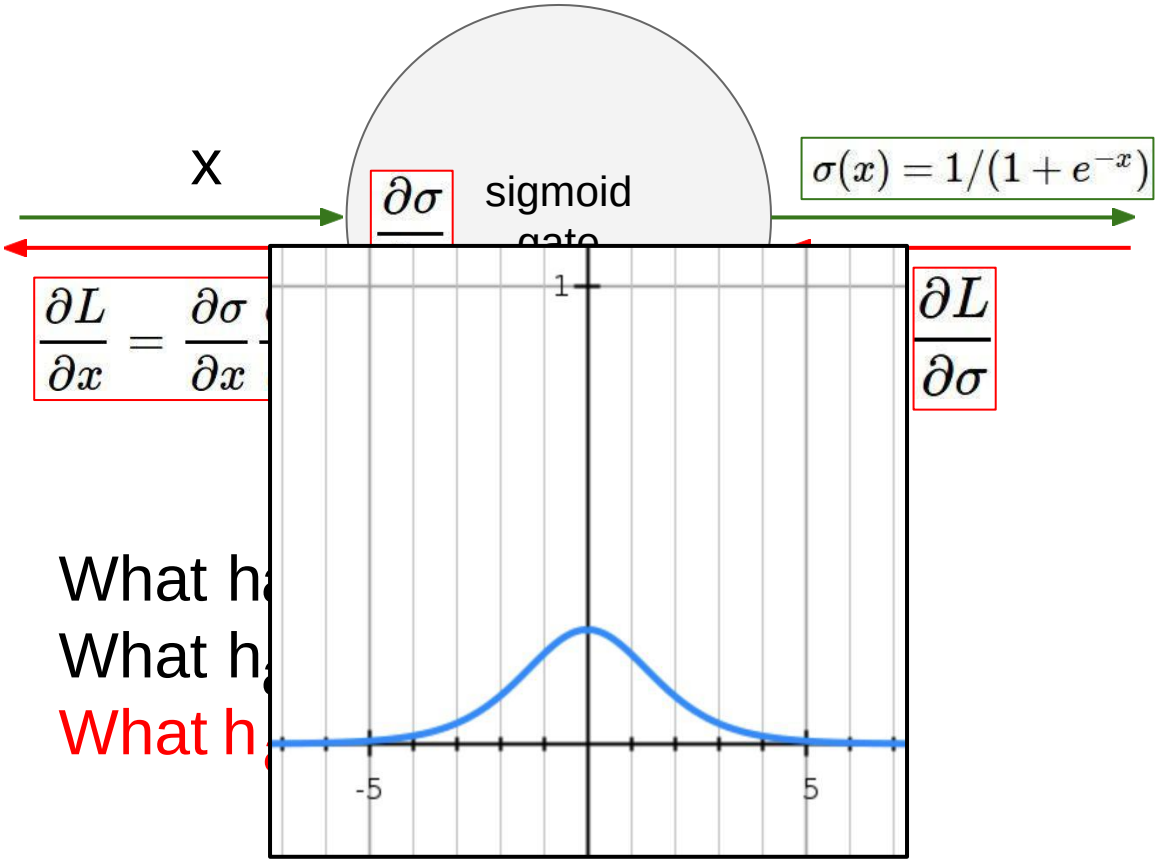
Sigmoid

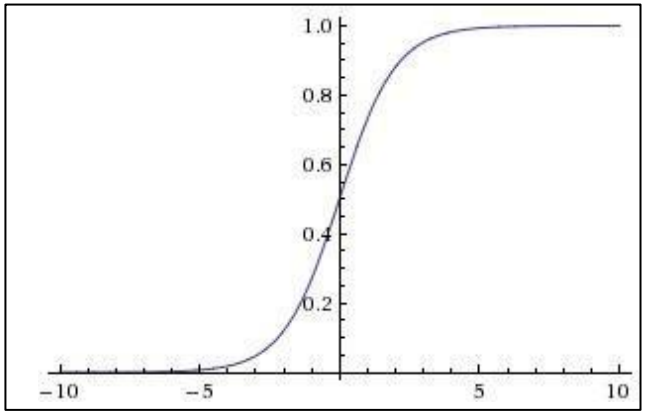
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron





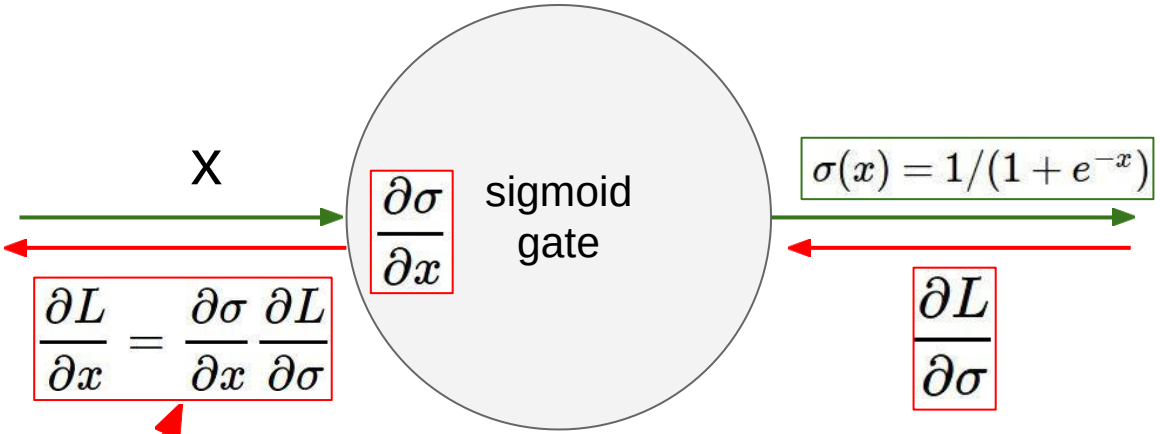
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$

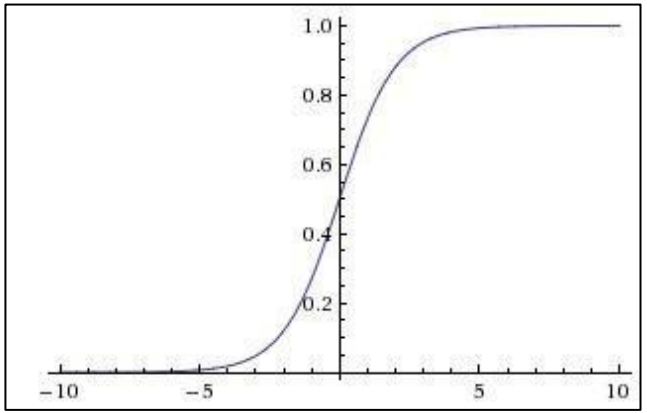




What h
What h
What h

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$





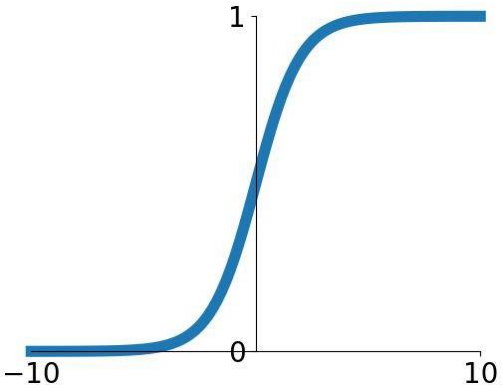
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$

Why is this a problem?

If all the gradients flowing back will be zero and weights will never change

Activation Functions

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



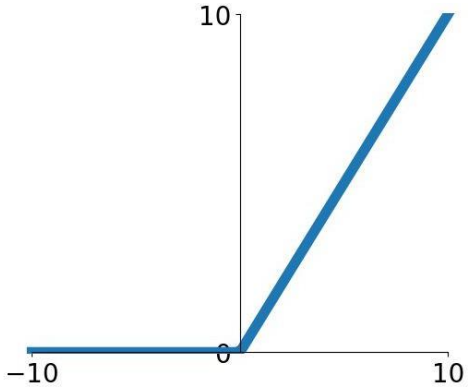
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

Activation Functions



ReLU
(Rectified Linear Unit)

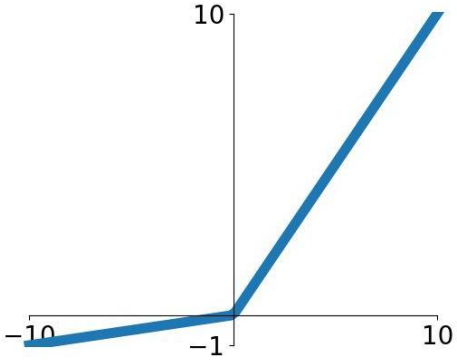
- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

[Krizhevsky et al., 2012]

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



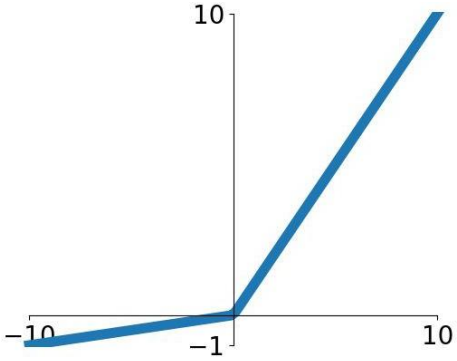
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

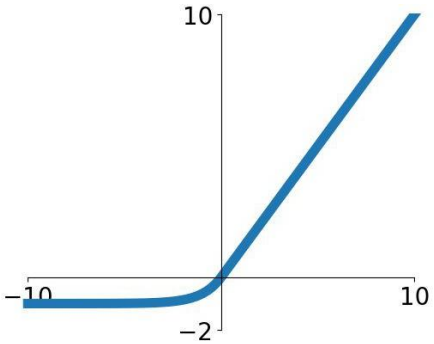
$$f(x) = \max(\alpha x, x)$$

backprop into α (parameter)

[Clevert et al., 2015]

Activation Functions

Exponential Linear Units (ELU)



- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- Computation requires exp()

(Alpha default = 1)

TLDR: In practice:

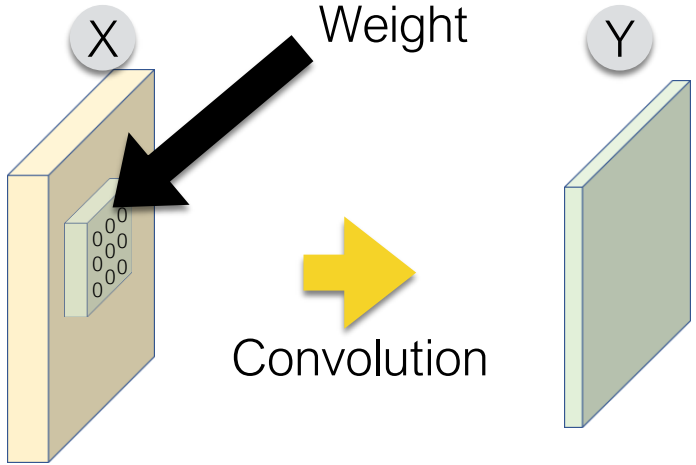
- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
  - To squeeze out some marginal gains
- Don’t use sigmoid or tanh

Training Neural Networks

Weight W Initialization

Weight initialization

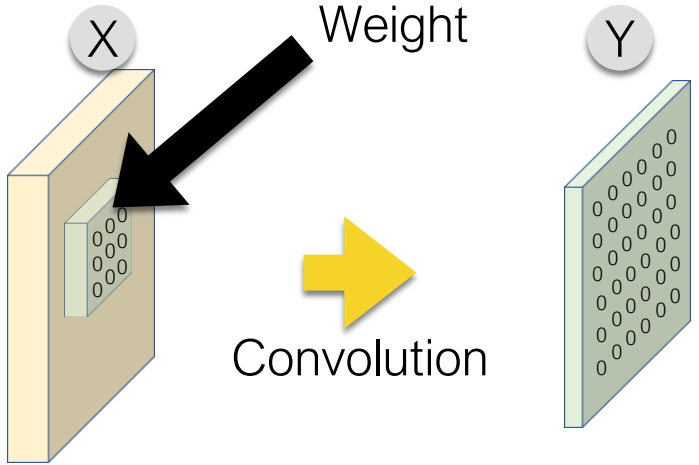
If all weights are 0?



Slide credit: CS 231N

Weight initialization

If all weights are 0?



Gradient vanishing problem

Output **Y** will be always 0 and then gradients will go to 0.

Slide credit: CS 231N

Weight initialization

- Small random numbers
- Xavier
- Xavier / 2 (He et. al.)

Slide credit: CS 231N

Small Random Numbers

- Sample from a gaussian distribution with 0 mean and given standard deviation (such as 0.01)

$$W \sim \mathcal{N}(\mu, \sigma^2)$$

- Doesn't work for deeper networks

```
W = 0.01 * np.random.randn(Din, Dout)
```

Weight Initialization: Activation statistics

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

What will happen to the activations for the last layer?

Slide credit: CS 231N

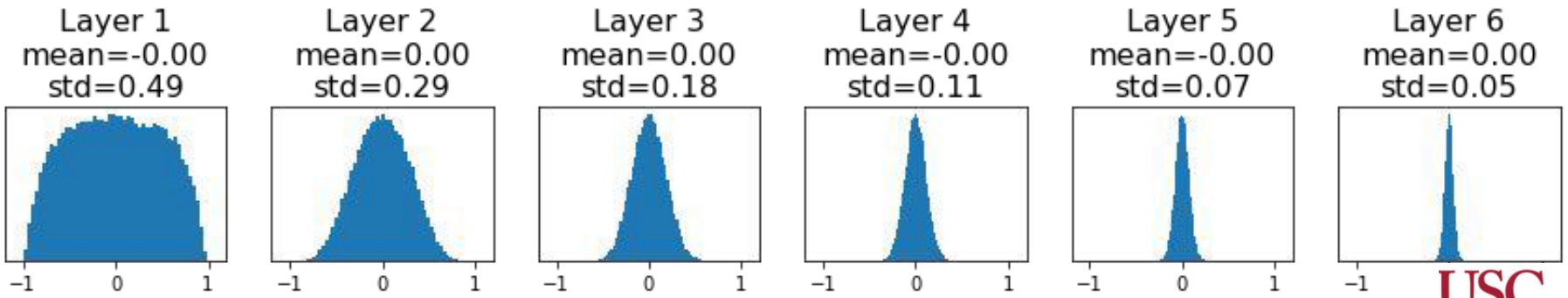
Weight Initialization: Activation statistics

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?



Slide credit: CS 231N

Weight Initialization: Activation statistics

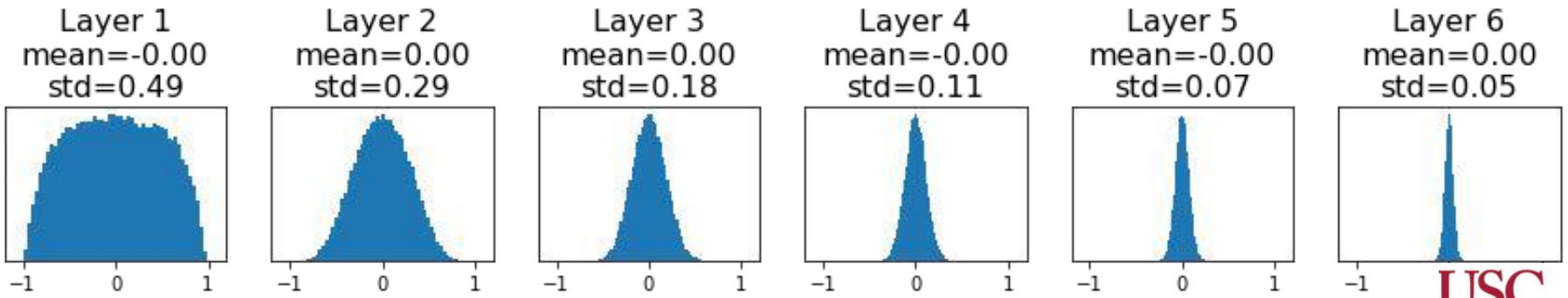
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?

A: All zero, no learning =(



Slide credit: CS 231N

Xavier Initialization

- Set the variance according to # of input neurons
- Intuition: input and output with the same variance to prevent vanishing or exploding problem

$$W \sim \mathcal{N}(\mu, \sigma^2) \longleftarrow Var(W) = \frac{1}{n_{in}}$$

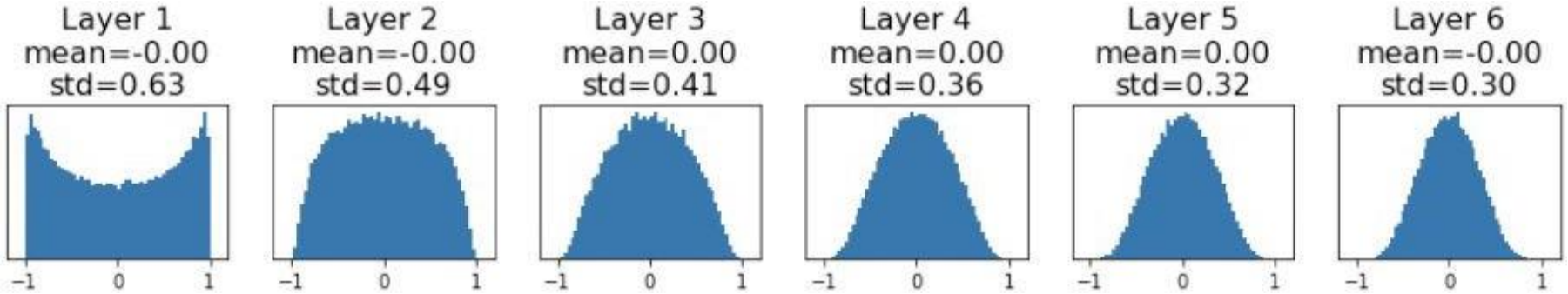
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Weight Initialization: “Xavier” Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

“Just right”: Activations are nicely scaled for all layers!



Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

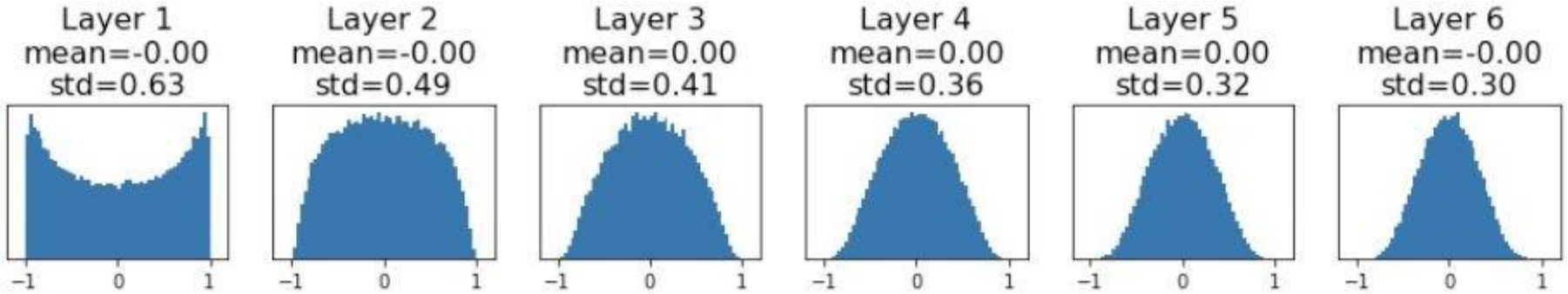
Weight Initialization: “Xavier” Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

“Just right”: Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels



Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels

Let: $y = x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din}$

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels

Let: $y = x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din}$

Assume: $Var(x_1) = Var(x_2)= \ldots=Var(x_{Din})$

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels

Let: $y = x_1w_1+x_2w_2+...+x_{Din}w_{Din}$

Assume: $Var(x_1) = Var(x_2)= \ldots=Var(x_{Din})$

We want: $Var(y) = Var(x_i)$

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels

Let: $y = x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din}$

Assume: $Var(x_1) = Var(x_2)= \ldots=Var(x_{Din})$

We want: $Var(y) = Var(x_i)$

$Var(y) = Var(x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din})$
[substituting value of y]

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels

Let: $y = x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din}$

Assume: $Var(x_1) = Var(x_2)= \ldots=Var(x_{Din})$

We want: $Var(y) = Var(x_i)$

$Var(y) = Var(x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din})$

$= Din\ Var(x_iw_i)$

[Assume all x_i, w_i are iid]

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels

Let: $y = x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din}$

Assume: $Var(x_1) = Var(x_2)= \ldots=Var(x_{Din})$

We want: $Var(y) = Var(x_i)$

$$Var(y) = Var(x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din})$$
$$= Din\ Var(x_iw_i)$$
$$= Din\ Var(x_i)\ Var(w_i)$$

[Assume all x_i, w_i are zero mean]

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Weight Initialization: "Xavier" Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2 * input_channels

Let: $y = x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din}$

Assume: $Var(x_1) = Var(x_2)= \ldots=Var(x_{Din})$

We want: $Var(y) = Var(x_i)$

$$Var(y) = Var(x_1w_1+x_2w_2+\ldots+x_{Din}w_{Din})$$
$$= Din\ Var(x_iw_i)$$
$$= Din\ Var(x_i)\ Var(w_i)$$

[Assume all x_i, w_i are iid]

So, $Var(y) = Var(x_i)$ only when $Var(w_i) = 1/Din$

Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Weight Initialization: What about ReLU?

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Change from tanh to ReLU

Slide credit: CS 231N

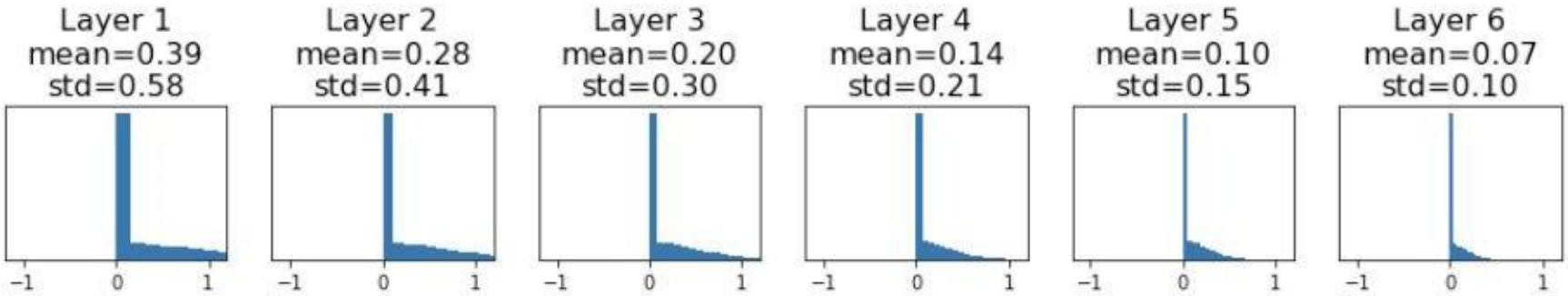
Weight Initialization: What about ReLU?

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Change from tanh to ReLU

Xavier assumes zero centered activation function

Activations collapse to zero again, no learning =(



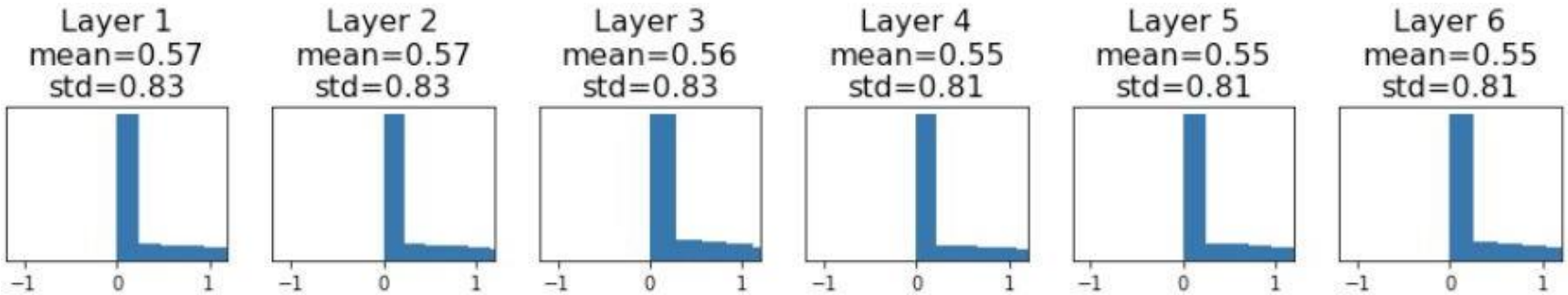
Slide credit: CS 231N

Weight Initialization: Kaiming / MSRA Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)

"Just right": Activations are nicely scaled for all layers!



He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

Slide credit: CS 231N

Kaiming Initialization

$$\frac{Random\ Numbers\ of\ dimension\ (D_{in}, D_{out})}{\sqrt{D_{in}\ /\ 2}}$$

- Default choice of training a deep network

Kaiming Initialization

- MIT Professor
- Author of ResNet (covered in this our CNN lecture), Faster R-CNN, and more

Default Choice for Activation & Initialization

Relu + Kaiming Initialization

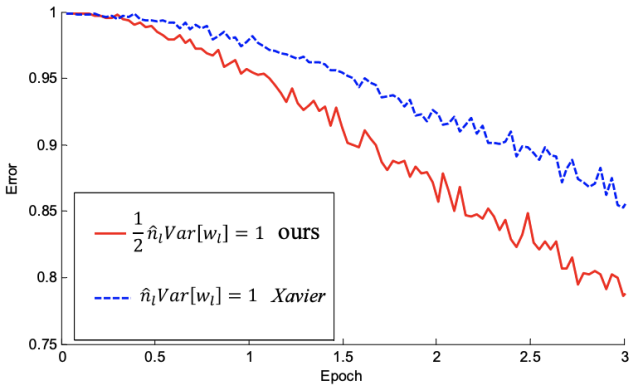


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

Xavier vs. He

Xavier: If activation is symmetric around 0 and gradients have equal variance in all directions, e.g., sigmoid and tangent.

He: ReLU does not have variance-preserving properties during backpropagation

Important Interview Questions

Question: In the context of weight initialization, what is the vanishing gradients problem, and how does initialization affect it?

Answer: The vanishing gradients problem occurs when the gradients of the network's loss with respect to the weights become very small, effectively preventing the weights from changing their values during training. This issue is particularly problematic in deep networks with many layers.

Proper initialization, like Xavier or Kaiming initialization, ensures that the gradients stay at an appropriate scale, not too small or too large, maintaining a healthy flow through the layers, thus mitigating the vanishing gradient problem.

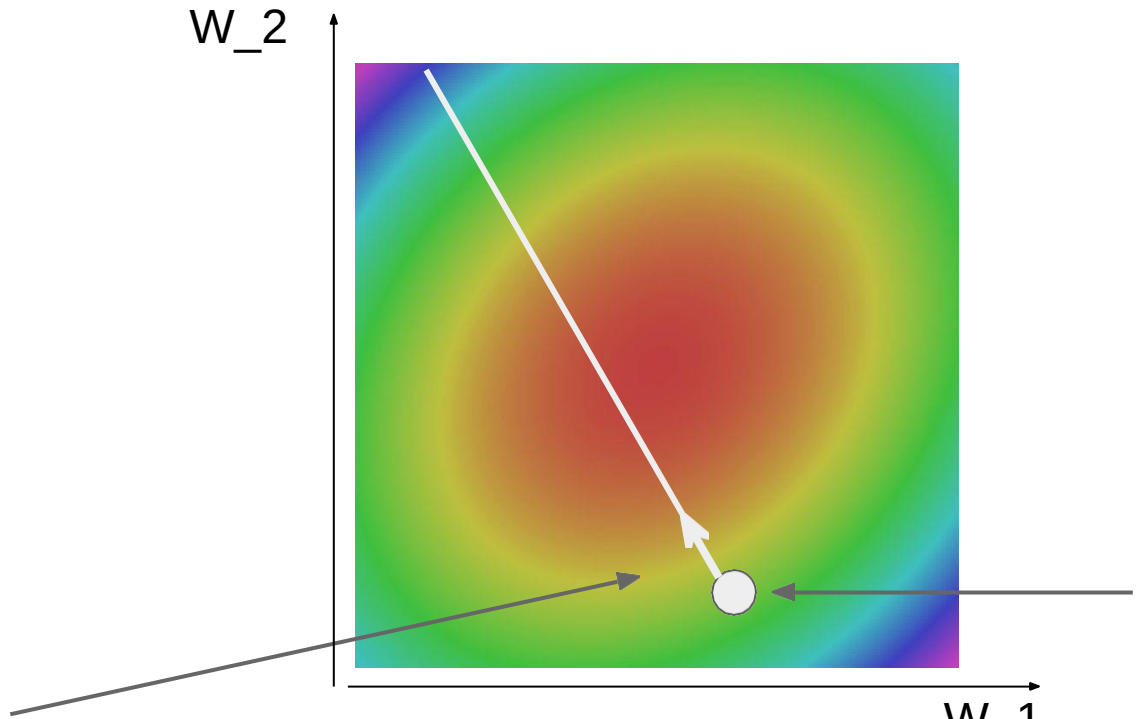
Training Neural Networks

Optimization

Gradient Descent

Use **all** data here to update the model weights

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

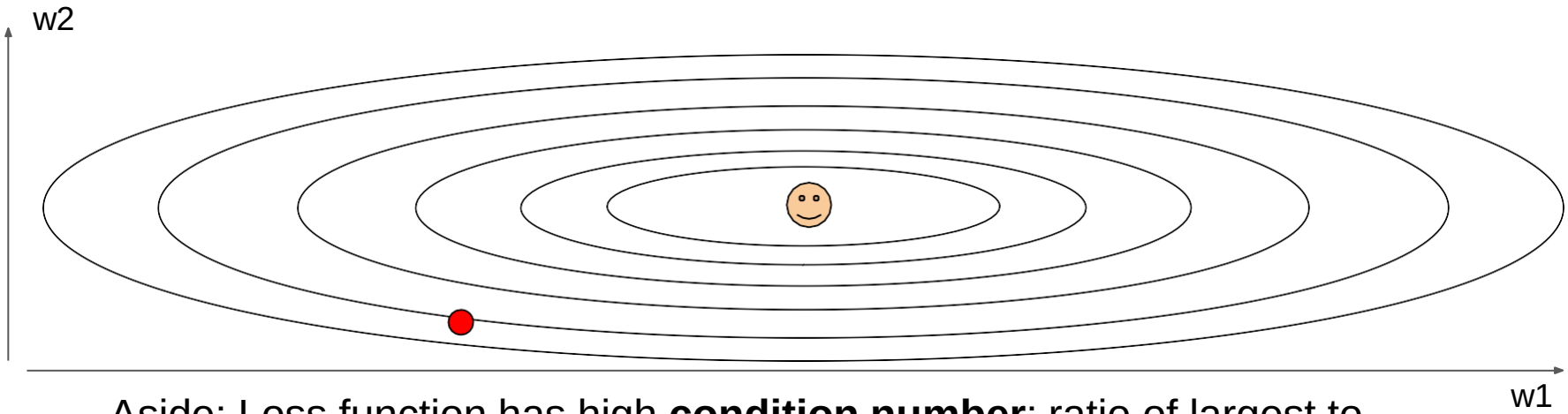
Full sum expensive when **N** is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Optimization: Problem #1 with SGD

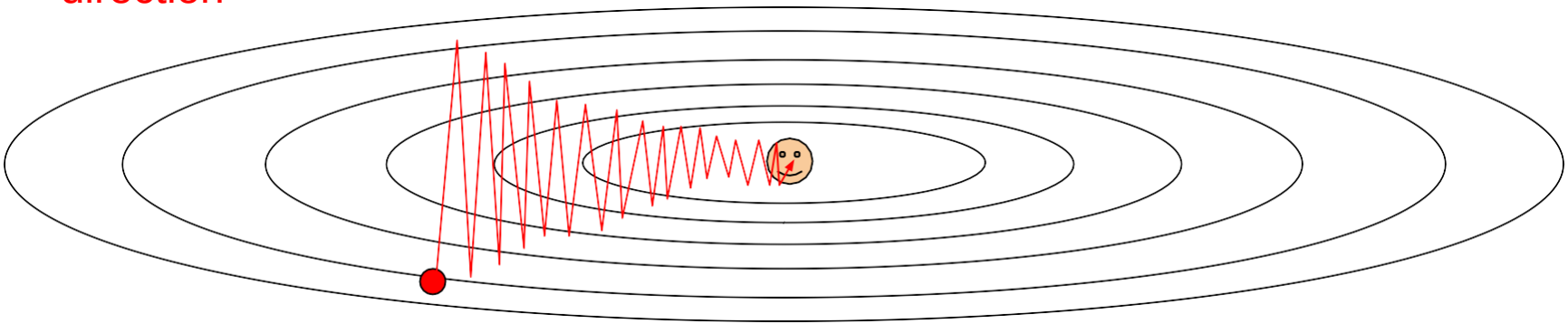
What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?



Aside: Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large. (**no need to know details though**)

Optimization: Problem #1 with SGD

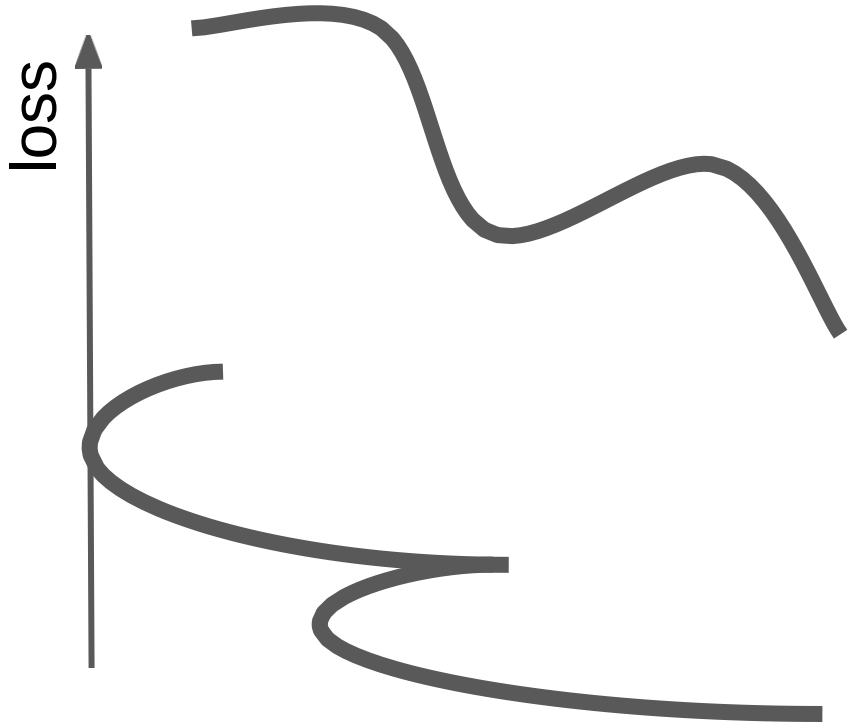
What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter (zigzag) along steep direction



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Optimization: Problem #2 with SGD

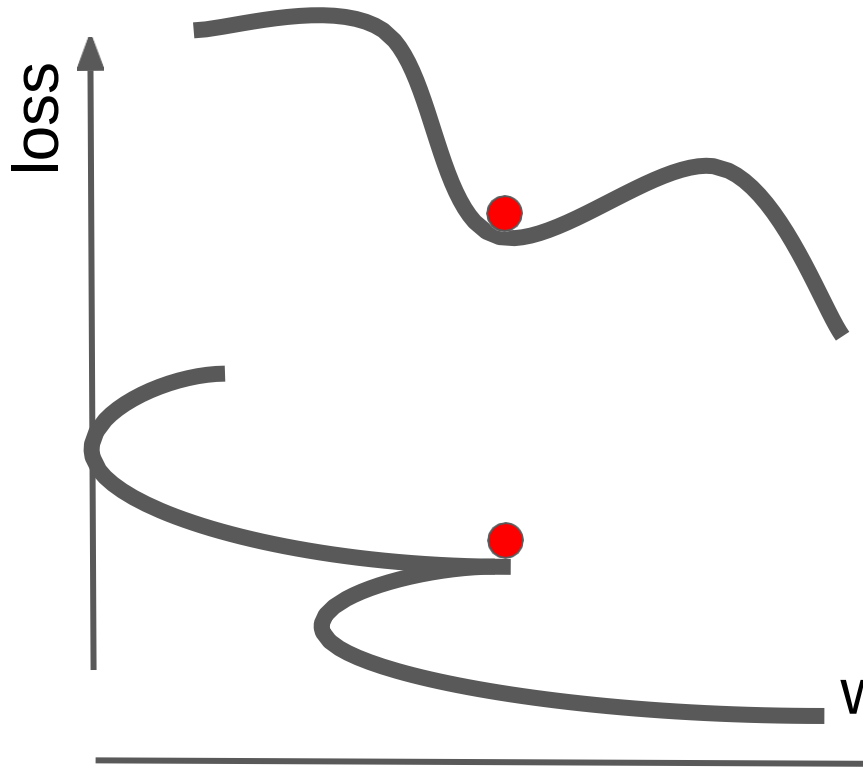
What if the loss function has a **local minima** or **saddle point**?



Optimization: Problem #2 with SGD

What if the loss function has a **local minima** or **saddle point**?

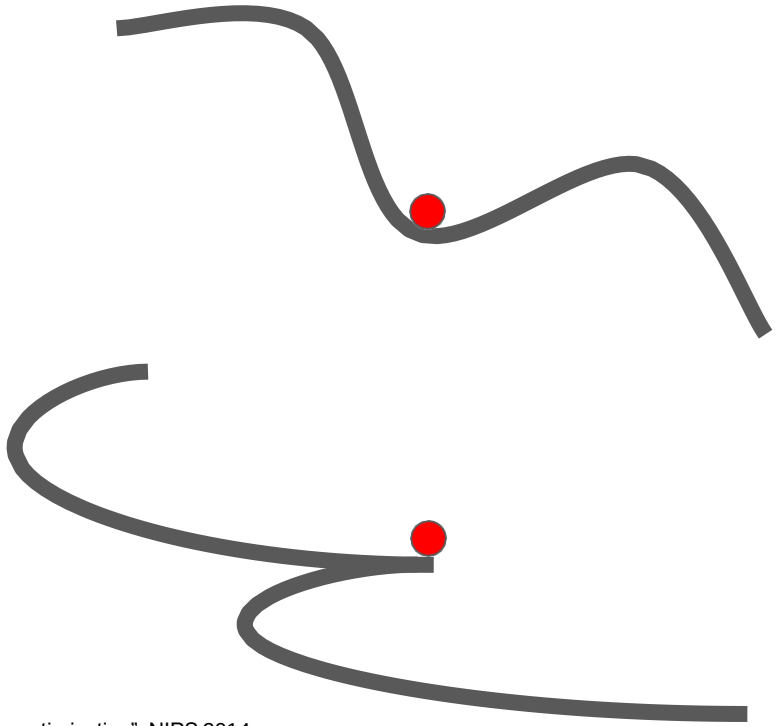
Zero gradient, gradient descent gets stuck



Optimization: Problem #2 with SGD

What if the loss function has a **local minima** or **saddle point**?

Saddle points much more common in high dimension



Dauphin et al, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”, NIPS 2014

Optimization: Problem #2 with SGD

saddle point in two dimension

$$f(x, y) = x^2 - y^2$$

$$\frac{\partial}{\partial x}(x^2 - y^2) = 2x \rightarrow 2(0) = 0$$

$$\frac{\partial}{\partial y}(x^2 - y^2) = -2y \rightarrow -2(0) = 0$$

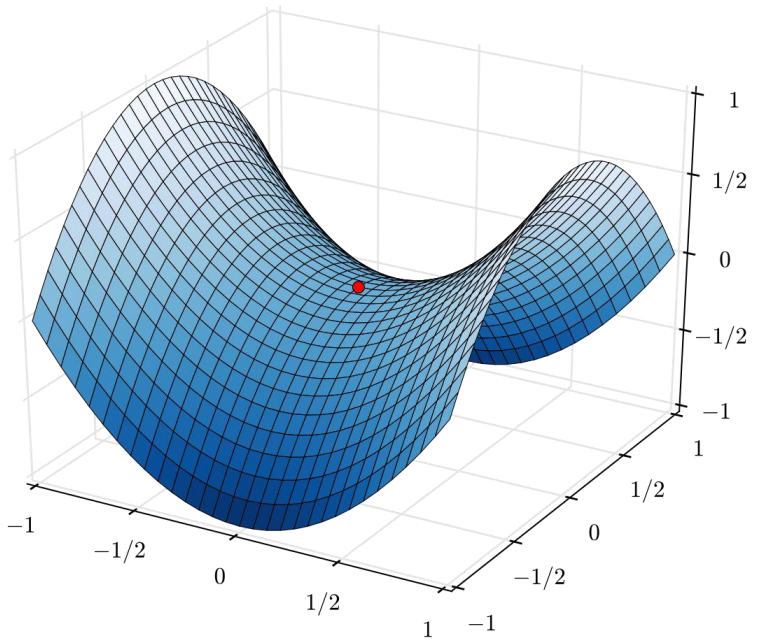


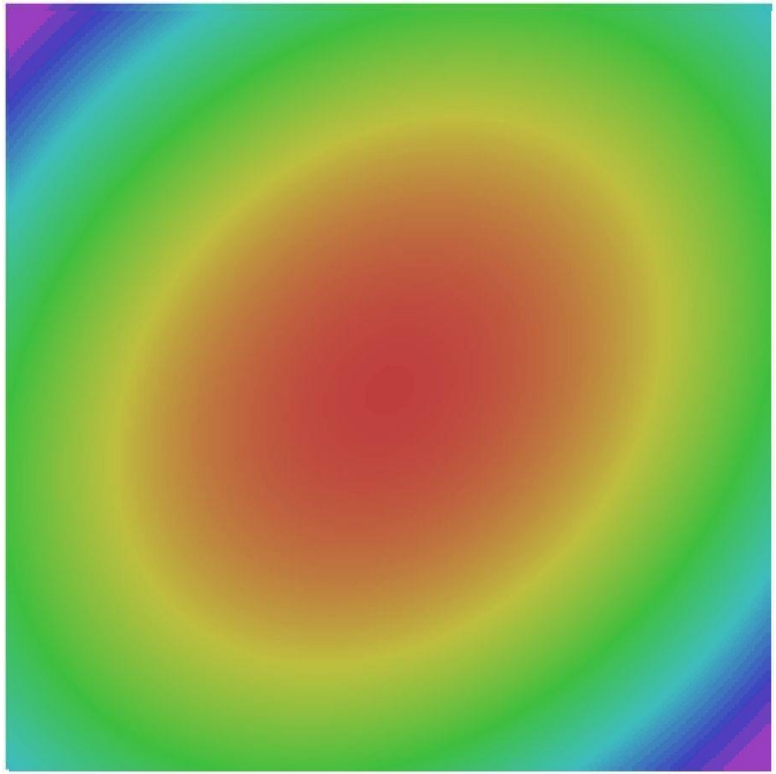
Image source: https://en.wikipedia.org/wiki/Saddle_point

Optimization: Problem #3 with SGD

Our gradients come from minibatches so they can be noisy!

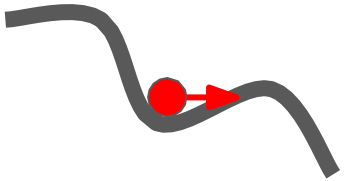
$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

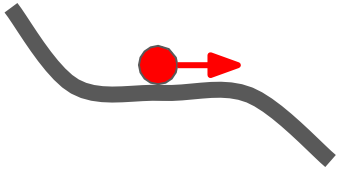


SGD + Momentum

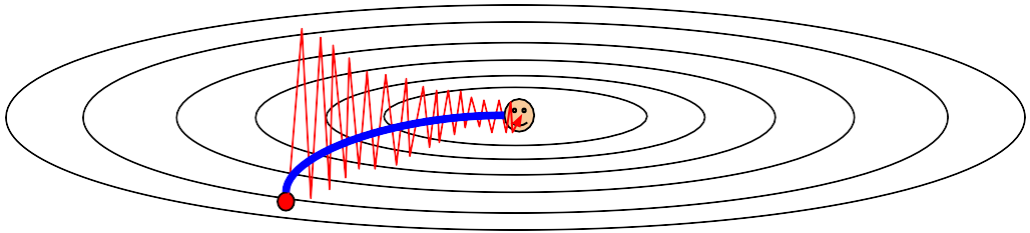
Local Minima



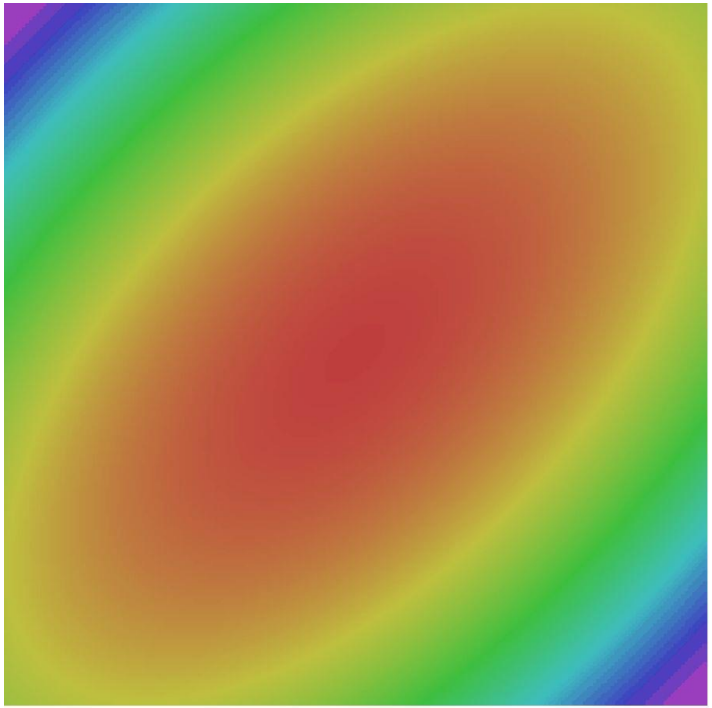
Saddle points



Poor Conditioning



Gradient Noise



SGD SGD+Momentum

SGD: the simple two line update code

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD + Momentum:

continue moving in the general direction as the previous iterations

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

SGD + Momentum:

continue moving in the general direction as the previous iterations

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Reduce the oscillations in the rapid change directions.

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

SGD + Momentum:
alternative equivalent formulation

SGD+Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx - learning_rate * dx
  x += vx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

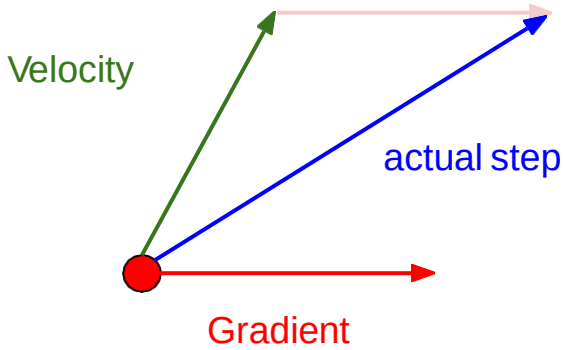
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

You may see SGD+Momentum formulated different ways, but they are equivalent - give same sequence of x

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

SGD+Momentum

Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k^2)”, 1983
Nesterov, “Introductory lectures on convex optimization: a basic course”, 2004
Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

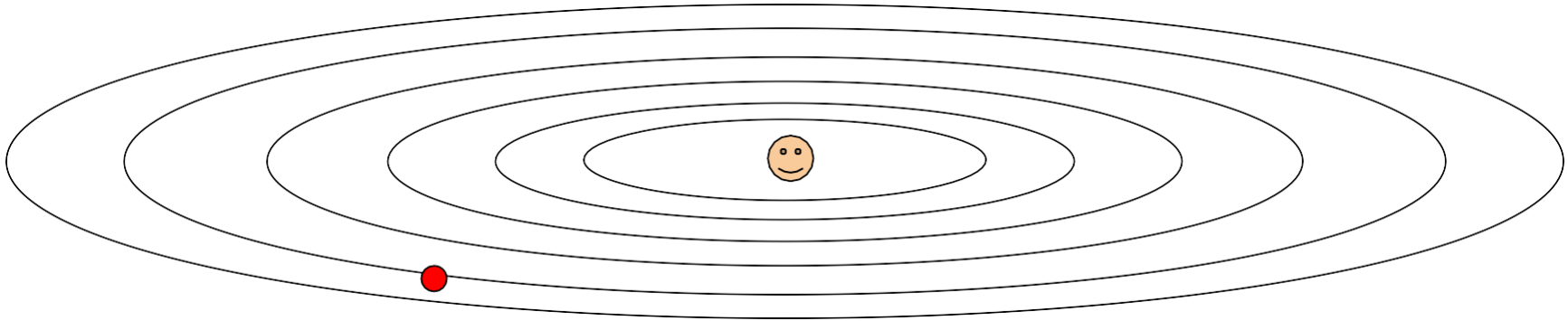
Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

"Per-parameter learning rates"
or "adaptive learning rates"

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

AdaGrad

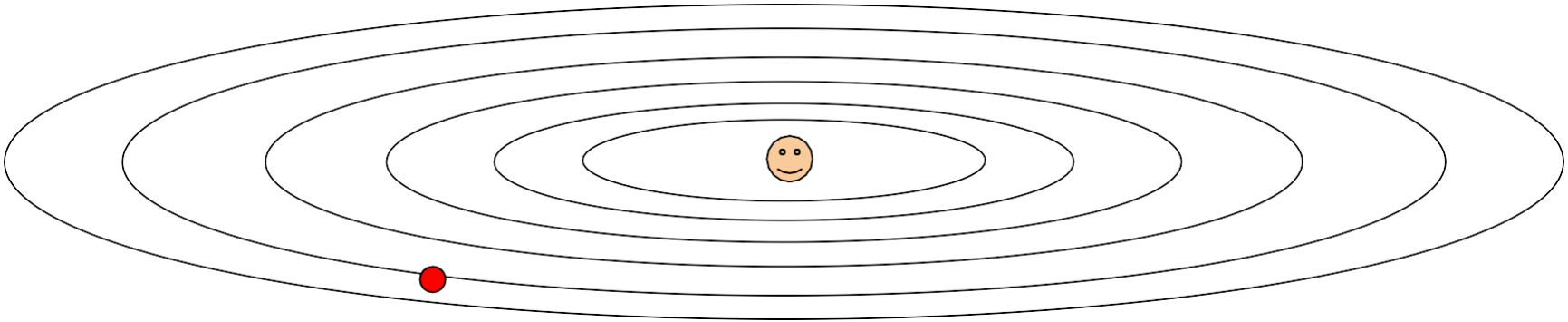
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q: What happens with AdaGrad?

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

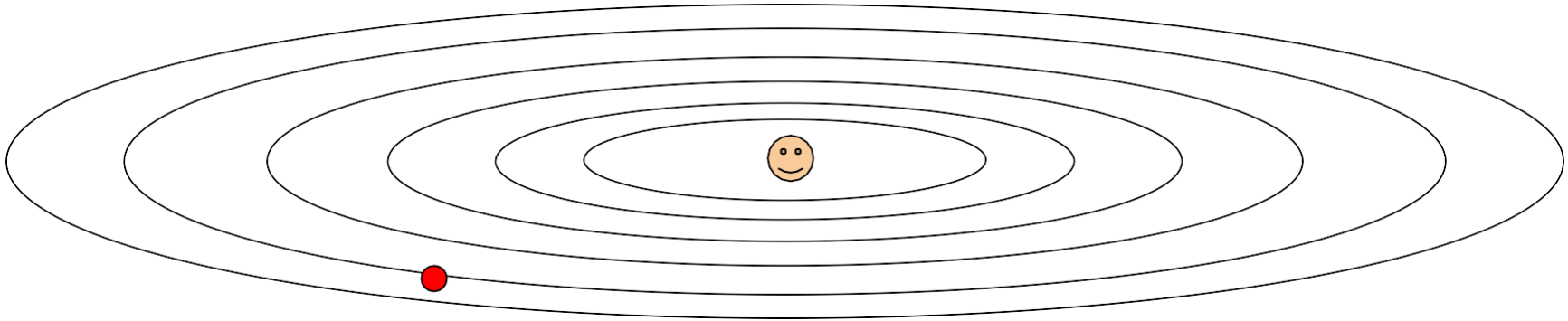


Q: What happens with AdaGrad?

Progress along “steep” directions is damped; progress along “flat” directions is accelerated

AdaGrad

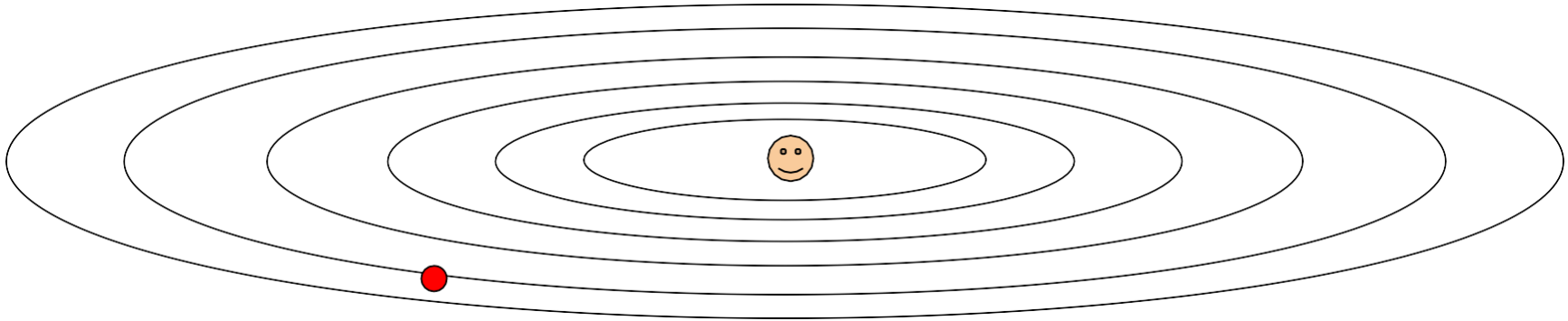
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q2: What happens to the step size over long time?

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q2: What happens to the step size over long time?

Decays to zero

RMSProp: “Leaky AdaGrad”

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Kingma and Ba, “Adam: Amethod for stochastic optimization”, ICLR 2015

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Sort of like RMSProp with momentum

Q: What happens at first timestep?

Kingma and Ba, "Adam: Amethod for stochastic optimization", ICLR 2015

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Kingma and Ba, "Adam: Amethod for stochastic optimization", ICLR 2015

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

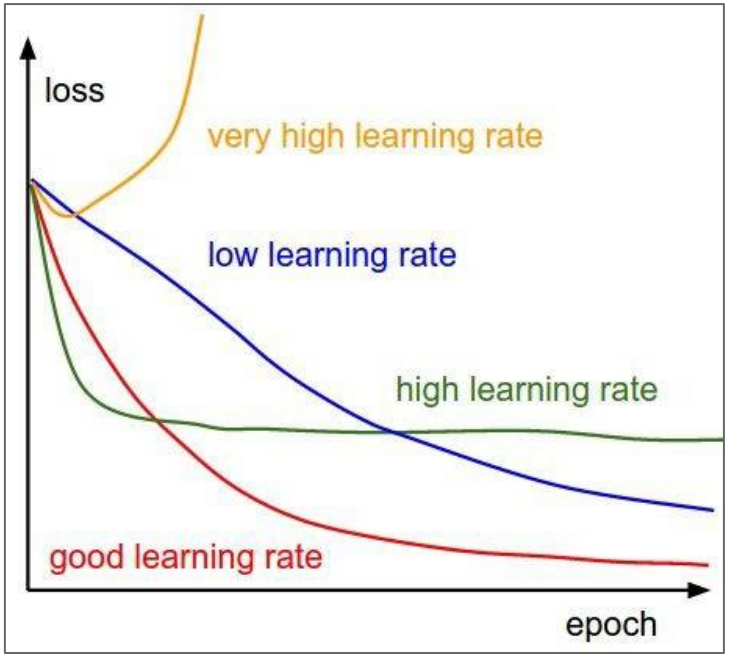
Kingma and Ba, "Adam: Amethod for stochastic optimization", ICLR 2015

Learning rate schedules

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

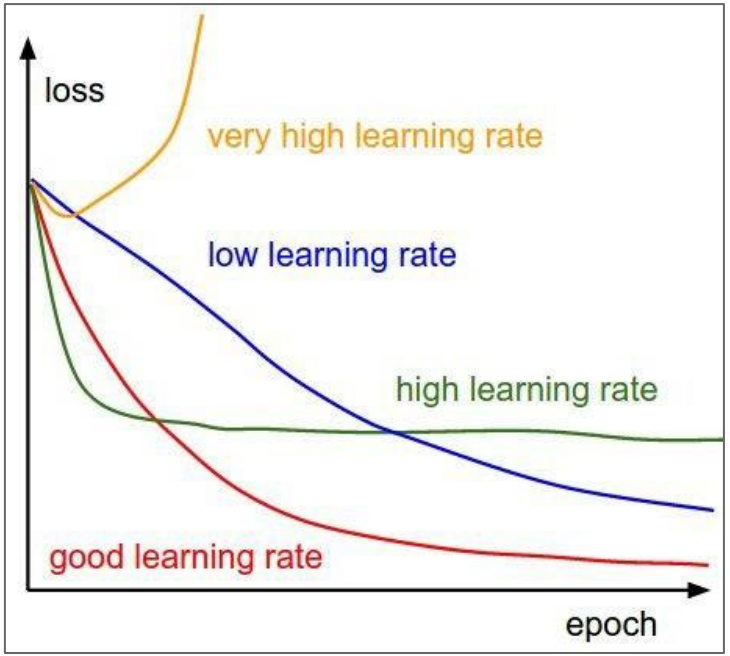
Learning rate

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?

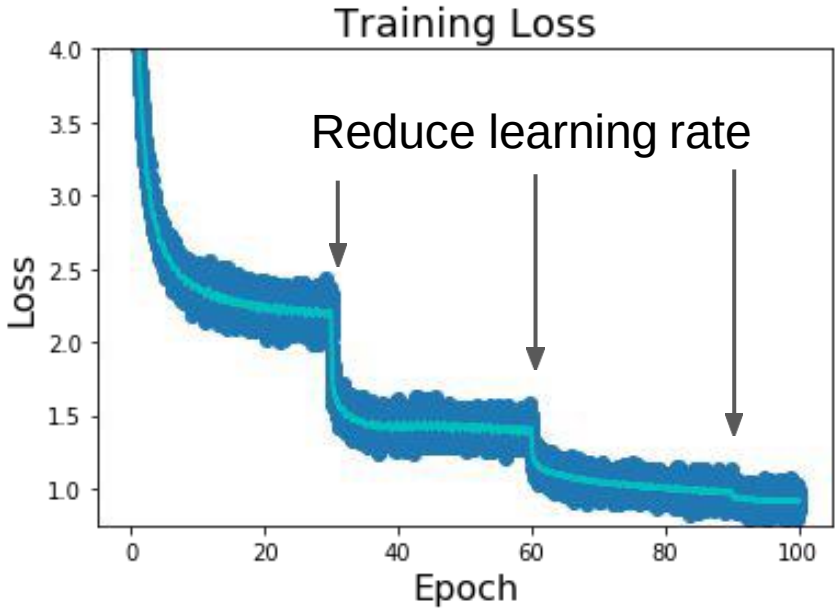
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?

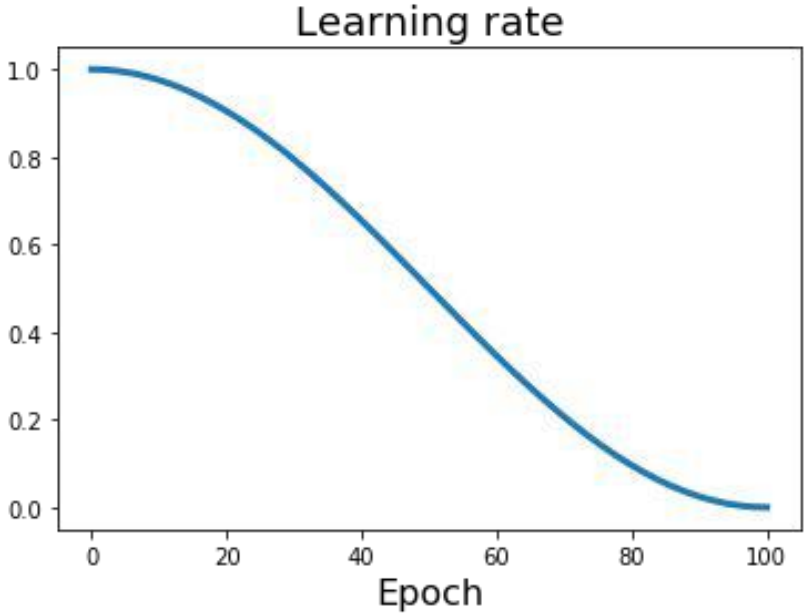
A: In reality, all of these are good learning rates (except the yellow one?).

Learning rate decays over time



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine:

$$\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$$

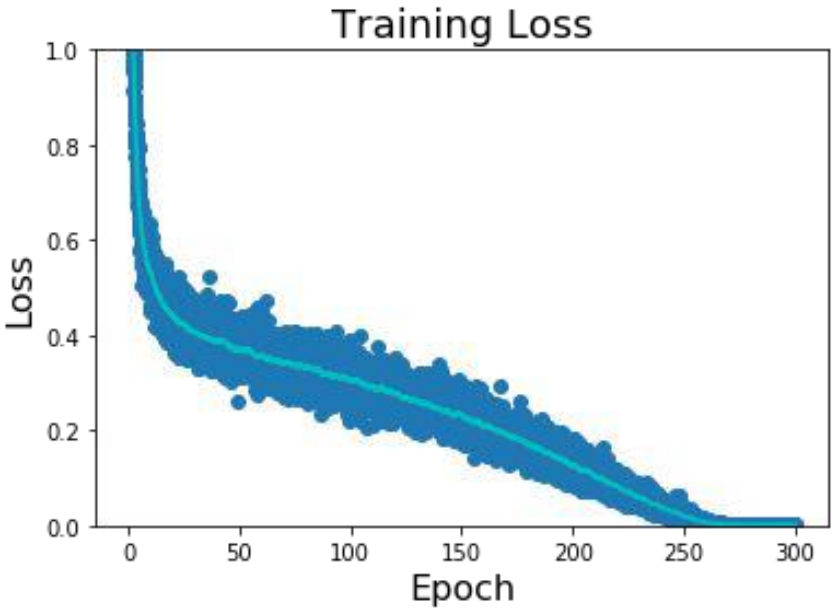
α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs

Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
Radford et al, "Improving Language Understanding by Generative Pre-Training", 2018
Feichtenhofer et al, "SlowFast Networks for Video Recognition", arXiv 2018
Child at al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine:

$$\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$$

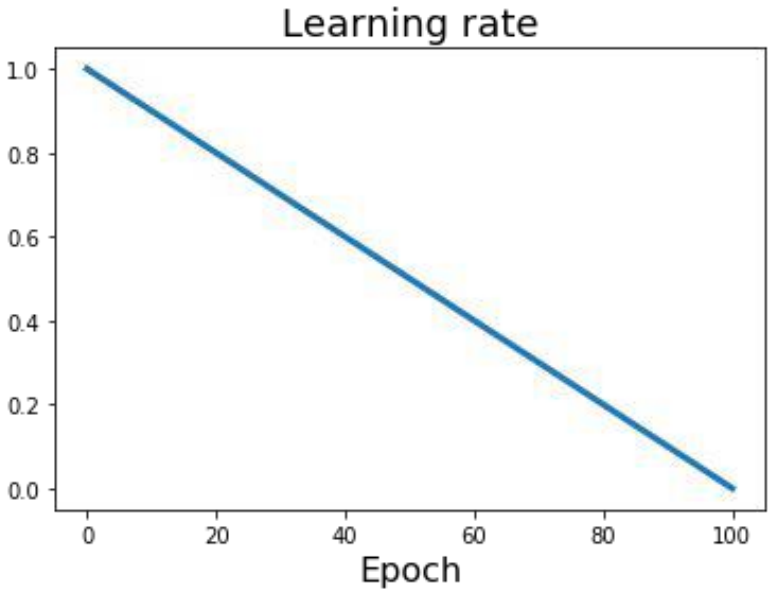
α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs

Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
Radford et al, "Improving Language Understanding by Generative Pre-Training", 2018
Feichtenhofer et al, "SlowFast Networks for Video Recognition", arXiv 2018
Child at al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

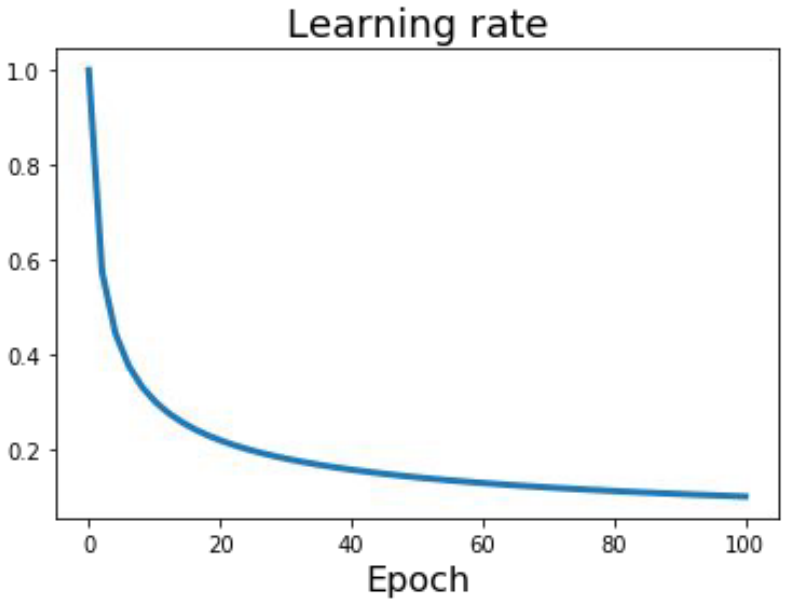
Cosine: $\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$

Linear: $\alpha_t = \alpha_0(1 - t/T)$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2018

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: **Linear:** **Inverse**

sqrt:

$$\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$$

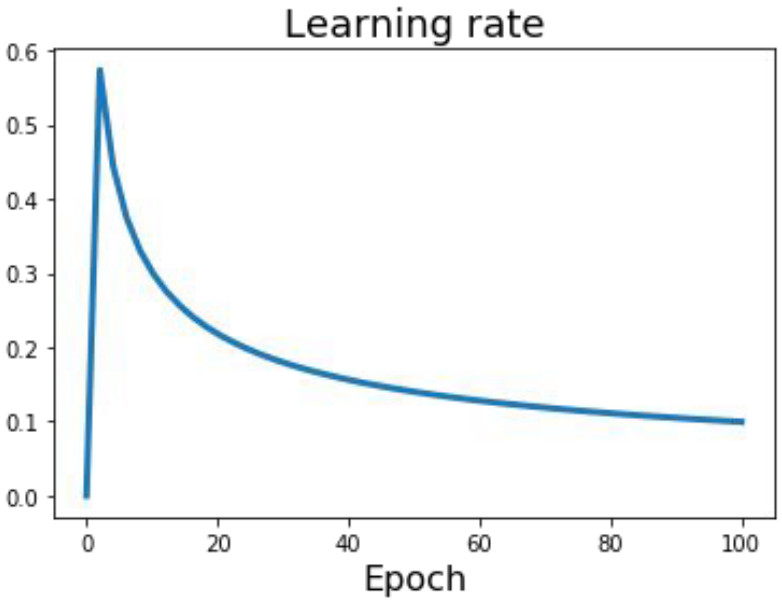
$$\alpha_t = \alpha_0(1 - t/T)$$

$$\alpha_t = \alpha_0/\sqrt{t}$$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

Vaswani et al, “Attention is all you need”, NIPS 2017

Learning Rate Decay: Linear Warmup



High initial learning rates can make loss explode; linearly increasing learning rate from 0 over the first ~5,000 iterations can prevent this.

Empirical rule of thumb: If you increase the batch size by N, also scale the initial learning rate by N

Goyal et al, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”, arXiv 2017

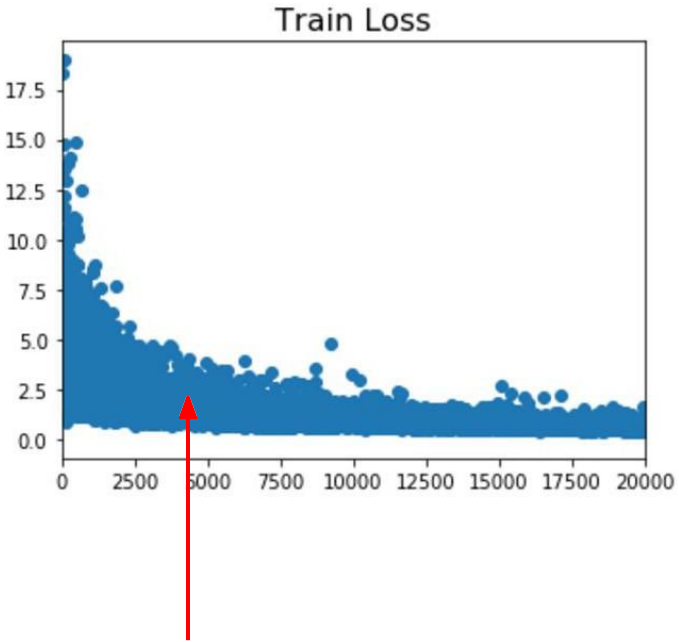
In practice:

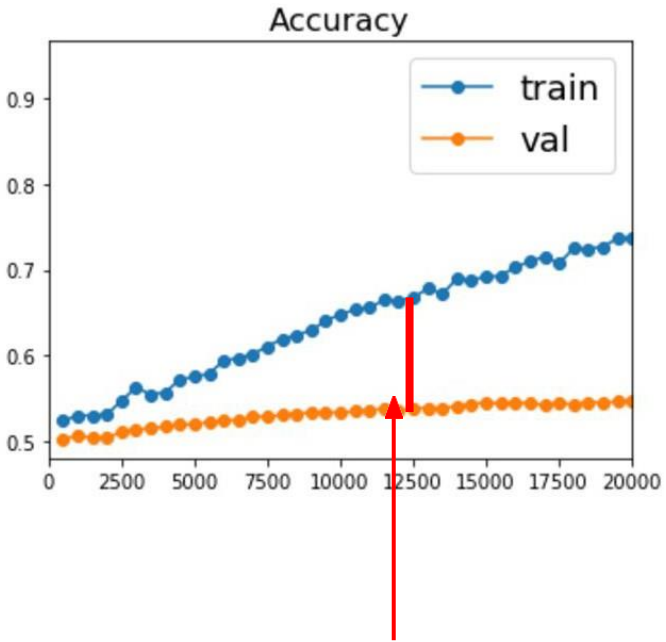
- **Adam** is a good default choice in many cases; it often works ok even with constant learning rate
- **SGD+Momentum** can outperform Adam but may require more tuning of LR and schedule

Training Neural Networks

Training vs. Test Error

Beyond Training Error

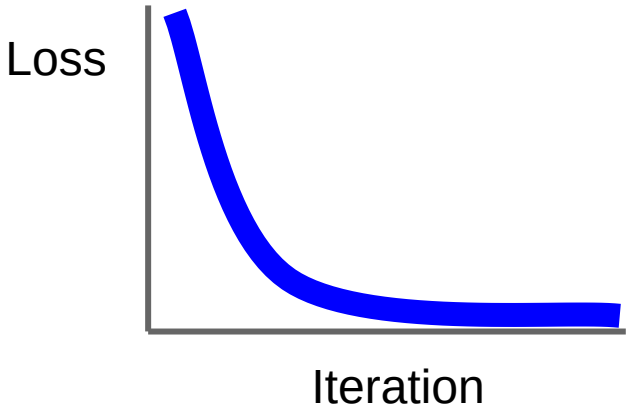


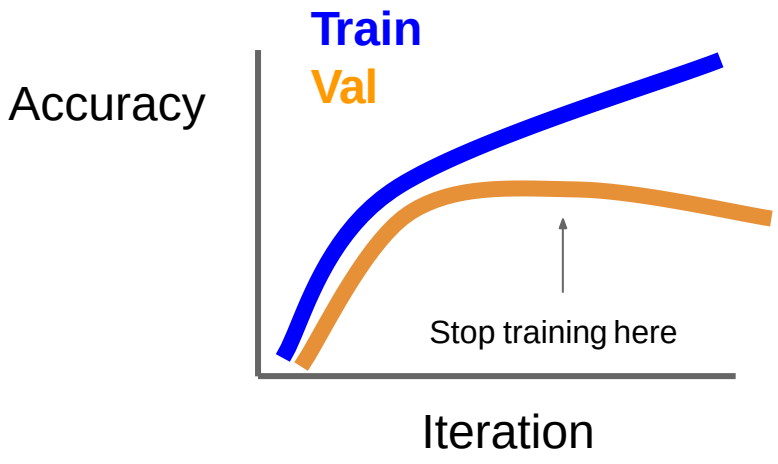


Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

Early Stopping: Always do this





Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked best on val

Model Ensembles

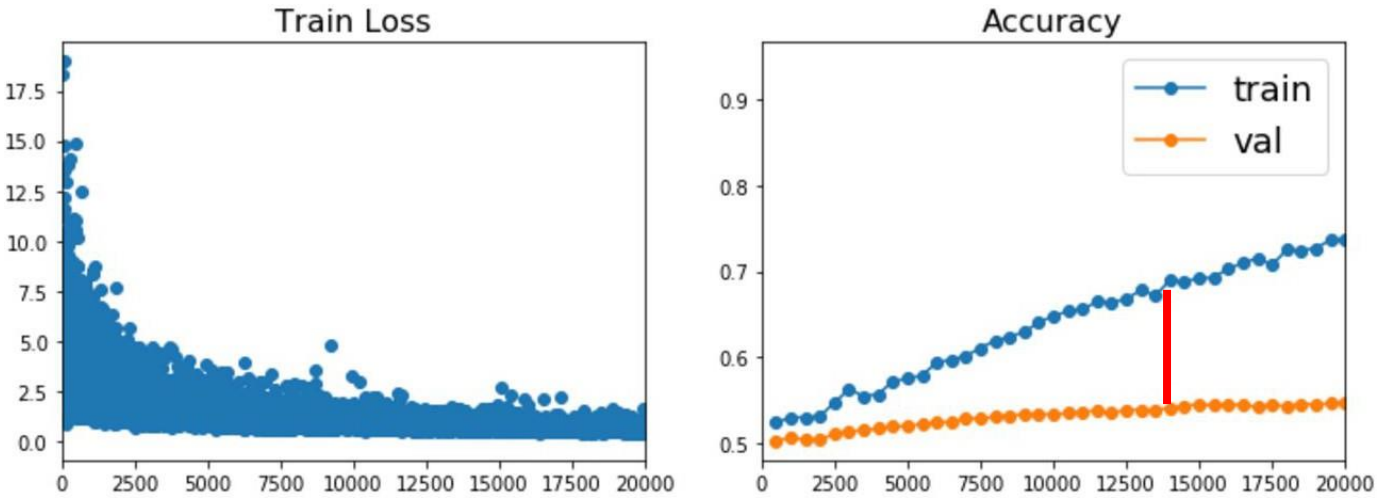
1. Train multiple independent models
2. At test time average their results

   (Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance

Recall CNN ensembles we studied

How to improve single-model performance?



Regularization

Regularization: Add term to loss

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

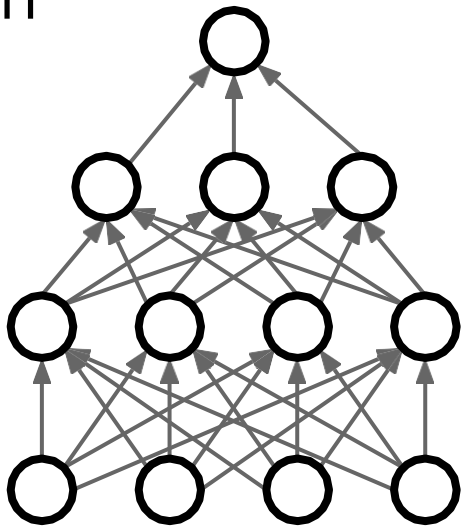
L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

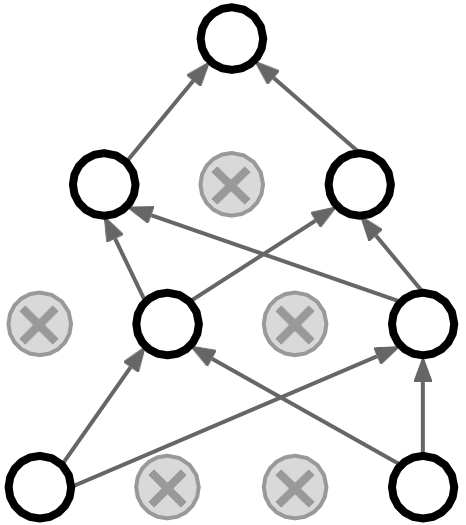
L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization: Dropout

In each forward pass, randomly set some neurons to zero Probability of dropping is a hyperparameter; 0.5 is common



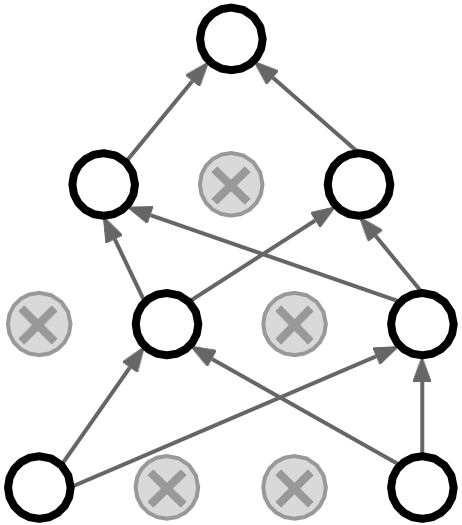


Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Regularization: Dropout

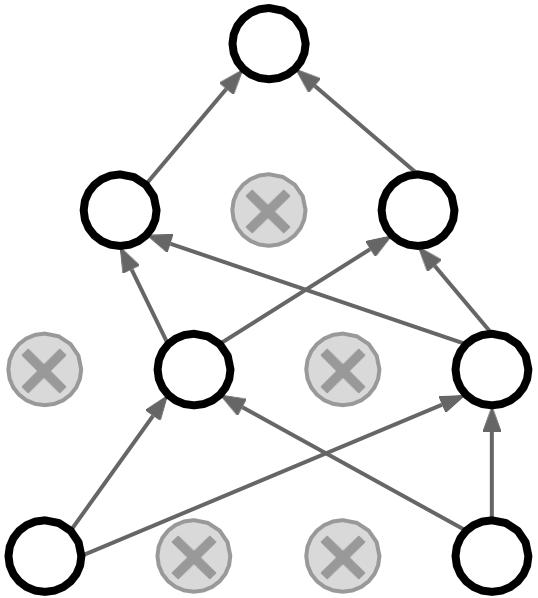
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Example forward pass with a 3-layer network using dropout



Regularization: Dropout

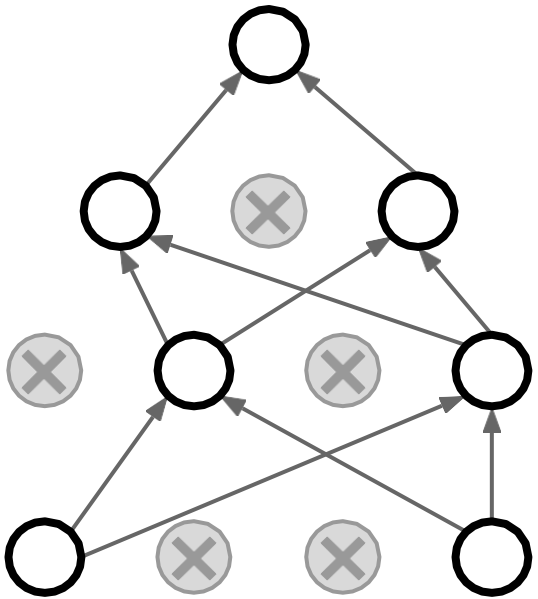
How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has
$2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Dropout: Test time

Dropout makes our output random!

Output (label) Input (image)

$$y = f_W(x, z)$$

Random mask

Want to "average out" the randomness at test-time

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

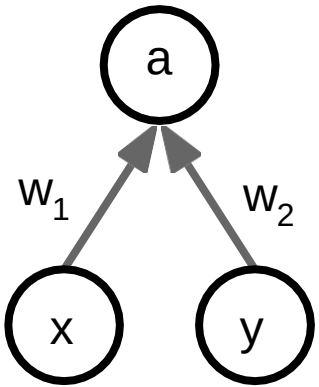
But this integral seems hard …

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

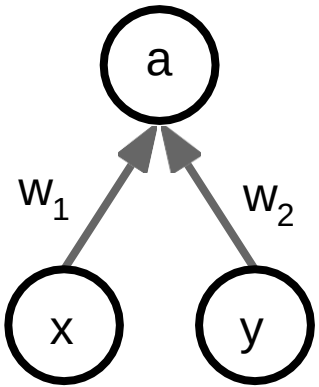
Consider a single neuron.



Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



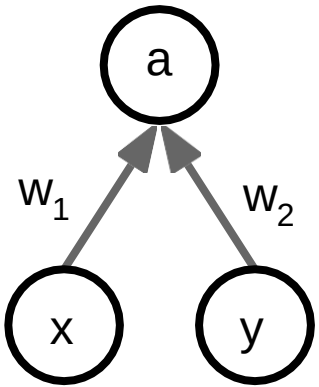
Consider a single neuron.

At test time we have: $E\big[a\big] = w_1 x + w_2 y$

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



Consider a single neuron with two inputs x and y.

At test time we have: $E\big[a\big] = w_1 x + w_2 y$

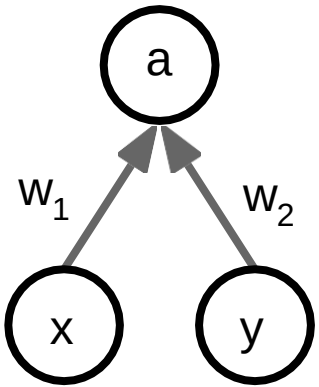
During training we have:

$$\begin{aligned} E\big[a\big] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



Consider a single neuron.

At test time we have: $E\big[a\big] = w_1 x + w_2 y$

During training we have:

$$\begin{aligned} E\big[a\big] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

At test time, **multiply** by dropout probability

Dropout: Test time

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

Dropout Summary

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

drop in train time

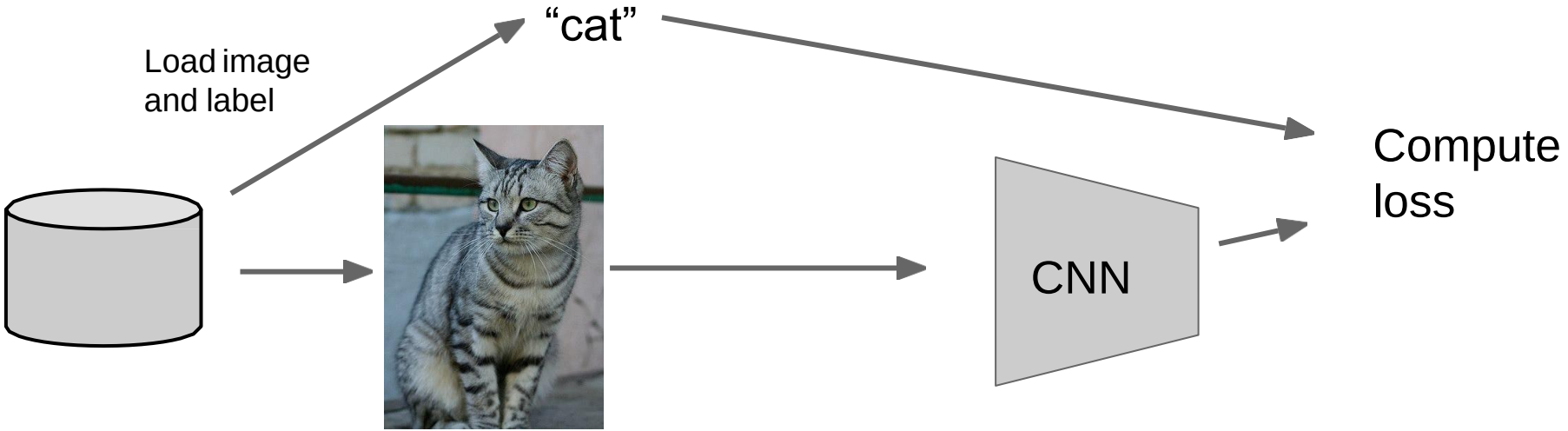
scale at test time

More common: “Inverted dropout”

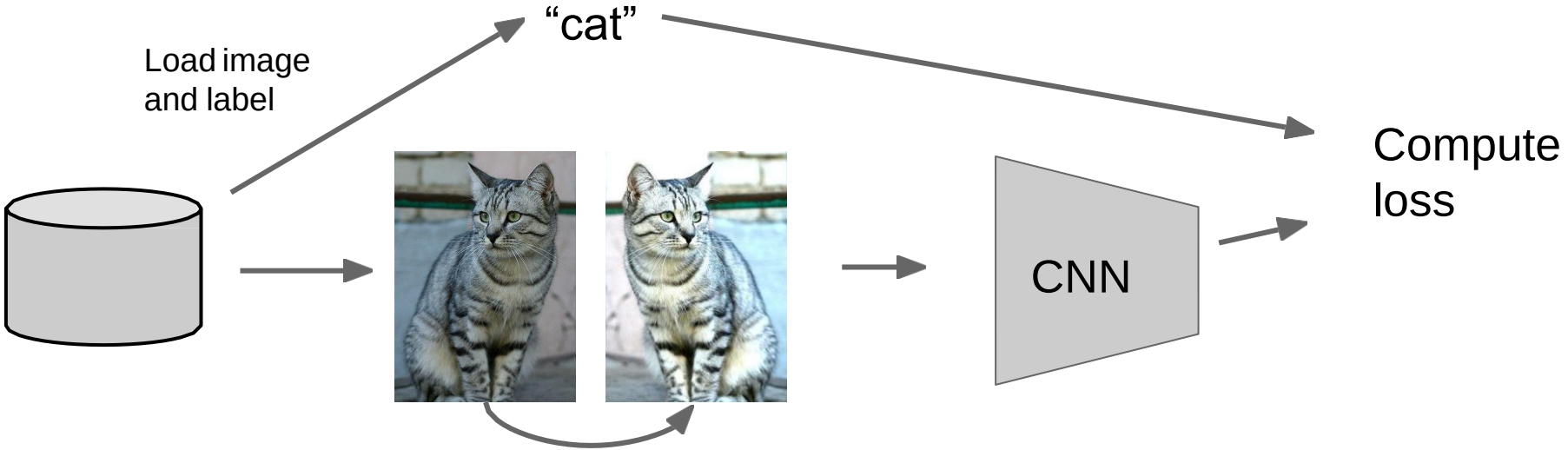
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test time is unchanged!

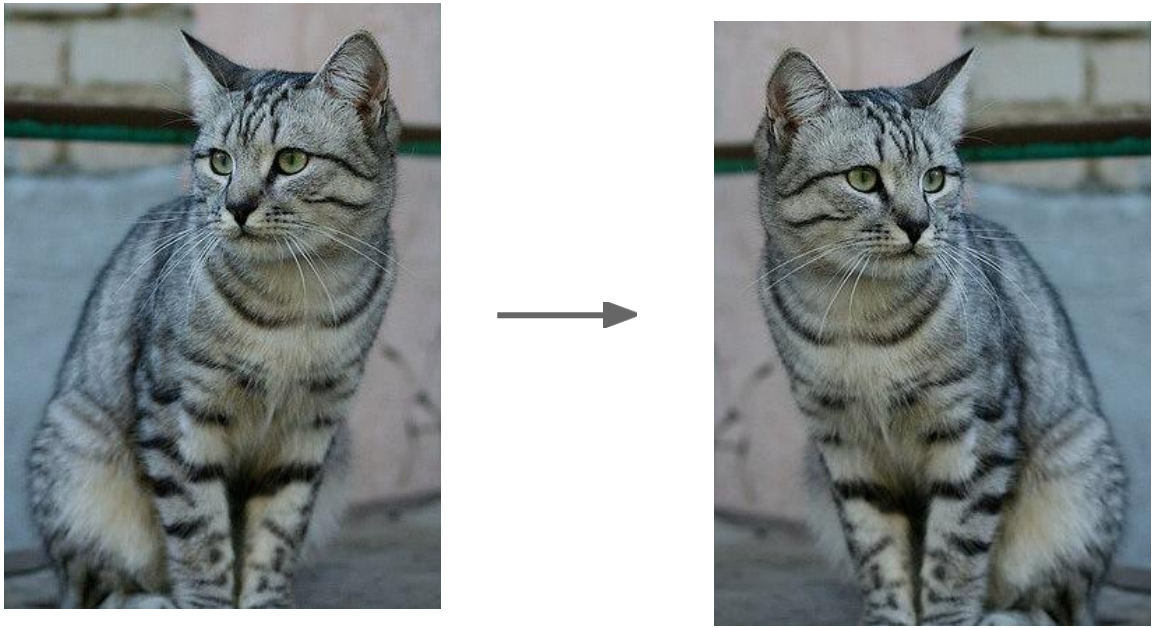
Regularization: Data Augmentation



Regularization: Data Augmentation



Data Augmentation
Horizontal Flips



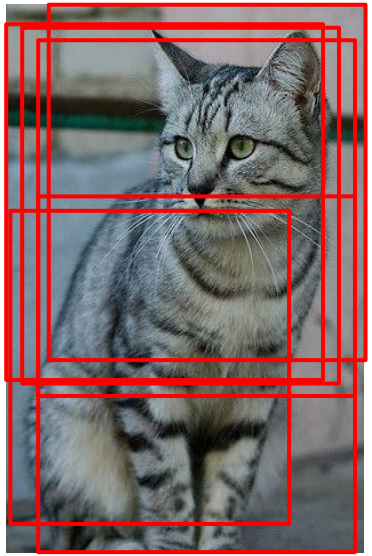
Data Augmentation

Random crops and scales

Training: sample random crops / scales

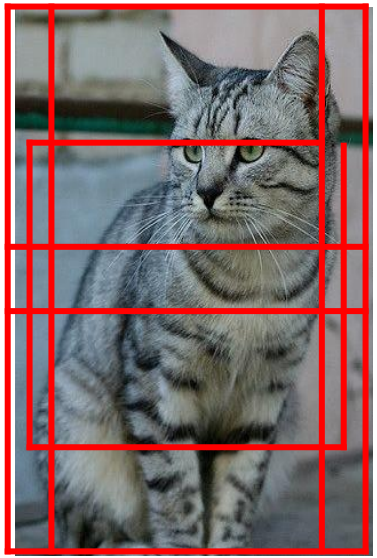
ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Data Augmentation

Random crops and scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Data Augmentation

Color Jitter

Simple: Randomize
contrast and brightness

Data Augmentation
Color Jitter

Simple: Randomize contrast and brightness

More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012],* ResNet, etc)

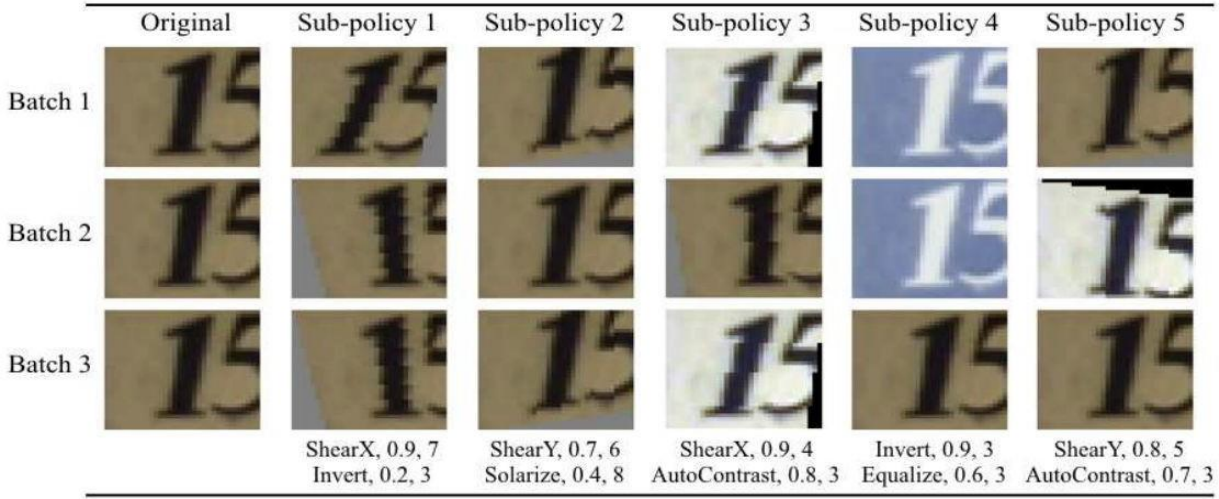
Data Augmentation

Get creative for your problem!

Examples of data augmentations:

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Automatic Data Augmentation



Cubuk et al., "AutoAugment: Learning Augmentation Strategies from Data", CVPR 2019

Regularization: A common pattern

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout

Batch Normalization

Data Augmentation

Regularization: DropConnect

Training: Drop connections between neurons (set weights to 0)

Testing: Use all the connections

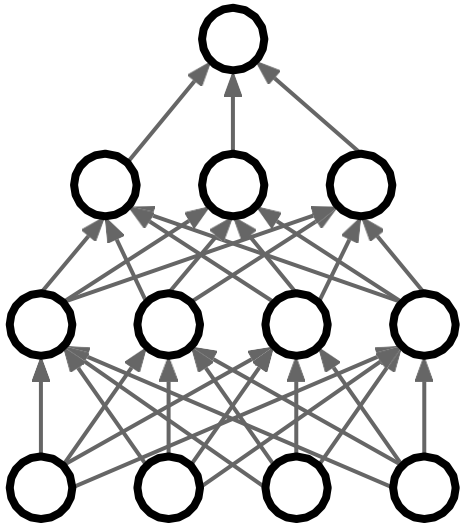
Examples:

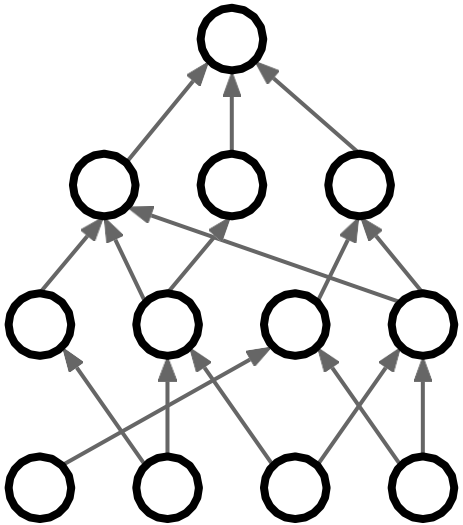
Dropout

Batch Normalization

Data Augmentation

DropConnect





Wan et al, “Regularization of Neural Networks using DropConnect”, ICML 2013

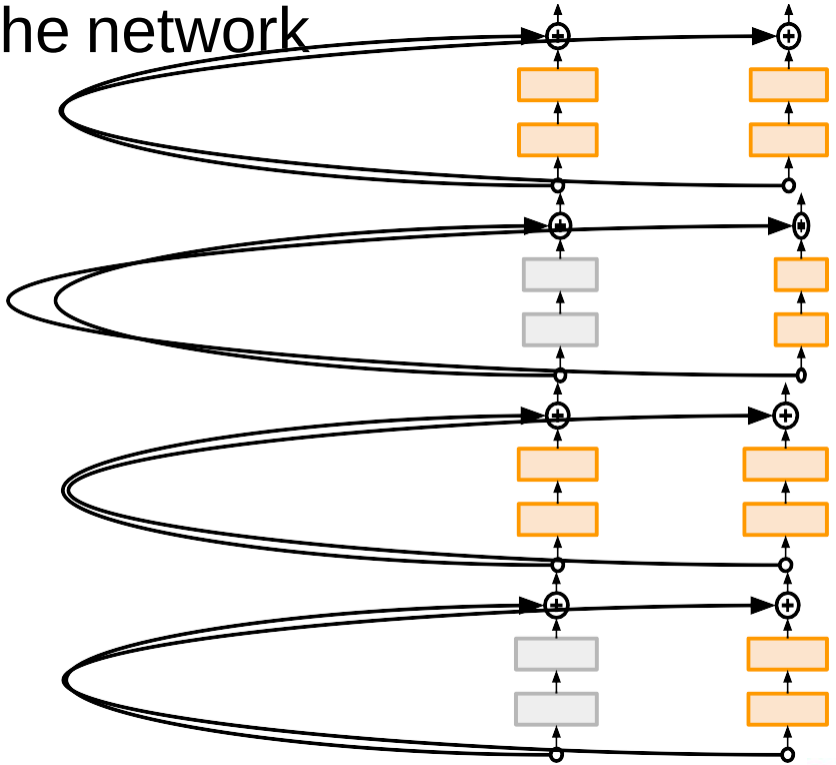
Regularization: Stochastic Depth

Training: Skip some layers in the network

Testing: Use all the layer

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth



Huang et al, “Deep Networks with Stochastic Depth”, ECCV 2016

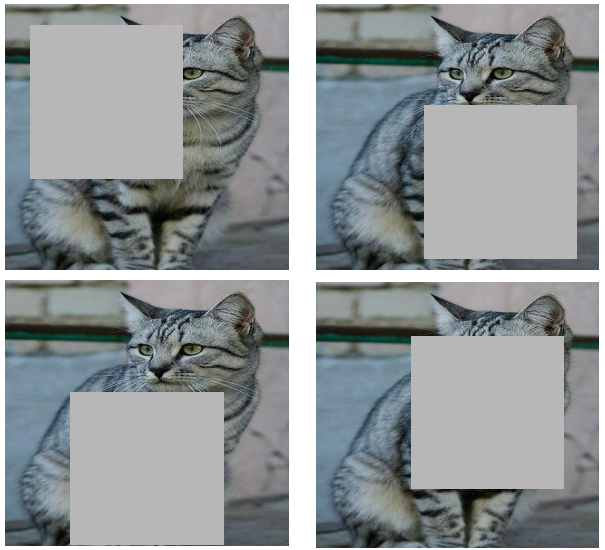
Regularization: Cutout

Training: Set random image regions to zero

Testing: Use full image

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop



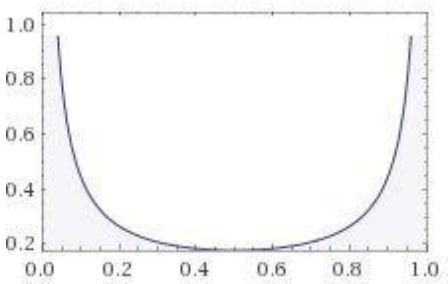
Works very well for small datasets like CIFAR, less common for large datasets like ImageNet

DeVries and Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout”, arXiv 2017

Regularization: Mixup

Training: Train on random blends of images

Testing: Use original images



Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop
Mixup

CNN

Target label:
cat: 0.4
dog: 0.6

Randomly blend the pixels of pairs of training images, e.g. 40% cat, 60% dog

Zhang et al, "*mixup*: Beyond Empirical Risk Minimization", ICLR 2018

Regularization - In practice

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop
Mixup

- Consider dropout for large fully-connected layers
- Batch normalization and data augmentation almost always a good idea
- Try cutout and mixup especially for small classification datasets

Training Neural Networks

Choosing Hyperparameters

Choosing Hyperparameters

Hyperparameters: usually set beforehand

Parameters: learnable, e.g., model weights

Choosing Hyperparameters

Step 1: Check initial loss

Turn off weight decay, sanity check loss at initialization
e.g. log(C) for softmax with C classes

Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample

Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization

Loss not going down? LR too low, bad initialization
Loss explodes to Inf or NaN? LR too high, bad initialization

Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down

Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations

Good learning rates to try: 1e-1, 1e-2, 1e-3, 1e-4

Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs

Choose a few values of learning rate and weight decay around what worked from Step 3, train a few models for ~1-5 epochs.

Good weight decay to try: 1e-4, 1e-5, 0

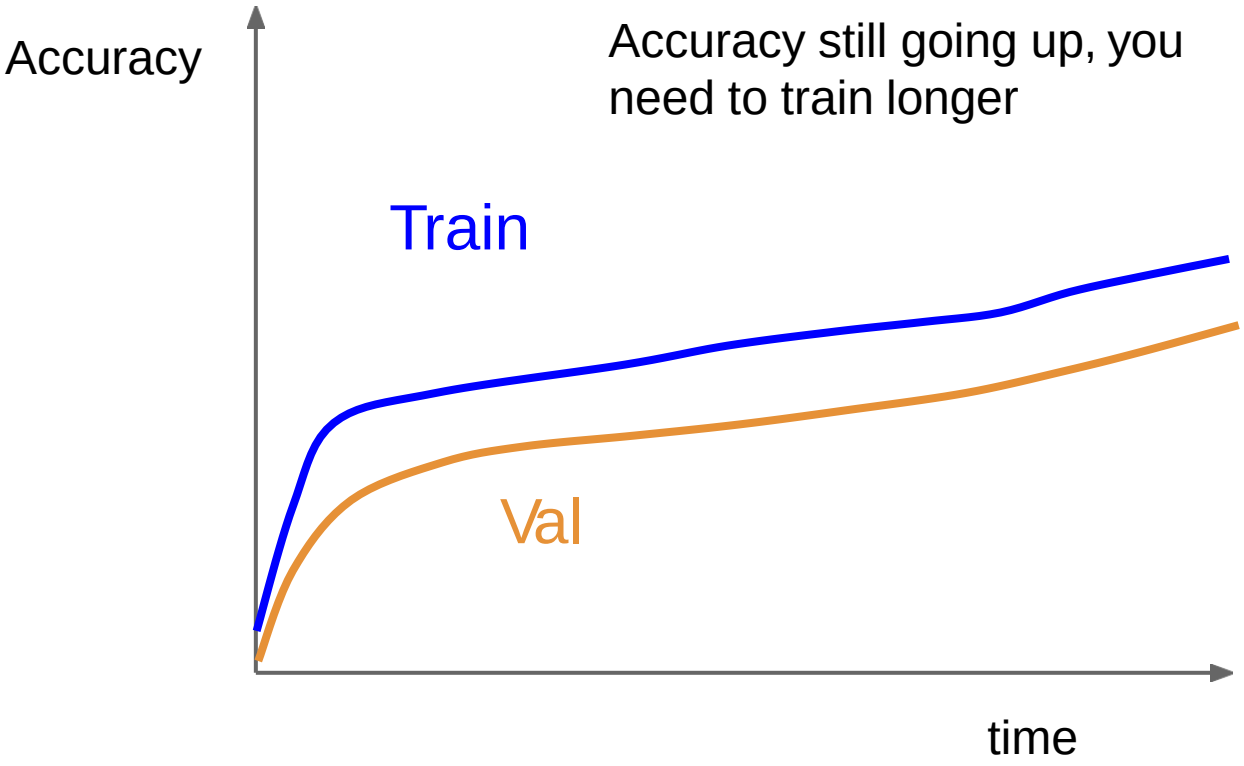
Choosing Hyperparameters

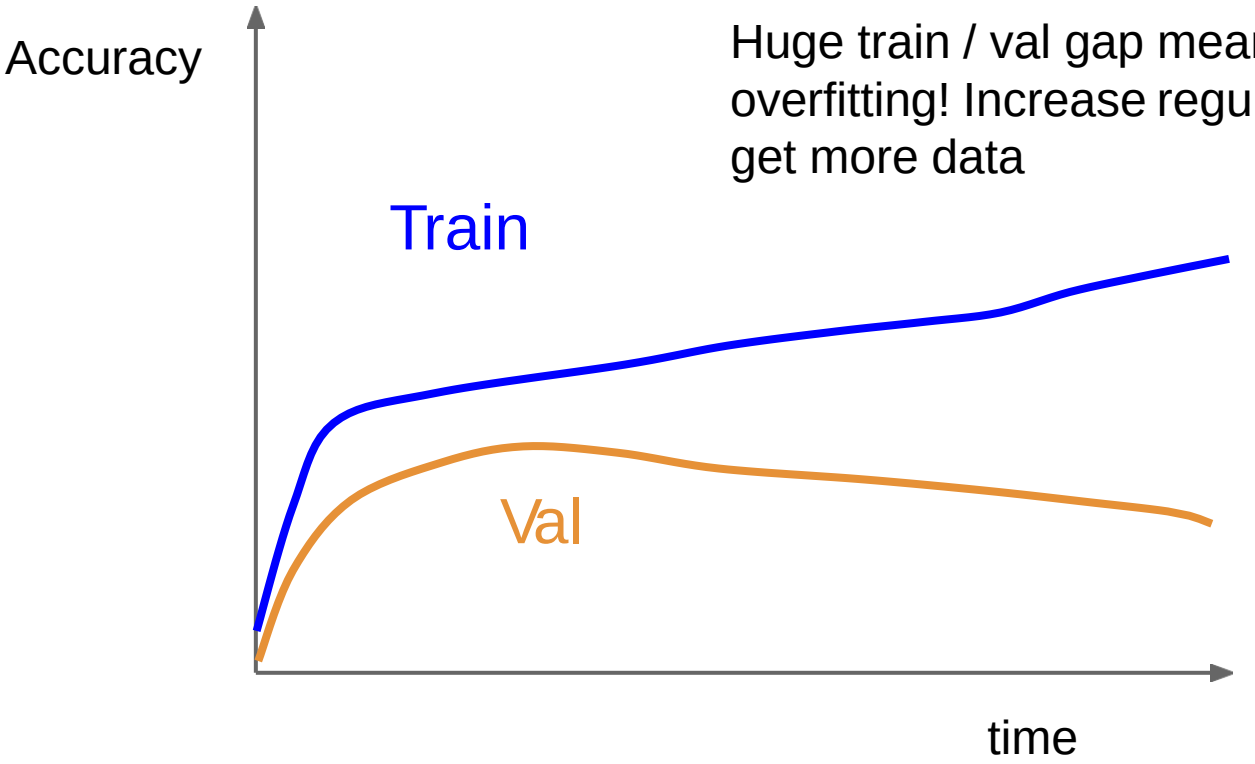
Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer

Pick best models from Step 4, train them for longer (~10-20 epochs) without learning rate decay

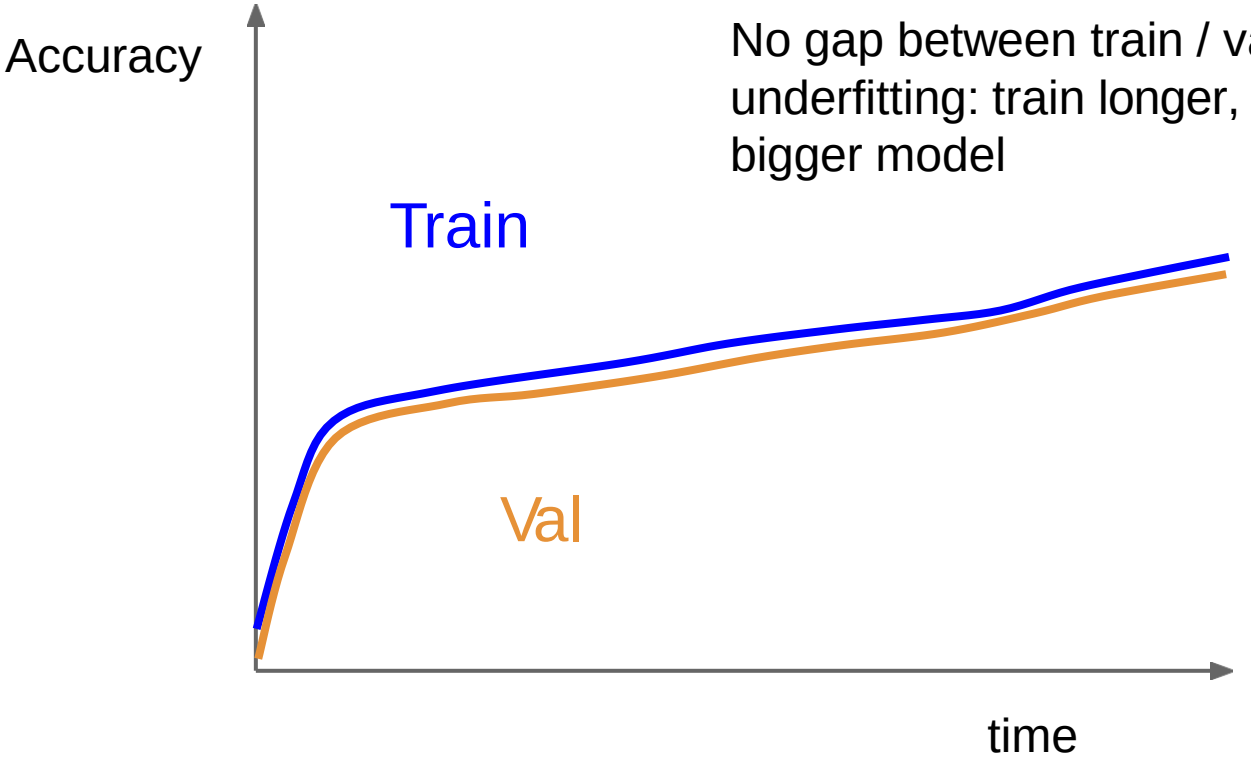
Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at loss and accuracy curves





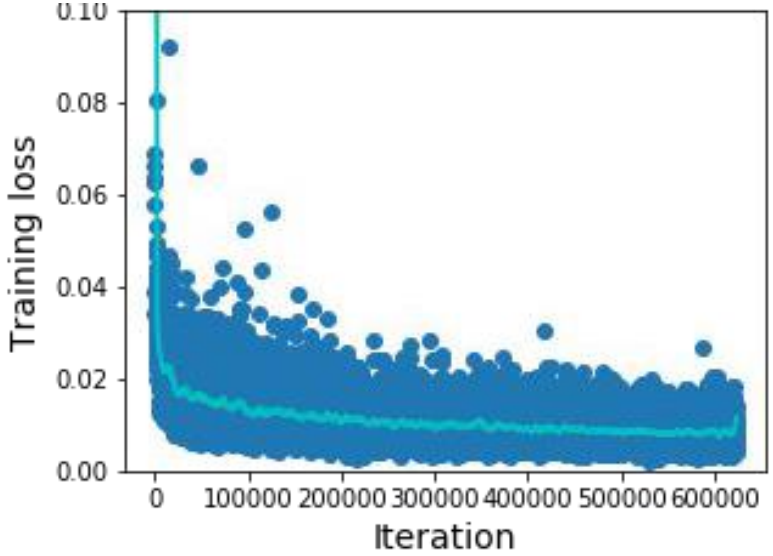
Huge train / val gap means overfitting! Increase regularization, get more data



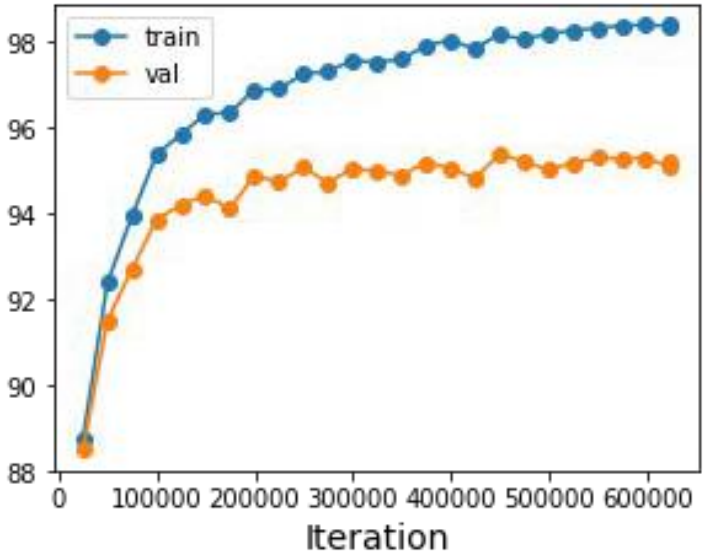
No gap between train / val means underfitting: train longer, use a bigger model

Look at learning curves!

Training Loss

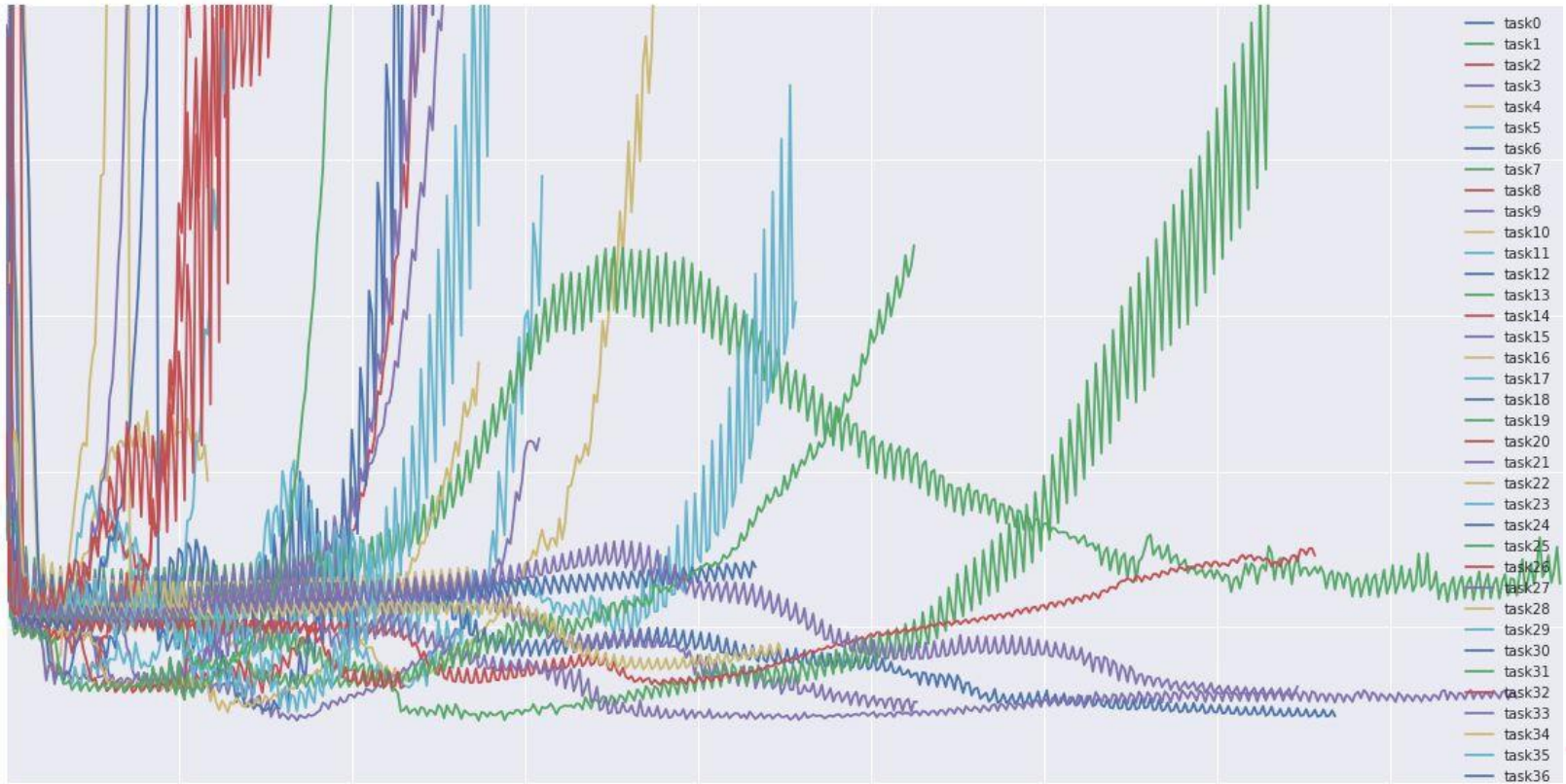


Train / Val Accuracy



Losses may be noisy, use a scatter plot and also plot moving average to see trends better

You can plot all your loss curves for different HPs on a single plot



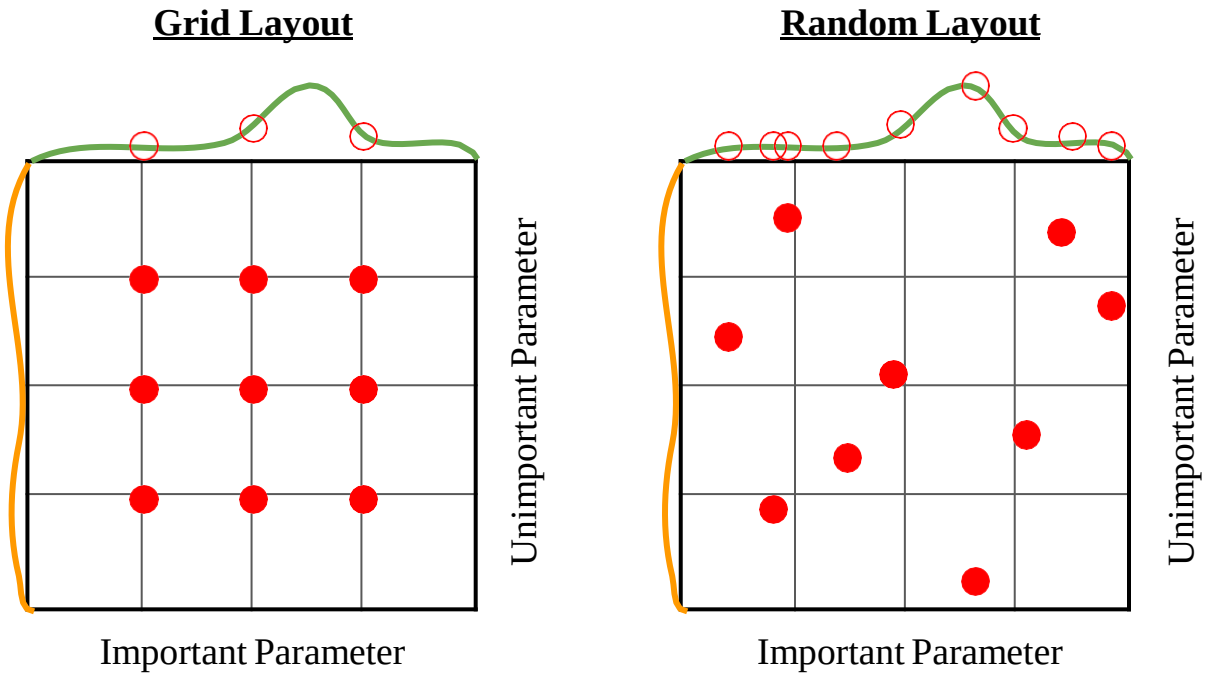
Don't look at accuracy or loss curves for too long!

Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at loss and accuracy curves
Step 7: GOTO step 5

Random Search vs. Grid Search

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012



Summary

- Improve your training error:
  - Optimizers
  - Learning rate schedules
- Improve your test error:
  - Regularization
  - Choosing Hyperparameters

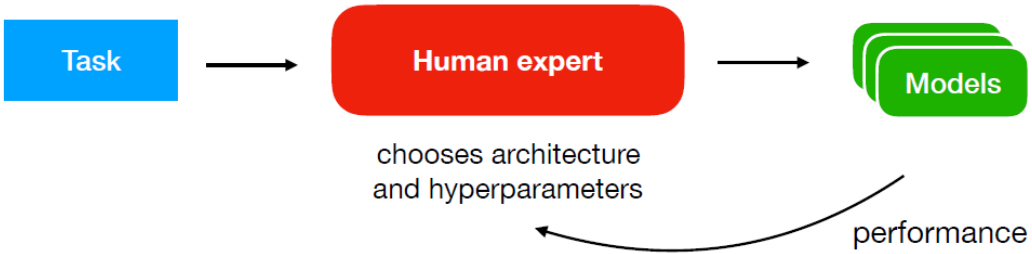
Hyperparameter Tuning

Automated ML

Classical ML Pipelines

Human experts choose ML models and set their HPs.

For instance, in assignment 1 you choose a regression **model** like decision tree and then decide its **hyperparameter**, e.g., tree depth.



The key elements here include (i) **model** and (ii) **hyperparameter.**

Classical ML Pipelines: Algorithms

The key elements in ML pipelines include (i) **algorithms** and (ii) hyperparameters.

Algorithms:

1. *Data preprocessing algorithms*, e.g., dimensionality reduction like PCA
2. *Classical ML algorithms*, e.g., kNN, logistic regression, random forests, etc.
3. *Modern ML algorithms* (a.k.a, deep learning), e.g., RNN, LSTM, and CNN

Classical ML Pipelines: Hyperparameters

Once an algorithm is chosen, we need to decide its associated **HPs**:

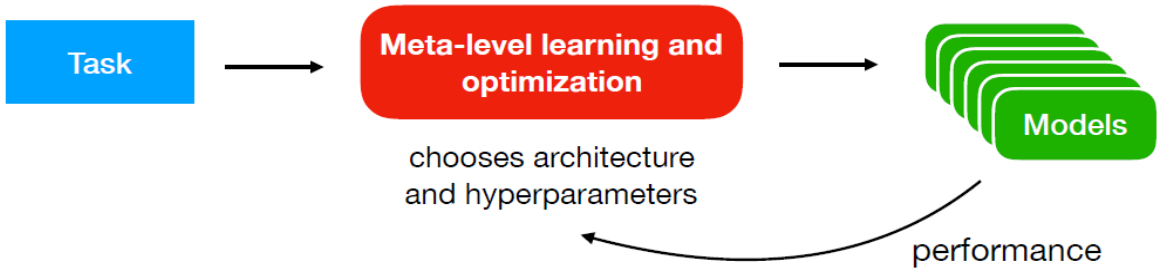
1. Data preprocessing algorithm, e.g., dimensionality reduction like PCA (**how many principal components**? i.e., what is the target dimension?)
2. Classical ML algorithms, e.g., kNN (**# nearest neighbors**), random forests (**# base trees**), etc.
3. Modern ML algorithms, like deep learning algorithms (e.g., **learning rate** and **dropout rate**)

We will revisit these definitions in a few slides…

How will you choose algorithm/hyperparameters for a given task?

Automated ML (AutoML) & Key components

The key of AutoML is to automate the selection process of algorithms and hyperparameters.



Unlike classical ML models, deep learning may have more flexible architectures, so people typically categorize AutoML into ***three*** buckets:

1. **HP tuning** and/or **algorithm selection: for a single dataset (covered in this lecture)**
2. Meta Learning (learning to learn): generalize to many different datasets
3. Neural architecture search (may be considered as a subarea/combination of 1&2)

AutoML Use Cases

Google Cloud AutoML: "Cloud AutoML is a suite of machine learning products that enables developers with limited machine learning expertise to train high-quality models specific to their business needs."

AutoML Vision, *AutoML Video Intelligence*, *AutoML Natural Language*, *AutoML Translation*

AutoML Tables: "Automatically build and deploy state-of-the-art machine learning models on structured data."

Do you know any use cases of AutoML in your daily study/work? How would you leverage this technique to make your life easier?

Warning: Should We Always Use AutoML?

AutoML is not necessarily superior to human selection for many reasons:

- Human selection (regarding algorithm and hyperparameter) is indeed based on prior knowledge and historical information of the problem.
- It is not always possible to perform AutoML due to software and hardware limitations, especially the cost. AutoML is often costly to run.

Alternatives to AutoML:

- One may use stable & robust methods like ***ensemble learning*** to combine multiple models together. For instance, one may not know how to select the ML algorithms and hyperparameters. It is possible to randomly build multiple models with different hyperparameters and then combine them, e.g., taking the average.
    - ***Drawbacks***: building many diversified models may be slow for prediction.

Preliminaries: Algorithm & Hyperparameter Revisit

ML Algorithms, e.g., decision tree, kNN, etc.

Hyperparameters are specific settings of a selected **algorithm**, such as the tree depth of decision tree or the number of nearest neighbors for kNN.

Models are the **combination** of **algorithms** and **hyperparameters**:

- {Decision tree, tree_depth=5}
- {Decision tree, tree_depth=10}
- {kNN, n_neighbors=5}
- {kNN, n_neighbors=10}
- …

Preliminaries: Configuration

AutoML helps us to find an "**optimal**" model in large model space.

- **Situation 1 (reduced to hyperparameter search): Same algorithm, searching for best hyperparameters**

Models are referred as the **combination** of **algorithms** and **hyperparameters**:

- {Decision tree, tree_depth=3}
- {Decision tree, tree_depth=5}
- {Decision tree, tree_depth=10}

Preliminaries: Configuration

AutoML helps us to find an "**optimal**" model in large model space.

- **Situation 2 (reduced to algorithm search): Different algorithms with their default hyperparameters, searching for the best model (merely algorithm)**

Models are referred as the **combination** of **algorithms** and **hyperparameters**:

- {Decision tree, default hyperparameters}
- {kNN, default hyperparameters}
- {Logistic regression, default hyperparameters}
- …

Preliminaries: Configuration

AutoML helps us to find an "**optimal**" model in large model space.

- **Situation 3: Different algorithms with varying hyperparameters, searching for best model+ associated algorithm(s)**

Models are referred as the **combination** of **algorithms** and **hyperparameters**:

- {Decision tree, tree_depth=5}
- {Decision tree, tree_depth=10}
- {kNN, n_neighbors=5}
- {kNN, n_neighbors=10}
- …

Algorithms and Hyperparameters

Most of the methods introduced in this lecture can be used for **model search**, although they may be put under hyperparameter search or algorithm search.

1. You may think in this way, **the selection of algorithms (kNN, Decision Tree, etc.) can also be viewed as a "hyperparameter" of the ML pipeline**.
2. Selecting an algorithm and tuning its corresponding hyperparameters uses human knowledge (priors) to fix the algorithm, where the search space is restricted.

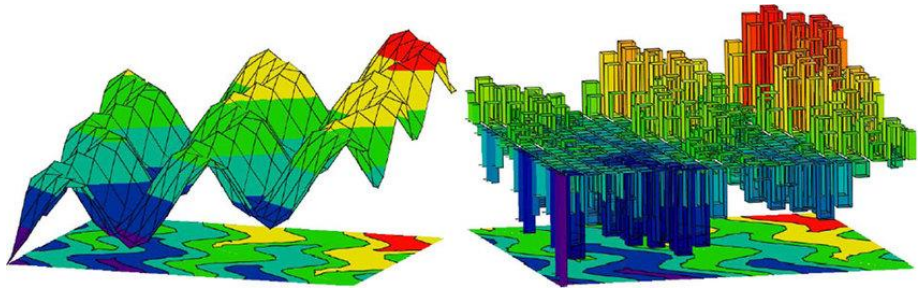
Why do we want to do hyperparameter search only?

1. The search space is smaller, so the optimization is easier
2. The diff between algorithms can be huge. If the algorithm choice is included, **the search process will be less smooth**, especially for learning-based algorithms.

Hyperparameter Search w/ a More Smooth Objective

Why do we want to do hyperparameter search only?

1. The search space is smaller, so the optimization is easier
2. The difference between algorithms can be significant. If the algorithm choice is included, **the search process will be less smooth**, especially for learning based algorithms.



Smooth(er) objective vs. nonsmooth objective

Part 1: Given a Dataset and an Algorithm, how to choose the best hyperparameter(s)?

This is the most common case we encounter in our daily work. In practice, **most real-world applications focus on one or just a (few) dataset(s)** and try to find the **hyperparameters** that maximize the prediction accuracy of a given algorithm.

1. **Random and Grid search**: if cheap evaluation is possible
2. **Bayesian optimization**: if evaluation is expensive, and the optimization is restricted

Other than these approaches, have you tried any other algorithms?

Hyperparameter Optimization (HPO)

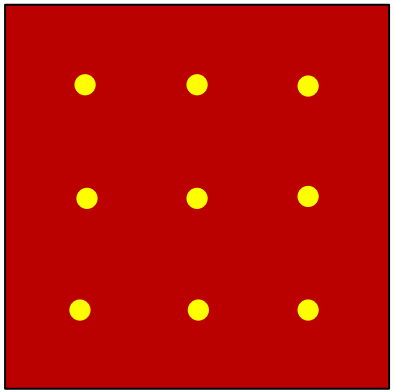
Problem definition: given a selected algorithm, how to choose its hyperparameter to achieve the best performance (accuracy, F1, etc.) on the training/validation set.

1. $\{X_{train}, X_{test}\}$, and the corresponding $\{y_{train}, y_{test}\}$.
2. The chosen model f, i.e., ridge regression, with tunable hyperparameters:
    1. *alpha*: Regularization strength; must be a positive float
    2. *max_iter*: Maximum number of iterations for conjugate gradient solver.

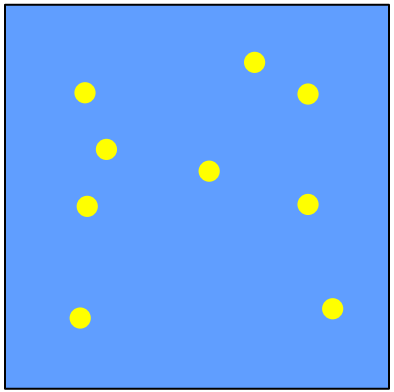
The goal is to only use $\{X_{train}, y_{train}\}$ (and possibly $\{X_{valid}, y_{valid}\}$) to select {*alpha**, *max_iter**} so that the best accuracy of f is achieved on $\{X_{test}, y_{test}\}$, where train and test data are assumed from the same data distribution.

HPO: Grid Search & Random Search

The most straightforward way is to evaluate many different combinations of hyperparameters, either using **grid search (left) with a "plan"** or **random search (right). Output the best one!**



Grid Search



Random Search

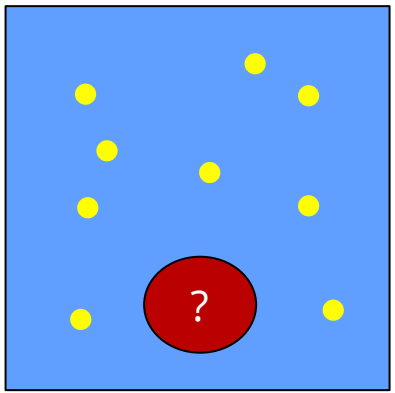
What are the advantages of grid search and random search?

HPO: Grid Search & Random Search

Grid Search: not all the hyperparameters are equally important. Grid search may overemphasize the importance of "unimportant hyperparameters".

Fix: decrease the granularity of unimportant hyperparameters

Random Search: its heuristic does not guarantee a good coverage of the searching space.



The red area is unexplored in random search

Summary: Grid Search & Random Search

Advantages:

1. Easy to set up
2. Appropriate when you have no knowledge of the hyperparameter space
3. It is a **parallel** process with no dependency, which can be efficiently parallelized

Disadvantages:

1. There is no learning process, but pure model evaluation (computation)
2. Can be extremely expensive when you have many hyperparameters. Given k hyperparameters with 2 possible values, it has exponential search space $O(2^k)$. In practice, most HPs are continuous, leading to an intractable computations.
3. Only select from the predefined space!

Takeaway: most of time, this should be sufficient for our daily jobs.

Summary: Grid Search & Random Search (advanced)

Speed-up scikit-learn based hyperparameter tuning:

1. Multi-thread: using n_jobs option in scikit-learn, see here

Even faster:

1. Distributed scikit-learn via dask: https://ml.dask.org/joblib.html

Fix the Limitations of Grid and Random Search

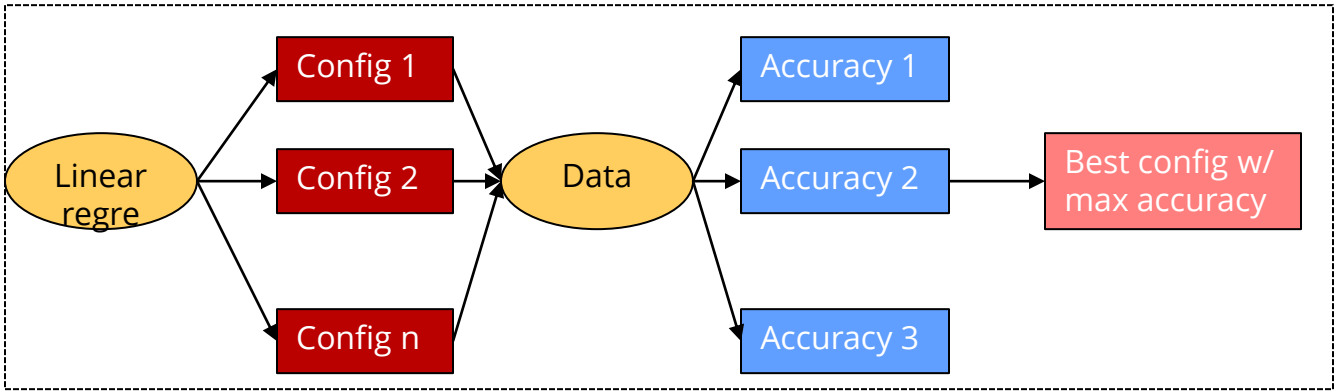
1. If trying different hyperparameters is expensive, **how to reduce the number of evaluations?**
2. **How to learn from the process?** Note grid and random search do not involve learning but pure evaluation.

How to do intelligent "searching" or hyperparameter tuning?

Fix the Limitations of Grid and Random Search

The figure below shows what we are doing in the grid and random search, i.e., trying different **hyperparameter configuration (this is changing)** on the **linear regression** and **the data (these are fixed)**, and evaluate the accuracy.

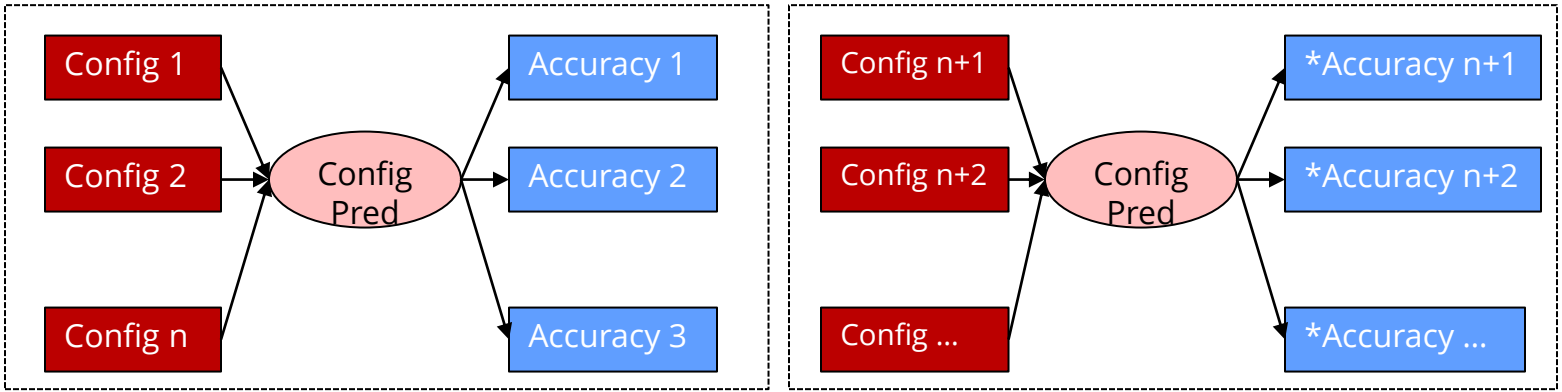
Fitting on the data is expensive! How to prevent this?



Fix the Limitations of Grid and Random Search

Prediction-based hyperparameter tunning:

1. First do some evaluations as before, and then build a **predictor** to forecast the performance of any hyperparameters!
    1. **learn** the relationship between the **hyperparameter configs** and the **performance**!
    2. We are no longer constrained in a fixed hyperparameter space (prediction is cheap)!



Prediction-based Hyperparameter Tunning

A few things to consider:

1. We still need to do some hyperparameter evaluation in the beginning, and we would expect the number of evaluations as small as possible.
2. **Bottom line: # configuration evaluation** should not be more than that of grid and random search, while the final selected hyperparameter performance should be better!
3. Can we **gradually** build this up and find a nice hyperparameter?
    1. Train the predictor in an iterative way
    2. Train the predictor in a smart way

Prediction-based Hyperparameter Tunning

Sequential prediction-based hyperparameter tuning:

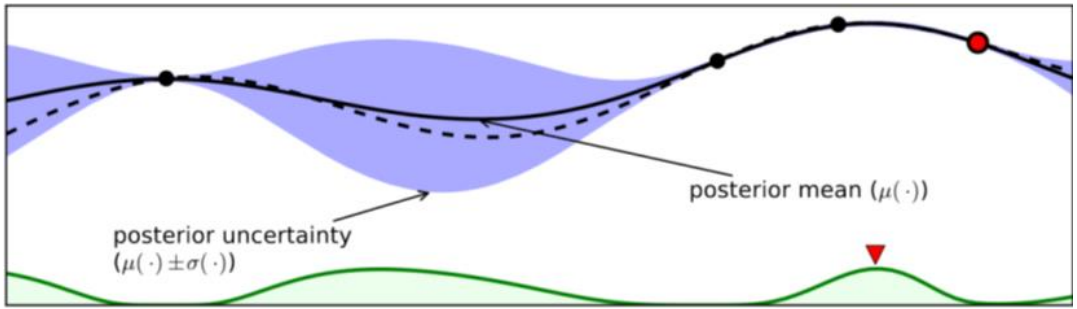
1. Randomly select a small number of hyperparameters $\boldsymbol{C_s}$, and evaluate their accuracy $\boldsymbol{acc_s}$
2. Train a predictor p to map $\boldsymbol{C_s}$ to $\boldsymbol{acc_s}$, i.e., $p(\boldsymbol{C_s}) \coloneqq \boldsymbol{acc_s}$
3. **While not converged:**
    1. Sample a large number of new hyperparameters $\boldsymbol{C_i}$ randomly
    2. Use p to predict **the best** hyperparameter in $\boldsymbol{C_i}$: $\boldsymbol{c^*} \coloneqq \text{argmax}_{\boldsymbol{C_i}} \boldsymbol{p}(\boldsymbol{C_i})$
    3. Evaluate the accuracy of the current best hyperparameter $\boldsymbol{c^*}$, i.e., $\boldsymbol{acc_{c^*}}$
    4. Retrain p with new information $\{C_s \cup c^*, acc_s \cup acc_{c^*}\}$
4. Once converged, use p to predict which hyperparameter has the highest accuracy

we **iteratively** learn **a good hyperparameter performance predictor** p!

Prediction-based Hyperparameter Tunning

One more thing to consider:

- Each time we update the performance predictor p, so that the prediction result will be different in each round.
- However**, it is likely the predictor does not change and always give you the same result!**
- This observation tells us, while selecting the next best hyperparameter to evaluate, we need to consider **how accurate it might be (accuracy)** and **how uncertain (so we do not get trapped)** it is!



Prediction-based Hyperparameter Tunning

To predict both the **accuracy** and the **uncertainty/variation** of a new hyperparameter, we need specialized ML models for this! Some algorithms can estimate the prediction uncertainty.

1. Random forest (need additional implementation)
2. **Gaussian process** -> scikit-learn directly support it (https://scikit-learn.org/stable/modules/gaussian_process.html). We will discuss this shortly.

predict(*X*, *return_std=False*, *return_cov=False*) [source]

Predict using the Gaussian process regression model.

We can also predict based on an unfitted model by using the GP prior. In addition to the mean of the predictive distribution, optionally also returns its standard deviation (`return_std=True`) or covariance (`return_cov=True`). Note that at most one of the two can be requested.

Parameters:

X : *array-like of shape (n_samples, n_features) or list of object*
Query points where the GP is evaluated.

return_std : *bool, default=False*
If True, the standard-deviation of the predictive distribution at the query points is returned along with the mean.

Prediction-based Hyperparameter Tunning

Basically, this sequential prediction-based method **keeps finding the next most promising hyperparameter** to evaluate and **improve the prediction quality gradually**.

Finally, we could have a nice predictor between the hyperparameter to the accuracy, and we could do the prediction on the entire hyperparameter space without constraints.

Formally, this is called **Bayesian Optimization**!

Bayesian Optimization—A Learning Based Strategy

Bayesian optimization (BO) is a sequential, learning-based hyperparameter optimization method. It is especially usefully when we have no explicit expression of the objective function, so traditional optimization, e.g., gradient descent, does not work.

Quick example, Bayesian optimization has been widely used in some **human-in-the-loop applications**. For instance, use the current model to generate an image and then submit to human annotators to score its quality. Based on the feedback from the human annotator, the model hyperparameters can iterate.

Another example provided by the professor is his students are using BO to test different trading strategies. BO can be useful for financial market.

Can you come up with use cases for Bayesian Optimization?

HPO: Bayesian Optimization

Bayesian optimization (BO) is often used when:

1. **objective function f is costly to evaluate**, such as tuning a set of hyperparameter for expensive deep learning models. For instance, there is no explicit formula for evaluation.
2. **the objective function f does not have a simple way of gradient calculation** that involves first and second order derivatives. We can observe the value of f only, but not its explicit form.
3. *f does not have a known structure of concavity or linearity that we may use mature optimization methods to solve it.
4. *The hyperparameter space is not in the high-dimensional space. As a rule of thumb, BO is generally good for fewer than 20 hyperparameters' search

In short, BO can be used when the objective function is a "blackbox" and no explicit optimization can be used. **Any questions so far?**

HPO: Bayesian Optimization

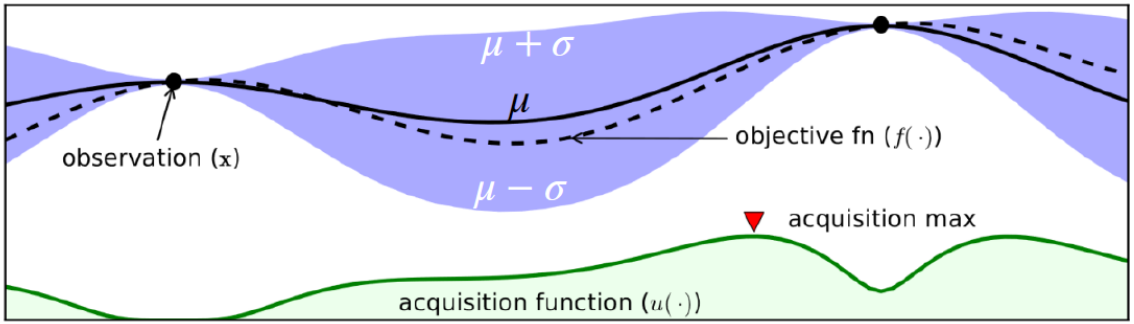
Bayesian optimization (BO) is a sequential, learning-based hyperparameter optimization method.

1. Start with a few (random) configurations, $c_1, c_2, \ldots, c_j$ (X), and we know the objective value of them (y).
2. Build a **surrogate model (this is our p)** to predict how well other configurations will work. In general, we would like to consider the configuration that can yield "high expectation" but with "some uncertainty".
3. Recall reinforcement learning, the searching may be done in an "exploration first, and exploitation later" fashion. In BO, we use **acquisition function** to balance and guide the search.
4. We choose to evaluate the most potential hyperparameter config by acquisition function.

HPO: Bayesian Optimization—Prediction and Uncertainty

Gaussian process (GP) is often chosen as the model for the acquisition function.

1. It can characterize both **predicted value (μ)** and its associated **uncertainty (σ)**
2. If we have more known pairs around a given $\{c, f_c(.)\}$, then this point has lower uncertainty.



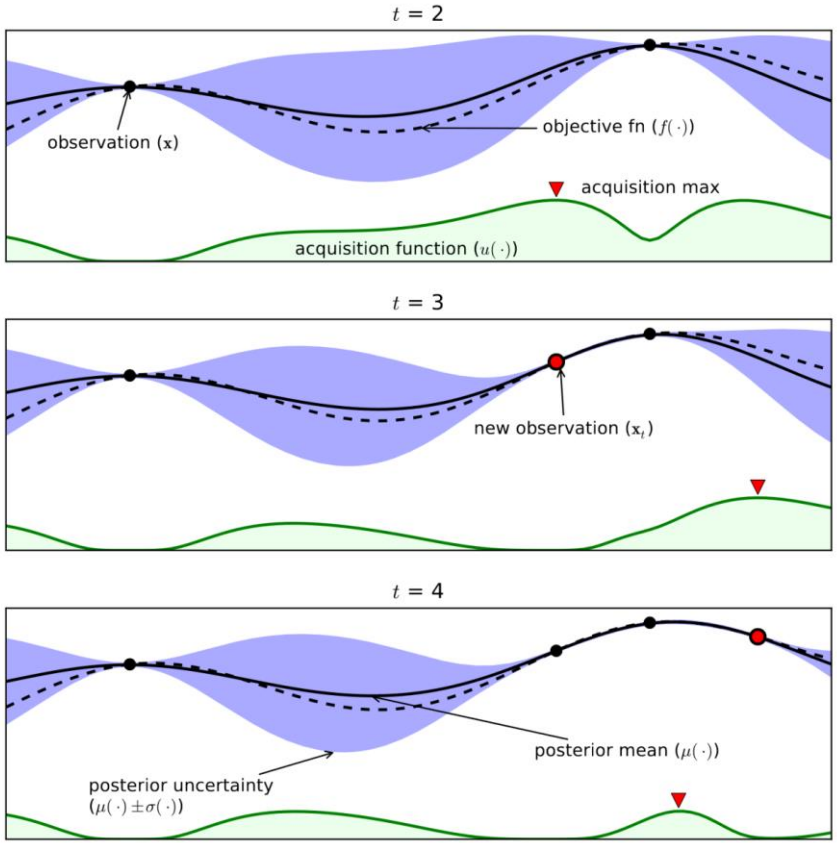
Blue band: uncertainty; black curve: objective function; dashed line: GP regressor

See http://krasserm.github.io/2018/03/19/gaussian-processes/ for a quick tutorial.

HPO: Bayesian Optimization

Bayesian optimization (BO) is a sequential, learning-based hyperparameter optimization method. At t=2,

1. we know the actual value of two pairs of $\{c, f_c(.)\}$. Here c is the hyperparameter combination, and $f_c(.)$ is the model accuracy under c
2. Fitting acquisition function with these two pairs, build an acquisition function a
3. **Randomly generate m configurations** which are not explored yet
4. Use the acquisition function to predict each hyperparameters' expected accuracy and uncertainty



HPO: Bayesian Optimization—Selection Criteria

How to use the acquisition function to select the next hyperparameter. The popular ones:

- **upper confidence bound (UCB)**: a combination of expected value **(μ)** and the uncertainty associated **(σ)** with it.
- **Selection criteria**: $c^* = \max_c (\mu + \beta\sigma)$, β is a balance factor, like a weight.
- **Intuition**: we want to select a hyperparameter with high predicted return and some uncertainty. If we simply select the hyperparameter with high return but no uncertainty, we may be "trapped". **Recall the exploration vs. exploitation in reinforcement learning**.
- We generally encourage to **explore first** and **then exploit**. To reflect this, β can be a time-decay function $\beta = 0.8^t$. At t=1,2,3..., $\beta = 0.8, 0.64, 0.512$ .... In the later iterations, uncertainty contributes much less, and the selection is mainly based on the expected return.

HPO: Bayesian Optimization—Selection Criteria

How to use the acquisition function to select the next hyperparameter combination. The popular ones:

- **Expected Improvement (EI)**: a combination of expected value **(μ)** and the uncertainty associated **(σ)** with it.
- **Selection criteria**: maximize the expected improvement regarding the current situation
- **Intuition**: we want to select a hyperparameter which bring us most expected improvement. It also has an exploration-exploitation taste: $\mathrm{EI}(c) = (\mu - a(c^{+}) - \delta)\Phi(\mathrm{Z}) + \sigma\Phi(\mathrm{Z})$
- Again, the first part corresponds to expected return and the second part corresponds to uncertainty. See http://krasserm.github.io/2018/03/21/bayesian-optimization/ for more information.

HPO: Bayesian Optimization Summary

Bayesian optimization (BO) is a sequential, learning-based hyperparameter optimization method. It is especially usefully when we have no explicit expression of the objective function, so traditional optimization, e.g., gradient descent, does not work.

Quick example, Bayesian optimization has been widely used in some human-in-the-loop applications. For instance, use the current model to generate an image and then submit to human annotators to score its quality. Based on the feedback from the human annotator, the model hyperparameters can iterate.

Stability concerns: one problem associated with BO is that the learning process may not be stable. For instance, the annotators in the example have varying expertise in scoring. To stabilize the process, the update of the Gaussian process regressor can happen in batches.

Resource: https://machinelearningmastery.com/what-is-bayesian-optimization/

Part 2: Given many datasets, how to learn a model to select model for datasets?

Meta-learning focuses on learning to learn. There are generally two "definitions" of meta-learning.

One is under the deep learning context: https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html; we will not cover it today.

Another goal of more classical meta-learning is to **learn a general model that can select model for arbitrary datasets**.

Meta-learning: Problem Formulation

Given we have:

1. **A collection of historical datasets**: $D_{train} = \{D_1, D_2, \dots, D_n\}$. For each dataset, we have the full information with both X and y: $D_i = (X_i, y_i)$
2. **A collection of configurations** $M = \{M_1, M_2, \dots, M_m\}$, and we know their performance on each of the n datasets
3. **The performance matrix** $P \in R^{n \times m}$ space, P_{ij} refers to the jth configuration's performance on the ith dataset.

	kNN	LR	NN	SVM
Data1	0.8	0.7	0.3	0.2
Data2	0.5	1	0.6	0.9
Data3	0.3	0.5	1	0.1

For a new dataset D', how to recommend/select a configuration from M with "best" performance?

Meta-learning: Challenges

For a new dataset D', how to recommend/select a configuration from M for "best" performance?

In other words, we try to learn a model so it can recommend/select from the predefined configuration pool. **For an arbitrary dataset, the selected configuration is expected to have high performance** .

There are various challenges regarding this problem:

1. **Data shape inconsistency**: all datasets may have different number of samples and number of features.
2. **It is not always possible to conduct evaluation**. For clustering and outlier detection, which are generally unsupervised, there are no way to do iterative learning. We just have "one shot" to select the best configuration.

P3: Neural Architecture Search

Neural nets may be considered as a computational graph, and most of time it is sequential. For **Neural architecture search (NAS)**, we are trying to decide:

1. Number of layers
2. Types of layers: dense, conv, pooling,
3. Hyperparameters of layers
4. Branching/join/skips

Approaches: the classical HPO algorithms may still be possible, but faces the issue with dimensionality.

Current trend: leverage reinforcement learning in this setting.

AutoML Tools and Benchmarks

There are a few tools falling under the category of AutoML:

- AutoWEKA: https://www.cs.ubc.ca/labs/beta/Projects/autoweka/
- Auto-Sklearn: https://automl.github.io/auto-sklearn/master/
- H2O: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html
- TPOT: http://epistasislab.github.io/tpot/

Observations:

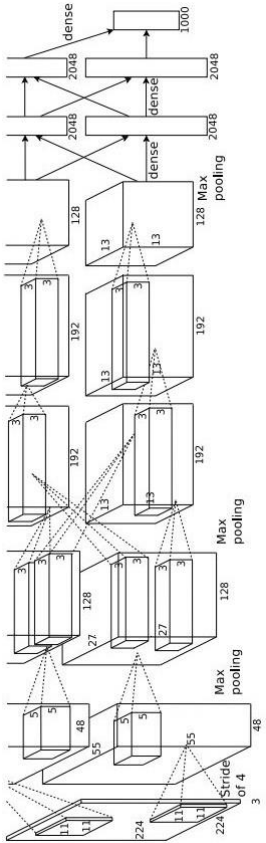
1. No one tool always outperforms.
2. They may face issues under high number of classes, and unbalanced classes.
3. The good benchmark for NAS is absent

Resources: Automated Machine Learning (open access); AutoML.org

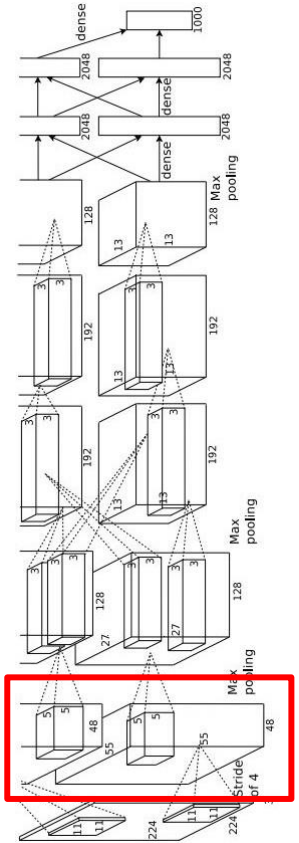
Beyond Training Neural Networks

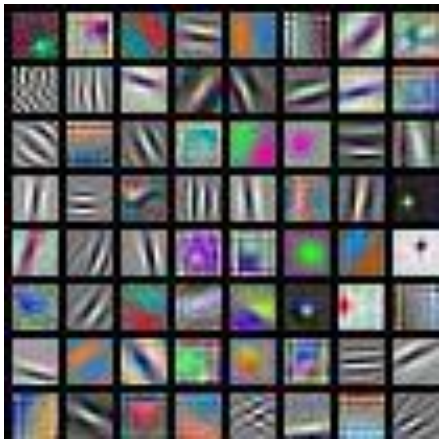
Transfer Learning

Transfer Learning with CNNs



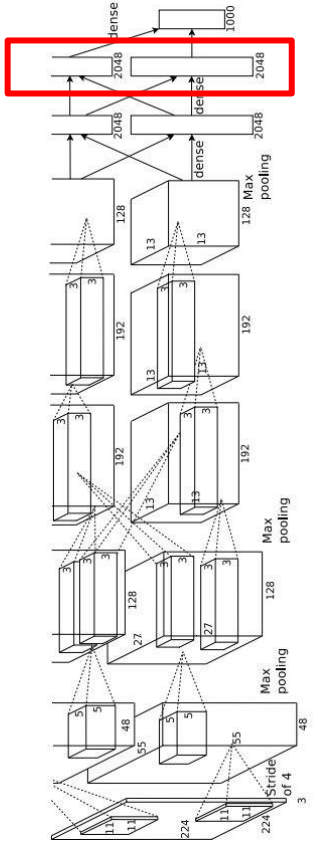
Transfer Learning with CNNs





AlexNet:
64 x 3 x 11 x 11

Transfer Learning with CNNs



Test image

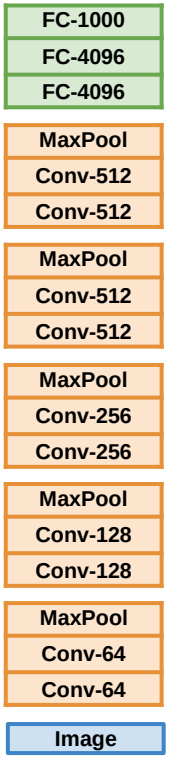
L2 Nearest neighbors in <u>feature</u> space

USC Viterbi School of Engineering

Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

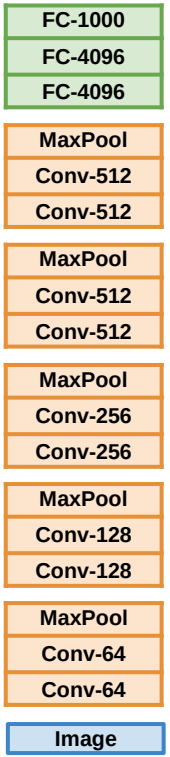
1. Train on Imagenet



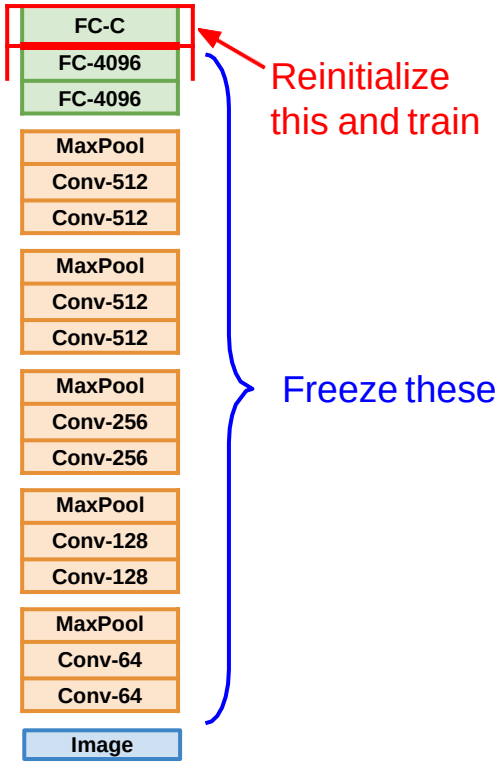
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet



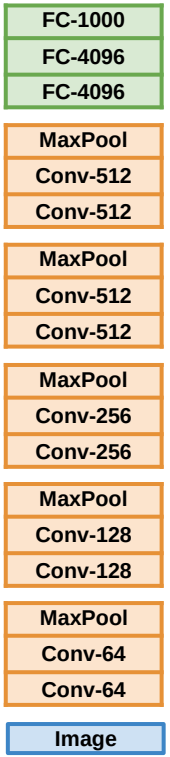
2. Small Dataset (C classes)



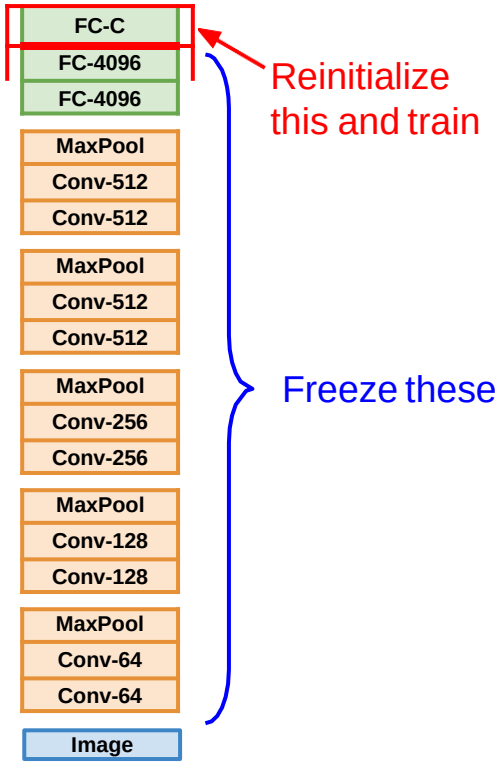
Transfer Learning with CNNs

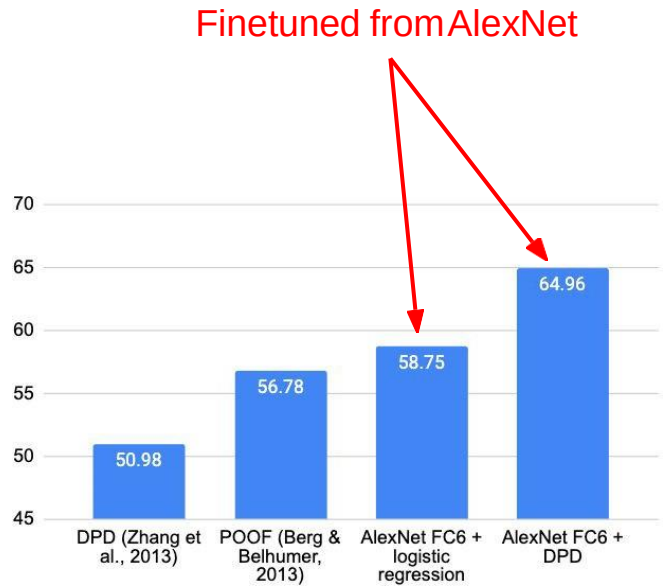
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet



2. Small Dataset (C classes)





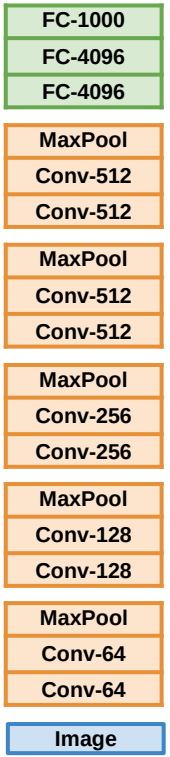
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

USC Viterbi School of Engineering

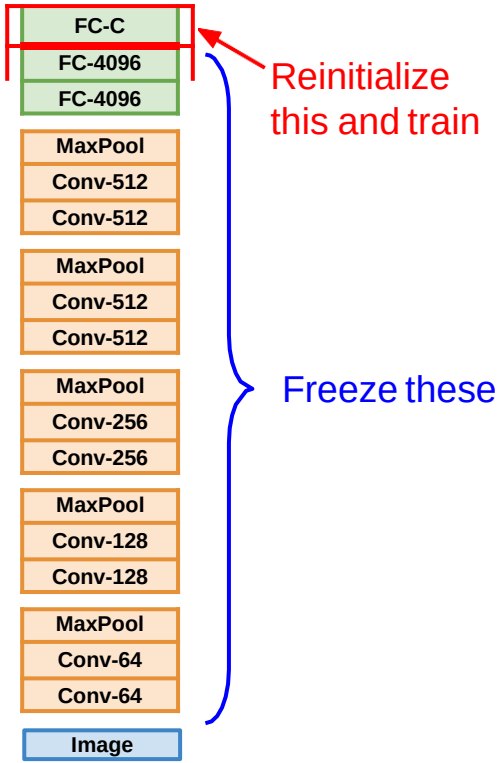
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

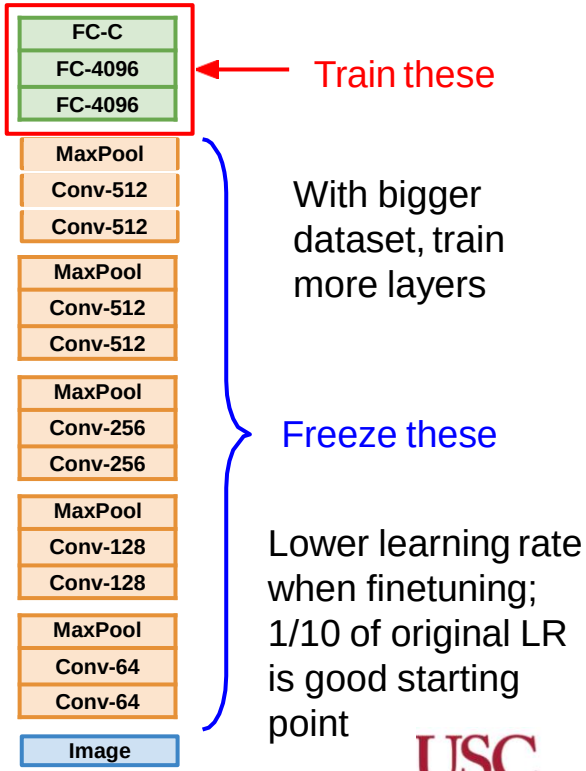
1. Train on Imagenet

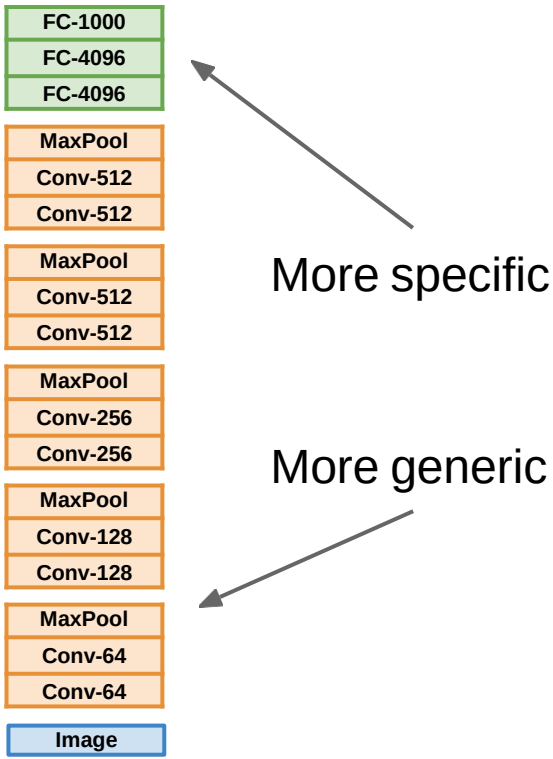


2. Small Dataset (C classes)

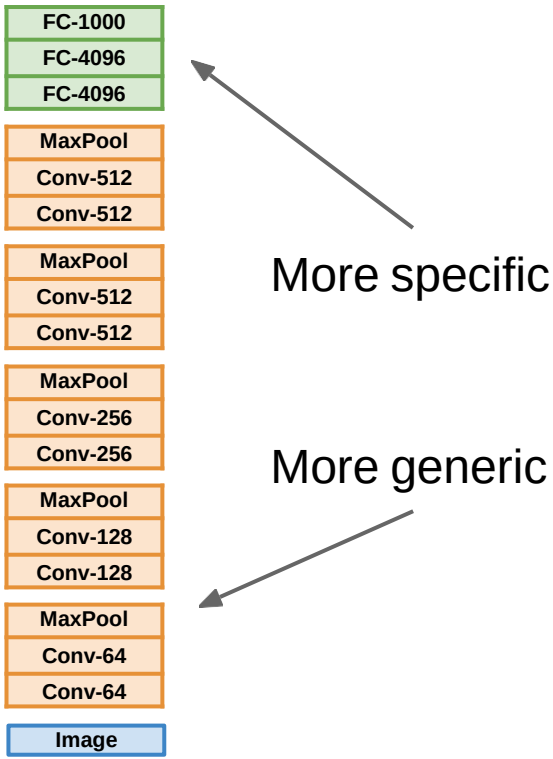


3. Bigger dataset

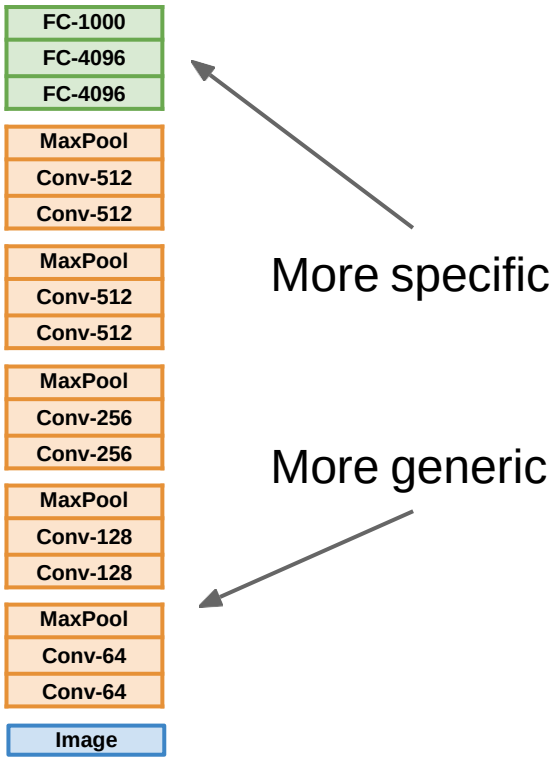




	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?



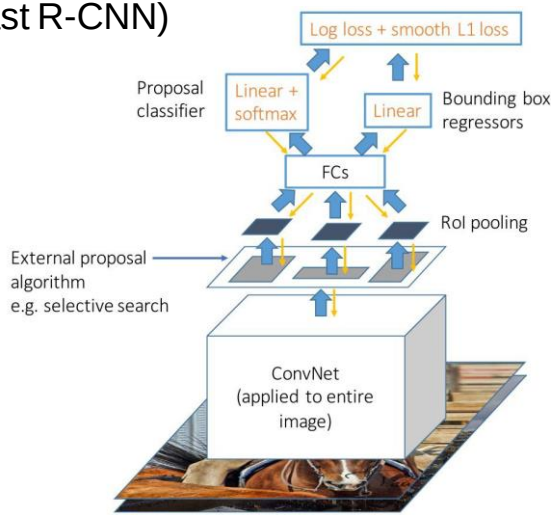
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

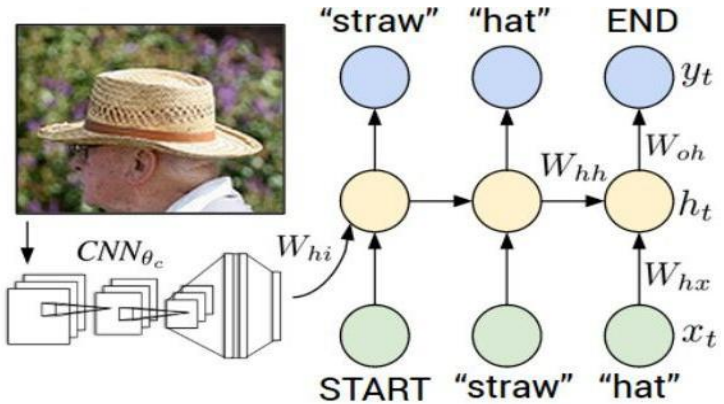
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Image Captioning: CNN + RNN



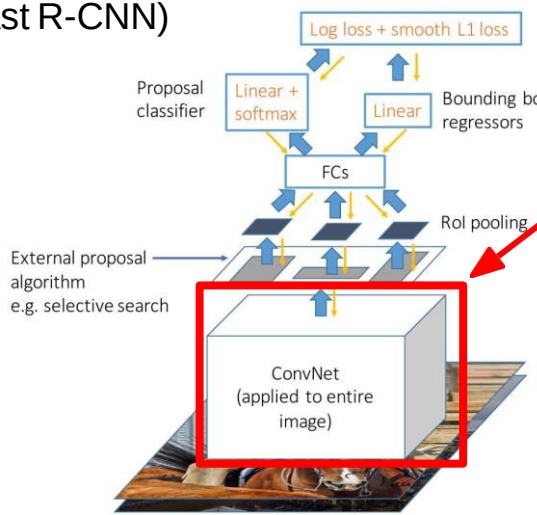
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

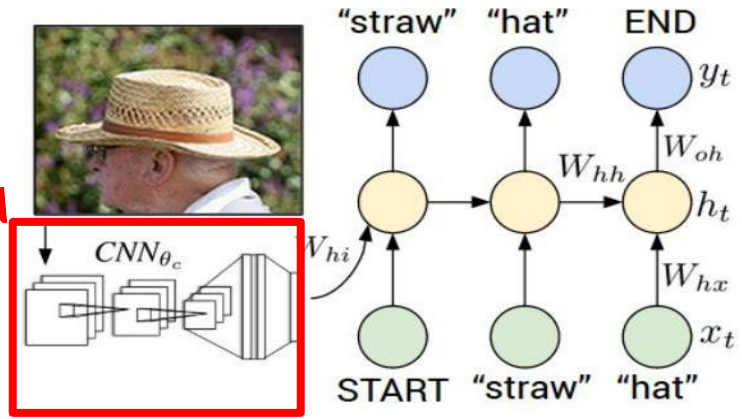
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)

CNN pretrained
on ImageNet

Image Captioning: CNN + RNN



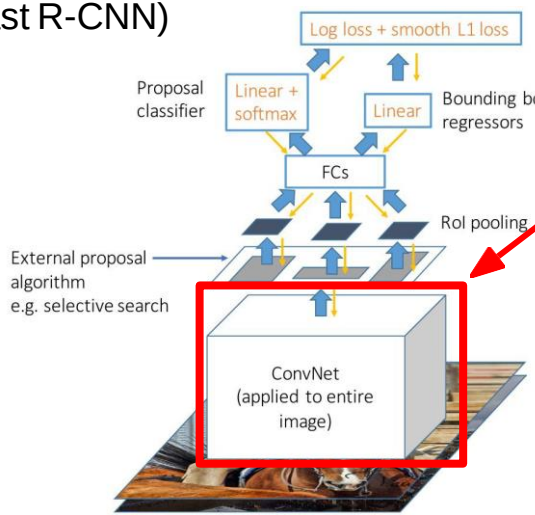


Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

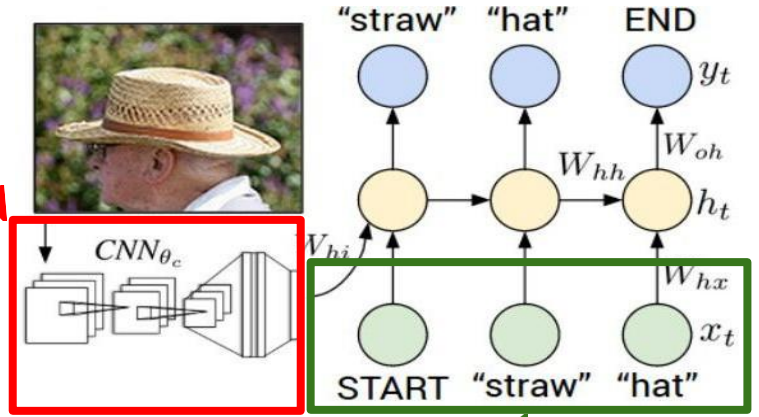
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN

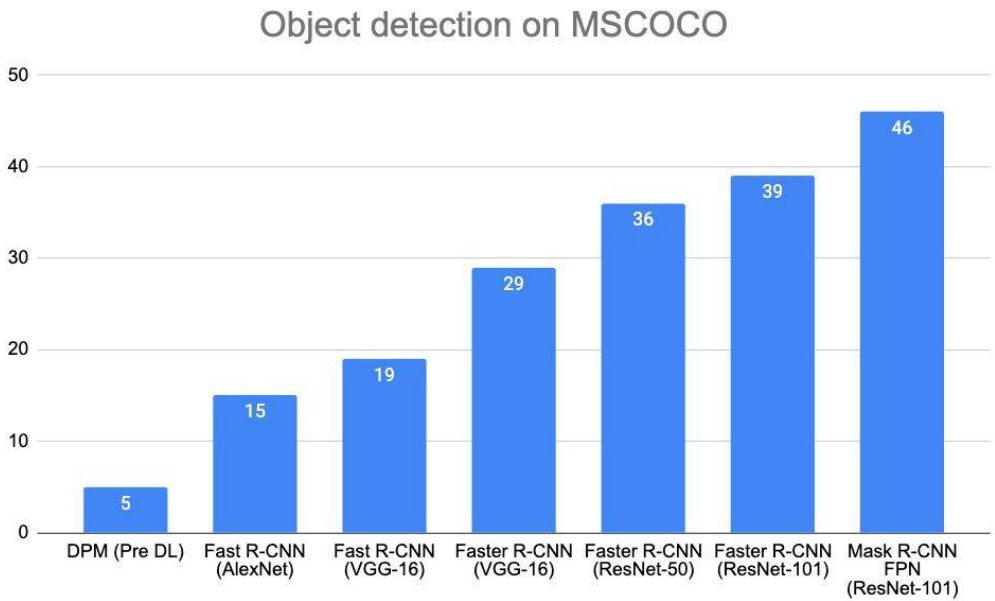


Word vectors pretrained with word2vec

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

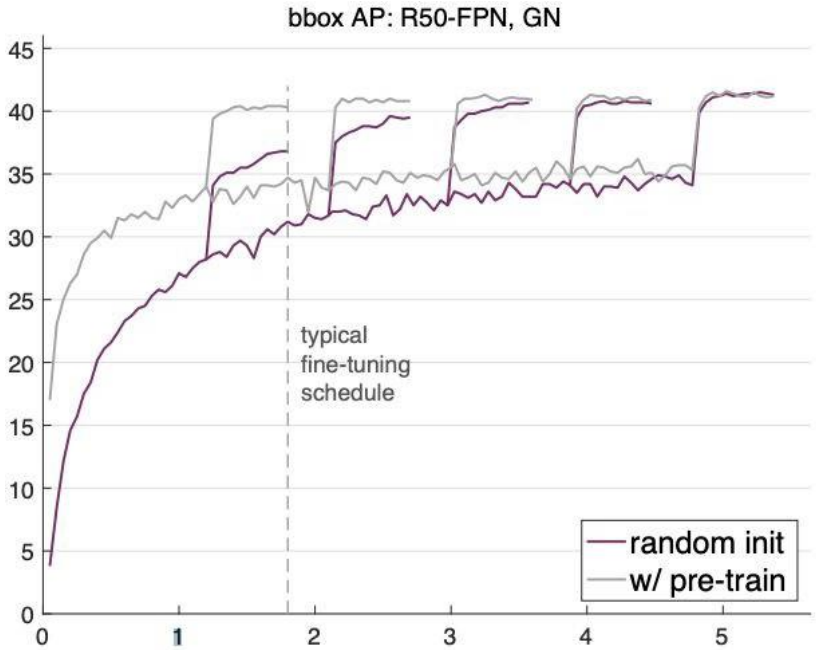
Transfer learning with CNNs - Architecture matters



Girshick, "The Generalized R-CNN Framework for Object Detection", ICCV 2017 Tutorial on Instance-Level Visual Recognition

Transfer learning with CNNs is pervasive…

But recent results show it might not always be necessary!



He et al, "Rethinking ImageNet Pre-training", ICCV 2019

Training from scratch can work just as well as training from a pretrained ImageNet model for object detection

But it takes 2-3x as long to train.

They also find that collecting more data is better than finetuning on a related task

Takeaway for your projects and beyond:

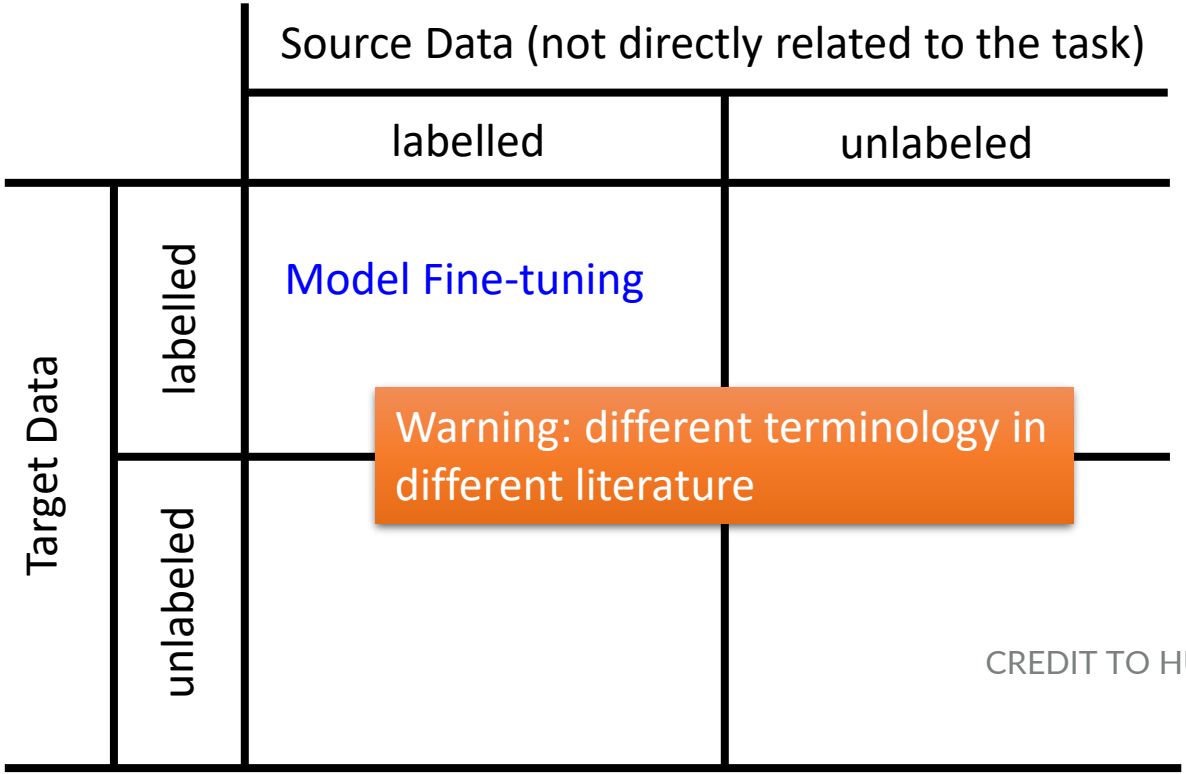
Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision

Transfer Learning - Overview



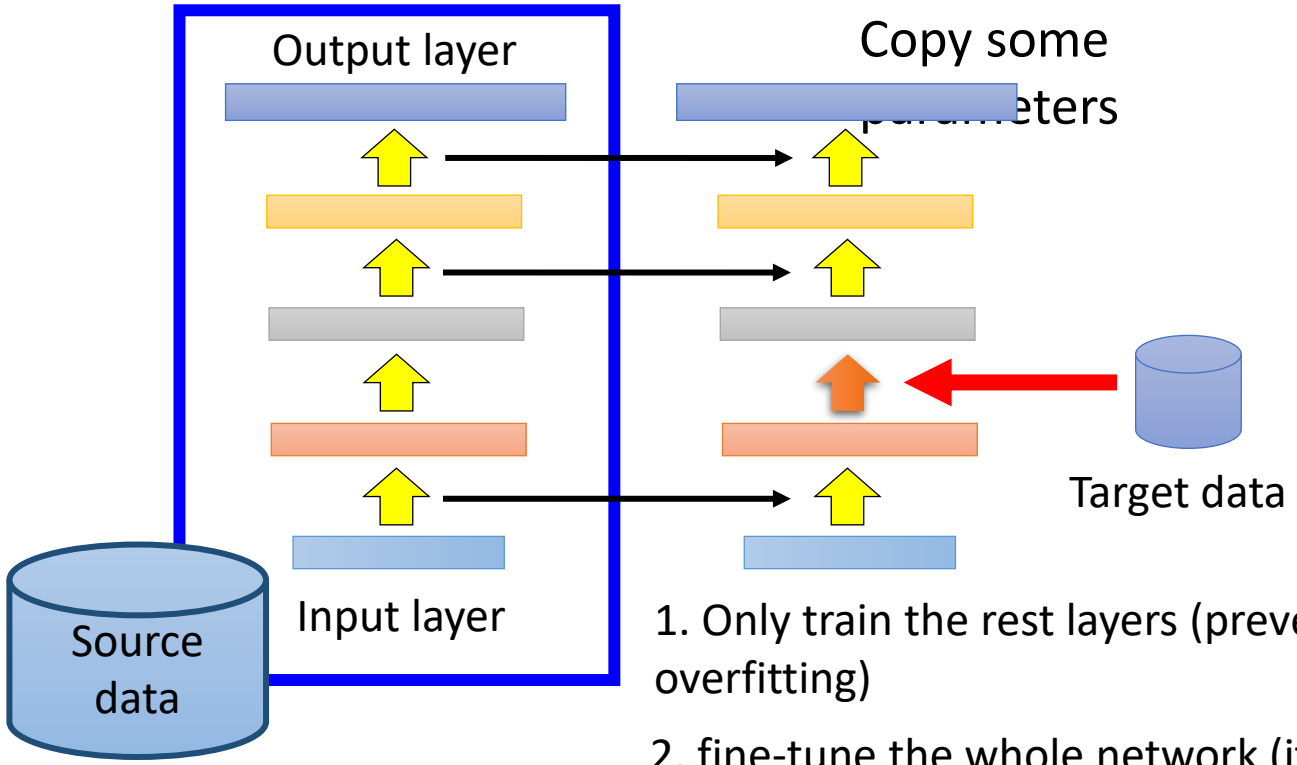
CREDIT TO HUNG-YI LEE@NTU

Model Fine-tuning

One-shot learning: only a few examples in target domain

- Task description
  - Source data: (x^s, y^s) ⬅ A large amount
  - Target data: (x^t, y^t) ⬅ Very little
- Example: (supervised) speaker adaption
  - Source data: audio data and transcriptions from many speakers
  - Target data: audio data and its transcriptions of specific user
- Idea: training a model by source data, then fine-tune the model by target data
  - Challenge: only limited target data, so be careful about overfitting

Layer Transfer

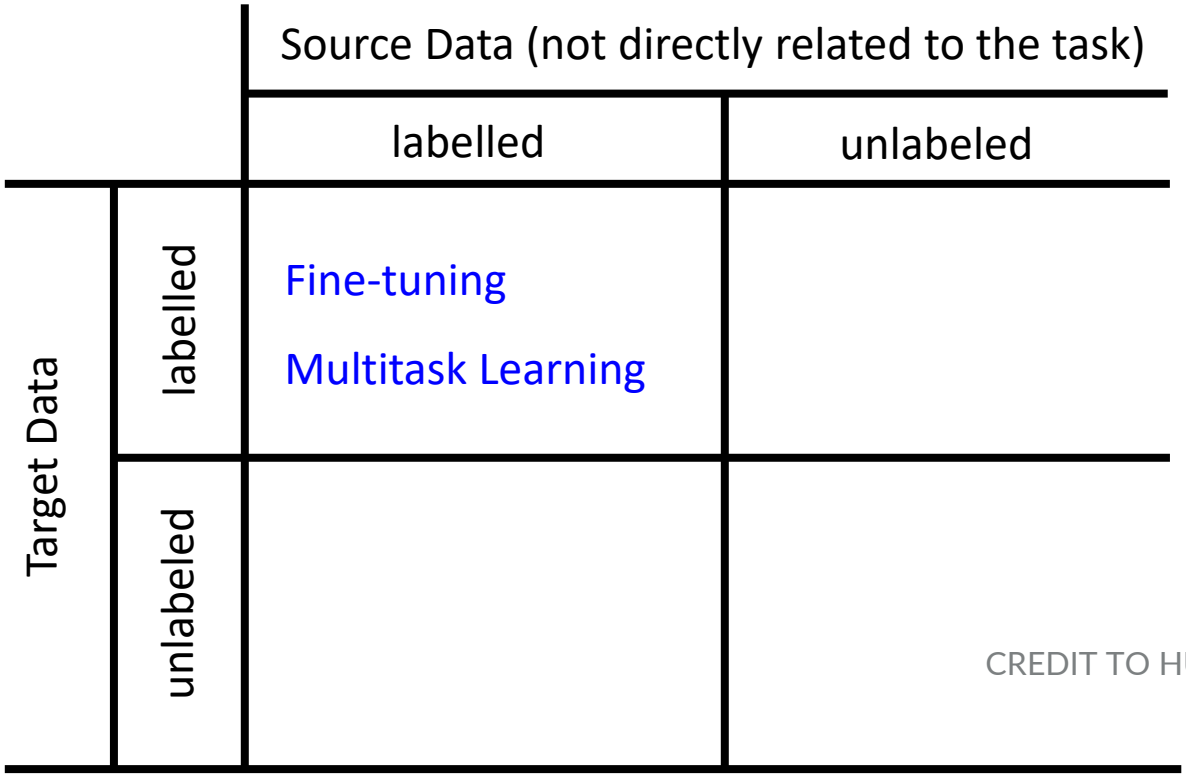


1. Only train the rest layers (prevent overfitting)

2. fine-tune the whole network (if there is sufficient data)

CREDIT TO HUNG-YI LEE@NTU

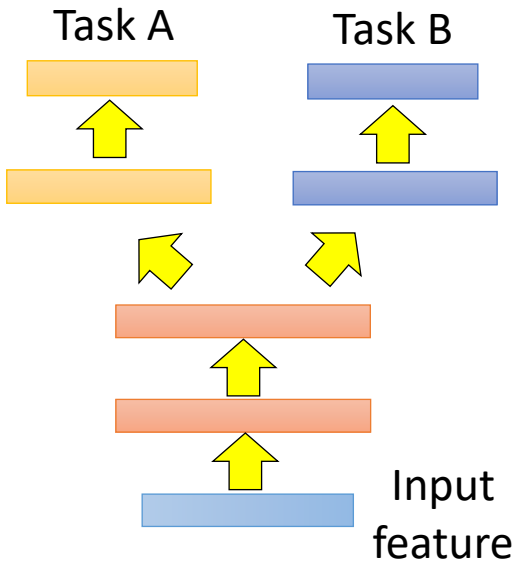
Transfer Learning - Overview

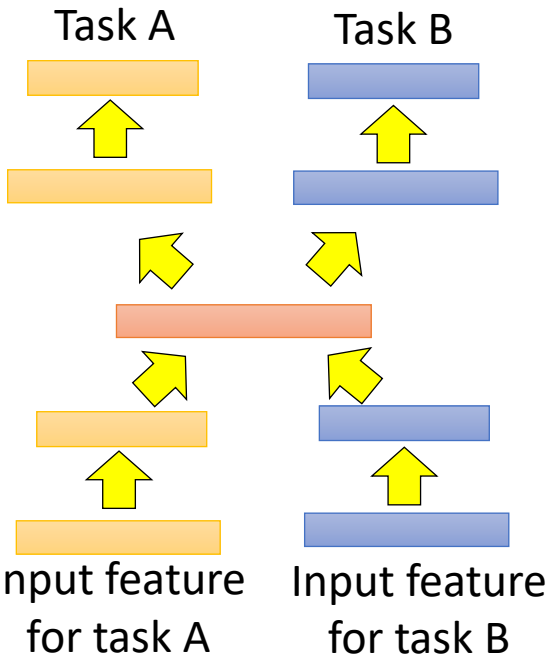


CREDIT TO HUNG-YI LEE@NTU

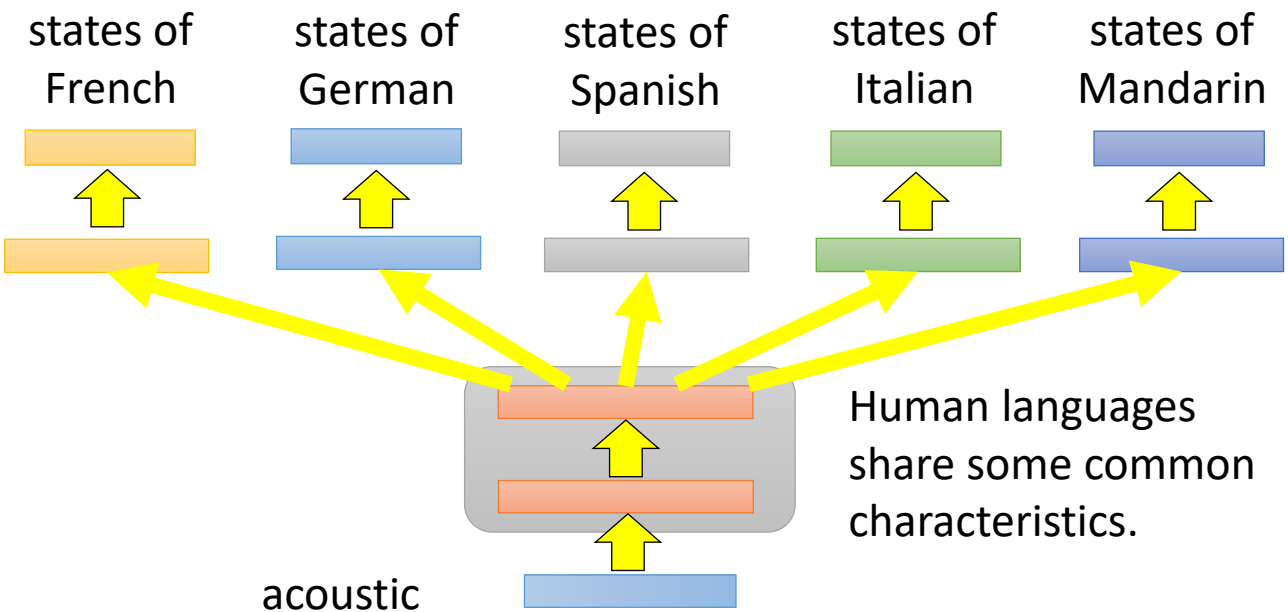
Multitask Learning

- The multi-layer structure makes NN suitable for multitask learning



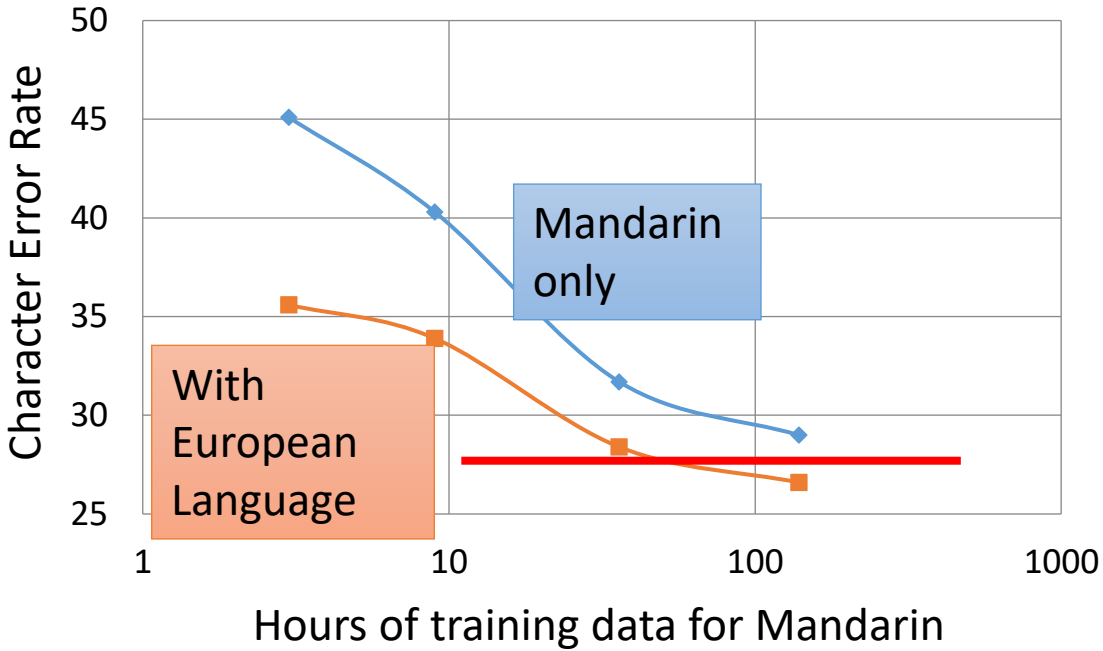


Multitask Learning
- Multilingual Speech Recognition



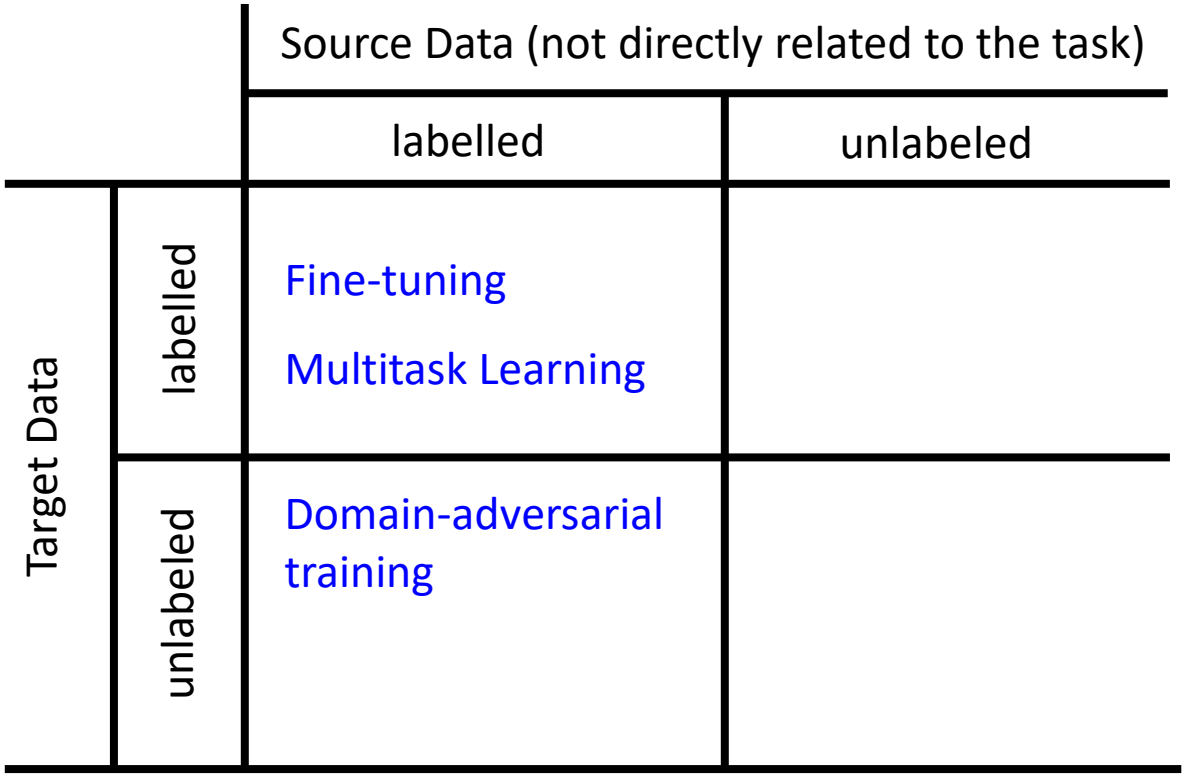
Similar idea in translation: Daxiang Dong, Hua Wu, Wei He, Dianhai Yu and Haifeng Wang, "Multi-task learning for multiple language translation.“, ACL 2015

Multitask Learning - Multilingual



Huang, Jui-Ting, et al. "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers." *ICASSP, 2013*

Transfer Learning - Overview



Task description

- Source data: (x^s, y^s)
- Target data: (x^t)

→ Training data

→ Testing data

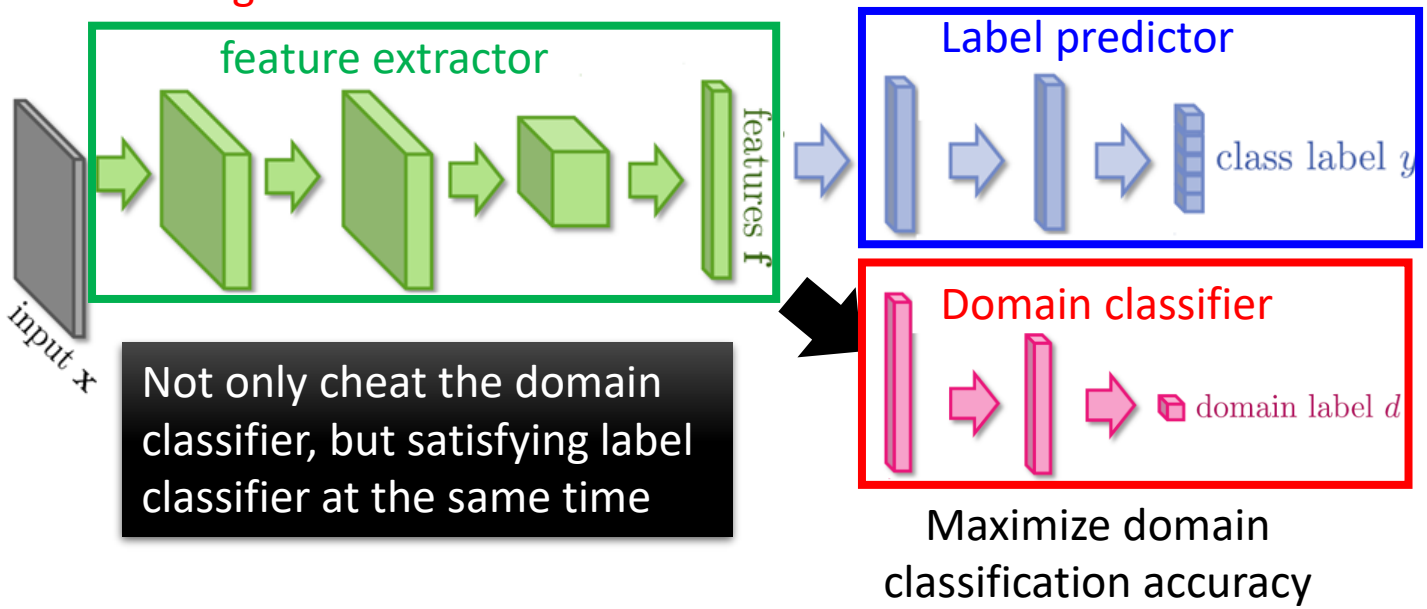
} Same task, mismatch

CREDIT TO HUNG-YI LEE@NTU

Domain-adversarial training

Maximizing label classification means the features should be indistinguishable – thus adversarial

Maximize label classification accuracy

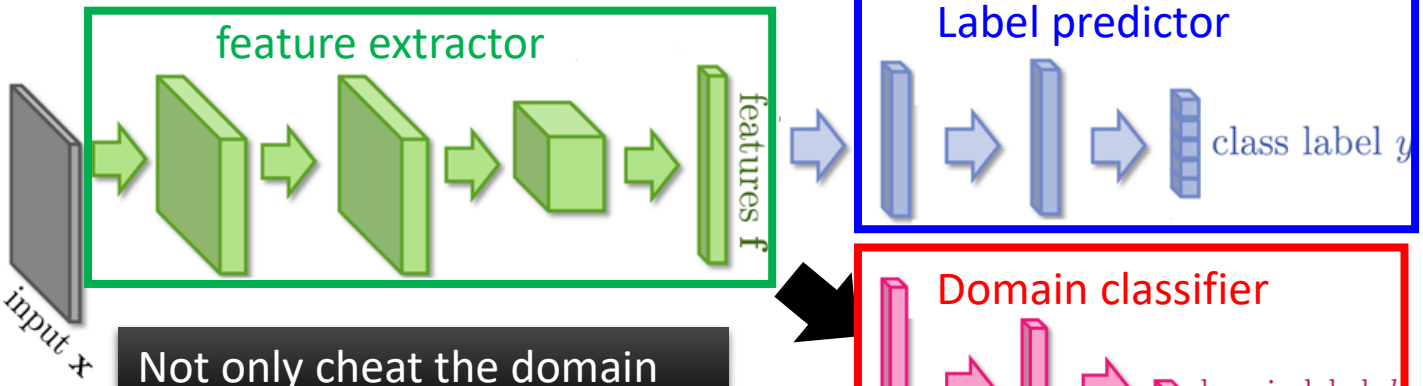


This is a big network, but different parts have different goals.

Domain-adversarial training

Maximize label classification accuracy + minimize domain classification accuracy

Maximize label classification accuracy

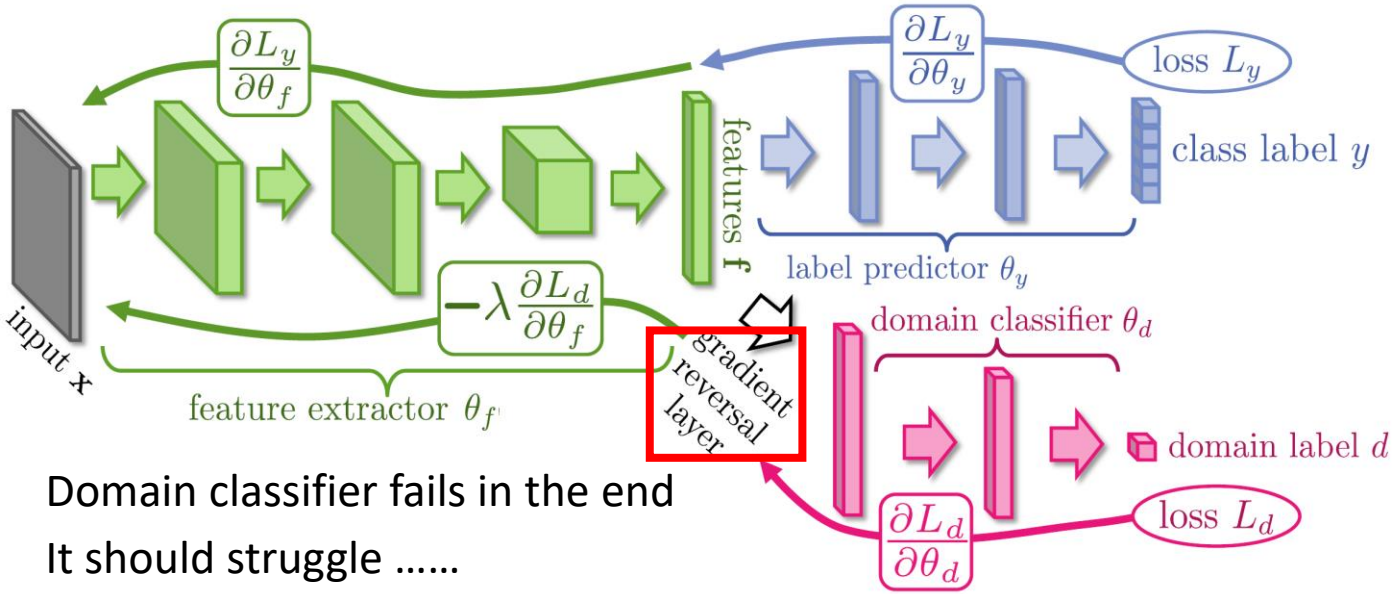


Not only cheat the domain classifier, but satisfying label classifier at the same time

Maximize domain classification accuracy

This is a big network, but different parts have different goals.

Domain-adversarial training



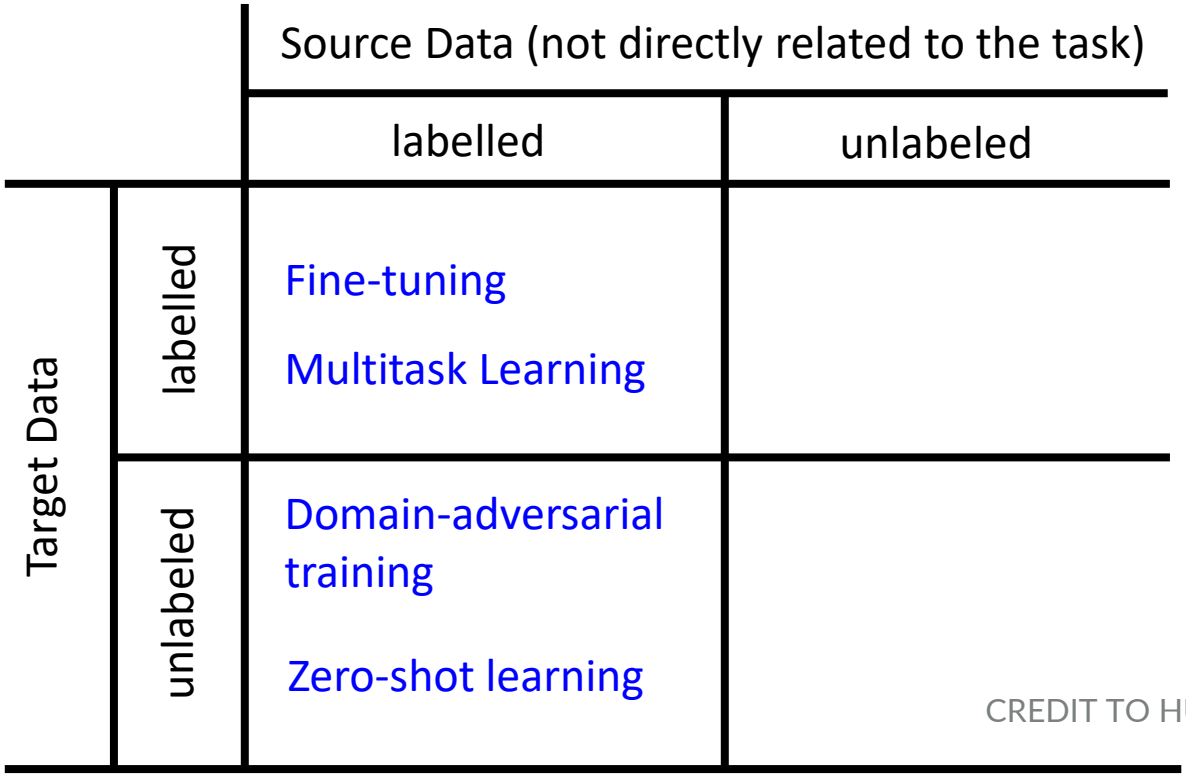
Domain classifier fails in the end

It should struggle ……

Yaroslav Ganin, Victor Lempitsky, Unsupervised Domain Adaptation by Backpropagation, ICML, 2015

Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, Domain-Adversarial Training of Neural Networks, JMLR, 2016

Transfer Learning - Overview



CREDIT TO HUNG-YI LEE@NTU

Zero-shot Learning

http://evchk.wikia.com/wiki/%E8%8D%89%E6%B3%A5%E9%A6%AC

- Source data: (x^s, y^s) → Training data
- Target data: (x^t) → Testing data

Different tasks

x^s:   ……

y^s: cat dog ……

x^t:  

In speech recognition, we can not have all possible words in the source (training) data.

How we solve this problem in speech recognition?

Summary

TLDRs

We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier/He init)
- Batch Normalization (use this!)
- Transfer learning (use this if you can!)

Next Few Lectures

Attention, relation, and memory in neural networks

Reinforcement learning

Generative models – VAE and GAN