EECS 498/598: Deep Learning
Lecture 13 Other Applications

Honglak Lee

4/12/2019

Outline

- Advice for projects
- Limitations of Deep Learning
- Emerging Deep Learning Research Topics:
  - Adversarial Attacks
  - AutoML

Outline

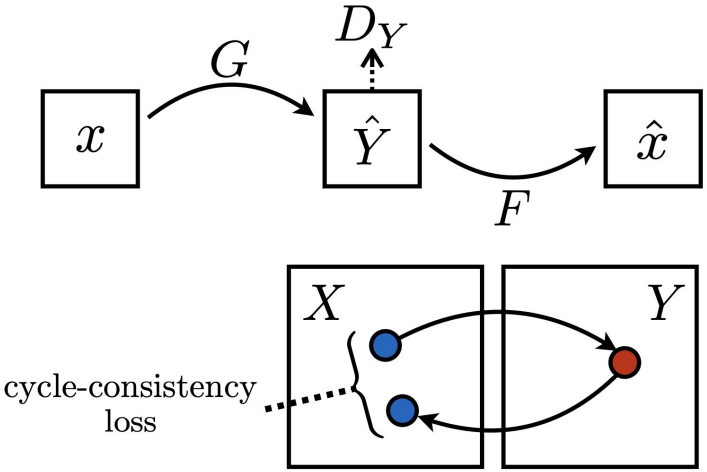
- **Advice for projects**
- Limitations of Deep Learning
- Emerging Deep Learning Research Topics:
  - Adversarial Attacks
  - AutoML

Novelty

- **Problem novelty:**
  - Has anyone investigated an approach to a problem using deep learning?

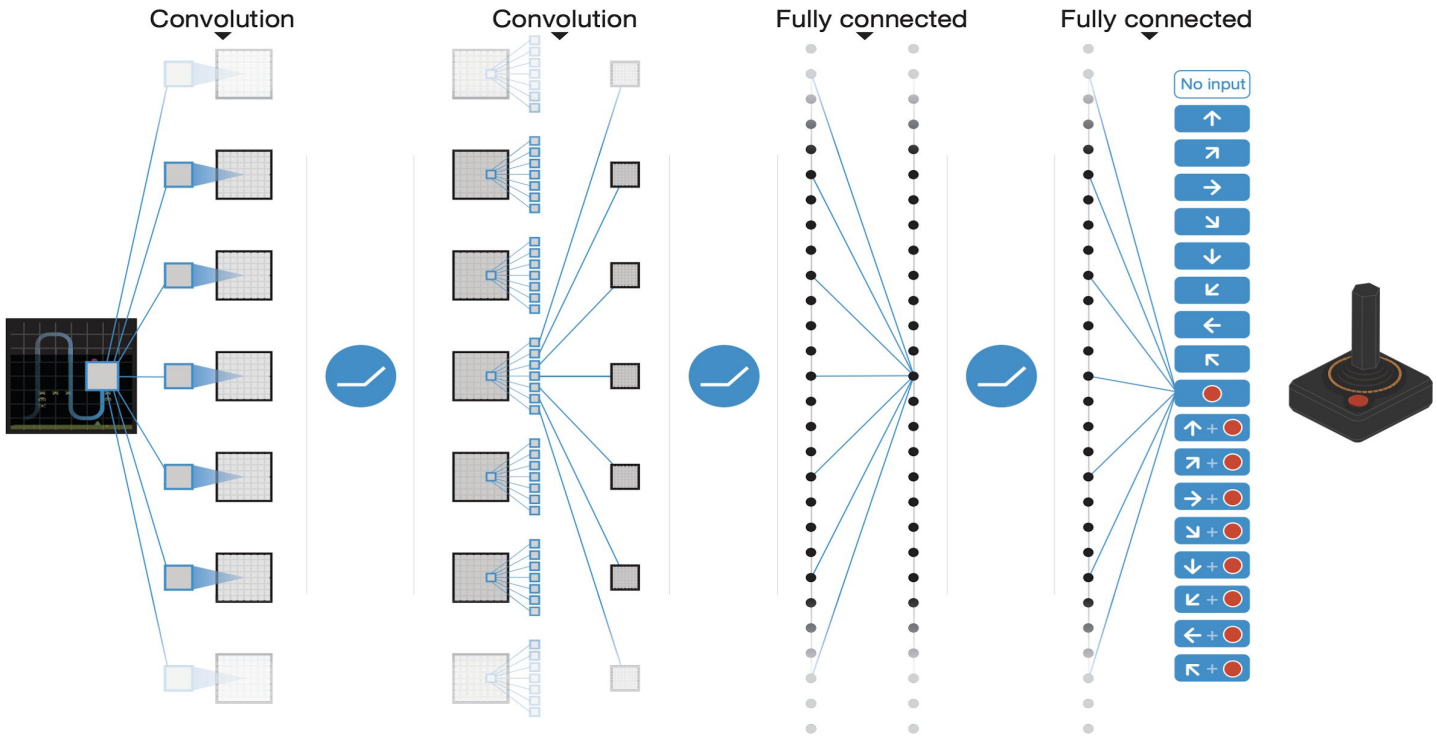
Novelty

- **Problem novelty:**
  - Unsupervised image-to-image translation



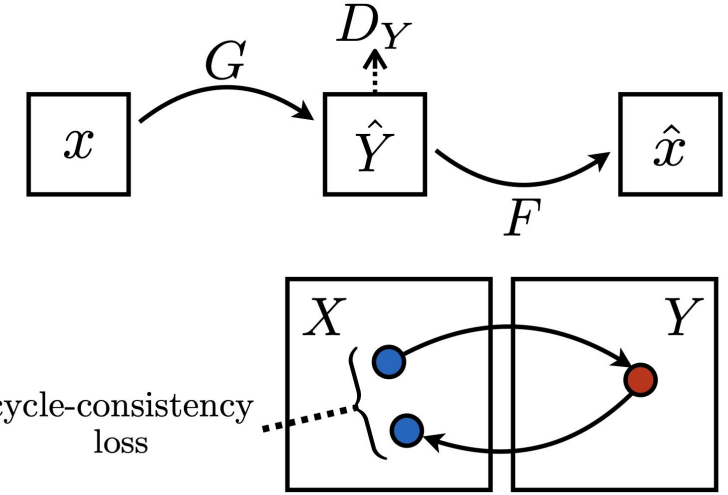
Novelty

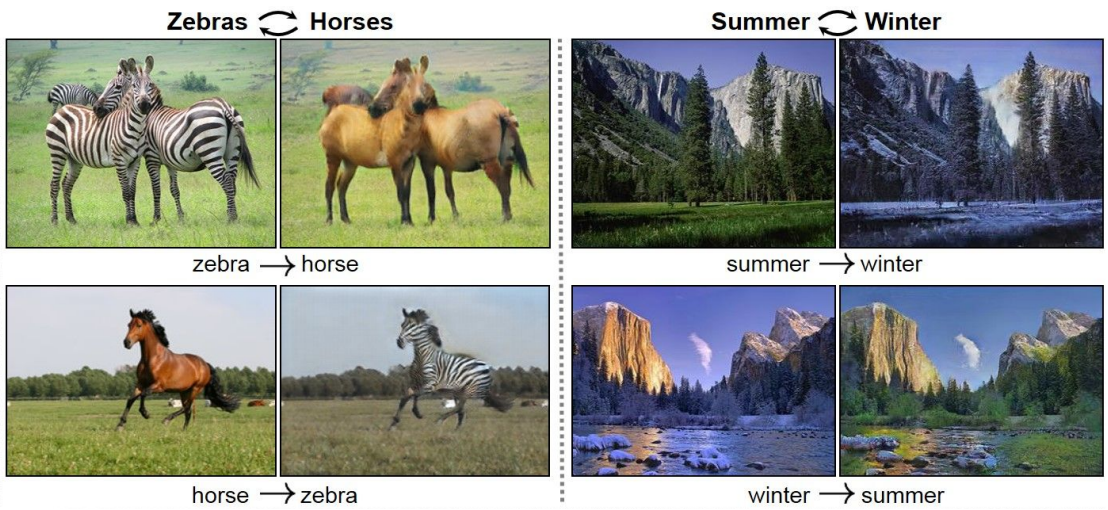
- **Problem novelty:** Deep Learning for atari games



Novelty

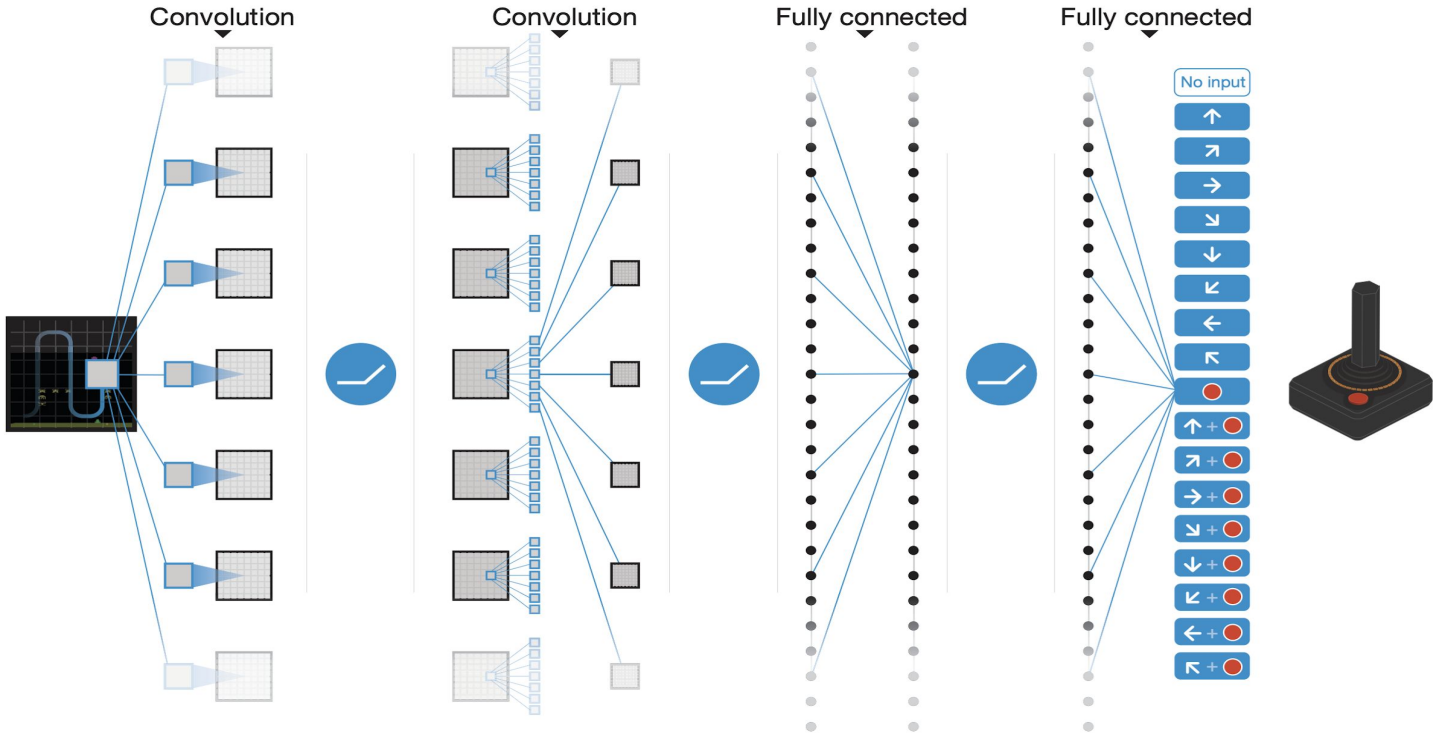
- **Algorithm novelty:**
  - Using cycle consistency and GAN objectives together to accomplish image-to-image translation





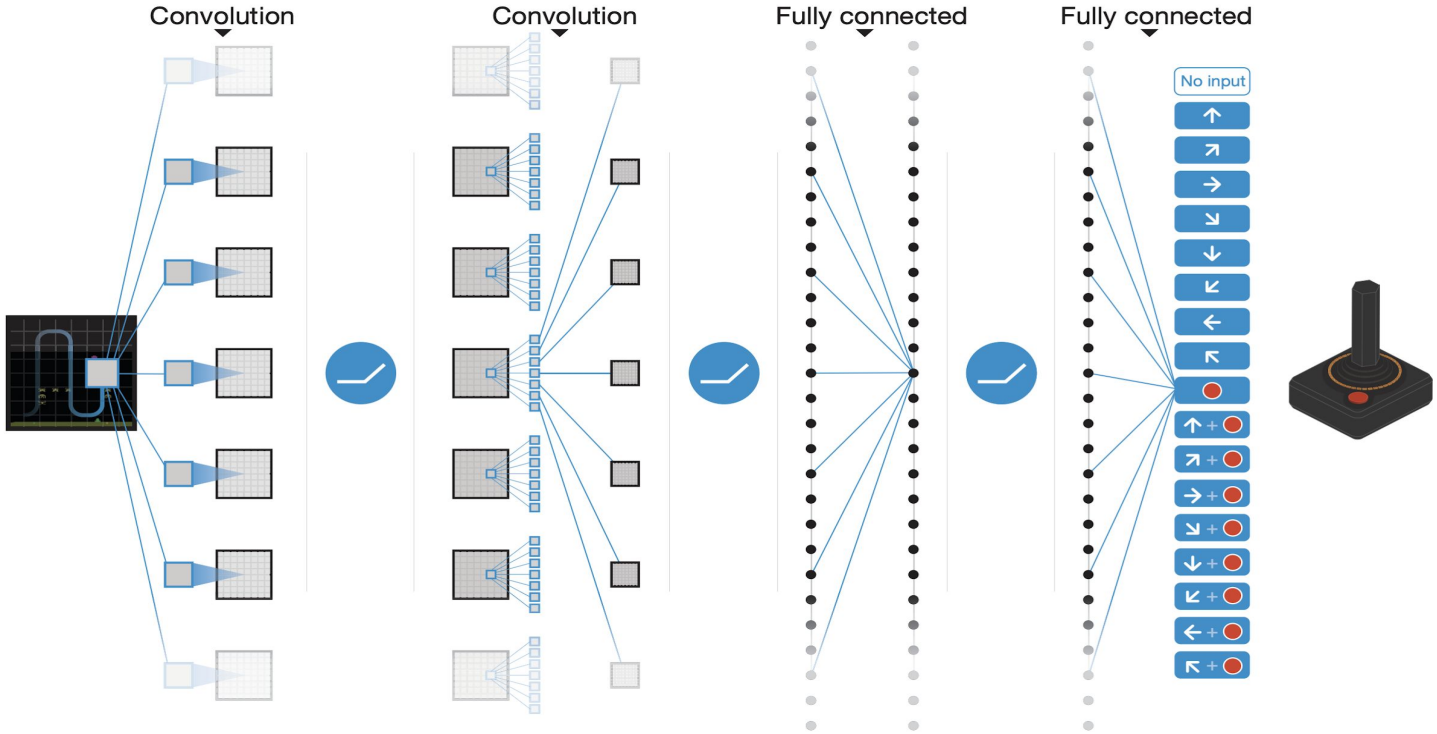
Novelty

- **Algorithm novelty:**
  - Experience replay buffer for stabilizing network training



Novelty

- **Architecture novelty**
  - Using a new neural network architecture to model the Q-function



Novelty

- **Architecture novelty**
  - Attention mechanism for image captioning

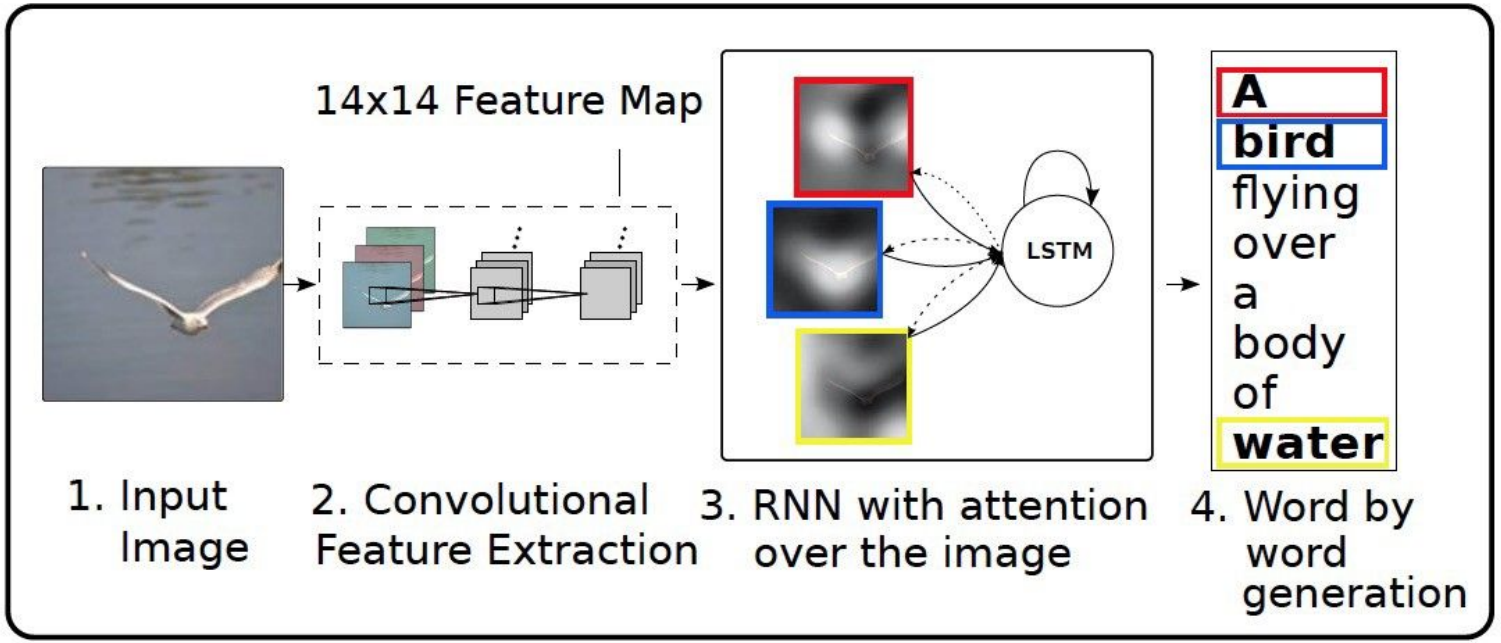


Figure credit: Xu et. al 2015

Novelty

- **Objective function**
  - Generative adversarial networks objective

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

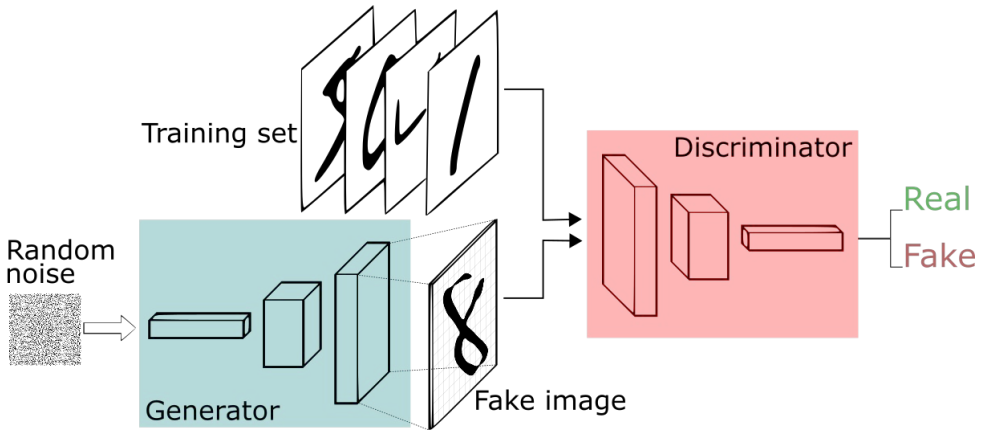


Figure credit: Thalles Silva

Technical quality

- Make sure that the proposed method makes sense
  - Problem statement is well defined
  - Hypothesis of the proposed methodology is clearly stated
  - Contributions of the paper are well highlighted (e.g., bullet points listing contributions)
  - Equations are correct
  - There is clear intuition behind any formulations
  - Experiments are designed test the hypothesis and generate results easy to analyze

Significance

- **Performance significance**
  - Has anyone before shown that using an existing method with minimal changes can significantly outperform previous methods?
  - Example: **AlexNet**

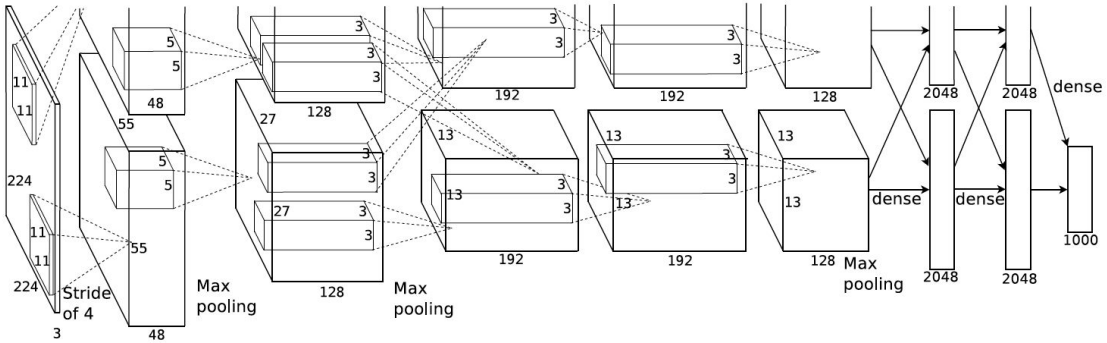


Figure credit: krizhevsky et. al 2012

Significance

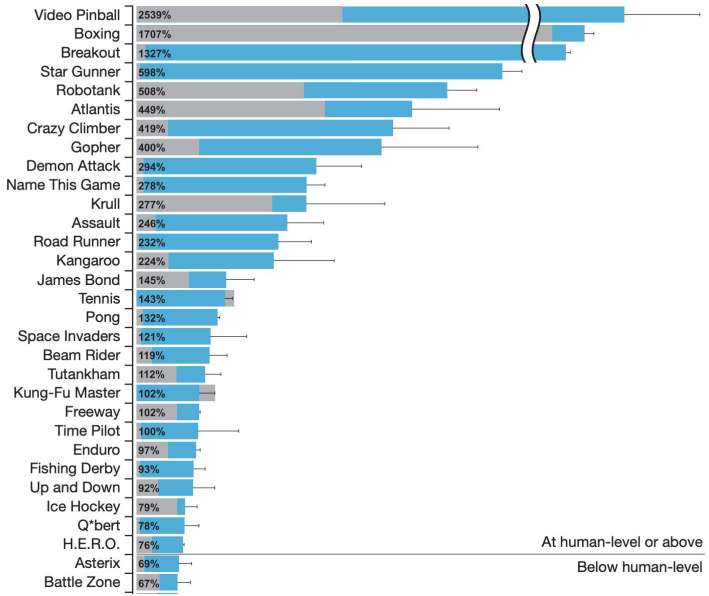
- **Performance significance**
  - Has anyone before shown that using an existing method with minimal changes can significantly outperform previous methods?
  - Example: **AlexNet**

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table from: krizhevsky et. al 2012

Significance

- **Performance significance**
  - Has anyone before shown that deep neural networks beat the state-of-the-art in atari games?
  - Example: **DQN**
    - It beat humans at many games



Significance

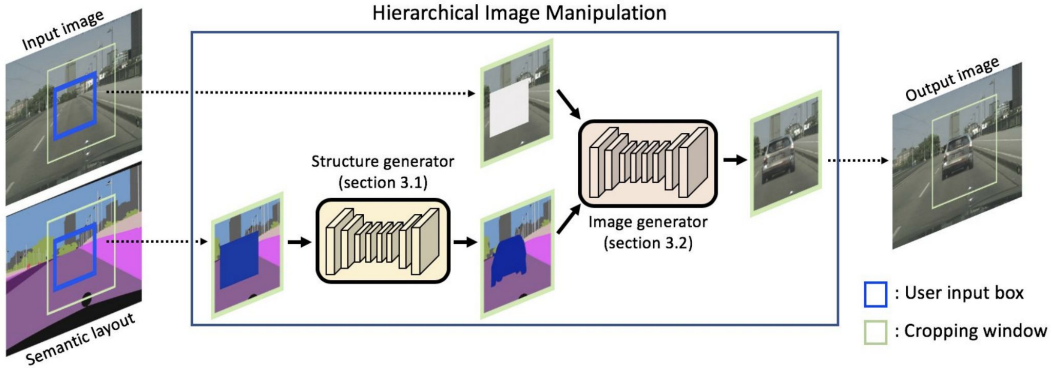
- Justify your method of choice through theoretical and experimental validations
  - Compare your method against the baselines
  - Provide ablation study on different design choices and hyperparameter settings
- Show that your model can work well over many examples/scenarios (rather than a single example)
- Show evidence that your method can generalize beyond the training data
- Provide as much quantitative and qualitative results to validate your claims.

Presentation

- Write as clear and self contained as possible:
  - Do not assume the person reading your paper “should understand you”.
  - It is your responsibility as a writer to make it crystal clear to reach as many people as possible
  - Distribute your paper to many people to see how many of them get confused by it, and address the parts where they become confused

Presentation

- Make write ups look as good as possible.
- Make **figures** look professional:
  - Please refrain yourself from making figures by hand.
  - There is free software only such as Google drawings that can be used to make good looking figures

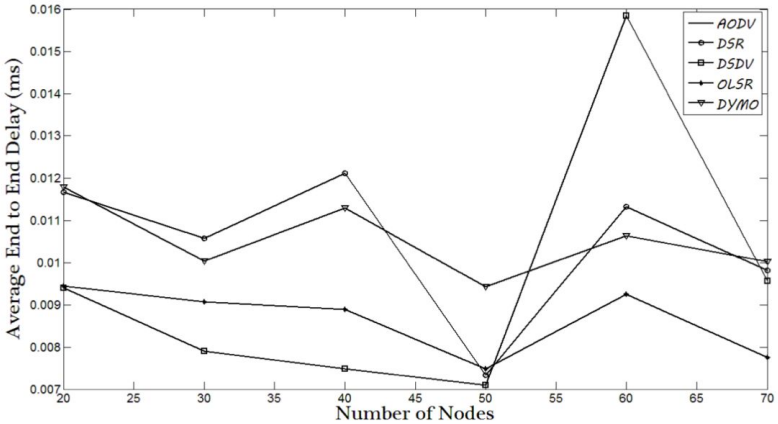


Do this!

Figure credit: Hong et. al 2018

Presentation

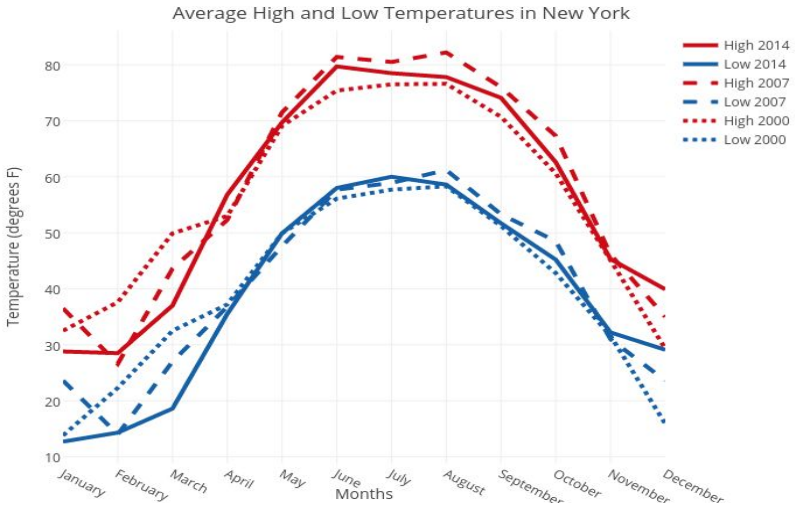
- Make write ups look as good as possible.
- Make sure **plots** are self contained
  - Label as well as you can: Plot title, axis, colors that are easy to identify, etc



Please don’t do this

Presentation

- Make write ups look as good as possible.
- Make sure **plots** are self contained
  - Label as well as you can: Plot title, axis, colors that are easy to identify, etc



Do this!

Research process advice

- Define your problem
  - Make sure the problem is clear to you (e.g., given input A, I want to get output B)
  - Make sure the problem is clear to others
    - If others find a flaw in your problem logic, there may be a flaw in the way you think of the problem

Research process advice

- Design the approach to the problem
  - Has the problem been approached with deep learning (e.g., neural network) before?
    - If not, why has it not?
      - Is there very little data?
      - Are there no real patterns in the data?
      - Did people just not think about it?
    - If yes, what are others doing?
      - Look at what others have done, find flaws with their approach, and fix them.

Research process advice

- Prepare your data
  - Partition your data into train/validation/test (if not available)
  - Make sure your data is balanced if partitions are not available
    - Example: If the problem is classification, make sure there are similar number of examples per class
  - Make sure you data is clean:
    - Make sure it does not contain misleading examples (e.g., wrong class labels assigned to images)
  - Normalize your data (e.g., subtract mean and divide by stdev)
    - You do not want to force your neural network to output values that are too large, it may diverge

Research process advice

- Design experiments
  - What do you want to show?
    - Obviously, you want to show your method does what it was designed to do (e.g., classify images, generate text, etc)
    - Generalization to unseen data domains?
    - Use your method outputs to help other methods?
      - Example: Predict future human motion and give it to a method that can infer intention of a human
    - Does your method show an interesting behavior?
      - Example: Show attention mechanism focuses on relevant parts of the input to generate an output

Research process advice

- Document results
  - Make a write up about your findings
  - Present a coherent analysis about your results such that readers see the value you see in your experiments
  - Think of any type of plots, tables, etc that could highlight the significance of your findings
  - Analyze failures and successes, and present a coherent write up of why your method fails or succeeds and how any failures could be addressed
  - Think about the “expected questions” from the audience and try to answer them in your report!

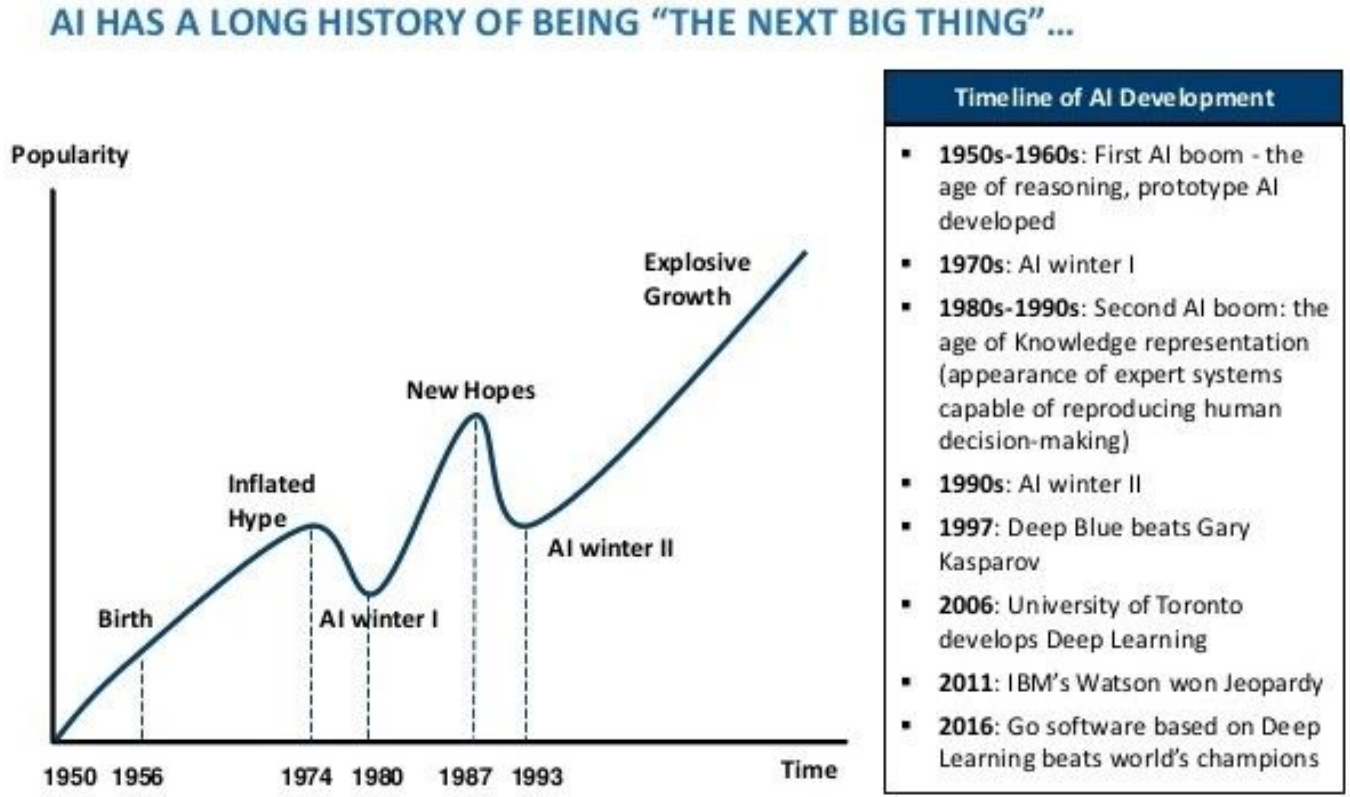
Outline

- Advice for projects
- **Limitations of Deep Learning**
- Emerging Deep Learning Research Topics:
  - Adversarial Attacks
  - AutoML

The Rise of Deep Learning

Slide credit: Ava Soleimani

Artificial intelligence “hype”: Historical Perspective



Slide credit: Ava Soleimani

Limitation of Neural Networks

- Very data hungry (eg. often millions of examples)
- Computationally intensive to train and deploy (tractably requires GPUs)
- Large representation capacity may lead to memorization
- Can be subject to algorithmic bias
- Easily fooled by adversarial examples
- Suboptimal/unexpected performance for data regime underrepresented or unseen during training (how do you know what the model knows?)
- Uninterpretable black boxes, difficult to trust
- Finicky to optimize: non-convex, choice of architecture, learning parameters
- Often require expert knowledge to design, fine tune architectures

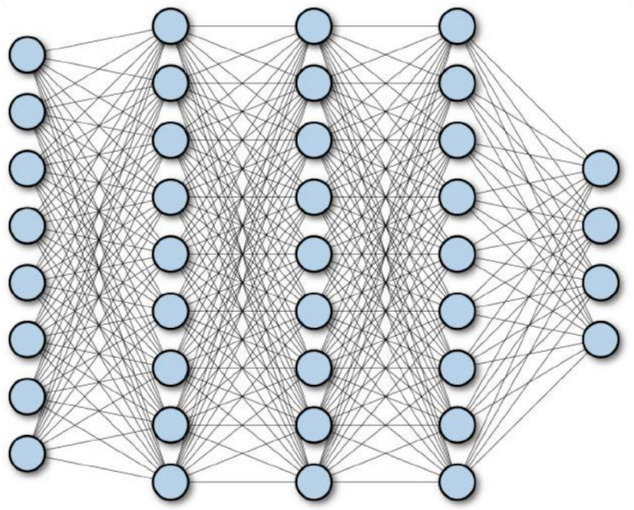
Limitation of Neural Networks

- Very data hungry (eg. often millions of examples)
- Computationally intensive to train and deploy (tractably requires GPUs)
- Large representation capacity may lead to memorization
- Can be subject to algorithmic bias
- Easily fooled by adversarial examples
- Suboptimal/unexpected performance for data regime underrepresented or unseen during training (how do you know what the model knows?)
- Uninterpretable black boxes, difficult to trust
- Finicky to optimize: non-convex, choice of architecture, learning parameters
- Often require expert knowledge to design, fine tune architectures

The power of neural networks

Universal Approximation Theorem

A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.



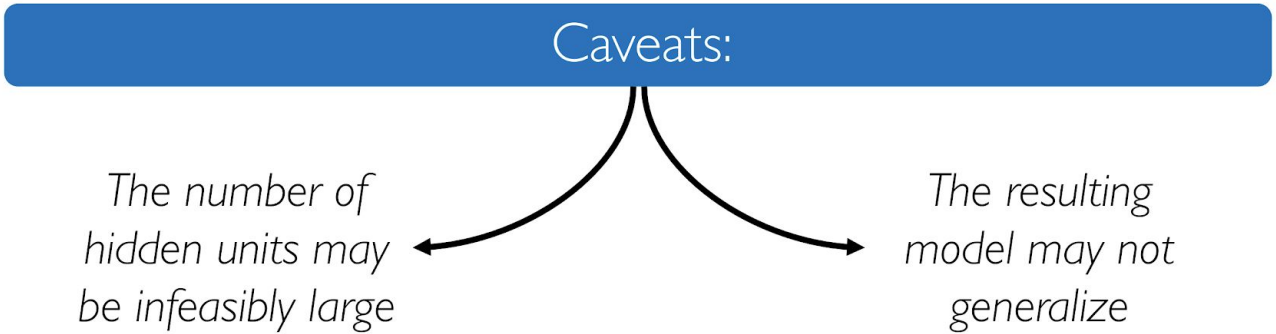
Hornik et al. *Neural Networks.* (1989)

Slide credit: Ava Soleimani

The power of neural networks

Universal Approximation Theorem

A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.



Hornik et al. *Neural Networks.* (1989)

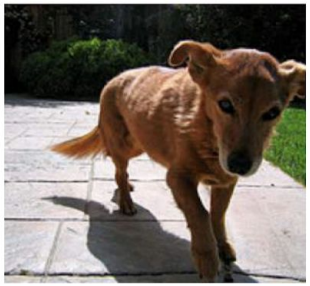
Slide credit: Ava Soleimani

Rethinking generalization

"Understanding Deep Neural Networks Requires Rethinking Generalization"

dog

banana



dog

tree

Zhang et al. *ICLR*. (2017)

Slide credit: Ava Soleimani

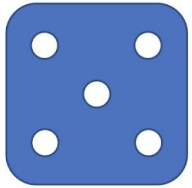
Rethinking generalization

"Understanding Deep Neural Networks Requires Rethinking Generalization"

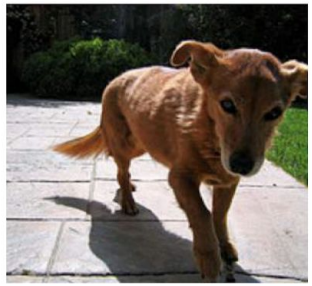
dog

banana

banana



dog



dog

tree

tree

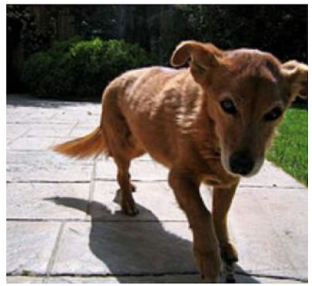
dog

Zhang et al. *ICLR*. (2017)

Slide credit: Ava Soleimani

Rethinking generalization

"Understanding Deep Neural Networks Requires Rethinking Generalization"



Zhang et al. *ICLR*. (2017)

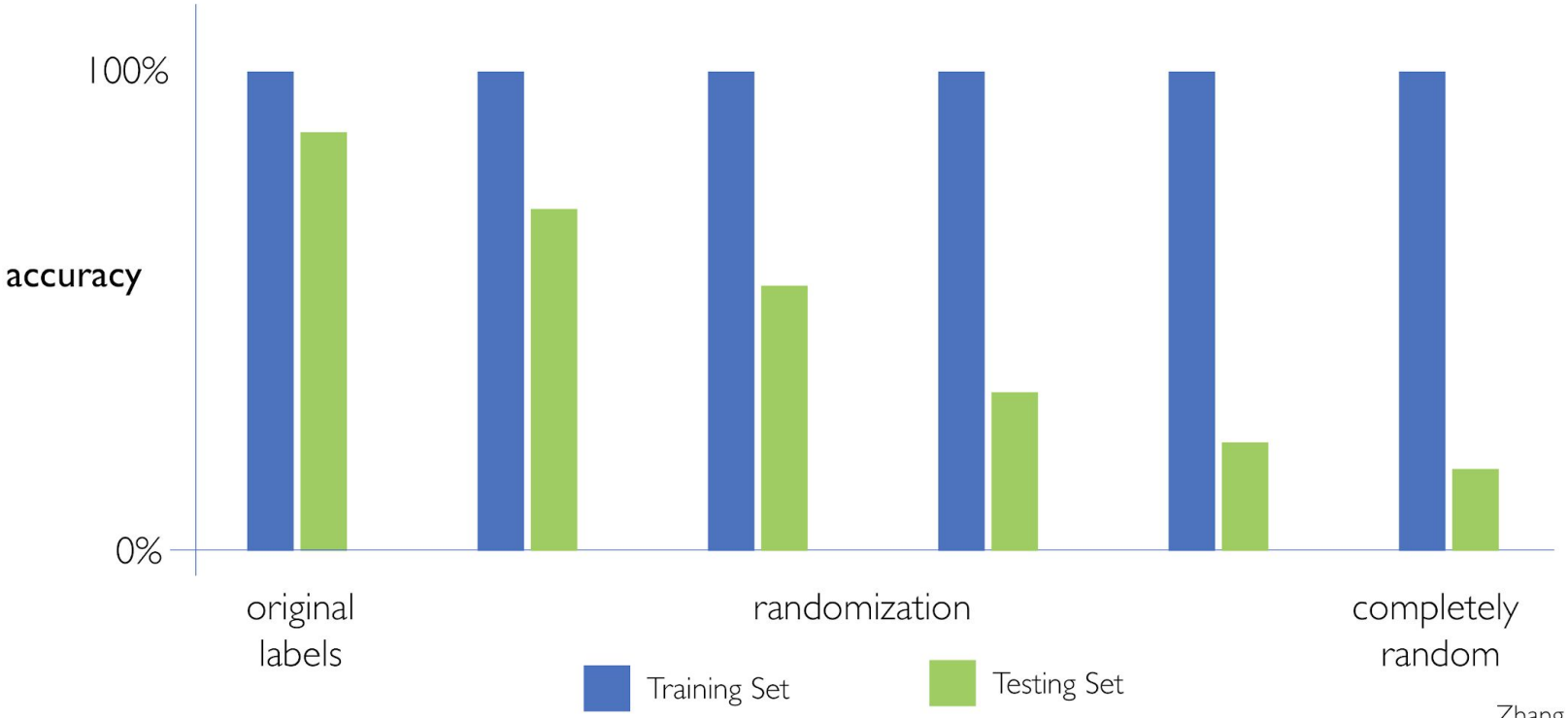
Slide credit: Ava Soleimani

Capacity of deep neural networks

Zhang et al. *ICLR*. (2017)

Slide credit: Ava Soleimani

Capacity of deep neural networks



Zhang et al. *ICLR*. (2017)

Slide credit: Ava Soleimani

Capacity of deep neural networks

Zhang et al. *ICLR*. (2017)

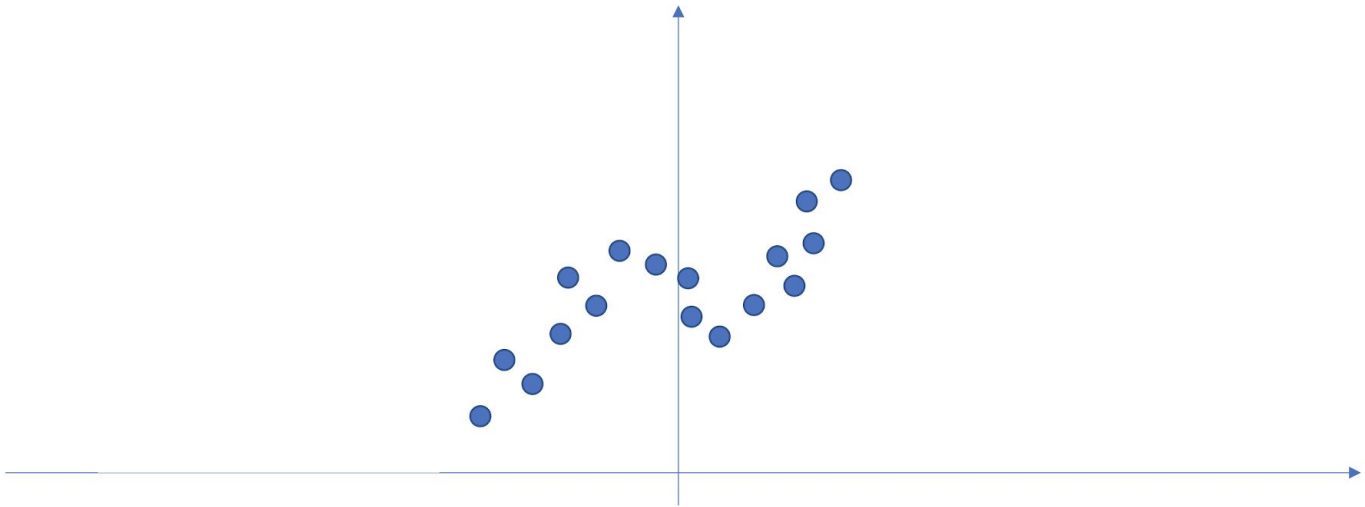
Slide credit: Ava Soleimani

Limitation of Neural Networks

- Very data hungry (eg. often millions of examples)
- Computationally intensive to train and deploy (tractably requires GPUs)
- Large representation capacity may lead to memorization
- Can be subject to algorithmic bias
- Easily fooled by adversarial examples
- Suboptimal/unexpected performance for data regime underrepresented or unseen during training (how do you know what the model knows?)
- Uninterpretable black boxes, difficult to trust
- Finicky to optimize: non-convex, choice of architecture, learning parameters
- Often require expert knowledge to design, fine tune architectures

Neural networks as function approximators

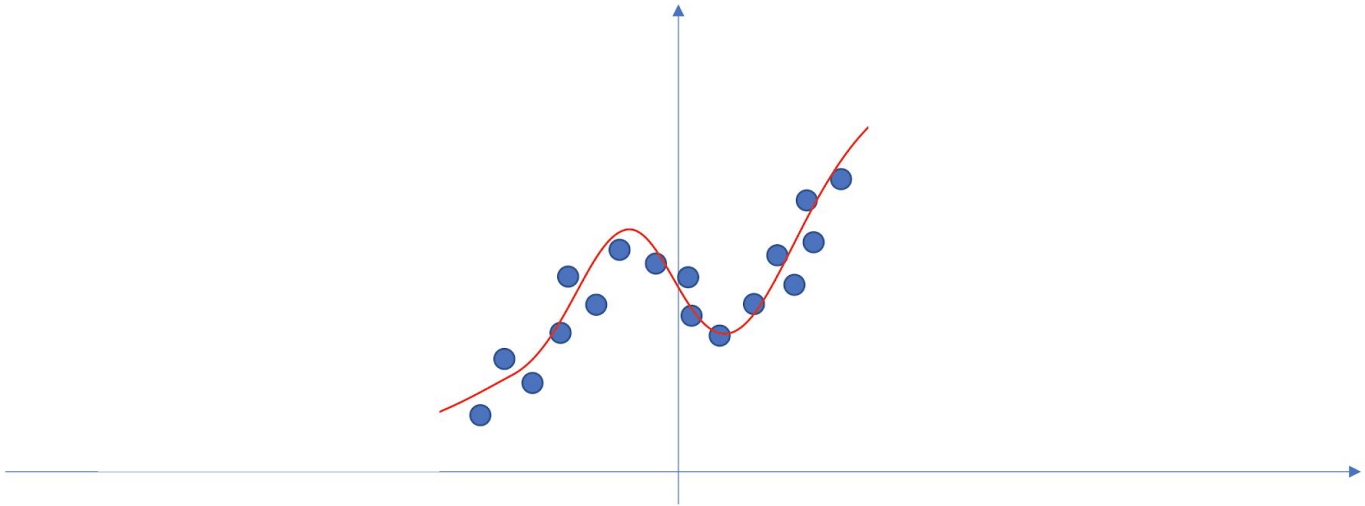
Neural networks are **excellent** function approximators



Slide credit: Ava Soleimani

Neural networks as function approximators

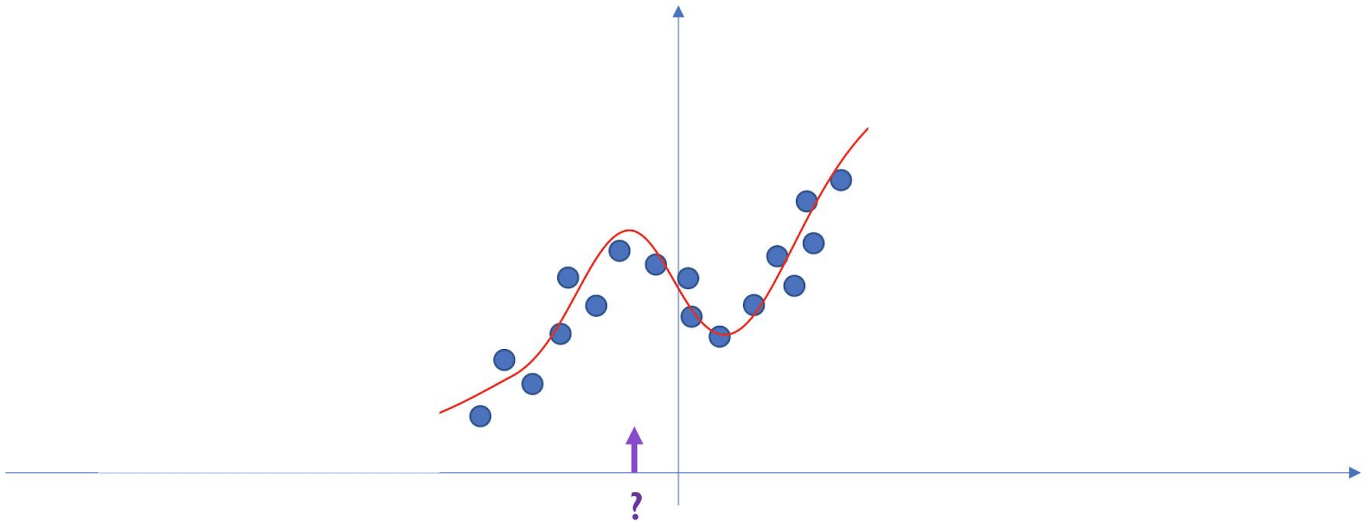
Neural networks are **excellent** function approximators



Slide credit: Ava Soleimani

Neural networks as function approximators

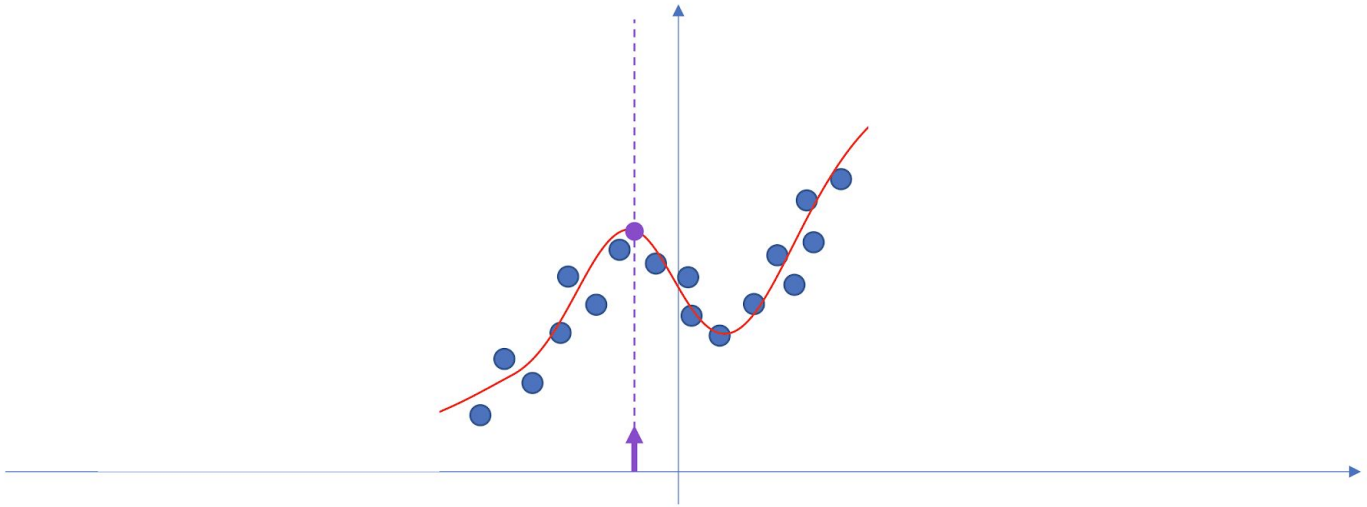
Neural networks are **excellent** function approximators



Slide credit: Ava Soleimani

Neural networks as function approximators

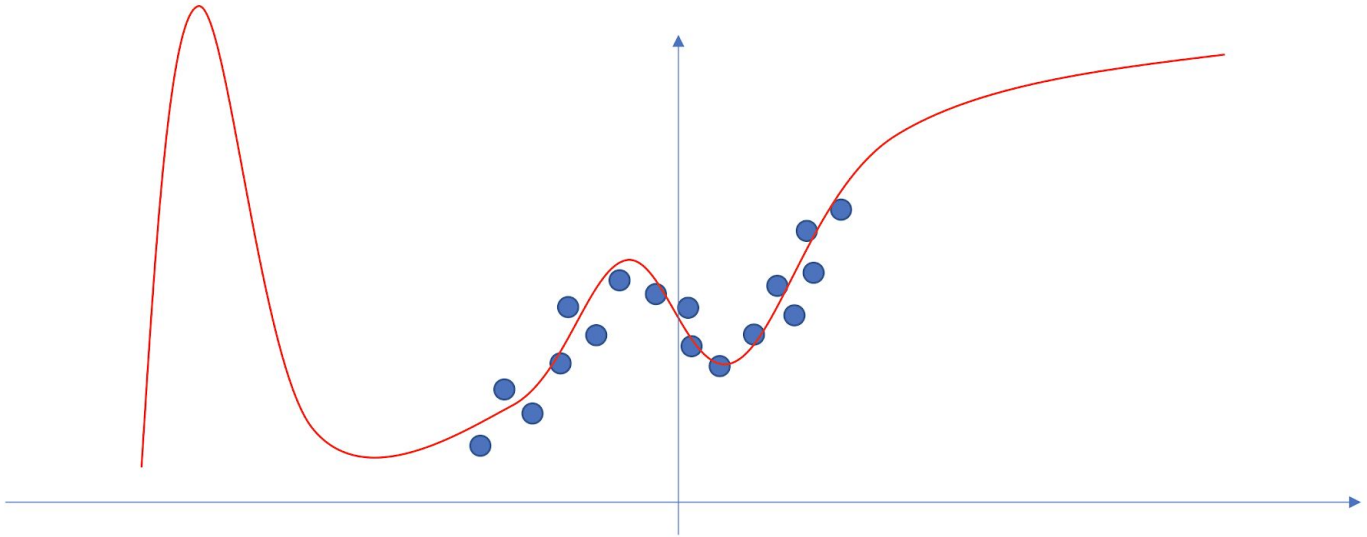
Neural networks are **excellent** function approximators



Slide credit: Ava Soleimani

Neural networks as function approximators

Neural networks are **excellent** function approximators

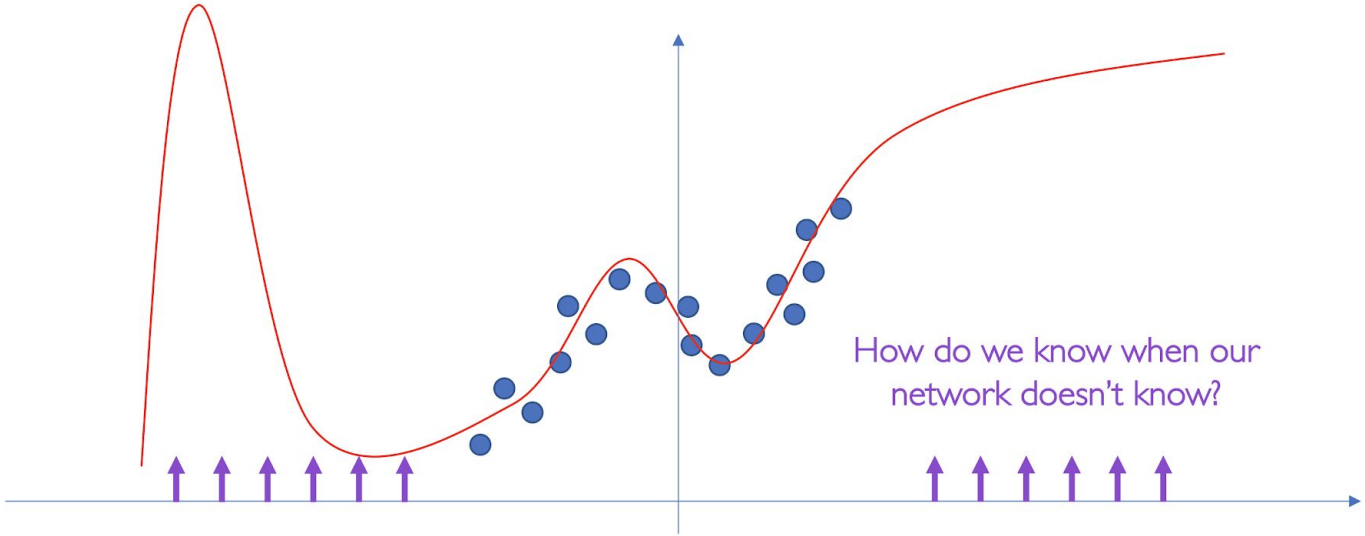


Slide credit: Ava Soleimani

Neural networks as function approximators

Neural networks are **excellent** function approximators

…when they have training data



Slide credit: Ava Soleimani

Approaches for improving reliability of neural networks

Two related problems:

- Confidence-calibrated neural networks
- Detecting out-of-distribution samples

Key questions:

- How can we train a neural network so that it can provide “confidence” of its outputs (i.e., how likely its prediction will be correct)?
- How can we tell if the testing example (or query) looks very different from the training distribution?

Calibrating Neural Networks Outputs and Detecting Out-of-distribution Samples

Related work

- On Calibration of Modern Neural Networks
- Simple and scalable predictive uncertainty estimation using deep ensembles
- Enhancing the reliability of out-of-distribution image detection in neural networks
- Training confidence-calibrated classifiers for detecting out-of-distribution samples
- A simple unified framework for detecting out-of-distribution samples and adversarial attacks
- Learning confidence for out-of-distribution detection in neural networks

Outline

- Advice for projects
- Limitations of Deep Learning
- **Emerging Deep Learning Research Topics:**
  - **Adversarial Attacks & Defense**
  - AutoML

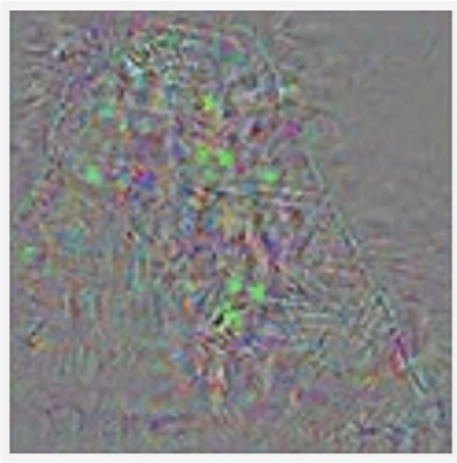
Limitation of Neural Networks

- Very data hungry (eg. often millions of examples)
- Computationally intensive to train and deploy (tractably requires GPUs)
- Large representation capacity may lead to memorization
- Can be subject to algorithmic bias
- Easily fooled by adversarial examples
- Suboptimal/unexpected performance for data regime underrepresented or unseen during training (how do you know what the model knows?)
- Uninterpretable black boxes, difficult to trust
- Finicky to optimize: non-convex, choice of architecture, learning parameters
- Often require expert knowledge to design, fine tune architectures

Adversarial attacks on neural networks

Original image

Temple (97%)



Perturbations

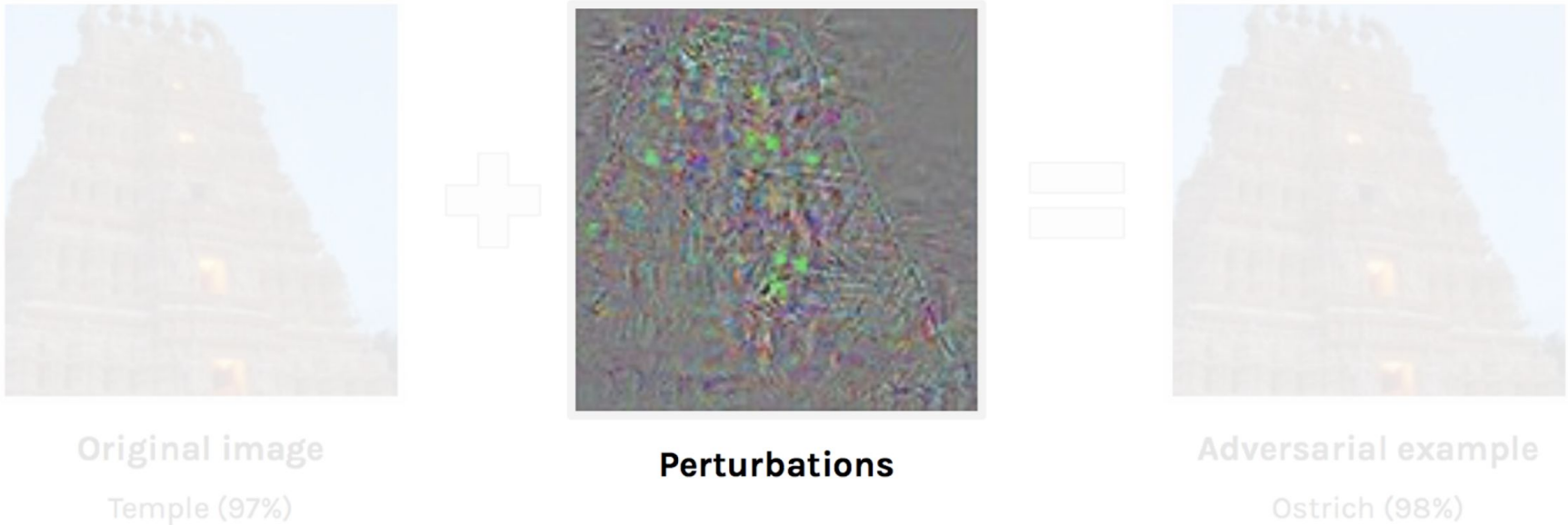
Adversarial example

Ostrich (98%)

Despois. “Adversarial examples and their implications” (2017)

Slide credit: Ava Soleimani

Adversarial attacks on neural networks



Despois. “Adversarial examples and their implications” (2017)

Slide credit: Ava Soleimani

Adversarial attacks on neural networks

Remember:

We train our networks with gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta, x, y)}{\partial \theta}$$

"How does a small change in weights decrease our loss"

Slide credit: Ava Soleimani

Adversarial attacks on neural networks

Remember:

We train our networks with gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta, x, y)}{\partial \theta}$$

"How does a small change in weights decrease our loss"

Slide credit: Ava Soleimani

Adversarial attacks on neural networks

Remember:

We train our networks with gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta, x, y)}{\partial \theta}$$

Fix your image x, and true label y

"How does a small change in weights decrease our loss"

Slide credit: Ava Soleimani

Adversarial attacks on neural networks

Adversarial Image:

Modify image to increase error

$$x \leftarrow x + \eta \frac{\partial J(\theta, x, y)}{\partial x}$$

"How does a small change in the input increase our loss"

Goodfellow et al. *NIPS* (2014)

Slide credit: Ava Soleimani

Adversarial attacks on neural networks

Adversarial Image:

Modify image to increase error

$$x \leftarrow x + \eta \frac{\partial J(\theta, x, y)}{\partial x}$$

"How does a small change in the input increase our loss"

Slide credit: Ava Soleimani

Adversarial attacks on neural networks

Adversarial Image:

Modify image to increase error

$$x \leftarrow x + \eta \frac{\partial J(\theta, x, y)}{\partial x}$$

Fix your weights θ, and true label y

"How does a small change in the input increase our loss"

Slide credit: Ava Soleimani

Types of adversarial attacks

- **White-Box**: Attackers know full knowledge about the ML algorithm, ML model, (i.e., parameters and hyperparameters), architecture, etc.
- **Black-Box**: Attackers almost know nothing about the ML system (perhaps know number of features, ML algorithm).

Slide credit: Binghui Wang

White-box attack

- Given a function (LogReg, SVM, DNN, etc) $\mathbf{F}: \mathbf{X} \mapsto \mathbf{Y}$, where X is a input feature vector, and Y is an output vector.
- An attacker expects to construct an adversarial sample X* from X by adding a perturbation vector $\delta_{\mathbf{X}}$ such that

$$\arg\min_{\delta_{\mathbf{X}}} \|\delta_{\mathbf{X}}\| \text{ s.t. } \mathbf{F}(\mathbf{X} + \delta_{\mathbf{X}}) = \mathbf{Y}^*$$

- where $\mathbf{X}^* = \mathbf{X} + \delta_{\mathbf{X}}$ and Y* is the desired adversarial output.
- Solving this problem is non-trivial, when F is nonlinear or/and nonconvex.

Slide credit: Binghui Wang

White-box attack

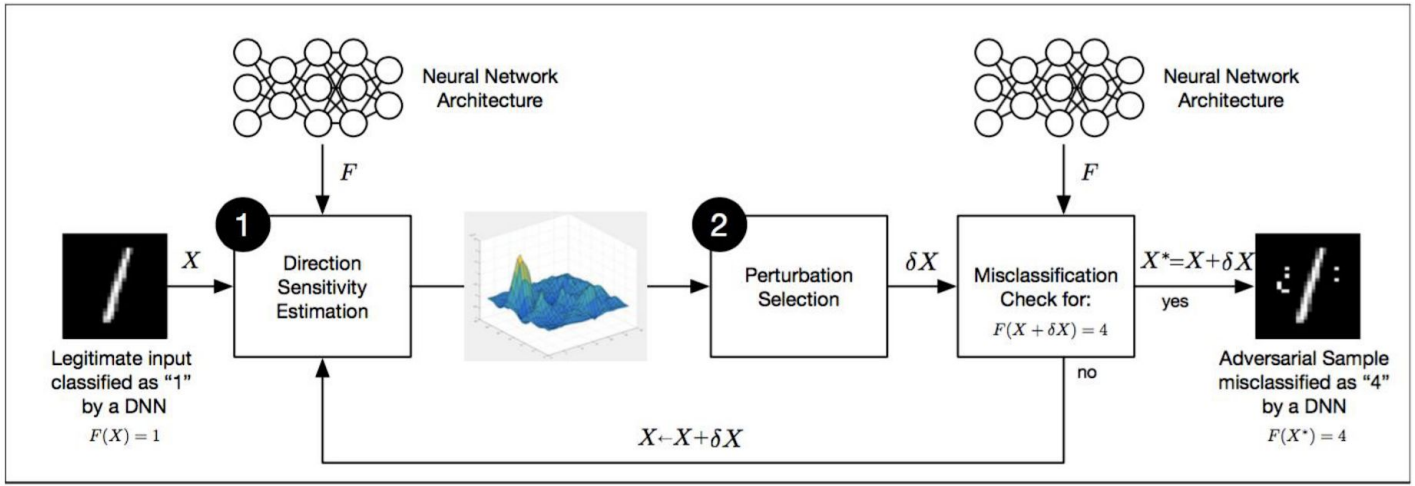
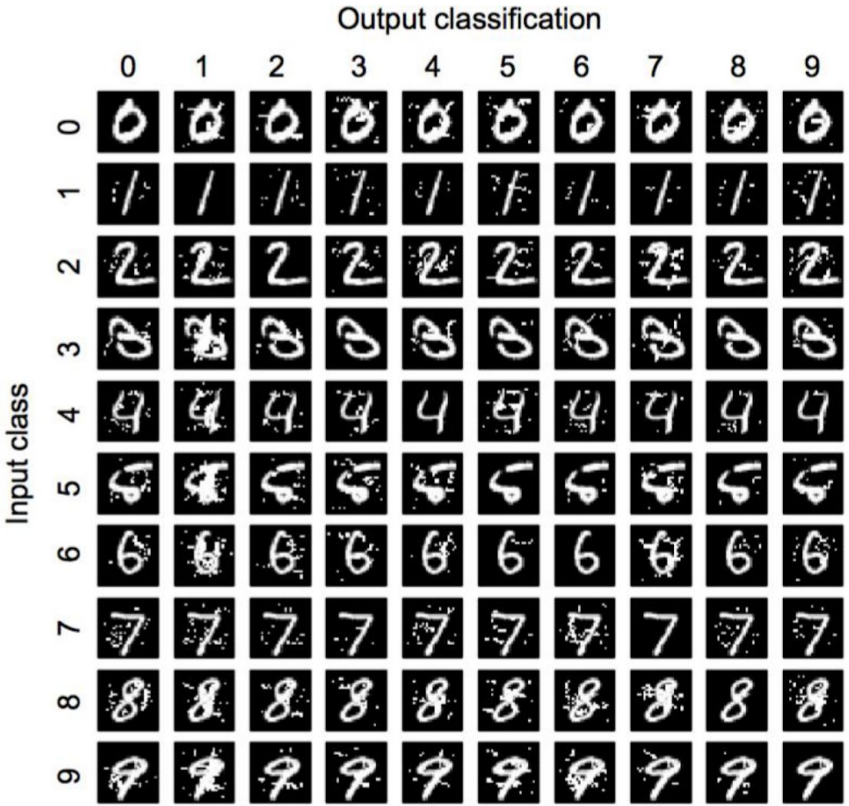
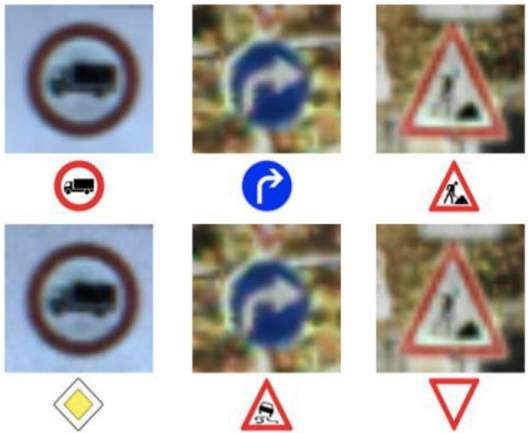


Fig. 3: **Adversarial crafting framework:** Existing algorithms for adversarial sample crafting [7], [9] are a succession of two steps: (1) *direction sensitivity estimation* and (2) *perturbation selection*. Step (1) evaluates the sensitivity of model F at the input point corresponding to sample X. Step (2) uses this knowledge to select a perturbation affecting sample X's classification. If the resulting sample $X + \delta X$ is misclassified by model F in the adversarial target class (here 4) instead of the original class (here 1), an adversarial sample X^* has been found. If not, the steps can be repeated on updated input $X \leftarrow X + \delta X$.

Slide credit: Binghui Wang

White-box attack





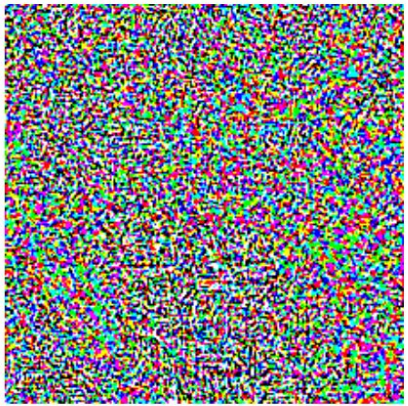
Slide credit: Binghui Wang

Adversarial examples

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

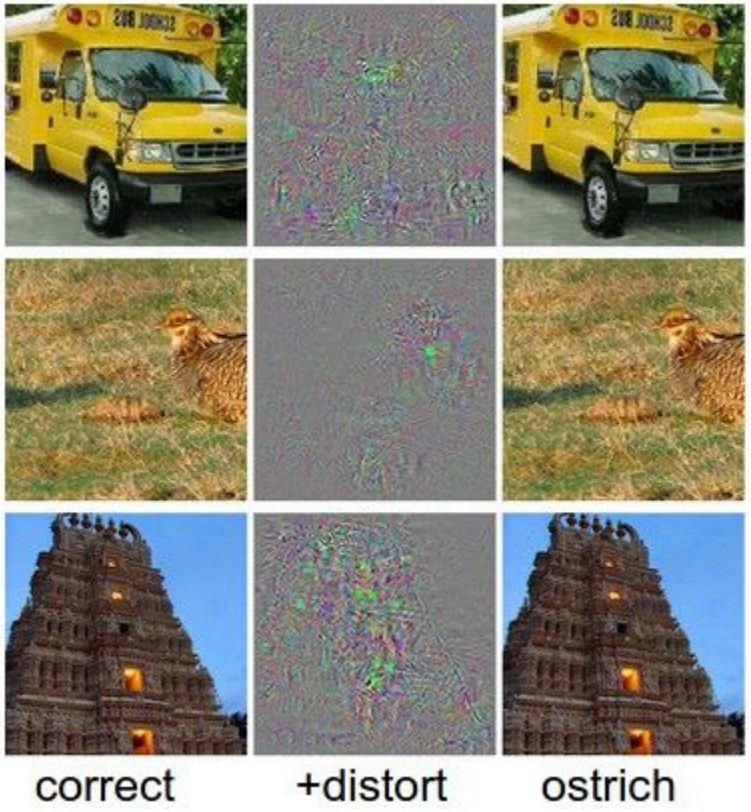
$=$

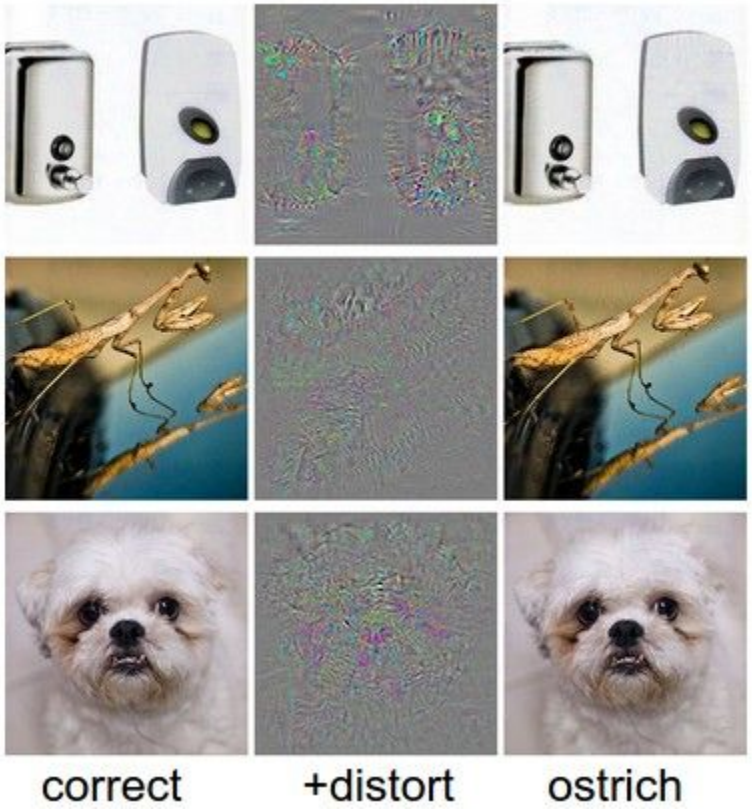
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv:1312.6199*.
Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

More adversarial examples





Black-box attack

- Adversarial Sample Transferability
  - Cross model transferability: The same adversarial sample is often misclassified by a variety of classifiers with different architectures
  - cross training-set transferability: The same adversarial sample is often misclassified trained on different subsets of the training data.
- Therefore, an attacker can
  - First train his own (white-box) substitute model
  - Then generate adversarial samples
  - Finally, apply the adversarial samples to the target ML model

Slide credit: Binghui Wang

Synthesizing **robust** adversarial examples

A sample of photos of unperturbed 3D prints. The unperturbed 3D-printed objects are consistently classified as the true class.

classified as turtle classified as rifle
classified as other

Photos of **Adversarially perturbed 3D-printed turtle** taken in different viewpoints (turtle → rifle).

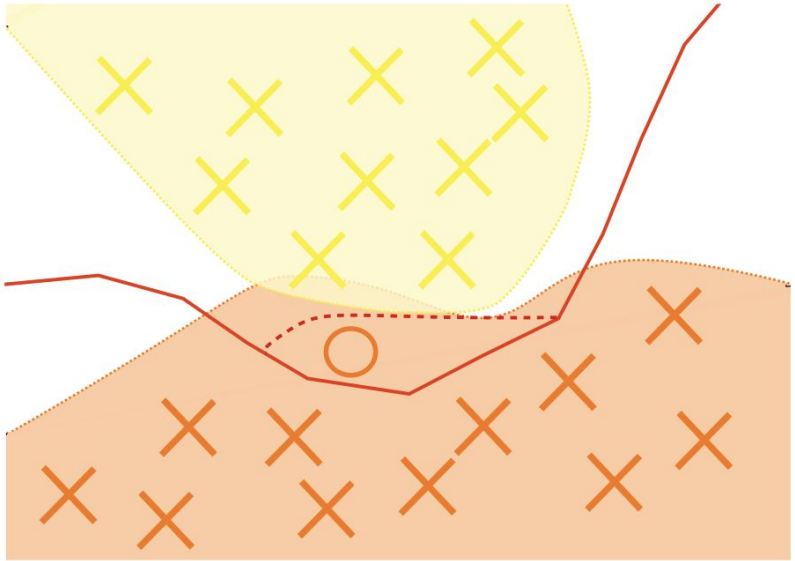
classified as turtle classified as rifle
classified as other

Athalye, A., Engstrom, L., Ilyas, A., & Kwok, K. (2017). Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*.

Defense against adversarial attacks

- Data Augmentation (e.g., dropout, cutout, mixup).
- Adversarial Training.
  - Generate adversarial examples and include them as part of the training data.

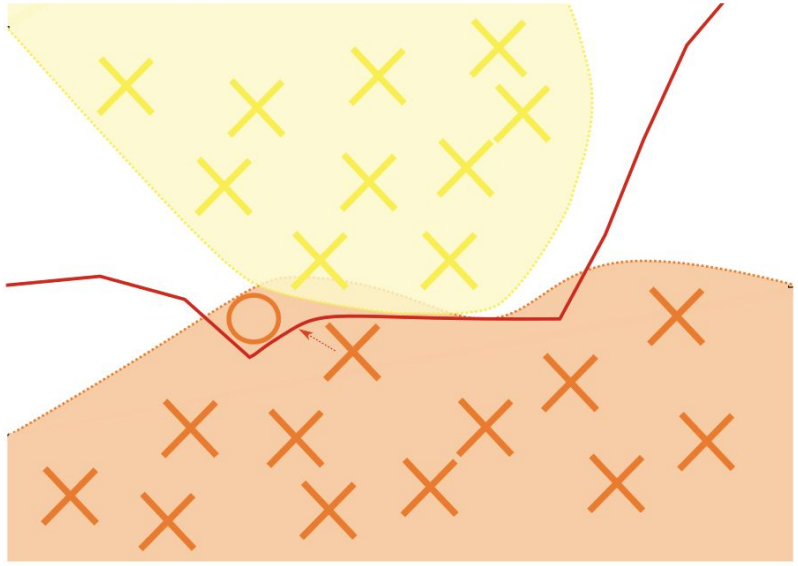
Defending against adversarial attacks



- Adapt the classifier to attack directions by including adversarial data at training.

Slide credit: Valentina Satendeschi

Defending against adversarial attacks



- Adapt the classifier to attack directions by including adversarial data at training.
- But there are always new adversarial samples to be crafted.

Defense against adversarial examples

Adversarial training:

- Inject adversarial examples into the training set, continually generating new adversarial examples at every step of training (Goodfellow et al., 2014).

$$Loss = \frac{1}{(m-k)+\lambda k}\left(\sum_{i \in CLEAN} L(X_i|y_i) + \lambda \sum_{i \in ADV} L(X_i^{adv}|y_i)\right)$$

Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.

Optimization View on Adversarial Robustness

Learning a robust model (e.g., classifier) can be viewed as a following saddle point problem:

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y) \right]$$

- L: loss function
- S: a set of allowed perturbations

Here, the **inner maximization** problem can be viewed as **attack,** and the **outer minimization problem** as **defense**.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv:1706.06083*.

Outline

- Advice for projects
- Limitations of Deep Learning
- **Emerging Deep Learning Research Topics:**
  - Adversarial Attacks & Defense
  - **AutoML**

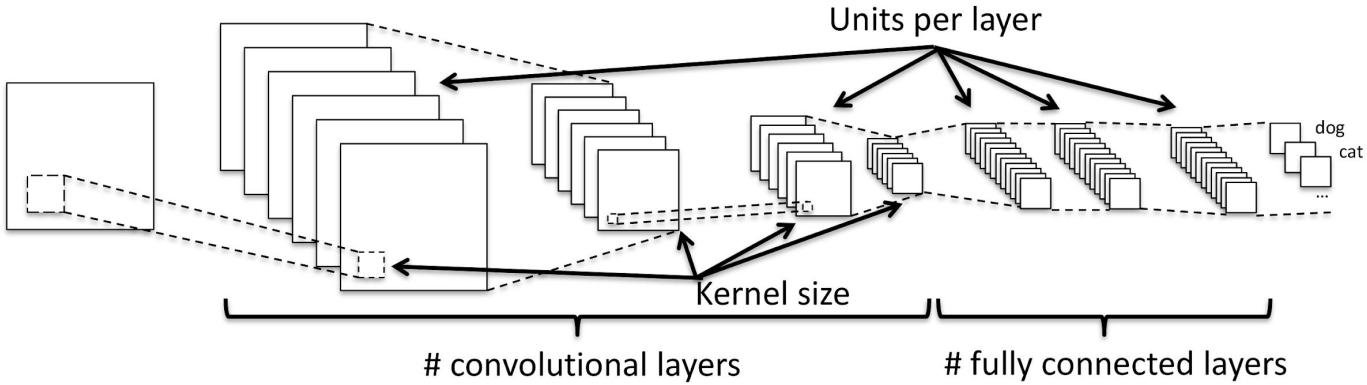
Limitation of Neural Networks

- Very data hungry (eg. often millions of examples)
- Computationally intensive to train and deploy (tractably requires GPUs)
- Large representation capacity may lead to memorization
- Can be subject to algorithmic bias
- Easily fooled by adversarial examples
- Suboptimal/unexpected performance for data regime underrepresented or unseen during training (how do you know what the model knows?)
- Uninterpretable black boxes, difficult to trust
- Finicky to optimize: non-convex, choice of architecture, learning parameters
- Often require expert knowledge to design, fine tune architectures

One problem with deep learning

Performance is very **sensitive** to **many hyperparameters**

- Architectural hyperparameters



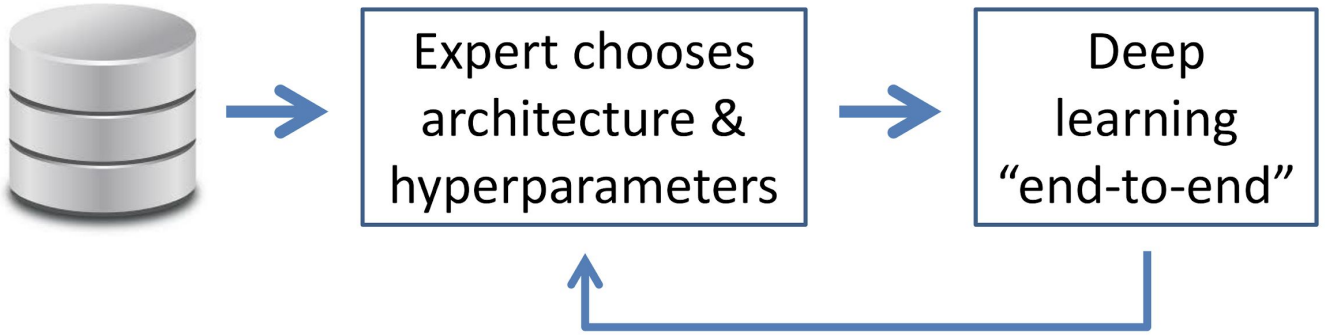
- Optimization algorithm, learning rates, momentum, batch normalization, batch sizes, dropout rates, weight decay, data augmentation, …

→ **Easily 20-50 design decisions**

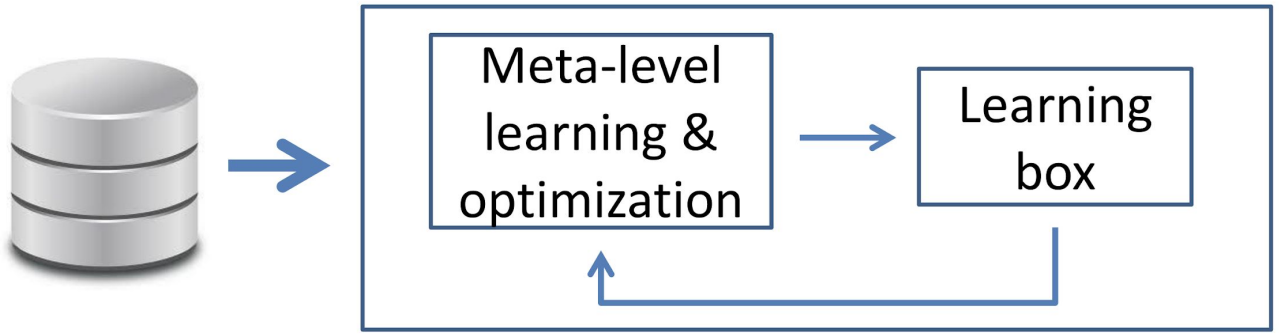
Slide credit: Hutter & Vanschoren

Deep learning and AutoML

Current deep learning practice



AutoML: true end-to-end learning

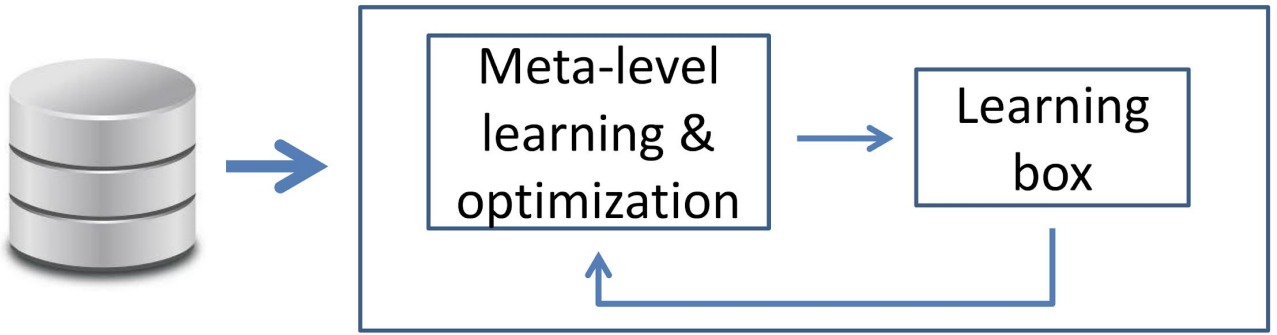


Slide credit: Hutter & Vanschoren

Learning box is not restricted to deep learning

- Traditional machine learning pipeline:
  - Clean & preprocess the data
  - Select / engineer better features
  - Select a model family
  - Set the hyperparameters
  - Construct ensembles of models
  - …

AutoML: true end-to-end learning



Slide credit: Hutter & Vanschoren

Hyperparameter optimization

Definition: Hyperparameter Optimization (HPO)

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of a ML algorithm A with domain $\boldsymbol{\Lambda}$,
- $\mathcal{L}(A_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ denote the loss of A, using hyperparameters $\boldsymbol{\lambda}$ trained on D_{train} and evaluated on D_{valid}.

The hyperparameter optimization (HPO) problem is to find a hyperparameter configuration $\boldsymbol{\lambda}^*$ that minimizes this loss:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \mathcal{L}(A_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$$

Slide credit: Hutter & Vanschoren

Types of hyperparameters

- Continuous
  - Example: learning rate
- Integer
  - Example: #units
- Categorical
  - Finite domain, unordered
    - Example 1: algo ∈ {SVM, RF, NN}
    - Example 2: activation function ∈ {ReLU, Leaky ReLU, tanh}
    - Example 3: operator ∈ {conv3x3, separable conv3x3, max pool, …}
  - Special case: binary

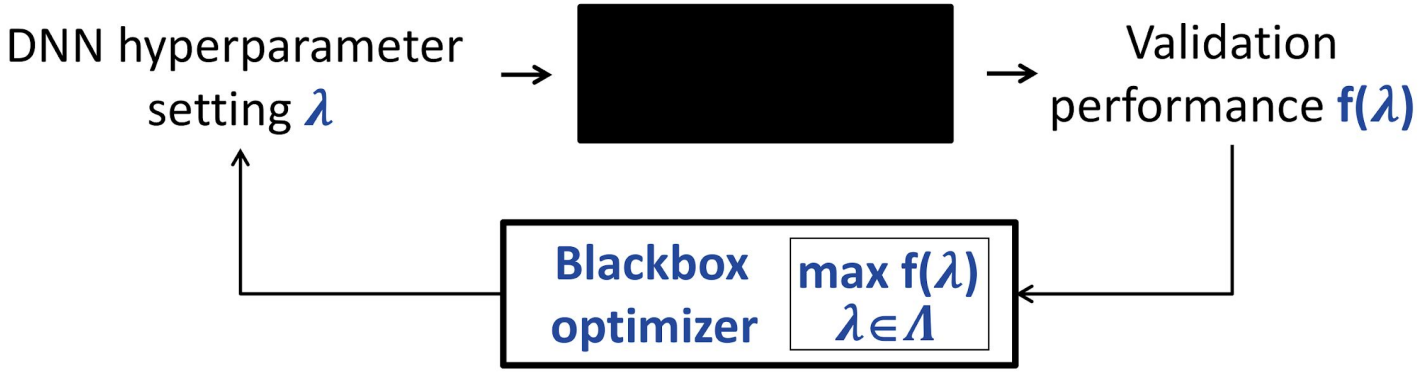
Slide credit: Hutter & Vanschoren

Conditional hyperparameters

- Conditional hyperparameters B are only active if other hyperparameters A are set a certain way
  - Example 1:
    - A = choice of optimizer (Adam or SGD)
    - B = Adam‘s second momentum hyperparameter (only active if A=Adam)
  - Example 2:
    - A = type of layer k (convolution, max pooling, fully connected, ...)
    - B = conv. kernel size of that layer (only active if A = convolution)
  - Example 3:
    - A = choice of classifier (RF or SVM)
    - B = SVM‘s kernel parameter (only active if A = SVM)

Slide credit: Hutter & Vanschoren

Blackbox hyperparameter optimization

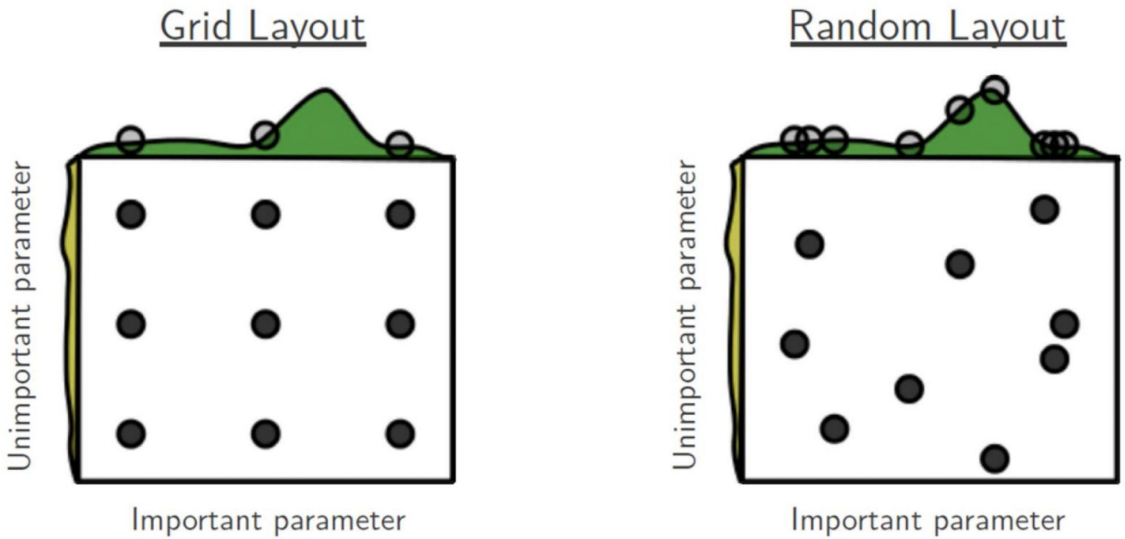


- The blackbox function is expensive to evaluate
  $\rightarrow$ sample efficiency is important

Slide credit: Hutter & Vanschoren

Grid search and random search

- Both completely uninformed
- Random search handles unimportant dimensions better
- Random search is a useful baseline



Slide credit: Hutter & Vanschoren

Image source: Bergstra & Bengio, JMLR 2012

Bayesian optimization

- Approach
  - Fit a proabilistic model to the function evaluations $\langle \lambda, f(\lambda) \rangle$
  - Use that model to trade off exploration vs. exploitation

- Popular since Mockus [1974]
  - Sample-efficient
  - Works when objective is nonconvex, noisy, has unknown derivatives, etc
  - Recent convergence results [Srinivas et al, 2010; Bull 2011; de Freitas et al, 2012; Kawaguchi et al, 2016]

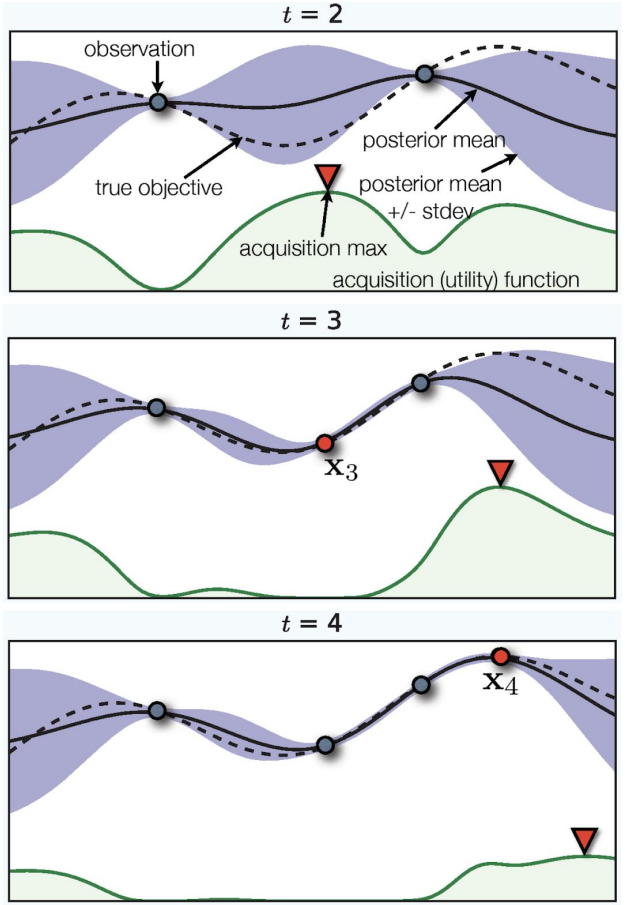


Image source: Brochu et al, 2010

Slide credit: Hutter & Vanschoren

Example: Bayesian optimization in AlphaGo

[Source: email from Nando de Freitas, today; quotes from Chen et al, forthcoming]

- During the development of AlphaGo, its many hyperparameters were tuned with Bayesian optimization multiple times.
- This automatic tuning process resulted in substantial improvements in playing strength. For example, prior to the match with Lee Sedol, we tuned the latest AlphaGo agent and this improved its win-rate from 50% to 66.5% in self-play games. This tuned version was deployed in the final match.
- Of course, since we tuned AlphaGo many times during its development cycle, the compounded contribution was even higher than this percentage.

Slide credit: Hutter & Vanschoren

AutoML Challenges for bayesian optimization

- Problems for standard Gaussian Process (GP) approach:
  - Complex hyperparameter space
    - High-dimensional (low effective dimensionality) [e.g., Wang et al, 2013]
    - Mixed continuous/discrete hyperparameters [e.g., Hutter et al, 2011]
    - Conditional hyperparameters [e.g., Swersky et al, 2013]
  - Noise: sometimes heteroscedastic, large, non-Gaussian
  - Robustness (usability out of the box)
  - Model overhead (budget is runtime, not #function evaluations)

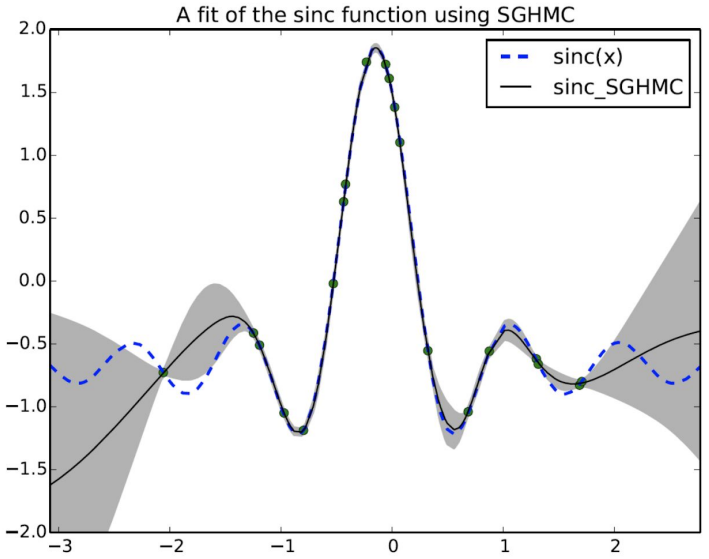
Slide credit: Hutter & Vanschoren

Bayesian optimization with neural networks

- Two recent promising models for Bayesian optimization
  - Neural networks with Bayesian linear regression using the features in the output layer [Snoek et al, ICML 2015]
  - Fully Bayesian neural networks, trained with stochastic gradient Hamiltonian Monte Carlo [Springenberg et al, NIPS 2016]

- Strong performance on low-dimensional HPOlib tasks

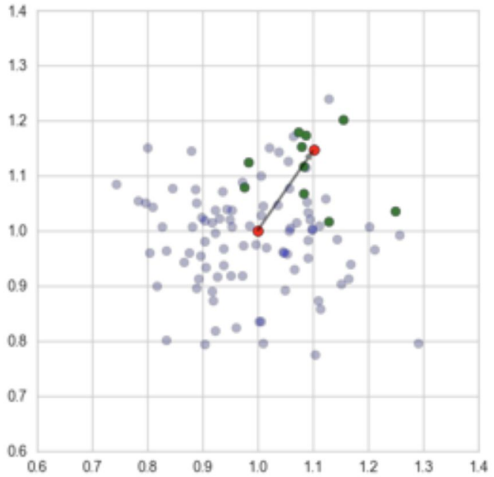
- So far not studied for:
  - High dimensionality
  - Conditional hyperparameters



Slide credit: Hutter & Vanschoren

Population-based methods

- Population of configurations
  - Maintain diversity
  - Improve fitness of population
- E.g, evolutionary strategies
  - Book: Beyer & Schwefel [2002]
  - Popular variant: CMA-ES [Hansen, 2016]
    - Very competitive for HPO of deep neural nets [Loshchilov & Hutter, 2016]
    - Embarassingly parallel
    - Purely continuous



Slide credit: Hutter & Vanschoren

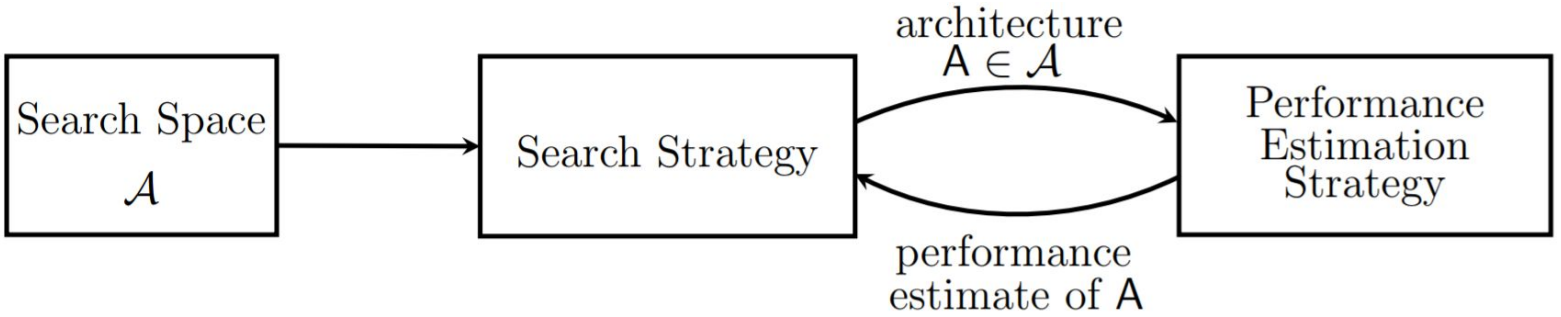
Open-source AutoML Tools based on HPO

- Auto-WEKA [Thornton et al, KDD 2013]
  - 768 hyperparameters, 4 levels of conditionality
  - Based on WEKA and SMAC
- Hyperopt-sklearn [Komer et al, SciPy 2014]
  - Based on scikit-learn & TPE
- Auto-sklearn [Feurer al, NIPS 2015]
  - Based on scikit-learn & SMAC / BOHB
  - Uses meta-learning and posthoc ensembling
  - Won AutoML competitions 2015-2016 & 2017-2018
- TPOT [Olson et al, EvoApplications 2016]
  - Based on scikit-learn and evolutionary algorithms
- H2O AutoML [so far unpublished]
  - Based on random search and stacking

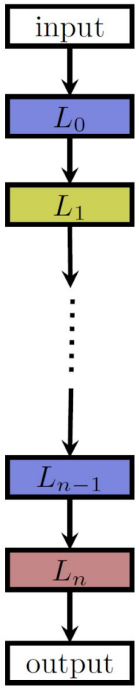
Slide credit: Hutter & Vanschoren

Neural architecture search

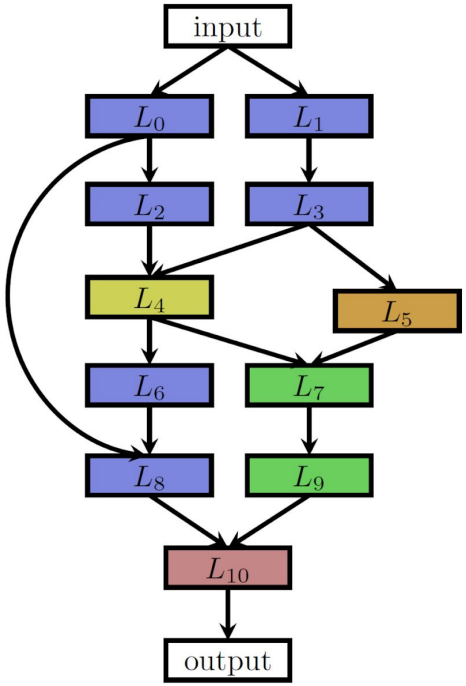
- Automatically find an optimal neural network architecture
- Sample architecture from search space, train it, evaluate it, and repeat.



Basic Neural Architecture Search Spaces



Chain-structured space
(different colours:
different layer types)

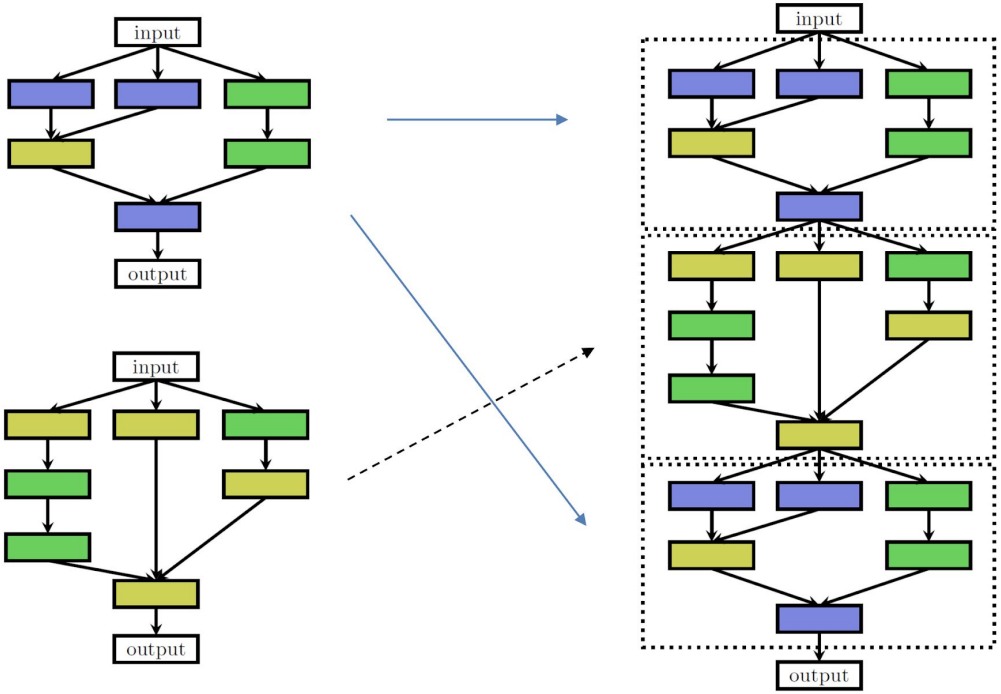


More complex space
with multiple branches
and skip connections

Slide credit: Hutter & Vanschoren

Cell Search Spaces

Introduced by Zoph et al [CVPR 2018]



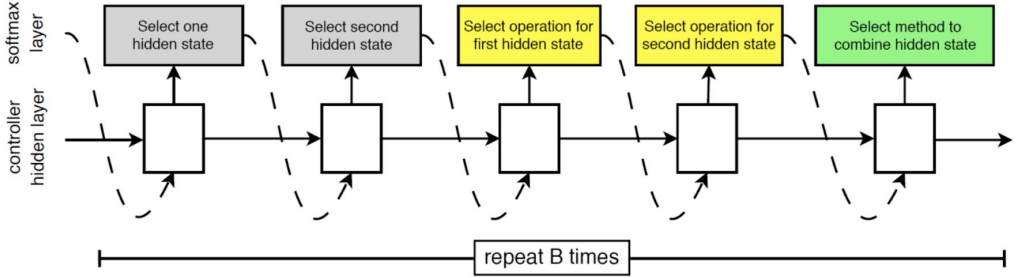
Two possible cells

Architecture composed of stacking together individual cells

Slide credit: Hutter & Vanschoren

NAS as Hyperparameter Optimization

- Cell search space by Zoph et al [CVPR 2018]



  - 5 categorical choices for Nth block:
    - 2 categorical choices of hidden states, each with domain {0, ..., N-1}
    - 2 categorical choices of operations
    - 1 categorical choice of combination method

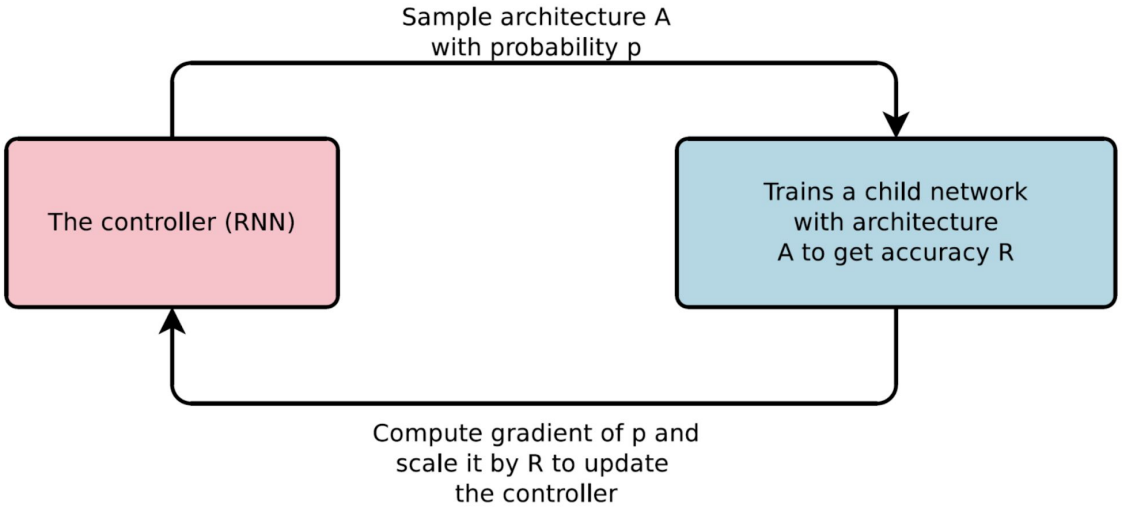
    → Total number of hyperparameters for the cell: 5B (with B=5 by default)

- Unrestricted search space
  - Possible with conditional hyperparameters (but only up to a prespecified maximum number of layers)
  - Example: chain-structured search space
    - Top-level hyperparameter: number of layers L
    - Hyperparameters of layer k conditional on L >= k

Slide credit: Hutter & Vanschoren

Reinforcement Learning

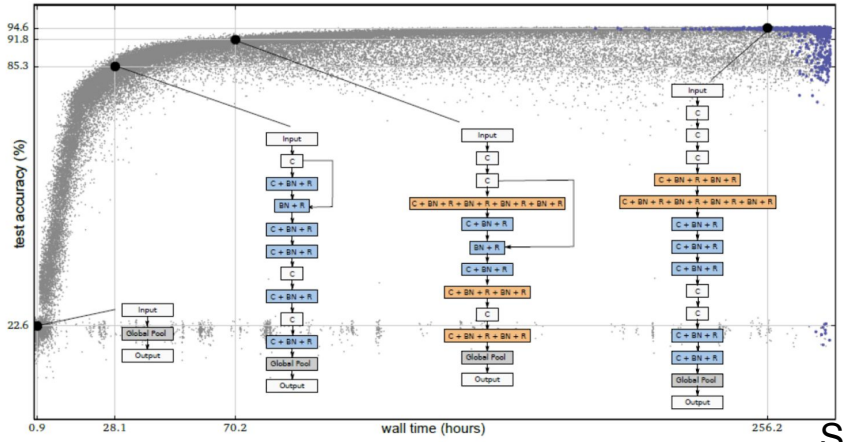
- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands
    - **800 GPUs for 3-4 weeks, 12.800 architectures evaluated**



Slide credit: Hutter & Vanschoren

Evolution

- Neuroevolution (already since the 1990s)
  - Typically optimized both architecture and weights with evolutionary methods
    [e.g., Angeline et al, 1994; Stanley and Miikkulainen, 2002]
  - Mutation steps, such as adding, changing or removing a layer
    [Real et al, ICML 2017; Miikkulainen et al, arXiv 2017]



Slide credit: Hutter & Vanschoren

Main approaches for making NAS efficient

- Weight inheritance & network morphisms
- Weight sharing & one-shot models
- Multi-fidelity optimization
  [Zela et al, AutoML 2018, Runge et al, MetaLearn 2018]
- Meta-learning [Wong et al, NIPS 2018]

Slide credit: Hutter & Vanschoren

AutoML: Further Benefits and Concerns

+ Democratization of data science
+ We directly have a strong baseline
+ We can codify best practices
+ Reducing the tedious part of our work,
  freeing time to focus on problems humans do best
  (creativity, interpretation, …)

− People will use it without understanding anything

Slide credit: Hutter & Vanschoren

Summary

- Adversarial attacks
  - Neural networks can be easily fooled by the “right noise” regardless of the input still being recognizable by human perception
- AutoML
  - Finding the right hyperparameters can be also automated.
    - Example: Search for optimal architecture, optimizer, loss weighing, etc