DM873 / DS809

Deep Learning

Fall 2022

Regularization

Regularization

- Parameter Penalties
- Data Augmentation
- Early stopping
- Bagging
- Dropout
- Adversarial training

What we have seen so far

- Last lecture, we discussed the capacity of a model and the risk of overfitting.
  - There are many way of adjusting the model's capacity
    - fit polynomials of different degree
    - Change the number of layers/neurons in a network
    - Limit the training time
    - …
- Deep Neural Networks have very high capacity and thus we need measures to tame them
  - Due to the complexity, we need more clever approaches than just changing the architecture
  - Way harder to control than with simple linear regression

What is Regularization?

- Central problem of ML is to design algorithms that will perform well not just on training data but on new inputs as well

- Regularization is:
  - "any modification we make to a learning algorithm to reduce its generalization error but not its training error"
  - Reduce test error even at the expense of increasing training error

- Some goals of regularization
  1. Encode prior knowledge
  2. Express preference for simpler model
  3. Needed to make underdetermined problem determined

Model Types and Regularization

- Three types of model families
  1. Excludes the true data generating process (Implies underfitting and inducing high bias)
  2. Matches the true data generating process
  3. Overfits (Includes true data generating process but also many other processes)

- Goal of regularization is to take model from third regime to second

- Best fitting model obtained not by finding the right number of parameters
- Instead, best fitting model is a large model that has been regularized appropriately

Regularization

- **Parameter Penalties**
- **Data Augmentation**
- **Early stopping**
- **Bagging**
- **Dropout**
- **Adversarial training**

Limiting Model Capacity

- Regularization has been used for decades prior to advent of deep learning
- Linear- and logistic-regression allow simple, straightforward and effective regularization strategies:
  - Adding a parameter norm penalty $\Omega(\theta)$ to the objective function J:

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

- with α is a hyperparameter that weight the relative contribution of the norm penalty term Ω
- Setting α to 0 results in no regularization. Larger values correspond to more regularization

Norm Penalty

- When our training algorithm minimizes the regularized objective function J and some measure of the size of the parameters θ
- Different choices of the parameter norm Ω can result in different solutions preferred
- Norm penalty Ω penalizes only weights at each layer and leaves biases unregularized
  - Biases require less data to fit than weights
  - Each bias controls only a single variable
- Let $\boldsymbol{w}$ indicate all weights affected by norm penalty, $\boldsymbol{\theta}$ denotes both $\boldsymbol{w}$ and biases

L^2 parameter Regularization

- Simplest and most common kind
- Called Weight decay
- Drives weights closer to the origin by adding a regularization term to the objective function

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

- In other communities also known as **ridge regression** or Tikhonov regularization

A closer look

- Objective function (with no bias parameter)

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Corresponding Gradient:

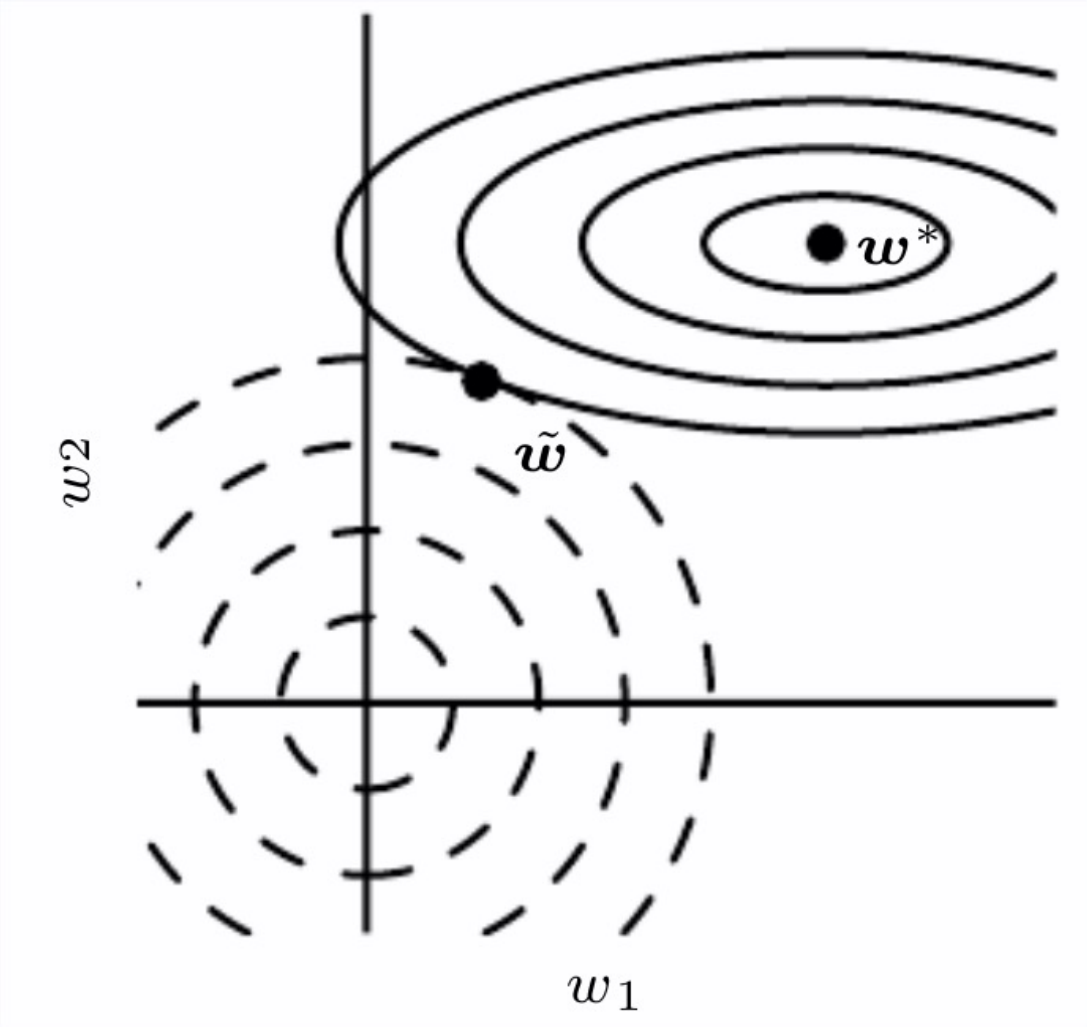
$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- To perform single gradient step, perform update:

$$\boldsymbol{w}' \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- We have modified learning rule to shrink $\boldsymbol{w}$ by constant factor $1 - \epsilon\alpha$ at each step

An illustration of the effect of weight decay



L^1 Regularization

- While L^2 weight decay is the most common form of weight decay there are other ways to penalize the size of model parameters
- L^1 regularization is defined as

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

- Which is the sum of the absolute values of the individual parameters

A closer look

- Objective function (with no bias parameter)

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Corresponding Gradient:

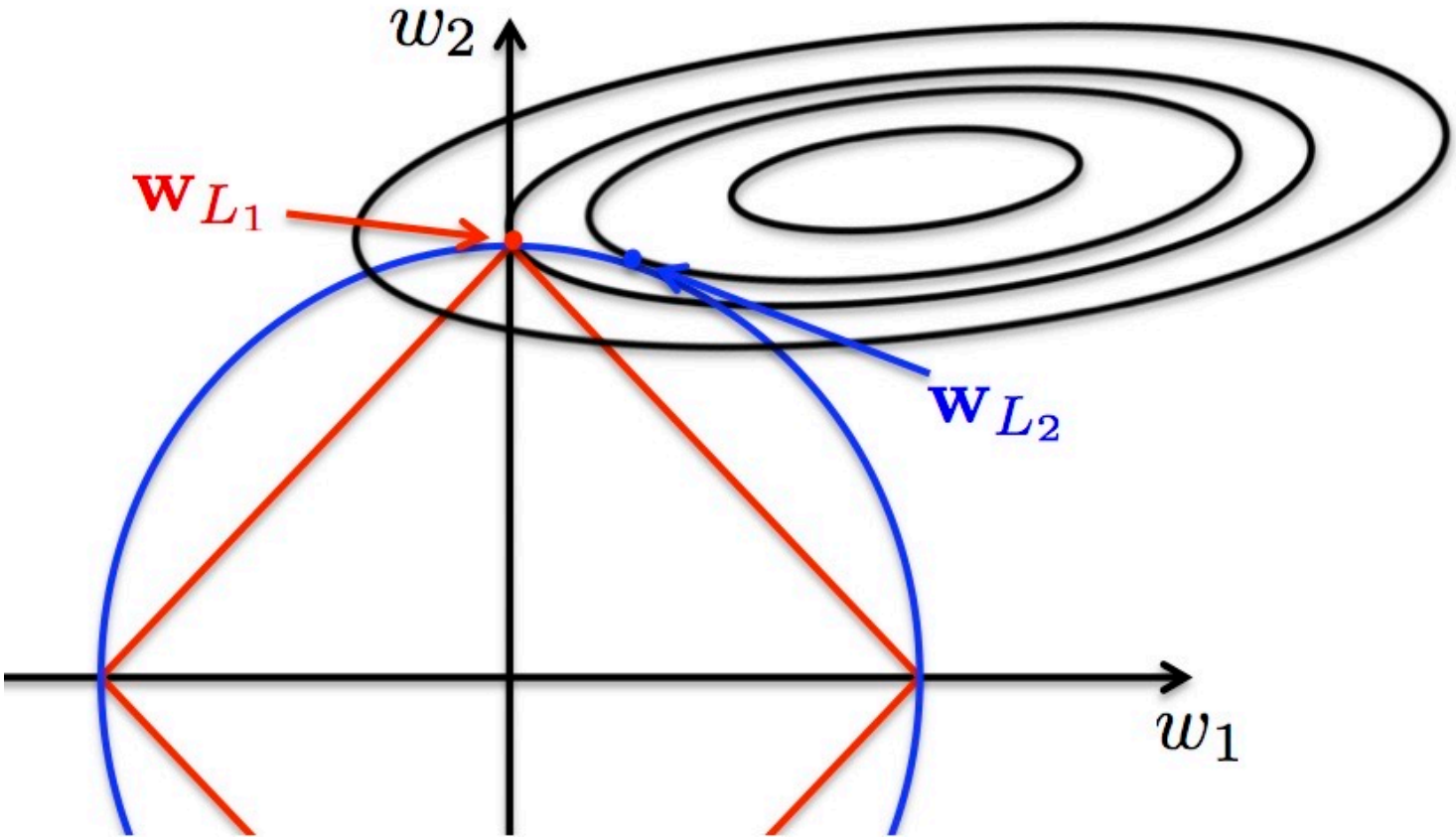
$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Where $\text{sign}(\boldsymbol{w})$ is simply the element-wise sign of $\boldsymbol{w}$
  - Always a strong gradient unless $w_i = 0$
  - Leads to a sparse solution

Sparsity and Feature Selection

- The sparsity property induced by L^1 regularization has been used extensively as a feature selection mechanism
- Feature selection simplifies an ML problem by choosing subset of available features
- LASSO (Least Absolute Shrinkage and Selection Operator) integrates an L^1 penalty with a linear model and least squares cost function
- The L^1 penalty causes a subset of the weights to become zero, suggesting that those features can be discarded

L^1 vs. L^2 Regularization



Regularization

- Parameter Penalties
- Data Augmentation
- Early stopping
- Bagging
- Dropout
- Adversarial training

More Data is Better

- Best way to make a ML model to generalize better is to train it on more data
- In practice amount of data is limited
- **Get around the problem by creating synthesized data**
- For some ML tasks it is straightforward to synthesize data
- Especially popular for classification/object recognition

Data Augmentation for Classification

- Data augmentation is easiest for classification
  - Classifier takes high-dimensional input $\boldsymbol{x}$ and summarizes it with a single category identity y
  - Main task of classifier is to be invariant to a wide variety of transformations

- Generate new samples $(\boldsymbol{x}, y)$ just by transforming inputs
  - Approach not easily generalized to other problems
    - E.g.: For density estimation problem: generating new data requires solving density estimation first
  - Good example: Images are high-dimensional and include a variety of variations, may easily simulated
  - Translating the images a few pixels can greatly improve performance
  - Rotating and scaling are also effective
  - Some other augmentations are hard to perform (e.g. out-of-pane rotation)

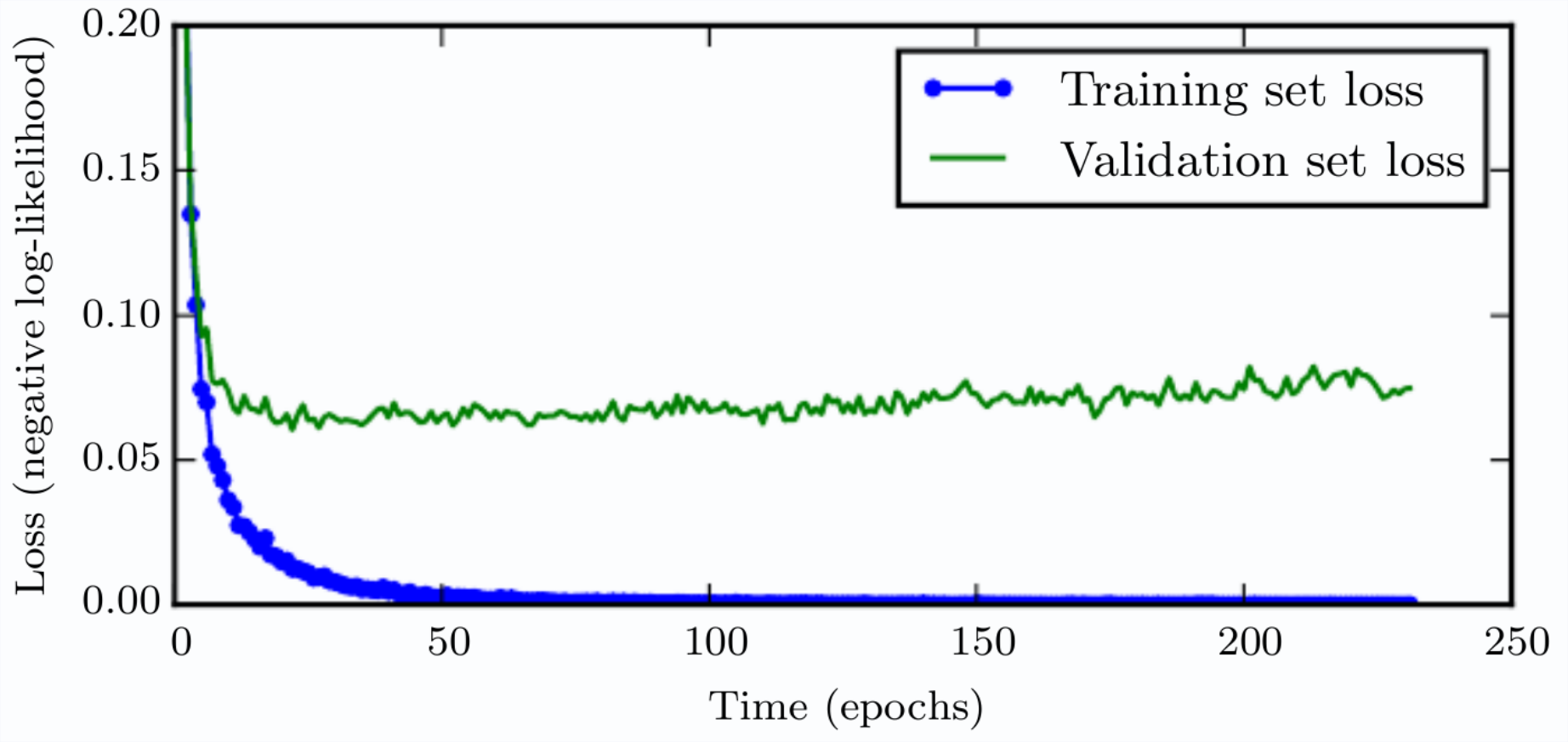
Injecting Noise

- Injecting noise into the input of a neural network can be seen as data augmentation
- **Neural networks are not robust to noise**
- To improve robustness, train them with random noise applied to their inputs
- Noise can also be applied to hidden units
- Dropout, a powerful regularization strategy, can be viewed as constructing new inputs by multiplying by noise

Regularization

- Parameter Penalties
- Data Augmentation
- Early stopping
- Bagging
- Dropout
- Adversarial training

Typical Learning Curves



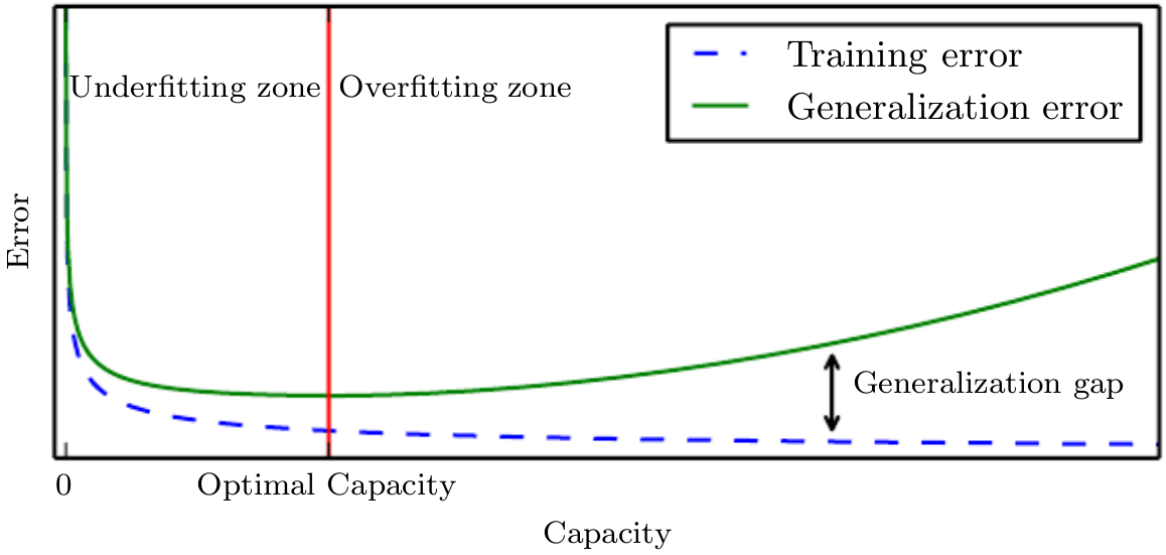
- In this example, we train a maxout network on MNIST. Observe that the training objective decreases consistently over time, but the validation set average loss eventually begins to increase again, forming an asymmetric U-shaped curve.

Early Stopping

- We can obtain a model with better validation set error (and thus better test error) by returning to the parameter setting at the point of time with the lowest validation set error
- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather than the latest set
- It is the most common form of regularization in deep learning due to its effectiveness and its simplicity

Early Stopping as Hyperparameter Selection

- We can think of early stopping as a very efficient hyperparameter selection algorithm
  - In this view no. of training steps is just a hyperparameter
  - This hyperparameter has a U-shaped validation set performance curve
  - Most hyperparameters have such a U-shaped validation set performance curve, as seen below



Costs of Early Stopping

- Cost of this hyperparameter is running validation evaluation periodically during training
  - Ideally done in parallel to training process on a separate machine
  - Separate CPU or GPU from main training process, Or using small validation set or validating set less frequently
- Need to maintain a copy of the best parameters
  - This cost is negligible because they can be stored on a slower, larger memory
  - E.g., training in GPU, but storing the optimal parameters in host memory or on a disk drive

Early Stopping as Regularization

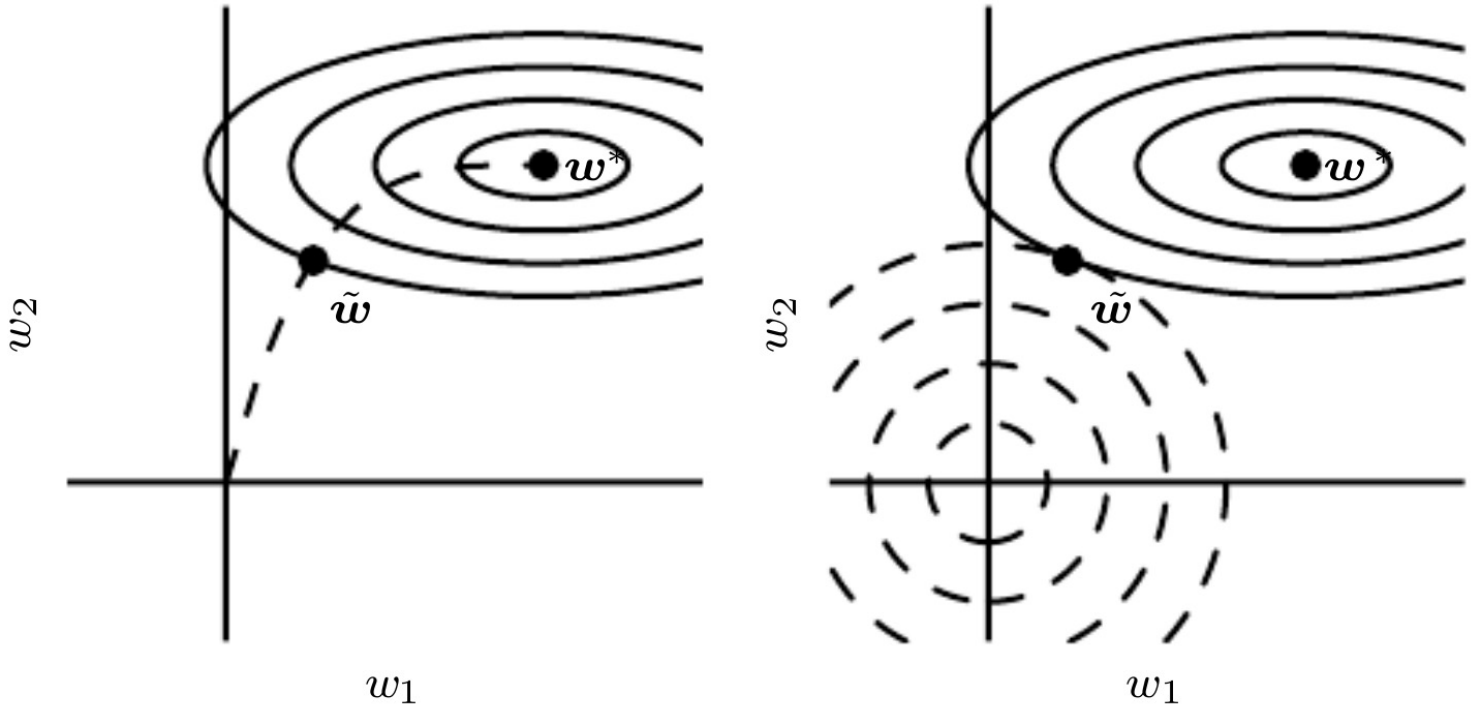
- Early stopping is an unobtrusive form of regularization
- It requires almost no change to the underlying training procedure, the objective function, or the set of allowable parameter values
- So it is easy to use early stopping without damaging the learning dynamics
- In contrast to weight decay, where we must be careful not to use too much weight decay
  - Otherwise we trap the network in a bad local minimum corresponding to pathologically small weights

Use of a second training step

- Early stopping requires a validation set
  - Thus some training data is not fed to the model
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed

- There are two basic strategies for including the test data
  1. Reinitialize the model to the untrained state and run for the same number of steps again
  2. Continue Training
     - Keep all parameters and continue with now the entire dataset
     - When to stop?
     - The training error will increase with including the entire dataset, stop when we reach the same training error as before

Early Stopping vs L^2 Regularization



- We now restrict the "distance" the weights can travel to find an optimal solution

Regularization

- Parameter Penalties
- Data Augmentation
- Early stopping
- Bagging
- Dropout
- Adversarial training

What is Bagging (Bootstrap Aggregating)

- It is a technique for reducing generalization error by combining several models
  - Idea is to train several models separately, then have all the models vote on the output for test examples
- This strategy is called model averaging
- Techniques employing this strategy are known as ensemble methods
- Model averaging works because different models will not make the same mistake

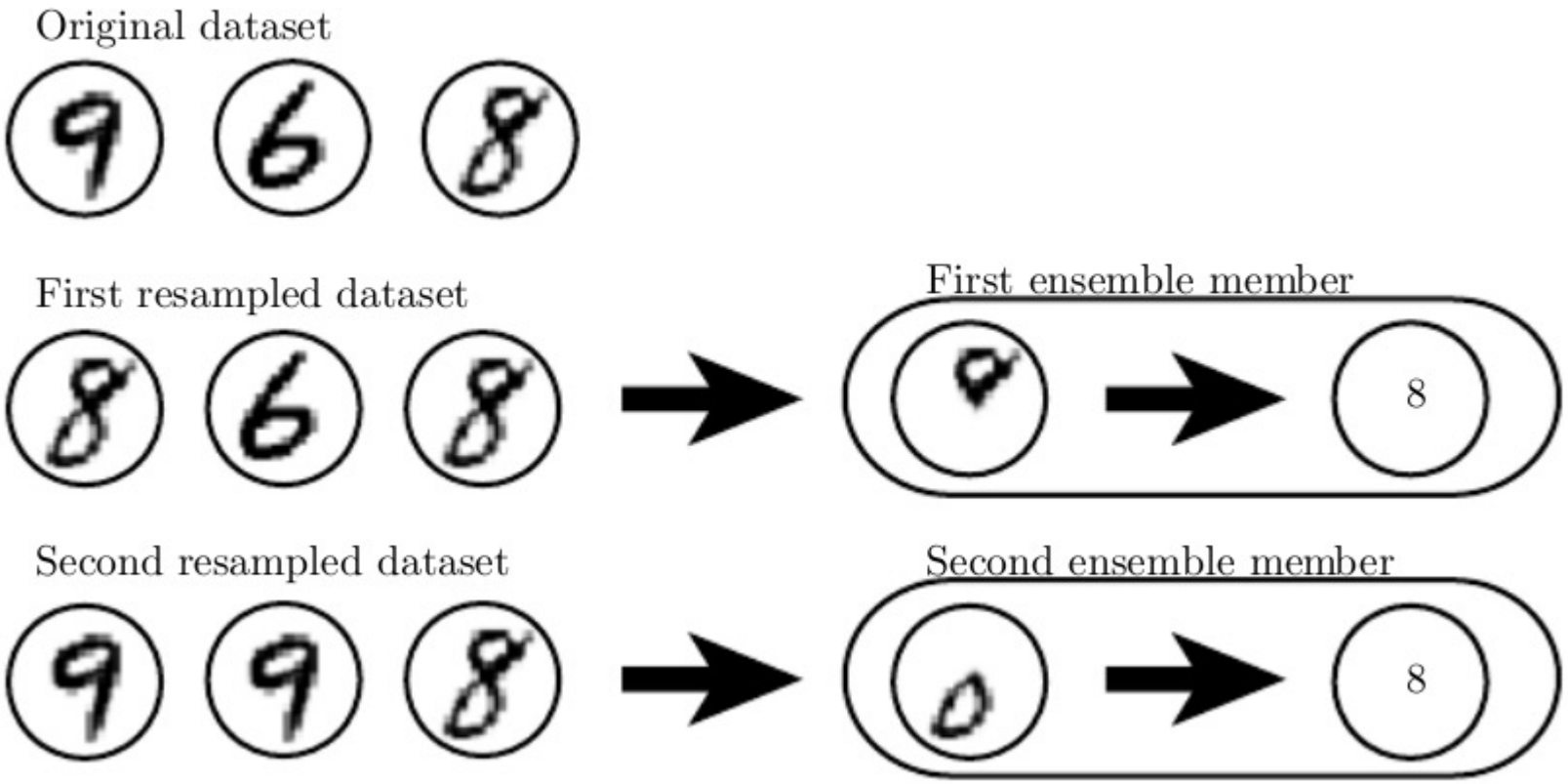
Ensemble vs Bagging

- Different ensemble methods construct the ensemble of models in different ways
- Example: each member of ensemble could be formed by training a completely different kind of model using a different algorithm or objective function
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times

The Bagging Technique

- Given training set D of size N, generate k data sets of same no of examples as original by sampling with replacement
- Some observations may be repeated in D_i, some others are missing. This is known as a bootstrap sample.
- The differences in examples will result in differences between trained models
- The k models are combined by averaging the output (for regression) or voting (for classification)

Example


Original dataset
First resampled dataset
First ensemble member
8
Second resampled dataset
Second ensemble member
8

Bagging in Neural Nets

- Neural nets reach a wide variety of solution points
  - Thus they benefit from model averaging when trained on the same dataset
  - Differences in:
    - random initializations
    - random selection of minibatches, in hyperparameters,
  - cause different members of the ensemble to make partially independent errors

- Model averaging is a reliable method for reducing generalization error
  - Machine learning contests are usually won by model averaging over dozens of models, e.g., the Netflix grand prize
- But comes with significant computational costs

Regularization

- Parameter Penalties
- Data Augmentation
- Early stopping
- Bagging
- Dropout
- Adversarial training

Overfitting in Deep Neural Nets

- Deep nets have many non-linear hidden layers
  - Making them very expressive to learn complicated relationships between inputs and outputs
  - But with limited training data, many complicated relationships will be the result of training noise
  - So they will exist in the training set and not in test set even if drawn from same distribution

- Many methods developed to reduce overfitting
  - Early stopping with a validation set
  - Weight penalties (L^1 and L^2 regularization)
  - Soft weight sharing

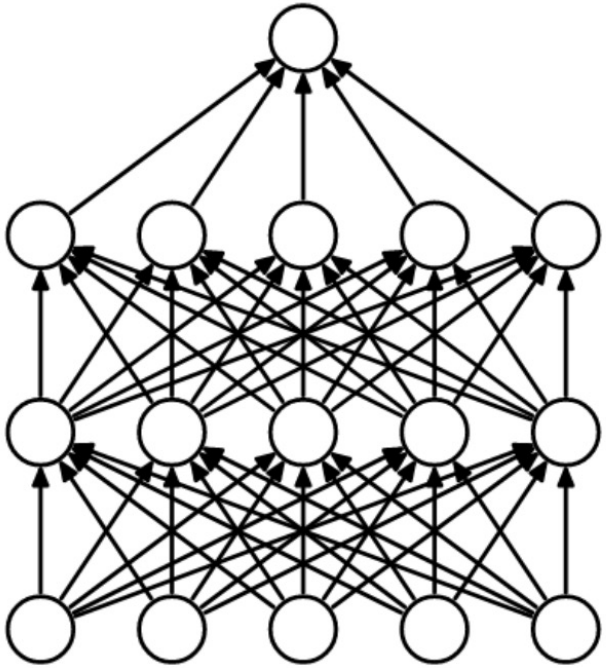
Dropout as Bayesian Approximation

- Best way to regularize a fixed size model is:
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
  - This would be the Bayesian approach

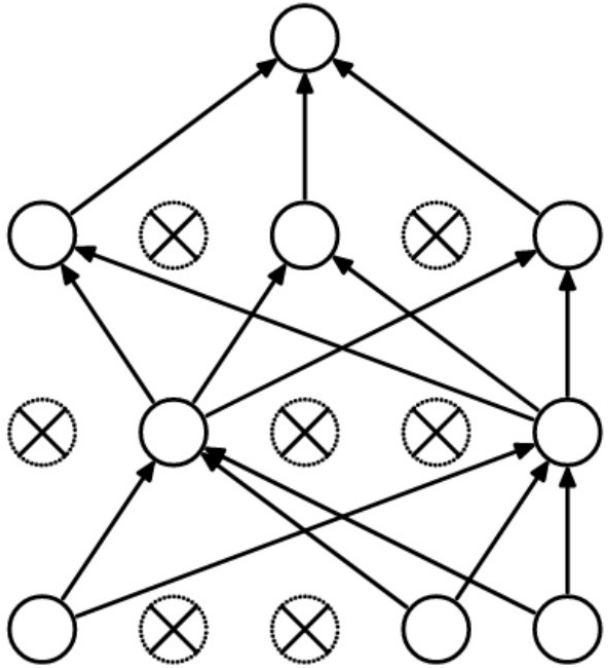
- Dropout does this using considerably less computation
  - Approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters
  - Dropout makes it practical to apply bagging to very many large neural networks

Creating new Models by Removing Units

- Dropout trains an ensemble of subnetworks
  - formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero



(a) Standard Neural Net



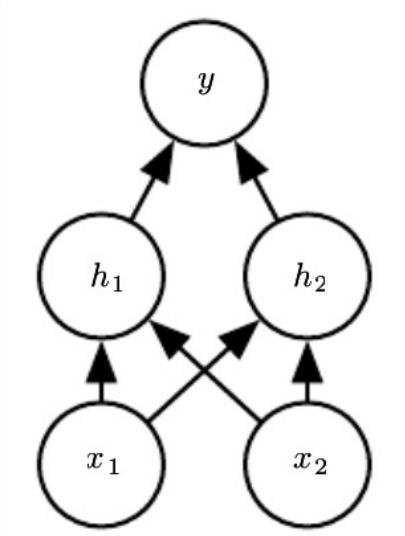
(b) After applying dropout.

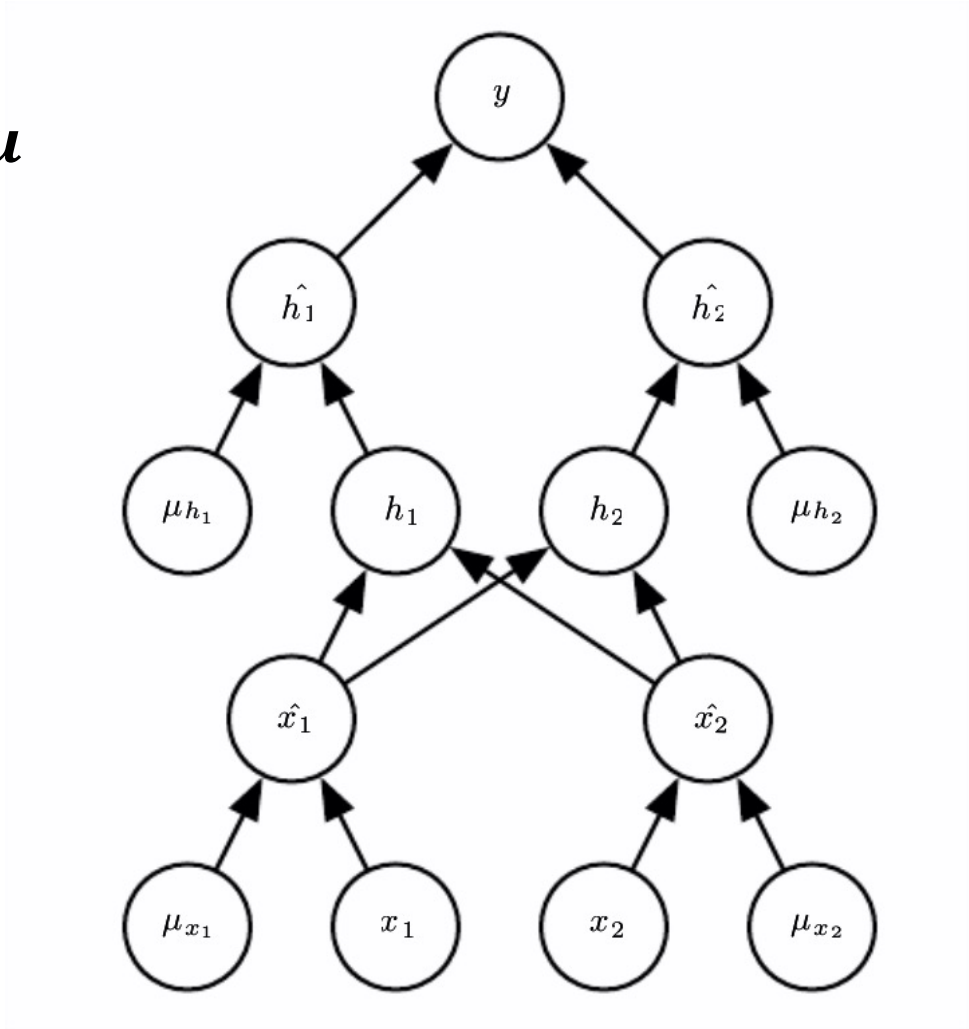
Mask for dropout training

- To train with dropout we use minibatch based learning algorithm that takes small steps such as SGD
- At each step randomly sample a binary mask
  - Probability of including a unit is a hyperparameter
  - Normally: 0.5 for hidden units and 0.8 for input units
- We run forward & backward propagation as usual
- Equivalent to randomly selecting a subnetwork of the entire network

Example

- Modified network incorporating a binary vector $\boldsymbol{\mu}$
- Training and evaluation as normal





Formal Description

- Suppose that mask vector μ specifies which units to include
- Cost of the model is specified by
$$J(\boldsymbol{\theta}, \boldsymbol{\mu})$$
- Dropout training consists of minimizing
$$\mathbb{E}_{\boldsymbol{\mu}}\big(J(\boldsymbol{\theta}, \boldsymbol{\mu})\big)$$
- Expected value contains exponential no. of terms
- We can get an unbiased estimate of its gradient by sampling values of $\boldsymbol{\mu}$

Dropout Prediction

- Submodel defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p(y|\boldsymbol{x}, \boldsymbol{\mu})$

- At testing time after training has finished, we would ideally like to find a sample average of all possible 2^n dropped-out networks
  - This is unfeasible for any realistic number n

- Approximation:
  - Each node's output weighted by a factor of p, so the expected value of the output of any node is the same as in the training stages.
  - This is the biggest contribution of the dropout method:
    - although it simulates 2^n different neural nets, at test time only a single network needs to be evaluated and tested

Another Interpretation of Dropout

- So far we have described dropout purely as a means of performing efficient, approximate an ensemble method

- Dropout trains an ensemble of models that share hidden units
  - Each hidden unit must be able to perform well regardless of which other hidden units are in the model
  - Hidden units must be prepared to be swapped and interchanged between models

- **Dropout thus regularizes each hidden unit to be not merely a good feature but a feature that is good in many contexts.**

Regularization

- Parameter Penalties
- Data Augmentation
- Early stopping
- Bagging
- Dropout
- Adversarial training

The Understanding of Deep Nets

- In many cases, neural networks have begun to reach human level performance when evaluated on an i.i.d. test set
- Have they reached human level understanding?
- To probe the level of understanding we can construct examples that the model misclassifies
- Even neural networks that perform at human level accuracy have a 100% error rate on examples intentionally constructed!

Adversarial examples

- An optimization procedure is used to search for an input $\boldsymbol{x}$' near data point $\boldsymbol{x}$ such that the model output is very different at $\boldsymbol{x}$'
  - In many cases, $\boldsymbol{x}$' can be so similar to $\boldsymbol{x}$ that a human observer cannot tell the difference between the original example and the adversarial example

- **But the network makes a highly different prediction**

- Adversarial examples have many implications
  - E.g., they are useful in computer security since they are hard to defend against
  - They are interesting in the context of regularization
    - Using adversarial perturbed samples we can reduce error rate on test set

Example

$+ .007 \times$

$=$

$\boldsymbol{x}$

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

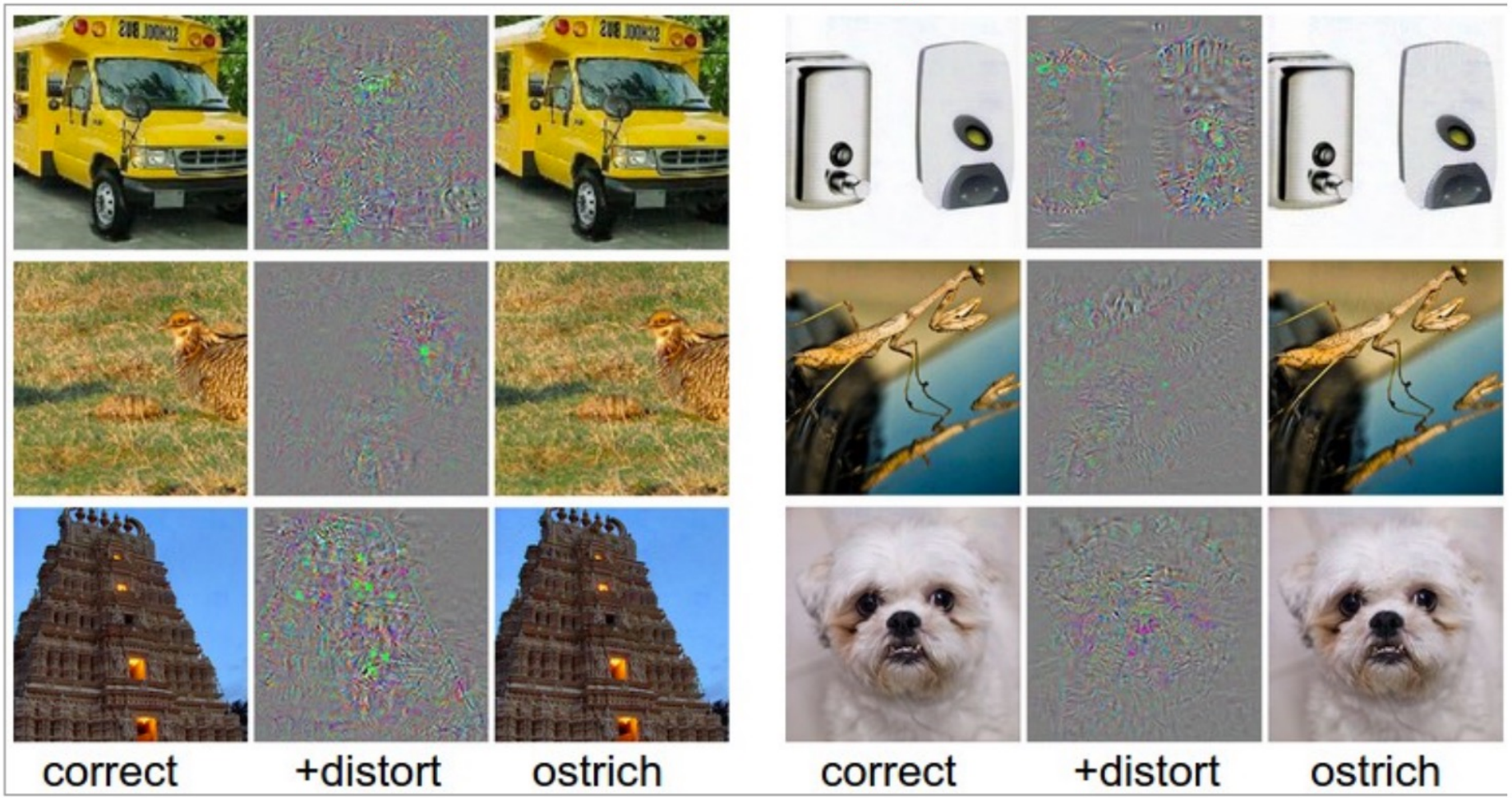
$\boldsymbol{x} + \epsilon\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

y ="panda"
with 58% confidence

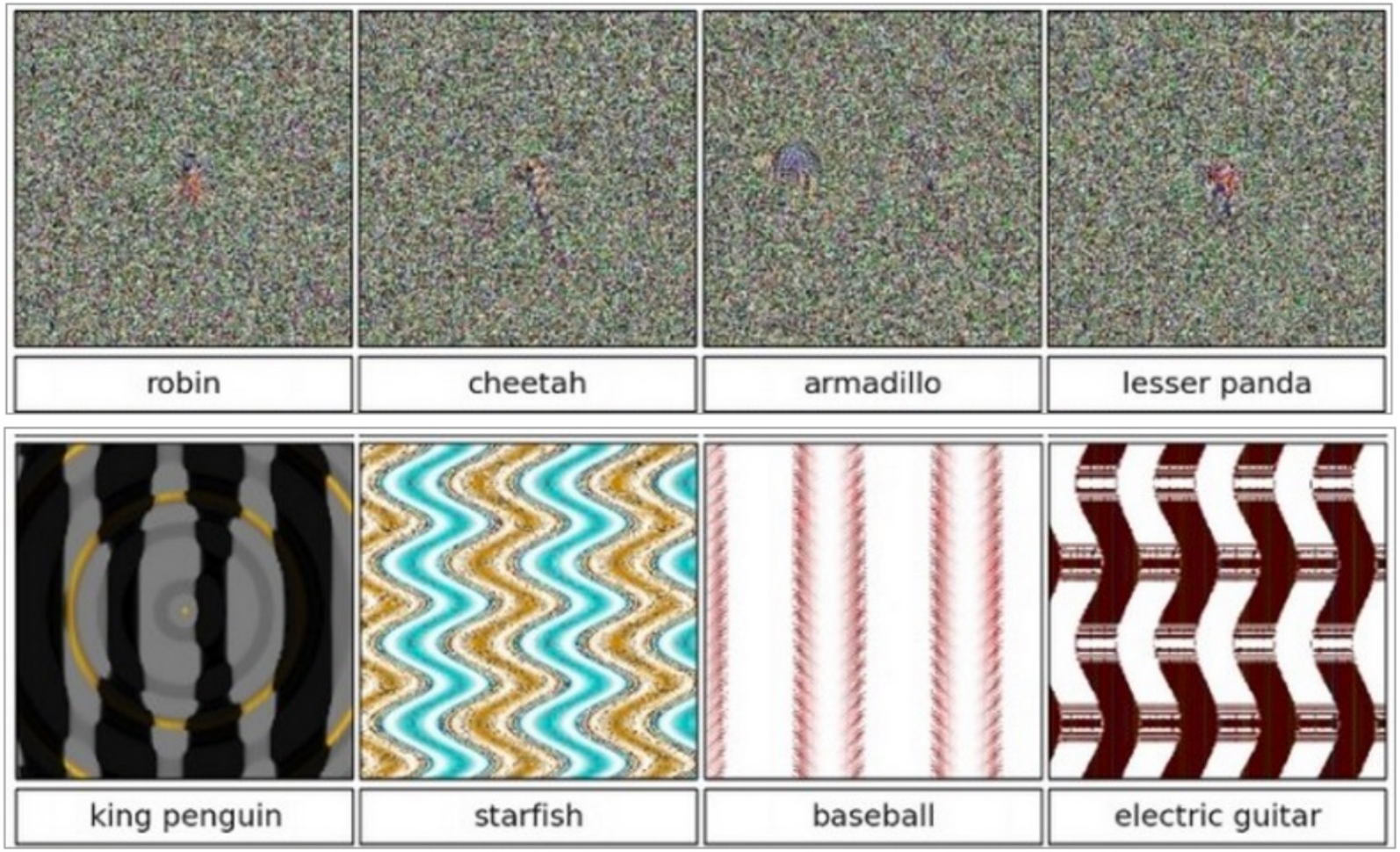
y ="nermatode"
With 8.2% confidence

y ="gibbon"
With 99% confidence

Example



Example



These images are classified with >99.6% confidence as the shown class by a Convolutional Network.

Cause of adversarial examples

- Primary cause is excessive linearity
  - Neural networks are built primarily out of linear building blocks
  - The overall function often proves to be linear
  - Linear functions are easy to optimize
  - But the value of a linear function can change rapidly with numerous inputs

- If we change input by ϵ then a linear functions with weights $\boldsymbol{w}$ can change by $\epsilon\|\boldsymbol{w}\|$ which can be very large in high-dimensional spaces

Adversarial Training

- Adversarial training discourages highly sensitive local behavior
- By encouraging network to be locally constant in the neighborhood of the training data
- This can be seen as a way of explicitly introducing a local constancy prior into supervised neural nets
- Adversarial training illustrates the power of using a large function family in combination with aggressive regularization
  - Purely linear models, like logistic regression, are unable to resist adversarial examples because they are forced to be linear

Virtual Adversarial Examples

- Adversarial examples generated with using not the true label but a label provided by a trained model are called **Virtual Adversarial Examples**

- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}$'

- This encourages the classifier to learn a function that is robust to small changes anywhere along the manifold
  - Assumption motivating this approach:
    - different classes lie on disconnected manifolds
    - A small perturbation should not be able to jump from one class manifold to another class manifold