Generalization and Regularization

Matias Valdenegro-Toro

Hochschule Bonn-Rhein-Sieg & German Research Centre for Artificial Intelligence
(Bremen, Germany)

May 28, 2018

This lecture covers two topics:

Generalization

Ability of a learning model to predict accurately for inputs that are considerably different from the ones in the training set.

Regularization

Techniques to control the "complexity" of the model in order to increase its generalization power.

Generalization

Introduction

- The key idea behind Machine Learning is that a concept can be learned from a limited training set, and still be "useful" if used on completely new inputs.
- But how to make sure that the model has learned the "right things" and not noise in the data?
- This concept is called Generalization, the ability of a model to *generalize* to samples outside its training set.
- There are many questions that stem from this concept:
  - How to measure generalization?
  - How to design the model and training set to improve generalization?
  - What to do when the model does not generalize (fails)?

Learning Performance

The first thing is to define how to measure the performance of a learning model.

Losses

Objective function that guides learning during the optimization process. It defines the task to be learned and the quality of solutions. Usually it has to be differentiable.

Metrics

Measurements of quality that let the ML developer evaluate the learning process' success. Usually non-differentiable. Losses can be Metrics as well.

Loss Functions - Classification

Categorical Cross-Entropy

Used for classification.

For this loss, labels y^c should be one-hot encoded.

$$L(y, \hat{y}) = -\sum_i \sum_c y_i^c \log(\hat{y}_i^c)$$

yi is the ground truth

Binary Cross-Entropy

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Loss Functions - Regression

Mean Average Error (MAE)

$$L(y, \hat{y}) = n^{-1} \sum_i |\hat{y}_i - y_i|$$

Mean Square Error (MSE)

This loss often converges to the mean/median of the targets.

$$L(y, \hat{y}) = n^{-1} \sum_i (\hat{y}_i - y_i)^2$$

Loss Functions - Others

Kullback-Leibler Divergence

crossentropy is a particular case of this.

Distance measure between probability distributions.

$$L(p,q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)} dx\right) \qquad L(y,\hat{y}) = \sum_i y_i \log\left(\frac{y_i}{\hat{y}_i}\right)$$

Cosine Similarity

Used to compare vectors

$$L(y,\hat{y}) = \sum_i \frac{\text{dot}(\hat{y}_i, y_i)}{||\hat{y}_i|| ||y_i||}$$

Metrics - Classification

Accuracy

$$M(y, \hat{y}) = n^{-1} \sum_i 1[y_i = \hat{y}_i]$$

Top-k Accuracy

Accuracy computed as considering any of the top k class predictions as correct. Typically used with classifiers that can rank their class predictions.

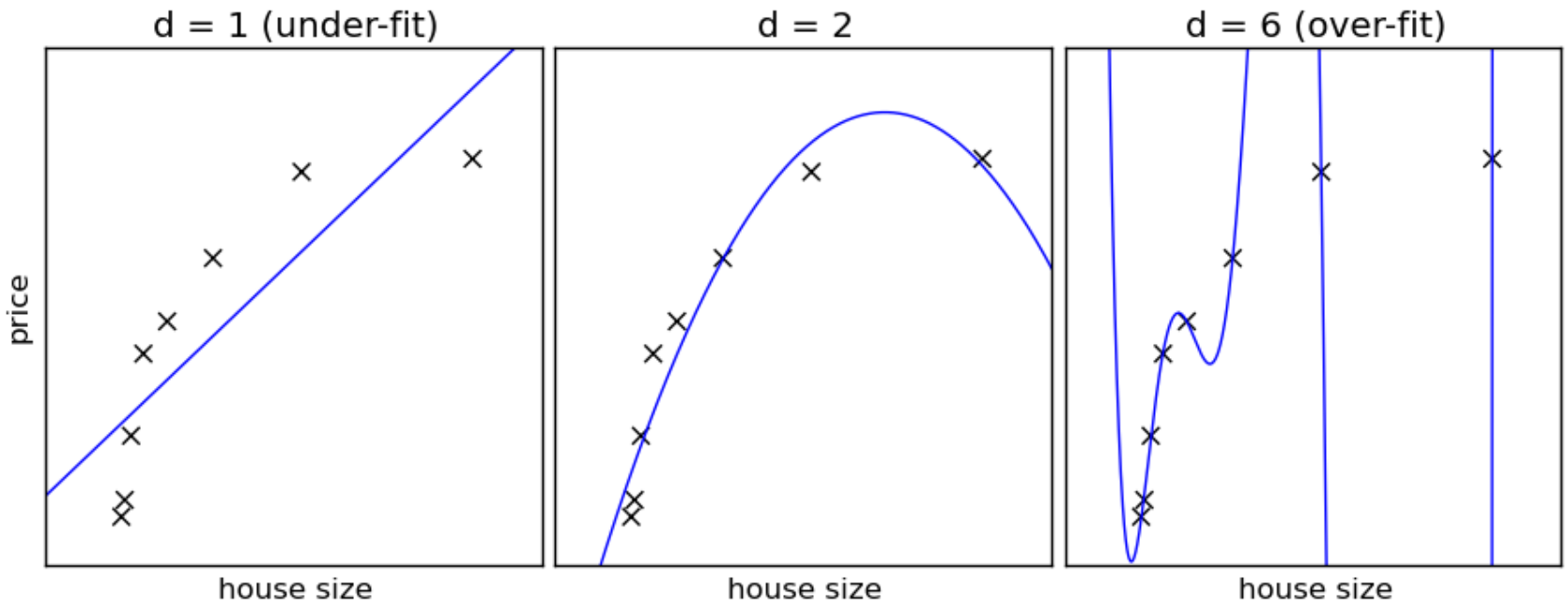
Learning Capacity

It is an abstract "measurement" of the ability of a model to represent different functions. A higher capacity model is able to represent highly non-linear functions, while a low capacity model may not.
Informally it could be defined as the "size" of the function or hypothesis space generated by the model.

Overfitting

- Overfitting is when the trained model memorizes undesirable patterns from training data.
- Like models learning noise in the data or irrelevant features.
- It reduces generalization performance. Models perform badly on the test set or on different data.
- Happens when model has too much learning capacity or it is trained for too long.

Overfitting



Overfitting

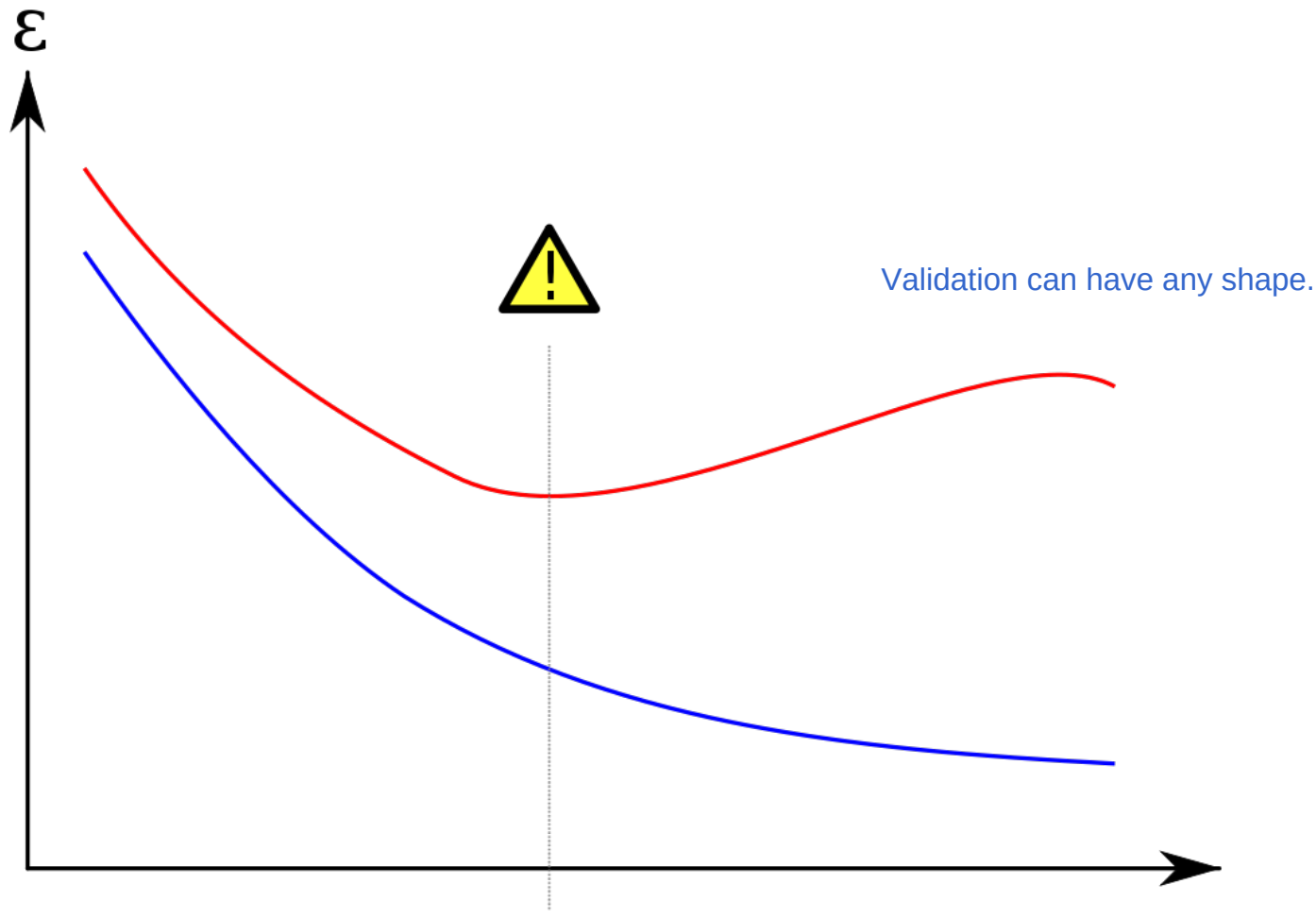
Golden Rule

When training a model, compare the loss value on the training set and on the testing set.

If the loss on the test set is **much larger** than in the training set, then the model is overfitting, specially if the training loss is low.

It is also normal that the test loss is **slightly** larger than training loss.

Overfitting



Blue is training loss, Red is validation/test loss.

VC Dimension

Shattering

Given a binary classifier f with parameters θ and a set of points $(x_1, x_2, ..., x_n)$. The classifier is said to *shatter* these points if for any possible (binary) labeling of these points, there is a set of parameters *theta* that obtains zero error.

VC-Dimension

The Vapnik-Chervonenkis dimension D of a binary classifier f is the maximum number of data points that can be shattered, irrespective of the actual values of the points.

VC Dimension

Given a m-dimensional feature space:

- Linear Classifier: $m + 1$.
- Multilayer Perceptron with w parameters: $O(w^2)$
- Directed Acyclic Neural Network with $s \geq 2$ internal nodes each of VC Dimension d: $2d \cdot s \log_2 e \cdot s$
- Random Forest: Believed to be ∞.

VC Dimension: Theoretical Bounds

Given train and testing errors, assuming that training points (x_i, y_i) are independent and identically distributed (iid), and model parameters α:

$$\text{TRAINERR}(\alpha) = 0.5 N^{-1} \sum_i |y_i - f(x_i, \alpha)|$$

$$\text{TESTERR}(\alpha) = 0.5 E[|y - f(x, \alpha)|]$$

over the test set

Then the following bound applies with probability $1 - \eta$:

$$\text{TESTERR}(\alpha) \leq \text{TRAINERR}(\alpha) + \sqrt{\frac{d(\log \frac{2N}{d} + 1) - \log \frac{\eta}{4}}{N}}$$

This implies that test error is usually larger than training error. Note that the VC Dimension and this bound do not consider the training data itself in its calculation.

Theory Failures

Recently a paper titled "Understanding Deep Learning Requires Rethinking Generalization" presented at ICLR'17 tells a slightly different story:

- VC Dimension tells you the generalization power of a model and it can bound the test error (indicating generalization).
- Large neural networks were widely believed to not generalize due to their high VC Dimension, indicating large changes of overfitting.
- But we know that deep neural networks do generalize, despite of their large VC Dimension (think of AlexNet, Inception, VGG, etc).
- In this paper the authors trained a given neural network with **random labels**, in which case the model did not generalize at all (obviously). VC Dimension bounds cannot predict this.

Theory Failures

- This effectively meant that with random labels, training error was zero, but test error was very large.
- The effect remains when regularization was applied.
- This paper basically destroyed all theory about generalization, as it cannot explain a very simple example that happens in practice.
- Explanations are quite simple. The information available in the labels and the task complexity have to be considered to explain why a model generalizes and other does not.

How to Prevent Overfitting?

- Reduce model learning capacity or use a different model.
- Use regularization or methods that combat overfitting.
- Train with more data, including capturing more real data, using data augmentation, or synthetic data.
- Use early stopping. Stop training if **validation** loss is not improving.
- Auxiliary tasks and Multi-Task learning can also improve generalization. We will see this later in the course.

No Free Lunch Theorem

Another important theoretical result is the NFL theorem, which basically indicates:

In the average, no learning algorithm is superior to another.

Meaning that if you average performance of all learning algorithms over all random training and testing sets, then average performance is pretty much the same.

Bias-Variance Trade-off

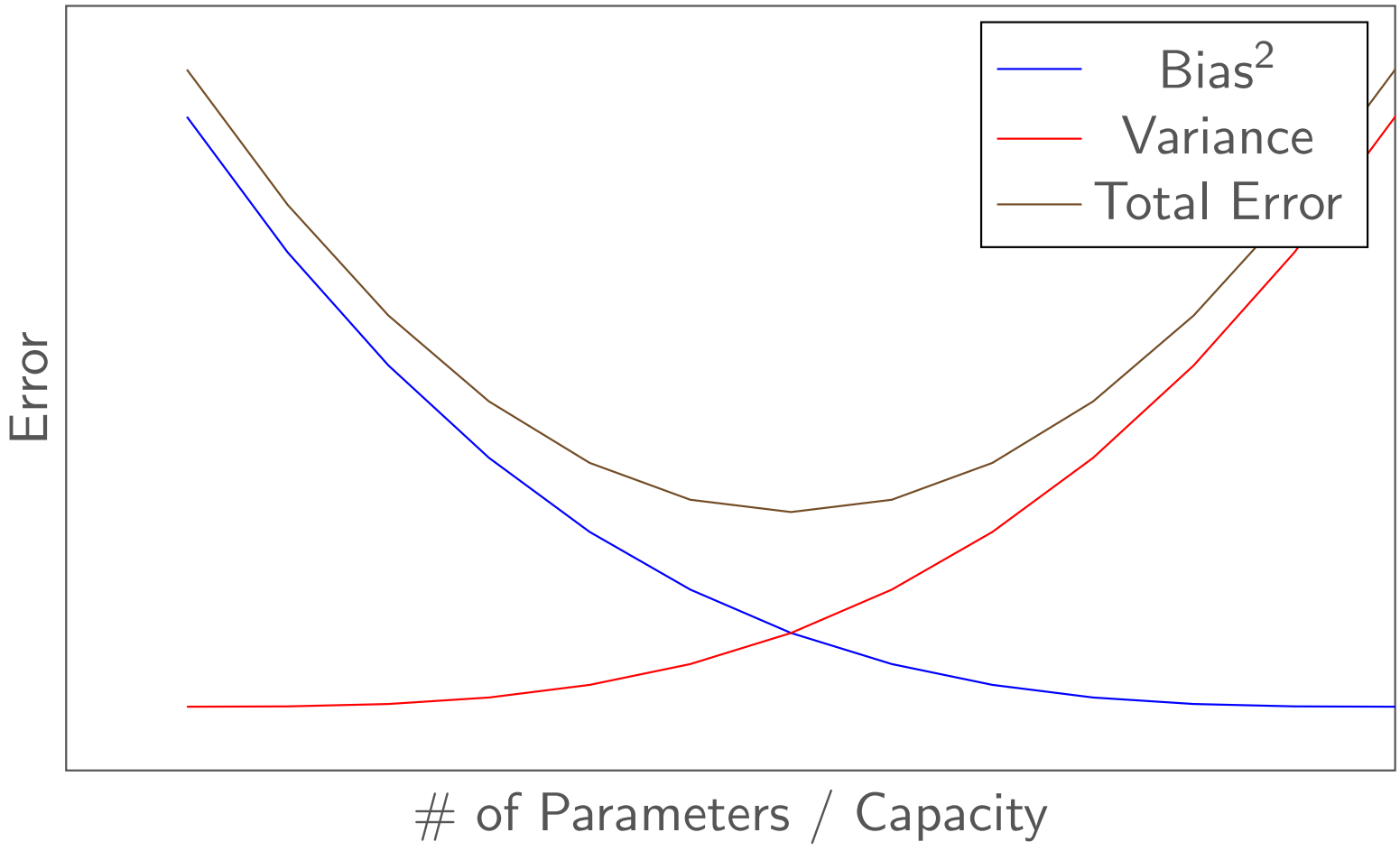
The mean squared error can always be decomposed as follows:

$$
\begin{aligned}
E[(y - \hat{f}(x))^2] = & \overbrace{E[\hat{f}(x) - f(x)]^2}^{\text{Bias}} + \underbrace{E[\hat{f}(x)^2] - E[\hat{f}(x)]^2}_{\text{Variance}} + \sigma^2 \\
= & \text{Bias}[\hat{f}(x)]^2 + \text{Var}[\hat{f}(x)] + \sigma^2
\end{aligned}
$$

Where $\sigma^2 = \text{Var}(y)$ is an irreducible error, usually triggered by noise in the data or labels.
The bias comes from assumptions in the model, like fitting non-linear data with a linear model (**underfitting**). Variance comes from the model fitting noise in the training set (**overfitting**).

Bias-Variance Trade-off

There is an optimal model for each task.



Bias-Variance Trade-off

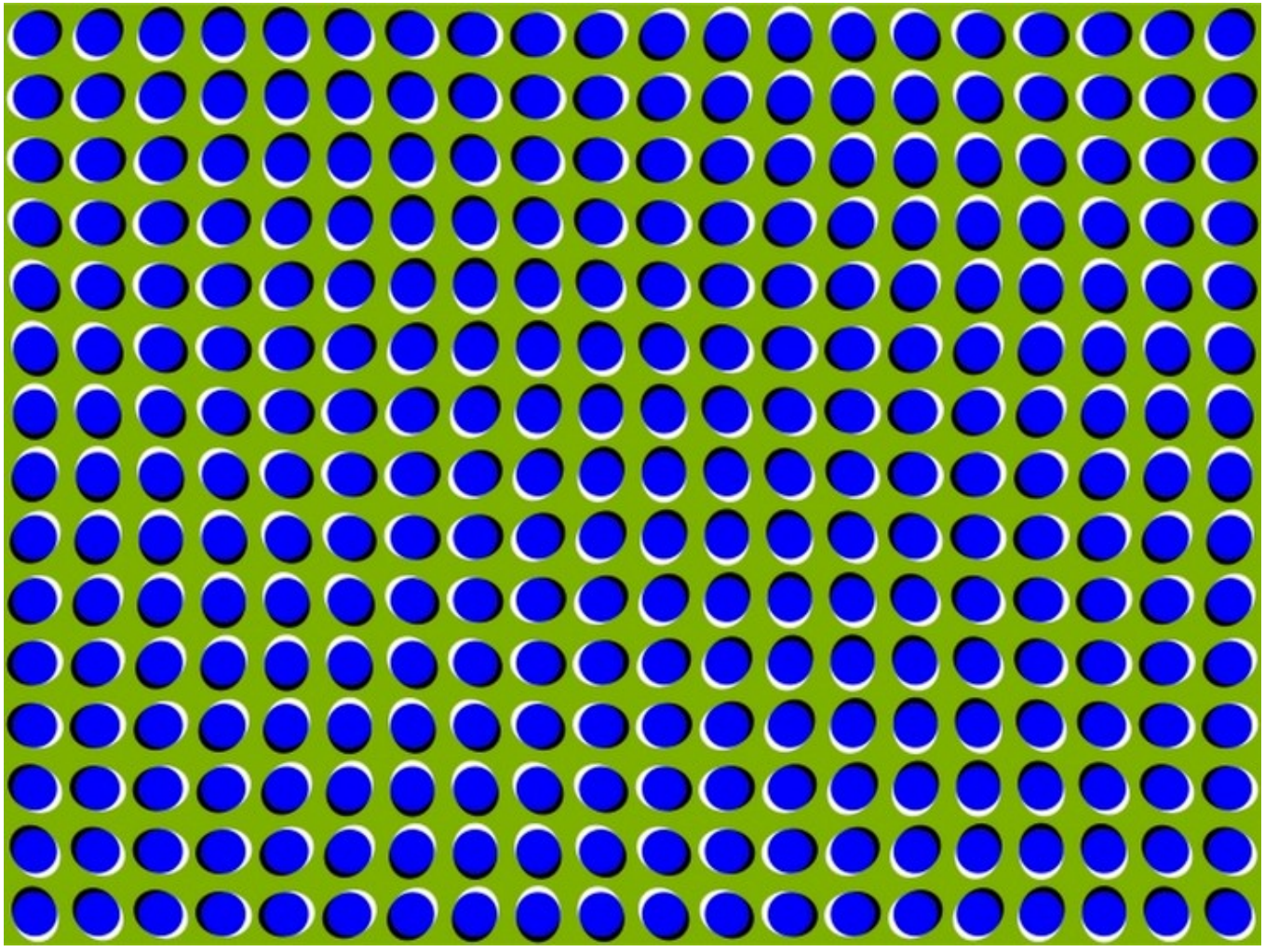
The trade-off is as follows:

- A model with small variance usually has a large bias (its underfitting).
- A model with small bias usually has a large variance due to overfitting.
- Too many parameters/capacity leads to overfitting, and not enough parameters/capacity leads to underfitting. It is not really trivial to determine when under/overfitting happens.
- The number of parameters/capacity needs to be carefully adjusted in order to "match" the tasks' complexity and number of training samples.
- This should always be done on a validation set!

Adversarial Examples

- One big issue related to generalization is that any ML model is vulnerable to adversarial examples.
- These are testing samples that are specifically crafted to fool the classifier.
- One has always have in consideration that samples that are very far from the training set distribution are usually misclassified.
- The problem with adversarial examples is related with uncertainty estimation, as the confidence scores usually produced for these examples are very high, and in practice it is desirable that the confidence scores reflect the certainty of the model.
- An adversarial example is usually closely looking to a real one, but it could also be just noise.

(Human) Adversarial Examples



(Machine Learning) Adversarial Examples

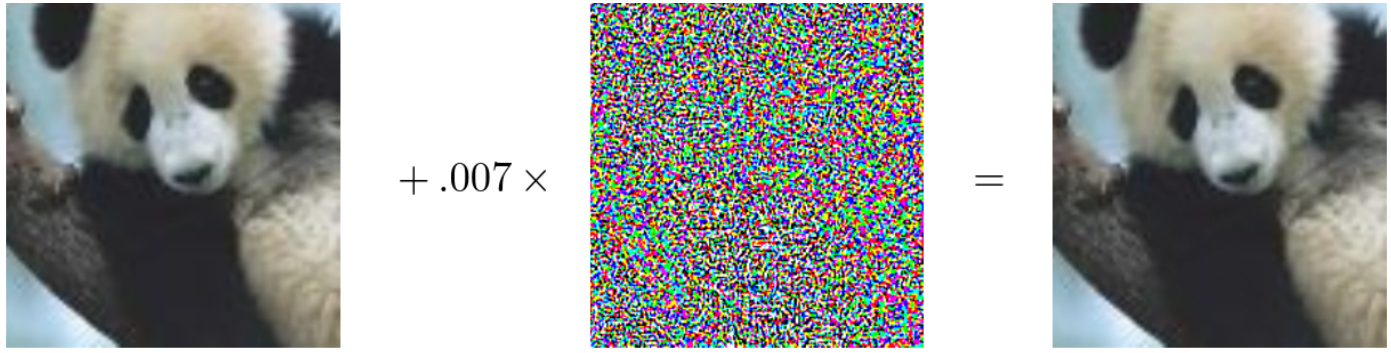


Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

Figure extracted from "Explaining and Harnessing Adversarial Examples" by Goodfellow et al. 2015

Adversarial Examples

Adversarial examples can be generated with:

$$x_{adv} = x + \epsilon \, \text{sign}(\nabla_x L(x, y))$$

Meaning that moving along the sign of the gradient of the loss with respect to the input can reliably generate adversarial examples. ϵ is a hyper-parameter that can be tuned. Some important properties of adversarial examples:

- All ML models are susceptible to them, and it is dataset-independent.
- Adversarial examples generated by one model also work in other models.
- The theory of why this happens is not 100% clear. Some researchers believe that it is because our models are still **too linear**.

Universal Approximators

The Universal Approximation Theorem [1] states that MLPs with one hidden layer can learn any function to arbitrary precision. But it does not say:

- What network configuration can achieve it.
- How it can be trained.
- How much data is required.

[1] Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions"

Regularization

Regularization

It is a way to "guide" or introduce additional information to the learning process in order to reduce overfitting and improve generalization.

- Learning a function mapping from a set of data points is not trivial due to the number of degrees of freedom (N, D, P).
- Given N samples of D-dimensional features, if $N < D$ then the problem is called "ill-posed" or "ill-conditioned", as not all possible feature values are defined.
- Usually you need $N >> D$ in order to learn a concept properly, but it can still be done if additional information is provided.
- Also if a model has P learnable parameters, then $P > N$, else not all model parameters have unique values. This is called an underspecified problem.

Regularization

- Even if $P < N$ or $N < D$ a concept can still be learned, if additional information to the learning process is provided in order to obtain unique solutions.
- But while unique solutions are desirable in theory, in practice they are not necessarily "better". Deep Neural Networks have multiple solutions and theory has proven that these are quite similar to each other.
- The relationship between all these variables is not completely understood, specially the relation between P and N, D, as the amount of information in each training sample might offset having a small sample size (for example, large images).

Data Augmentation

The simplest way to regularize a model is to obtain more data!

- Capturing and labeling data is usually expensive, but "free" data can be obtained through augmentation.
- New data can be generated from existing data, but transformations must be label invariant, or at least the labels should be transformable (for example, bounding boxes).
- Image rotations, flips (up down, left right), brightness changes.
- PCA, Scale, Rotation, Translation, Channel swaps, etc.
- Choice completely depends on application, domain and available data.
- One very important detail is to **only apply data augmentation to the training data**, not to validation or testing data.

Early Stopping

- Usually iterative methods are used to train machine learning models. Then the number of iterations (or epochs) becomes a tunable hyper-parameter.
- As we saw in bias-variance trade-off, there is usually a point where the model starts to overfit.
- Early stopping is just the process of stopping training right before the validation loss starts to increase. Most ML frameworks implement this mechanism.
- An important points is that to do this one needs a three-way split (train/val/test), as choosing the stopping point counts as hyper-parameter tuning, and only evaluation on a test will be unbiased.

L^p Regularization (Tikhonov's)

It is the most common regularization method. It consists of introducing a penalty that considers the norm of the model parameters (weights):

$$L^*(f(x), y) = L(f(x), y) + \lambda \sum_i |w_i|^p$$

The sum is over all trainable model parameters. The regularization coefficient λ is a hyperparameter that defines the strength of regularization. p is the norm dimensionality, typically set to 1 (called LASSO) or 2 (weight decay). This method heavily restricts the capacity of the model, achieving better generalization.

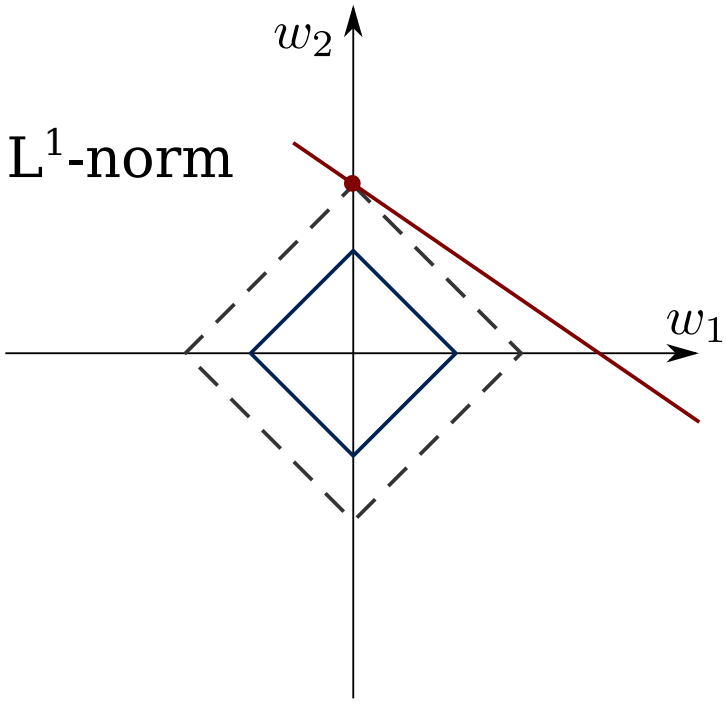
L^1 Regularization (LASSO)

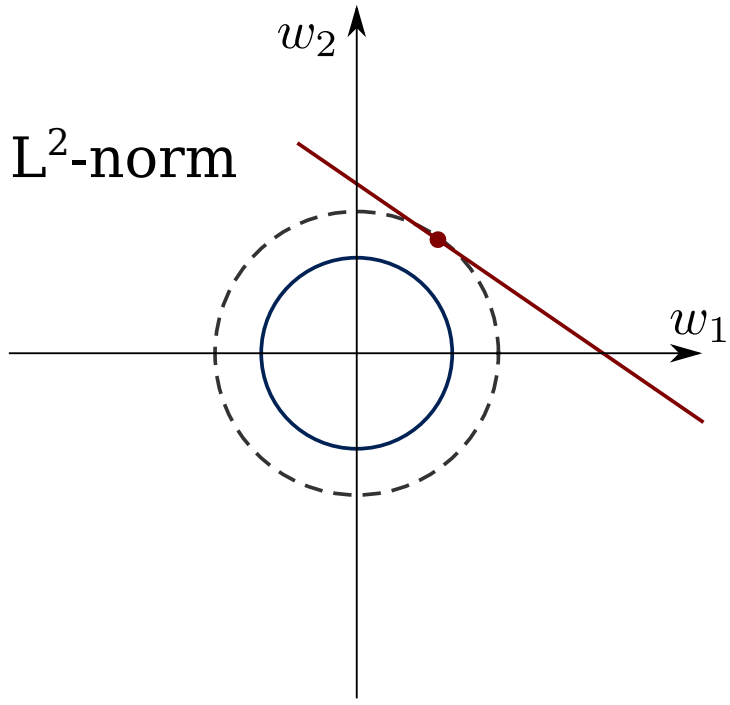
- It has the tendency to make some of the model parameters to be set to zero (a sparse solution).
- A model parameter will be non-zero only if it positively contributes to reduce the original loss of the model, in a way to "counter-act" the penalty from the regularization term.
- If used with a linear model, inputs associated to non-zero weights can be interpreted as relevant features, which constitutes a feature selection method.

L^2 Regularization (Weight Decay)

- Tends to make the model's parameters decay to small values, unless they are supported by the data and reduce the loss of the model.
- Early stopping is related to Weight Decay, as weights usually grow with time (iterations). Doing Early Stopping controls how much the weights can grow, effectively decaying them.
- Both regularization methods have a Bayesian point of view, which corresponds to setting a prior distribution on the model parameters.

Geometric Interpretation

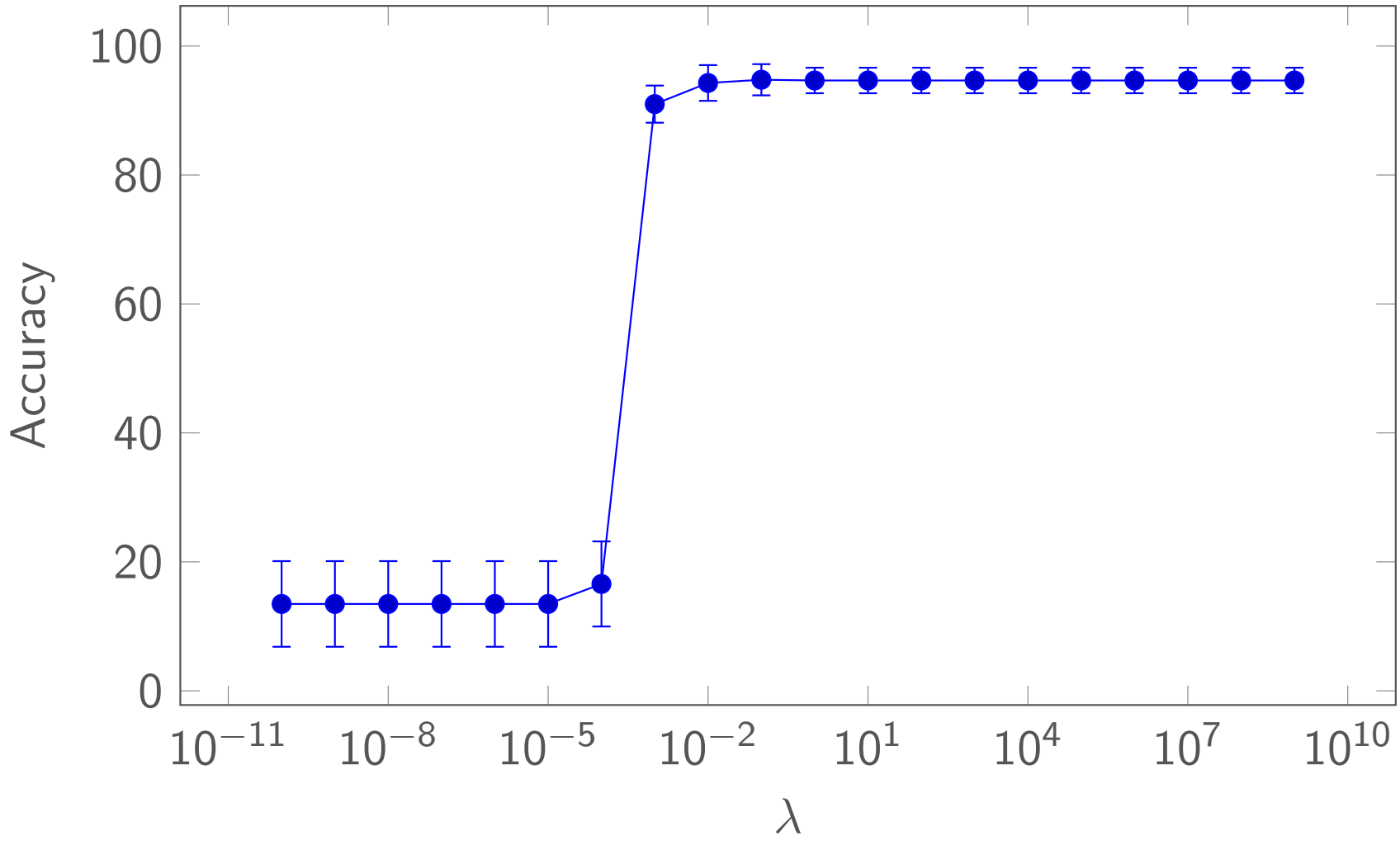




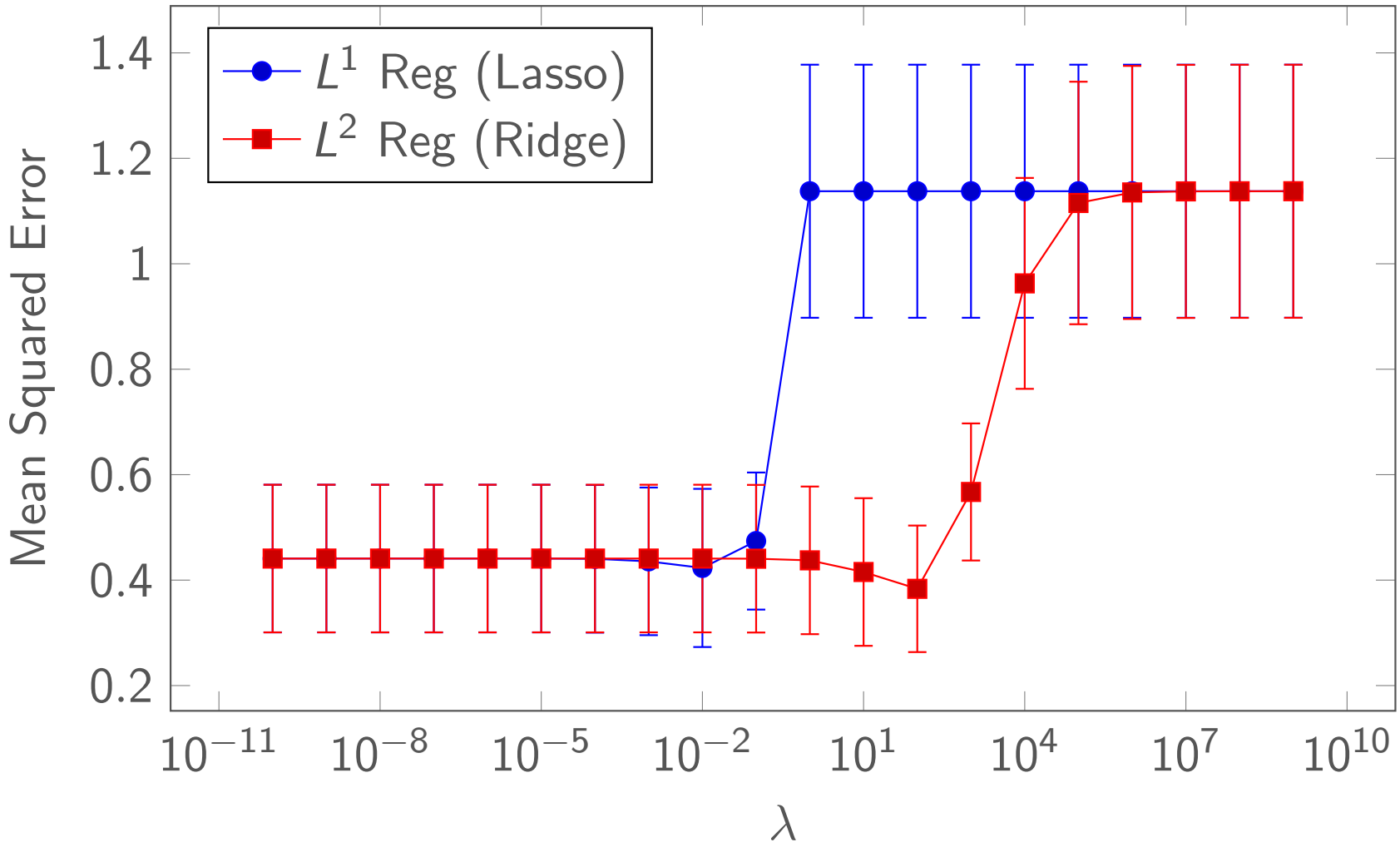
Tuning the regularization coefficient λ

- λ can be used to control the effective model capacity.
- If λ is too large, the model will underfit, and if λ is too small, then regularization is not efficient and the model could overfit.
- One way to tune this parameter is through grid search. Create a grid of values in **logarithmic** scale, like:
  $\lambda \in [10^{-l}, \cdots, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, \cdots, 10^{l}]$.
- Then train a model for each value of λ and evaluate on a validation set. Use the model with the best validation performance, and make a final evaluation on the test set.
- Cross-validation is preferable in order to learn a more robust estimate of λ.

Tuning λ example: 8×8 Digits SVM Classification



Tuning λ example: Boston Dataset Linear Regression

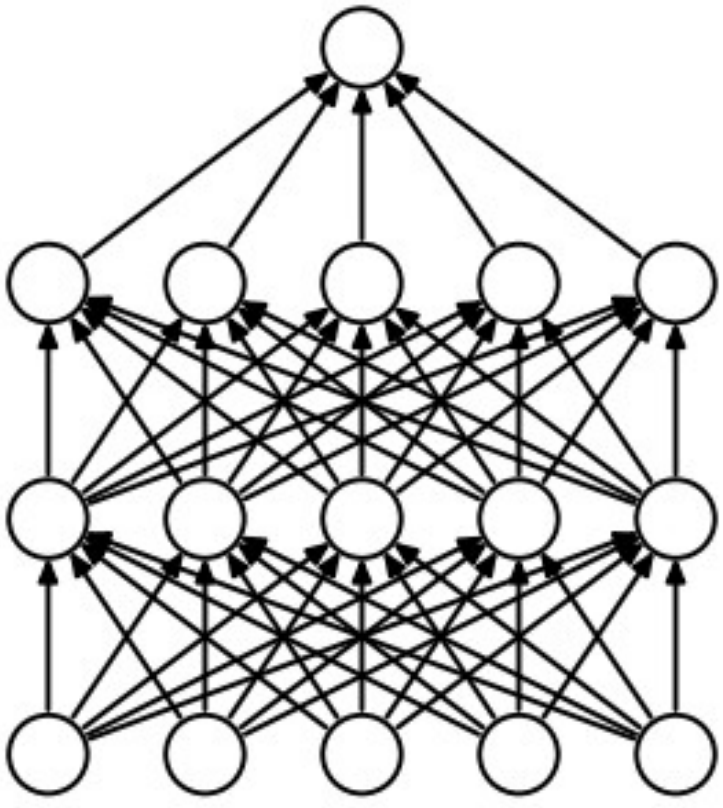


Dropout

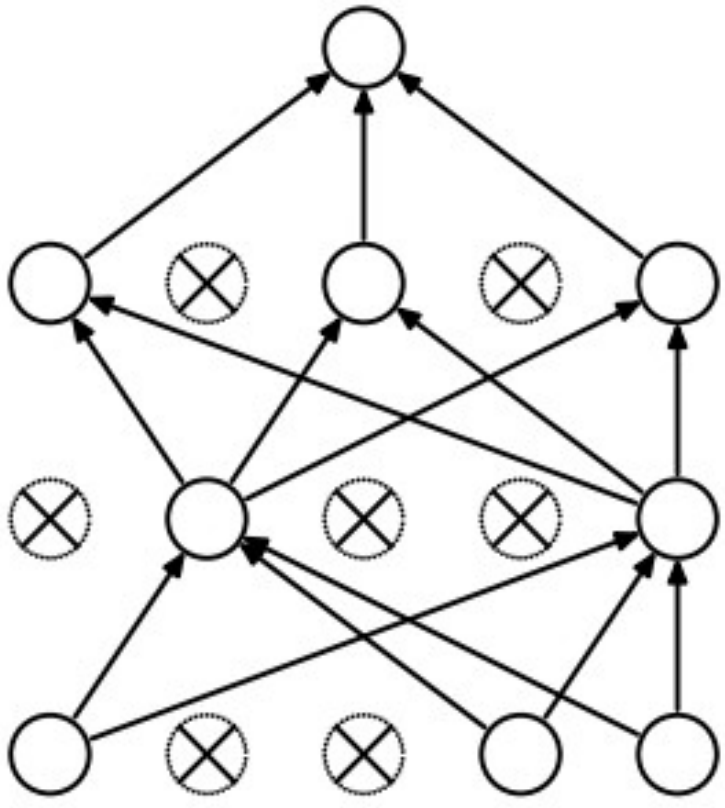
- Hinton et al. noticed that Neural Networks overfit due to "co-adaption" between neurons. This means that neuron activations correlate, which is undesirable.
- One way to break co-adaption is to introduce noise into the network during training.
- Each output from a layer is randomly set to zero with probability p. A "mask" sampled from a Binomial distribution is used for this purpose.
- At testing/inference time, activations of each layer are scaled by p as the random dropping during training has the effect of increasing the expected value due to all neurons activating.
- Common values are for p range in $[0.1, 0.5]$ [2].

[2] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." 2014

Dropout

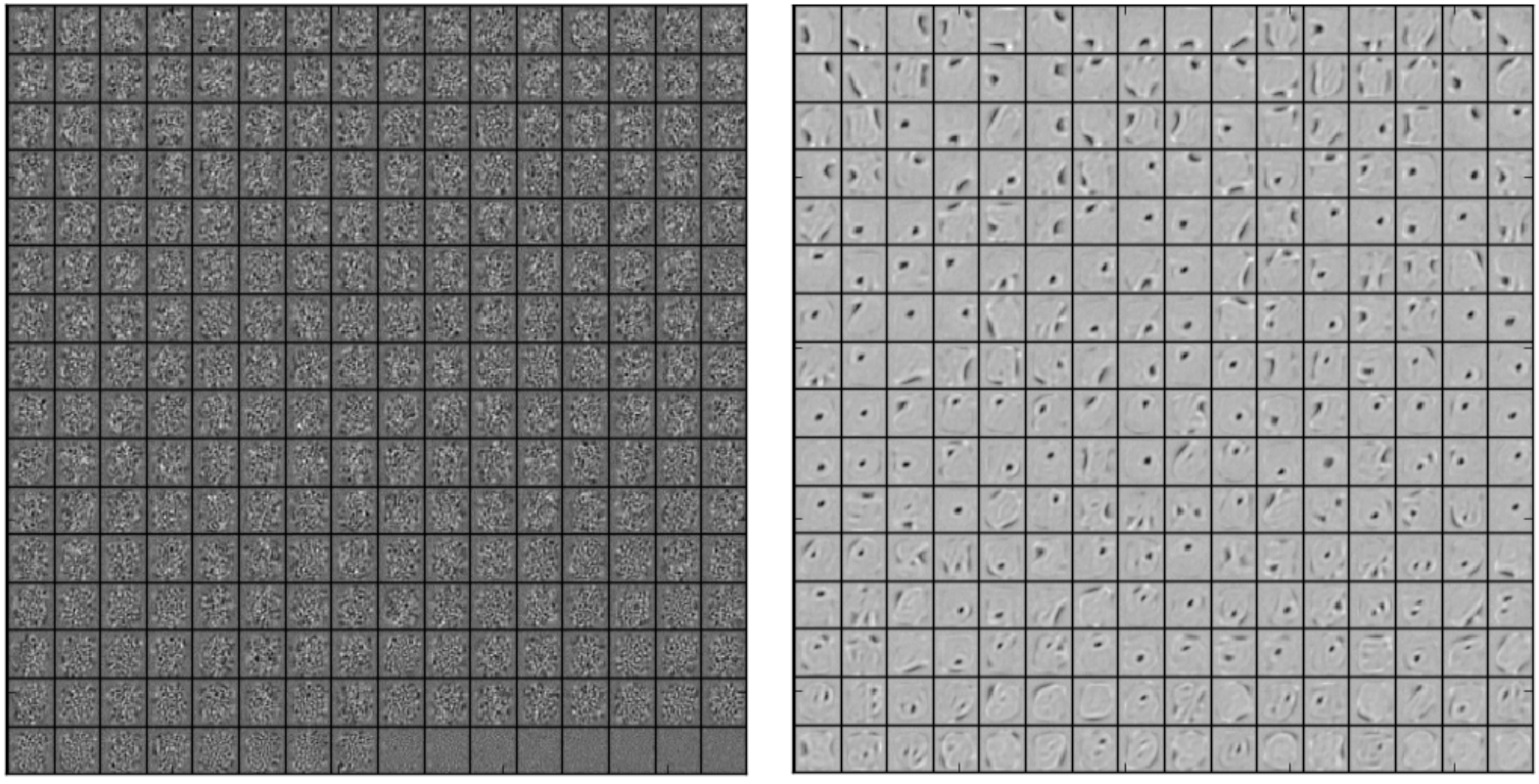


(a) Standard Neural Net



(b) After applying dropout.

Dropout



(a) Without dropout

(b) Dropout with $p = 0.5$.

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

Figure extracted from "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" by Srivastava et al. 2014

Dropout

- Dropout can also be used with non-neural network models as well.
- For example, while training, randomly set some of the input features to zero, and rescale them by p^{-1} (Q: Why?). This should make the model more robust to changes in the features, for example, if some features have missing values.
- The conceptual idea of introducing noise into the learning process is very popular in research. Examples are networks with "Stochastic depth", or corrupt training samples with noise in order to build a more robust model.
- Dropout also has applications in Uncertainty Quantification. We will see this later in Lecture 9.

Batch Normalization

For NN inputs need to be normalized.

- Machine learning models require normalized inputs and outputs, if not, training usually fails.
- Google researchers noticed that by normalizing inputs to layers, training is faster (less iterations) and it also introduces a small degree of regularization.
- Batch Normalization layers compute:

$$x = \frac{x - E[x]}{\sqrt{\text{Var}(x)}}$$

$$y = \gamma x + \beta$$

Batch Normalization

- Mean and Variance are computed at each batch. At test time population statistics are used.
- γ and β are learned scaling factors using back-propagation.
- Accelerates training by at least four times.
- Reduces the "co-variate shift" inside the network. This concept is the variation of parameters in one layer as layers before it are changed (like the butterfly effect).
- Introduces regularization, improving performance. This is because degrees of freedom are removed from the model as parameters are constrained to fulfill the normalization criteria.

Batch Normalization

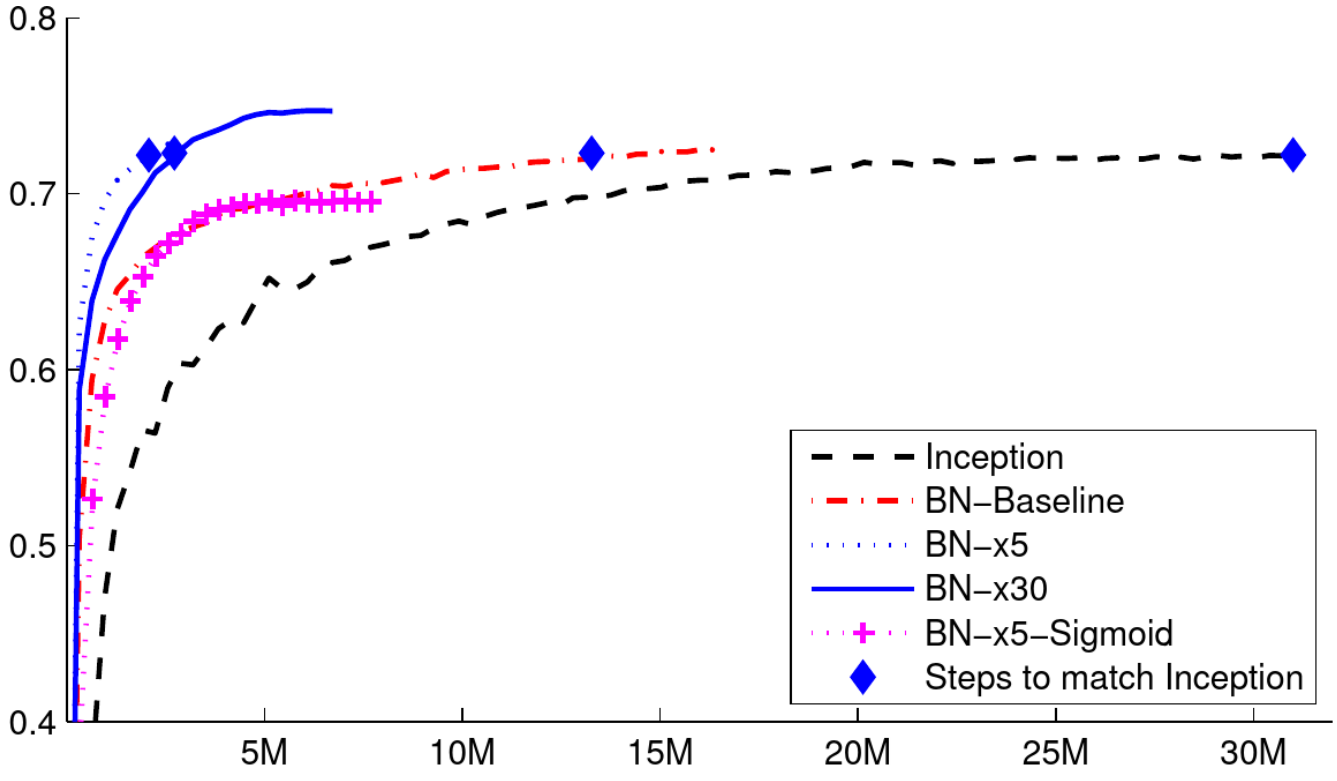


Figure 2. Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.

Figure extracted from "Batch normalization: Accelerating deep network training by reducing internal covariate shift" by Ioffe et al. 2015

Trivia Questions

To decide if the model is overfitting you need to evaluate on a validation set. Not necessary with the zero training error mean.

- Does a zero training error mean that the model is overfitting?
- What happens if you set the regularization coefficient λ to a very large number? Is there a uniquely optimal value for λ.

  There is no knowledge of what is going to happen. There is not a unique optimal value for it.
- How are model hyper-parameters and regularization coefficients tuned?
- Give an intuitive example of why adversarial examples happen.

Readings & References

- "Explaining and Harnessing Adversarial Examples" by Goodfellow et al. 2015.
- "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images" by Nguyen et al. 2015.
- "Understanding Deep Learning Requires Rethinking Generalization" by Zhang et al. 2017.
- "Random Search for Hyper-Parameter Optimization" by Bergstra and Bengio. 2012.
- "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" by Srivastava el at. 2014.

Questions?