Large Scale Data Analysis Using Deep Learning

Practical Methodology

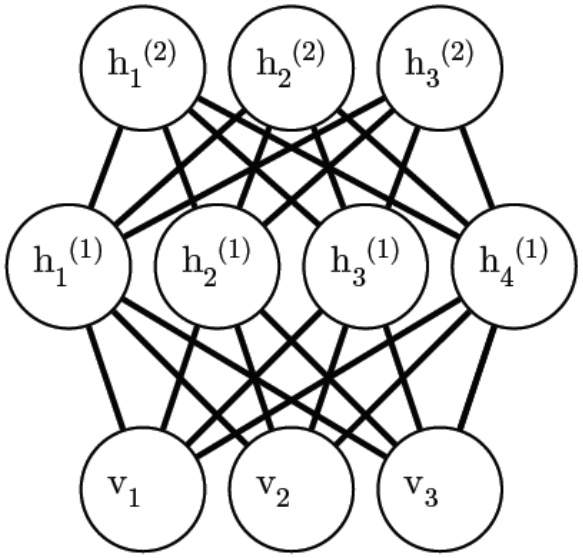
U Kang
Seoul National University

In This Lecture

- Process of developing an ML task
  - Setting goals
  - Building end-to-end system
  - Data-driven refinement
- Hyperparameter search

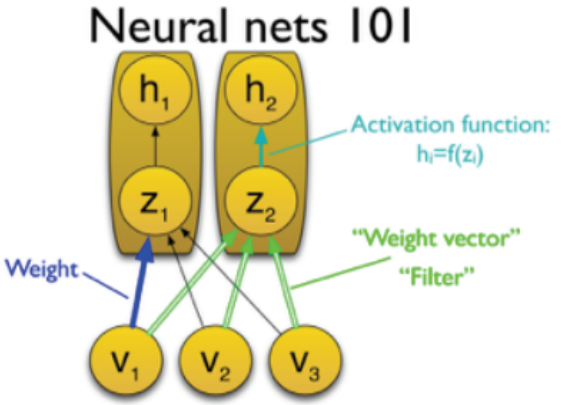
Key to Success in ML?

Arcane knowledge of dozens of obscure algorithms?

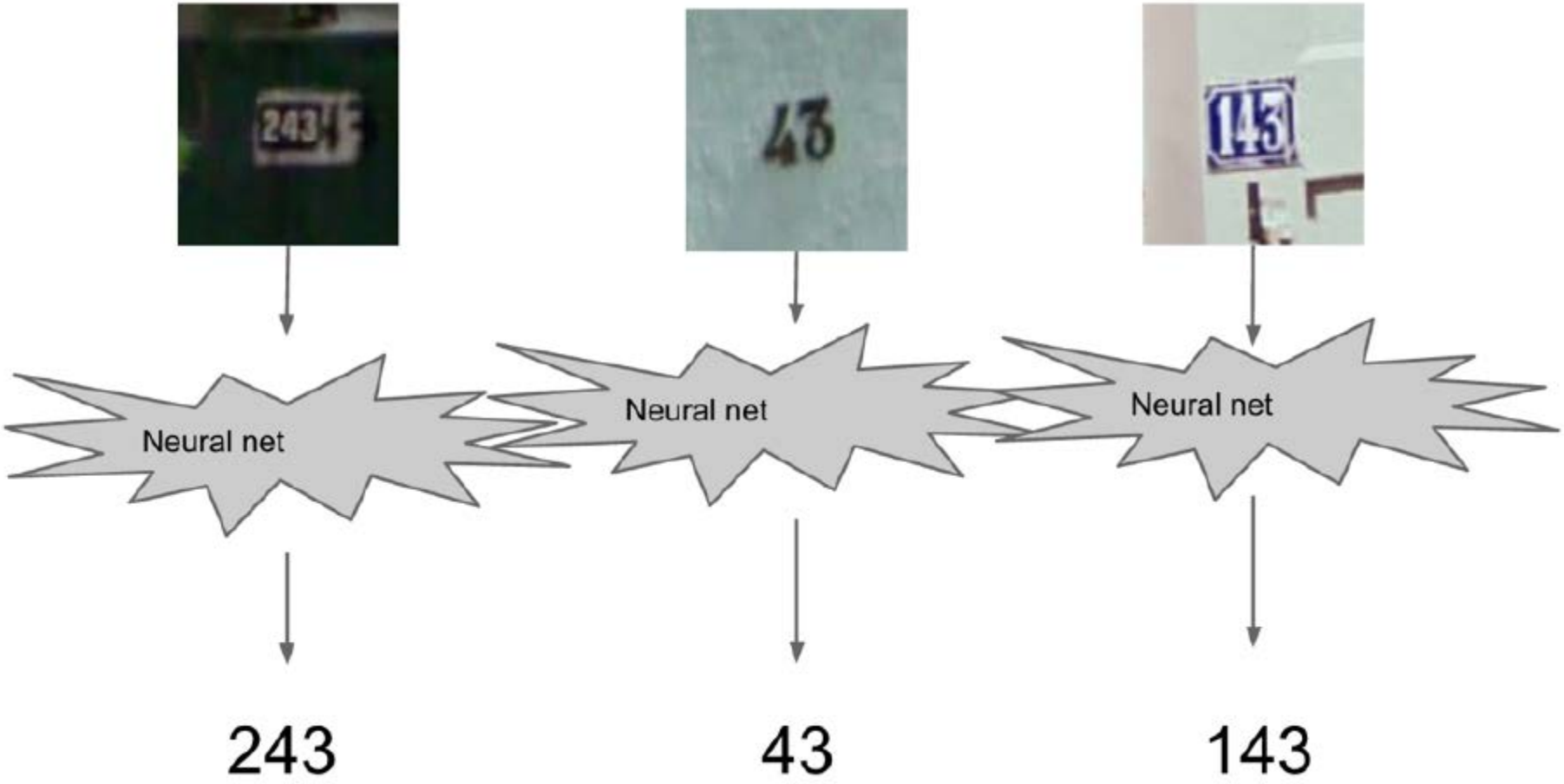


Mountains of data?

Knowing how to apply 3-4 standard techniques?



Example: Street View Address Number Transcription



Three Step Process

- Use needs to define metric-based goals
  - What error metric to use, and your target value for the metric
- Build an end-to-end system
  - Establish a working end-to-end pipeline as soon as possible
- Data-driven refinement
  - Repeatedly make incremental changes such as gathering new data, adjusting hyperparameters, or changing algorithms, based on specific findings from your instrumentation

Identify Needs

- High accuracy or low accuracy?
- Surgery robot: high accuracy
- Celebrity look-a-like app: low accuracy

Choose Metrics

- Accuracy? (% of examples correct)
- Coverage? (% of examples processed)
- Precision? (% of detections that are right)
- Recall? (% of objects detected)
- Amount of error? (For regression problems)

Choose Metrics

- Many applications require more advanced metrics
  - In some cases, it is much more costly to make one kind of a mistake than another
  - E-mail spam detection system
    - It can make two types of mistakes: incorrectly classifying a good email as spam, and incorrectly classifying spam as a good email
    - Between the two mistakes, what is worse?
  - Binary classifier to detect a rare disease
    - Suppose only one in every million people has this disease
    - Detector H: we can easily achieve 99.9999% accuracy by always outputting 'healthy'
    - To solve the problem we can instead measure *precision* and *recall*
    - 'Detector H' would achieve very low recall, with precision undefined (but, no mistake of classifying healthy person as unhealthy)
    - In general, precision and recall are inversely proportional: PR-curve (precision on the y axis and recall on the x axis) helps identify the relation
    - Such classifier outputs $\hat{y} = P(y = 1|x)$. We can choose to classify x as 'unhealthy' if $\hat{y} > \theta$. By varying θ, we can trade precision for recall
    - We can summarize the performance with a single number rather than a curve, by using F-score: $\mathrm{F} = \frac{2pr}{p+r}$

Choose Metrics

- Coverage: fraction of examples for which an ML system is able to produce a response
  - In some applications, it is possible for the ML system to refuse to make a decision
  - This is useful if the ML system can estimate how confident it should be about a decision
    - E.g., a deep neural network with softmax output layer for classification outputs probabilities for each classs
  - One can always obtain 100% accuracy by refusing to process any example, but this reduces the coverage to 0%

End-to-end System

- Get up and running ASAP
- Build the simplest variable system first
- What baseline to start with though?
  - Copy state-of-the-art from related publication

Deep or Not?

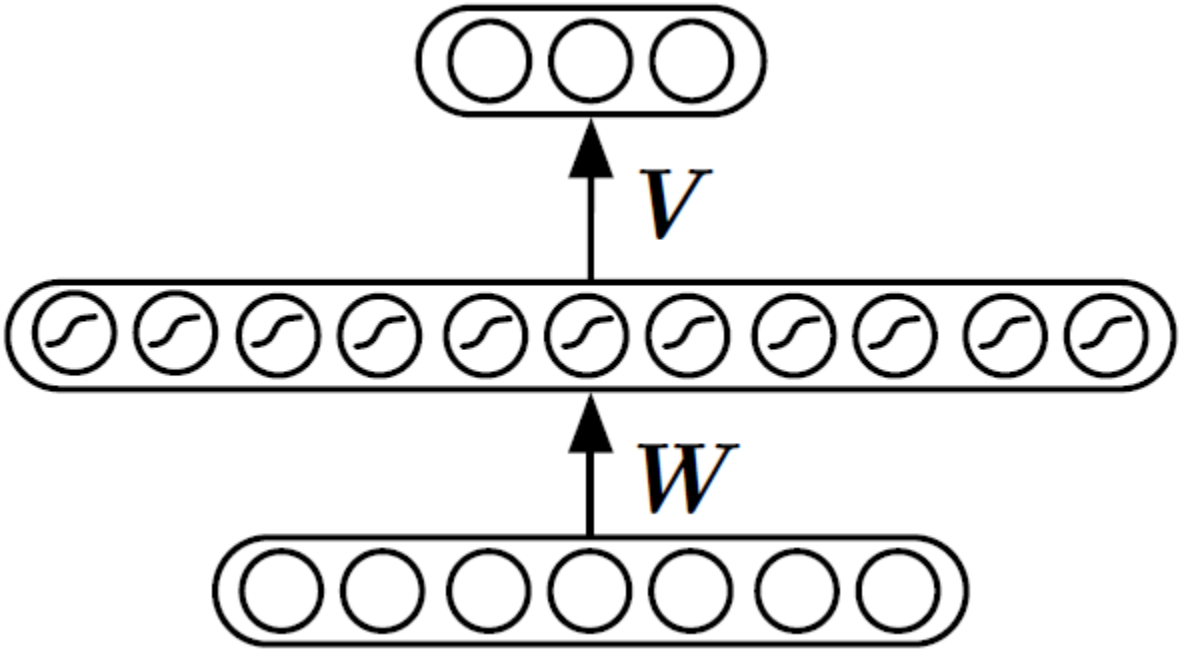
- Lots of noise, little structure -> not deep
- Little noise, complex structure -> deep
- Good shallow baseline:
  - Use what you know
  - Logistic regression, SVM, boosted tree are all good

Default Baseline Models

- Choose the general category of model based on the structure of data
  - Supervised learning with fixed-size vectors as input: use a feedforward network with fully connected layers
  - Input with known topological structure (e.g., image): use CNN
    - Begin with piecewise linear unit (ReLU, Leaky ReLU, etc.)
  - Sequence input or output: use gated recurrent net (LSTM or GRU)
- Optimization algorithm
  - SGD with momentum with a decaying learning rate
    - Decaying linearly until reaching a fixed minimum learning rate, decaying exponentially, or decreasing the learning rate by a factor of 2-10 each time validation error plateaus
  - Adam
  - Batch normalization can be very useful especially for convolutional networks and networks with sigmoidal nonlinearities

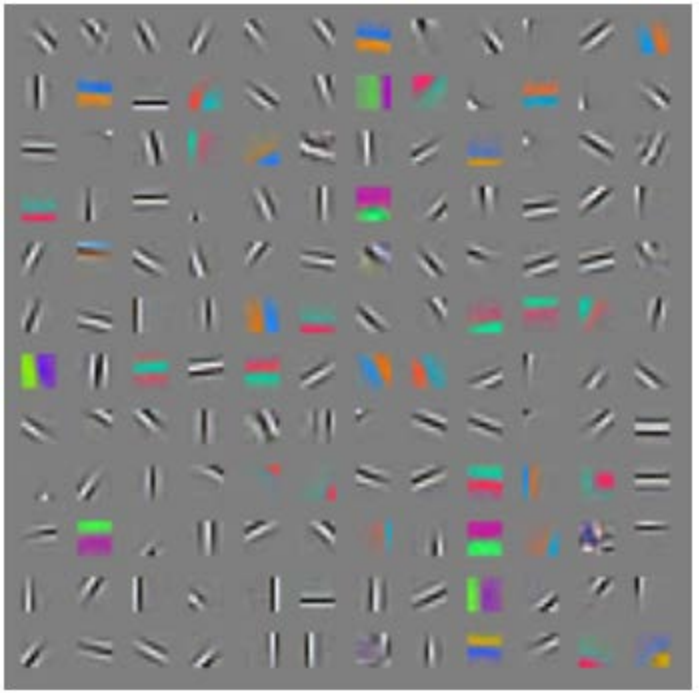
Fully Connected Baseline

- 2-3 hidden layer feedforward neural network
  - AKA “multilayer perceptron”
- Rectified linear units
- Batch normalization
- Adam
- Maybe dropout



Convolutional Network Baseline

- Download a pretrained network
- Or copy-paste an architecture from a related task
  - Or:
    - Deep residual network
    - Batch normalization
    - Adam

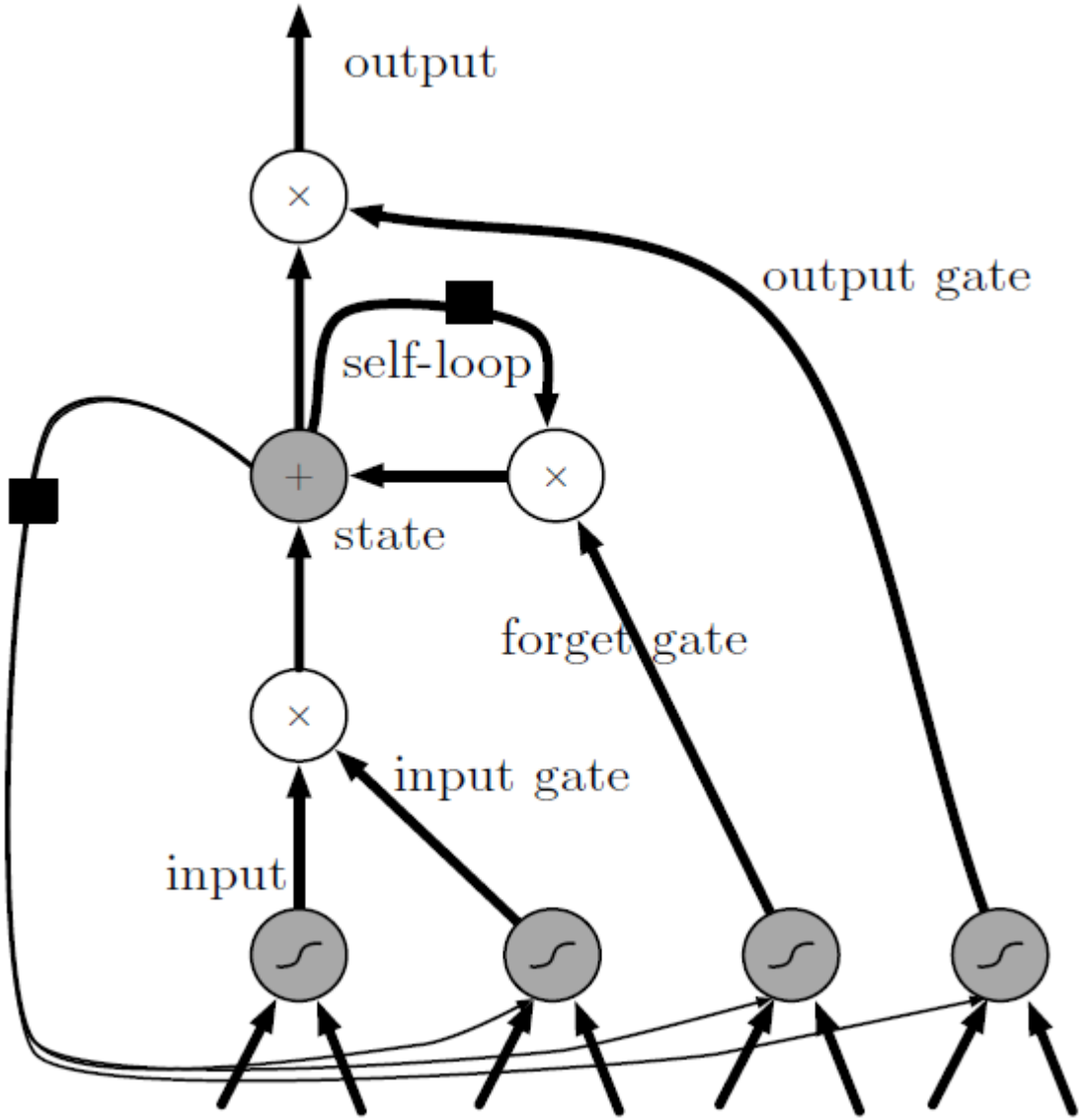


Recurrent Network Baseline

- LSTM
- SGD
- Gradient clipping
- High forget gate bias

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)})$$

Why? lets gradient flow



Default Baseline Models

- Regularization
  - Early stopping should be used almost universally
  - Dropout: an excellent regularizer easy to implement and compatible with many models and training algorithms
  - Batch normalization

- Reusing models and algorithms
  - If your task is similar to another task that has been studied extensively, copy a trained model from that task
  - E.g., it is common to use features from a CNN trained on ImageNet to solve other computer vision tasks

Determining Whether to Gather More Data

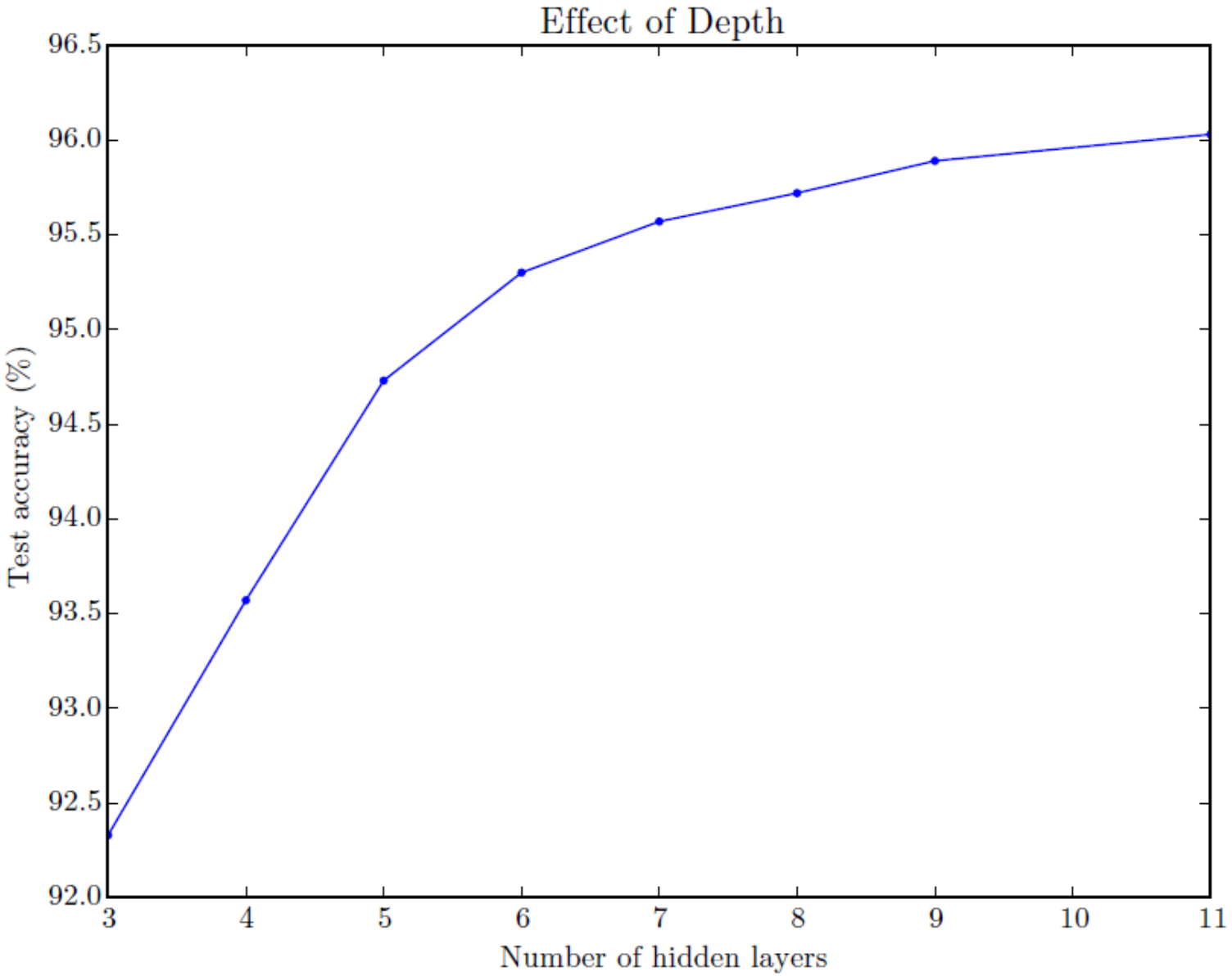
- It is often much better to gather more data than trying many different algorithms
- How to decide whether to gather more data?
  - Determine the performance on the training set
  - If the performance is poor, the learning algorithm is not effectively using the data
    - Inspect data for defects
    - Inspect software for bugs
    - Try to increase the size of the model by adding more layers or hidden units, or improve the learning algorithm, for example by tuning the learning rate hyperparameter
    - If large models and hyperparameter tuning do not work, the quality of the training data may be too low
    - This suggest collecting cleaner data or collecting richer set of features

Checking Data for Defects

- Can a human process it?

26624

Increasing Depth



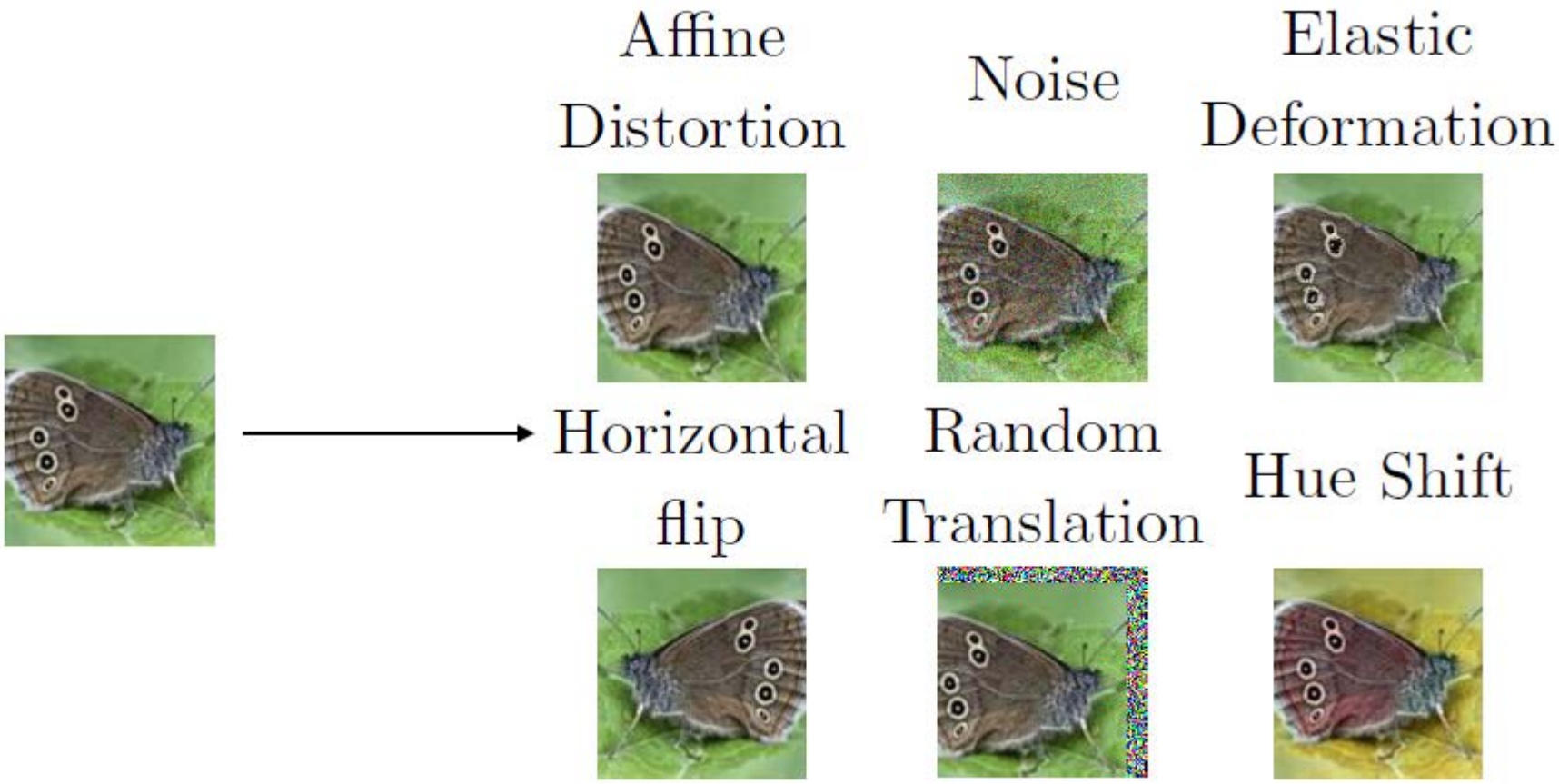
Determining Whether to Gather More Data

- How to decide whether to gather more data?
  - If the training error is acceptable, then measure the test error
  - If the test error is small, we are done
  - If the test error is much worse than training error (overfitting), than gathering more data is one of the most efficient solutions
    - Its feasibility depends on the cost
    - E.g., At large internet companies it may be easy to collect large amount of data; on the other hand, in medical applications it may be very costly to gather more data
  - Other options when the test error is unacceptable
    - Dataset augmentation
    - Reduce the size of the model
    - More regularization
    - Otherwise, improve the learning algorithm itself (an active area of research)
- How much data to gather?
  - Experiment with training set sizes on a log scale (e.g., 2x, 4x, 8x, …)

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data
- Dataset augmentation
  - Create fake data and add it to the training set
  - Has been effective especially for object recognition
- Image augmentation
  - Translation
  - Rotation
  - Scaling
- Injecting noise: a form of data augmentation
  - Neural networks are not very robust to noise; one way to improve the robustness of neural networks is to train them with random noise applied to their inputs

Dataset Augmentation

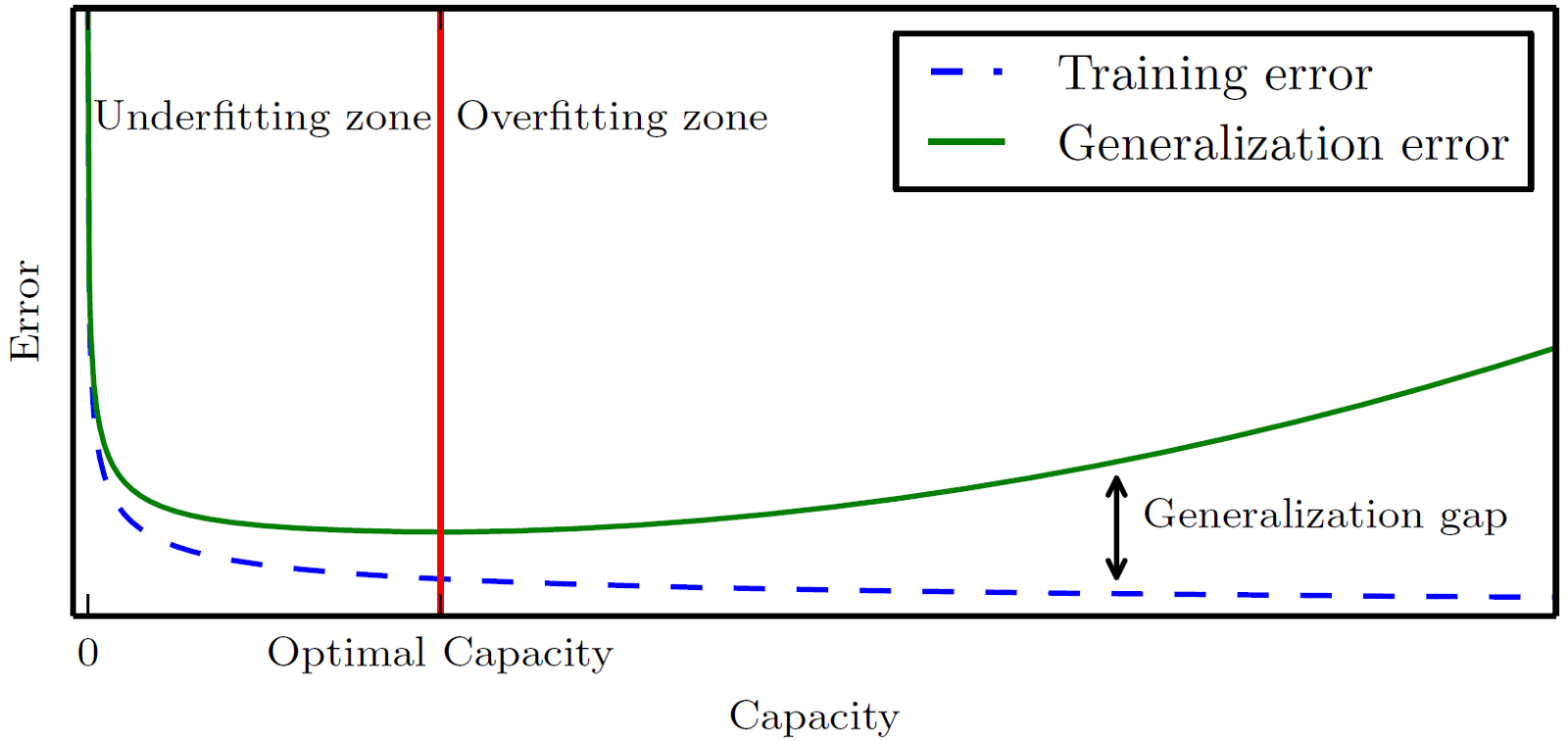


Selecting Hyperparameters

- Hyperparameters affect the performance of deep models significantly
  - Performance: time and memory cost
  - Accuracy
- Two basic approaches
  - Manual choice
  - Automatic choice: active research area
- Goal of hyperparameter tuning
  - Find the lowest generalization error
  - Adjust the effective capacity of the model to match the complexity of the task
- Effective capacity is constrained by three factors
  - Representational capacity of the model
  - The ability of the learning algorithm to successfully minimize the cost function
  - How much the cost function and training procedure regularize the model

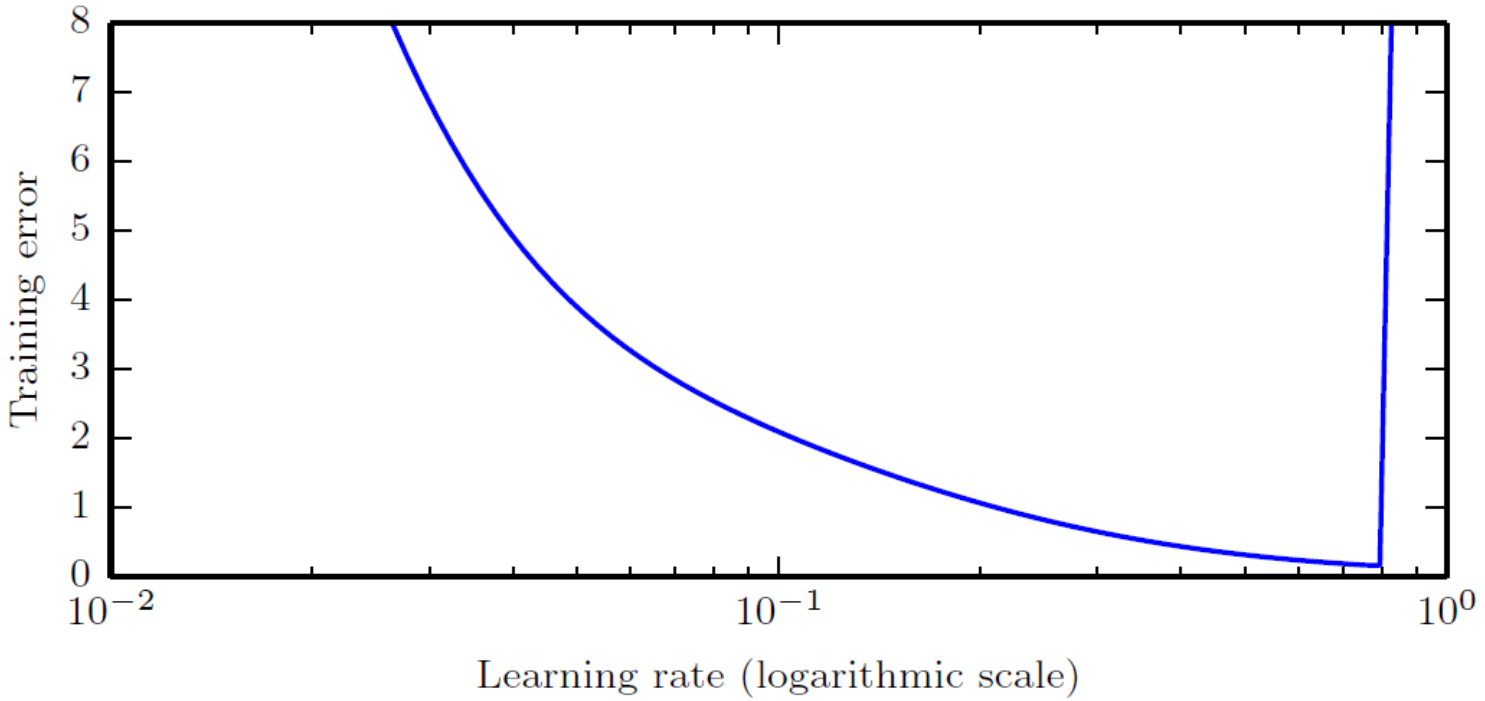
Generalization Error

- Generalization error follows a U-shaped curve
  - Underfitting: low capacity. Generalization error is high since the training error is high
  - Overfitting: too high capacity. Generalization error is high since the generalization gap is high



Learning Rate

- The most important hyperparameter
- Controls the effective capacity of the model
  - The effective capacity is the highest when the learning rate is correct for the optimization problem
  - When the learning rate is too large, gradient descent can inadvertently increase rather than decrease the training error
  - When the learning rate is too small, the error is also large (stuck at a local minimum)



Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units			
Learning rate			
Convolution kernel width			
Implicit zero padding			
Weight decay coefficient			
Dropout rate			

?

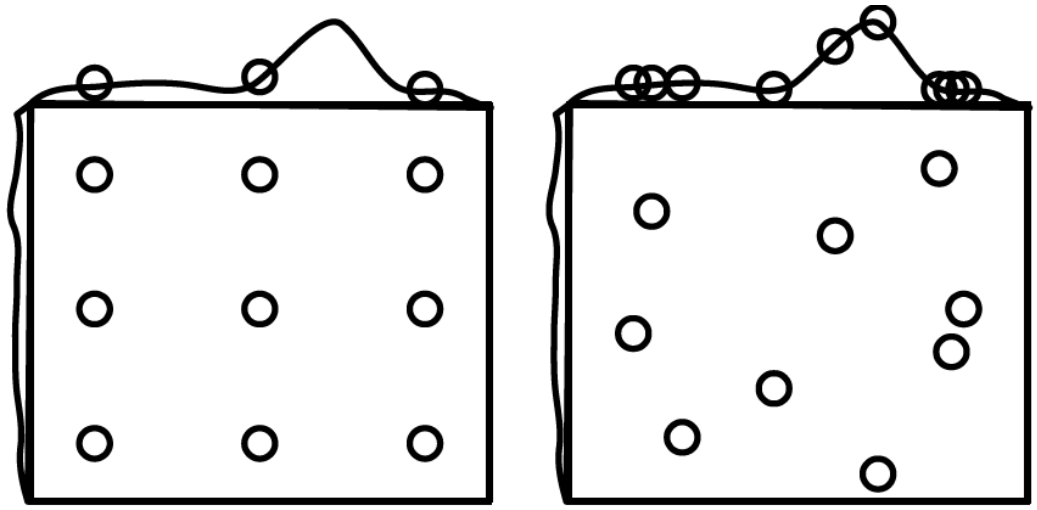
Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.
Learning rate	tuned optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure	
Convolution kernel width	increased	Increasing the kernel width increases the number of parameters in the model	A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.
Implicit zero padding	increased	Adding implicit zeros before convolution keeps the representation size large	Increased time and memory cost of most operations.
Weight decay coefficient	decreased	Decreasing the weight decay coefficient frees the model parameters to become larger	
Dropout rate	decreased	Dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set	

Hyperparameter Search

- Grid search
  - For each hyperparameter, select a small finite set of values to explore
  - Train a model for every joint specification of hyperparameter values
  - Typically, a grid search involves picking values approximately on a logarithmic scale (e.g., number of hidden units taken with the set {50, 100, 2500, 500, 1000, 2000}
  - Grid search usually performs best when it is performed repeatedly, using a refined range
    - Suppose we run a grid search over hyperparameter α using values of {-1, 0, 1}
    - If the best value found is 1, we should shift the grid and run another search with α in {0, 1, 2}
    - If the best value found is 0, then we may wish to refine our estimate by running another search over {-0.1, 0, 0.1}
  - Problem with grid search: computational cost grows exponentially with the number of hyperparameters: $O(n^m)$ for m hyperparameters each taking at most n values

Hyperparameter Search

- Random search
  - Simple to program, more convenient to use, and converges much faster to good values of the hyperparameters
  - Typically, use a uniform distribution on a log-scale for positive real-valued hyperparameters
    - E.g., log_learning_rate ~ uniform(-1, -5)
  - Why random search finds good solutions faster than grid search?
    - Because there are no wasted experimental runs



Grid Random

What you need to know

- Process of developing an ML task
  - Setting goals
    - Accuracy, precision, recall, coverage, error…
    - Depend on each application
  - Building end-to-end system
    - Use appropriate baseline methods
  - Data-driven refinement
    - Decide whether to collect more data based on training error and validation error
- Hyperparameter search
  - Random search and grid search are useful

Questions?