COMPSCI 689
Lecture 14: Neural Network Architectures, Generalization and Hyperparameters

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).

Introduction

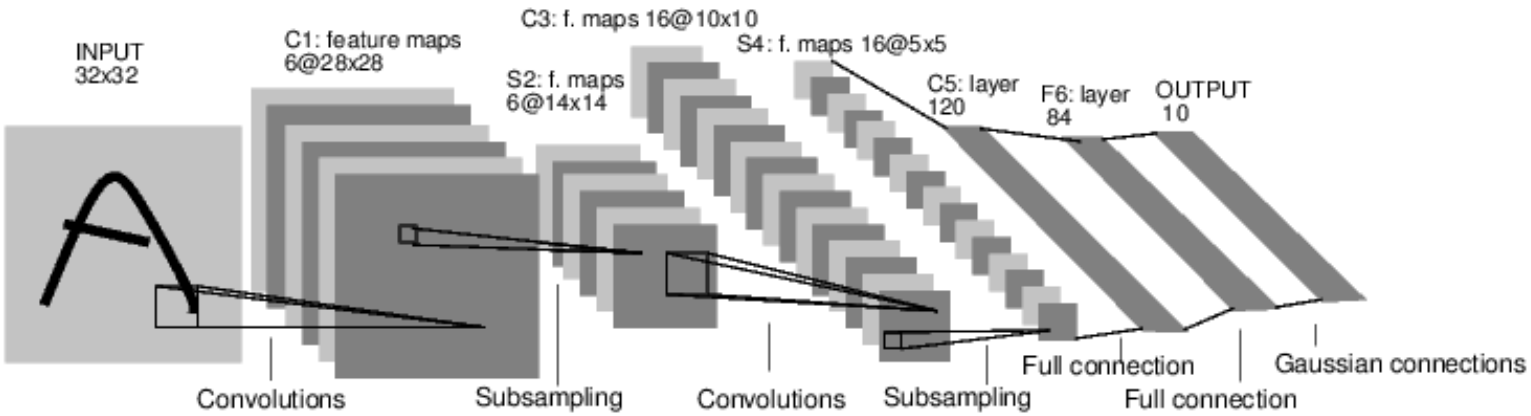
- Basic MLPs have multiple layers of hidden units that are fully connected with distinct weights on each connection.
- Such an architecture does not encode any domain knowledge about the relationship between the inputs and outputs.
- For specialized types of inputs like images or sequences, it's possible to encode domain knowledge about the structure of the inputs or input-output relationships into the architecture of a neural network.
- This can help make learning more efficient by using notions like sparsity and invariance to reduce the number of parameters that must be learned.

Images and Sequences

- The two most common types of structures encountered in the neural networks literature are images and sequences.
- In object detection, an object in an image can occur at any scale, orientation, location, etc., and the information about whether an image contains an object is typically spatially localized.
- In sequence data like text, the underlying grammar and vocabulary induce a set of equivalent representations of the same content, and provide insight into possible structures.

Architectures for Images

Deep learning for images uses a modified deep architecture called a *convolutional network* or convnet or CNN that learns small patches of weights (or filters) and replicates (convolves) them over an image.

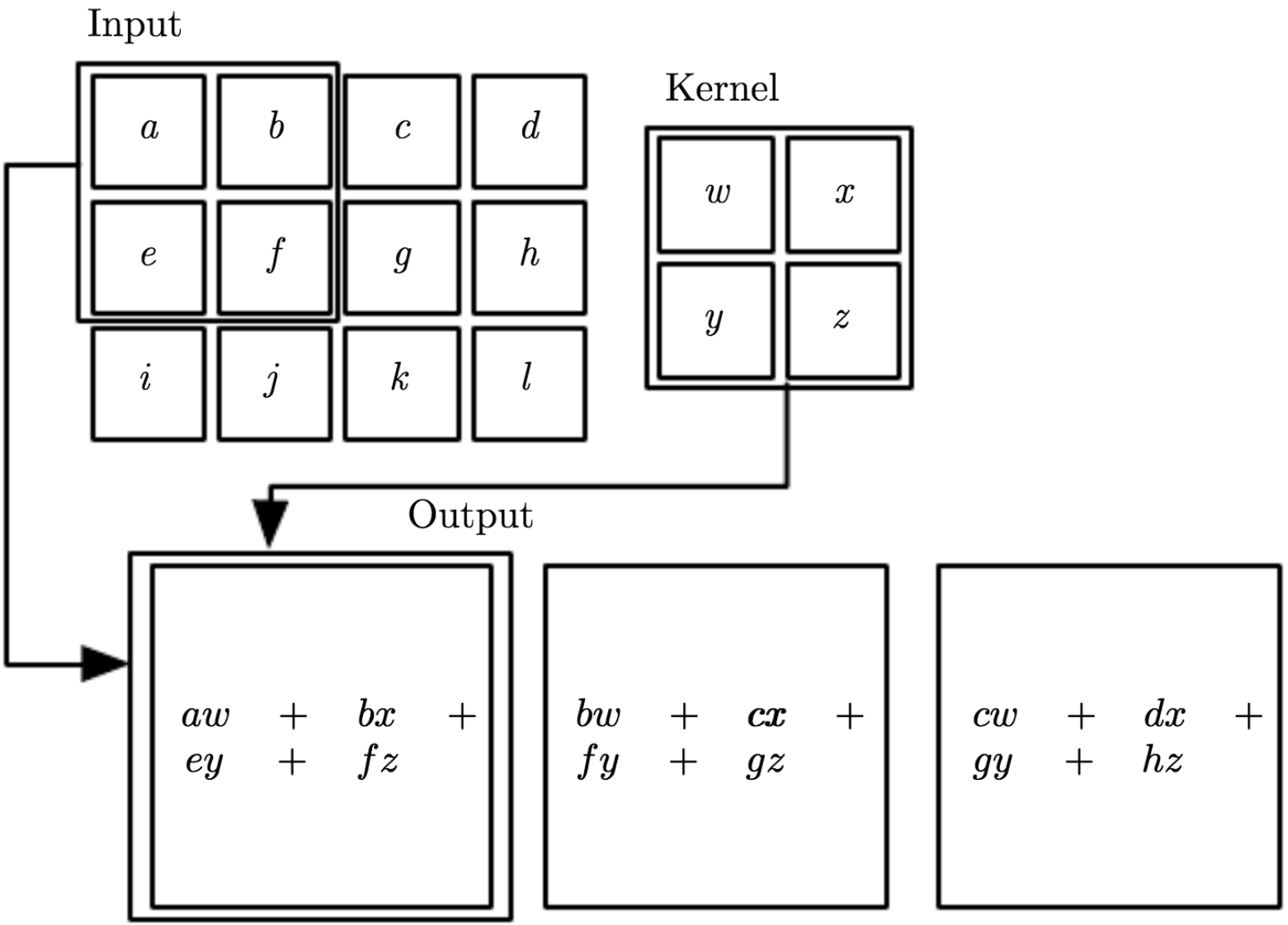


This architecture was developed from the 1980s and was directly inspired by the structure of the visual areas of the brain.

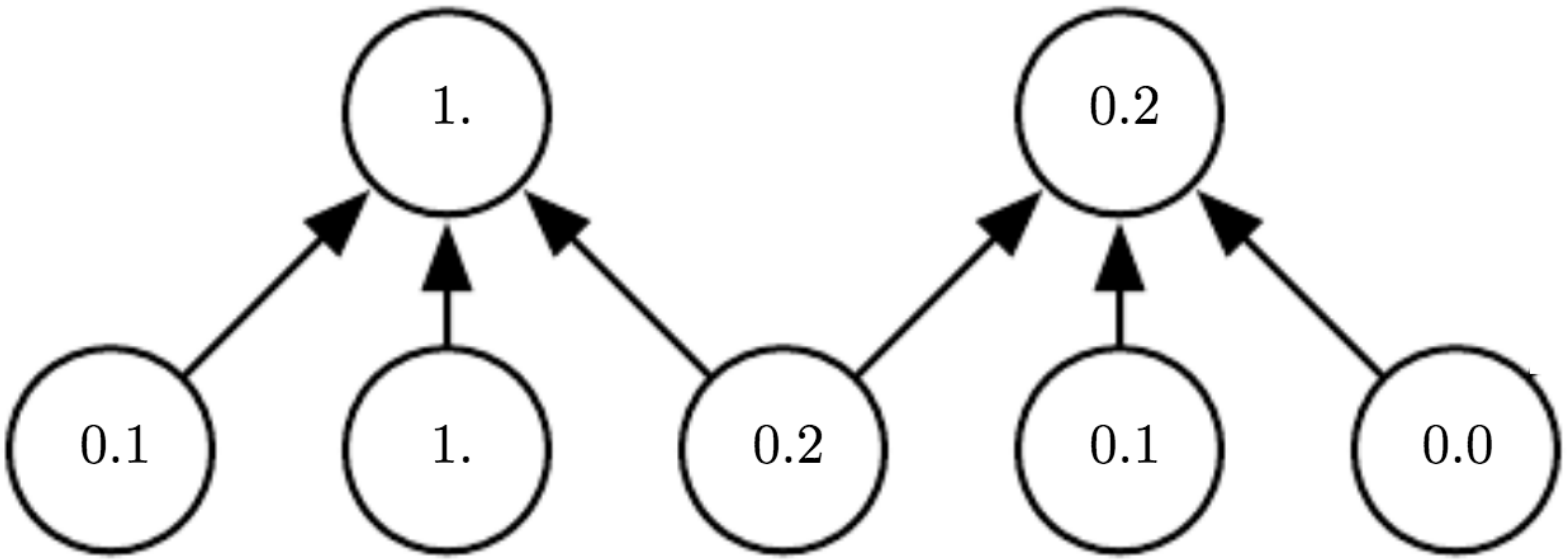
Convolutional Network Invariances

- The convolutional network architecture encodes two primary pieces of domain knowledge: translational invariance and spatial locality.
- Sub-sampling using max pooling (taking the max of the inputs in a spatial window) makes individual hidden units more invariant to local translations in an image.
- Translational invariance is achieved more globally by using the same set of weights in every image patch via convolution.
- The use of compact convolution kernels also results in a massive reduction in the number of weights and creates hidden units that have spatially localized inputs.

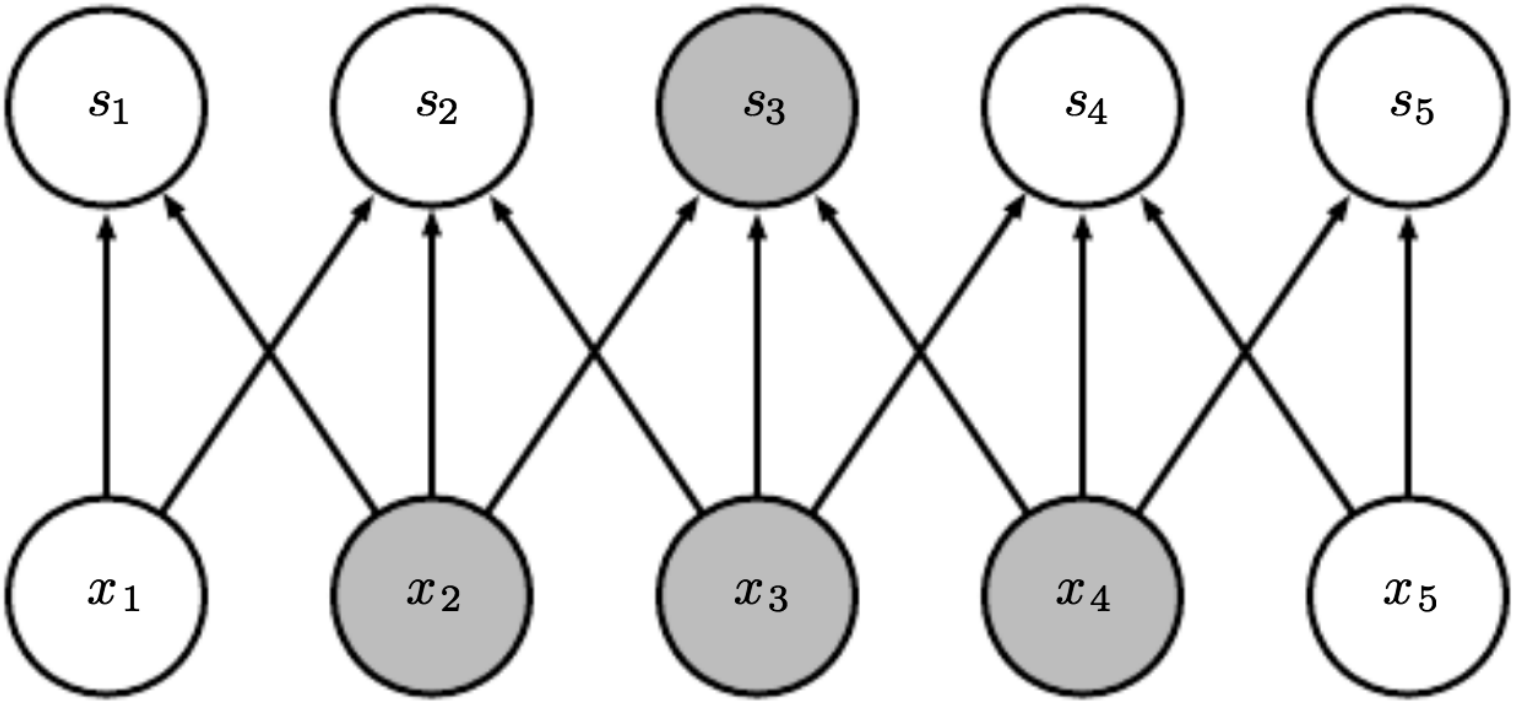
Example: 2D Cross Correlation



Example: 1D Pooling

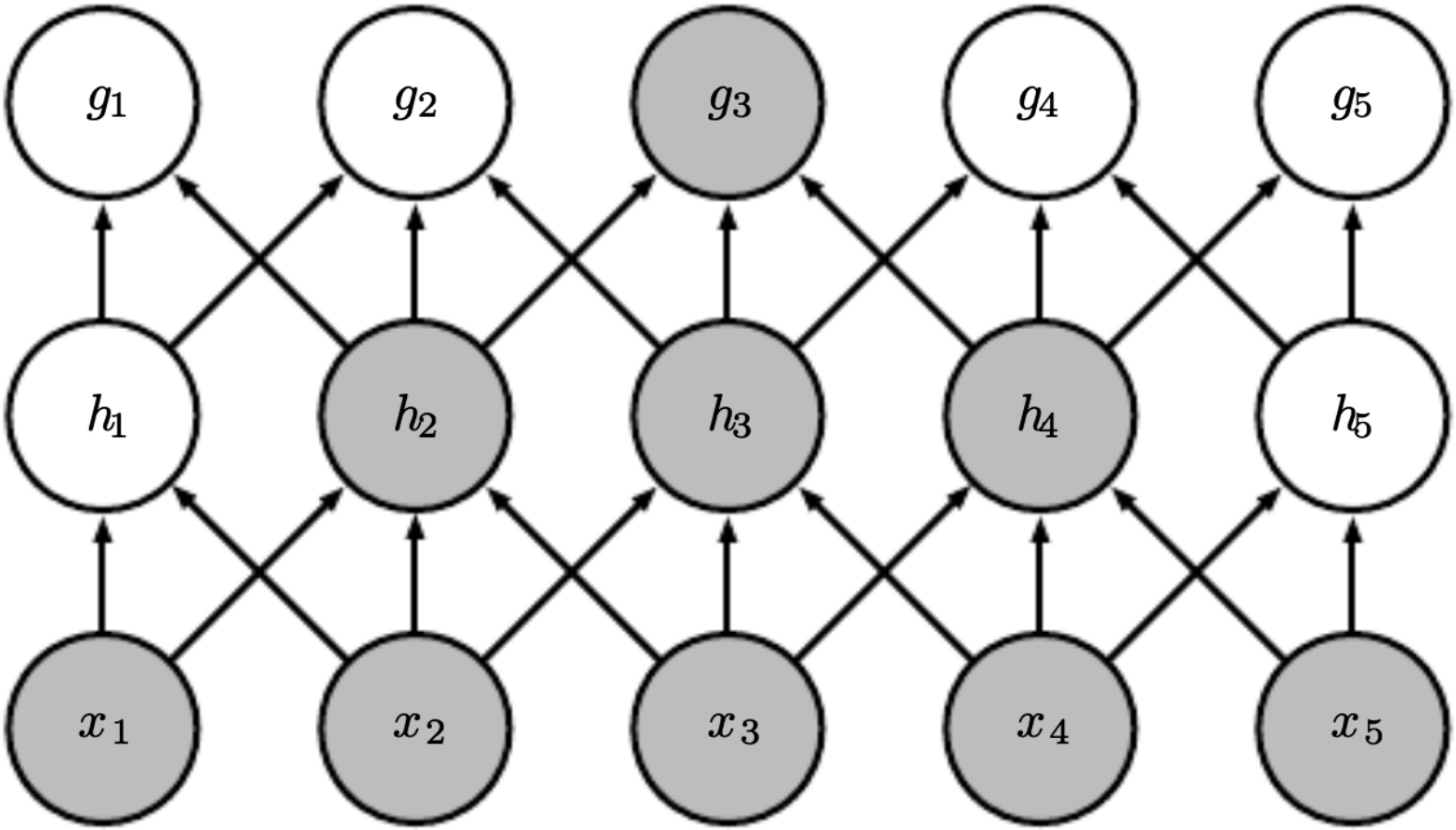


Example: 1D Receptive Field

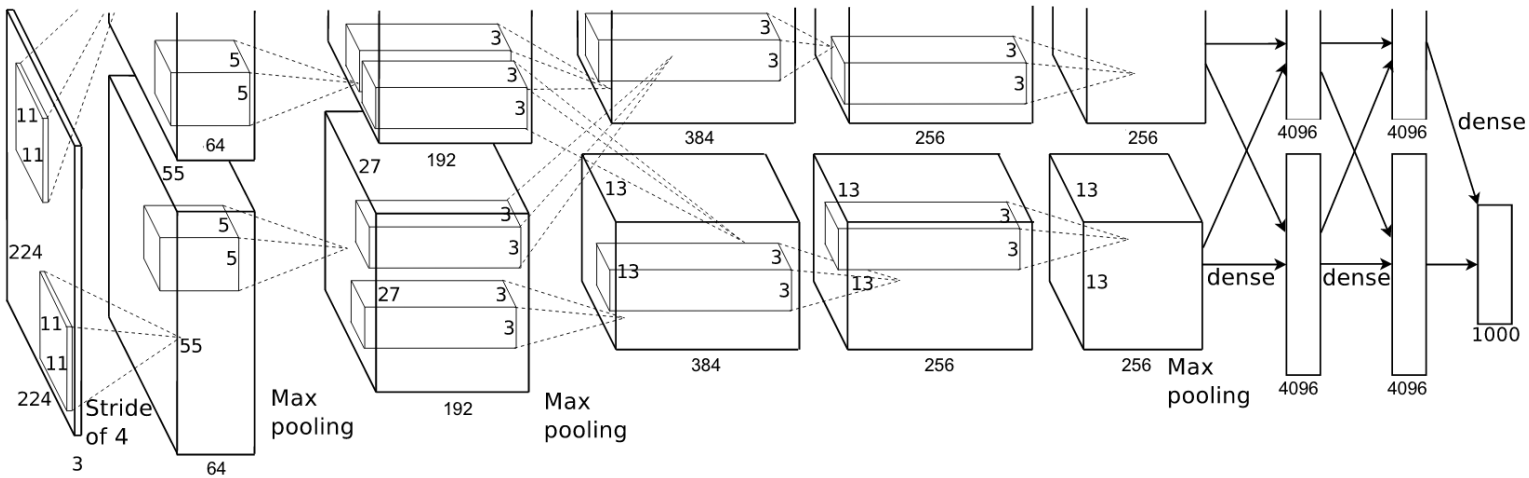


Via analogy to animal visual systems, the set of inputs that feed into a given hidden unit in a given feature map are referred to as that unit's *receptive field*.

Example: 1D Receptive Field



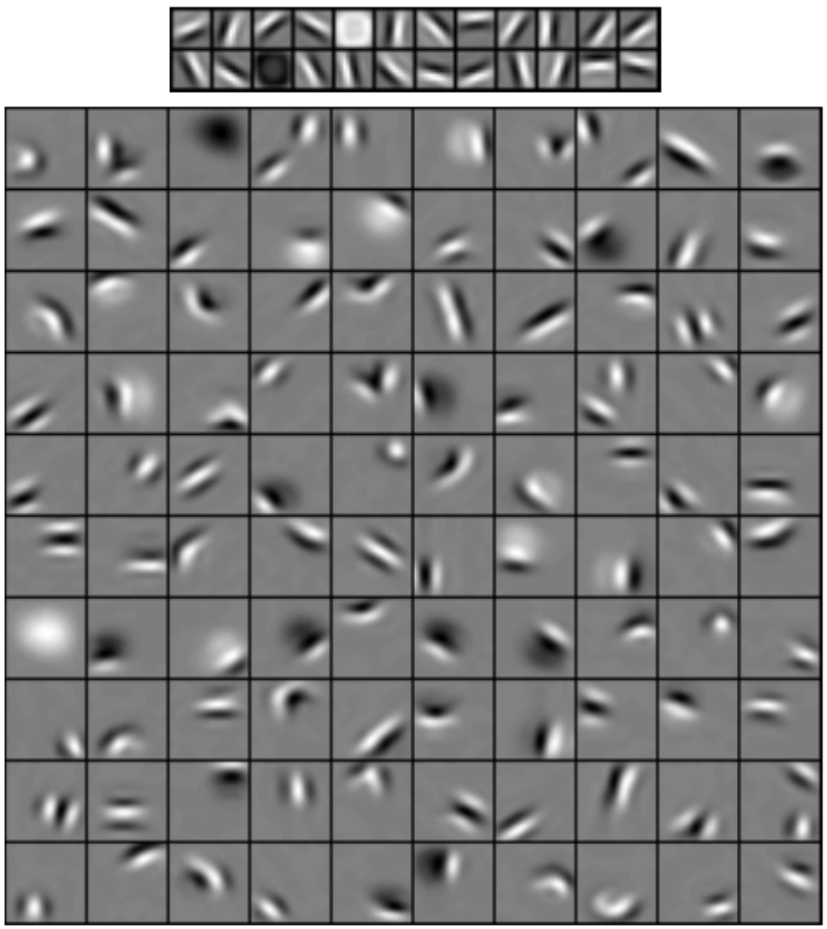
AlexNet (2012)




```
self.features = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(64, 192, kernel_size=5, padding=2),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(192, 384, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(384, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
)
```

```
self.classifier = nn.Sequential(
    nn.Dropout(),
    nn.Linear(256 * 6 * 6, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Linear(4096, num_classes),
)
```

Example: Learned Representations at Multiple Levels

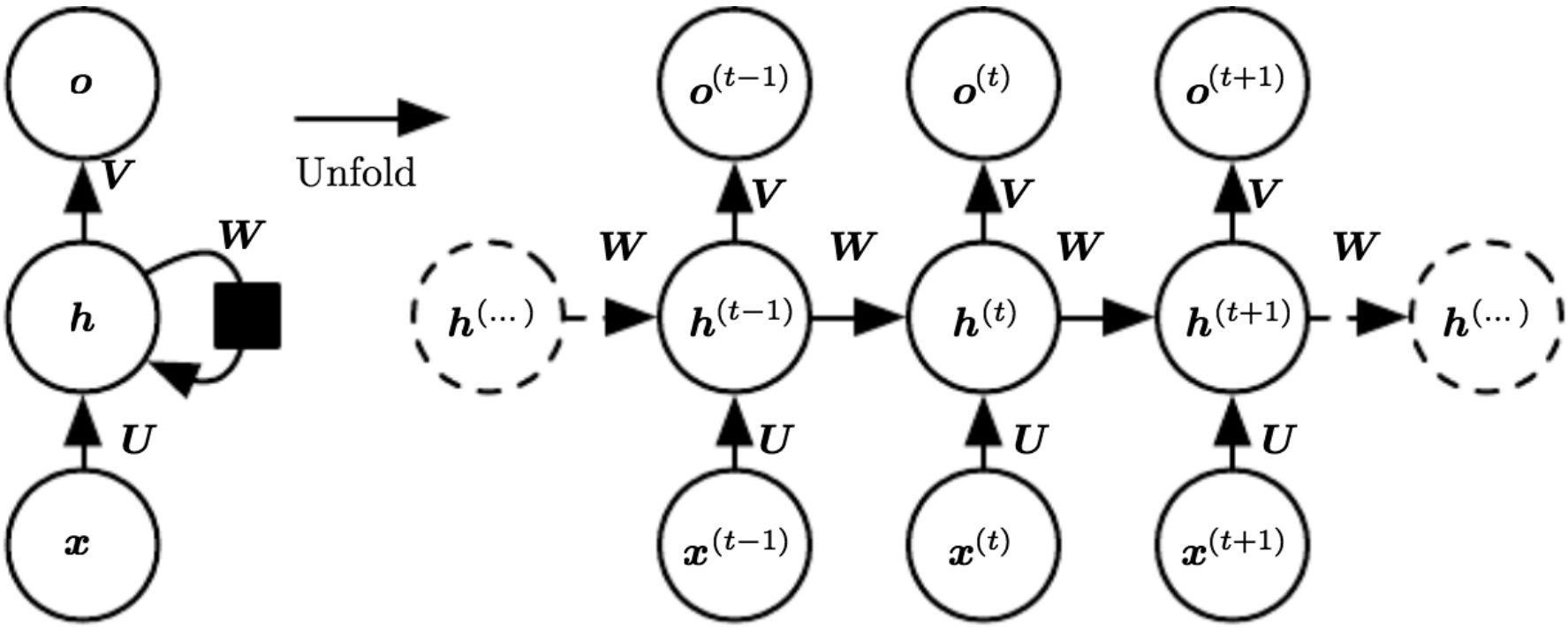


faces, cars, airplanes, motorbikes

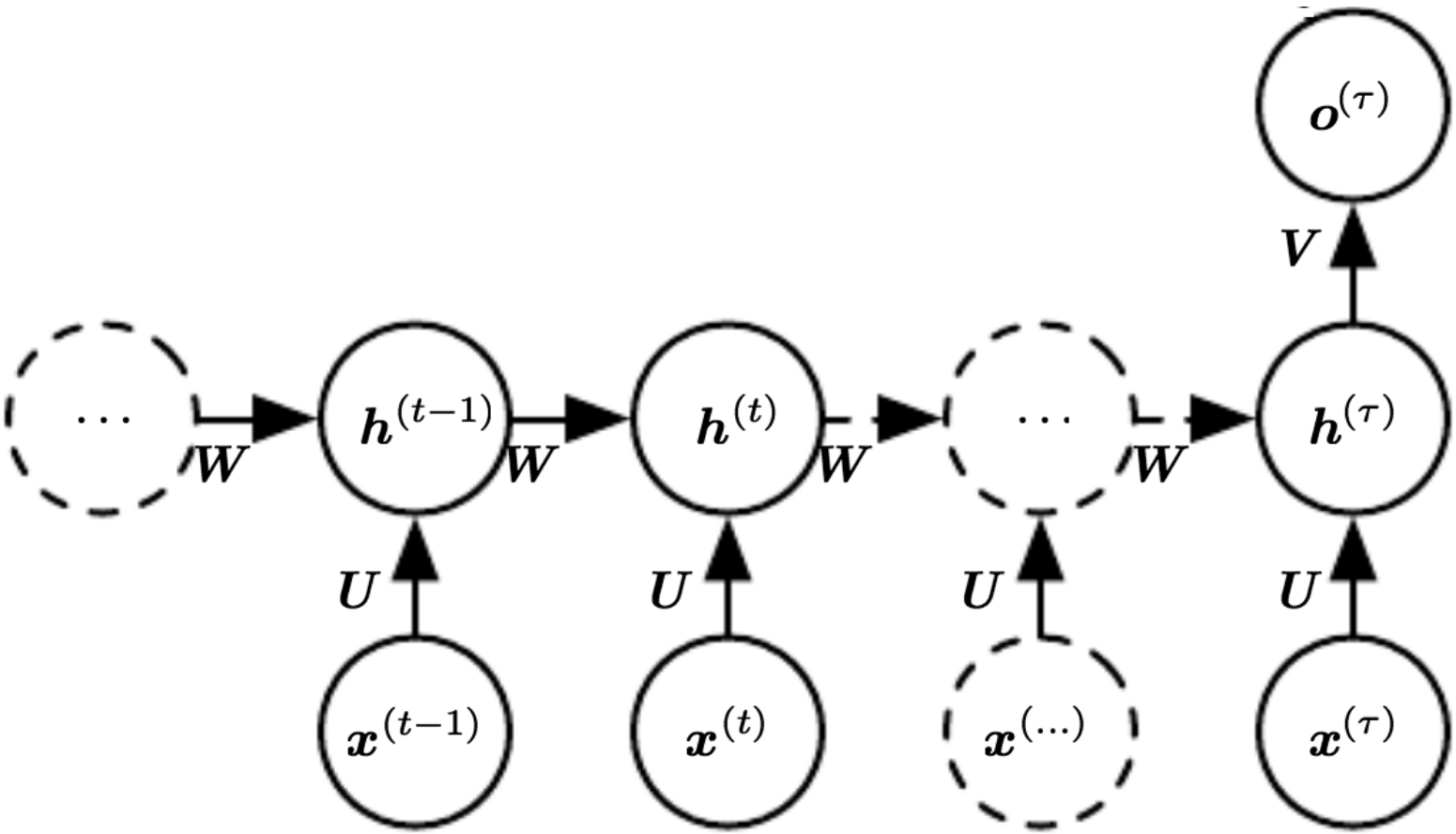
Sequences

- Like with images, sequences have special structure that can be exploited by custom network architectures.
- The fundamental idea is that of a *recurrent* architecture that applies the same function to the data (and hidden state) for every position in a sequence.
- This allows for a model to be applied to sequences of any length using a fixed number of parameters.
- To learn a recurrent model, it suffices to “unroll" it for a number of steps equal to the length of sequence, and then backprop through the unrolled network.

Recurrent Neural Network (Dense Output)



Recurrent Neural Network (Final Output Only)

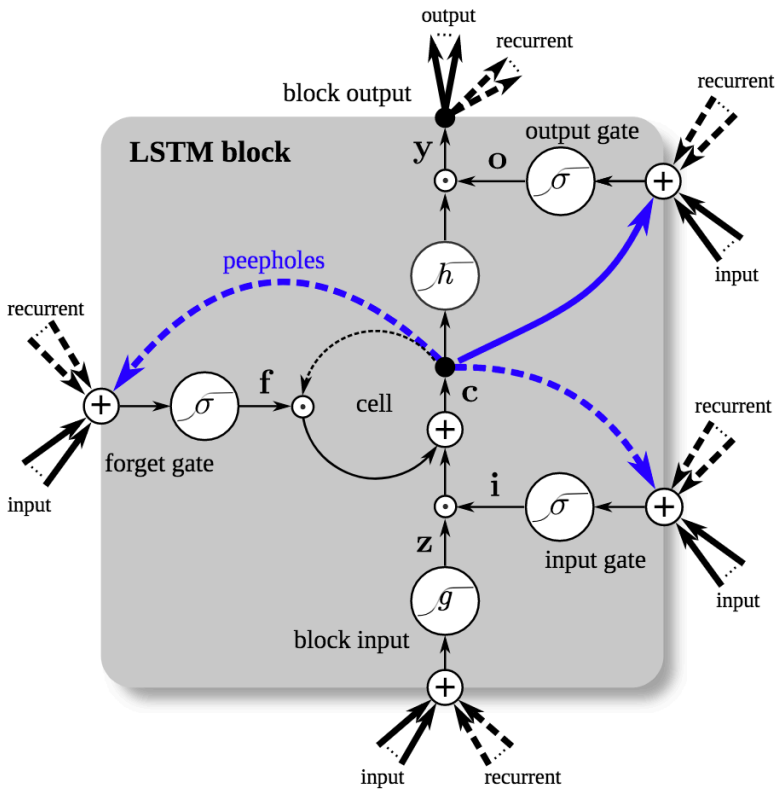


Learning with Long Sequences

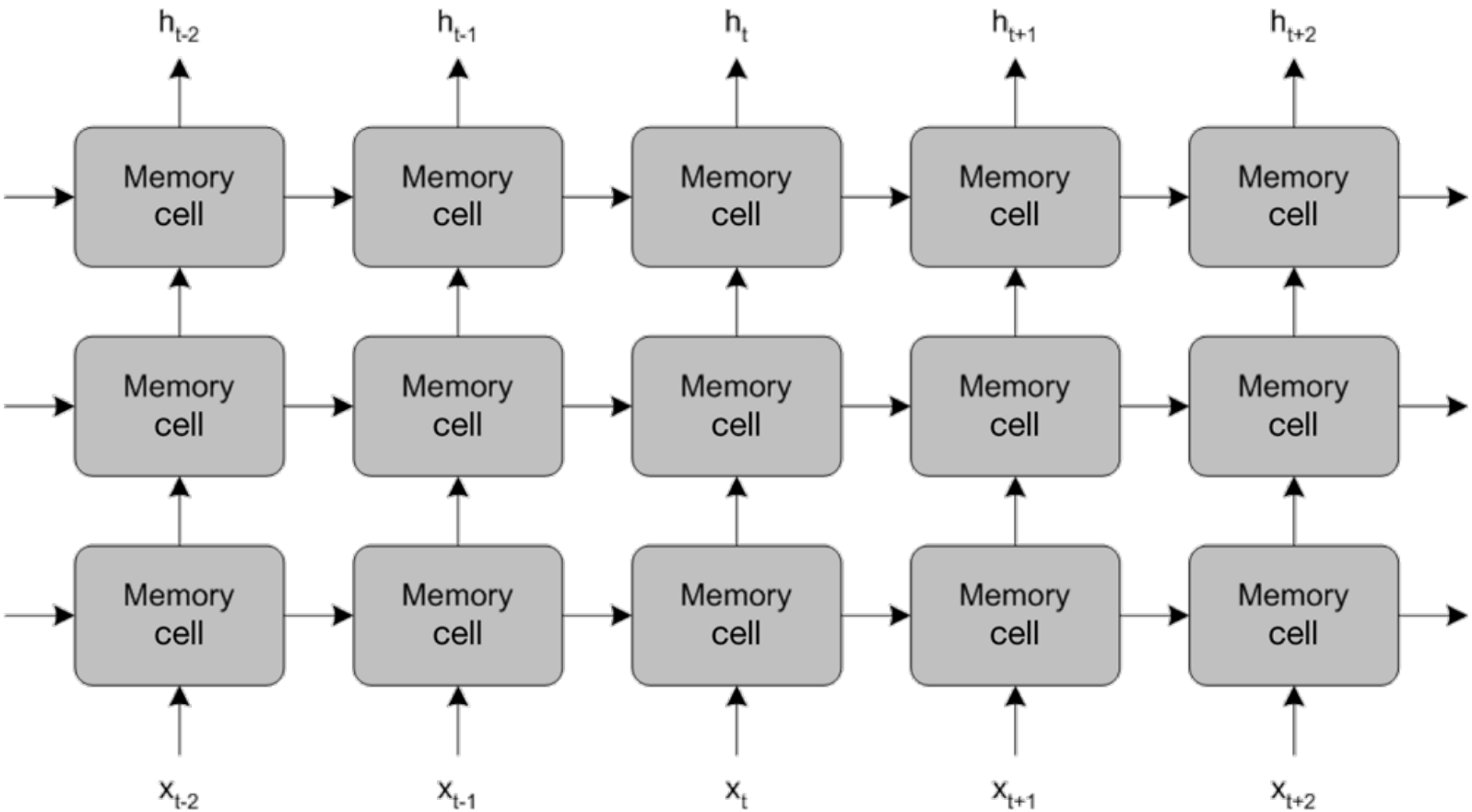
- Learning recurrent networks for long sequences, particularly for data with long-range dependencies between inputs and outputs, can be very challenging.
- One major problem is that gradients can explode or vanish depending on the non-linearities used since a recurrent network applied to a long sequence is equivalent to a very deep feed-forward network.
- To deal with this problem, specialized architectures have been developed to improve the propagation of gradient information backward through sequences during learning, and to make it easier to hold on to useful information for long periods of time.

The Long Short-Term Memory Cell

$$
\begin{aligned}
\bar{\mathbf{z}}^t &= \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z \\
\mathbf{z}^t &= g(\bar{\mathbf{z}}^t) && \textit{block input} \\
\bar{\mathbf{i}}^t &= \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i \\
\mathbf{i}^t &= \sigma(\bar{\mathbf{i}}^t) && \textit{input gate} \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f \\
\mathbf{f}^t &= \sigma(\bar{\mathbf{f}}^t) && \textit{forget gate} \\
\mathbf{c}^t &= \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t && \textit{cell} \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o \\
\mathbf{o}^t &= \sigma(\bar{\mathbf{o}}^t) && \textit{output gate} \\
\mathbf{y}^t &= h(\mathbf{c}^t) \odot \mathbf{o}^t && \textit{block output}
\end{aligned}
$$



Deep LSTM Models



Capacity

- **Model Capacity:** Different models provide access to different spaces of distributions or prediction functions. Informally, the capacity of a model is the size of space of distributions or prediction functions it can represent, and we have seen models with effectively unlimited capacity.
- **Question:** Why can't we always just use a model with infinite capacity for every problem?
- **Answer:** While this would always minimize loss on training data, what we really care about for prediction problems is *generalization loss*.

Generalization

- **Generalization loss:** The loss of a learned model on on *future, unseen examples*.
- **Capacity Control:** To achieve optimal generalization performance for a given training set, we often need to control model capacity carefully.
- How to control capacity to optimize generalization loss is an extremely important problem in machine learning.

Overfitting and Underfitting

- **Overfitting:** The generalization loss for a model is worse than the training loss. This results from choosing a classifier with too much capacity so that it models the noise in the training data.
- **Underfitting:** Occurs when the capacity of the model is too low to capture the actual structure in the training data, leading to both high training loss and high generalization loss.

Bias-Variance Trade-Off

- **Bias:** A model is said to have low *bias* if the true prediction function (or distribution) can be approximated closely by the model.
- **Variance:** A model is said to have low *variance* if the prediction function it constructs is stable with respect to small changes to the training data.
- **Bias-Variance Dilemma:** To achieve low generalization error, we need classifiers that are low-bias and low-variance, but this isn't always possible.
- **Bias-Variance and Capacity:** On complex data, models with low capacity have low variance, but high bias; while models with high capacity have low bias, but high variance.

Hyperparameters

- In order to control the capacity of a model, it needs to have capacity control parameters.
- Because capacity control parameters can not be chosen based on training error, they are often called *hyperparameters*.
- Thus, in order to minimize generalization loss, we need to method to optimally select hyperparameters.

Model Selection and Evaluation

- The problem of optimizing model complexity hyperparameters is often referred to as *model selection*.
- We often want to both perform model selection to optimize hyperparameters and estimate the generalization loss of the optimal hyper-parameters.
- We may also want to be able to compare two or more models to assess whether differences in their generalization loss values are statistically significant.

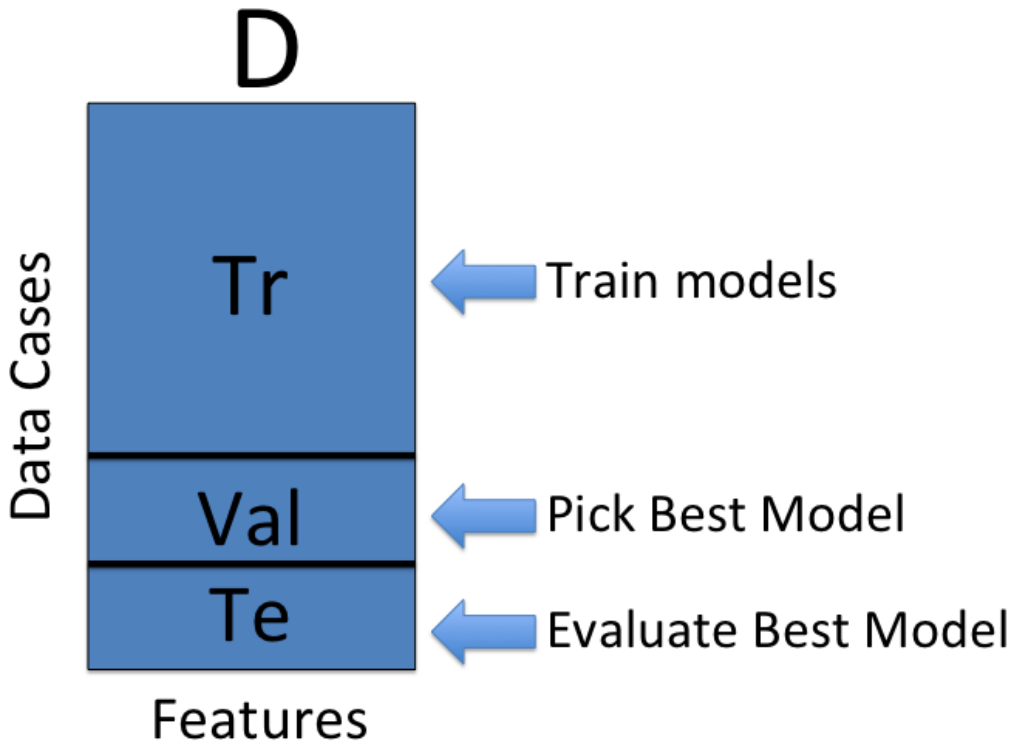
Model Selection and Evaluation Methods

- To obtain valid results, we need to use appropriate methodology and construct learning experiments carefully.
- **Guiding Principle:** Data used to estimate generalization error can not be used for any other purpose (ie: model training, hyperparameter selection, feature selection, etc.) or the results of the evaluation will be **biased**.
- The basic methods used to assess generalization performance thus require data partitioning.

Recipe 1: Train-Validation-Test

- Given a data set D, we randomly partition the data cases into a training set (Tr), a validation set (V), and a test set (Te). Typical splits are 60/20/20, 80/10/10, etc.
- Models M_i are learned on Tr for each choice of hyperparameters H_i
- The validation error Val_i of each model M_i is evaluated on V.
- The hyperparameters H_* with the lowest value of Val_i are selected and the classifier is re-trained using these hyperparameters on $Tr + V$, yielding a final model M_*
- Generalization performance is estimated by evaluating error/accuracy of M_* on the test data Te.

Example: Train-Validation-Test



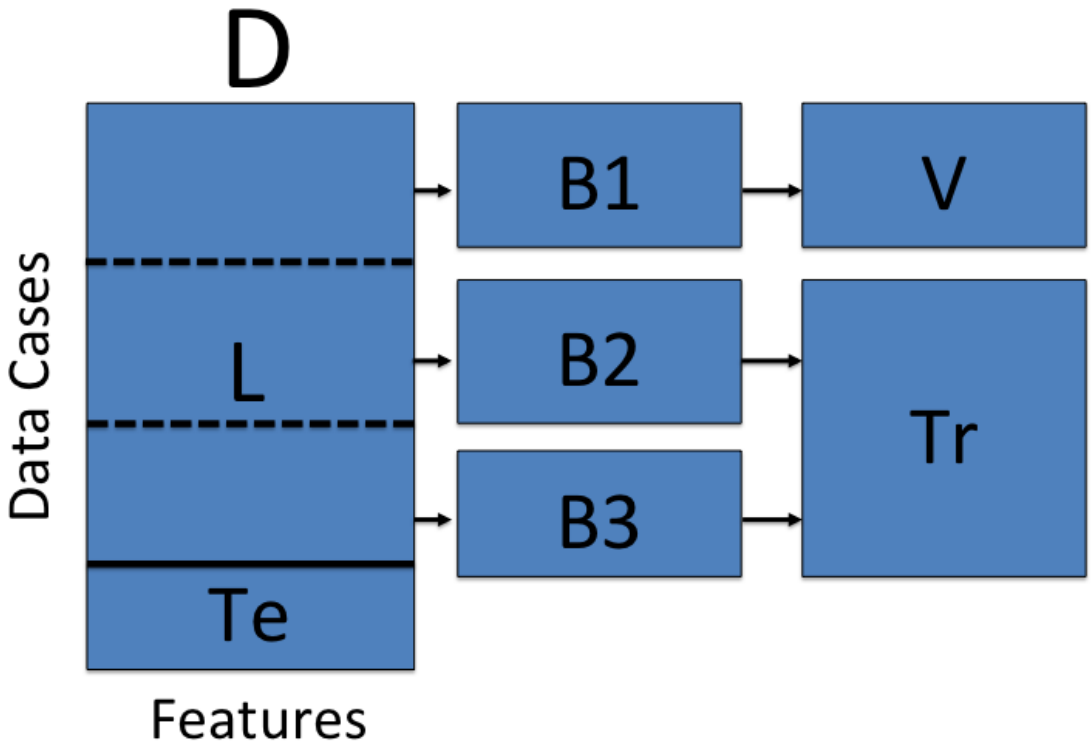
Note that the order of the data cases needs to be randomly shuffled before partitioning D.

Recipe 2: Crossvalidation-Test

- Randomly partition D into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- We next randomly partition L into a set of K blocks $B_1, ..., B_K$.
- For each crossvalidation fold $k = 1, ..., K$:
  - Let $V = B_k$ and $Tr = L/B_k$ (the remaining $K - 1$ blocks).
  - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
  - Compute Val_{ik} of M_{ik} on V.
- Select hyperparameters H_* minimizing $\frac{1}{K} \sum_{k=1}^{K} Val_{ik}$ and re-train model on L using these hyperparameters, yielding final model M_*.
- Estimate generalization performance by evaluating error/accuracy of M_* on Te.

Example: 3-Fold Cross Validation and Test

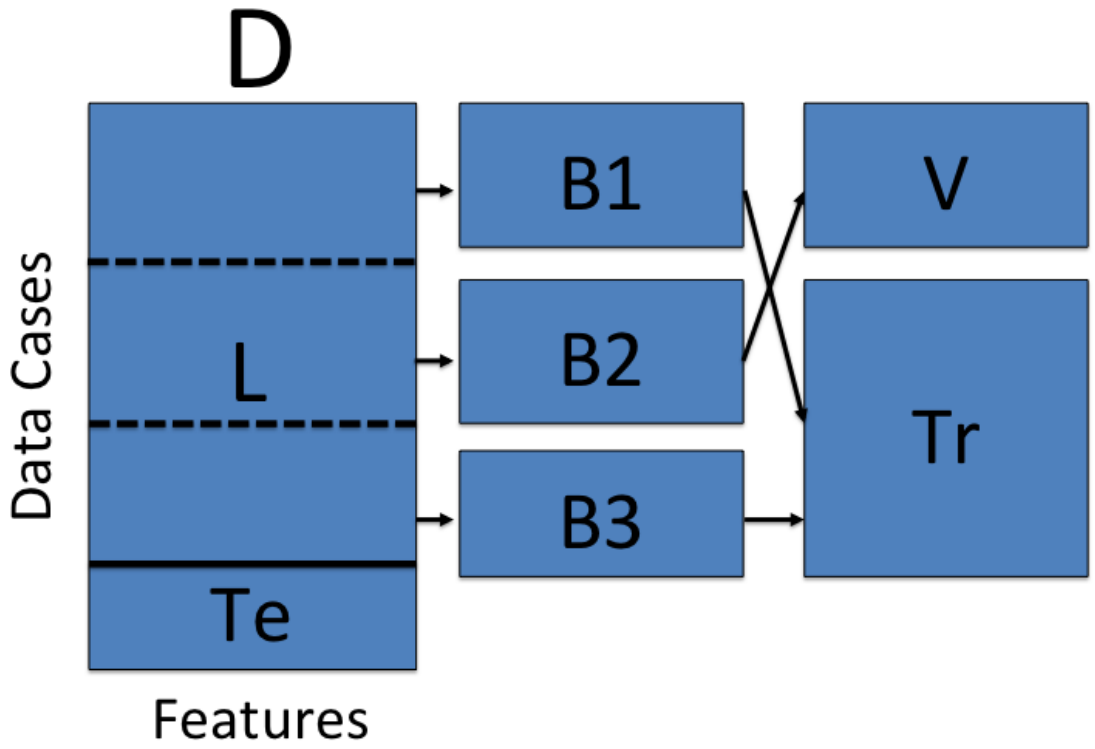
First Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning D into L and Te.

Example: 3-Fold Cross Validation and Test

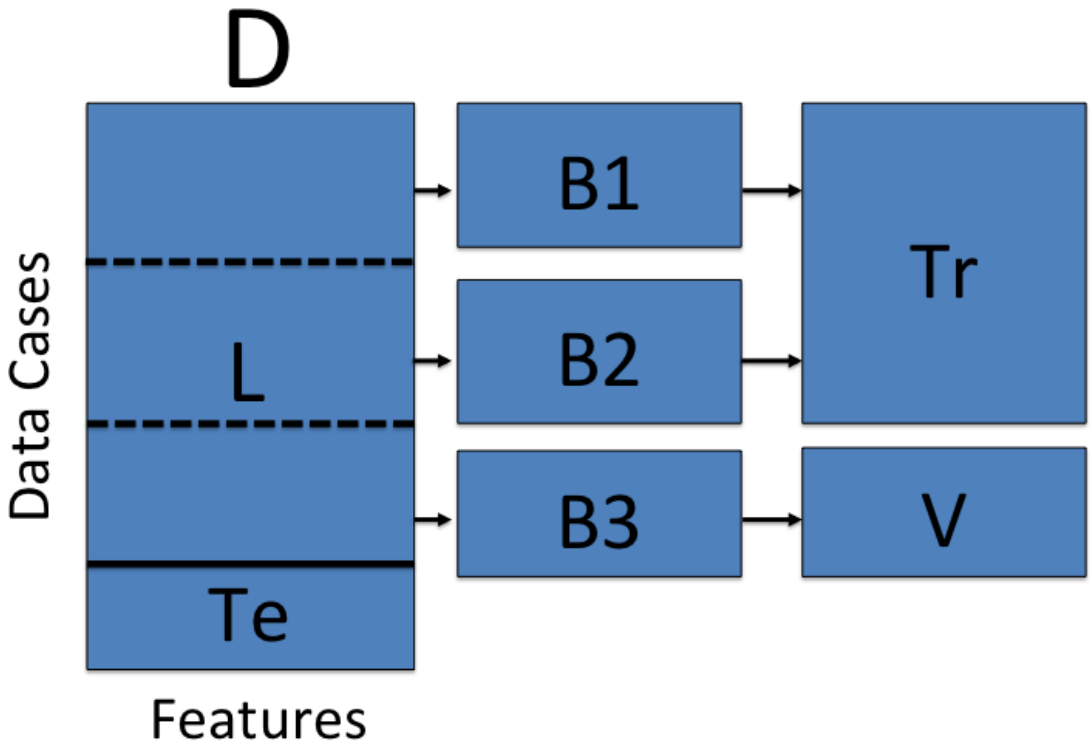
Second Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning D into L and Te.

Example: 3-Fold Cross Validation and Test

Third Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning D into L and Te.

Recipe 3: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:
  - Let $Te_j = C_j$ and $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, ..., B_K$.
  - For $k = 1, ..., K$:
    - Let $V = B_k$ and $Tr = L_j/B_k$.
    - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
    - Compute error Val_{ik} of M_{ik} on V.
  - Select hyperparameters H_* minimizing $\frac{1}{K} \sum_{k=1}^{K} Val_{ik}$ and re-train model on L_j using these hyperparameters, yielding model M_{*j}.
  - Compute Err_j by evaluating M_{*j} on Te_j.
- Estimate generalization error using $\frac{1}{J} \sum_{j=1}^{J} Err_j$
- We can define a similar nested random resampling validation procedure.

Trade-Offs

- In cases where the data has a benchmark split into a training set and a test set, we can use Recipes 1 or 2 by preserving the given test set and splitting the given training set into train and validation sets as needed.
- In cases where there is relatively little data, using a single held out test set will have high variance and potentially high bias. In these cases, Recipe 3 often provides a better estimate of generalization error, but has higher computational cost.
- Choosing larger K in cross validation will reduce bias. However, both increase computational costs. $K = 3, 5, 10$ are common choices for cross validation. $K = N$, also known as Leave-one-out cross validation is also popular when feasible.
- If data have nested or multi-level structure, that needs to be taken into account.

Significance Testing

- An advantage of using methods that produce multiple estimates of generalization loss using multiple test data partitions is that we can use these estimates to assess statistical significance of differences in mean loss.
- The hypothesis test that is most often used in machine learning is a paired t-test. This test leverages the fact that we can evaluate different models on exactly the same collection of test data partitions and looks at the difference in performance on a per test set basis.
- Note that the assumptions required for standard hypothesis tests are not exactly satisfied under cross validation, but application of statistical testing is preferred to only assessing mean performance.
- When comparing two models, more rigorous testing can be performed to correct for multiple comparisons.

Examples: Significance Testing

- For example, instead of selecting the model that yields the minimum validation loss in cross validation-test, we can select the least complex (lowest capacity) model that is not statistically significantly worse than the best model.
- Similarly, using the nested cross validation method, we can use a paired t-test to assess differences between different types of models (neural networks vs SVMs, for example), after optimizing away their unknown hyperparameter values.

Reporting Significance Testing

- It is common practice to report results in tables in terms of mean generalization loss plus/minus a variability estimate (e.g., standard error of the mean, standard deviation, or 95% confidence interval, etc).
- When plotting results, error bars can be used to provide information about variability. Use of box plots that show multiple statistics can be helpful (e.g., mean, median, 2nd and 3rd quartiles).
- It is important to know *which* variability measure has been reported. For example, when standard errors overlap, differences are not statistically significant under an un-paired t-test. If 95% confidence intervals overlap, differences still might be significant.