Introduction to Machine Learning

Alexey Svyatkovskiy

Princeton University

Outline

- Supervised machine learning
  - Classification and regression trees: ID3
  - Random forests: bootstrap aggregation
  - Feed-forward neural networks
    - Activation function, loss
    - Regularization: dropout
    - Training process: SGD
  - Convolutional neural networks
  - Recurrent neural networks
    - Gated RNNs: LSTM

Supervised machine learning

- Let $X = \{X_{ji}\}$ be the feature matrix (n rows, p columns)
- And y_j be a n-vector of labels
- **Supervised learning** is the machine learning task of inferring a function

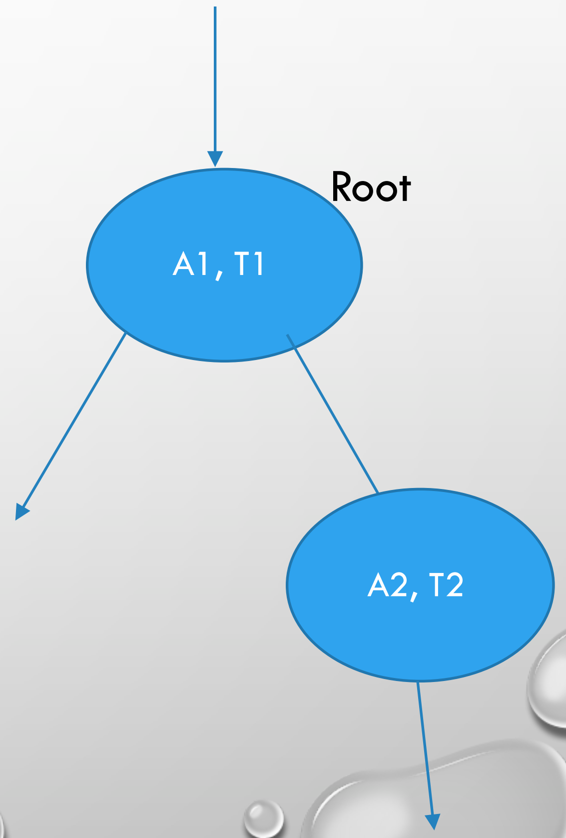
$$f(X_{j\bullet}) = y_j$$

from *labeled* training data

- Decision trees, random forests and deep neural networks are some commonly used supervised machine learning techniques

Decision trees

- A decision tree is a binary tree. At each of the internal nodes, it chooses a feature *i* and a threshold *T*
  - Each leaf has a value
- Evaluation of the model is just a traversal of the tree from the root
- At each node, for example *j*, we go down the left branch if $X_{ji} < T$ and the right branch otherwise
- The value of the model $f\left(X_{ji}\right)$ is the value at the value at the terminating leaf of this traversal
- **Classification And Regression Tree (CART)** analysis is an umbrella term used to refer to the decision trees which output the class label or a real value

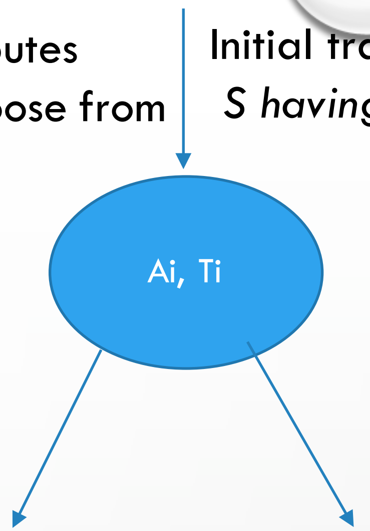


Decision trees: ID3

- The ID3 algorithm iterates through every attribute of the set {Ai, Ti} and calculates the information gain (or Gini index) of that attribute
  - It then selects the attribute which has the largest information gain
  - The set is then split by the selected attribute to produce subsets of the data
- The algorithm continues to recurse on each subset, considering only attributes never selected before
- Recursion on a subset may stop in one of these cases:
  - Every element in the subset belongs to the same class. Then the node is turned into a leaf and labelled with the class of the examples
  - There are no more attributes to be selected, but the examples still do not belong to the same class, then the node is turned into a leaf and labelled with the most common class of the examples in the subset
  - There are no more examples in the subset

{Ai, Ti} – set of attributes and thresholds to choose from

Initial training dataset *S having X classes*



$p(x)$ - fraction of elements of class x

Entropy of the set *S* would be:

$$H(S) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Information gain for a given attribute *A* on the set *S*:

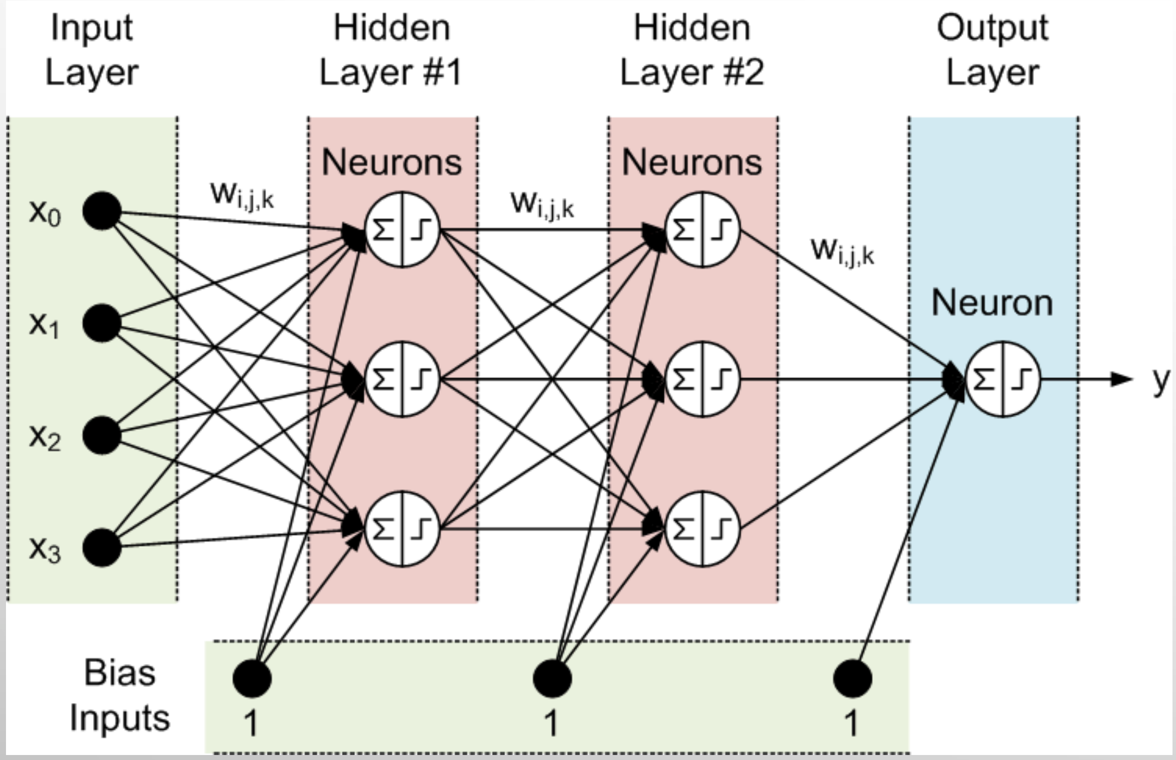
$$IG(A,S) = H(S) - \sum_{t \in T} p(t) H(t)$$

Random forest

- A **random forest** is just an ensemble of decision trees
- The predicted value is just the average of the trees (for both regression and classification problems - for classification problems, it is the probabilities that are averaged).
- Why “random”? There is two sources of randomness:
  - **Bootstrap aggregation (subsampling)**: each tree is trained on a subset of data selected at random with replacement
  - **Select subset of training features**
- **Extremely Random Forests**: Instead of choosing the optimal split amongst a subset of features, we choose random values amongst randomly generated thresholds

Feed forward nets

- Feed forward neural network is a sequence of neurons arranged in layers and interconnected with each other
  - Each neuron connected to all neurons from adjacent layers
  - No loops (recurrent connections) is allowed



Feed forward equations

$$a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)$$
$$h^{(k)}(x) = \mathrm{g}\left(a^{(k)}(x)\right)$$
$$o(x) = \ h^{(L)}\ (x) = o(a^{(L)}(x))$$

Activation functions

- Sigmoid

$$g(a) = \frac{1}{1-e^{-a}}$$

- Hyperbolic tangent

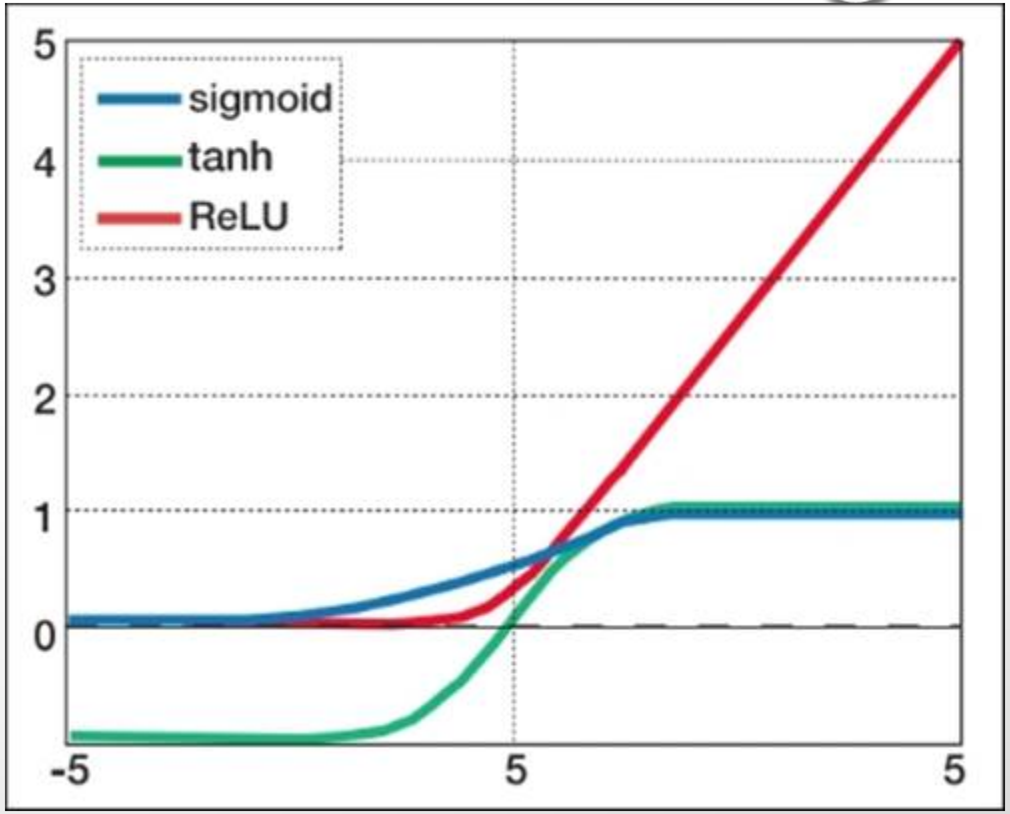
$$g(a) = \frac{e^{2a} - 1}{e^{2a} + 1}$$

- Rectified linear unit

$$g(a) = \max(0, a)$$

- Softmax (output activation)

$$g(a) = \frac{e^{a}}{\sum e^{a}}$$



Loss functions

- A loss function is a measure of "how good" a neural network did with respect to it's given training sample and the expected output
- During backpropagation, loss function is differentiated with respect to weights
- Quadratic cost, also known as means squared error, maximum likelihood and sum squared error

$$L(a) = 0.5 \sum_{j} (a_j^{(L)} - y_j \ )^2$$

- Cross-entropy loss, also known as Bernoulli negative log-likelihood and Binary Cross-Entropy

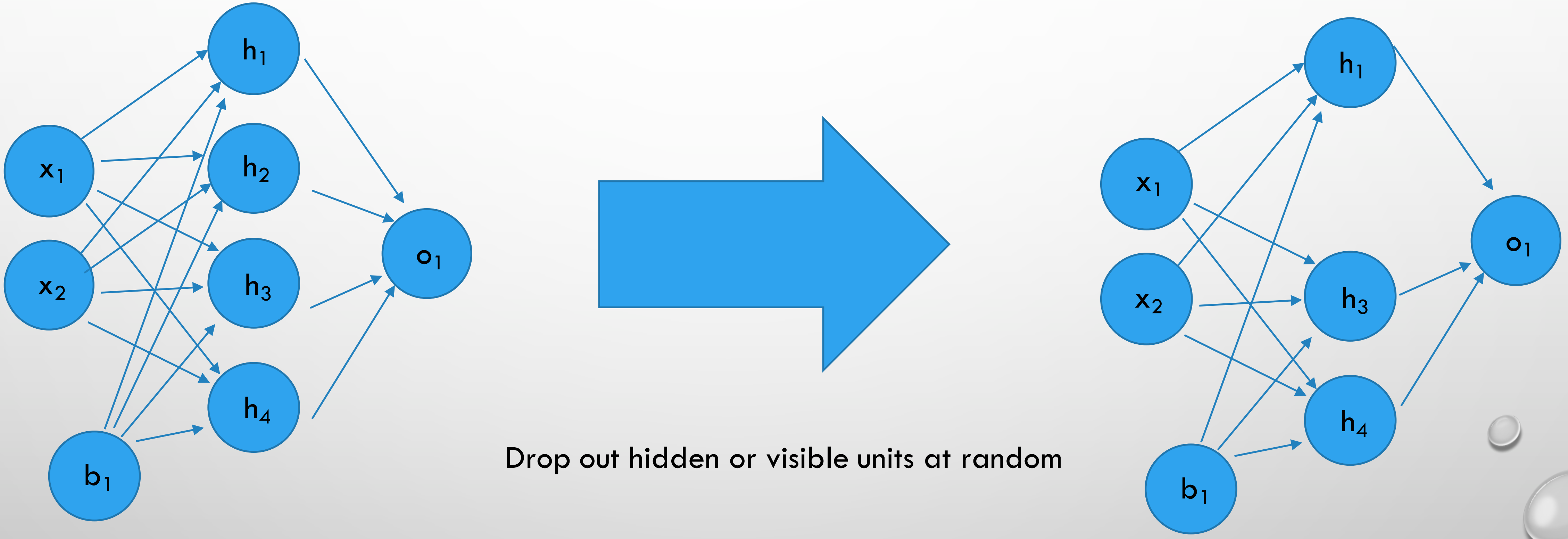
$$L(a) = -\sum_{j} (y_j \ln a_j^{(L)} + \left(1 - y_j\right) \ln(1 - a_j^{(L)}))^2$$

- Hinge loss also known as maximum margin loss

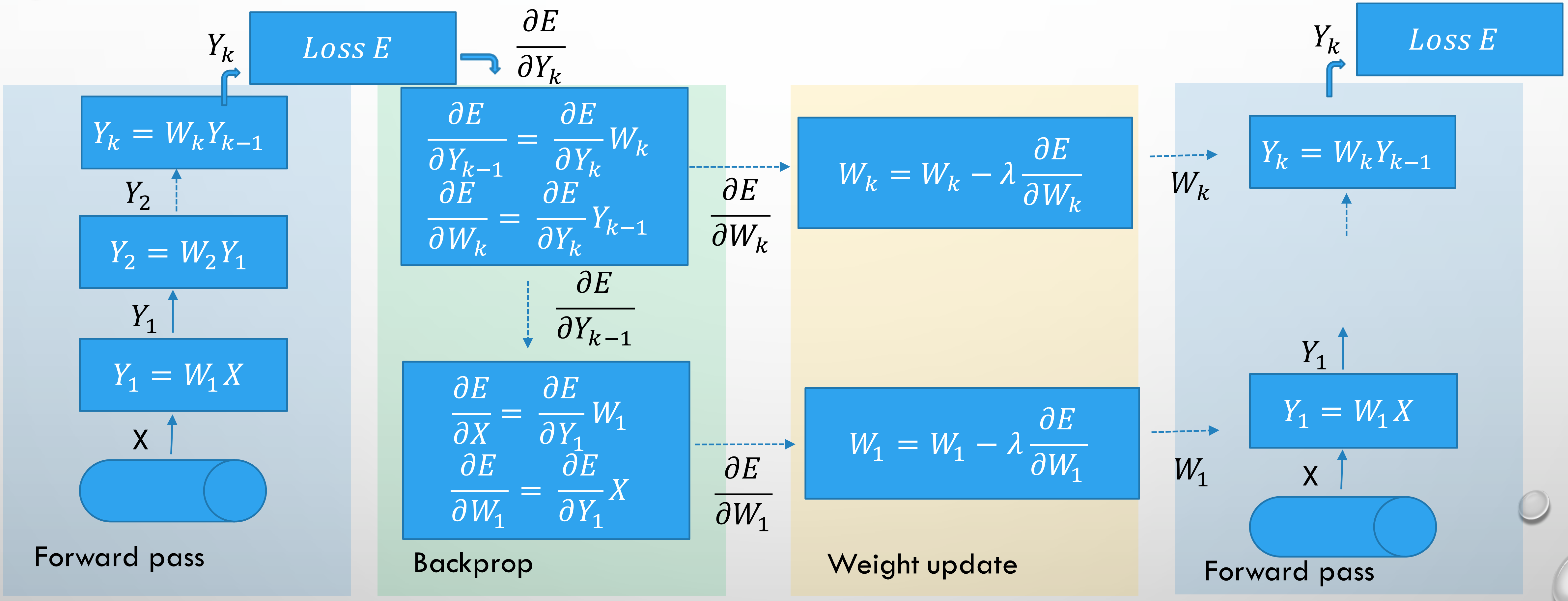
$$L(a) = \max(0,1 - y_j a_j^{(L)})$$

Dropout regularization

- Dropout is a regularization technique for reducing overfitting in neural networks by dropping out units (both hidden and visible) in a neural network



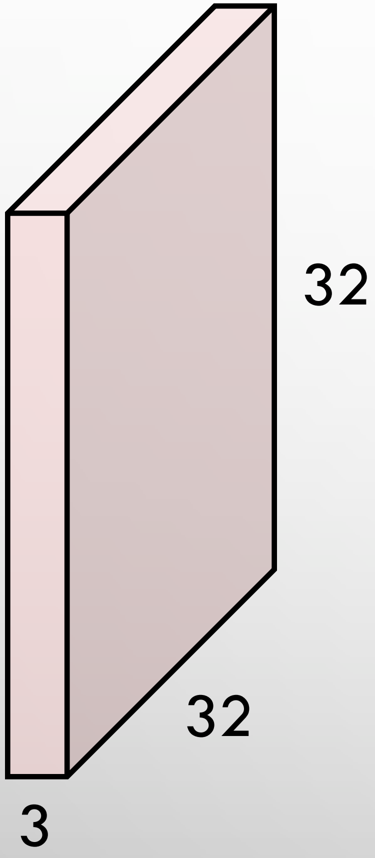
Training flow: SGD



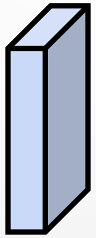
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

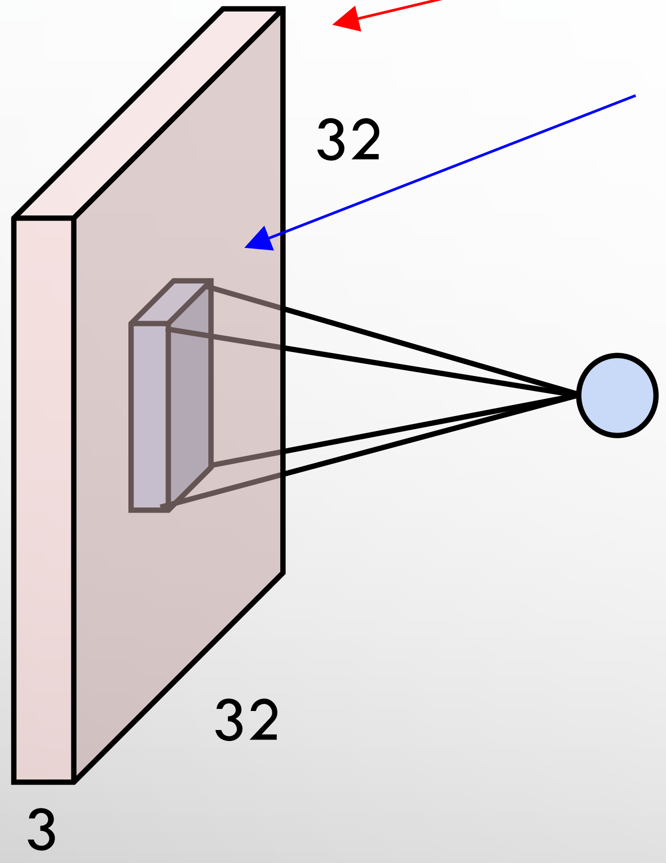


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

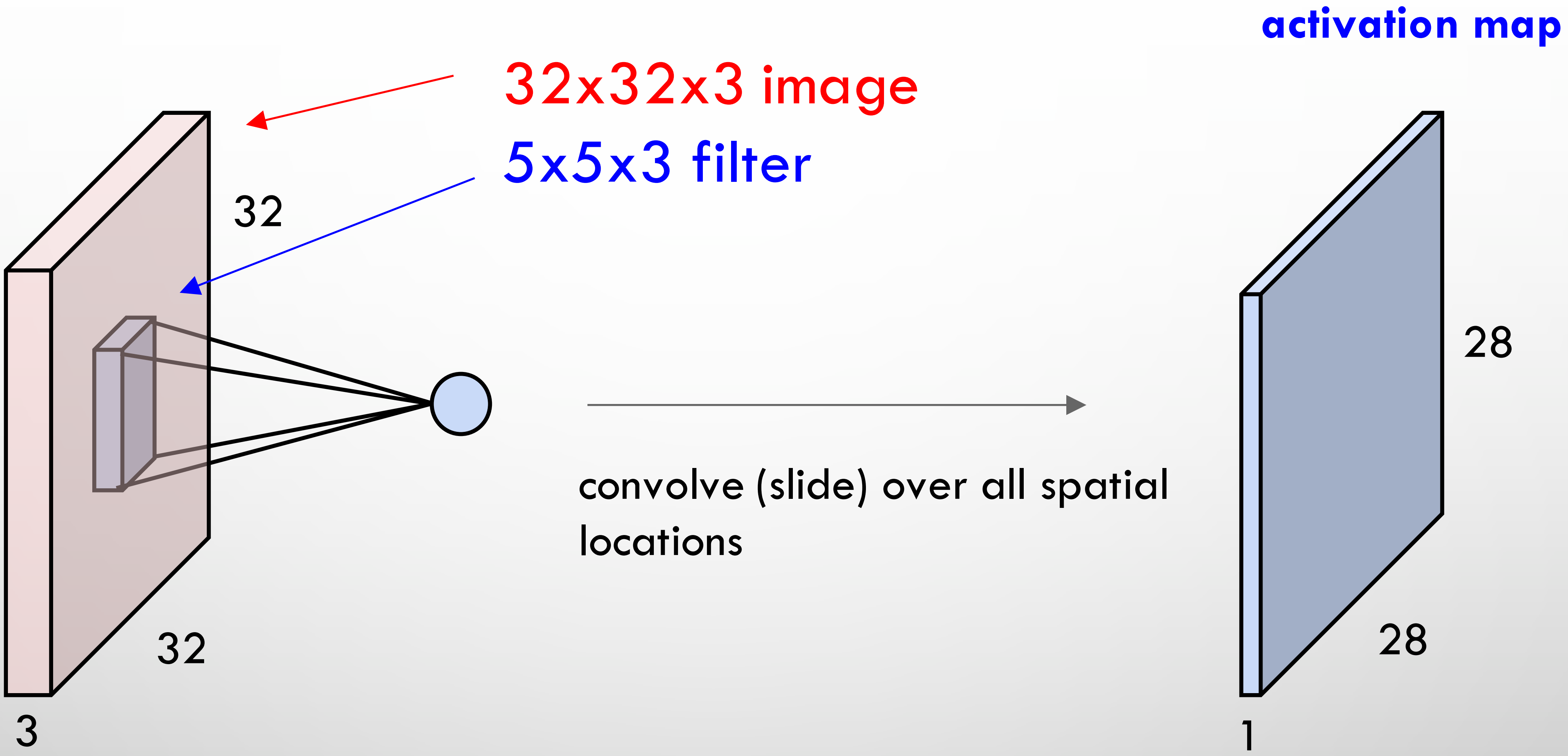
5x5x3 filter w



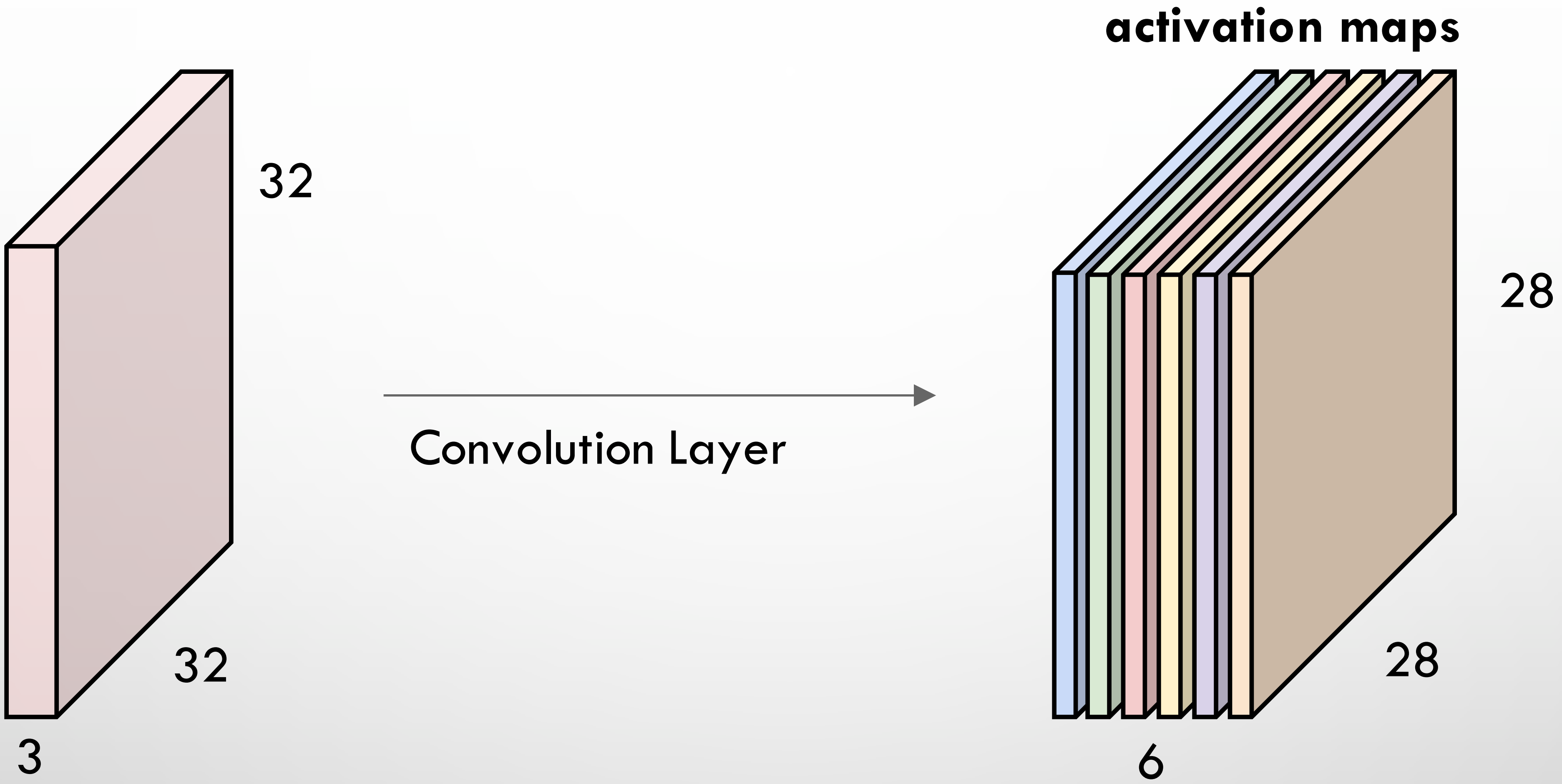
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

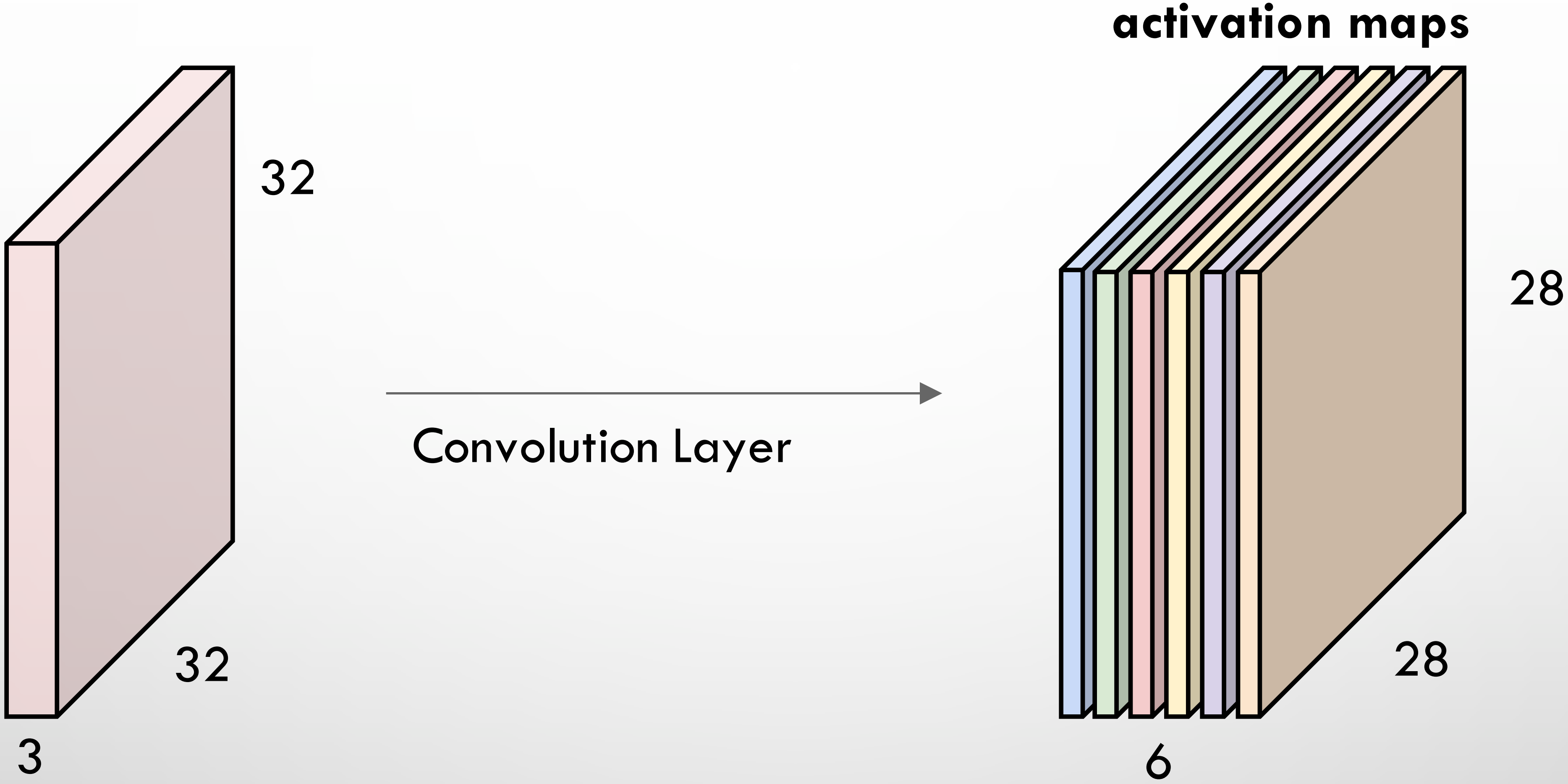


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

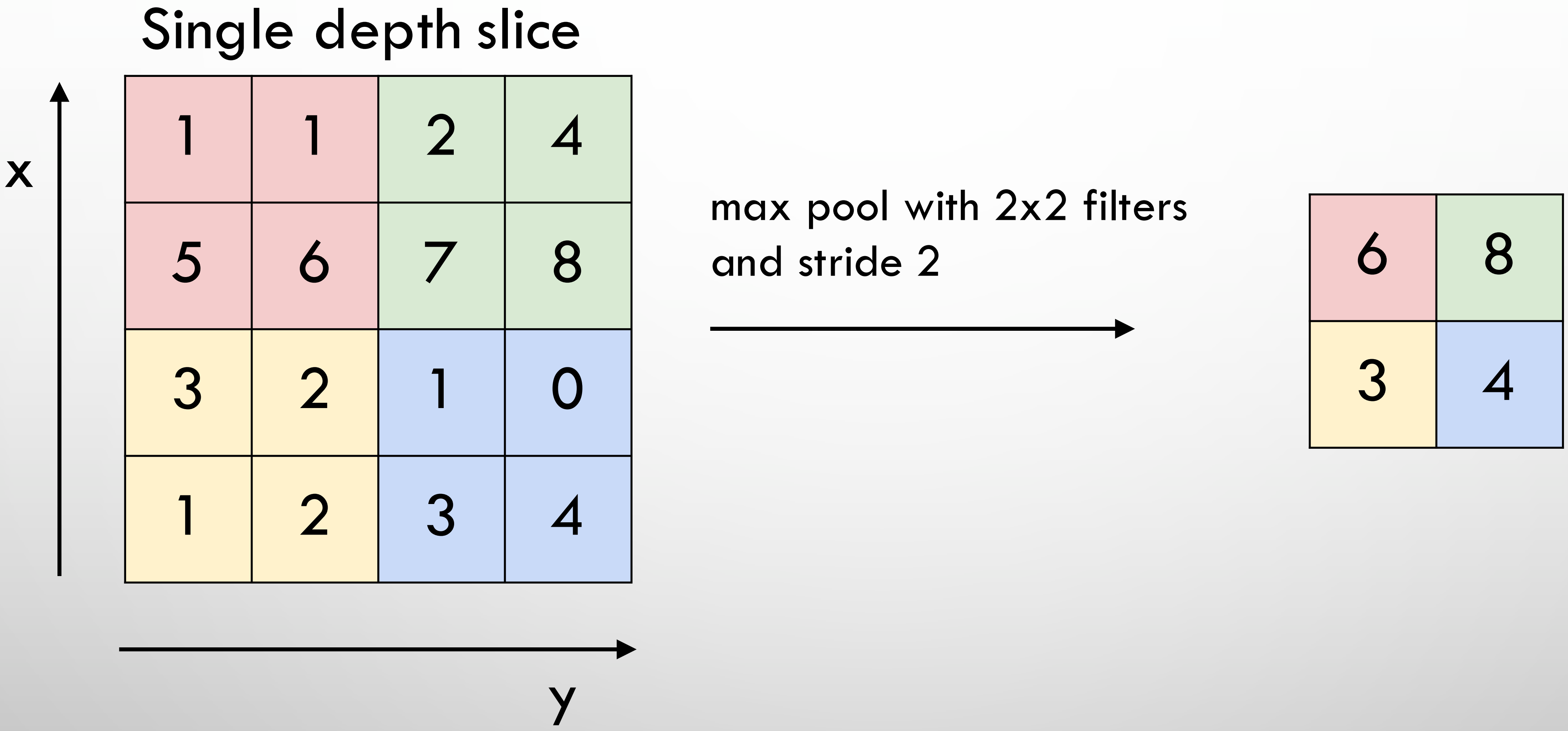


We processed [32x32x3] volume into [28x28x6] volume.

Q: how many parameters are used instead?

A: (5*5*3)*6 = **450 parameters**, (5*5*3)*(28*28*6) = **~350K multiplies**

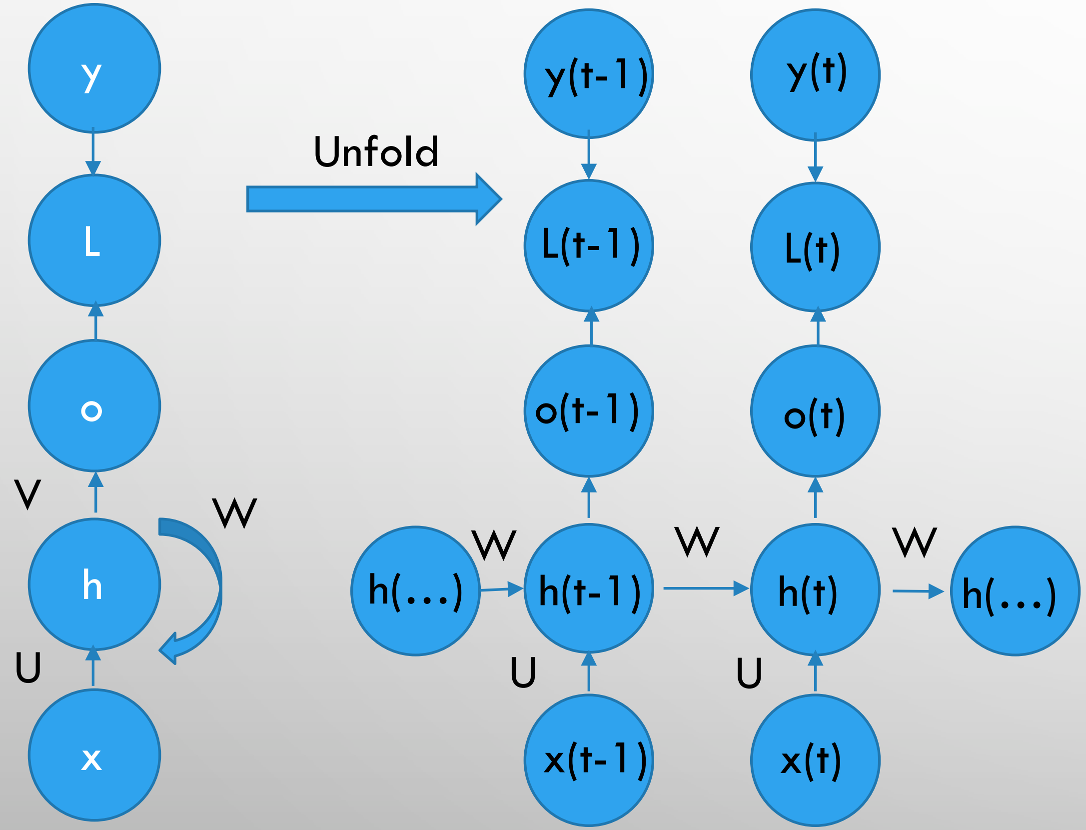
Max Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Recurrent Neural Nets: basic description

- RNNs are a family of neural networks to process sequential data
- Feed forward equations are recurrent:

$$a(t) = b + Wh(t-1) + Ux(t)$$
$$h(t) = \tanh\big(a(t)\big)$$
$$o(t) = c + Vh(t)$$



Notations:
x – input sequence,
U – is the input to hidden weight matrix,
W - hidden to hidden,
V – hidden to output weights
b,c are the biases
tanh() is the activation function (non-linearity)
o – output sequence
Loss **L** and target values are denoted as **y**

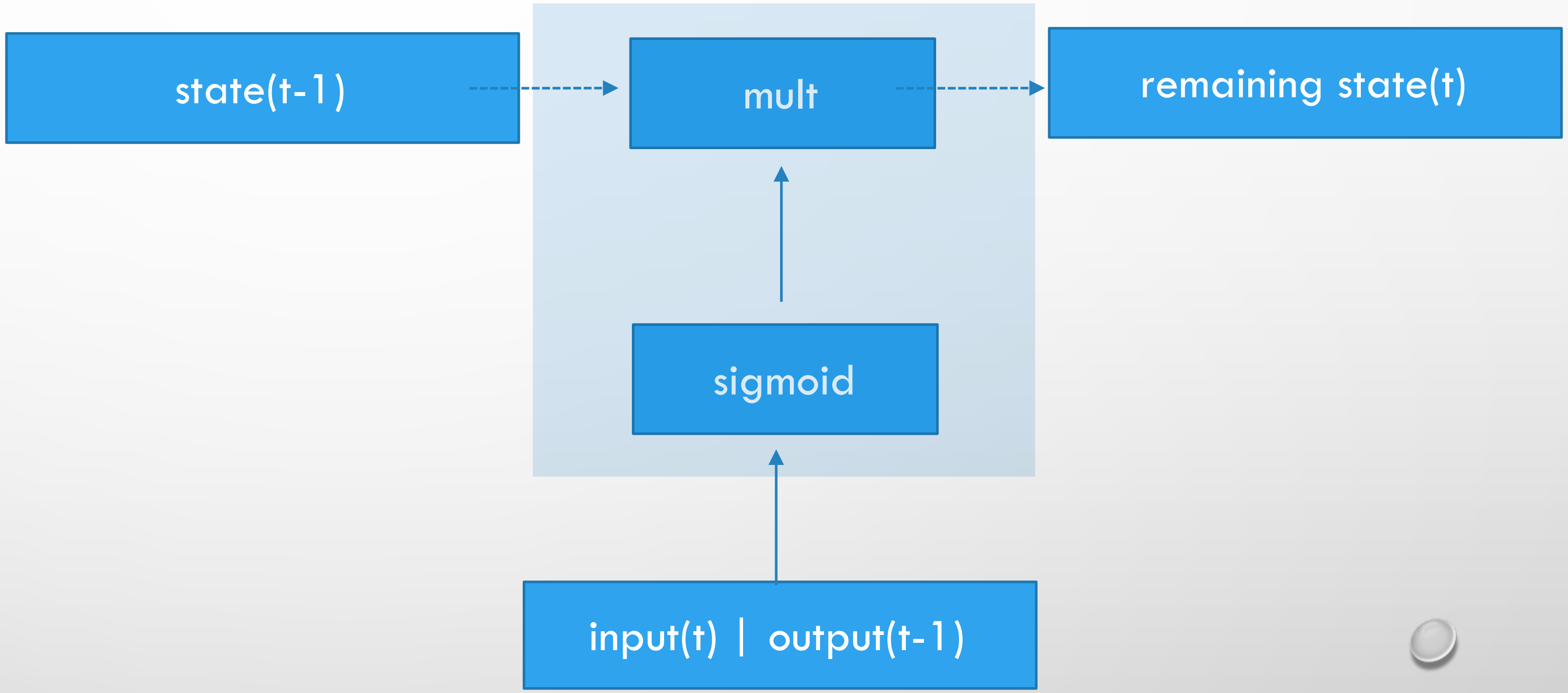
Gated units, LSTM cell

- LSTM is a gated RNN
- LSTM introduces a self-loop – an internal recurrence, in addition to the outer recurrence of the RNN
- The weight of this self-loop is controlled by a forget gate – a notion of memory as input sequence is fed to the model, some information is accumulated in the internal memory
- LSTMs are stateful, as opposed to feedforward neural networks
- $f_i(t) = \sigma(b_i^f + \sum_j U_{ij}^f x_j(t) + \sum_j W_{ij}^f h_j(t-1))$
- $s_i(t) = f_i(t) s_i(t-1) + g_i(t) \sigma(b_i + \sum_j U_{ij} x_j(t) + \sum_j W_{ij} h_j(t-1))$
- $g_i(t) = \sigma(b_i^g + \sum_j U_{ij}^g x_j(t) + \sum_j W_{ij}^g h_j(t-1))$
- $h_i(t) = \tanh(s_i(t)) q_i(t)$
- $q_i(t) = \sigma(b_i^o + \sum_j U_{ij}^o x_j(t) + \sum_j W_{ij}^o h_j(t-1))$

Notations:
x – input sequence,
U – is the input to hidden weight matrix,
W - hidden to hidden,
V – hidden to output weights
b,c are the biases
tanh() is the activation function (non-linearity)
s – state unit
f- forget gate unit
g-external input gate unit
q-output gate unit

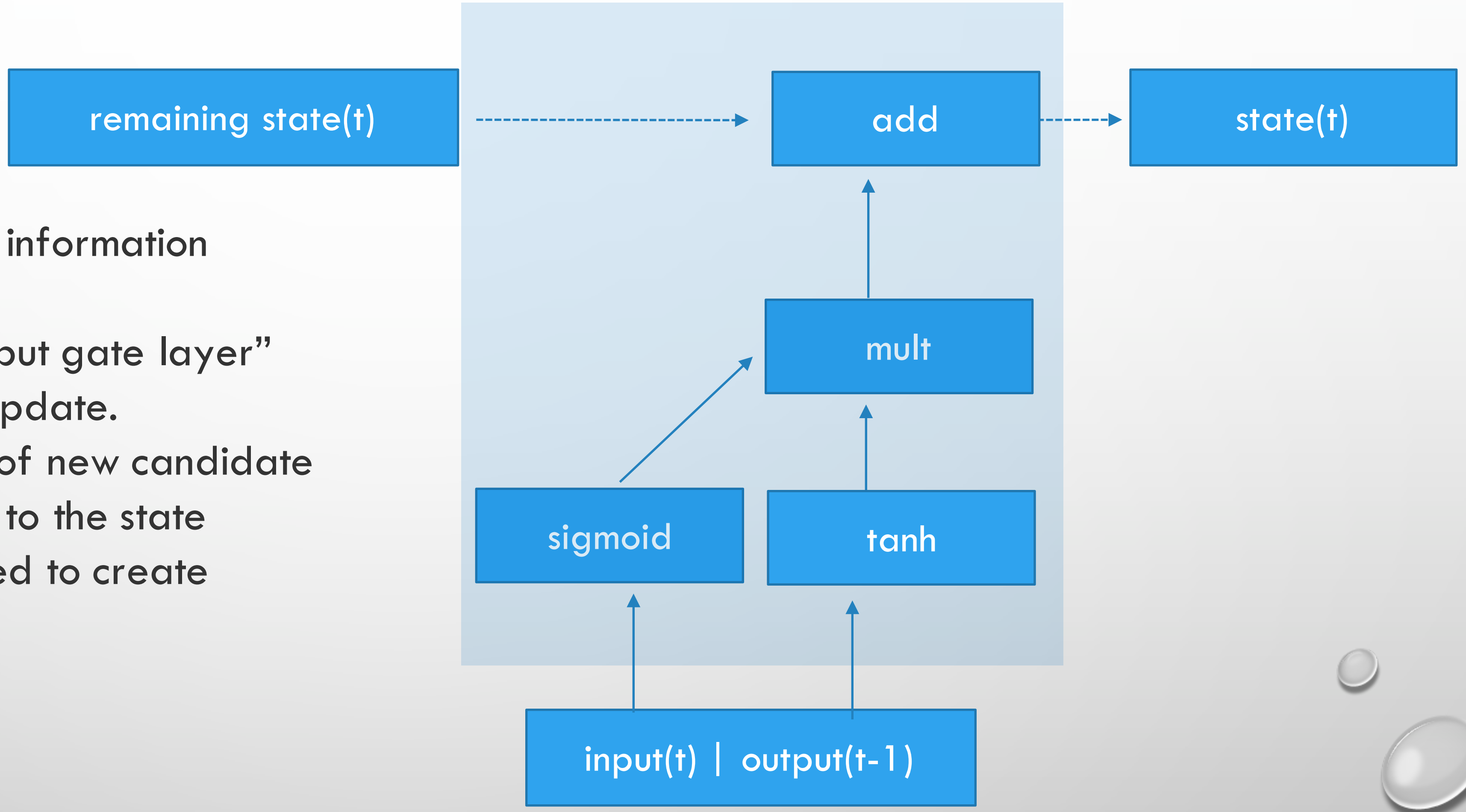
LSTM: forget gate

- The first step in the LSTM cell is to decide what information to throw away from the cell state. This decision is made by a sigmoid



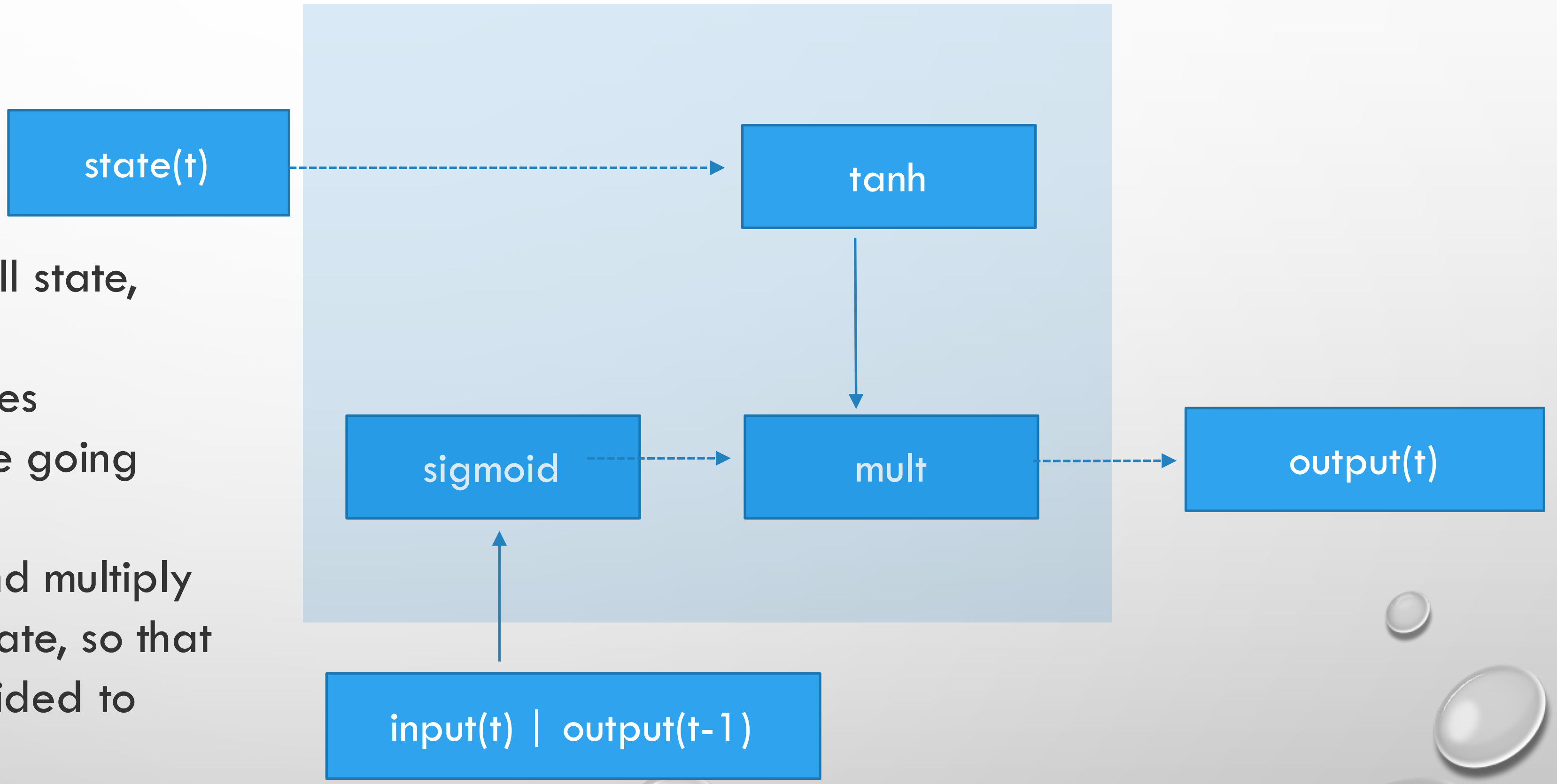
LSTM: input gate

- Next step is to decide what new information to store in the cell state:
  - sigmoid layer called the “input gate layer” decides which values we’ll update.
  - tanh layer creates a vector of new candidate values that could be added to the state
  - these two parts are combined to create an update to the state



LSTM: Output gate

- Output will be based on the LSTM cell state, but will be a filtered version:
  - Run a sigmoid layer which decides what parts of the cell state we're going to output
  - Put the cell state through tanh and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to



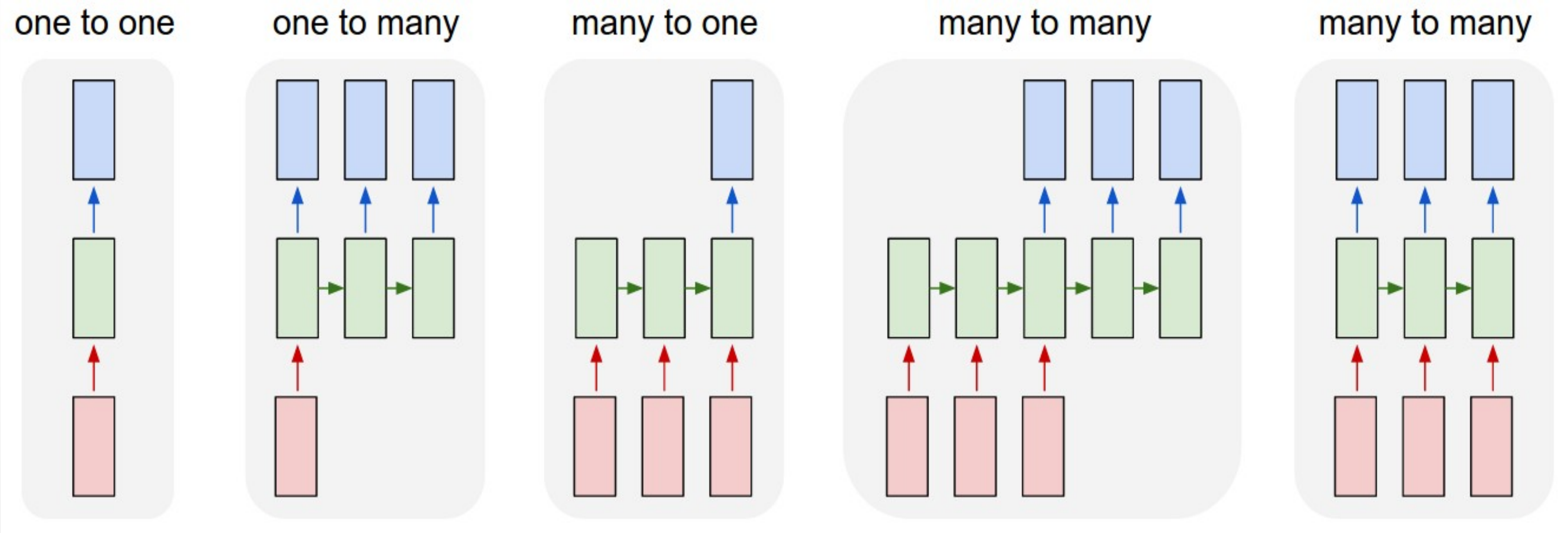
Recurrent Neural Networks (RNNs)

Common theme: sequential data

Output

Hidden

Input



e.g. image classification

e.g. image captioning

e.g. sentiment analysis

e.g. machine translation

e.g. time series prediction, **disruption forecasting**

Some of my work

Fusion Recurrent Neural Net (FRNN) schematic

Output: Disruption coming?

Time-distributed FC layer

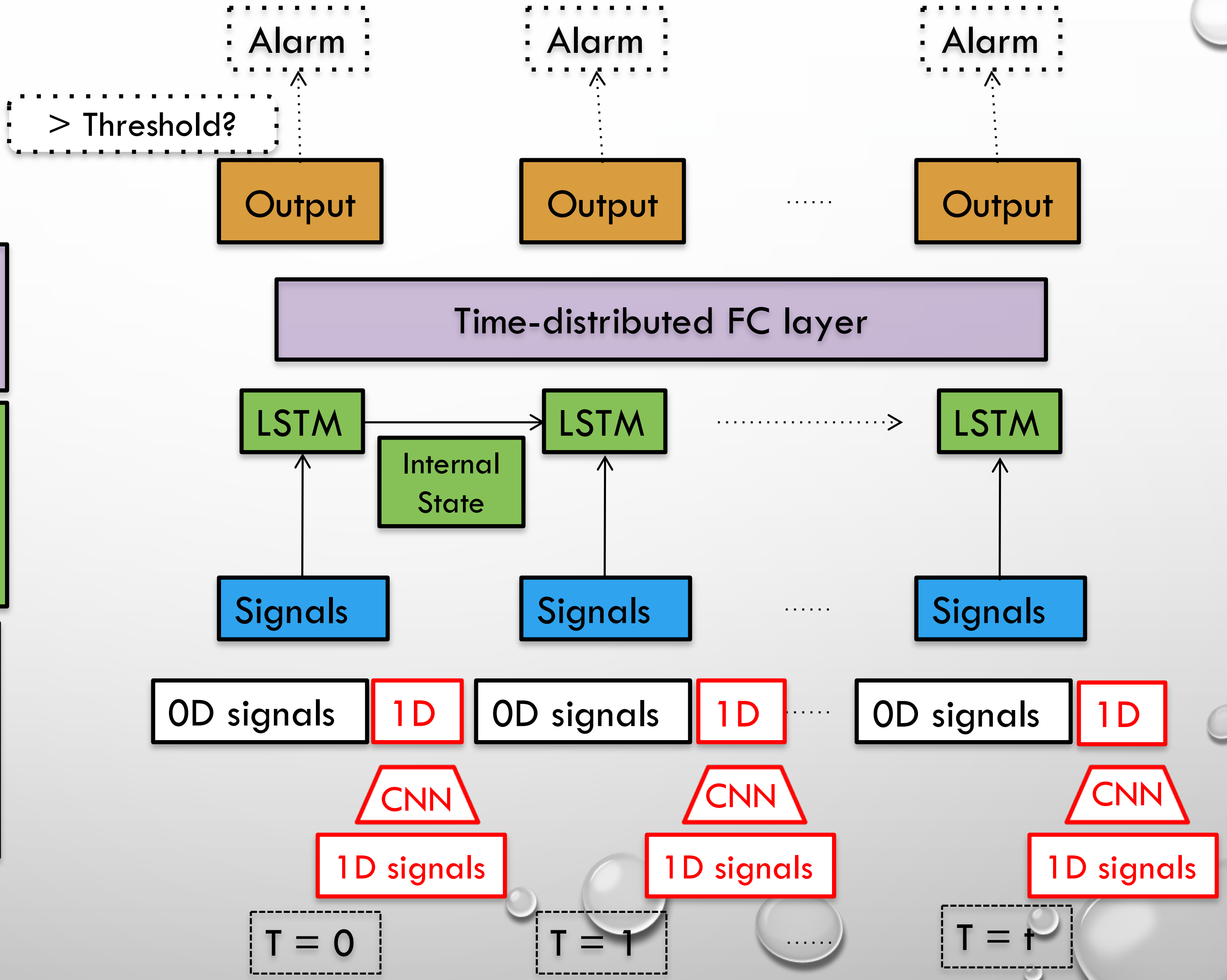
- apply to every temporal slice on LSTM output

RNN Architecture:

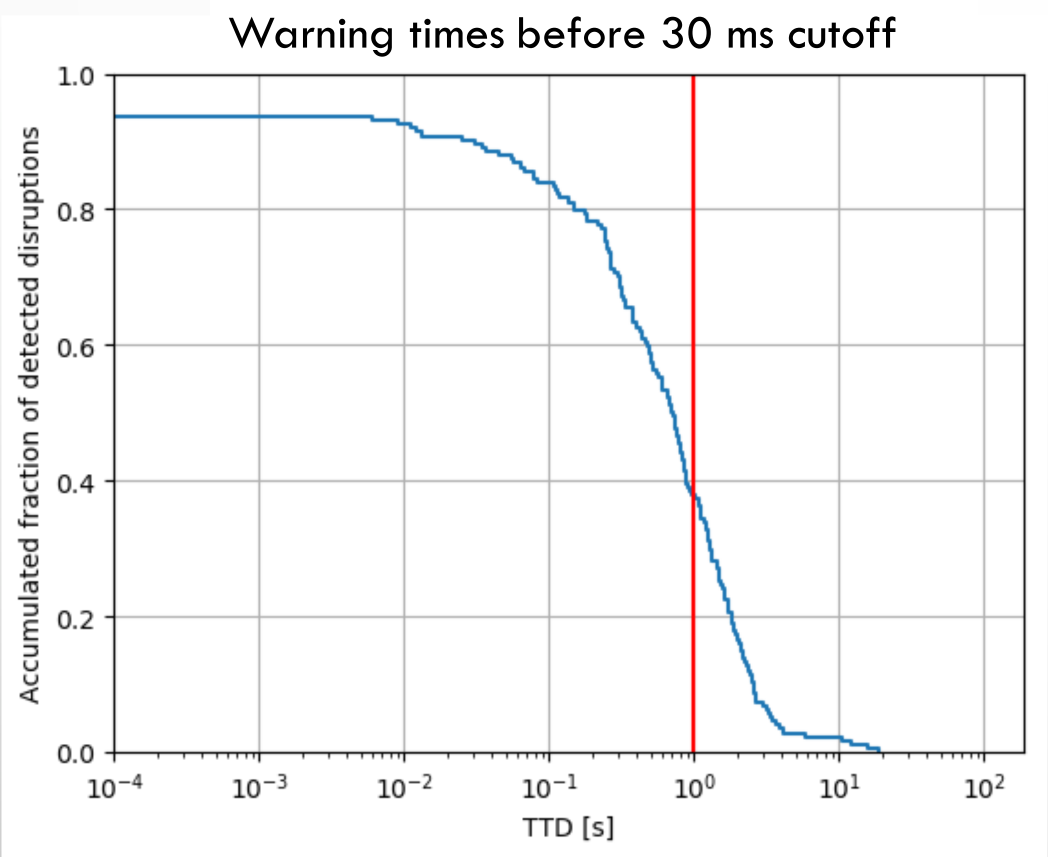
- LSTM, 3 layers
- 300 hidden units per cell
- Stateful, returns sequences

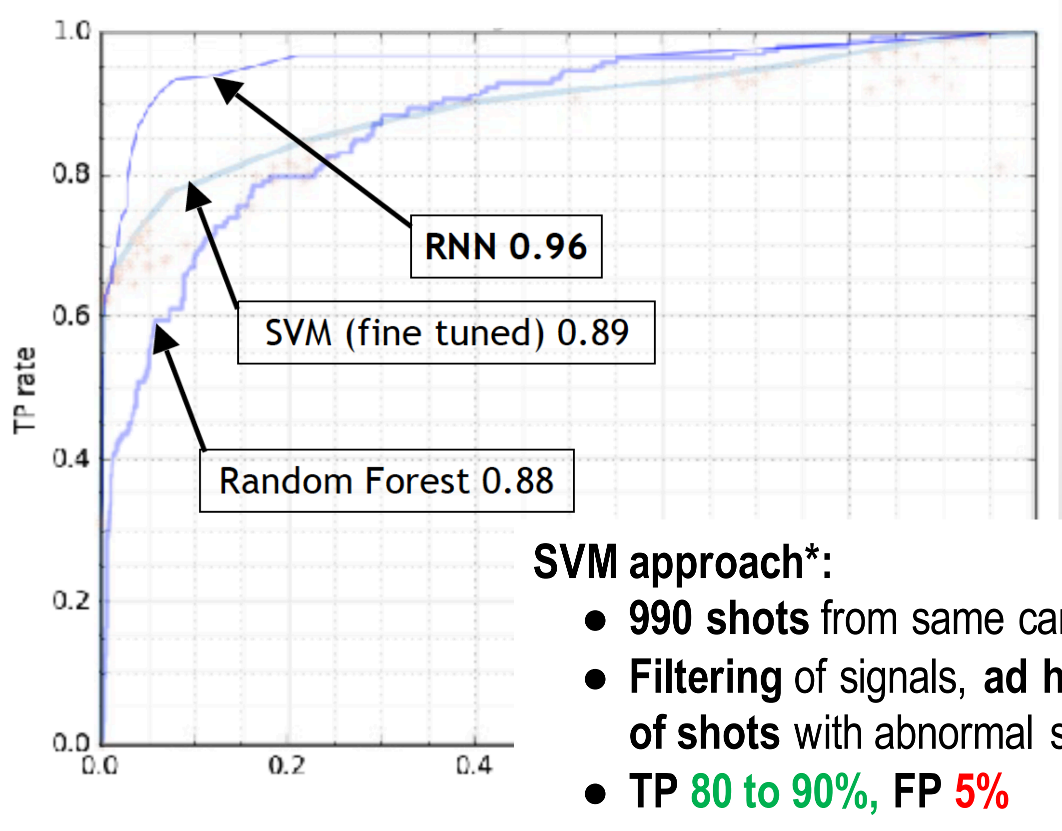
CNN architecture:

- Number of convolutional filters: 10
- Size of convolutional filters: 3
- Number of convolutional layers: 2
- Pool size: 2



JET ITER-like wall performance @30 ms before disruption





SVM approach*:

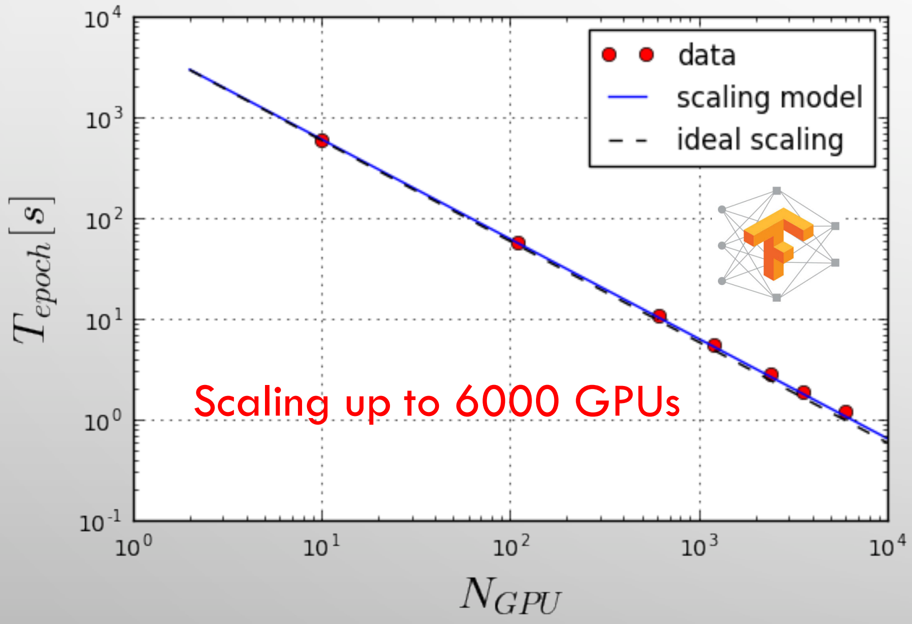
- **990 shots** from same campaigns
- **Filtering** of signals, **ad hoc removal of shots** with abnormal signals
- **TP** 80 to 90%, **FP** 5%

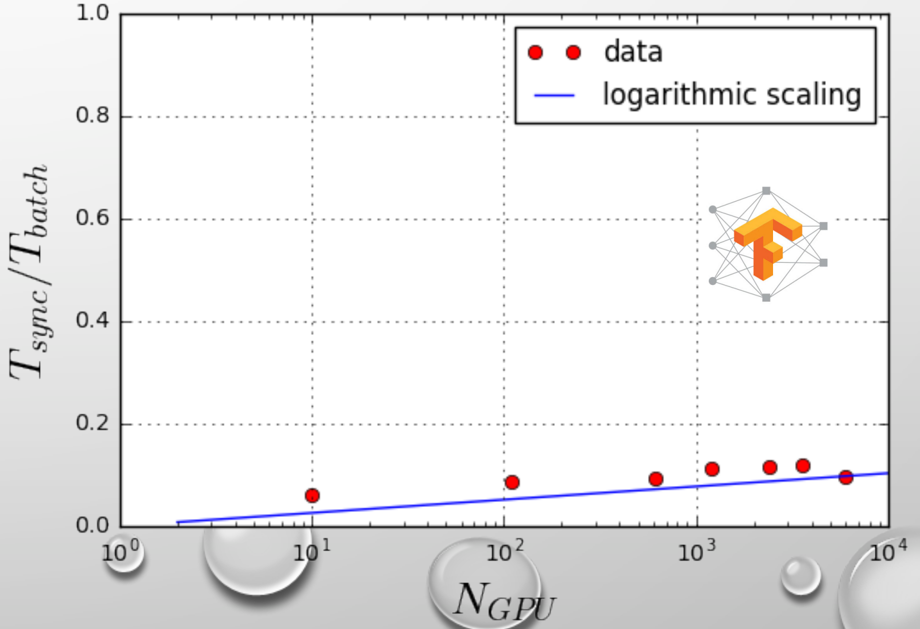
*Vega, Jesús, et al. "Results of the JET real-time disruption predictor in the ITER-like wall campaigns." *Fusion Engineering and Design* 88.6 (2013): 1228-1231.

FRNN scaling results on GPU: Part 2

- Tests on OLCF Titan CRAY supercomputer
  - *OLCF Director's Discretionary Award: Scaling Studies on Titan*
  - Thousands of Tesla K20 GPUs
  - Tensorflow+MPI (using Singularity containers), CUDA7.5, CuDNN 5.1

- We applied for Google Cloud TPUs, but have not heard back yet

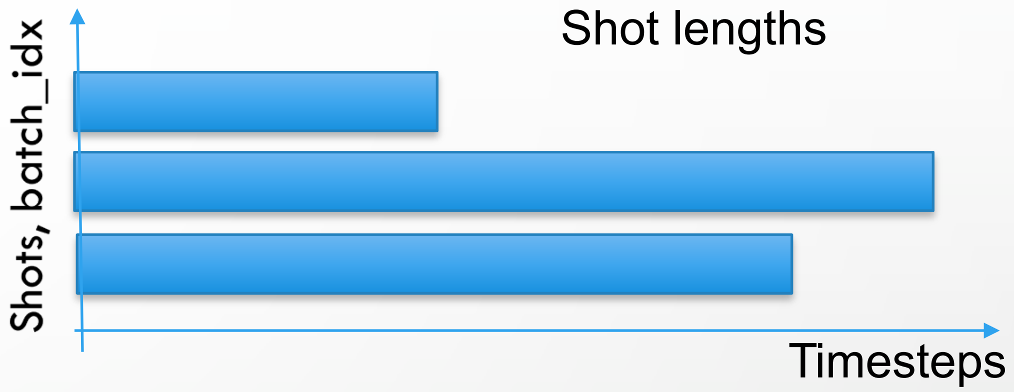




BACKUP

Challenges of stateful LSTM training, sequences of variable length

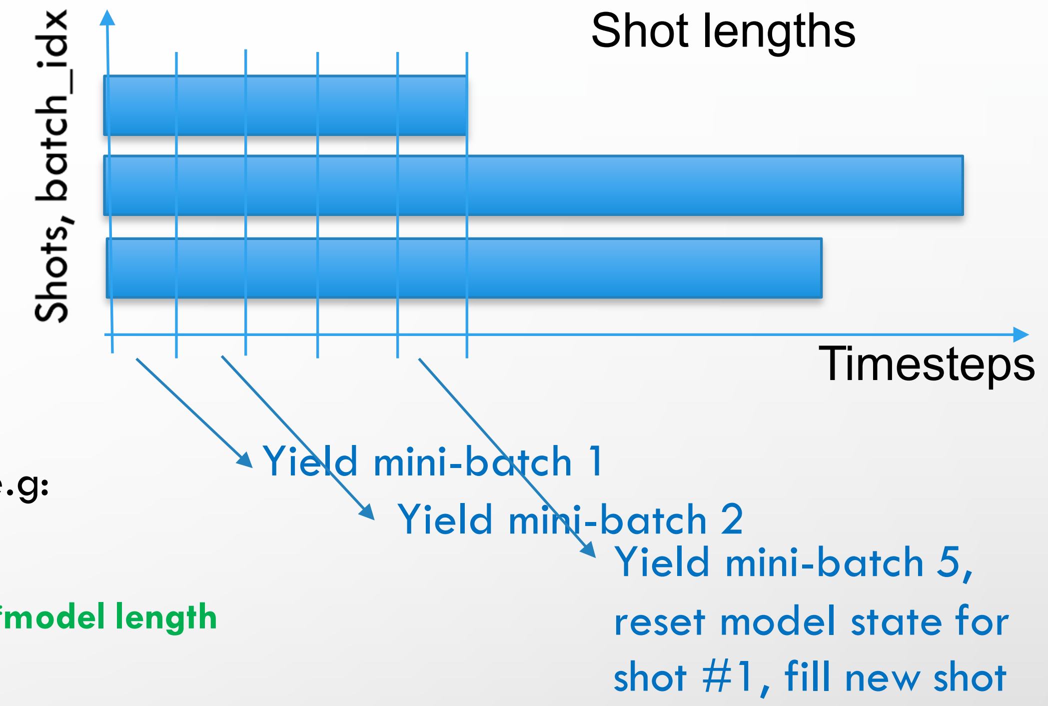
- Lengths of shots in e.g. JET data vary by orders of magnitude:
  - Minimum length: 1400
  - Mean length: ~27,000
  - Max: ~40,000 time-steps



- Zero-padding to the max length is not the best option with such spread in sequence lengths
- For a model to converge, the best approach is to feed subsequences of shot smaller length and do not reset states after each mini-batch
  - Training is stateful when the last state for each sample at a timestep ***i*** in a mini-batch will be used as initial state for the sample of timestep ***i*** in the following mini-batch
  - Reset states in the end of shot, individually
- The challenges is to implement a custom batch generator which would do that (see next slide)

Challenges of stateful LSTM training, sequences of variable lengths

- Implement a custom batch generator:
  - Takes a list of shots (for instance 2800 shots, each shot a time series of 1400-40000 timesteps). 9 scalar measurements at each time point
- Create Xbuff and Ybuff tensors each holding **batch_size** shots
  - Xbuff shape: **(batch_size, Maximum shot length, dimension of data)**
  - Ybuff shape: **(batch_size, Maximum shot length, 1)**
- For each shot adjust the length to be a multiplier of the LSTM model length, e.g:
  - Model length: 128 (hyper parameter, but generally << **shot length**)
  - Shot length: 25000 timesteps, adjusted shot length: **(Shot length//model length)*model length**
- Fill an array **end_indices**: which contains lengths of shots
- Create a **reset_batches** boolean array containing indicating whether a model states need to be reset (if current shot just ended)
- Each time batch generator yields a tensor of shape **(batch_size, model length, dim of data)**, re-adjusts the Xbuff and Ybuff shifting to the beginning of array by **model length**, decrements **end_indices** by **model length** and checks whether any of **end_indices** are less than zero (meaning we have hit end of shot for a shots at **batch_idx**)
- Once we hit the end of a shot, we do a partial batch reset, then fill in new shot at a **batch_idx**



BOOSTING: EPSILON BOOST

- BOOSTING IS AN ITERATIVE ALGORITHM TO REDUCE THE VARIANCE OF ENSEMBLE OF DECISION TREES (CAN BE APPLIED TO OTHER CLASSIFIERS AS WELL)
  - DECISION TREES ARE HIGH VARIANCE CLASSIFIERS
  - REWEIGHT MISCLASSIFIED EVENTS, REPEAT THE TRAINING ON THE WHOLE SAMPLE
- THE ALGORITHM:

- Initialize event weights: $W_i = \frac{1}{N}$

y_i – class labels

- Define index function: $T_m(x_i)$, +1 if the result of classification is correct, -1 otherwise
- Define loss function as $Err_m = \sum_{T_m(xi) \neq yi} W_i$ (Sum of weights for misclassified events for each tree m)
- Calculate score for each tree as: $B_m = A \bullet \log(\frac{1-Err_m}{Err_m})$
- Boost (or increase) weights $W_i \rightarrow W_i e^{B_m}$
- Renormalize all events W_i - $\rightarrow W_i \frac{}{\sum W_i}$
- Score by summing over trees, stop iteration once desired accuracy is reached