Deep learning in computer vision and natural language processing

Yifeng Tao
School of Computer Science
Carnegie Mellon University

Slides adapted from Matt Gormley, Russ Salakhutdinov

Review

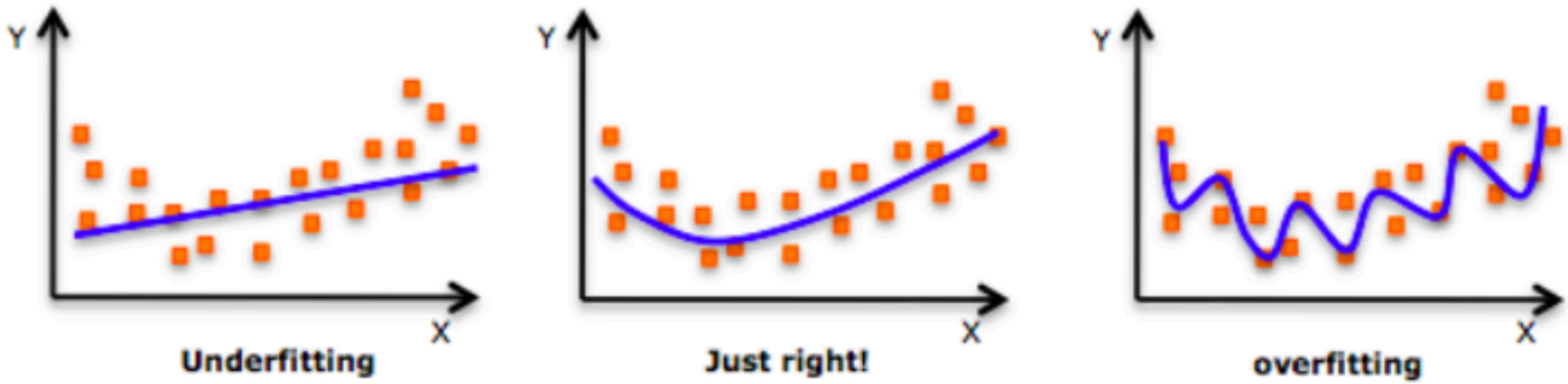
- Perceptron algorithm
- Multilayer perceptron and activation functions
- Backpropagation
- Momentum-based mini-batch gradient descent methods

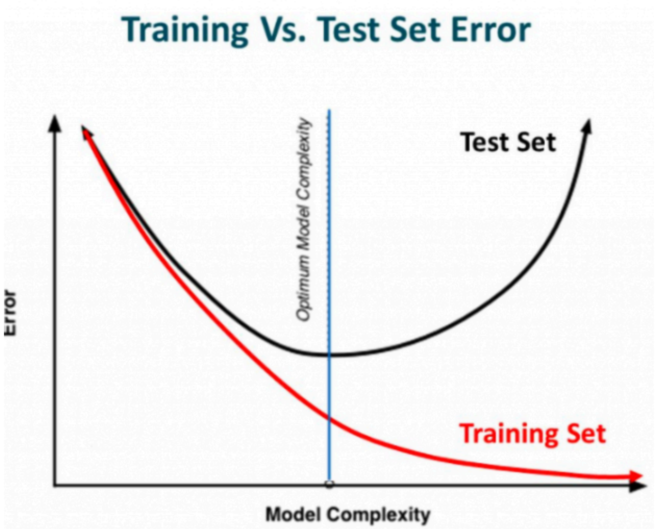
Outline

- Regularization in neural networks – methods to prevent overfitting
- Widely used deep learning architecture in practice
  - CNN
  - RNN

Overfitting

- The model tries to learn too well the noise in training samples





[Slide from https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/]

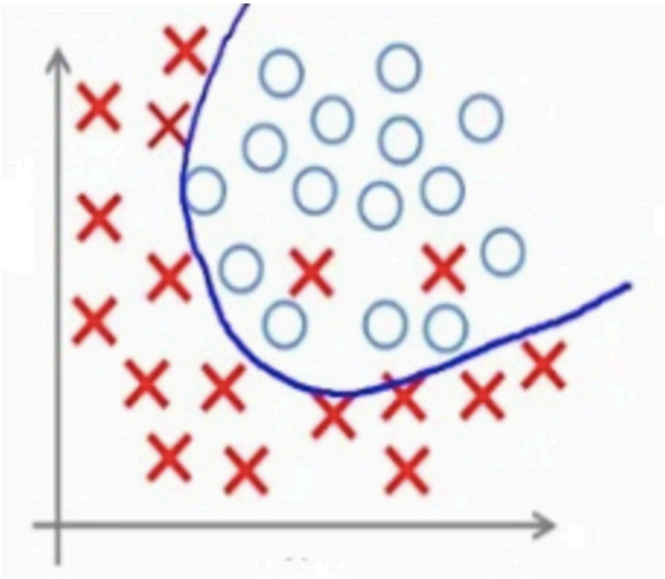
Model Selection

- Training Protocol:
  - Train your model on the Training Set $\mathcal{D}^{\text{train}}$
  - For model selection, use Validation Set $\mathcal{D}^{\text{valid}}$
    - Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
  - Estimate generalization performance using the Test Set $\mathcal{D}^{\text{test}}$
- Generalization is the behavior of the model on **unseen examples**.

[Slide from Russ Salakhutdinov et al.]

Regularization in Machine Learning

- Regularization penalizes the coefficients.
- In deep learning, it penalizes the weight matrices of the nodes.



Appropriate-fitting

[Slide from https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/]

Regularization in Deep Learning

- L2 & L1 regularization
- Dropout
- Data augmentation
- Early stopping
- Batch normalization

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T}\sum_t l(f(\mathbf{x}^{(t)};\boldsymbol{\theta}), y^{(t)}) + \lambda\Omega(\boldsymbol{\theta})$$

L2 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k ||\mathbf{W}^{(k)}||_F^2$$

L1 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

[Slide from Russ Salakhutdinov et al.]

Dropout

- Produces very good results and is the most frequently used regularization technique in deep learning.
- Can be thought of as an ensemble technique.
  - Use random binary masks $m^{(k)}$
    - layer pre-activation for k>0

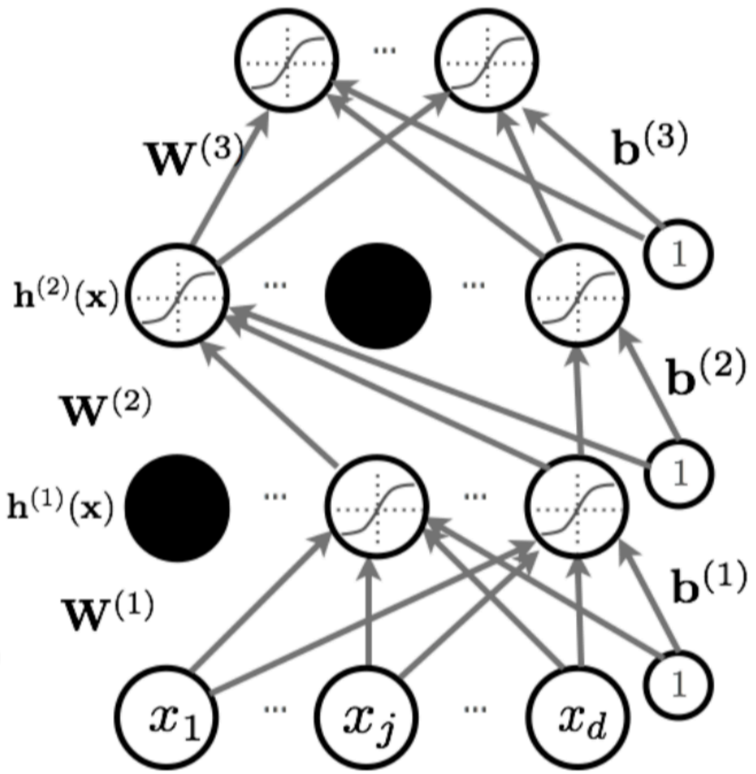
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

    - hidden layer activation (k=1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x})) \odot \mathrm{m}^{(k)}$$

    - Output activation (k=L+1)

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



[Slide from Russ Salakhutdinov et al.]

Dropout at Test Time

- At test time, we replace the masks by their expectation
  - ➢ This is simply the constant vector 0.5 if dropout probability is 0.5
  - ➢ For single hidden layer: equivalent to taking the geometric average of all neural networks, with all possible binary masks

- Can be combined with unsupervised pre-training

- Beats regular backpropagation on many datasets

- Ensemble: Can be viewed as a geometric average of exponential number of networks.

[Slide from Russ Salakhutdinov et al.]

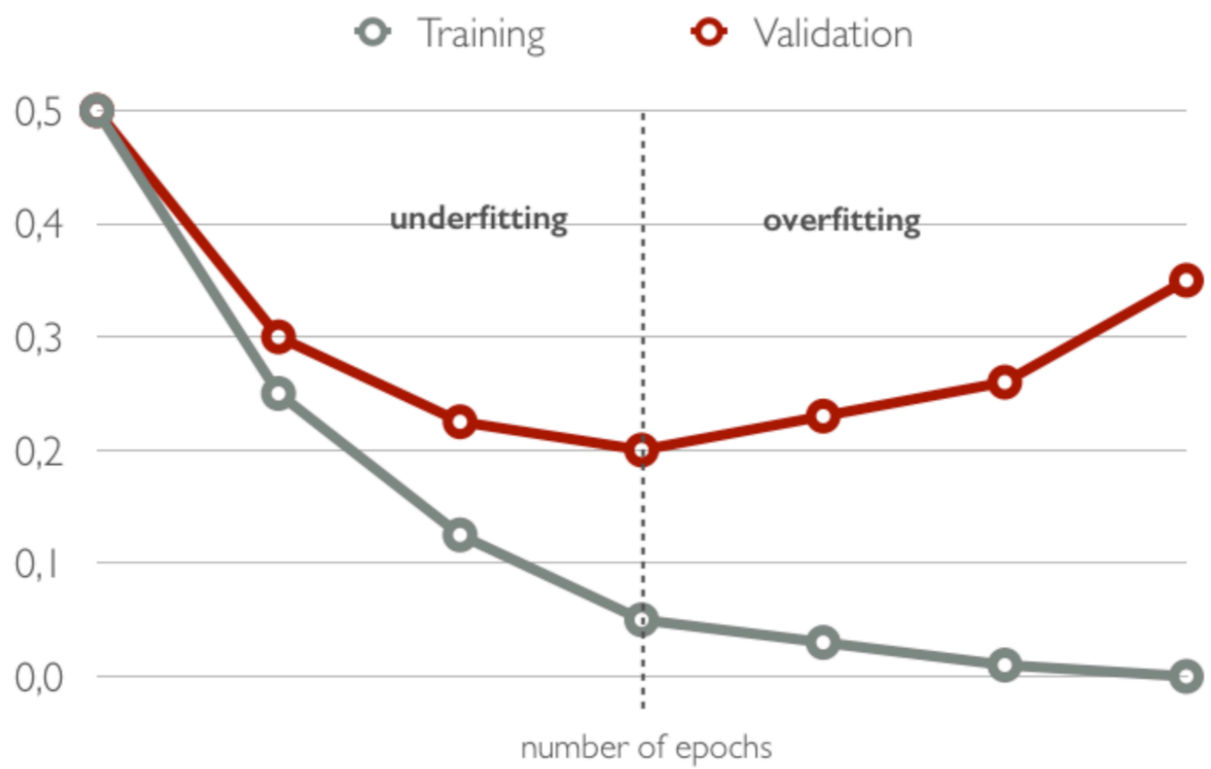
Data Augmentation

- Increase the size of the training data
- It can be considered as a mandatory trick to improve predictions

[Slide from https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/]

Early Stop

- To select the number of epochs, stop training when validation set error increases (with some look ahead)



[Slide from https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/]

Batch Normalization

- Normalizing the inputs will speed up training (Lecun et al. 1998)
  - could normalization be useful at the level of the hidden layers?
- Batch normalization is an attempt to do that (Ioffe and Szegedy, 2015)
  - each unit's pre-activation is normalized (mean subtraction, stddev division)
  - during training, mean and stddev is computed for each minibatch
  - backpropagation takes into account the normalization
  - at test time, the global mean / stddev is used

[Slide from Russ Salakhutdinov et al.]

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$\boxed{y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)} \qquad \text{// scale and shift}$$

Learned linear transformation to adapt to non-linear activation function (γ and β are trained)

[Slide from Russ Salakhutdinov et al.]

Batch Normalization

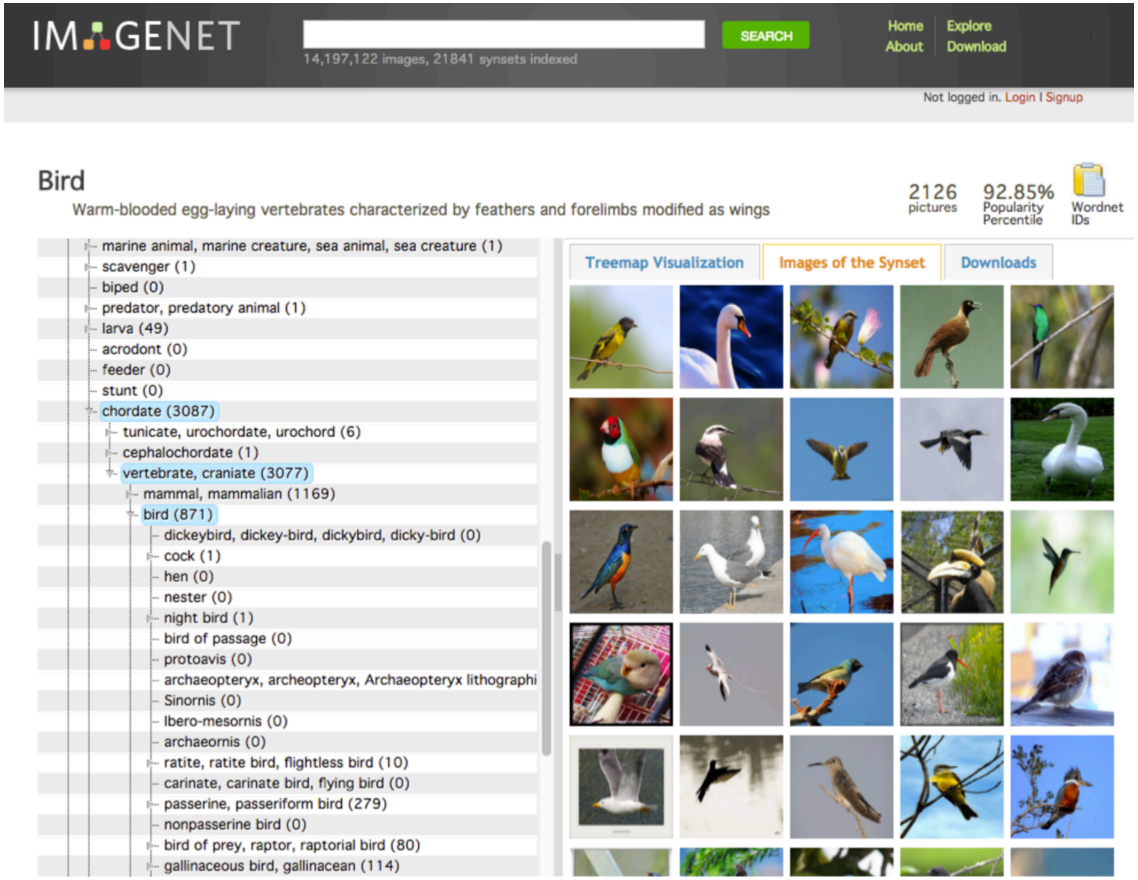
- Why normalize the pre-activation?
  - ➢ can help keep the pre-activation in a non-saturating regime (though the linear transform $y_i \leftarrow \gamma \widehat{x}_i + \beta$ could cancel this effect)

- Use the global mean and stddev at test time.
  - ➢ removes the stochasticity of the mean and stddev
  - ➢ requires a final phase where, from the first to the last hidden layer
    - propagate all training data to that layer
    - compute and store the global mean and stddev of each unit
  - ➢ for early stopping, could use a running average

[Slide from Russ Salakhutdinov et al.]

Computer Vision: Image Classification

- ImageNet LSVRC-2011 contest:
  - Dataset: 1.2 million labeled images, 1000 classes
  - Task: Given a new image, label it with the correct class



[Slide from Matt Gormley et al.]

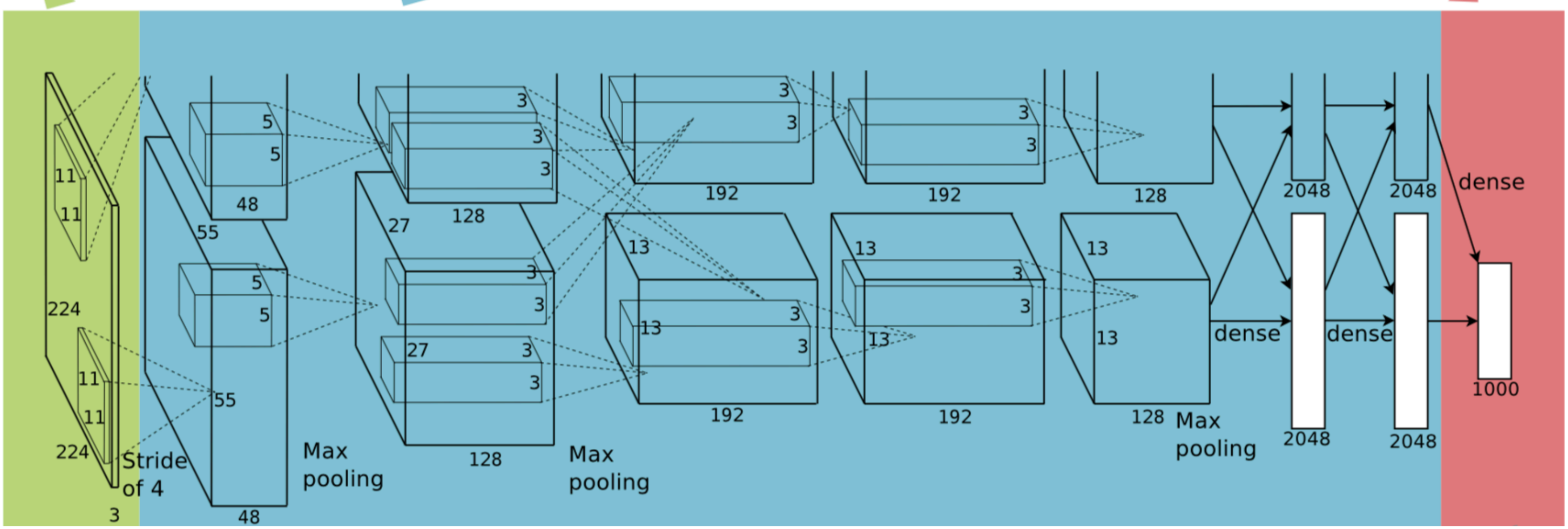
Computer Vision: Image Classification

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2012)
15.3% error on ImageNet LSVRC-2012 contest

Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

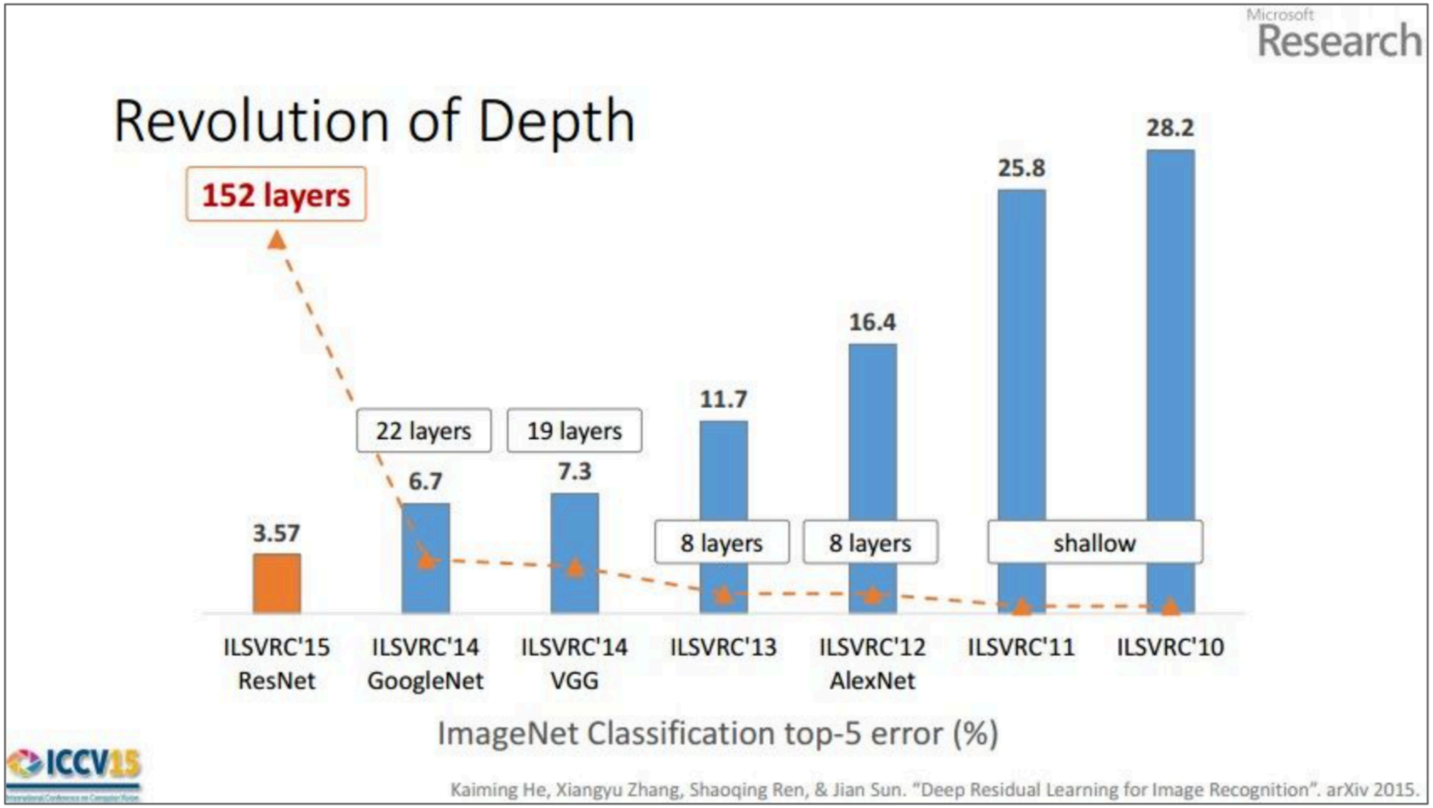
1000-way softmax



[Slide from Matt Gormley et al.]

CNNs for Image Recognition

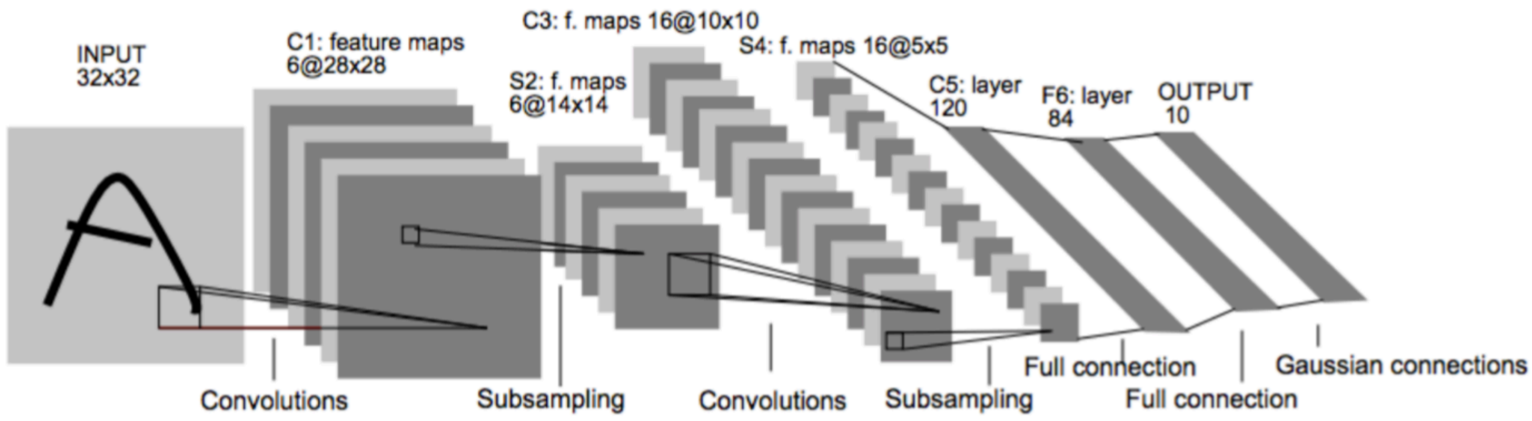
- Convolutional Neural Networks (CNNs)



[Slide from Matt Gormley et al.]

Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax
- These can be arranged into arbitrarily deep topologies
- Architecture #1: LeNet-5

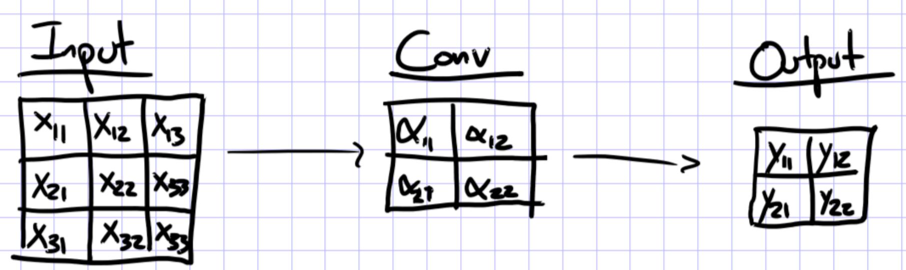


[Slide from Matt Gormley et al.]

What is a Convolution

- Basic idea:
  - Pick a 3x3 matrix F of weights
  - Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation
- Key point:
  - Different convolutions extract different low-level “features” of an image
  - All we need to vary to generate these different features is the weights of F
- A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Ex: 1 input channel, 1 output channel



$$y_{11} = \alpha_{11}x_{11} + \alpha_{12}x_{12} + \alpha_{21}x_{21} + \alpha_{22}x_{22} + \alpha_0$$
$$y_{12} = \alpha_{11}x_{12} + \alpha_{12}x_{13} + \alpha_{21}x_{22} + \alpha_{22}x_{23} + \alpha_0$$
$$y_{21} = \alpha_{11}x_{21} + \alpha_{12}x_{22} + \alpha_{21}x_{31} + \alpha_{22}x_{32} + \alpha_0$$
$$y_{22} = \alpha_{11}x_{22} + \alpha_{12}x_{23} + \alpha_{21}x_{32} + \alpha_{22}x_{33} + \alpha_0$$

[Slide from Matt Gormley et al.]

What is a Convolution

Input Image

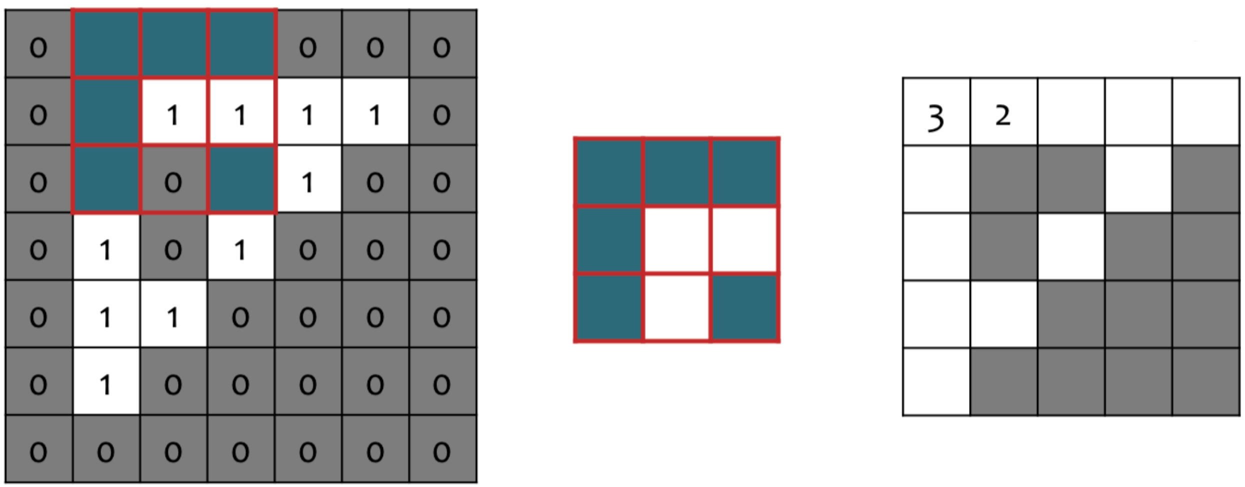
0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	1	1
0	1	0

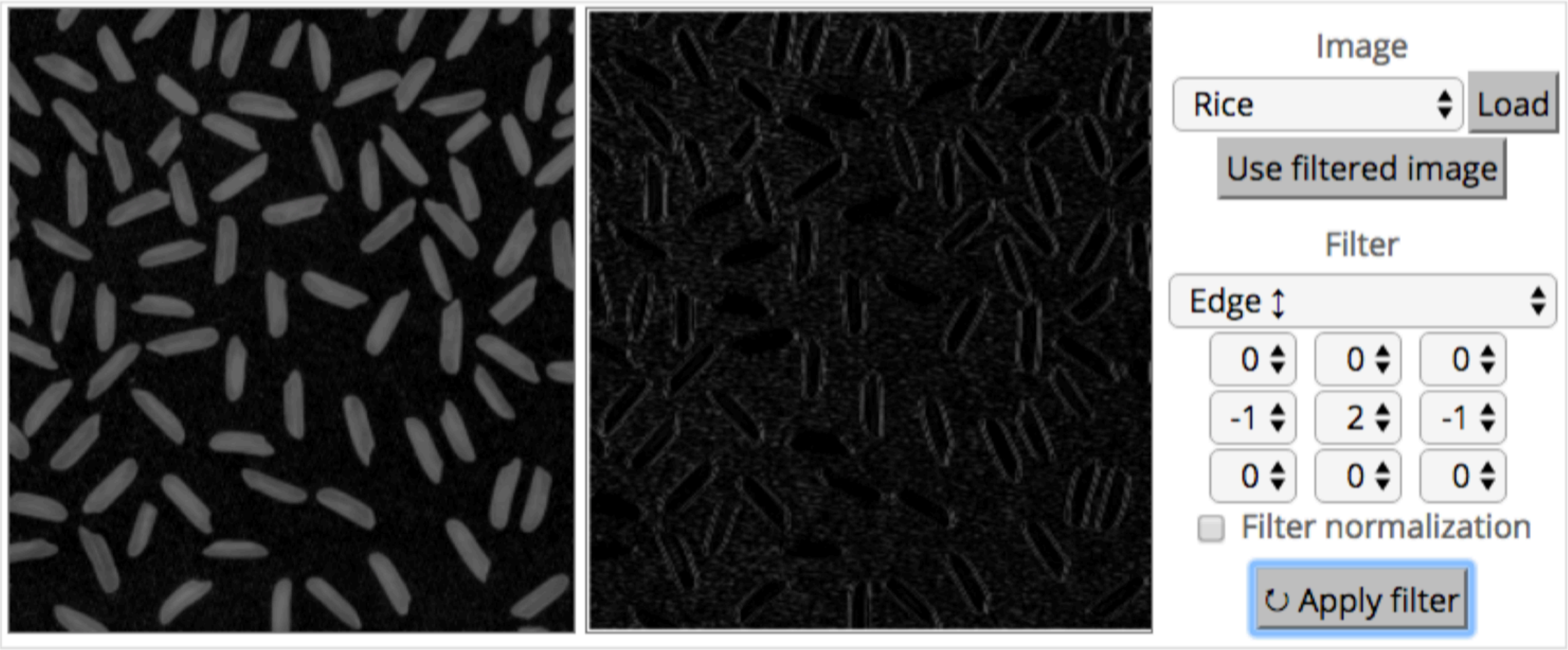
Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0



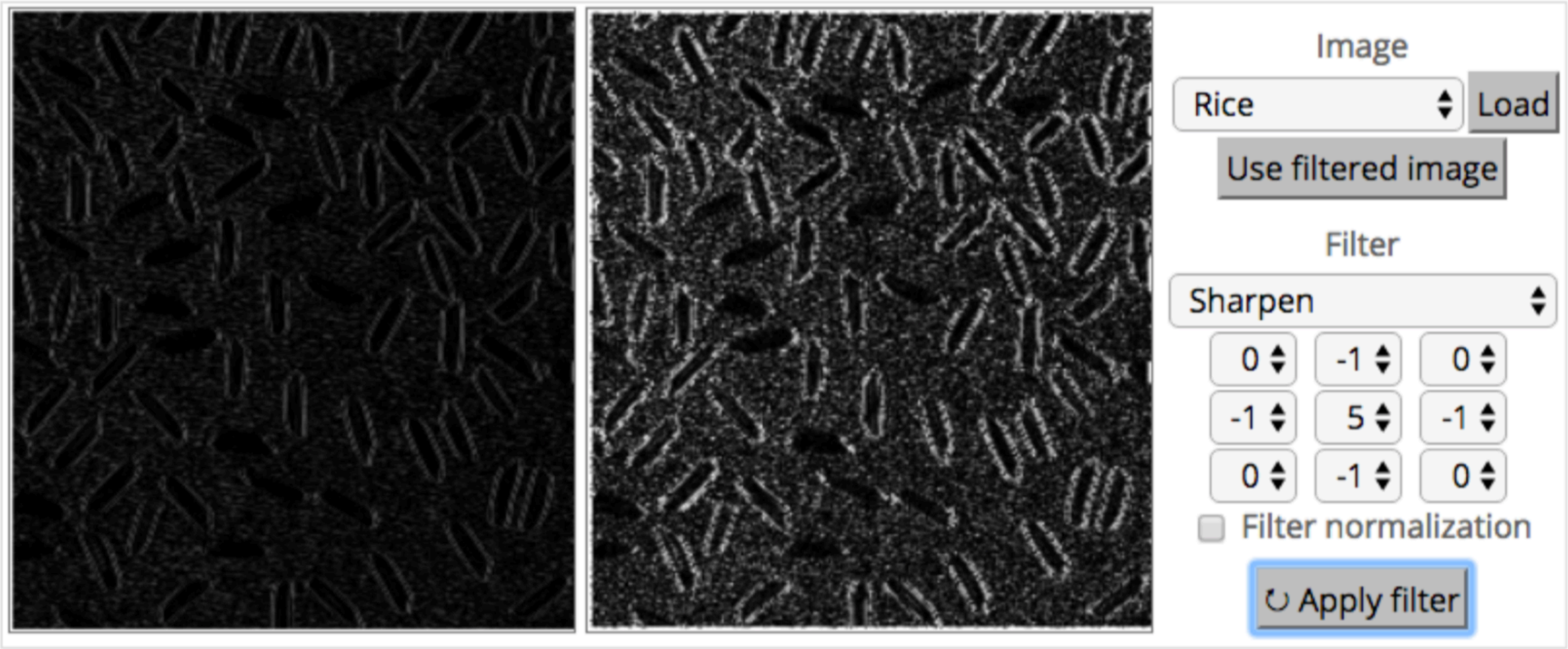
[Slide from Matt Gormley et al.]

What is a Convolution



[Slide from Matt Gormley et al.]

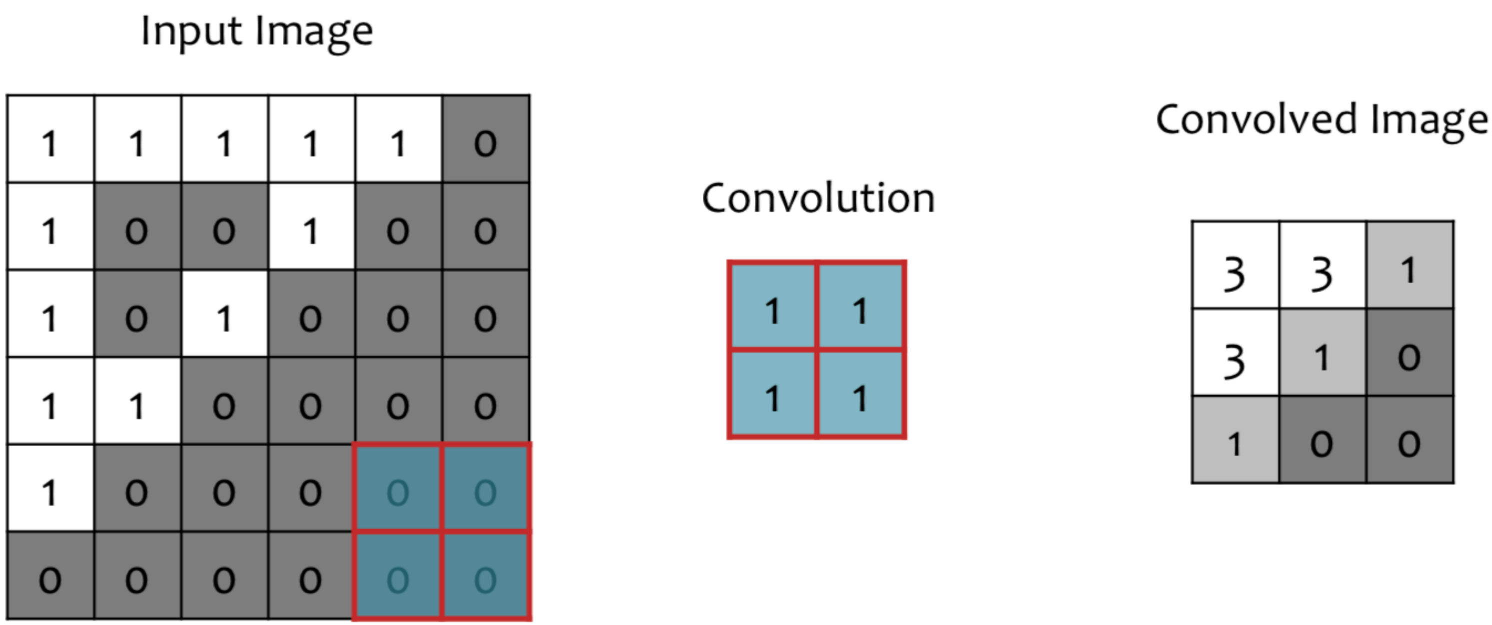
What is a Convolution



[Slide from Matt Gormley et al.]

Downsampling by Averaging

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output



[Slide from Matt Gormley et al.]

Downsampling by Max-Pooling

- Max-pooling is another (common) form of downsampling
- Instead of averaging, we take the max value within the same range as the equivalently-sized convolution
- The example below uses a stride of 2

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Max-pooling

$x_{i,j}$	$x_{i,j+1}$
$x_{i+1,j}$	$x_{i+1,j+1}$

$$y_{ij} = \max(x_{ij}, \\ x_{i,j+1}, \\ x_{i+1,j}, \\ x_{i+1,j+1})$$

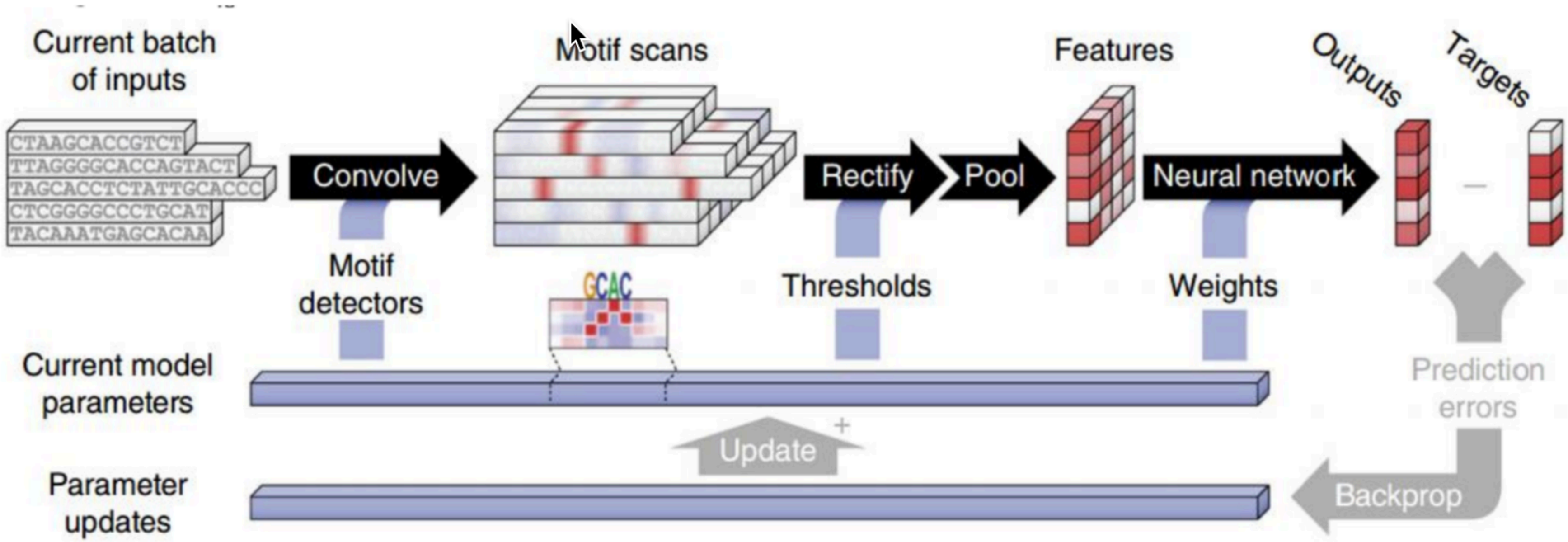
Max-Pooled Image

1	1	1
1	1	0
1	0	0

[Slide from Matt Gormley et al.]

CNN in protein-DNA binding

- Feature extractor for motifs

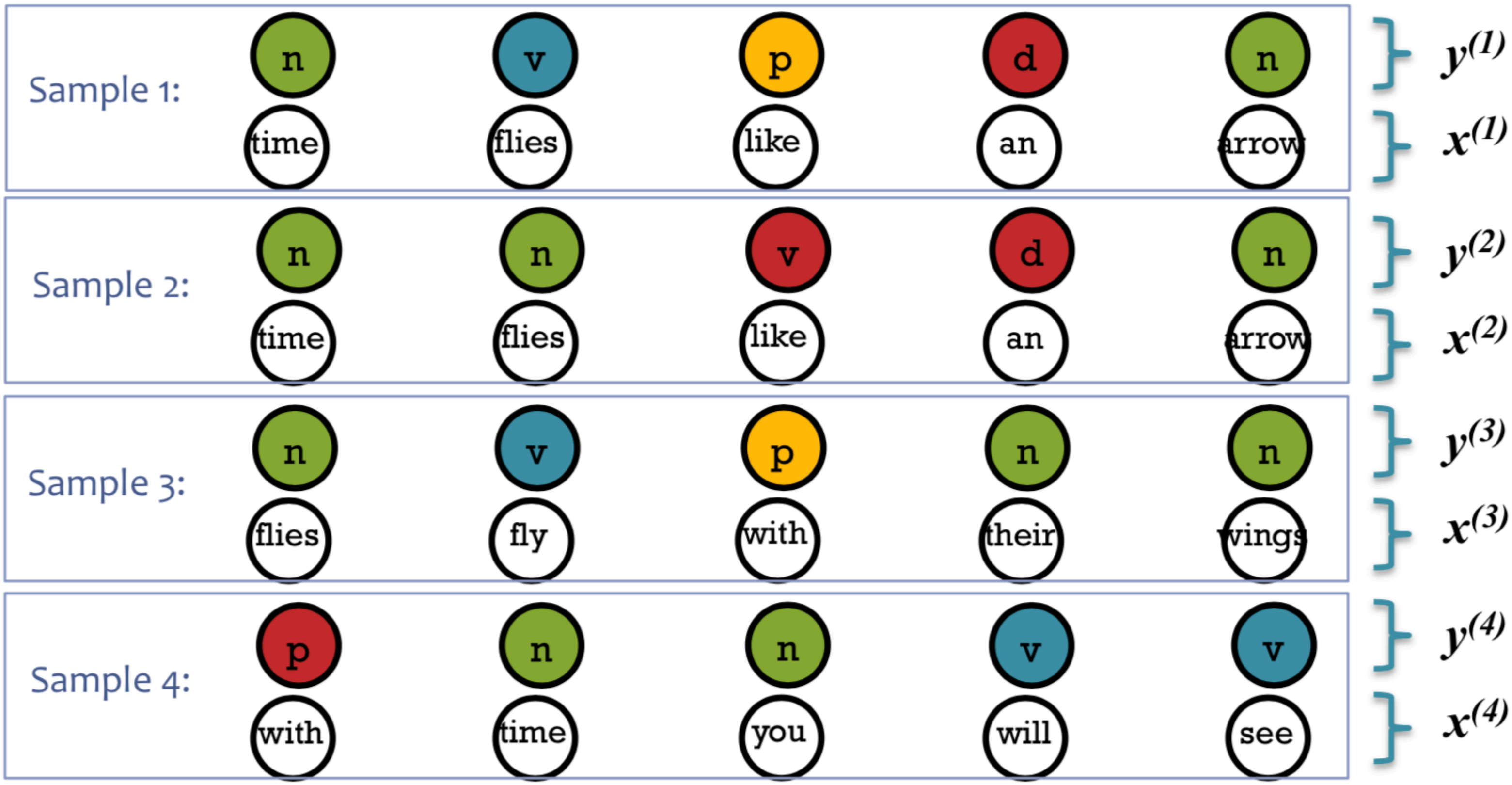


[Slide from Babak Alipanahi et al. 2015]

Recurrent Neural Networks

- Dataset for Supervised Part-of-Speech (POS) Tagging

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



[Slide from Matt Gormley et al.]

Recurrent Neural Networks

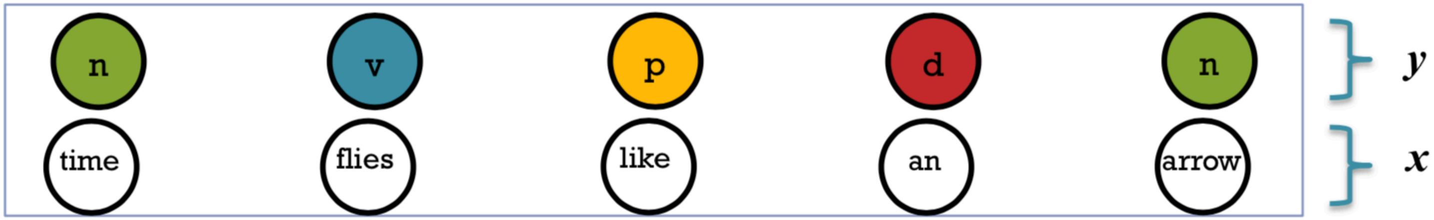
- Dataset for Supervised Handwriting Recognition

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$

[Slide from Matt Gormley et al.]

Time Series Data

- **Question 1**: How could we apply the neural networks we've seen so far (which expect fixed size input/output) to a prediction task with variable length input/output?
- **Question 2**: How could we incorporate context (e.g. words to the left/right, or tags to the left/right) into our solution?



[Slide from Matt Gormley et al.]

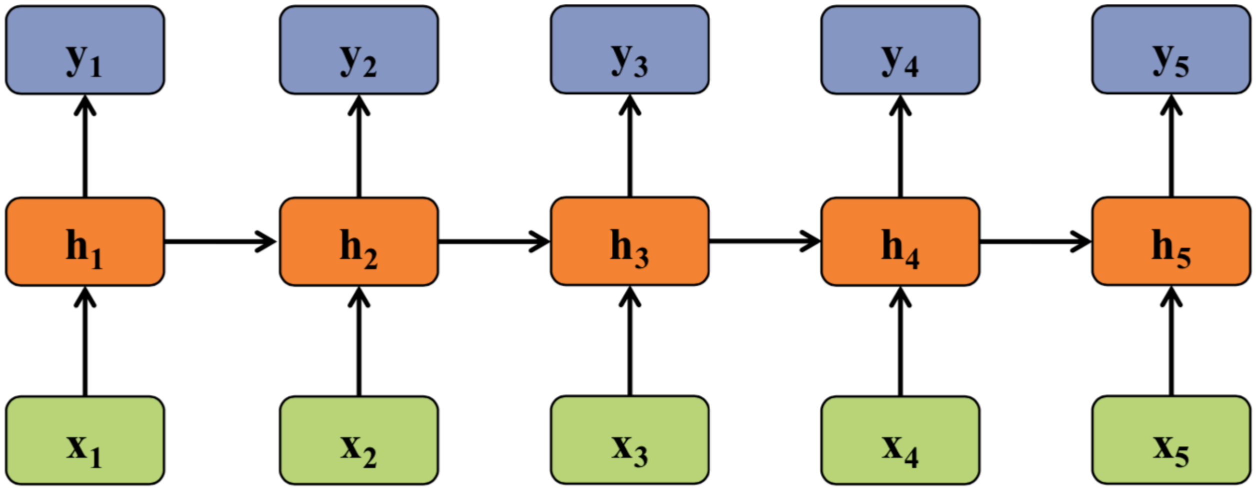
Recurrent Neural Networks (RNNs)

$$
\begin{aligned}
\text{inputs:}\ & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units:}\ & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs:}\ & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity:}\ & \mathcal{H}
\end{aligned}
$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



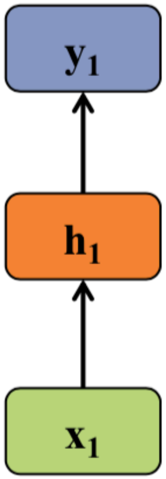
[Slide from Matt Gormley et al.]

Recurrent Neural Networks (RNNs)

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{hidden units: } \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$
$$y_t = W_{hy}h_t + b_y$$



- If $T=1$, then we have a standard feed-forward **neural net with one hidden layer**
- All of the deep nets from last lecture required **fixed size inputs/outputs**

[Slide from Matt Gormley et al.]

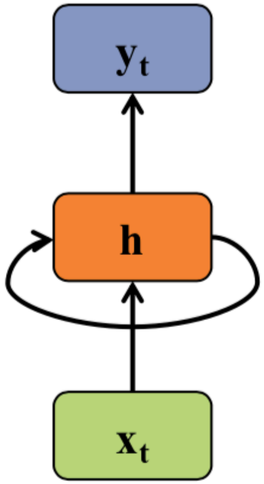
Recurrent Neural Networks (RNNs)

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{hidden units: } \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$
$$y_t = W_{hy}h_t + b_y$$

- By unrolling the RNN through time, we can **share parameters** and accommodate **arbitrary length** input/output pairs
- Applications: **time-series data** such as sentences, speech, stock-market, signal data, etc.



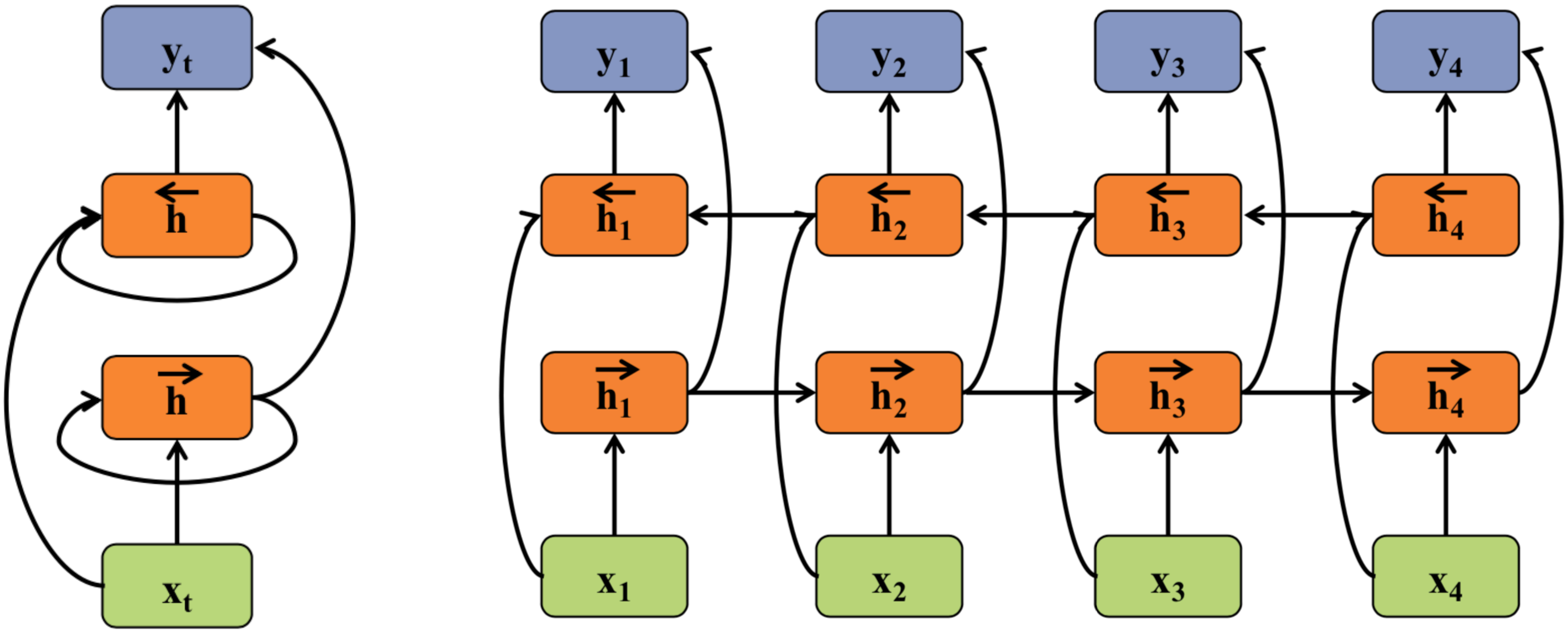
[Slide from Matt Gormley et al.]

Bidirectional RNN

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \dots, x_T), x_i \in \mathcal{R}^I$$
$$\text{hidden units: } \overrightarrow{\mathbf{h}} \text{ and } \overleftarrow{\mathbf{h}}$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \dots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

Recursive Definition:

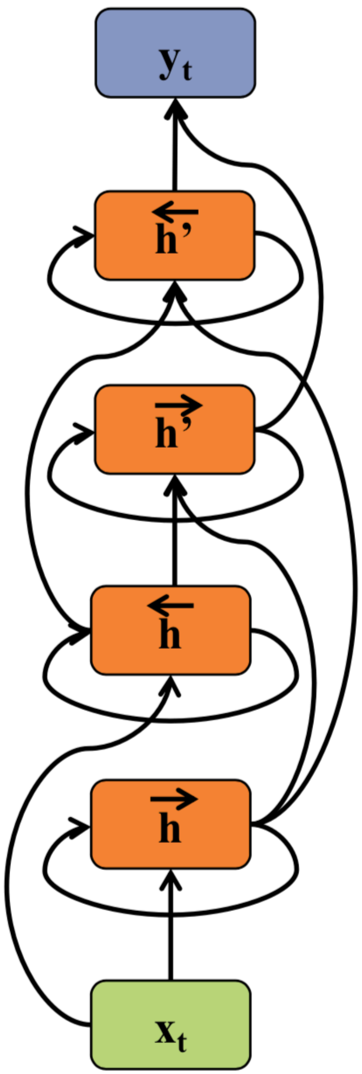
$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$
$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$
$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



[Slide from Matt Gormley et al.]

Deep Bidirectional RNNs

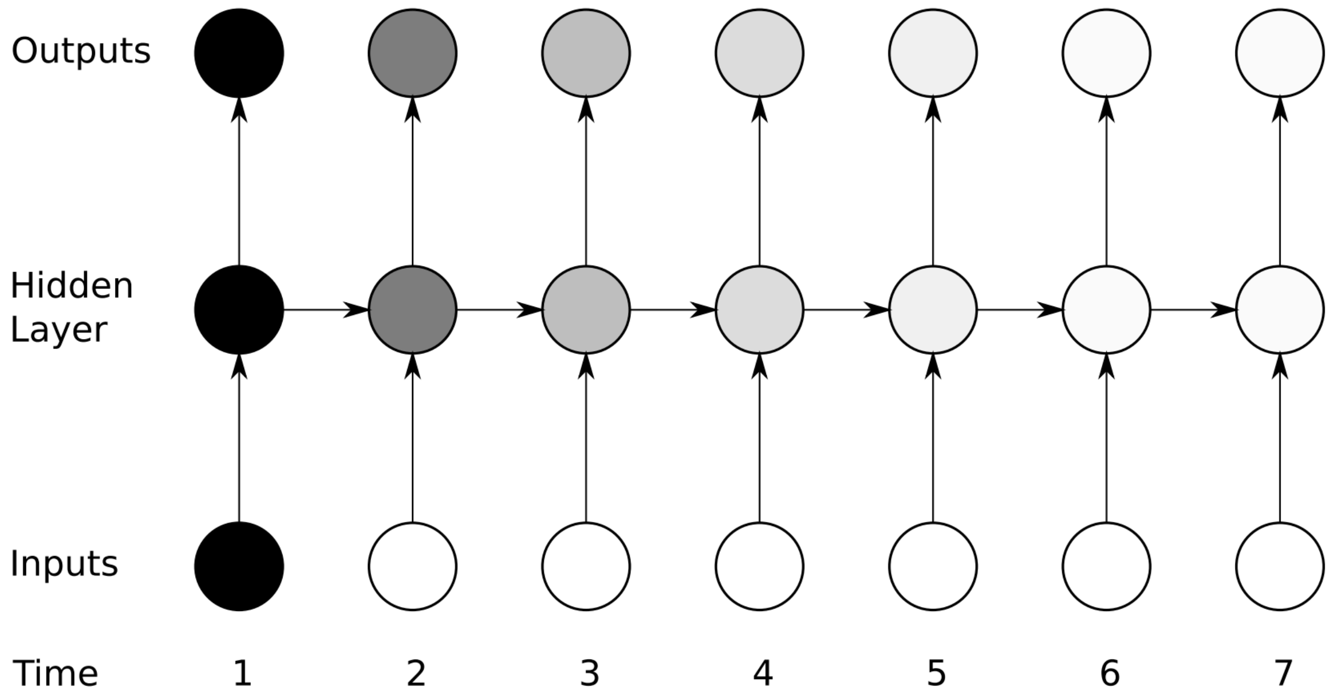
- Notice that the upper level hidden units have input from two previous layers (i.e. wider input)
- Likewise for the output layer



[Slide from Matt Gormley et al.]

Long Short-Term Memory (LSTM)

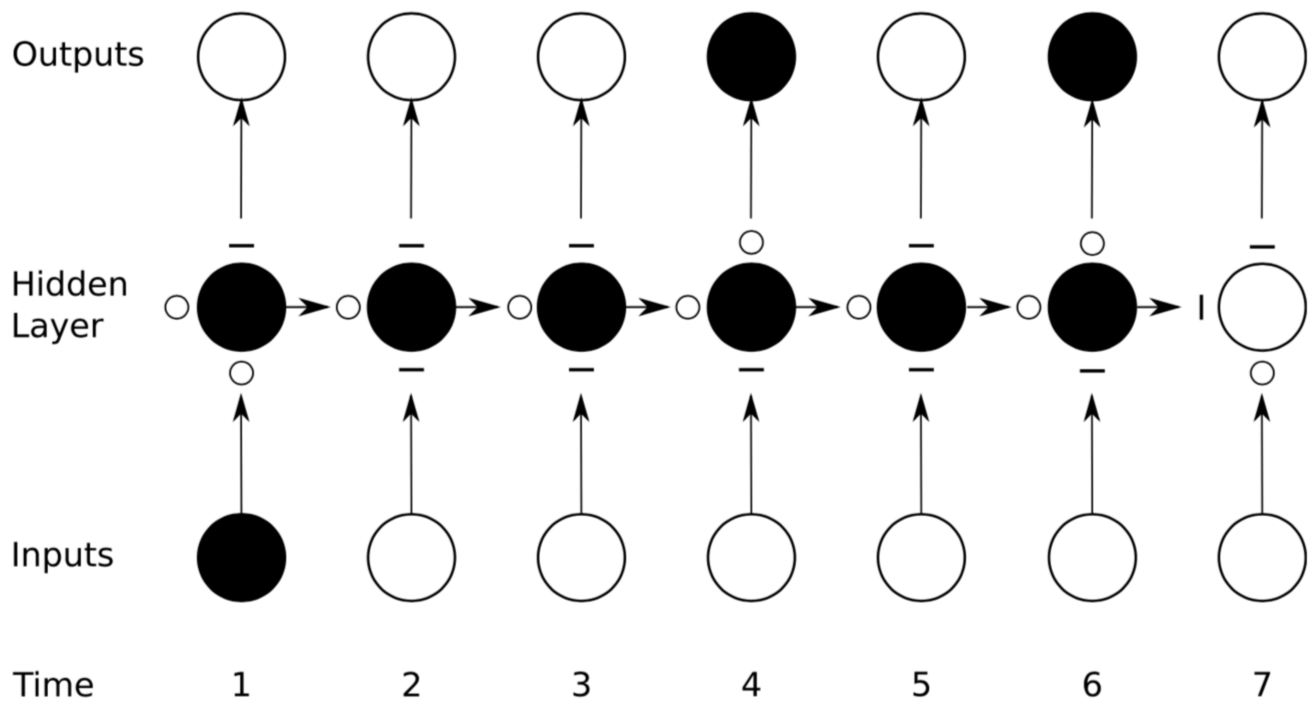
- Motivation:
  - Vanishing gradient problem for Standard RNNs
  - Figure shows sensitivity (darker = more sensitive) to the input at time t=1



[Slide from Matt Gormley et al.]

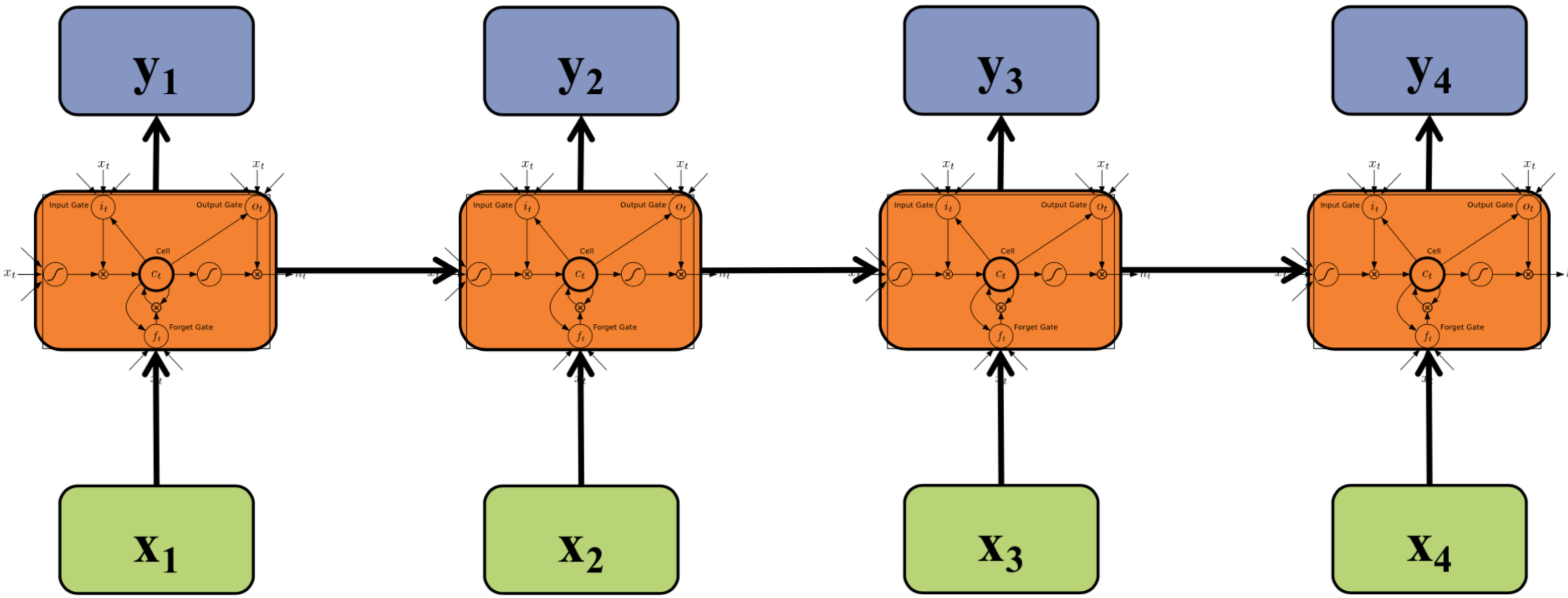
Long Short-Term Memory (LSTM)

- Motivation:
  - LSTM units have a rich internal structure
  - The various “gates” determine the propagation of information and can choose to “remember” or “forget” information



[Slide from Matt Gormley et al.]

Long Short-Term Memory (LSTM)

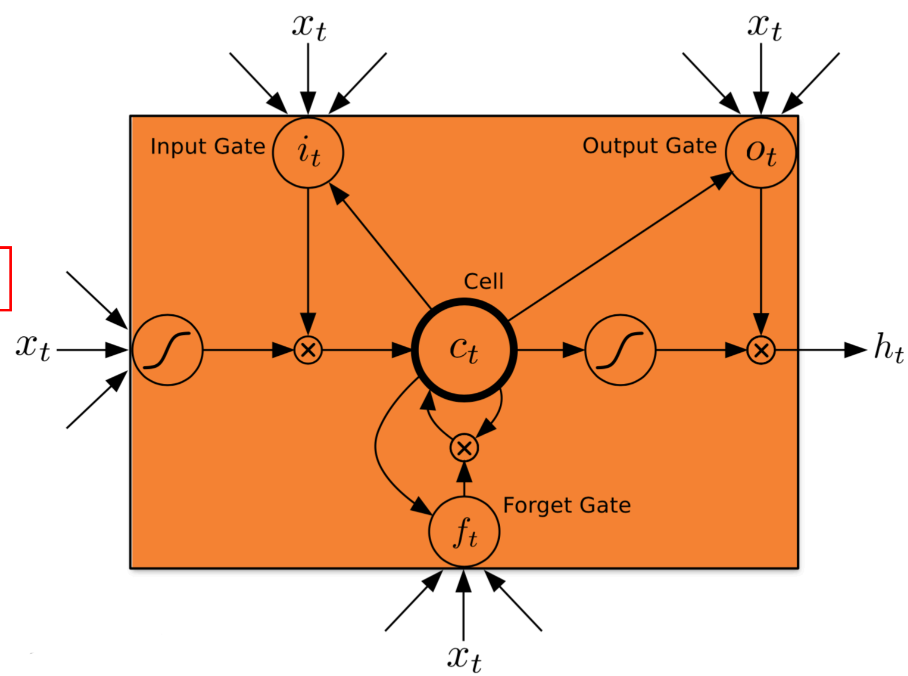


[Slide from Matt Gormley et al.]

Long Short-Term Memory (LSTM)

- **Input gate:** masks out the standard RNN inputs
- **Forget gate**: masks out the previous cell
- **Cell**: stores the input/forget mixture
- **Output gate:** masks out the values of the next hidden

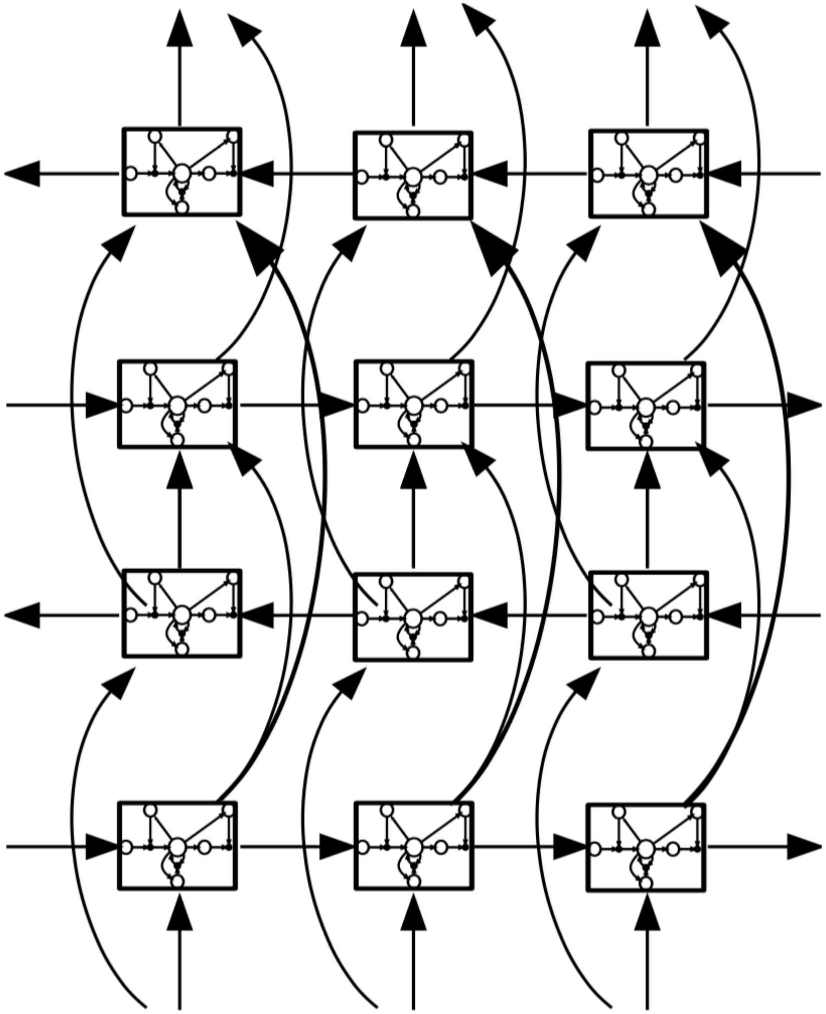
$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$\boxed{c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)}$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$
$$h_t = o_t \tanh(c_t)$$



[Slide from Matt Gormley et al.]

Deep Bidirectional LSTM (DBLSTM)

- How important is this particular architecture?
- Jozefowicz et al. (2015) evaluated 10,000 different LSTM-like architectures and found several variants that worked just as well on several tasks.



[Slide from Matt Gormley et al.]

Take home message

- Methods to prevent overfitting in deep learning
  - L2 & L1 regularization
  - Dropout
  - Data augmentation
  - Early stopping
  - Batch normalization
- CNN
  - Are used for all aspects of computer vision
  - Learn interpretable features at different levels of abstraction
  - Typically, consist of **convolution** layers, **pooling** layers, **nonlinearities**, and **fully connected** layers
- RNN
  - Applicable to sequential tasks
  - Learn context features for time series data
  - Vanishing gradients are still a problem – but LSTM units can help

References

- Matt Gormley. 10601 Introduction to Machine Learning: http://www.cs.cmu.edu/~mgormley/courses/10601/index.html
- Barnabás Póczos, Maria-Florina Balcan, Russ Salakhutdinov. 10715 Advanced Introduction to Machine Learning: https://sites.google.com/site/10715advancedmlintro2017f/lectures