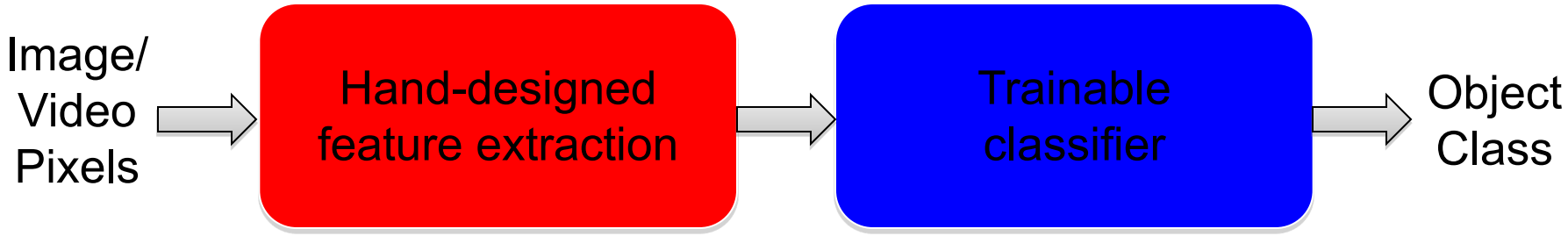
More about regularization and architectures in DNNs

Announcements

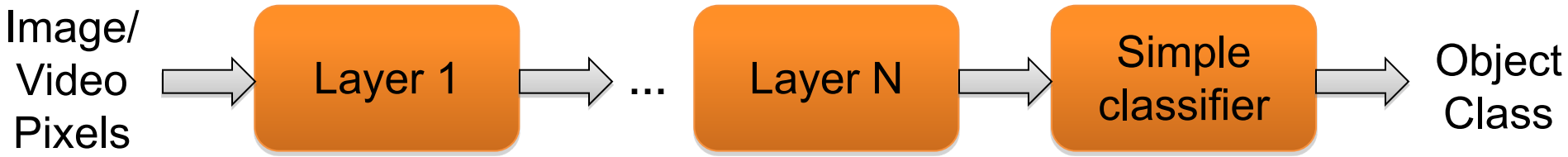
- A2 grades and comments posted
  - The three agents that we used for grading are hosted on sharks.luddy.indiana.edu, on ports 4996, 4997 and 4998 so you test your agents directly if you want.
- Optional A4 due on Sunday.
- Let us know if there is any outstanding regrade request
- Bump in points for A0 readme in the process

“Shallow” vs. “deep” learning

“Shallow” architecture



Deep learning: “Deep” architecture



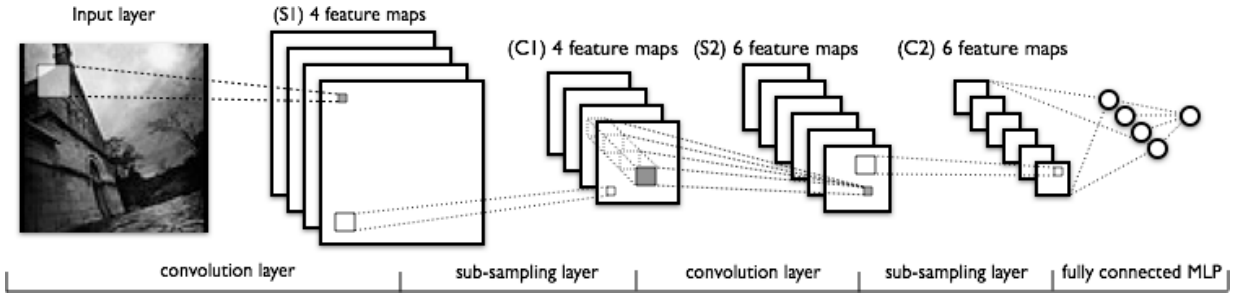
Slide credit: Rob Fergus

Regularization and optimization in Neural Networks

- Regularization and optimization in Neural Networks
  - Parameter sharing: e.g. Convolutional Neural Networks (CNNs)
  - Weight decay
  - Dropout
  - Early stopping
  - Change of learning rate during learning
  - Adding momentum
  - Dataset augmentation
  - Activation function selection
- Based on:
  - Deep learning book (http://www.deeplearningbook.org/), chapter 9 (CNN)

Convolutional Neural Networks

- Three principles:
  - local receptive fields
  - weight sharing
  - subsampling
- Multiple layers of Convolution, Activation, and Pooling may be used in a CNN
- These layers act as feature extraction, to find useful features from the input
- Generally, a final Fully Connected layer is added via a MLP for classification or regression purposes



Other optimization and regularization techniqeues

Weight decay

Weight decay:

- Penalizes complexity and prevent overfitting

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

Weight decay

Weight decay

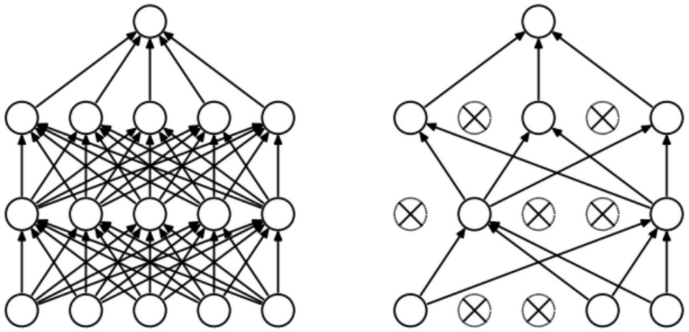
- Penalizes complexity and prevent overfitting

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2$$

Dropout

Dropout

- Dropping out units with (both hidden and visible) in a neural network. There is some probability that a unit will be dropped out (ignored in both forward and backward pass) at each training iteration.



(a) Standard Neural Net (b) After applying dropout.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Early stopping

Early stopping

- Stopping the training of a neural network once the model performance stops improving on a hold out validation dataset.

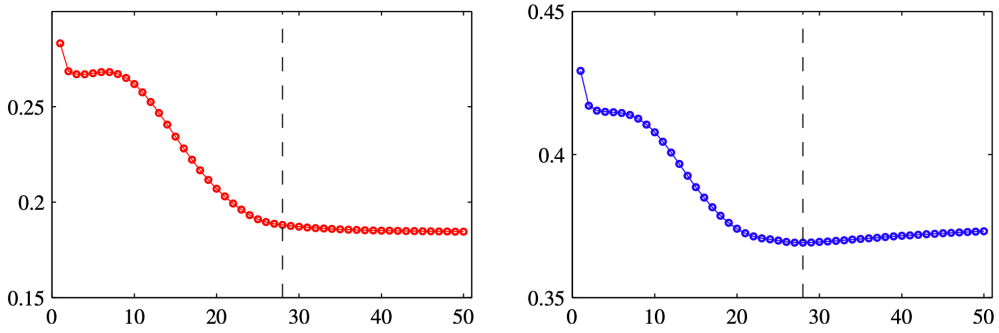


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

From Bishop's Machine Learning book, chapter 5 (Neural networks); 5.5 (Regularization)

Change learning rate during learning

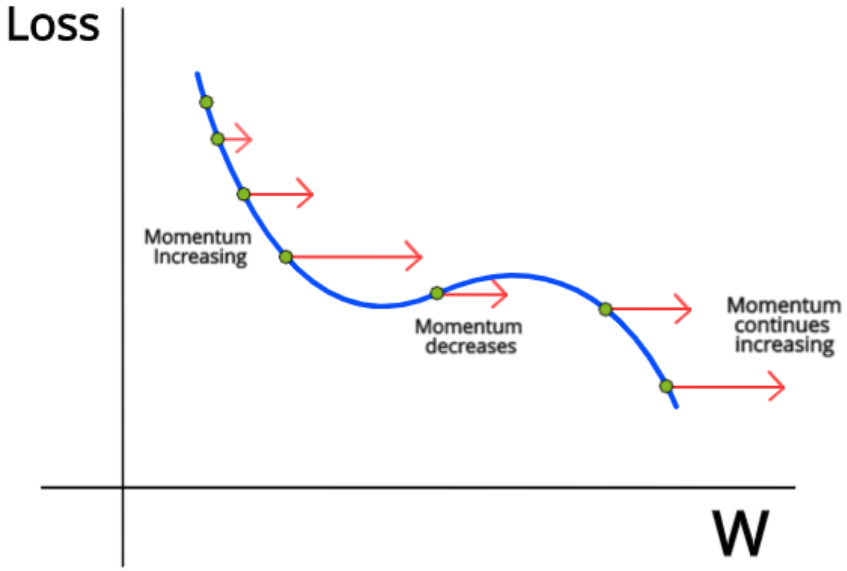
Change learning rate during learning:

- Make η large at the beginning of learning and decrease it later
- Simulated annealing: cleverly adjust learning rate using physics-inspired parameter called temperature

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

Momentum

Momentum: a way to escape small gradients



$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment — learning rate — weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

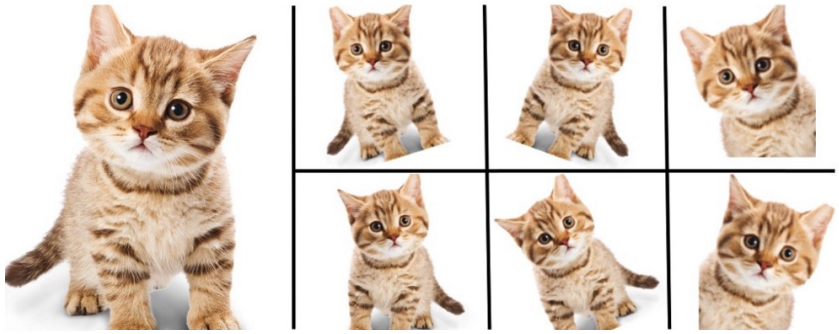
momentum factor — weight increment, previous iteration

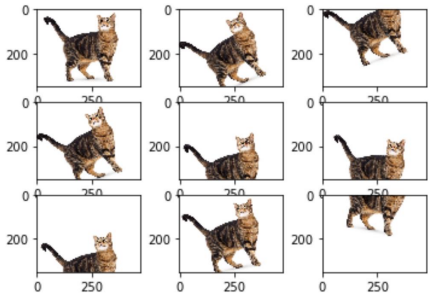
https://deeplearningdemystified.com/article/fdl-4

Dataset augmentation

Dataset augmentation

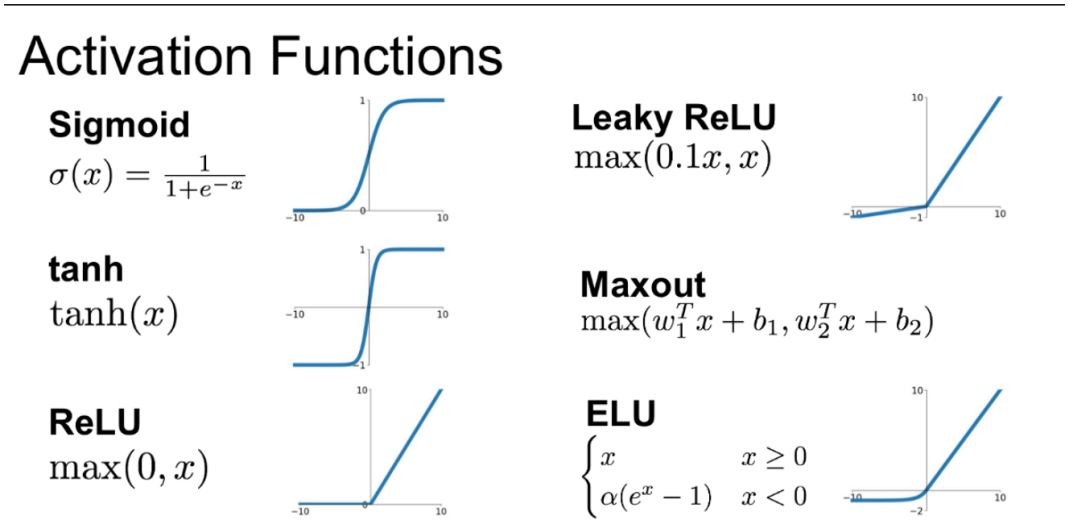
- E.g. adding noise to the input, applying spatial transformations to the input





Activation function selection

In deep (many layers) networks, to avoid small gradient we can use activations functions such as ReLU, which have a large gradient for a wide range of values.



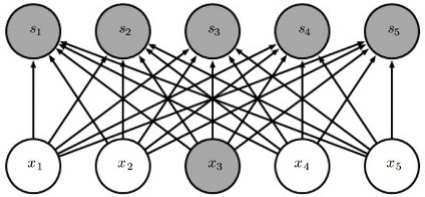
Recurrent neural networks: motivation

Consider time-series data **x** (x_1 is the first time point, x_2 is the second time point…)

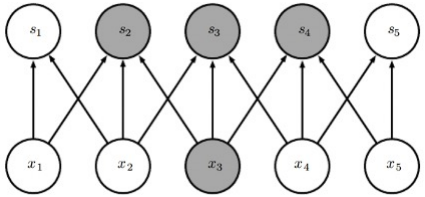
For long-time sequences, dense networks require a large number of parameters (all to all connections).

Convolutional networks share weights. This reduces the number of parameters, but can only capture local temporal relationships (within the width of the filter).

Dense



CNN



RNNs will not be in the exam

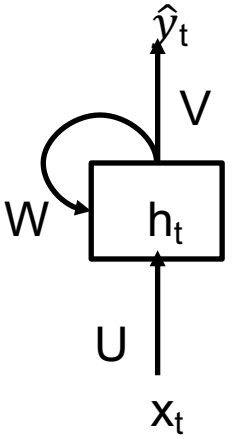
Introduction to RNNs

Recurrent neural networks or RNNs (Rumelhart et al., 1986) are a family of neural networks for processing sequential data.

Similar to CNNs, they use **parameter sharing (weight sharing)** – this makes it possible to extend and apply the model to examples of different forms (different lengths) and generalize across them.

A traditional fully connected feedforward network would have separate parameters for each input feature, so it would need to learn all of the rules of the sequence (e.g. language) separately at each position in the sequence.

A recurrent neural network shares the same weights across several time steps.



$$h_t = g(Wh_{t-1} + Ux_t) = g(z_t)$$

$$\hat{y}_t = g^*(Vh_t)$$

Training RNNs

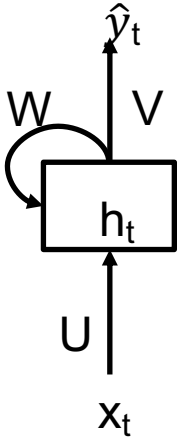
The total loss for a given sequence of **x** values paired with a sequence of **y** values is the sum of the losses over all the time steps.

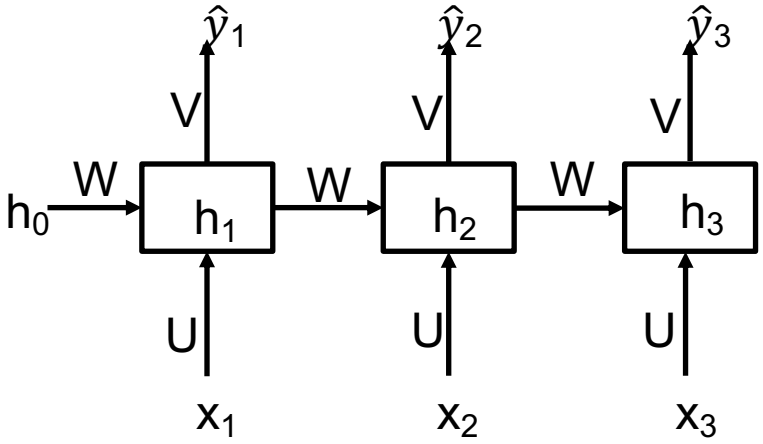
We need to compute the gradient of the error with respect to all of the weights and then use any gradient-based techniques to train the network.

The back-propagation algorithm applied to the unrolled graph is called **back-propagation through time.**

$$h_t = g(Wh_{t-1} + Ux_t) = g(z_t)$$

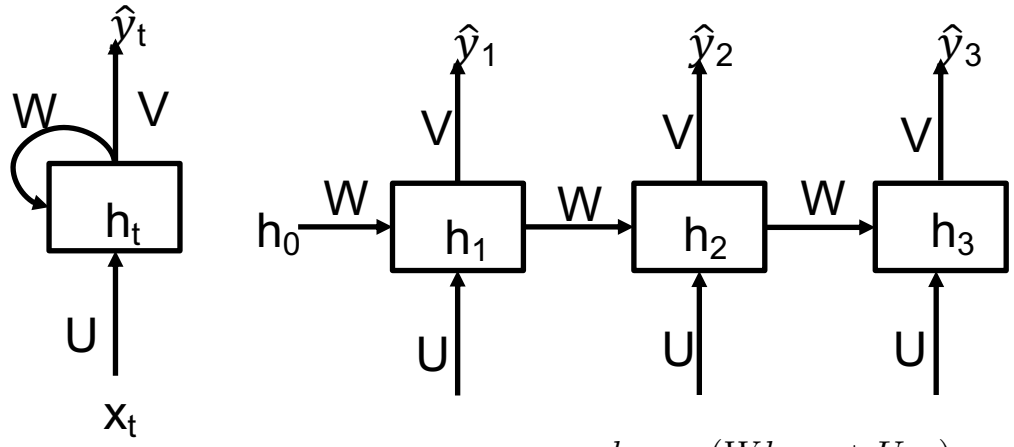
$$\hat{y}_t = g^*(Vh_t)$$





"Unfolded in time " RNN

Computing the gradient in an RNN using back-propagation through time



Product rule:

$$\frac{d}{dx}[f(x)g(x)] = f(x)g'(x) + f'(x)g(x)$$

$$h_t = g(Wh_{t-1} + Ux_t) = g(z_t)$$

$$\hat{y}_t = g^*(Vh_t)$$

We assume g^* is identity function: $\hat{y}_t = Vh_t$.

$$L = \sum_t L_t$$

$$L_3 = \frac{1}{2}(y_3 - \hat{y}_3)^2$$

$$\frac{\partial L_3}{\partial V} = \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = -(y_3 - \hat{y}_3)h_3$$

$$\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial W} = -(y_3 - \hat{y}_3)V\frac{\partial g(z_3)}{\partial W} = -(y_3 - \hat{y}_3)V\frac{\partial g(z_3)}{\partial z_3}(h_2 + W\frac{\partial h_2}{\partial W})$$

$$\frac{\partial L_3}{\partial U} = \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial U} = -(y_3 - \hat{y}_3)V\frac{\partial g(z_3)}{\partial U} = -(y_3 - \hat{y}_3)V\frac{\partial g(z_3)}{\partial z_3}(W\frac{\partial h_2}{\partial U} + x_3)$$

The Challenge of Long-Term Dependencies: vanishing and exploding gradient

The basic problem is that gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization), making it hard to learn log-term dependencies.

For simplicity, assume that activation functions are linear and that we are unfolding RNN over T steps, then we have:

$$\frac{\partial h_T}{\partial h_1} = \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \frac{\partial h_{T-2}}{\partial h_{T-3}} \cdots \frac{\partial h_2}{\partial h_1} = W^{T-1}$$

- So the gradient can easily explode or vanish if W is not very close to 1.
- Good resources for more details on math behind vanishing and exploding gradient:
  - https://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-gradient-problems/
  - http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf

Solutions for vanishing/exploding gradient problem: Gated RNNs

Gated RNNs:

- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

Gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode.

Example: If a sequence is made of sub-sequences and we want a leaky unit to accumulate evidence inside each sub-subsequence, we need a mechanism to forget the old state by setting it to zero.

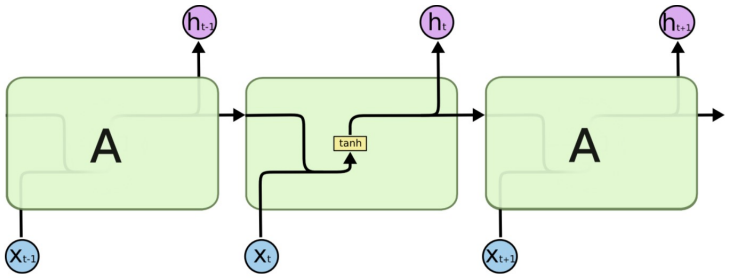
Instead of manually deciding when to clear the state, we want the neural network to learn to decide when to do it. This is what gated RNNs do.

This is related to my own research, see here for more resources

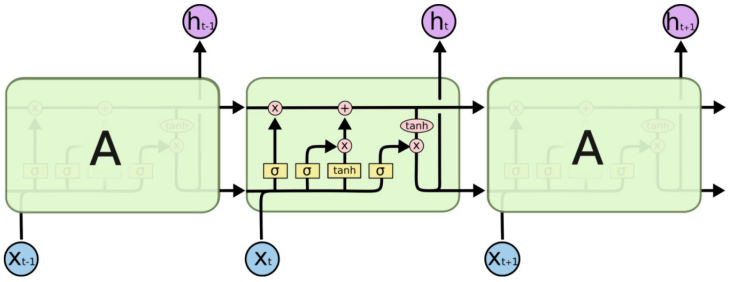
http://vision.soic.indiana.edu/ai-talk-series/

http://homes.luddy.indiana.edu/ztiganj/index.html

Long Short-Term Memory (LSTM)



The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural Network References

For additional examples and references, check out the following:

- The Data Science Blog - https://ujjwalkarn.me/blog/
- Deep Learning Book - http://www.deeplearningbook.org
  - See also video lectures on the same page

Video lectures:

- https://www.youtube.com/watch?v=Xogn6veSyxA (Ian Goodfellow)
- https://www.youtube.com/watch?v=H-HVZJ7kGI0&t=1057s (Ava Soleimany)

Practical perspective:

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition
- See here for the code:
  - https://github.com/ageron/handson-ml2
  - https://github.com/ageron/handson-ml2/blob/master/14_deep_computer_vision_with_cnns.ipynb

More Resources about RNNs

http://www.deeplearningbook.org/contents/rnn.html

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-gradient-problems/

Video lectures

- https://www.youtube.com/watch?v=ZVN14xYm7JA&feature=youtu.be
- https://www.youtube.com/watch?v=o2QuErsWp6k&t=1s