Session 1: Overview of Neural Networks and Deep Learning

Eric Fosler-Lussier

TDAI Foundations of Data Science & Artificial Intelligence

Deep Learning Summer School

Plan for today

- Basics of Neural Networks: how and why they work
  - (simple cases only!)
- The case for multiple layers
- Demo 1: Building a 4-2-4 autoencoder in Pytorch
- A brief tour of the neural network model zoo
  - A deeper look: convolutional networks for image processing
  - Take-home demo 2: Exploring multi-layer perceptrons vs convolutional networks
  - Sequence processing
  - Alternative learning strategies

Neural networks

A trainable, mathematical model for finding classification boundaries

(or regression values, or….)

Inspired by biological neurons

- Neurons collect input from receptors, other neurons
- If enough stimulus collected, then neuron fires

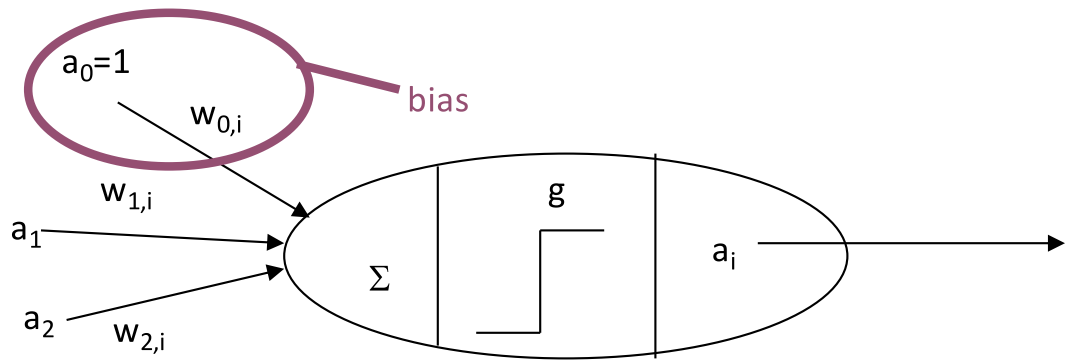
Inputs in artificial neurons are variables, outputs of other neurons

Output is an “activation” determined by the weighted inputs

A Basic Neural Network Unit

The input to a neuron is given as $in_i = \sum_j w_{ji} a_j = \boldsymbol{w^T a}$

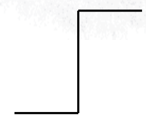
The activation of the unit is given as $a_i = g(in_i) = g\left(\sum_j w_{ji} a_j\right)$



Q: Is linear regression a neural network by this definition? If so, what is g(x)?

Activation function

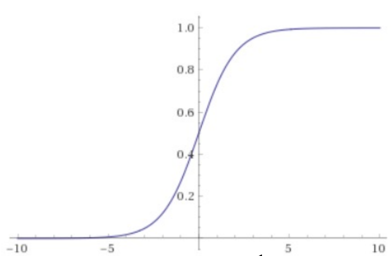
The g() function acts as a decision rule



Thresholding: hard boundary

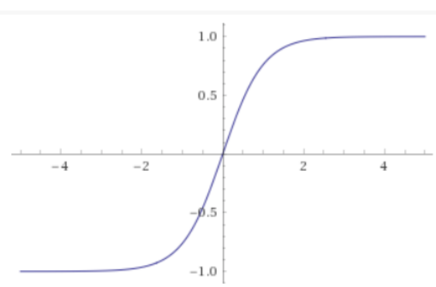
g(x)=0 if x<0, 1 otherwise (problem: not differentiable!)

Other popular nonlinear functions:



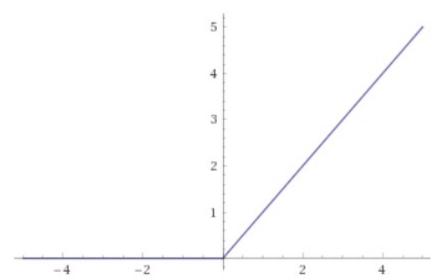
$$g(x) = \frac{1}{1+e^{-x}}$$

Sigmoid



g(x) = tanh(x)

Hyperbolic tangent



g(x) = x if x>0, 0 otherwise

ReLU (Rectified Linear Unit)

Multi-class classification

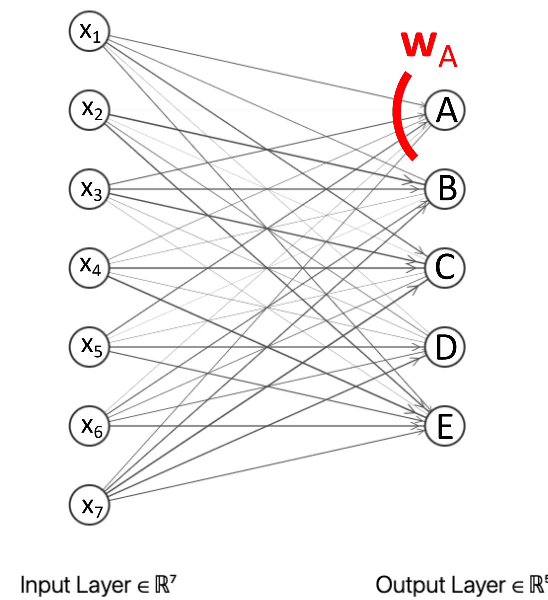
We can train a neural network to provide probability estimates over classes using a competitive classification

Train neural network with **one-hot** encoding for targets

- If there are *n* different outputs, use *n* output neurons
- Training signal is 1 for correct class, 0 for others
  - Example: correct class is "C" out of A,B,C,D,E: targets 0,0,1,0,0

Last layer is a competitive **softmax** layer

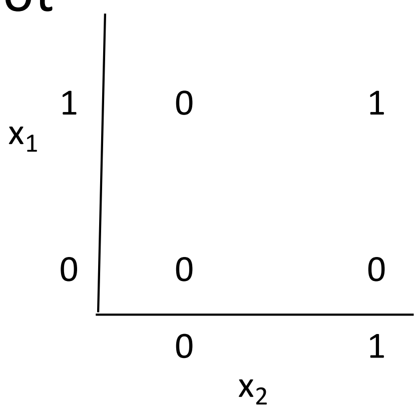
$$\frac{e^{\boldsymbol{w}_i^T \boldsymbol{x}}}{\sum_j e^{\boldsymbol{w}_j^T \boldsymbol{x}}}$$



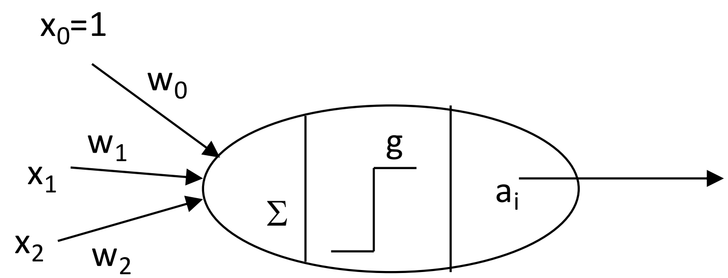
Single layer perceptron & logical functions

An array of neurons is called a perceptron

A single neuron can be used to represent the logical functions and, or, not



and

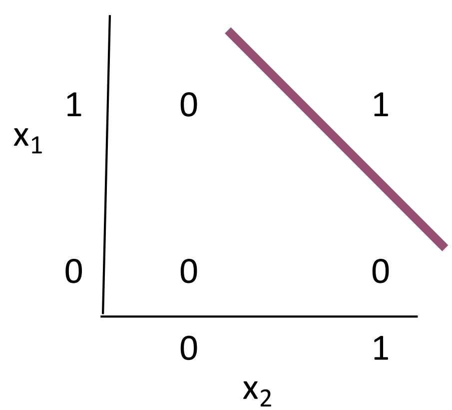


$$a_i = g\left(\sum_{j=0}^{2} w_j x_j\right) > 0$$

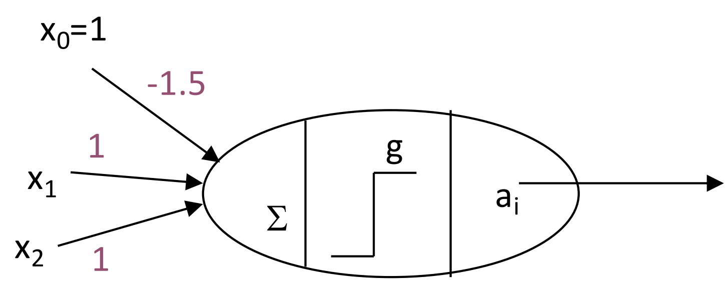
Single layer perceptron & logical functions

Setting w_0 to -1.5 and w_1,w_2 to 1 gives “and” rule.

Similar for or, not.



and

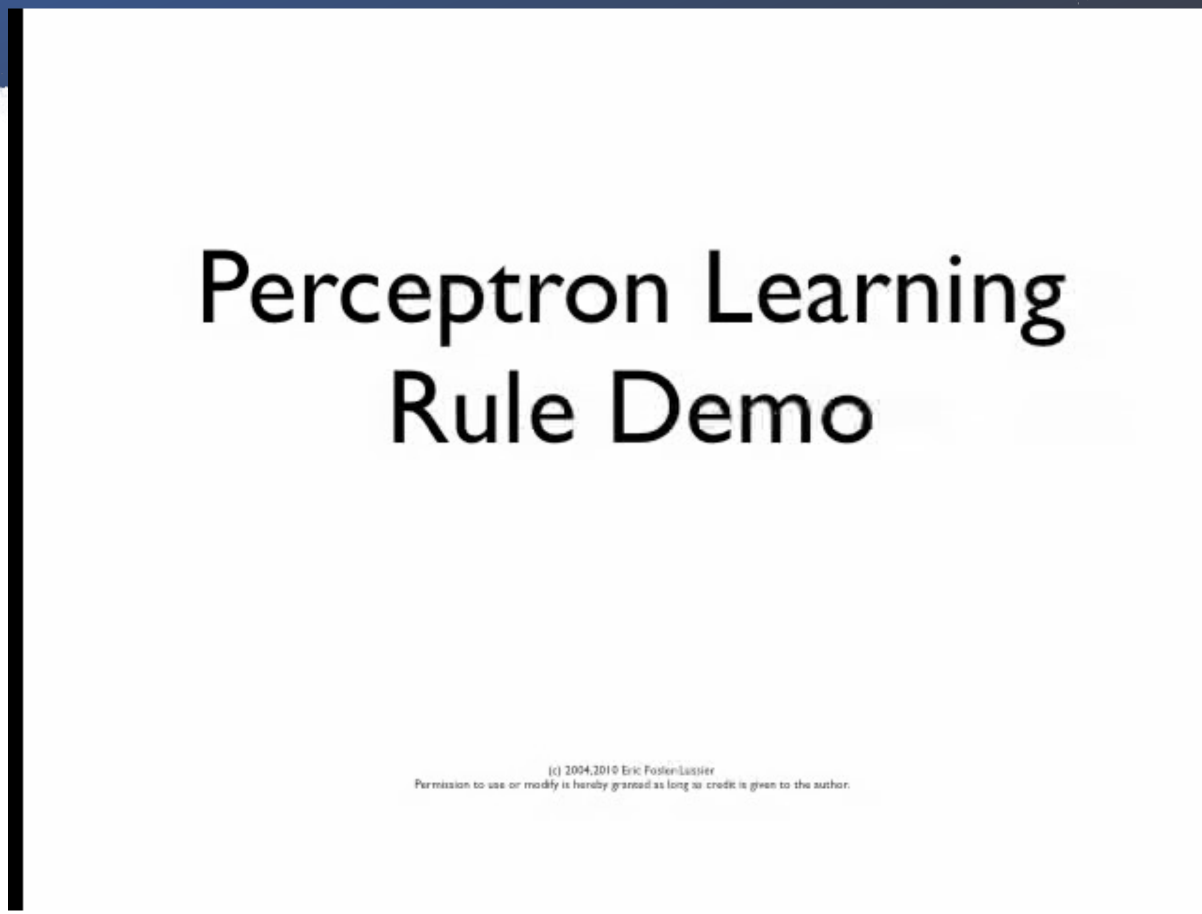


$$a_i = g\left(\sum_{j=0}^{2} w_j x_j\right) > 0$$

Training a Single Perceptron

True boundary is blue dotted line

Watch as red line moves - red points are ones with errors



Perceptron Learning Rule:

$w \leftarrow w + \eta[d - y]x$

Weight

Learning rate

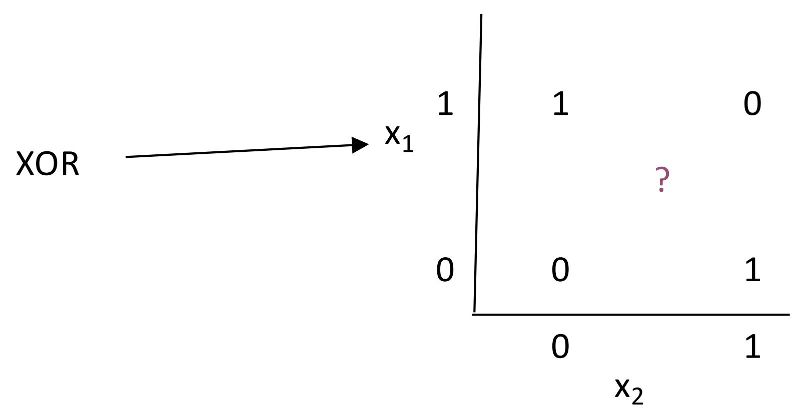
Desired output – current output

Input

XOR problem: need for multiple layers

Originally, a lot of excitement over neural networks

Minsky and Pappert (1969) then showed that there were problems that you couldn't represent using single layer perceptrons

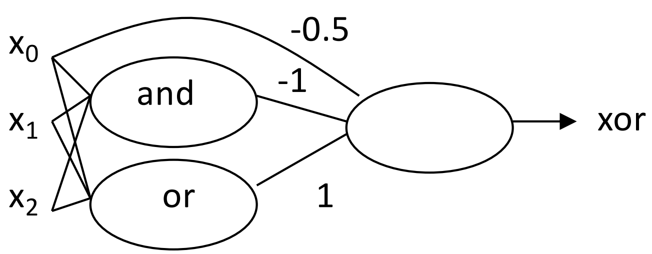


XOR problem

Notice that you can express XOR as a combination of other functions:

x_1 XOR x_2 = $(x_1 \vee x_2) \wedge \sim(x_1 \wedge x_2)$

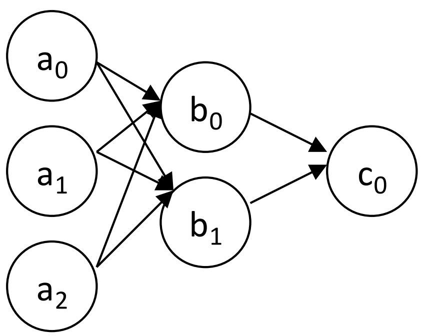
We can build an ensemble network:
multi-layer perceptron



Multi-layer perceptron

Key idea: output from first neurons becomes input for later neurons

Middle node known as **hidden layer**



$$o = g\left(\sum_j w_j^{bc} g\left(\sum_k w_{kj}^{ab} a_k\right)\right)$$

Training Neural Networks

Neural networks try to minimize some sort of loss/error function

How? Gradient Descent Optimization (see next session)!

In general: compute the gradient of the loss with respect to weights, take a step in direction opposite gradient

Ex: linear regression y=wx+b – minimize (mean) squared error to desired output d

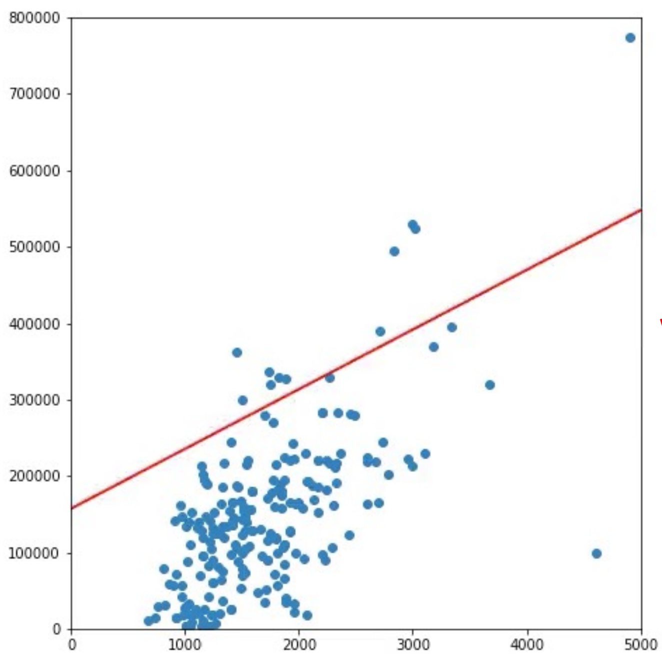
$$E = \frac{1}{2}(d - (wx + b))^2$$

$$W_j \leftarrow W_j - \eta \frac{\partial E}{\partial W_j}$$

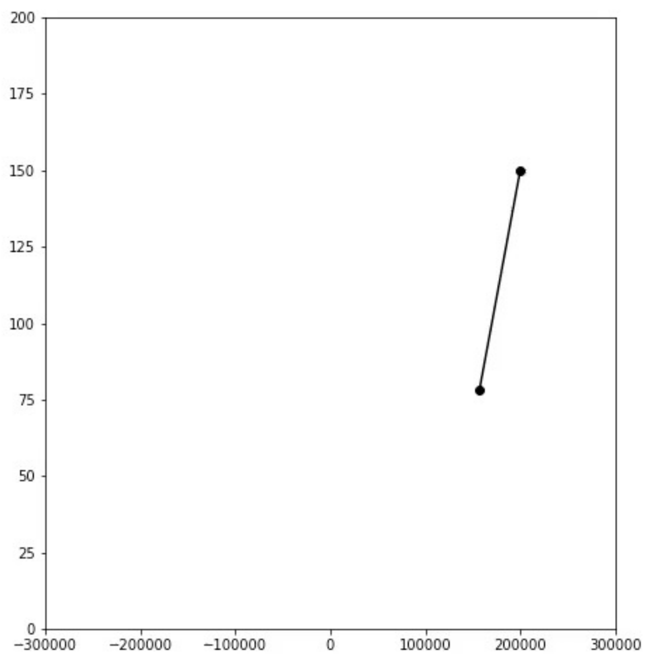
$$w \leftarrow w - \eta\big(d - (wx + b)\big)x$$
$$b \leftarrow b - \eta\big(d - (wx + b)\big)$$

Linear regression via gradient descent

y=wx+b



w



b

Training Multi-Layer Perceptrons

Error is now a function of multiple layers. Assuming single desired output d:

$$E = \frac{1}{2}(d - g_2(\boldsymbol{w}_2^T g_1(\boldsymbol{w}_1^T \boldsymbol{x})))^2$$

The general rule for updating is still

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

For multiple layers, need to use chain rule of calculus to update lower layers: **error backpropagation (backprop)**

[The math can get real messy!]

Modern network architectures keep track of gradients (derivatives of error) with variables and can do this automatically!

Intuition behind backprop

For every training pattern, we “forward” the input and get an output, which may be wrong (since we have targets).

We have that error and want to assign the blame to weights proportionally

We can compute the error derivative for the last layer

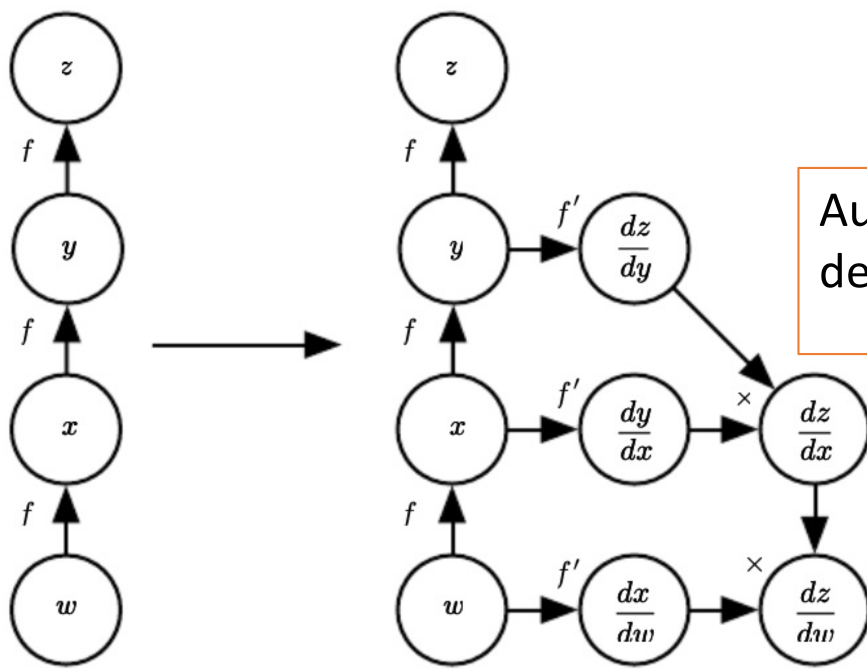
- Accumulate blame at each of the hidden nodes by summing over weights attached to that node

Now distribute blame to previous layer

Derivative Graph Computation

Consider a function $z = f\left(f\left(f(w)\right)\right)$, compute $\frac{dz}{dw}$.

User specified computational graph



Automatically derived derivative graph

Goodfellow, Bengio, Courville 2016
Goodfellow, Bengio, Courville 2016

Derivative Graph Computation for MLP

Graph for computing cross-entropy cost function in MLP with one hidden layer, ReLU units, weight decay.

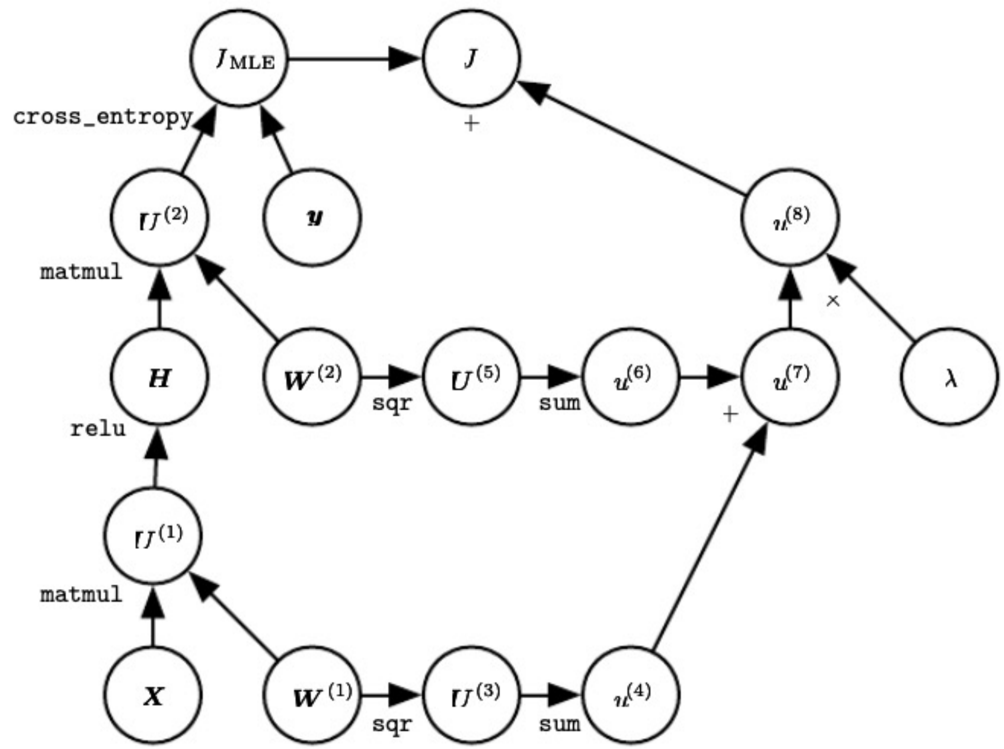
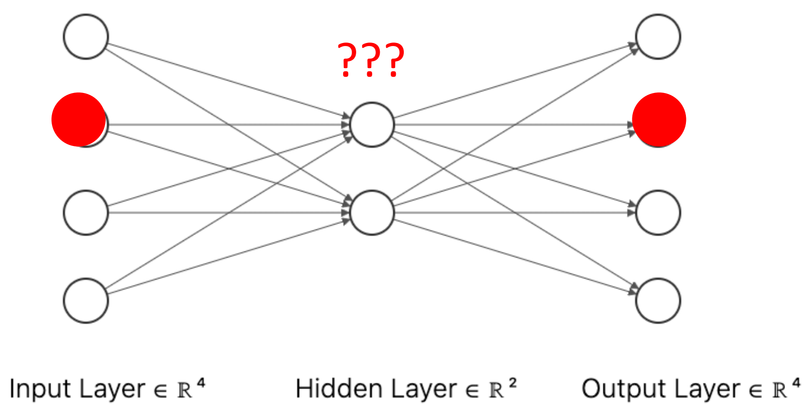


Figure 6.11: The computational graph used to compute the cost to train our example of a single-layer MLP using the cross-entropy loss and weight decay.

Goodfellow, Bengio, Courville 2016

Demo #1: Training a 4-2-4 autoencoder



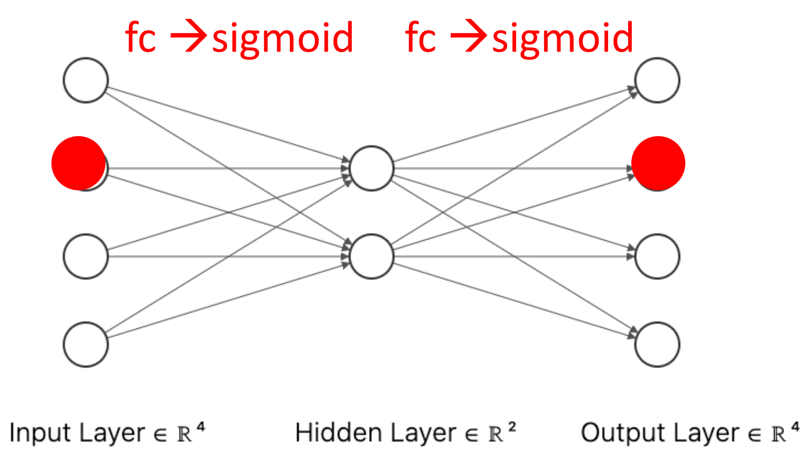
A simple MLP can encode "one-hot" bit strings

Idea: make the output match the input

"0100" -> hidden -> "0100"

https://bit.ly/38zV6XN

Demo #1: Training a 4-2-4 autoencoder



<u>This model has two layers</u>

Layer 1: 4x2 fully connected layer (+ bias term)

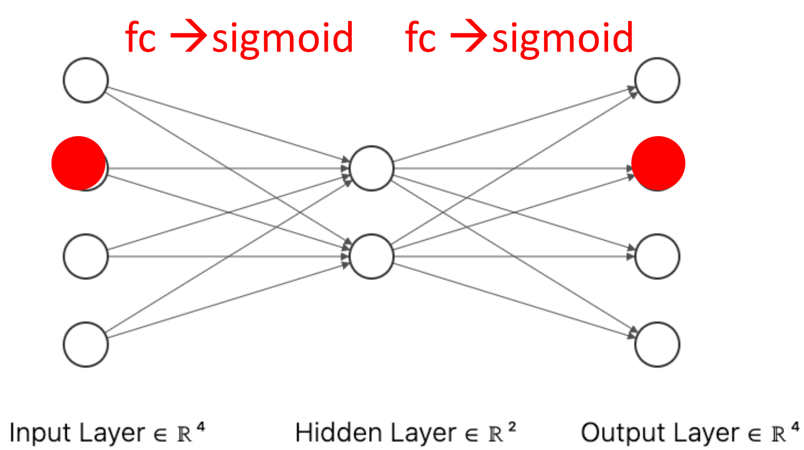
Each hidden node takes summed input, then passes through sigmoid

Layer 2: 2x4 fully connected layer (+bias term)

Again pass the summed input through sigmoid (0-1 output).

https://bit.ly/38zV6XN

Demo #1: Training a 4-2-4 autoencoder



<u>Building a learner in Pytorch</u>

1) Define your architecture
    1) Initialize structure
    2) Define a "forward" function
2) Choose a loss function
3) Choose an optimizer
4) Train the model

https://bit.ly/38zV6XN

Break

Model Zoo: Architectures

The Multilayer Perceptron (MLP) aka Deep Neural Network (DNN) is a workhorse, but is expensive in parameters

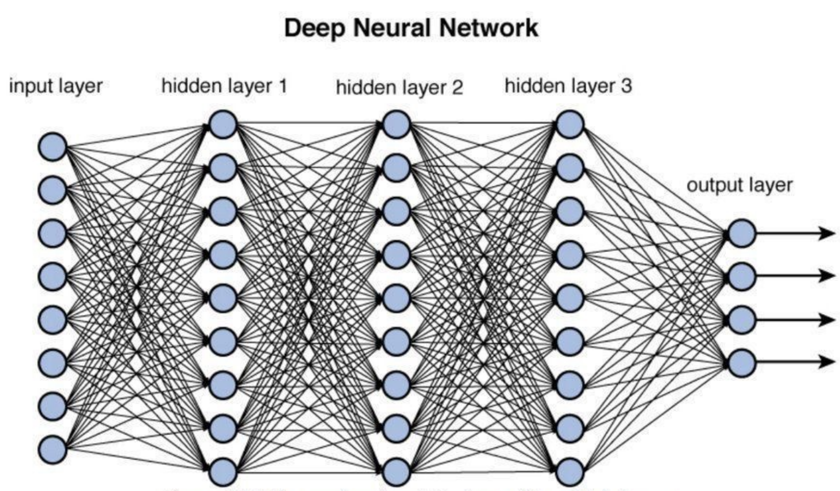


Figure 12.2 Deep network architecture with multiple layers.

https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Different models have been developed to handle different situations:

- Convolutional Networks: local feature detection, location invariant (images)
- Recurrent Networks: handle sequences of data (speech, text, economics)
- Attention Models/Transformers: for sequence/spatial data, focus attention more directly (images/text/speech)

Model Zoo: New Kinds of Training

We implied that we have to have a supervised signal to train neural networks. However there are newer techniques that can be used (perhaps in combination with supervision):

Student-teacher learning: use one model to train another (e.g. train a small model using a large model for training)

Generative adversarial networks (GANs): train a generation model (G) to create output (images, audio,…) that tries to fool a discrimination model (D) into thinking the generated output is real

Contrastive learning: make the representations of objects of the same class similar and of different classes farther apart

Deep Neural Networks

In theory, you can stack as many layers on top of each other as you want

- Complexity: how many parameters?
- Assume 100 inputs, 1000 hidden units per layer, 1 output

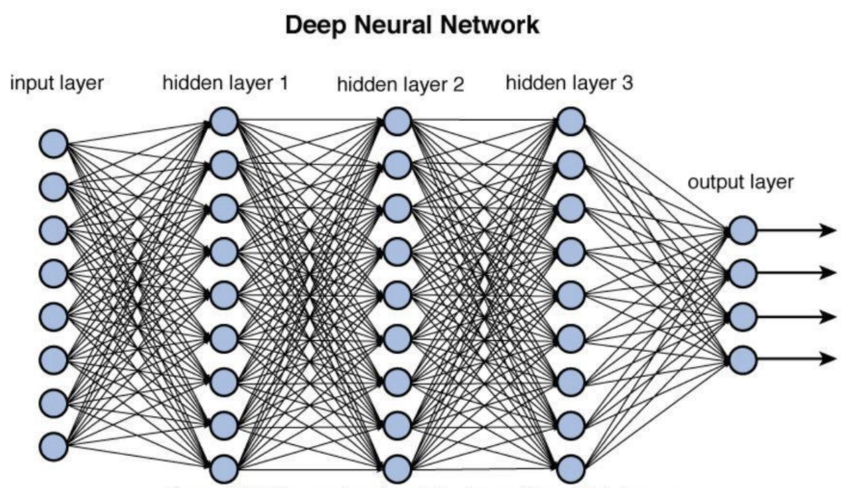


Figure 12.2 Deep network architecture with multiple layers.

https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Deep Neural Networks

In theory, you can stack as many layers on top of each other as you want

- Complexity: how many parameters?
- Assume 100 inputs, 1000 hidden units per layer, 1 output
  - Input layer: 100x1000 = 100,000 parameters
  - Each hidden layer: 1000x1000 = 1,000,000 parameters
  - Output layer: 1000x1 = 1000 parameters
  - Total: k x 1m + 101,000 parameters for k layers

Problem: as you add more layers, the gradient gets smaller and smaller (**vanishing gradient problem**)

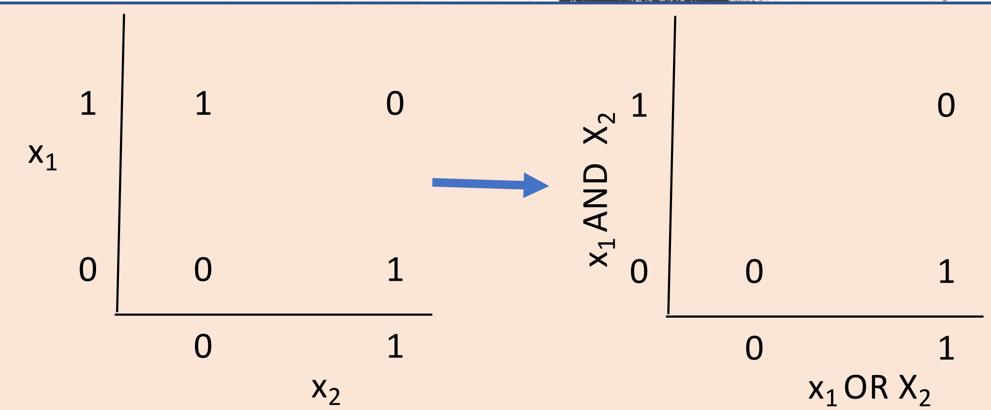
- Translation: it's harder to learn the farther away you are from the signal

What do multiple layers do?

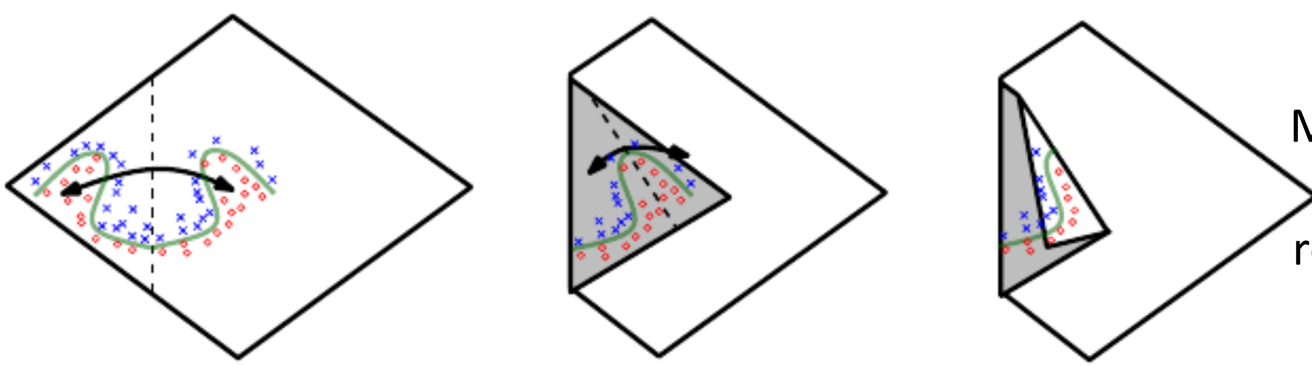
Linear layers only perform linear transforms of input space

(reflection, rotation, dilation,….)

Nonlinearities allow for a wider range of transformations (e.g. folding)



XOR network “folds” cases of OR, not AND together



Montufar et al 2014: folding around rectifier nonlinearity

https://www.deeplearningbook.org

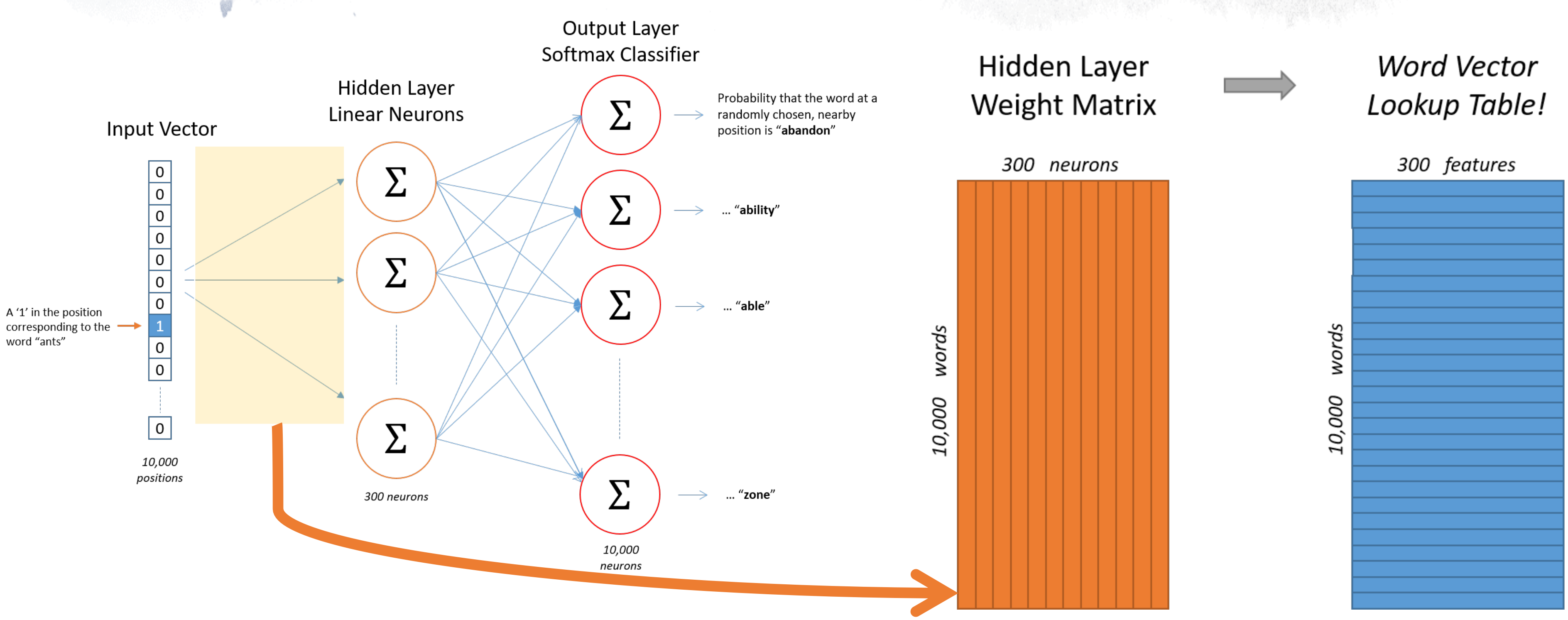
MLP applications
Word2Vec: Embeddings for words

We can use MLPs to train models that tell us something about the semantics of words

- Core idea: two words with similar meanings should have similar contexts
  - The _______ sat on the throne. (King – Queen)
- Try to predict the words that co-occur with each word

Initial weight matrix becomes a real valued **embedding** of the words of the vocabulary

Word2Vec: Embeddings for words

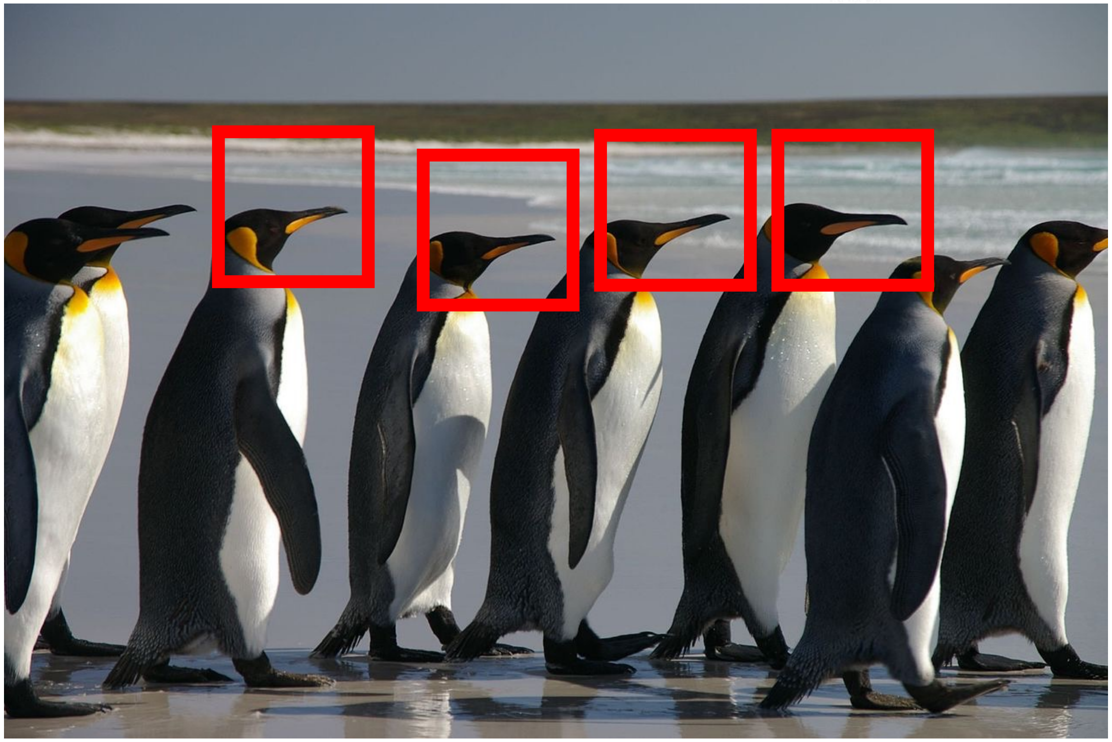


http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Find the penguins…

Wikimedia Commons

Image detectors might look for similar patches over an image…



Convolutional networks

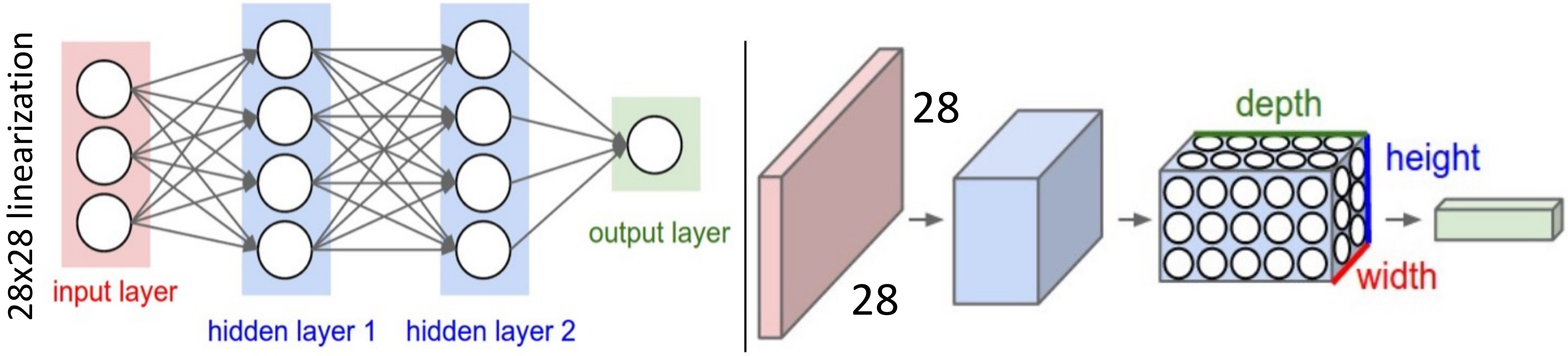
Fully connected networks have a lot of parameters

Sometimes want networks that have the same pattern detectors over different parts of the image

- In other words: **shift-invariant**

Convolutional networks use smaller layers but **convolve** across the entire input (image, sentence, etc)

From fully connected to convolutional networks



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

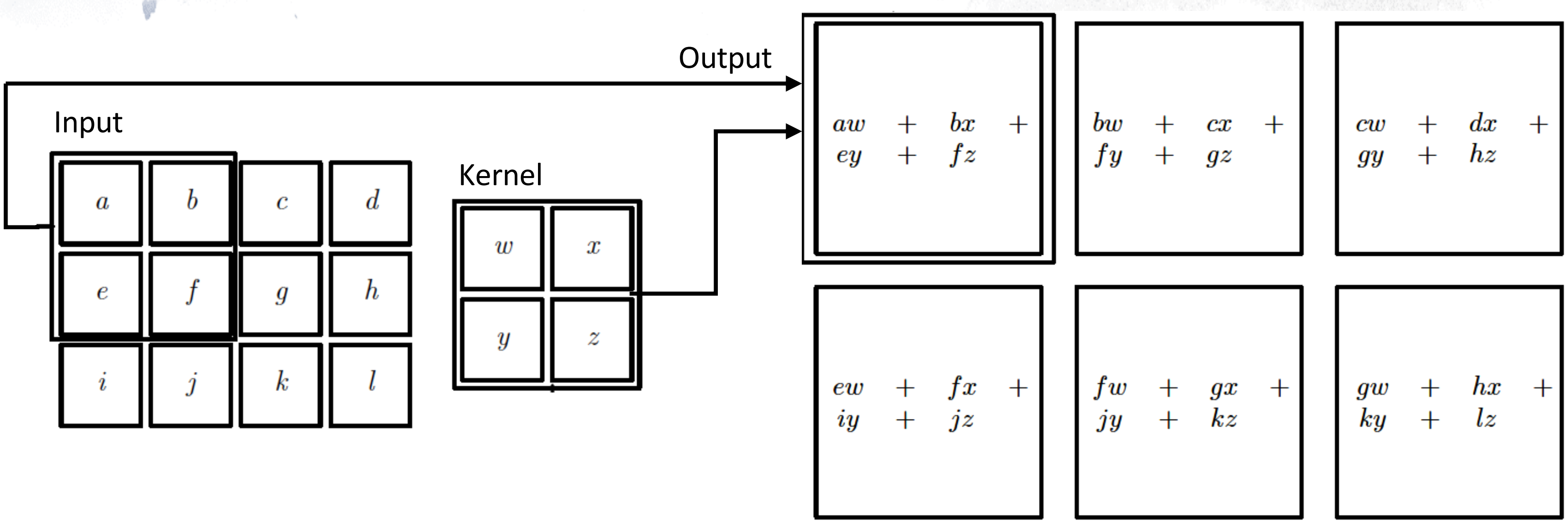
Thanks to DeLiang Wang for some of these slides

https://cs231n.github.io/convolutional-networks/

Convolutional Networks (CNNs) in more depth

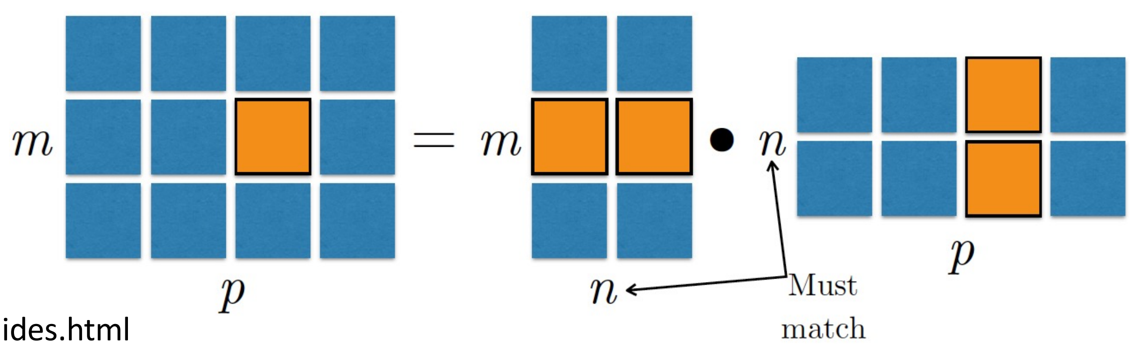
- Three kinds of layers to build a CNN architecture
  - Convolutional layer
  - Pooling layer
  - Fully connected layer, like in conventional MLP
- Three operations
  - Convolution (correlation)
  - Max pooling
  - Rectified linear unit (ReLU) as activation function

2-D convolution



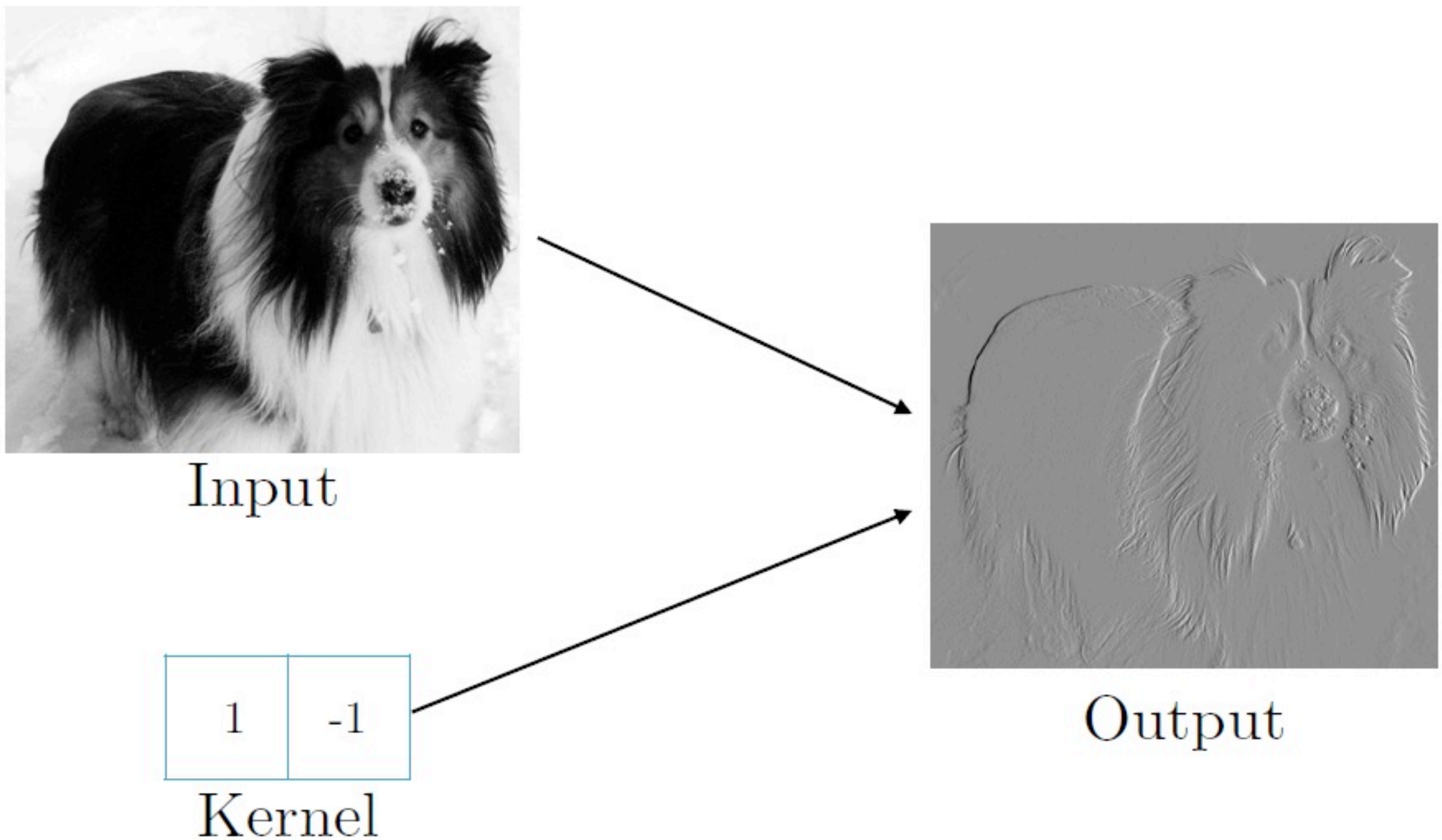
$$\boldsymbol{C} = \boldsymbol{AB}.$$

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}.$$



https://www.deeplearningbook.org/lecture_slides.html

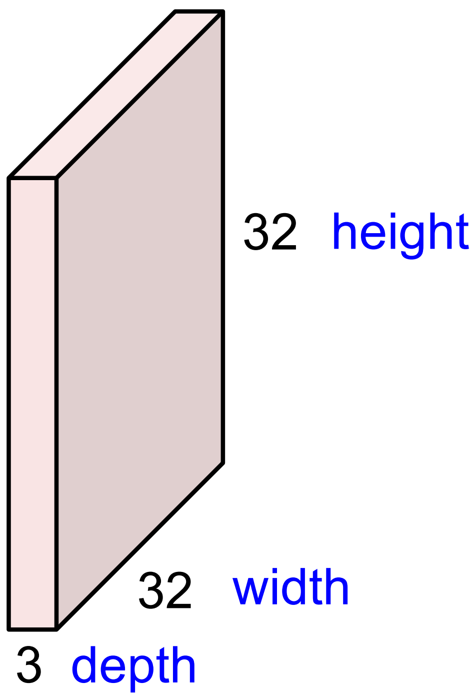
Example: edge detection by convolution



https://www.deeplearningbook.org/lecture_slides.html

Convolution layer

32x32x3 image, with spatial structure

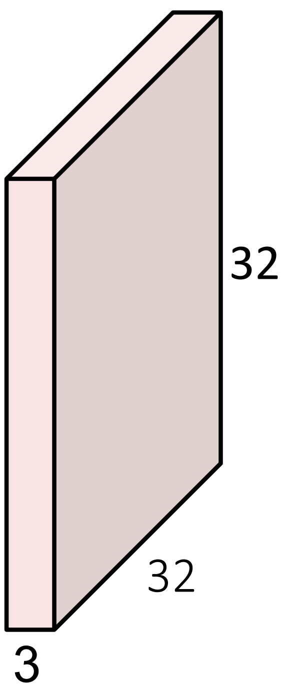


https://www.deeplearningbook.org/lecture_slides.html

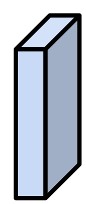
Convolution layer (cont.)

Filters extend the full depth of the input volume

- 32x32x3 image

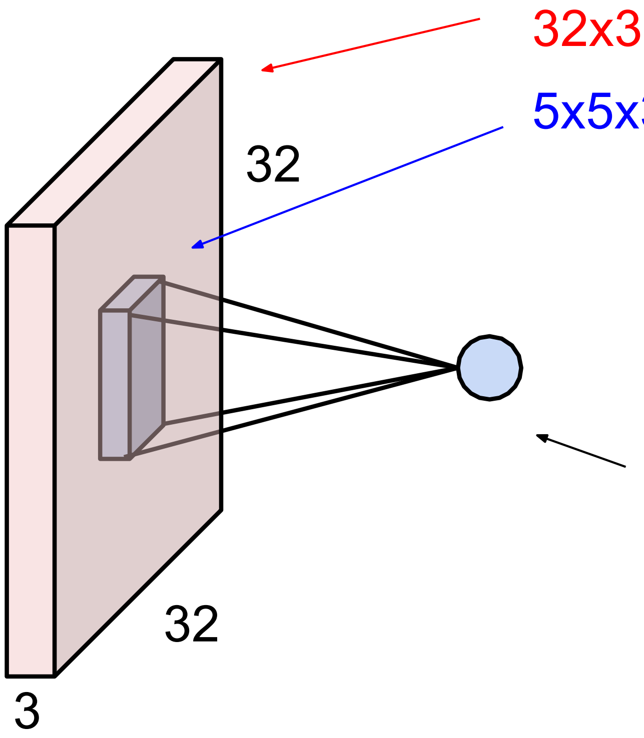


- 5x5x3 filter



- Convolve the filter with the image
- That is, “slide over the image spatially, computing dot products”

Convolution layer (cont.)

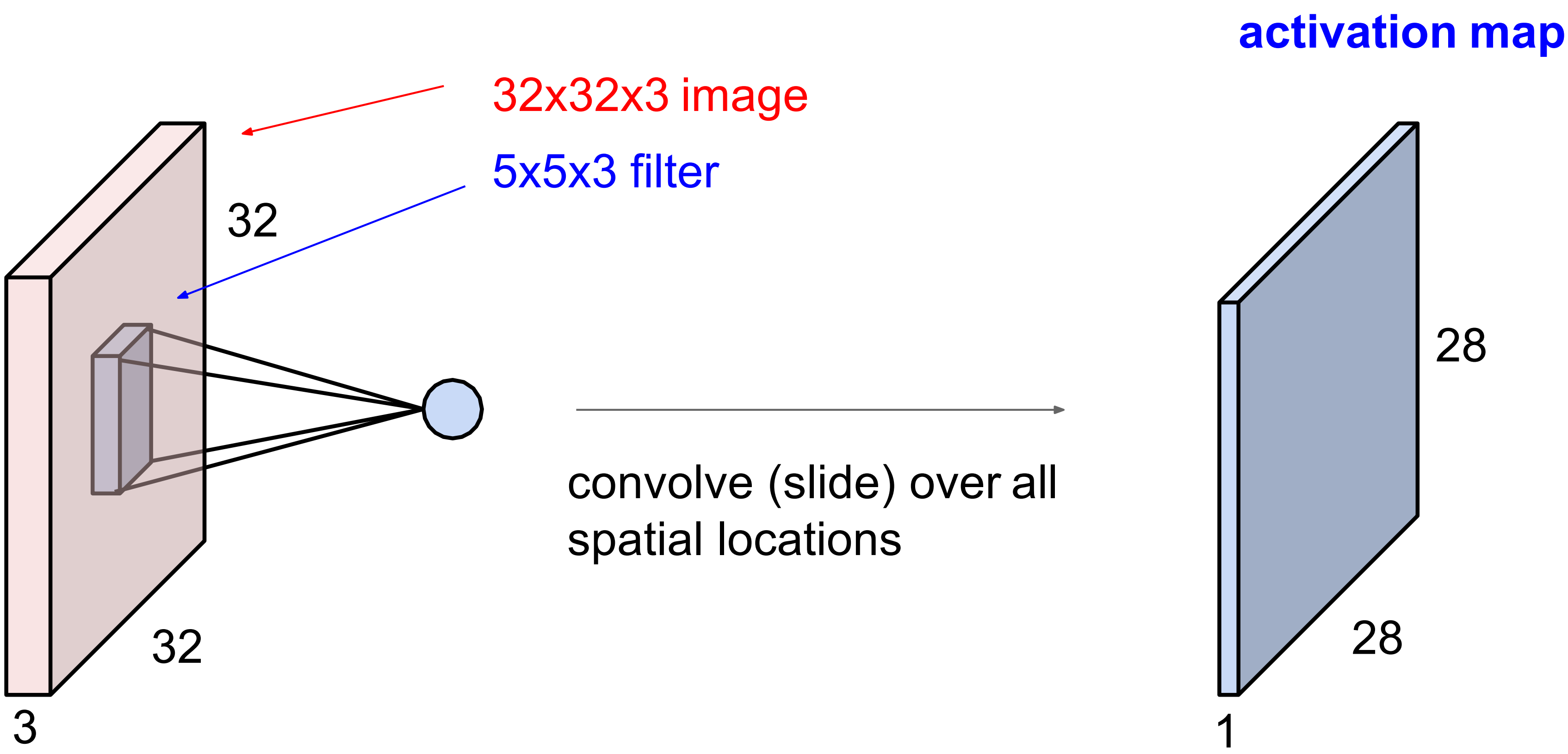


W

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

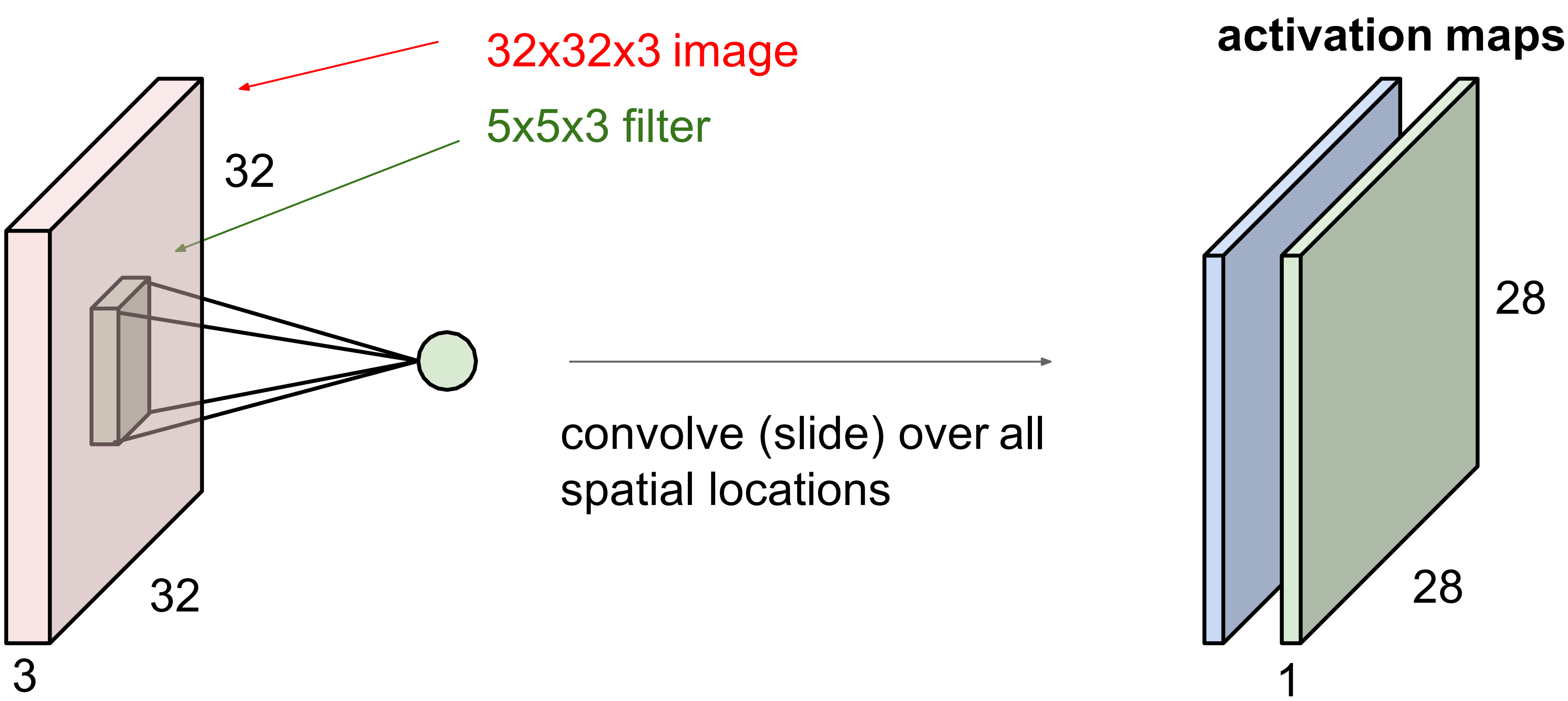
$$\mathbf{w}^T\mathbf{x} + b$$

Convolution layer (cont.)

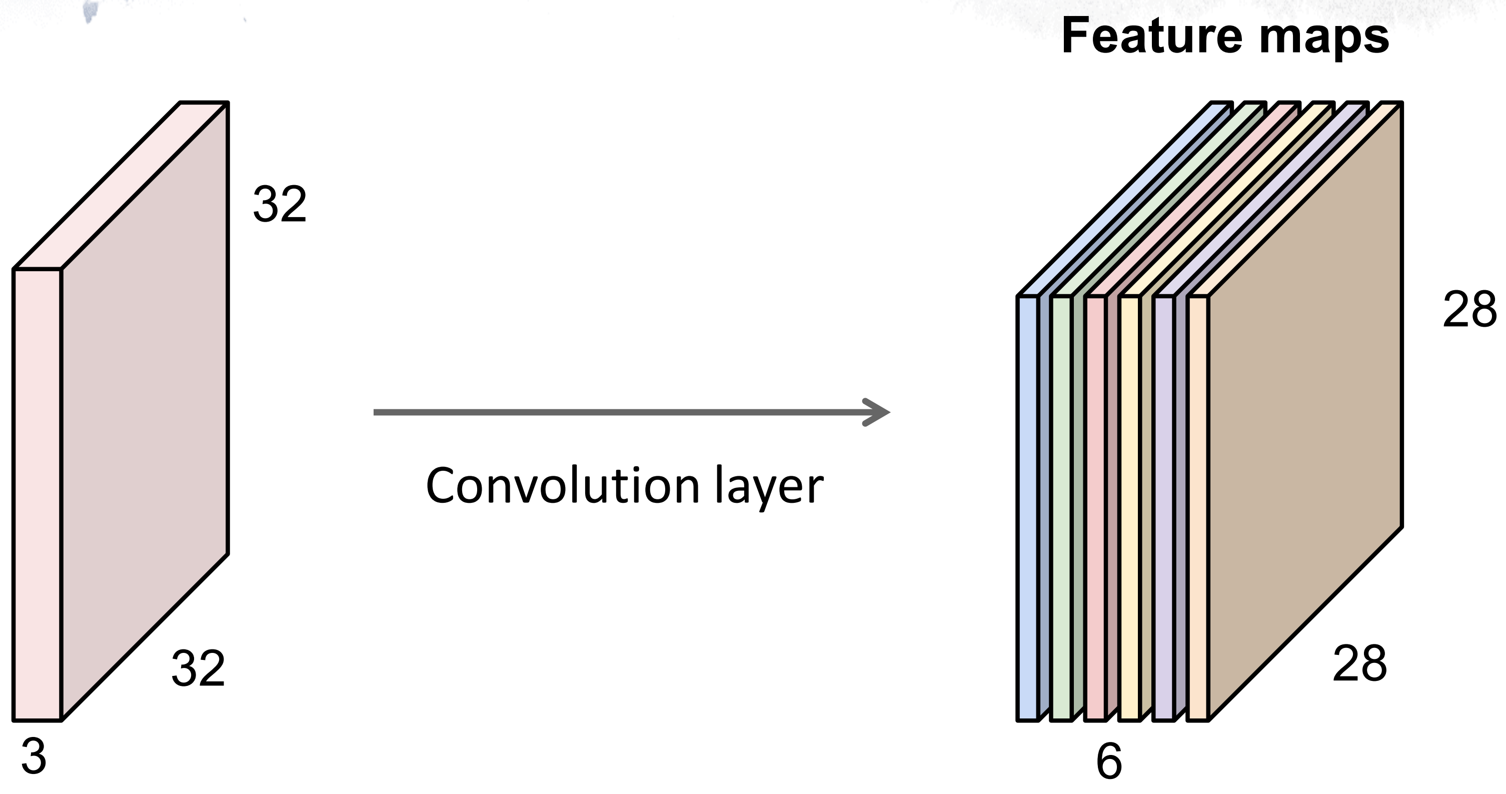


Convolution layer (cont.)

Consider a second, green filter

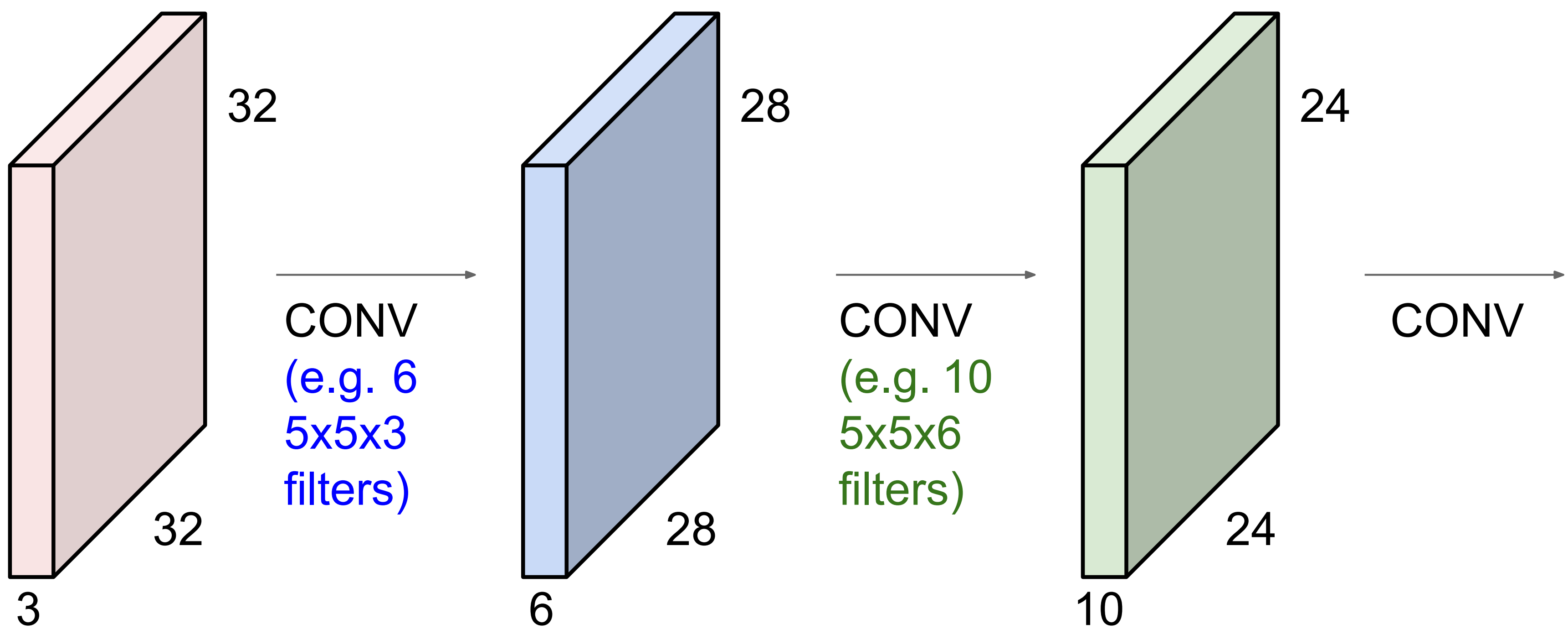


Convolution layer (cont.)



We stack these to get a new “image” of size 28x28x6

CNN with multiple layers



https://www.deeplearningbook.org/lecture_slides.html

Making CNNs more robust

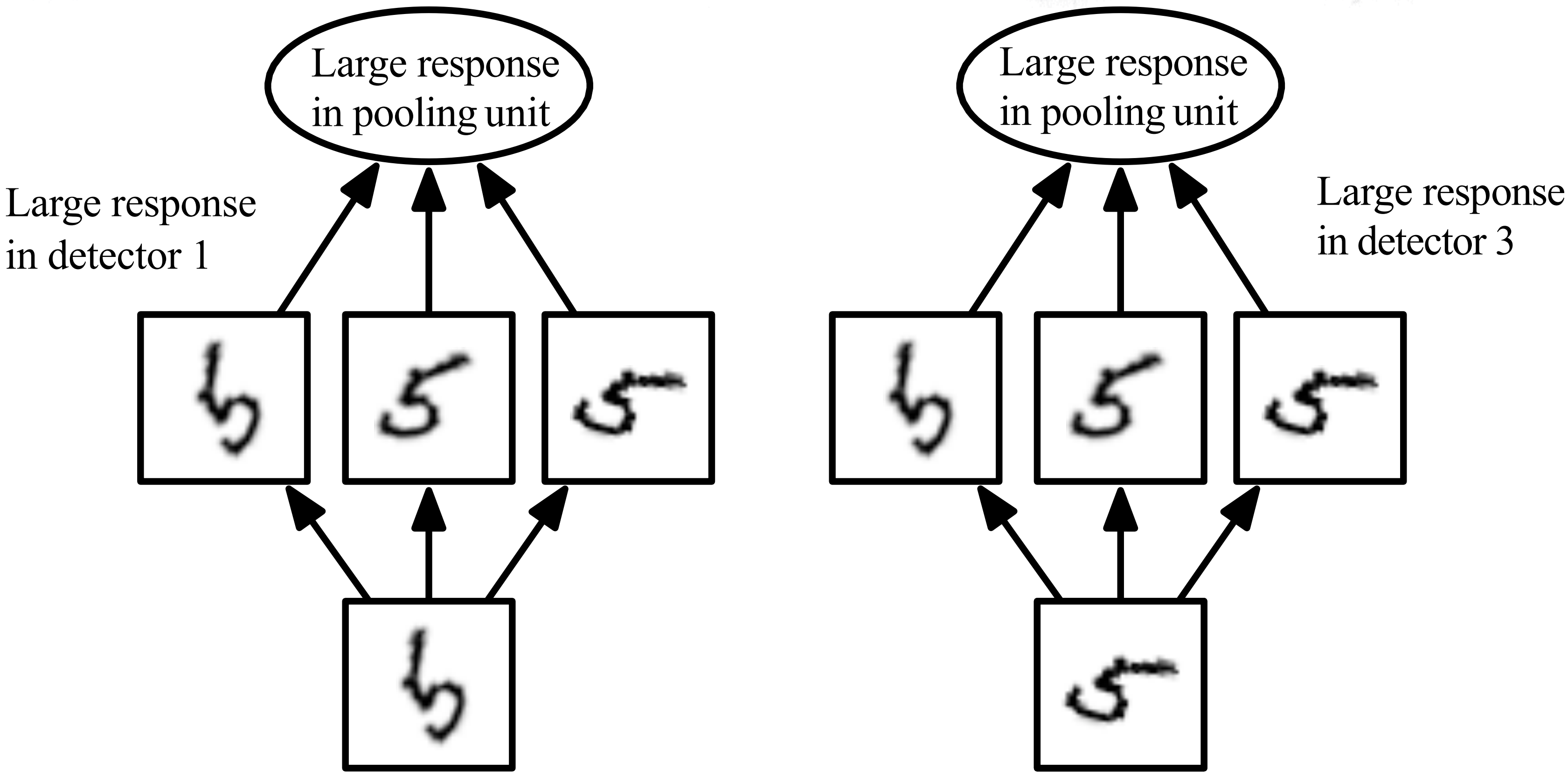
Max-pooling: over some group of convolutions, copy the value of the highest one

- Imagine you had a detector for “2”
  - Apply across all possible shifts of the detector, and then take the highest output.
  - Creates a shift-invariant detector

Dropout

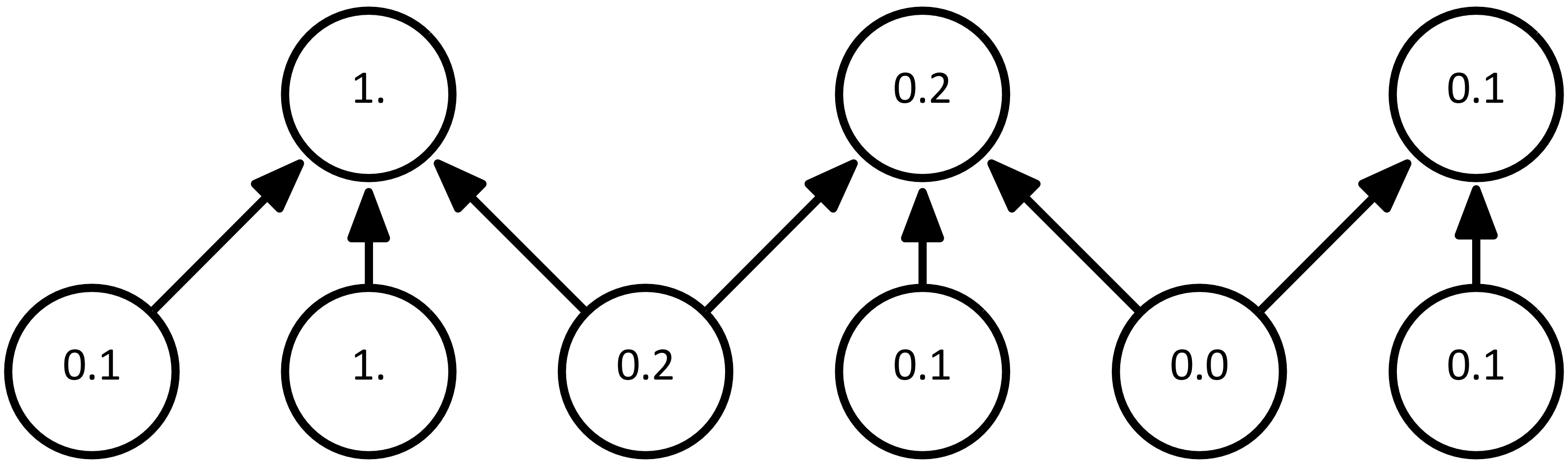
- Rather than computing the output of each unit exactly, set x% of them randomly to zero
- Promotes a more robust representation

Cross channel (filter) pooling and invariance to learned detectors



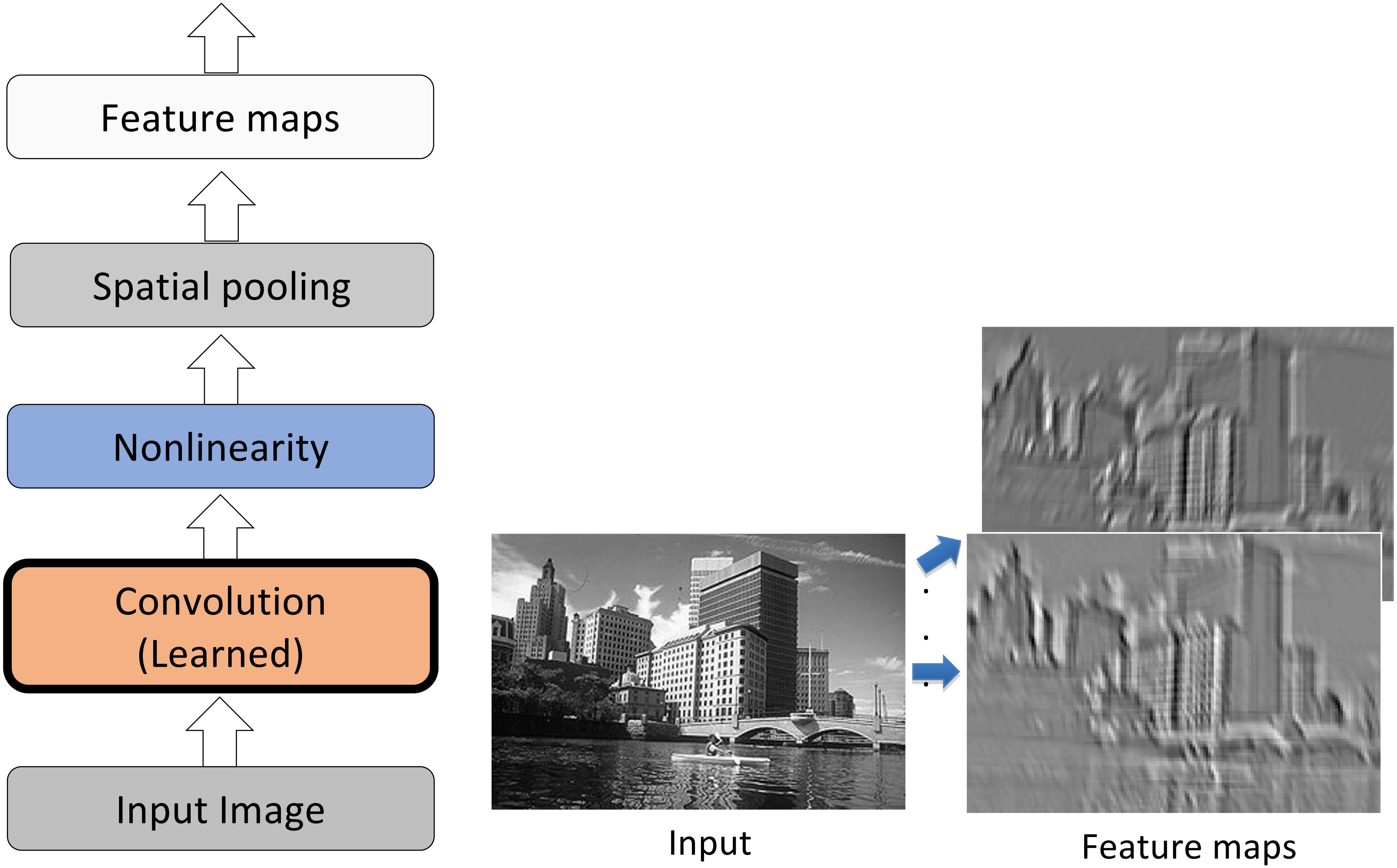
https://www.deeplearningbook.org

Max pooling with downsampling



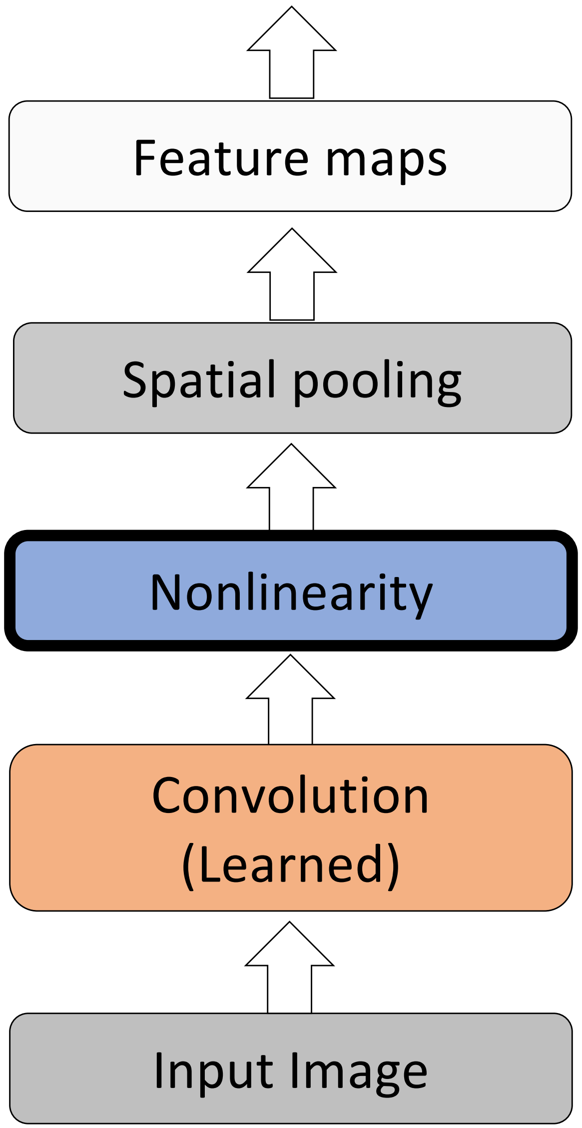
https://www.deeplearningbook.org

Typical CNN operations



https://www.deeplearningbook.org

Typical CNN operations

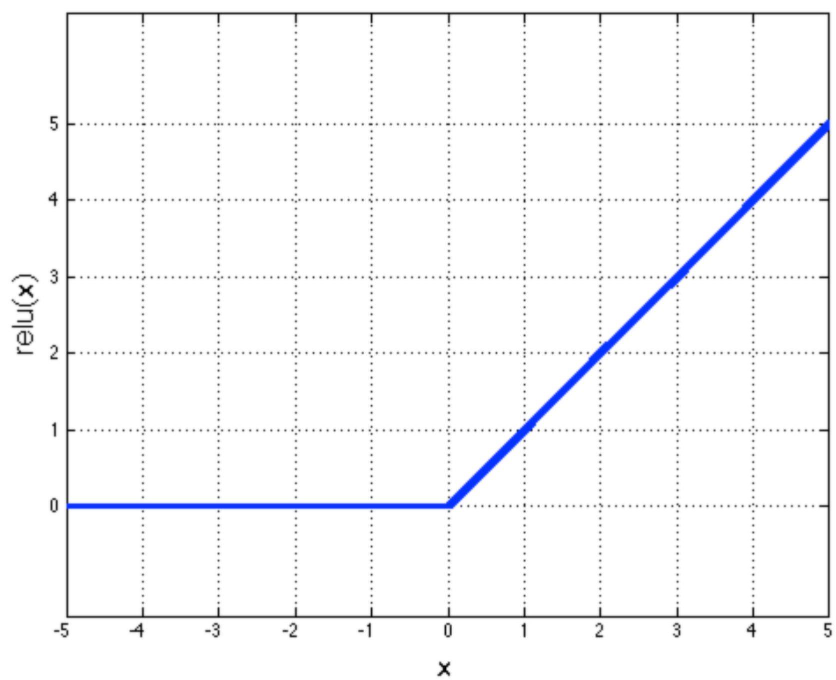


Modern activation function or nonlinearity:

Rectified Linear Unit (ReLU)

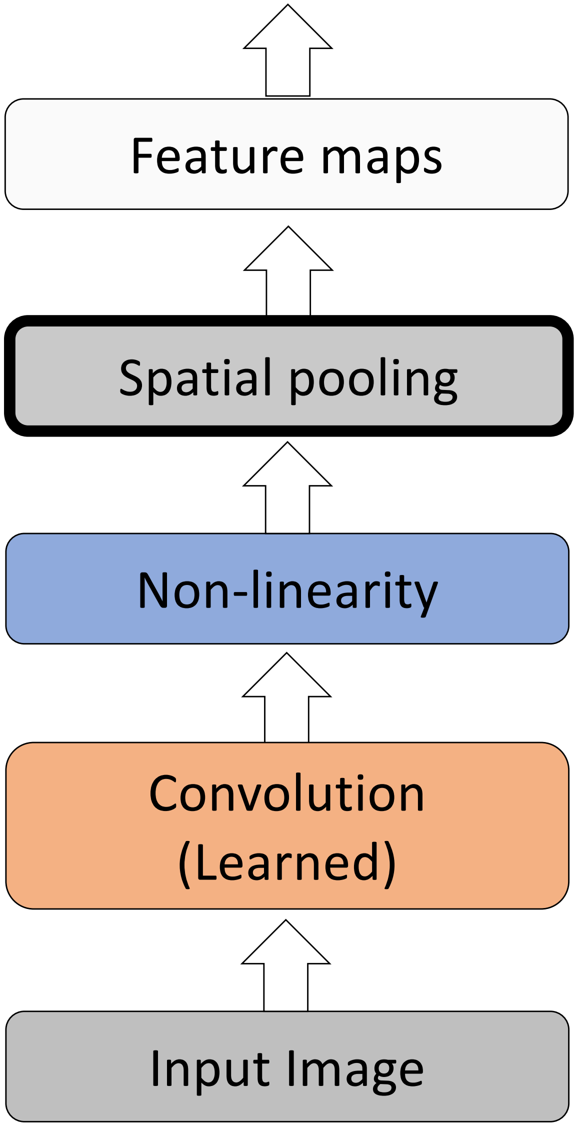
$$ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

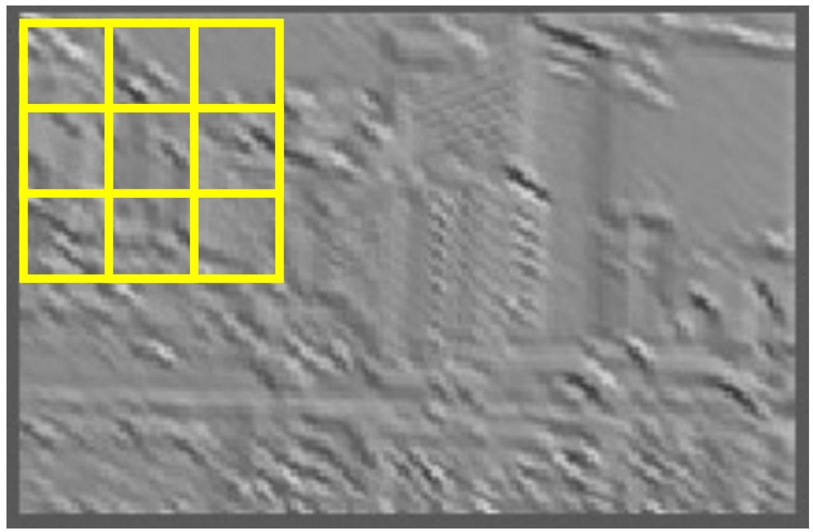
Rectified Linear Unit



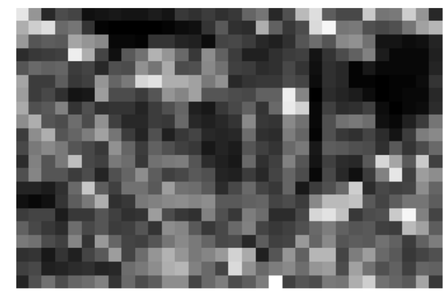
https://www.deeplearningbook.org

Typical CNN operations



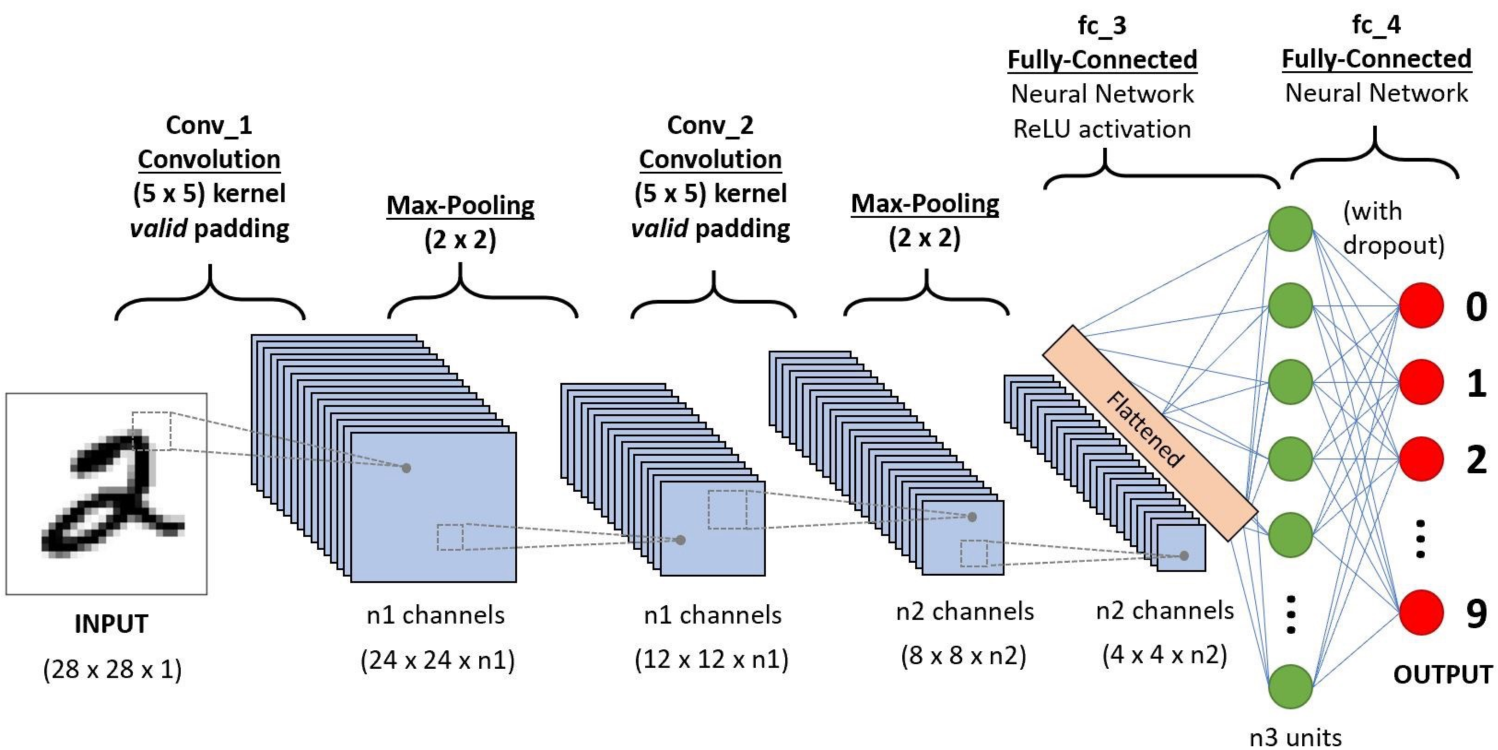


Max



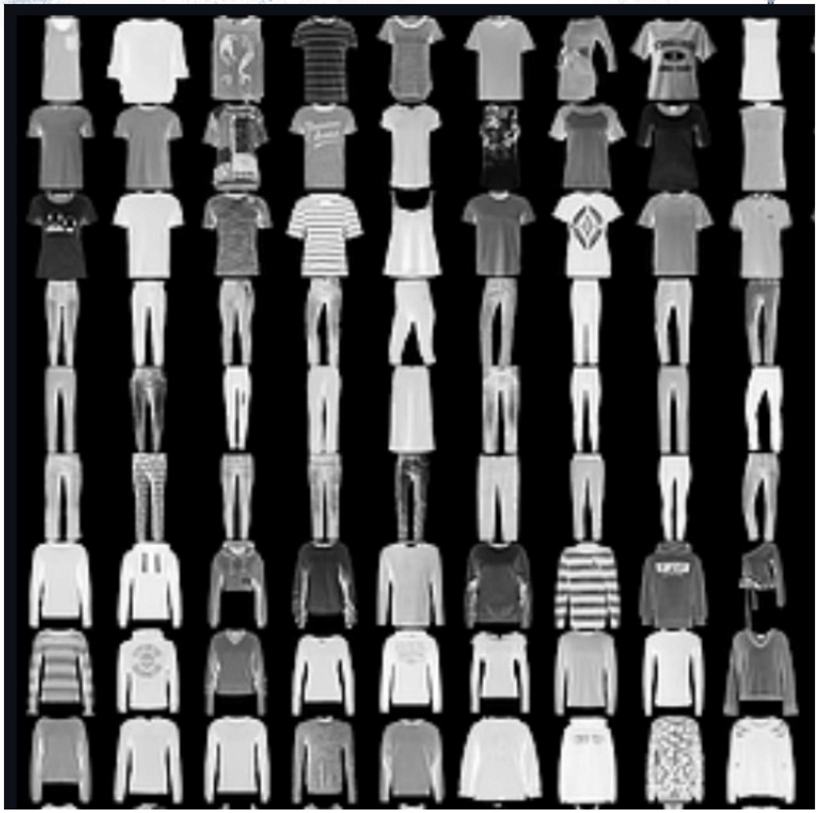
https://www.deeplearningbook.org

Convolutional Neural Network for Digit Recognition



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Fashion-MNIST



28

28

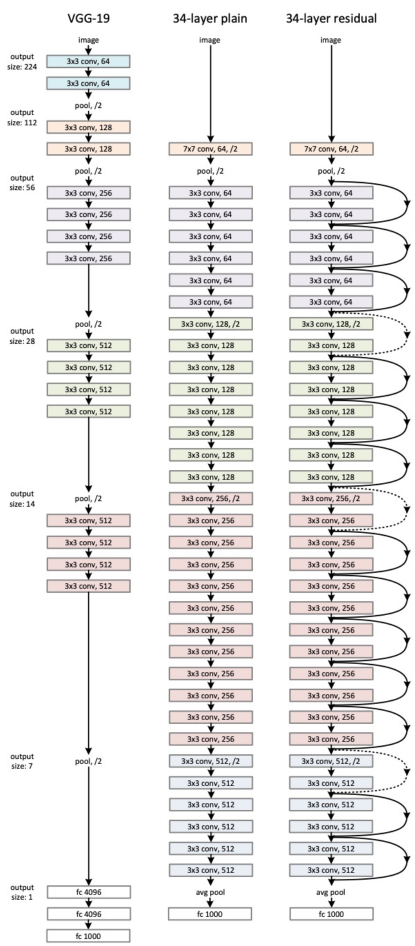
https://github.com/zalandoresearch/fashion-mnist

Take-Home Demo #2

Code to play around with Fashion-MNIST data set

https://bit.ly/3Q5Tp5B

Deep Architectures for Computer Vision



Convolutional NNs kept getting bigger and bigger

Left: VGG-19 and Resnet-34 architectures

Residual connections that allow connection between layers are important for deep learning

“Deep Residual Learning for Image Recognition“
He et. al, CVPR 2016
https://doi.org/10.48550/arXiv.1512.03385

Handling sequences

For sequences of input (text, speech, videos) several approaches for handling sequences:

Deep Neural Networks with windows: use a sliding window over input sequence to predict sequence of outputs

- **Time Delay Neural Networks (TDNNs)**

Recurrent networks which keep a "history" component

- **Recurrent Neural Networks (RNNs)** – introduce history
- **Long-Short Term Memory Networks (LSTMs), Gated Recurrent Units (GRUs)** – have the ability to remember/forget history

Attention-based models which learn which parts of the input to pay attention to

- **Transformers** – encode input with attention and then decode output

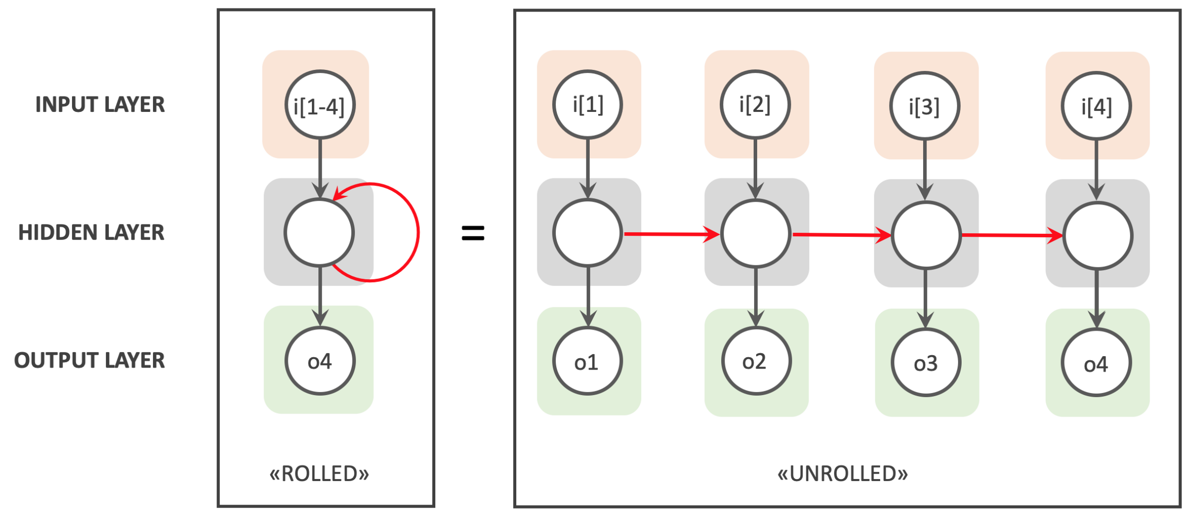
Recurrent networks

Key idea: like a deep neural network, but have the hidden units feed back to themselves

Hidden units keep a history state across inputs

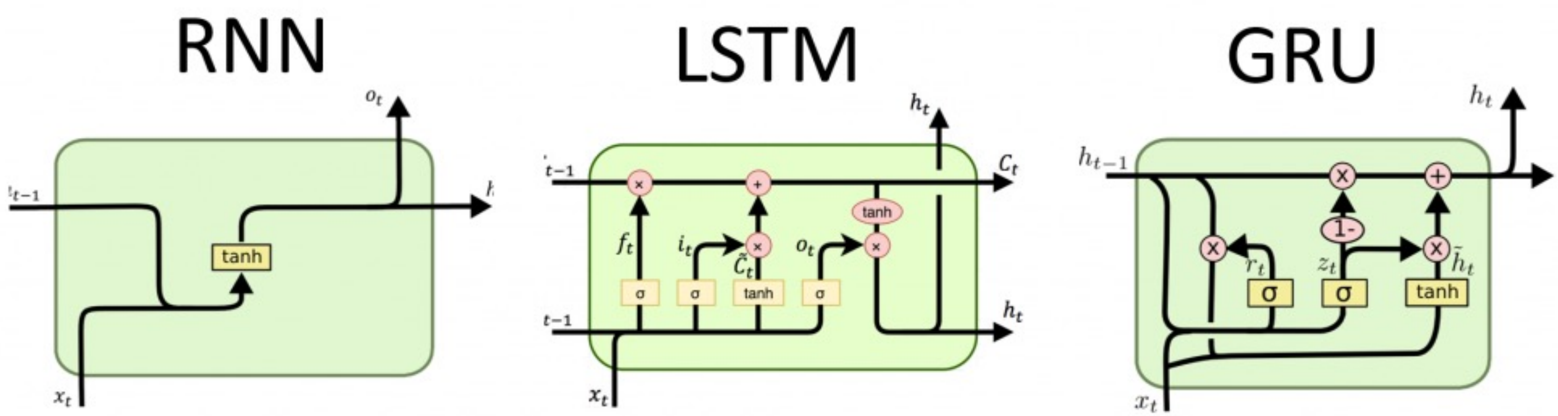
Training requires unrolling across time

Similar vanishing gradient problem for long sequences



https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks

LSTMs and GRUs make the history conditional



http://dprogrammer.org/rnn-lstm-gru

Both have gating mechanisms to let information through or not.
The ability to pass information across multiple time steps helps with the vanishing gradient problem.

Attention-based learning

Recurrent structures also have difficulty with long-term dependencies

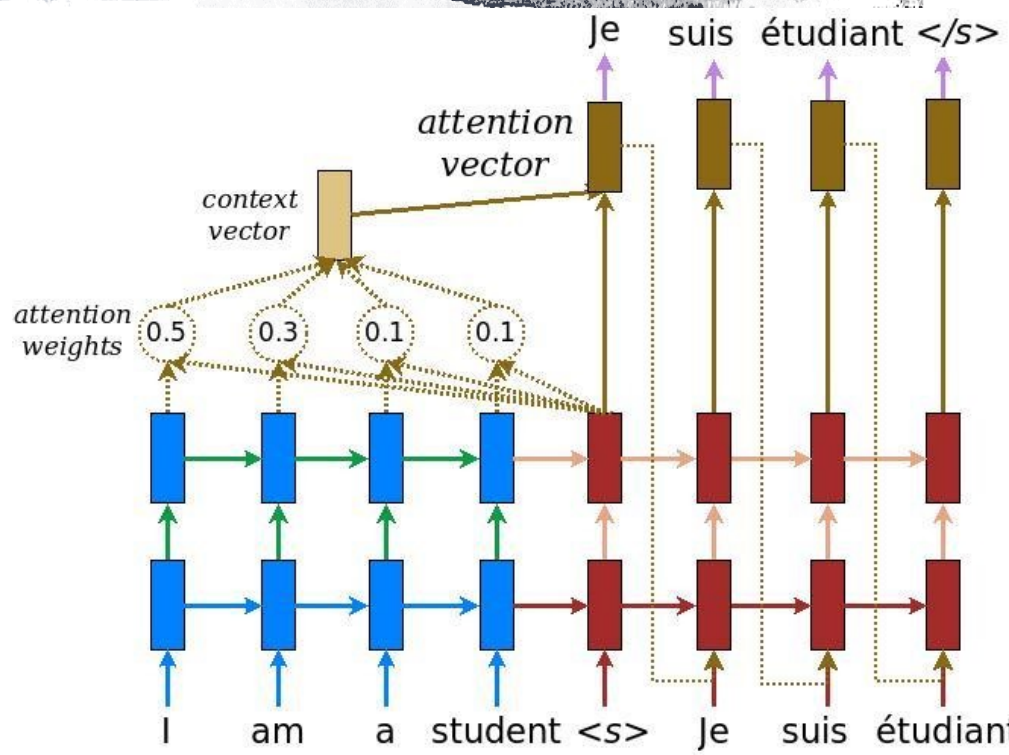
When we are making a local decision to output something, look differentially at the different parts of the input to create a context vector

- Model: what is the probability that this part of the input is important?
- Use softmax to determine probability distribution of using history vector

Attention can be used either with recurrent or non-recurrent networks

Attention example (Machine Translation)

- Model shown is an “encoder-decoder” model
- Input (blue) encodes a sentence; each time step of recurrent model
- For every output (decoded) word, we learn probability of which input words are likely to contribute to the correct translation.



https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

Transformers

Transformers are a non-recurrent model that use attention over sequences.

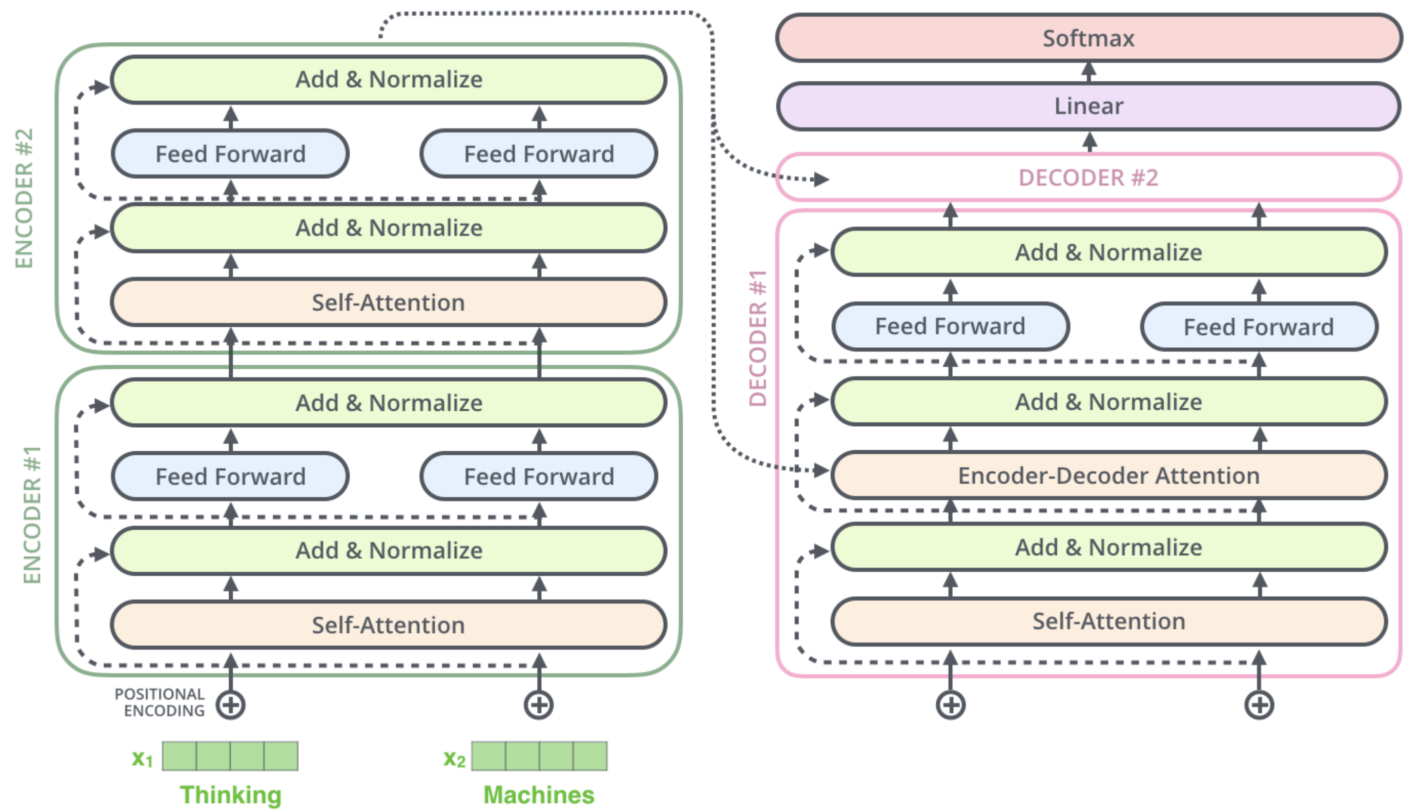
- Popularized by paper “Attention is All You Need” (Vaswani et al 2017)

Also uses “self attention” in the encoder to tell what parts are important for the history vector

Encoders and decoders are stacked to create different levels of representations

https://doi.org/10.48550/arXiv.1706.03762

Transformers



http://jalammar.github.io/illustrated-transformer/

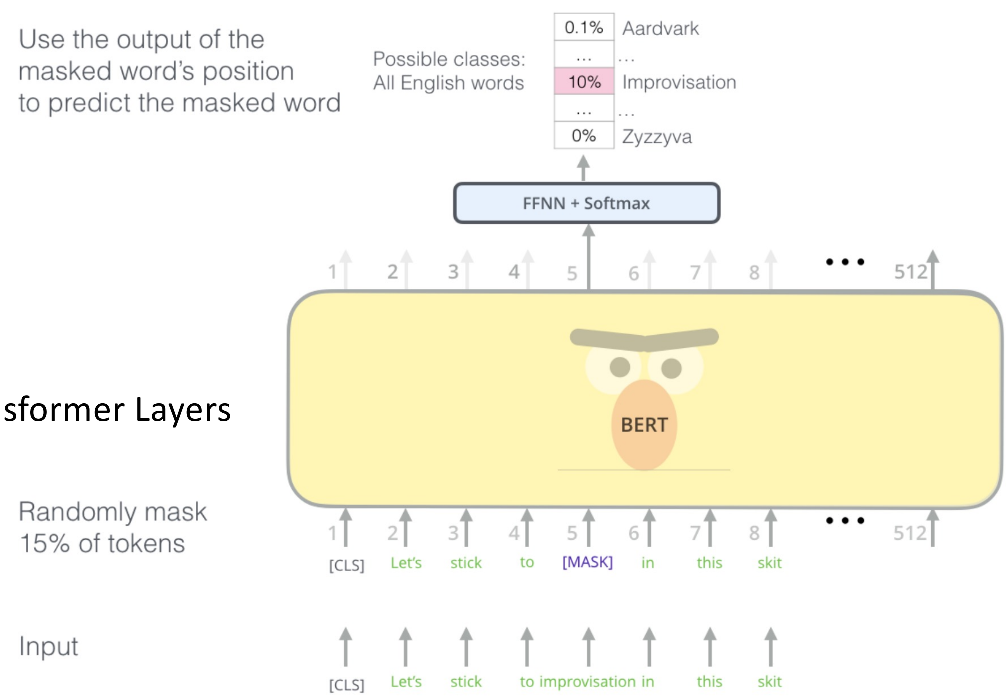
Application: BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers)

(Devlin et al 2018)

BERT (and similar models) are often used for transfer learning

Fine-tuning is one type of transfer learning – take pretrained net and train on new task (e.g. sentiment analysis)

12 Transformer Layers

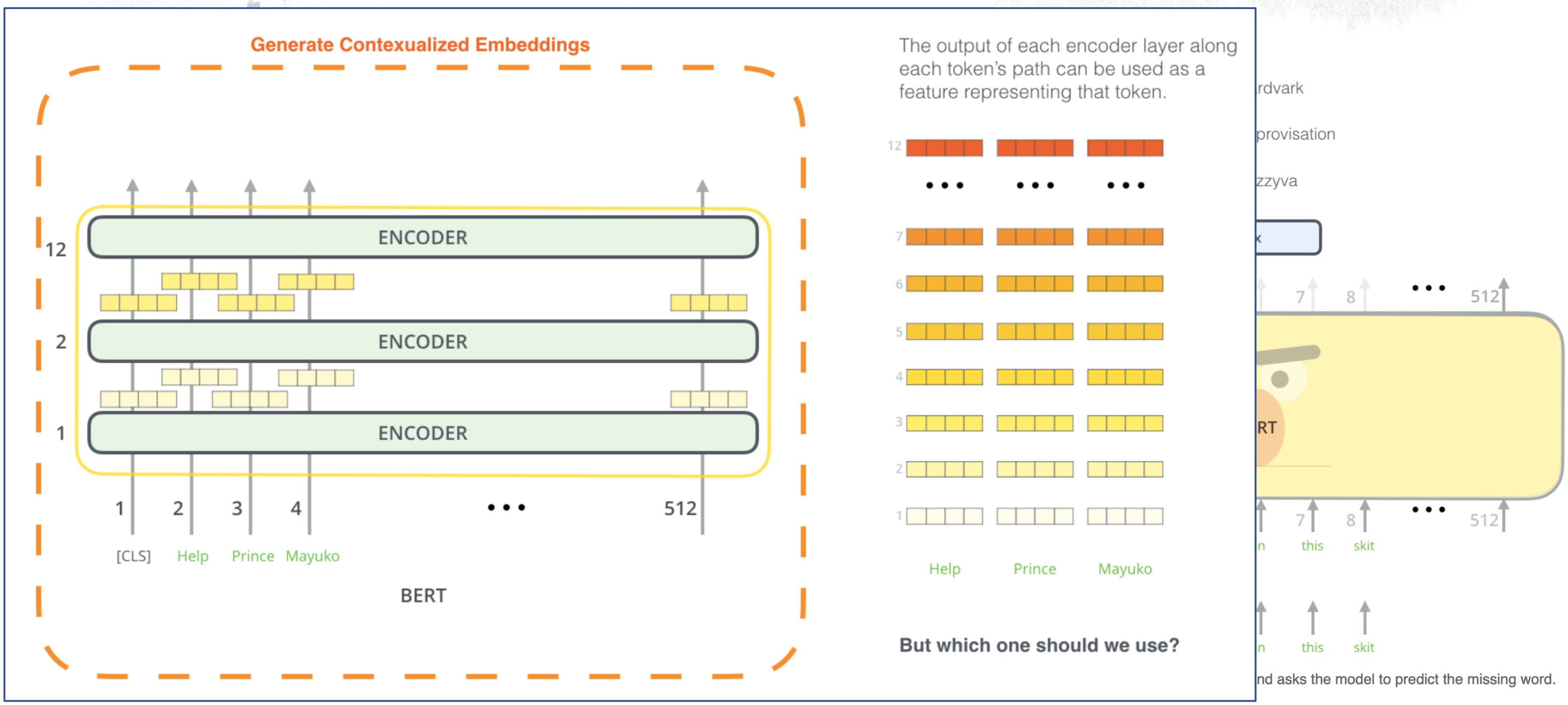


BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

https://doi.org/10.48550/arXiv.1810.04805

http://jalammar.github.io/illustrated-bert/

Application: BERT (Bidirectional Encoder Representations from Transformers)



http://jalammar.github.io/illustrated-bert/

Student-Teacher Learning

Up to now, "supervision" has only meant giving the correct label/answer to a network.

However, can we get networks to start training each other?

- Often, might have a "big" model that can handle most cases, but might not be what you want to use
  - Might want to make the model smaller
  - Might want to make the model more robust
  - Might want to combine several models
- Knowledge distillation (KD), or student-teacher learning, can be used to have one model help train another

Examples of Knowledge Distillation

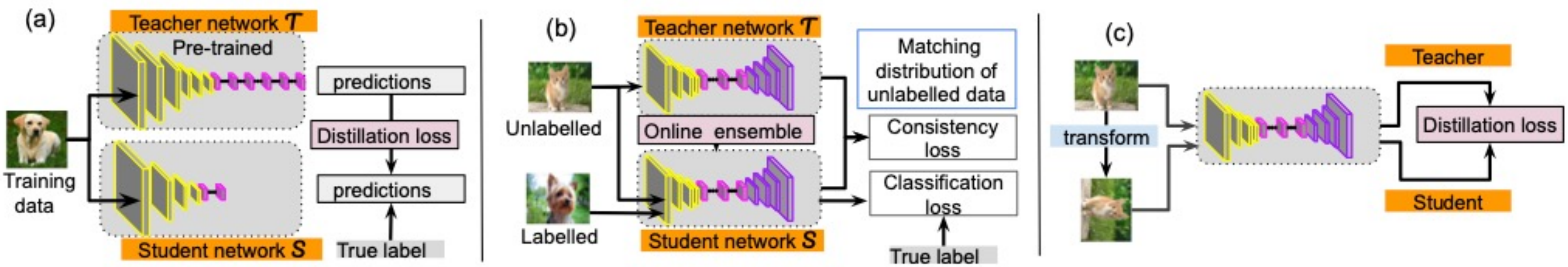


Fig. 1. Illustrations of KD methods with S-T frameworks. (a) for model compression and for knowledge transfer, *e.g.*, (b) semi-supervised learning and (c) self-supervised learning.

Model Compression | Model Combination | Model Robustness

Wang & Yoon TPAMI 2021, Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks

Generative Adversarial Networks

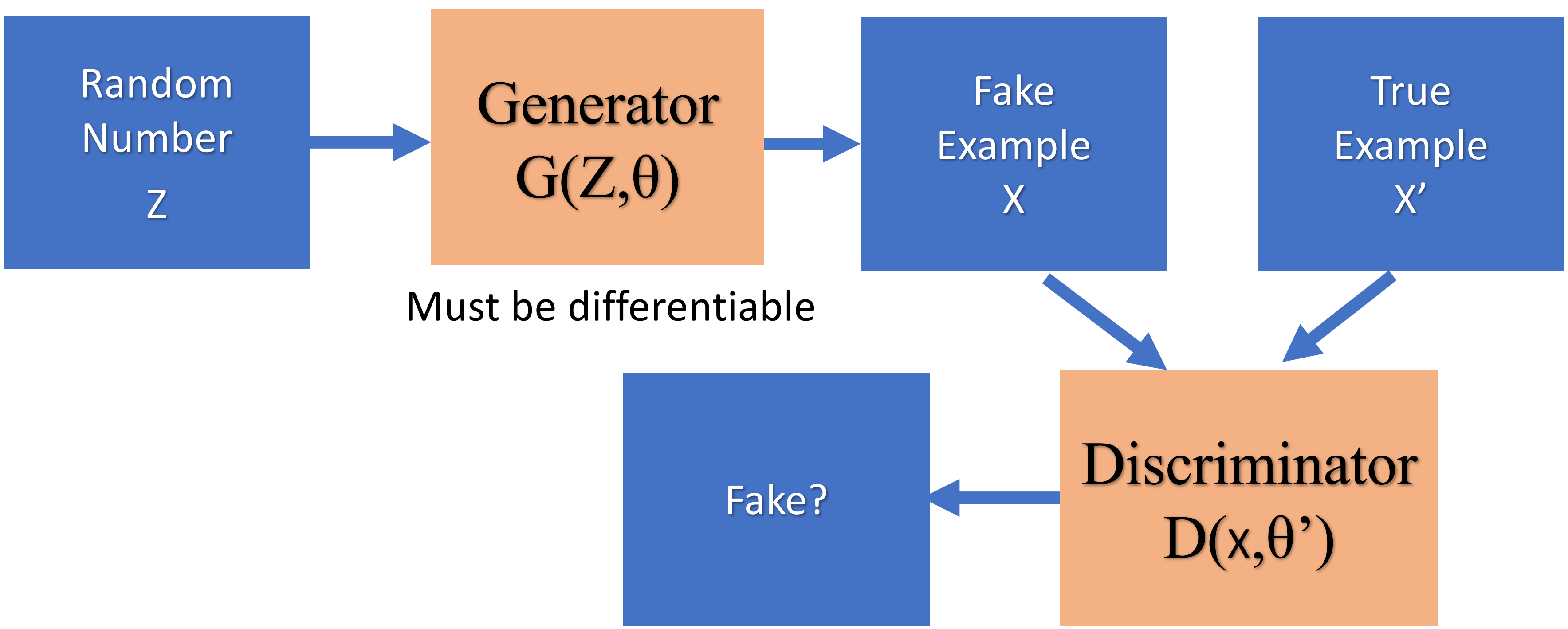
GANs use **adversarial** training that set two models against each other:

- G: Generator model that makes fake examples
- D: Discriminator that tries to tell real from fake models

Generative Adversarial Learning

$$\mathcal{L}^{G}(\boldsymbol{X}) = -\,\mathcal{L}^{D}(\boldsymbol{X})$$

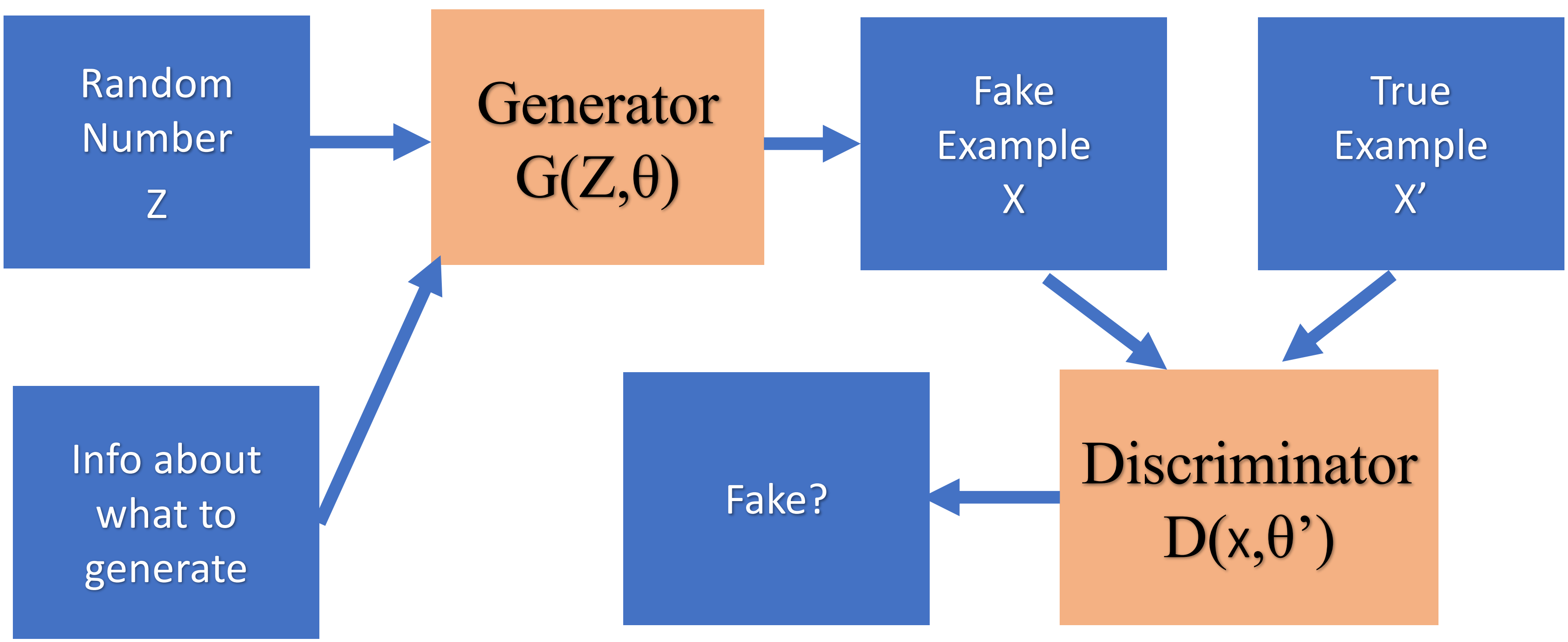
Minimax Game



$$\mathcal{L}^{D}(\boldsymbol{X}) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{X}\sim P_{data}} \log D(\boldsymbol{X}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{Z}} \log\left(1 - D\left(G(Z)\right)\right)$$

Goodfellow 2019

Conditional Generative Adversarial Learning



Goodfellow 2019

What can you do with GANs?

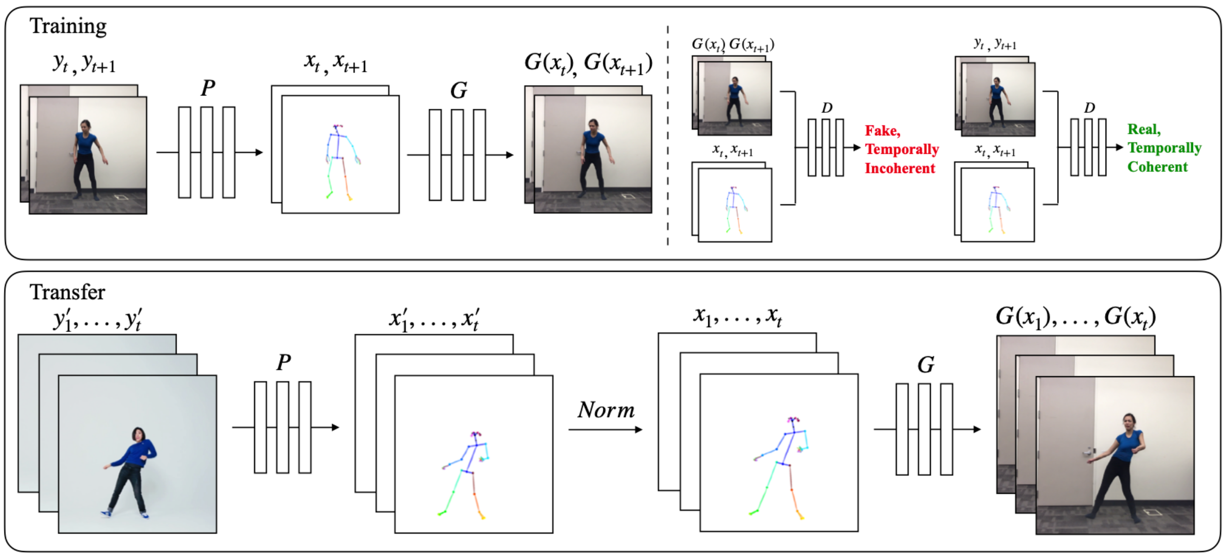
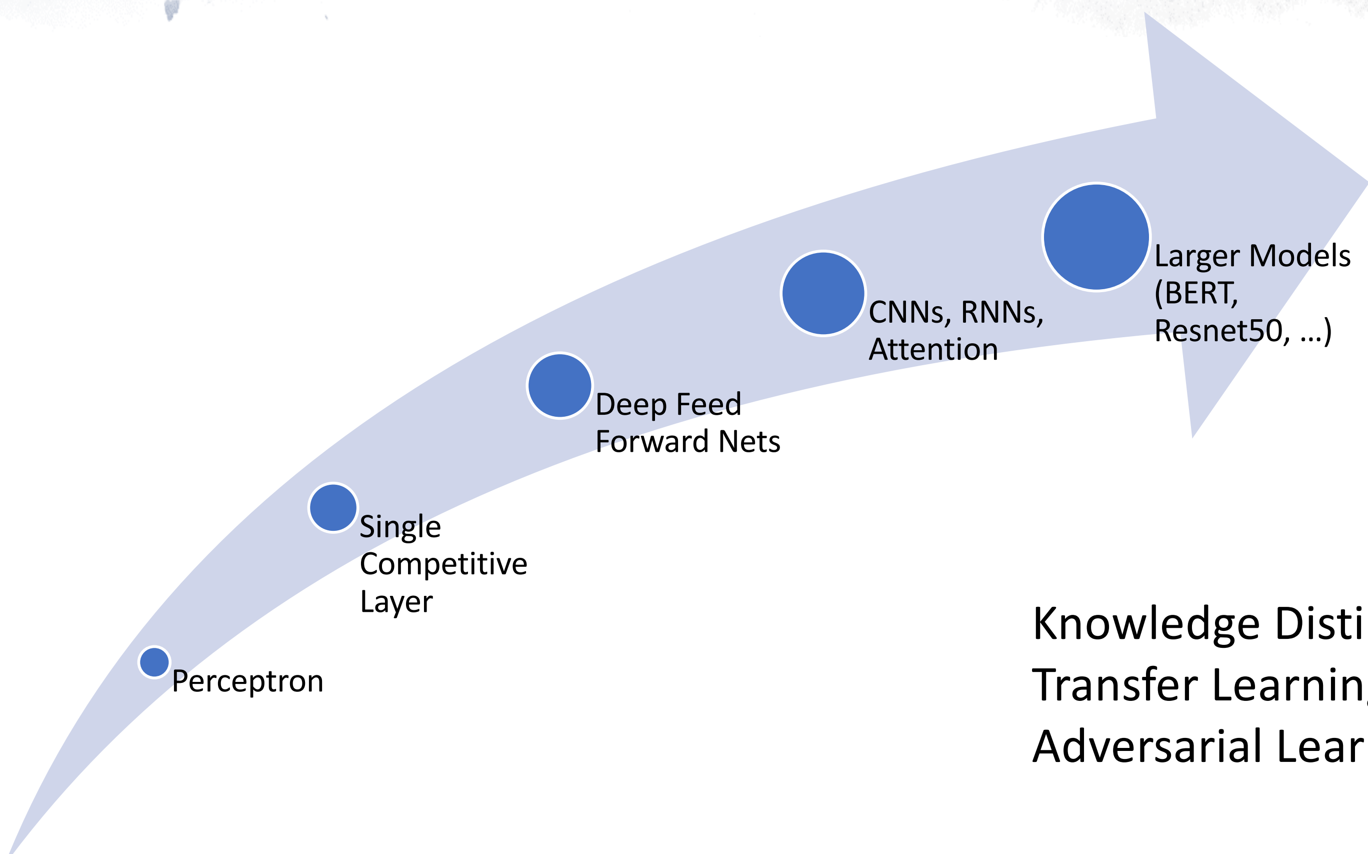


Figure 3: (Top) **Training**: Our model uses a pose detector P to create pose stick figures from video frames of the target subject. We learn the mapping G alongside an adversarial discriminator D which attempts to distinguish between the "real" correspondences $(x_t, x_{t+1}), (y_t, y_{t+1})$ and the "fake" sequence $(x_t, x_{t+1}), (G(x_t), G(x_{t+1}))$. (Bottom) **Transfer**: We use a pose detector P to obtain pose joints for the source person that are transformed by our normalization process $Norm$ into joints for the target person for which pose stick figures are created. Then we apply the trained mapping G.

Chan 2018 – Everybody Dance Now https://www.youtube.com/watch?v=PCBTZh41Ris

Everybody Dance Now (Chan 2018)

Summary



Knowledge Distillation
Transfer Learning
Adversarial Learning