Machine Learning

Session 11 Deep Learning 2

- Training and Validation (general)
- Practical training of deep learning models
- Some popular architectures
  - Convolutional Neural Networks (CNN)
  - Recurrent Neural Networks (RNN)
  - Autoencoders
- Recent Applications

Bibliography: Chap 5 of C. Bishop book
: DL book and slides from I. Goodfellow

Training and Validation

Training and Validation

- We can divide the set of all **training** data into disjoint sets: **training set** and **validation set.** Ex: **80%** and **20%**
- Assume hyper-parameters λ that determine how strong is our regularization, e.g., as in ridge regression
- Use the first set (**training set**) to estimate the weights $\mathbf{w}$ for different values of hyper-parameters λ
- Use the second set (**validation set**) to estimate the best λ, by evaluating how well the classifier does in this second set

Training and Validation

- **Cross-Validation (CV)**: a validation technique to estimate model generalization error

- Leave-*p*-out CV
  - Use only *p* examples from the training data as **validation set**
  - Use the rest as **training set**
  - Complexity scales exponentially with *p*

- Leave-one-out CV
  - Use only *1* example from the training data as **validation set**
  - Use the rest as **training set**

Training and Validation

- **Cross-Validation (CV)**: a validation technique to estimate model generalization error

- *K*-fold CV
  - Partition the training data into *k* groups (folds) of equal size
  - Select *K-1* groups as **training set** and one group as **validation set**
  - Repeat *K* times selecting each time a different group for validation
  - The *K* results can be combined to produce
    - A single estimation of the **generalization error**
    - The best value of the **hyper-parameters**
    - **NOT** the values of the parameters themselves!

Training and Validation

Assessing the quality of a trained model

Confusion (error) Matrix

Binary classification task

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Training and Validation

Assessing the quality of a trained model

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Precision

Positive Predictive value

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathbf{FP}}$$

Sensitivity / Recall

True positive rate

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathbf{FN}}$$

F1 Score

Harmonic Mean of Precision and Recall

$$\mathbf{F_1} = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}}$$

Training and Validation

Assessing the quality of a trained model

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Sensitivity / Recall

True positive rate

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathbf{TP + FN}}$$

Specificity / Selectivity

True negative rate

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathbf{TP + FN}}$$

Training and Validation

Assessing the quality of a trained model

Example:

		Predicted condition	
	Total 8 + 4 = 12	**Cancer** 7	**Non-cancer** 5
Actual condition	**Cancer** 8	6	2
	Non-cancer 4	1	3

Sensitivity / Recall TPR $= \frac{\mathrm{TP}}{\mathrm{P}} =$

Specificity / Selectivity TNR $= \frac{\mathrm{TN}}{\mathrm{N}} =$

Precision PPV $= \frac{\mathrm{TP}}{\mathrm{TP}+\mathbf{FP}} =$

F1 Score $\mathbf{F_1} = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} =$

Training and Validation

Assessing the quality of a trained model

Example:

		Predicted condition	
	Total 8 + 4 = 12	**Cancer** 7	**Non-cancer** 5
Actual condition	**Cancer** 8	6	2
	Non-cancer 4	1	3

Sensitivity / Recall TPR $= \frac{\mathrm{TP}}{\mathrm{P}} = \frac{6}{8} = 0.75$

Specificity / Selectivity TNR $= \frac{\mathrm{TN}}{\mathrm{N}} = \frac{3}{4} = 0.75$

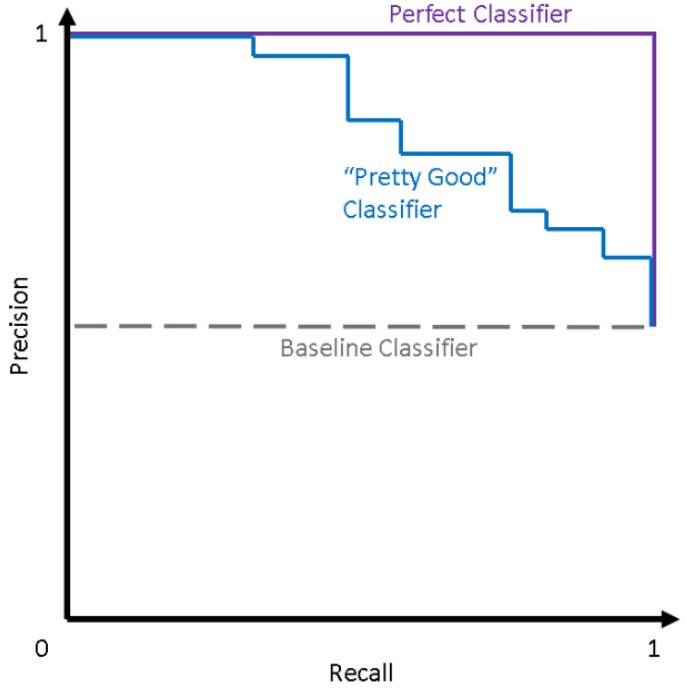
Precision PPV $= \frac{\mathrm{TP}}{\mathrm{TP}+\mathbf{FP}} = \frac{6}{7} = 0.88$

F1 Score $\mathbf{F_1} = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} = 0.81$

Training and Validation

- **Precision-Recall curve**

Shows the **precision** vs the **recall** as threshold value for classification varies



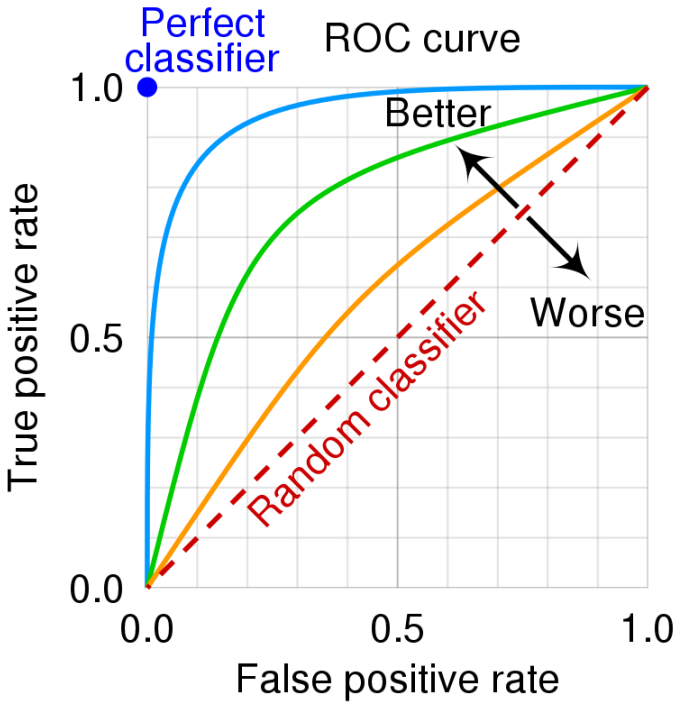
Baseline Classifier assigns all examples to positive class

  - **AUC-PR:** area under the precision-recall curve (the higher, the better)

Training and Validation

- **Receiver Operating Characteristic (ROC) curve**

Shows the **sensitivity** vs (1-**specificity**) as threshold value varies

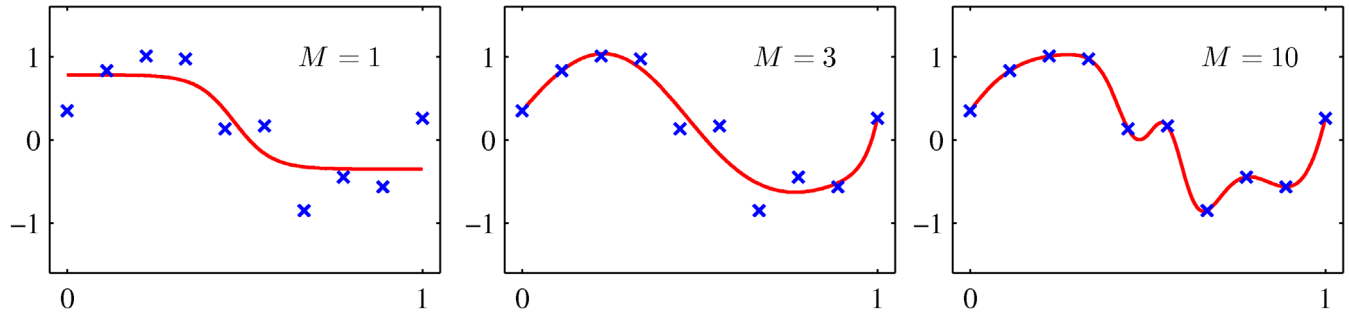


  - **AUC:** area under the (ROC) curve (the higher, the better)

Model Selection

Model Selection

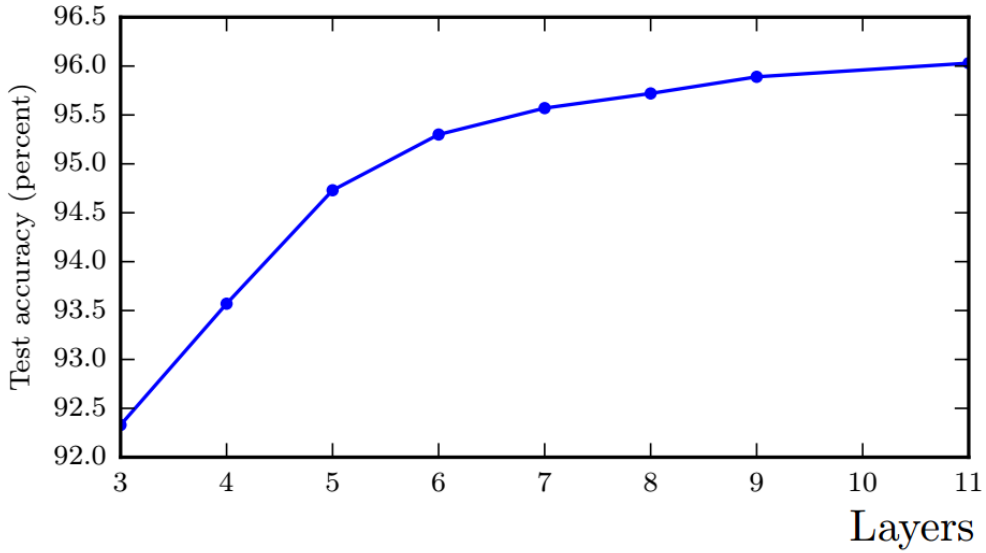
- *Model* in our case: error function, number of layers, neurons/layer.
- *Best generalization performance*: optimum balance between under-fitting and over-fitting
- Example: network output trained with sinusoidal data as a function of the number of hidden units M in a two-layer Network



- The *number of hidden layers and units per layer determines the model complexity*

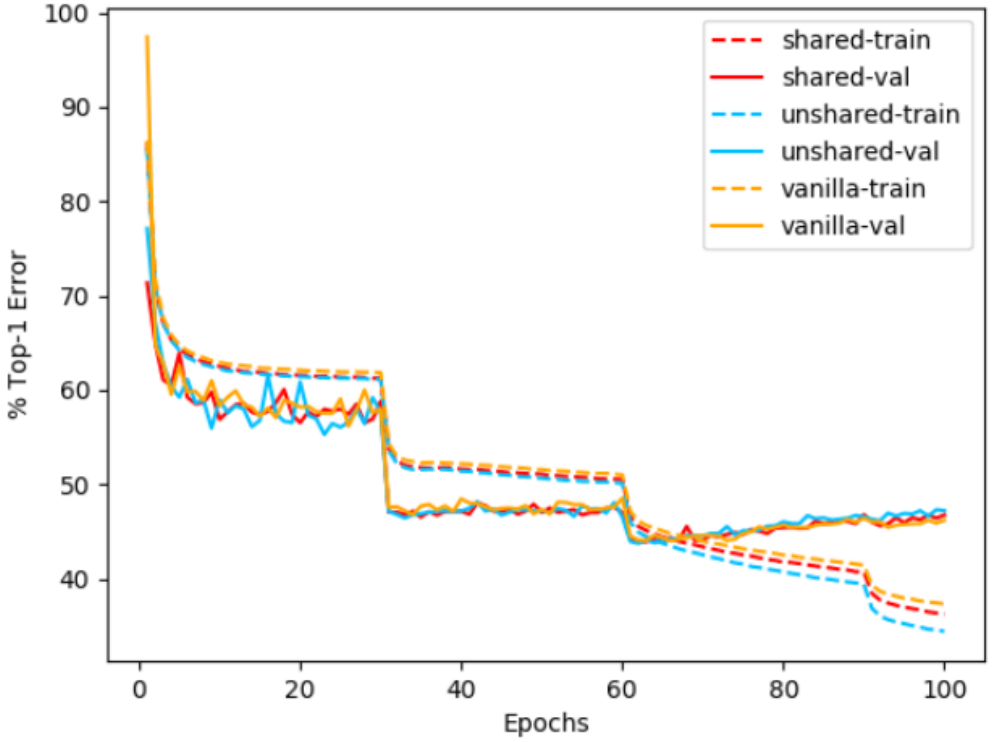
Model Selection

- Why deep networks are better?
  - Shallow networks may need (exponentially) more width
  - Shallow networks may overfit more
  - Deep networks exploit **compositionality**



Model Selection

- How to choose the learning rate?
  - Not a clear answer!
  - Exponentially decreasing the learning rate after a fixed number of epochs works well in practice



Regularization of a Neural Network

- Weight decay (L2-norm regularization)

  – Choose a relatively large value of units and layers
  – Control complexity by adding a regularization term to the error function

$$E_R(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

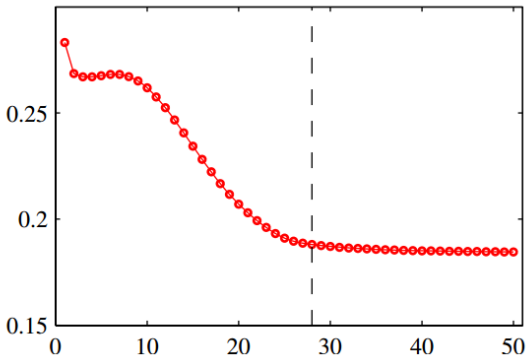
  – Large values of λ penalize more large magnitude weights
  – Small values of λ penalize less large magnitude weights

  – We already saw it previously (Ridge regression)

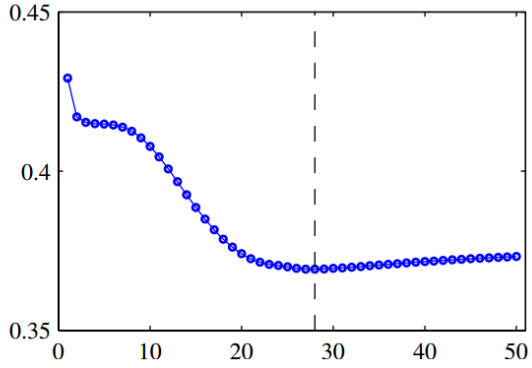
Regularization of a Neural Network

Early stopping

- During learning, error in the training dataset does not increase
- In contrast, error in the validation dataset shows a decrease at first, followed by an increase as the network start to over-fit



training error

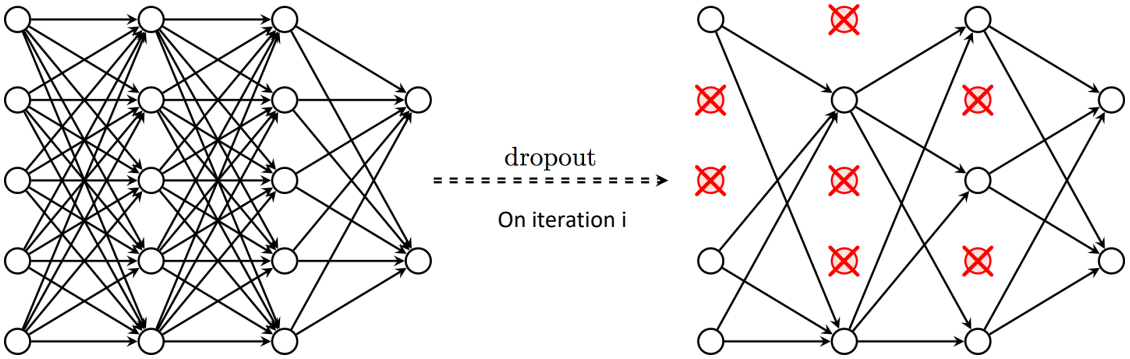


validation error

- Training should be stopped at the point shown by the vertical dashed lines
- Similar to weight decay for quadratic error functions

Regularization of a Neural Network

Dropout



- On a given training iteration, sample a binary mask (i.e., each element is 0 or 1) for all the input and hidden units in the network. This Bernoulli parameter, p , is a Hyperparameter. Typical values are p=0.8 for input and p=0.5 for hidden units
- Apply (i.e., multiply) the mask to all units. Then perform the forward pass and backwards pass, and parameter update.
- At test time, use the *mean* net that has all outgoing weights *halved*
- An implicit form of model averaging
- Dropout is not as effective for small datasets

Regularization of a Neural Network

Dataset augmentation

- Neural networks generalize better when more training data is available. One way to increase the dataset size is to generate fake data (with appropriate statistics) and add it to the training set

- This is particularly useful for image and object recognition. The dataset augmentation could be translating, rotating, or scaling the input images. This often results in better generalization in these tasks

- Augmenting the data by perturbing it intentionally resulted in better robustness

- We may also consider the injection of input noise as dataset augmentation

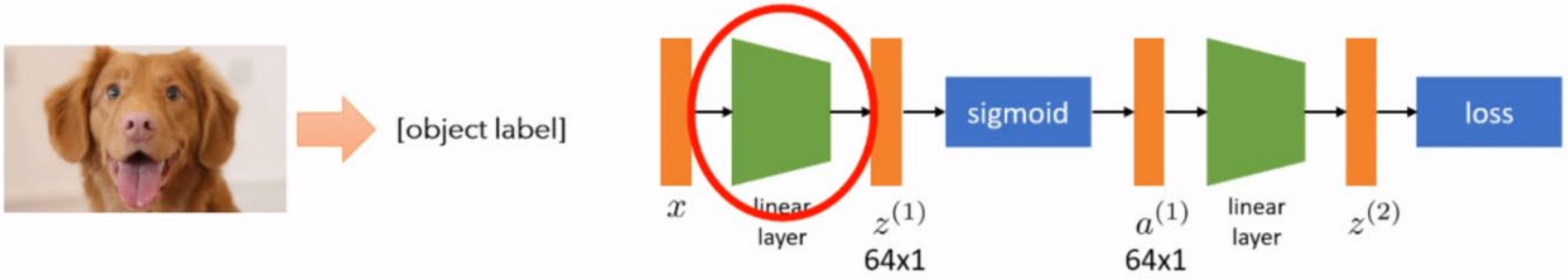
Convolutional Neural Networks

Convolutional Neural Networks

- A particularly successful architecture, inspired by neuroscience
- Scale up to very large images / video sequences: in general, **regular grid-like structures**
- Two main ideas
  - **Sparse connectivity** (few connections are active)
  - **Parameter sharing**
- Main benefits:
  - Computationally efficient: applicable to huge inputs
  - Generalize a lot better (invariant to spatial translation of the input)

Convolutional Neural Networks

Motivation



Limitations:

- Require many parameters

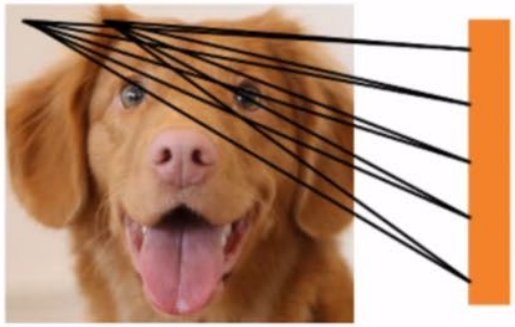


image is $128 \times 128 \times 3 = 49{,}152$

$z^{(1)}$ is 64-dim

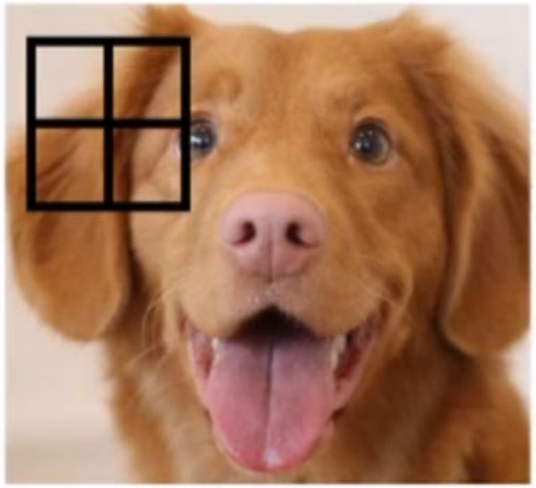
$64 \times 49,152 \approx 3,000,000$

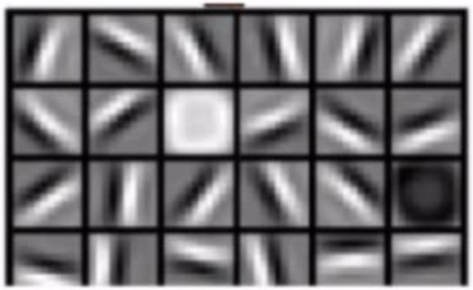
We need a better way!

- Does not exploit invariances such as shifts

Convolutional Neural Networks

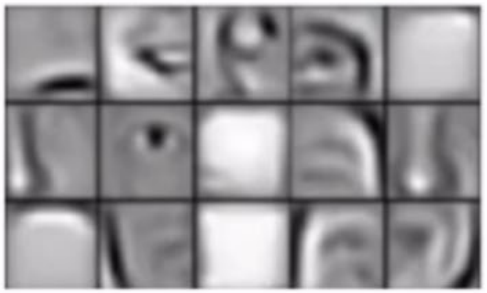
Motivation





Layer 1:

edge detectors?



Layer 2:

ears? noses?

- Many features are local
- Features are composed hierarchically
- The same features can appear in different patches

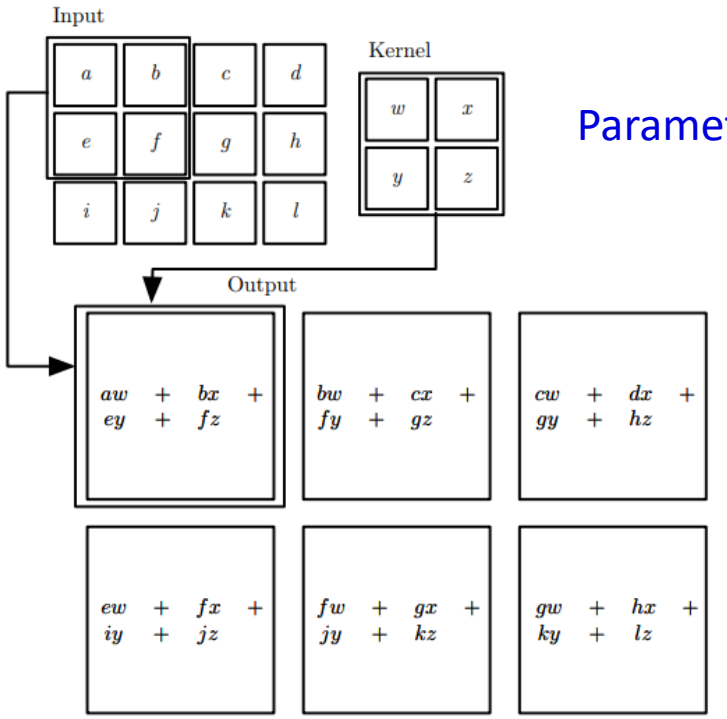
Convolutional Neural Networks

- **Key idea**: replace **matrix multiplications** of with **convolutions** (also a linear operation)

- Everything else stays the same
  - Training method: Back-propagation
  - Etc…

Convolutional Neural Networks

- 2D Convolution



Parameters? only w x y z

  - Input is fixed (big)
  - Kernel is learned (small)

- Colored Images require 3D Convolution (RGB)

See also: CS231n Convolutional Neural Networks for Visual Recognition

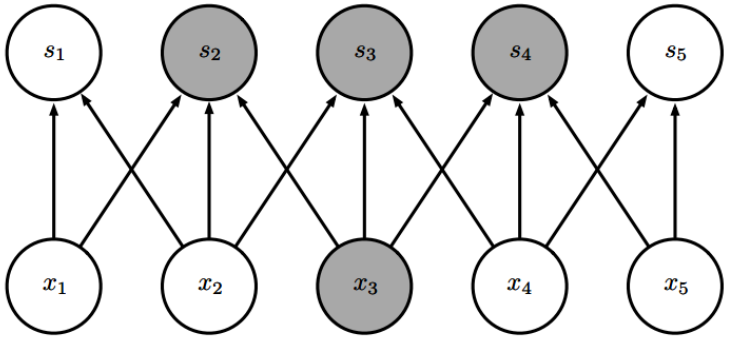
Convolutional Neural Networks

Sparse Connectivity

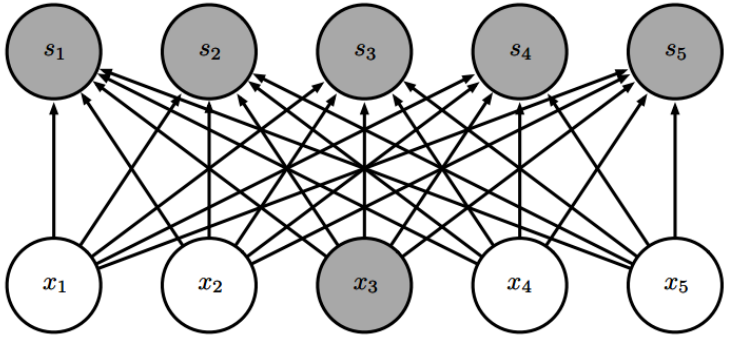
The same input is only *seen* by a few, **constant** number of units
Ex: kernel size of 3

In standard NNs the number of parameters is **quadratic**

Sparse connections due to small convolution kernel



Dense connections

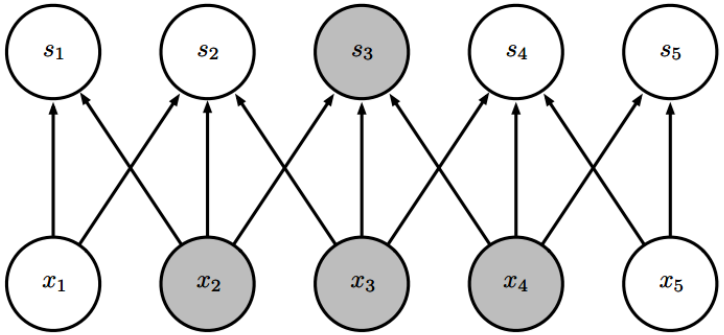


Convolutional Neural Networks

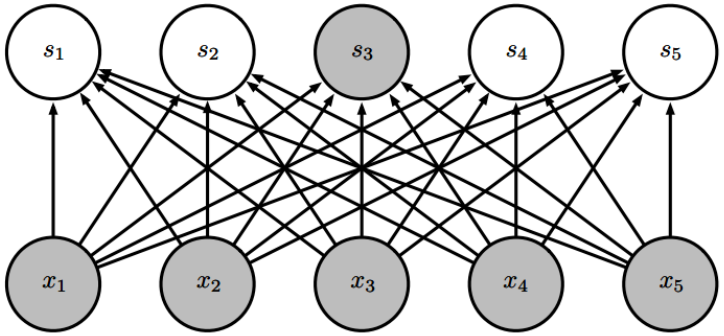
Sparse Connectivity

Similarly, units only *see* a small part of the input

Sparse connections due to small convolution kernel



Dense connections

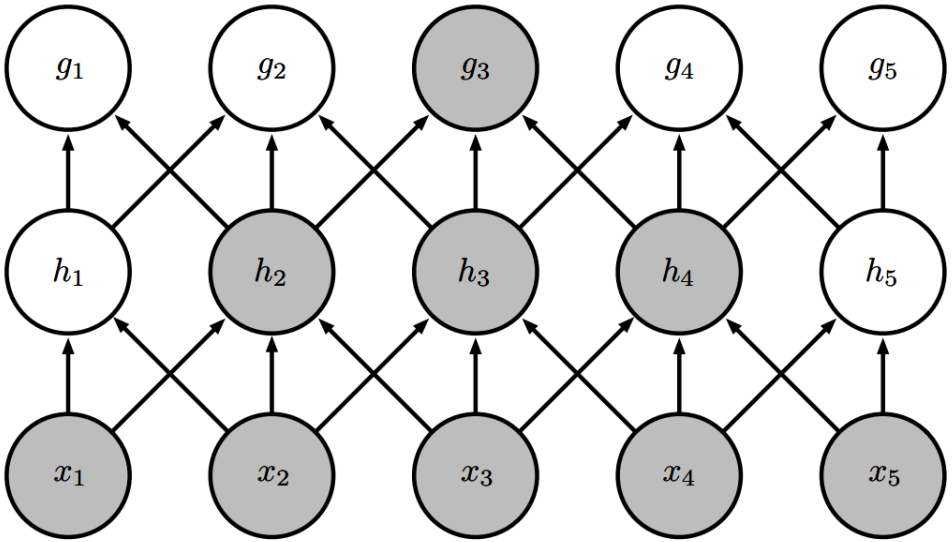


Convolutional Neural Networks

Growing receptive fields

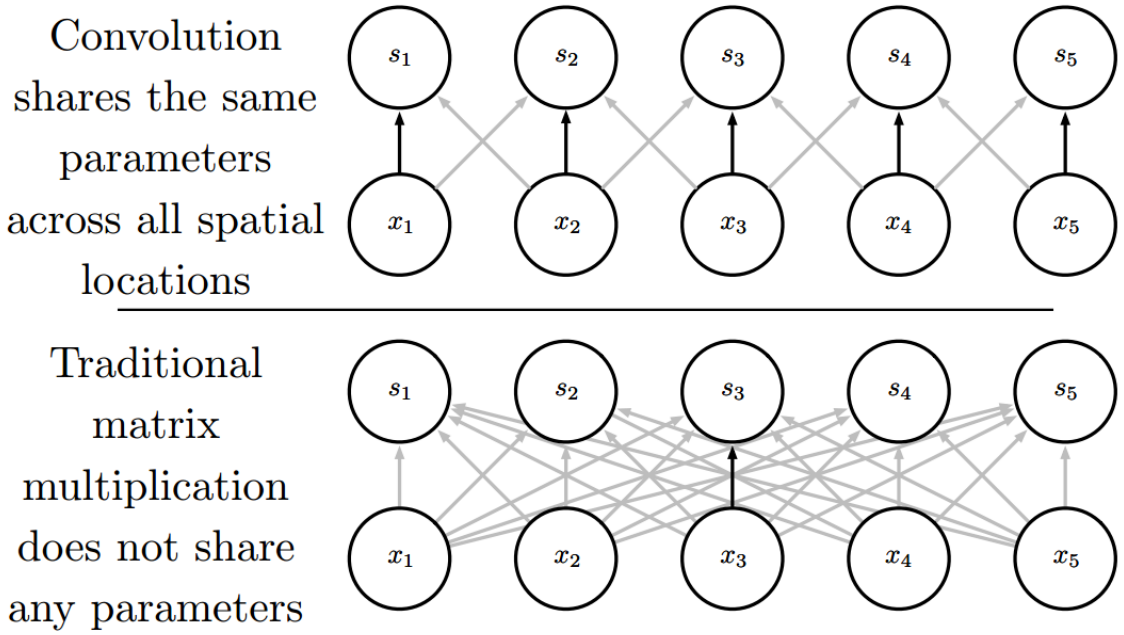
Hierarchical representations

Using **multiple layers** allows many units to see many parts of the input



Convolutional Neural Networks

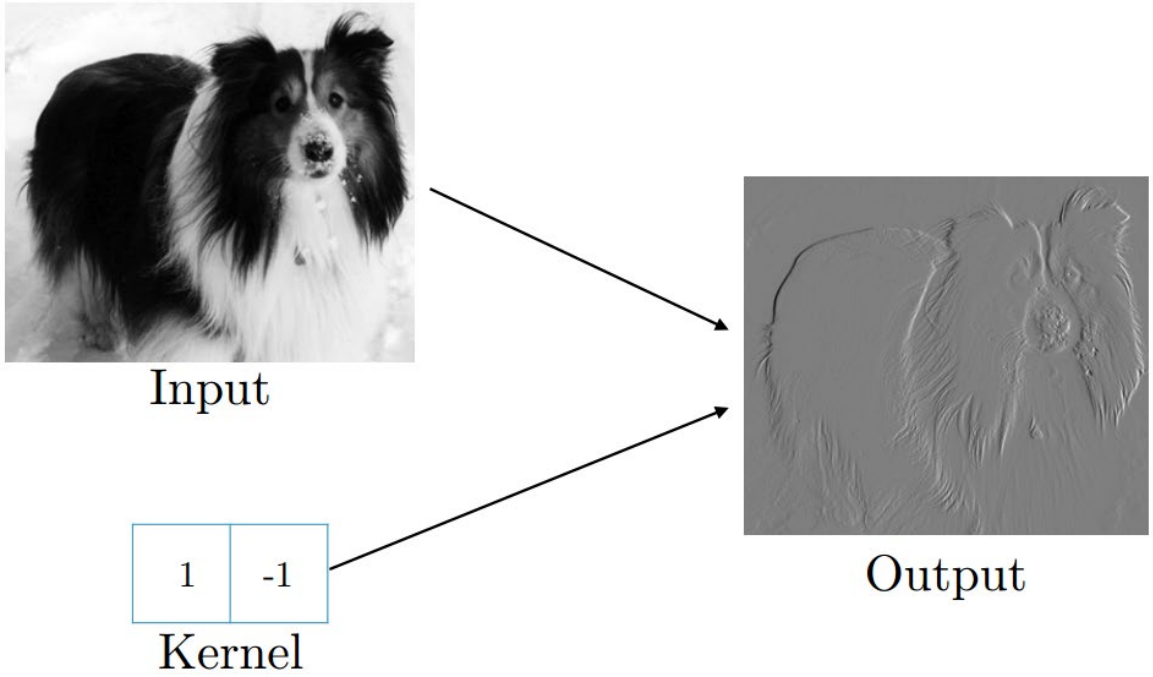
Parameter Sharing



(bold arrows are an example of a shared parameter)

Convolutional Neural Networks

Example: Edge detection by convolution



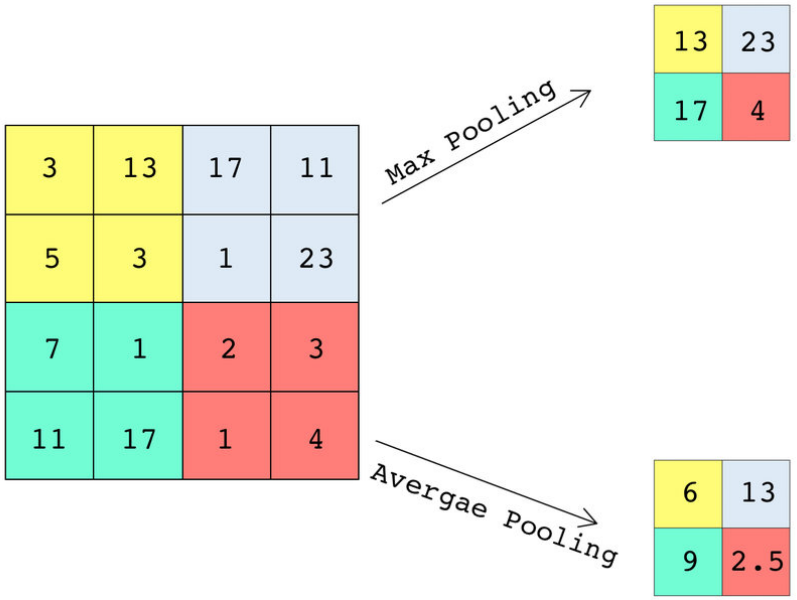
1 responds for light pixels, -1 for dark pixels: high resultant values in contrast pixels.

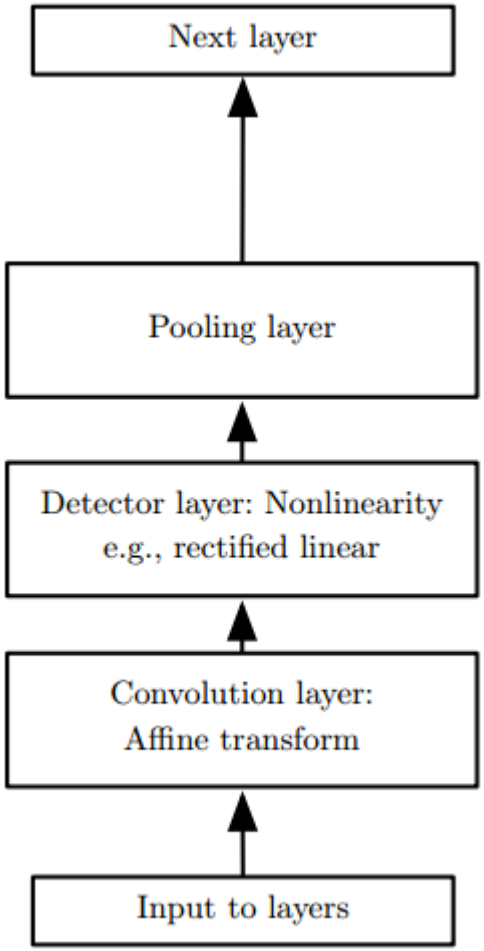
CNNs learn these features internally

Convolutional Neural Networks

Max Pooling and translation invariance

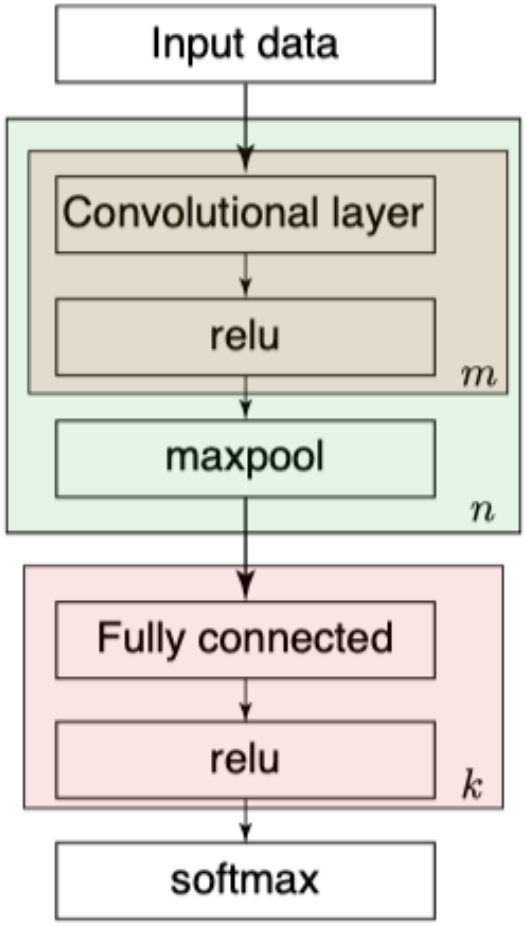
Small translations in the input
do not affect the outputs

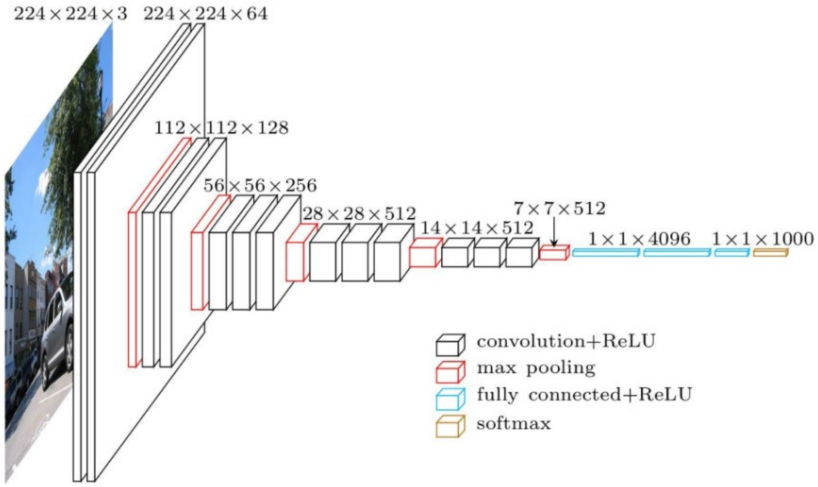


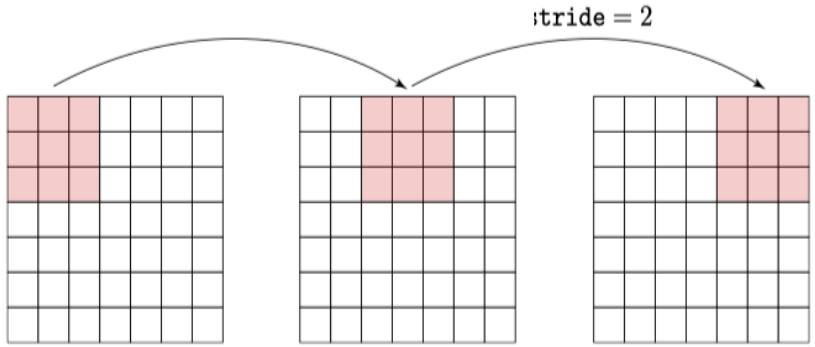


Convolutional Neural Networks

- A typical CNN:







The stride defines how much the filter moves in the convolution.

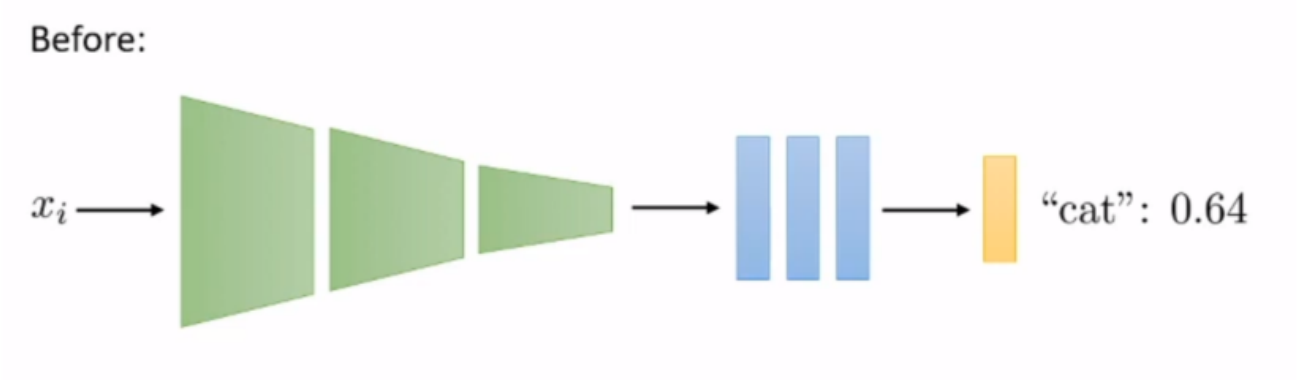
Examples:

- AlexNet (Krizhevsky et al., 2012) and ZFNet (Zeiler & Fergus, 2013)
- VGGNet and small filters (Simonyan & Zisserman, 2014)
- GoogLeNet and inception layers (Szegedy et al., 2014)
- ResNet and residual layers (He et al., 2015)
- FractalNet and the importance of reducing effective depth (Larsson et al., 2017)

Recurrent Neural Networks

Recurrent Neural Networks

What if we have variable-size inputs?



Now:

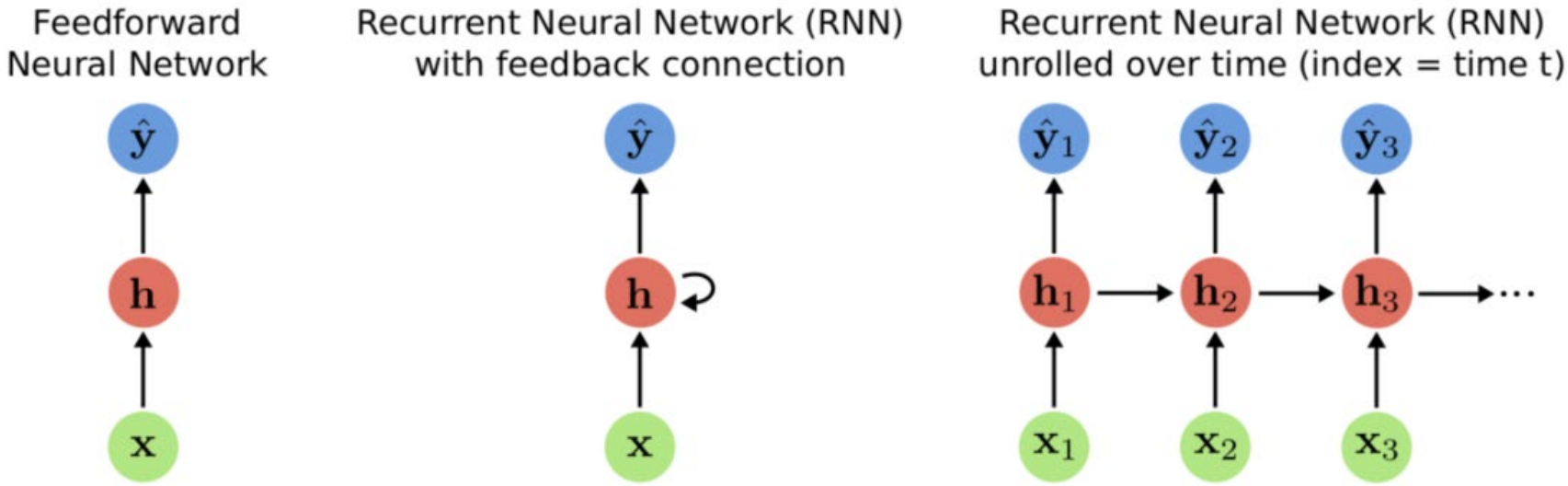
$$\mathbf{x}_1 = (\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \mathbf{x}_{1,3}, \mathbf{x}_{1,4})$$
$$\mathbf{x}_2 = (\mathbf{x}_{2,1}, \mathbf{x}_{2,2}, \mathbf{x}_{2,3})$$
$$\mathbf{x}_3 = (\mathbf{x}_{3,1}, \mathbf{x}_{3,2}, \mathbf{x}_{3,3}, \mathbf{x}_{3,4}, \mathbf{x}_{3,5})$$
$$\ldots$$

Examples

- Classifying sentiment for a phrase (sequence of words)
- Recognizing phoneme from sound (sequence of sounds)
- Classifying the activity in a video (sequences of images)

Recurrent Neural Networks

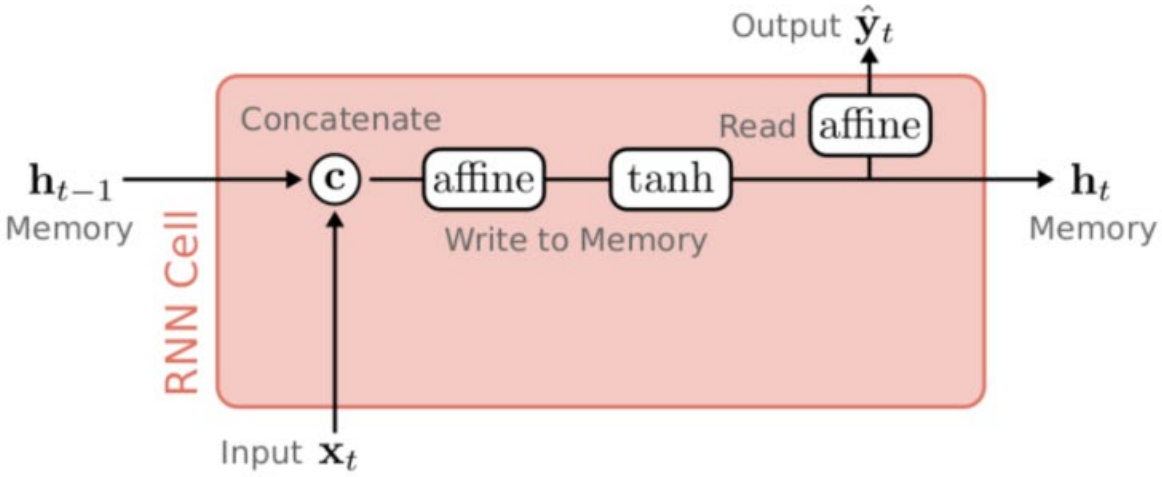
- An output $\hat{\mathbf{y}}_t$ can be computed at each time-step



Recurrent Neural Networks (RNNs):

- Core idea: update **hidden state** $\mathbf{h}$ based on input and previous hidden state using same update rule (same/shared parameters) at each **time step**
- Allows for processing **sequences of variable length**, not only fixed-sized vectors
- **Infinite memory:** $\mathbf{h}$ is function of all previous inputs (long-term dependencies)

Recurrent Neural Networks



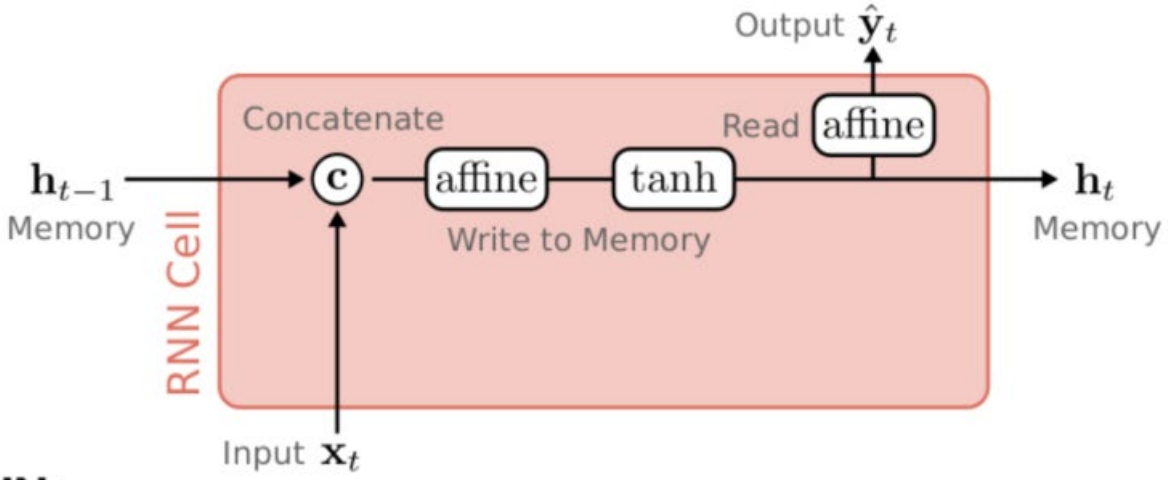
General Form:

$$\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\hat{\mathbf{y}}_t = f_y(\mathbf{h}_t)$$

- We use t as the **time index** (in feedforward networks we used i as layer index)
- General form does not specify the form of the hidden and output mappings
- Important: f_h and f_y **do not change over time**, unlike in layers of feedforward net

Recurrent Neural Networks



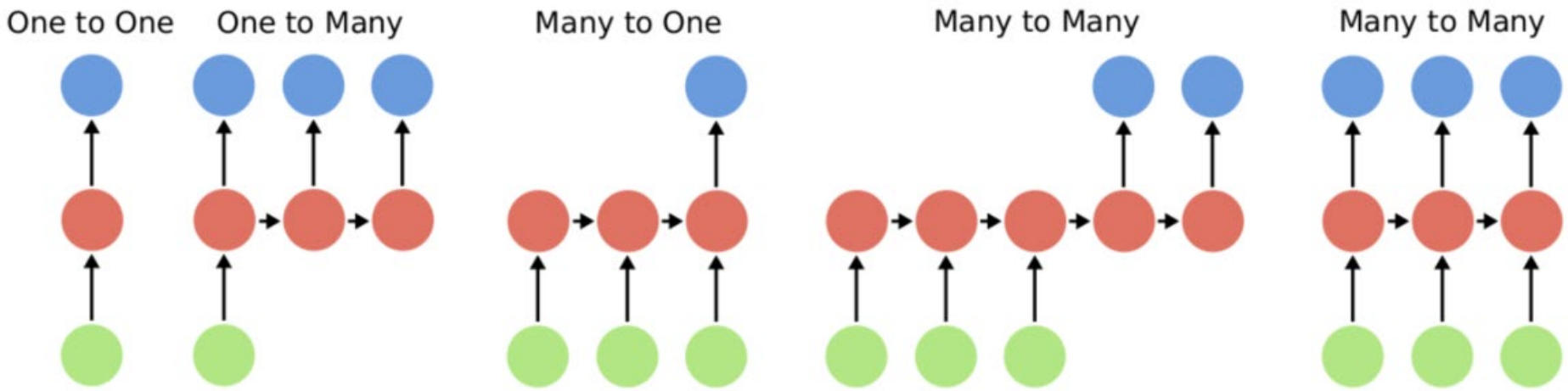
Single-Layer RNN:

$$\mathbf{h}_t = \tanh(\mathbf{A}_h \mathbf{h}_{t-1} + \mathbf{A}_x \mathbf{x}_t + \mathbf{b})$$

$$\hat{\mathbf{y}}_t = \mathbf{A}_y \mathbf{h}_t$$

- Hidden state $\mathbf{h}_t$ = linear combination of input $\mathbf{x}_t$ and previous hidden state $\mathbf{h}_{t-1}$
- Output $\hat{\mathbf{y}}_t$ = linear prediction based on current hidden state $\mathbf{h}_t$
- $\tanh(\cdot)$ is commonly used as activation function (data is in the range $[-1, 1]$)
- Parameters $\mathbf{A}_h$, $\mathbf{A}_x$, $\mathbf{A}_y$, $\mathbf{b}$ are constant over time (sequences may vary in length)

Recurrent Neural Networks

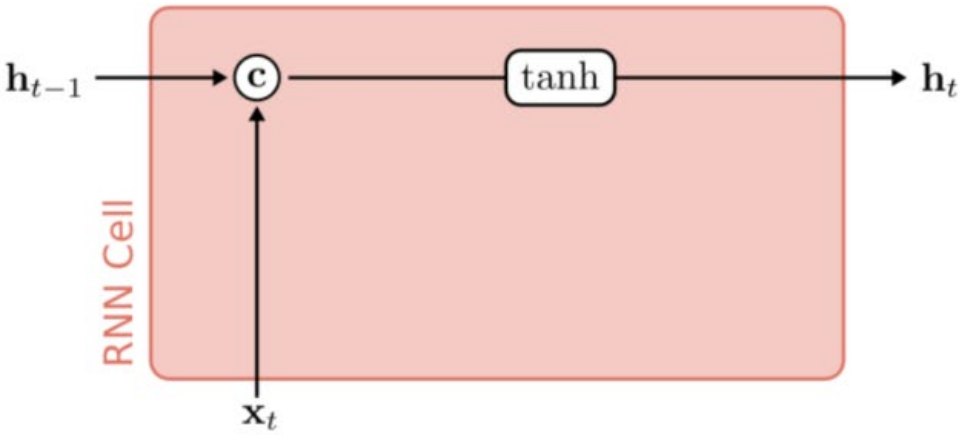


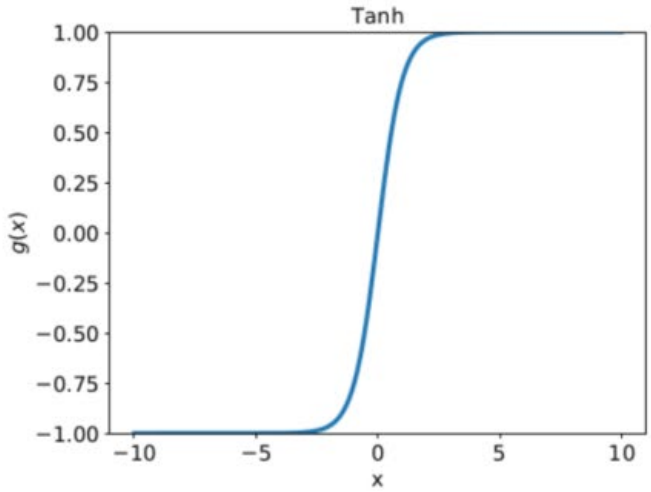
RNNs allow for processing **variable length** inputs and outputs:

- **One to Many:** Image captioning (image to sentence)
- **Many to One:** Action recognition (video to action)
- **Many to Many:** Machine translation (sentence to sentence)
- **Many to Many:** Object tracking (video to object location per frame)
- To determine the length of the output sequence, a **stop symbol** can be predicted

Recurrent Neural Networks

What is the problem with vanilla RNNs? **Vanishing Gradients**



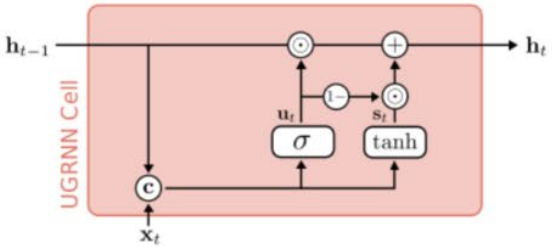


- The state update H_t is modeled using a zero-centered $\tanh(\cdot)$
- $\tanh(\cdot)$ assumes that the processed data is in the range $[-1, 1]$
- Remark: we omit the affine transformations and the output layer for clarity

Recurrent Neural Networks

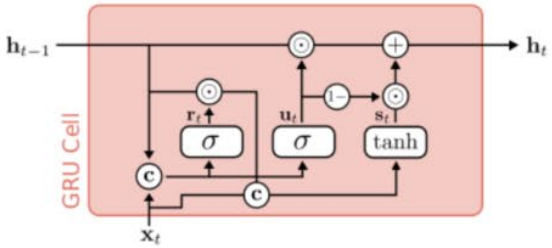
Solution: **Gated Recurrent Units**

UGRNN



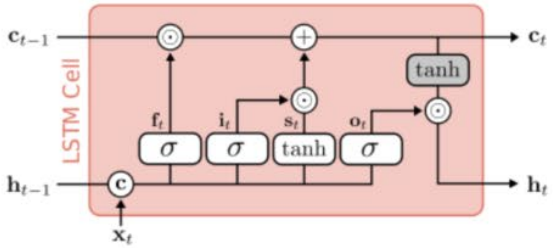
Collins, 2017

GRU



Cho, 2014

LSTM

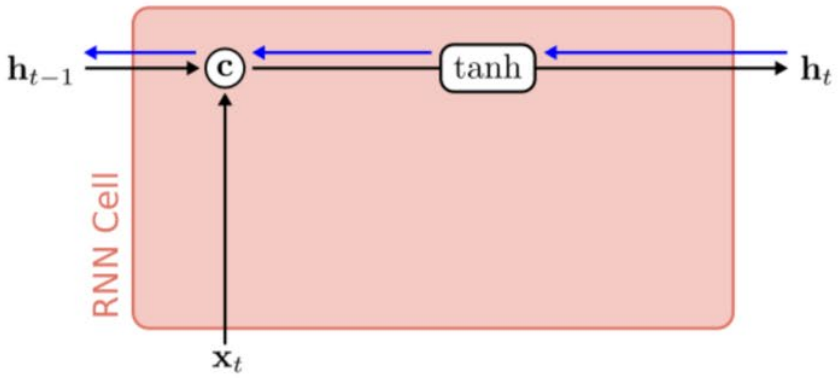


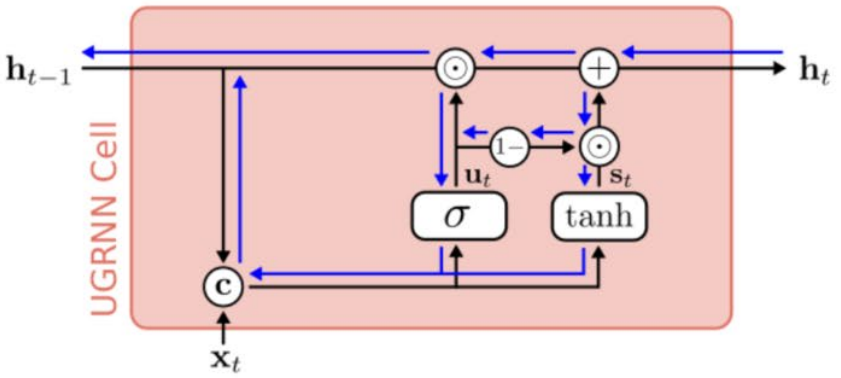
Hochreiter, 1997

- **UGRNN:** Update Gate Recurrent Neural Network
- **GRU:** Gated Recurrent Unit
- **LSTM:** Long Short-Term Memory
- LSTM was the first and most transformative (revolutionized NLP in 2015, e.g. at Google), but also most complex model. UGRNN and GRU work similarly well.
- Common to all architectures: **gates** for filtering information

Recurrent Neural Networks

Example: Update Gate RNN (UGRNN)





Cell that turns on inside quotes:

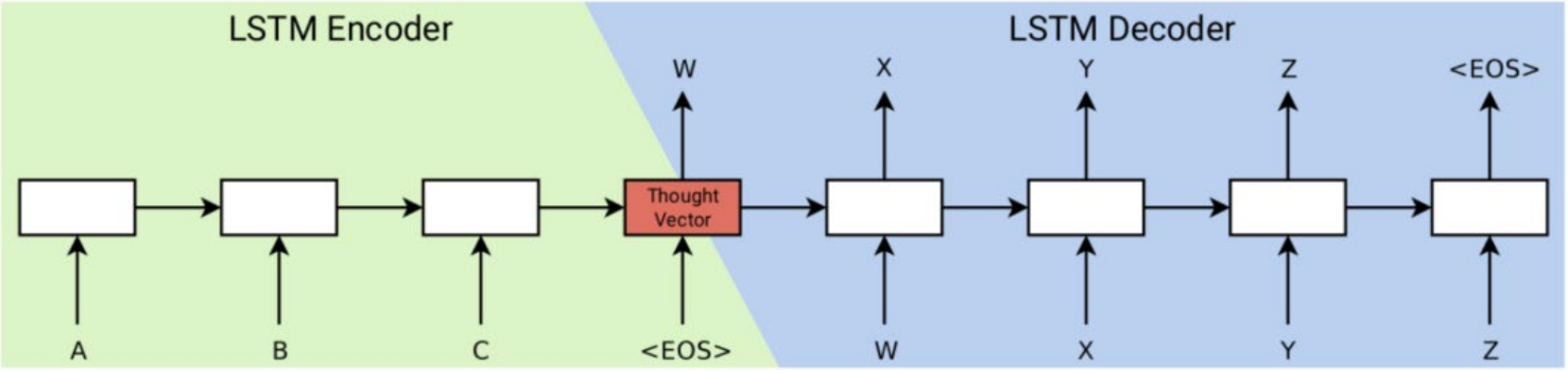
"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

- UGRNN is able to **keep the state** of a variable **over a long time horizon** ($u \approx 1$)
- For example, it can implement a logic to keep track of being inside a quote or not

Recurrent Neural Networks

Application: **Neural Machine Translation**



- **Two 4-Layer LSTMs** for encoding/decoding the source/target sentence
- Encoding operates in **reverse order** to introduce short-term dependencies
- Intermediate representation produced by the encoder is called **thought vector**
- Encoding using 1000 dim. word embeddings, decoding via **beam search**
- First end-to-end system that outperforms rule-based models ⇒ deployment

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014

Recurrent Neural Networks

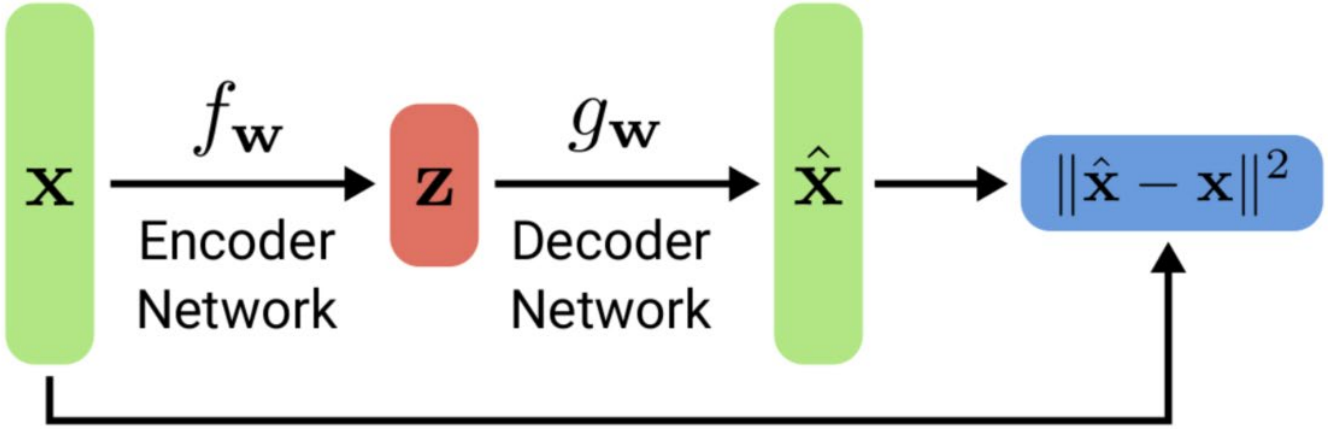
- RNNs process different variable length structured data
- Include feedback connections (depth = time)
- Weight sharing accross steps
- Training RNNs is not trivial: vanishing/exploding gradients
- Can be easily combined with other architectures, e.g., CNNs
- Many variants for various purposes

Until very recently, the state-of-the-art in sequence learning

Deep Autoencoders

Deep Autoencoders

Unsupervised Setting (latent variable models)



- An autoencoder is a **neural network** whose outputs are its own inputs
- The input $\mathbf{x} \in \mathbb{R}^D$ is compressed to a latent code $\mathbf{z} \in \mathbb{R}^Q$
- The goal is to **minimize the reconstruction error** (as in PCA)

Deep Autoencoders

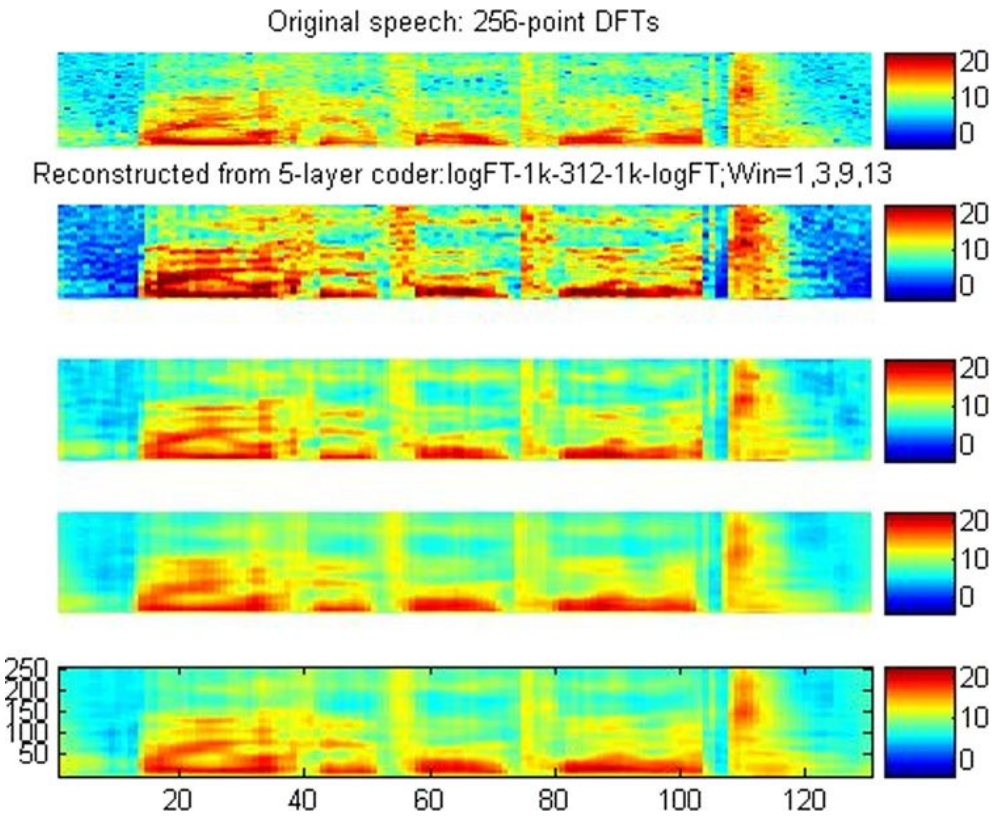


Fig. 2. Top to Bottom: Original spectrogram; reconstructions using input window sizes of $N = 1, 3, 9$, and 13 while forcing the coding units to be zero or one (i.e., a binary code). The y-axis values indicate FFT bin numbers (i.e., 256-point FFT is used for constructing all spectrograms).

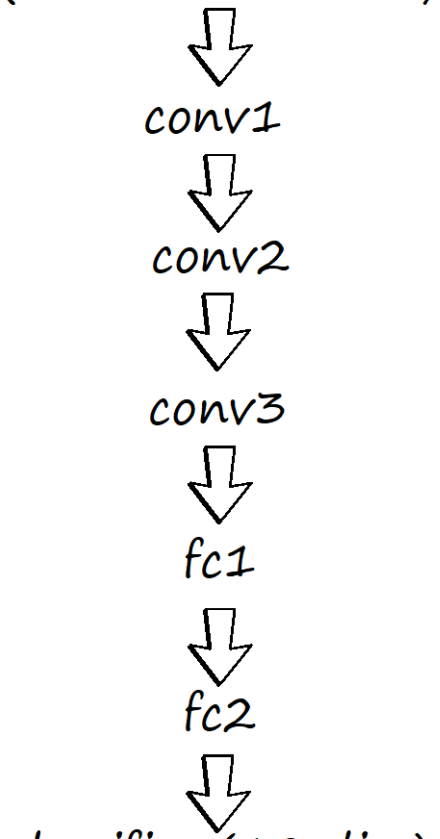
The real-valued activations of hidden units in the coding layer are quantized to be either zero or one with 0.5 as the threshold. These binary codes are then used to reconstruct the original spectrogram

Deng, L.; Seltzer, M.; Yu, D.; Acero, A.; Mohamed, A.; Hinton, G.: Binary coding of speech spectrograms using a deep auto-encoder, in *Proc. Interspeech*, 2010

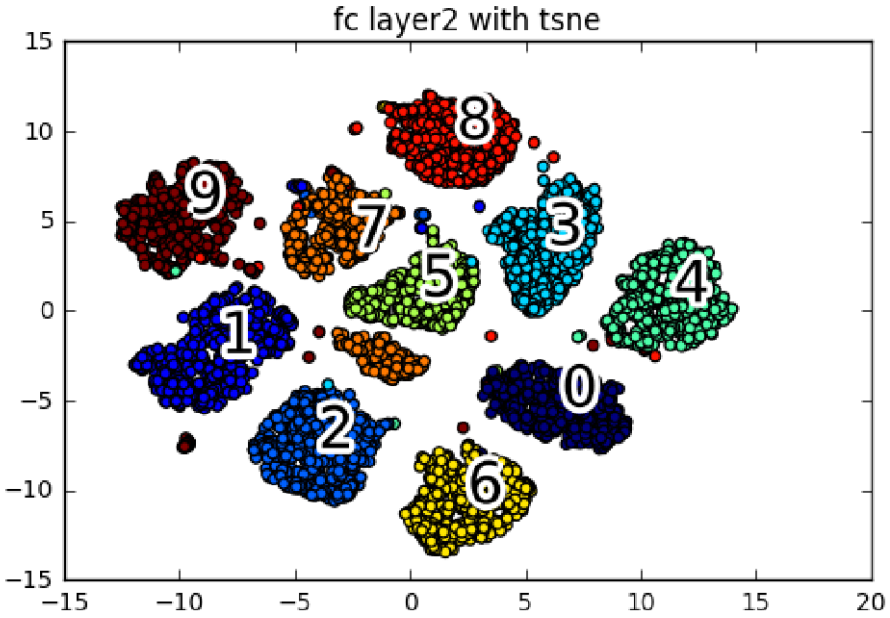
Deep Autoencoders

Example: embedding MNIST with LeNet5

MNIST data
(32x32=784 dim)



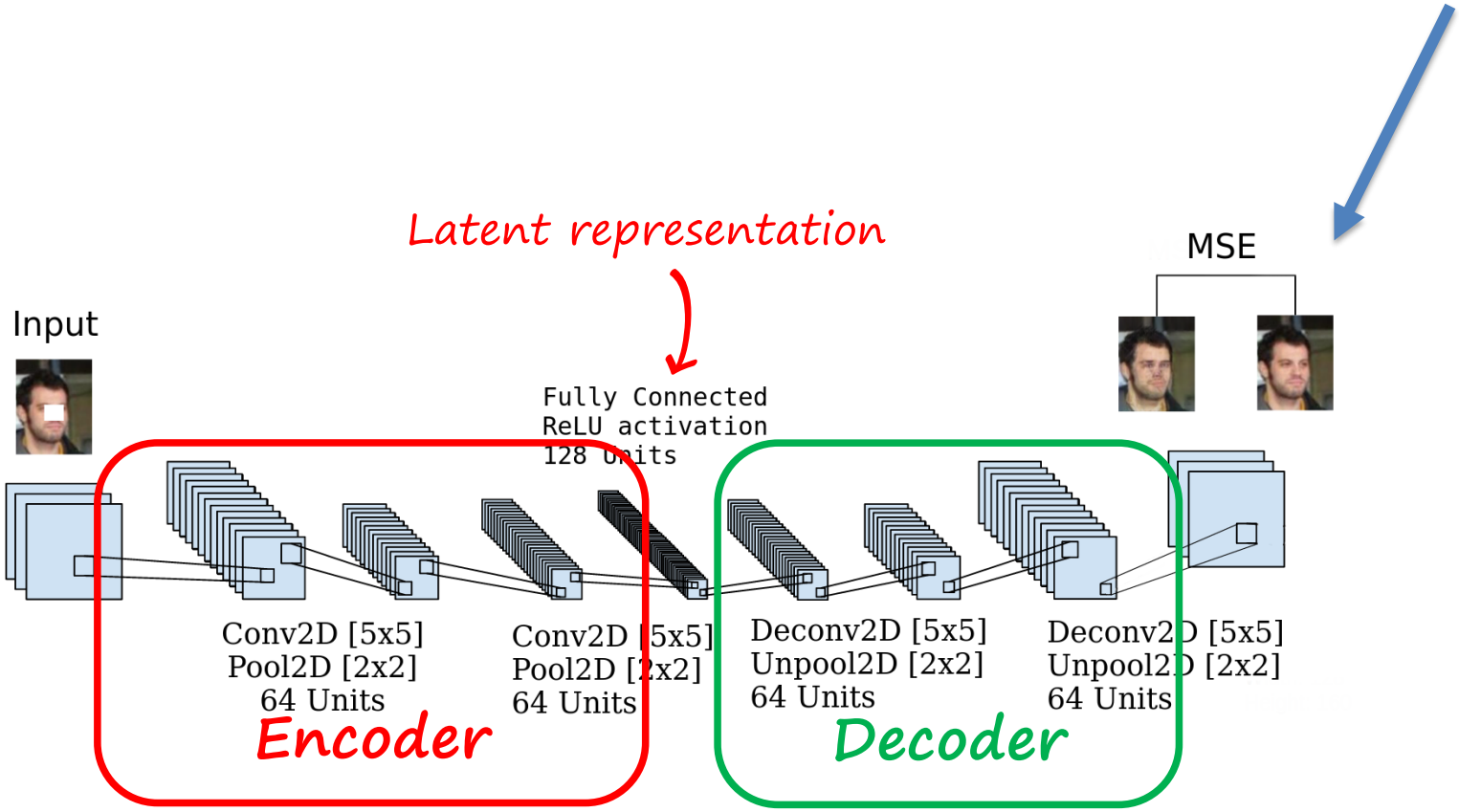
Easy to separate!!



https://ithelp.ithome.com.tw/articles/10187648

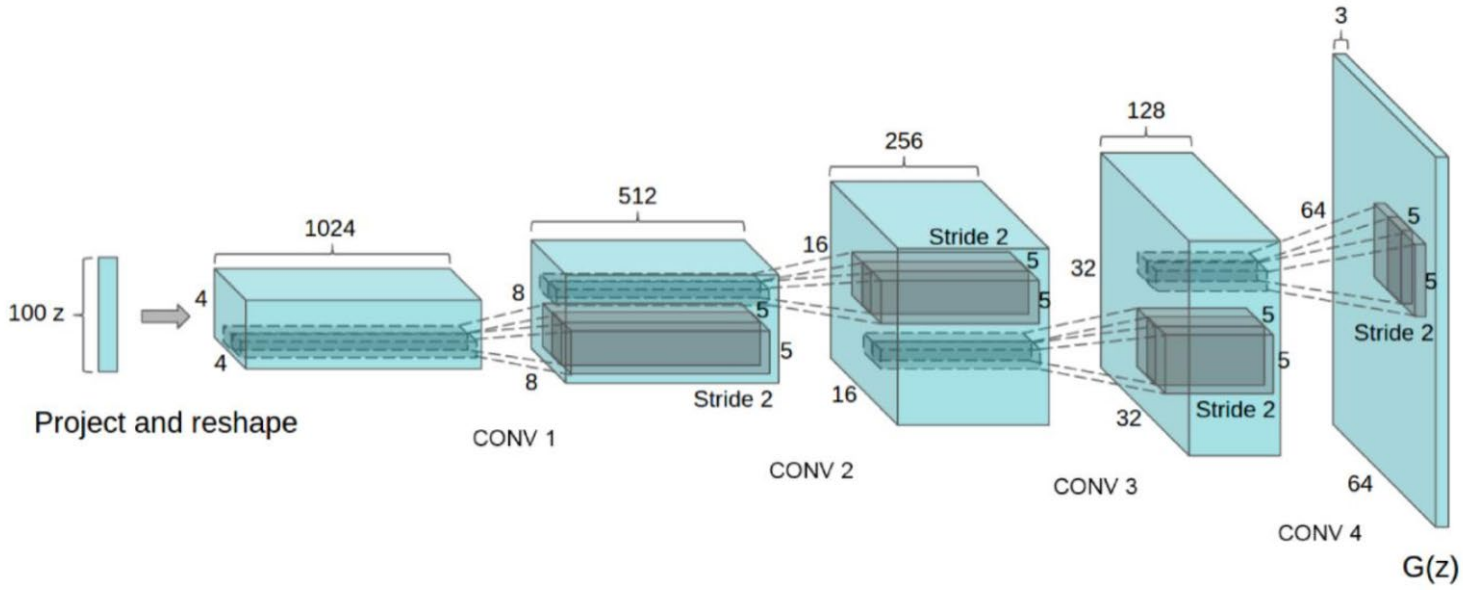
Deep Autoencoders

Example: autoencoders



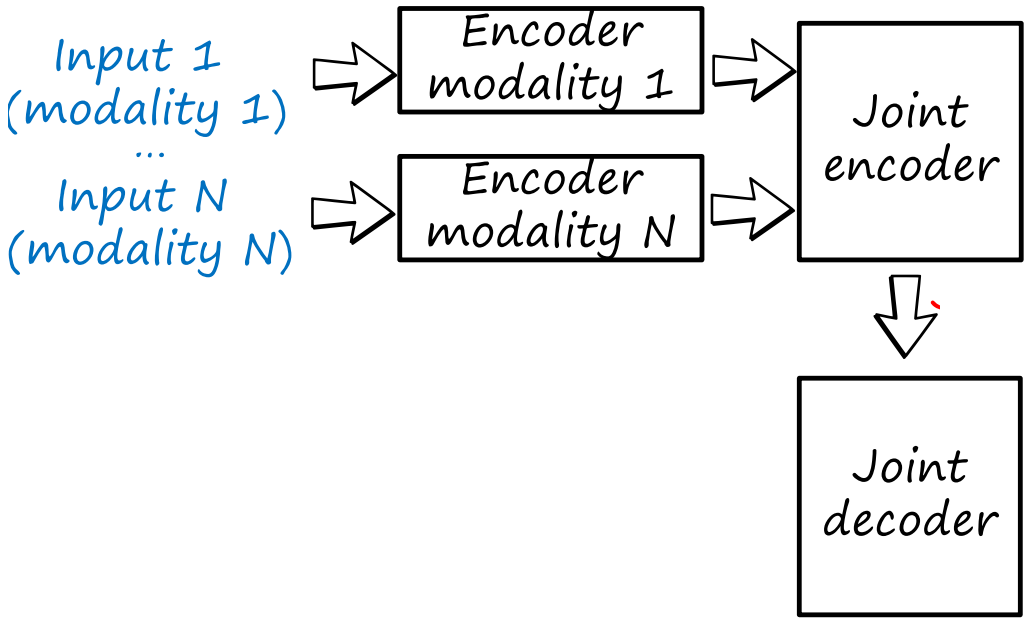
Deep Autoencoders

Image generation as decoder



Deep Autoencoders

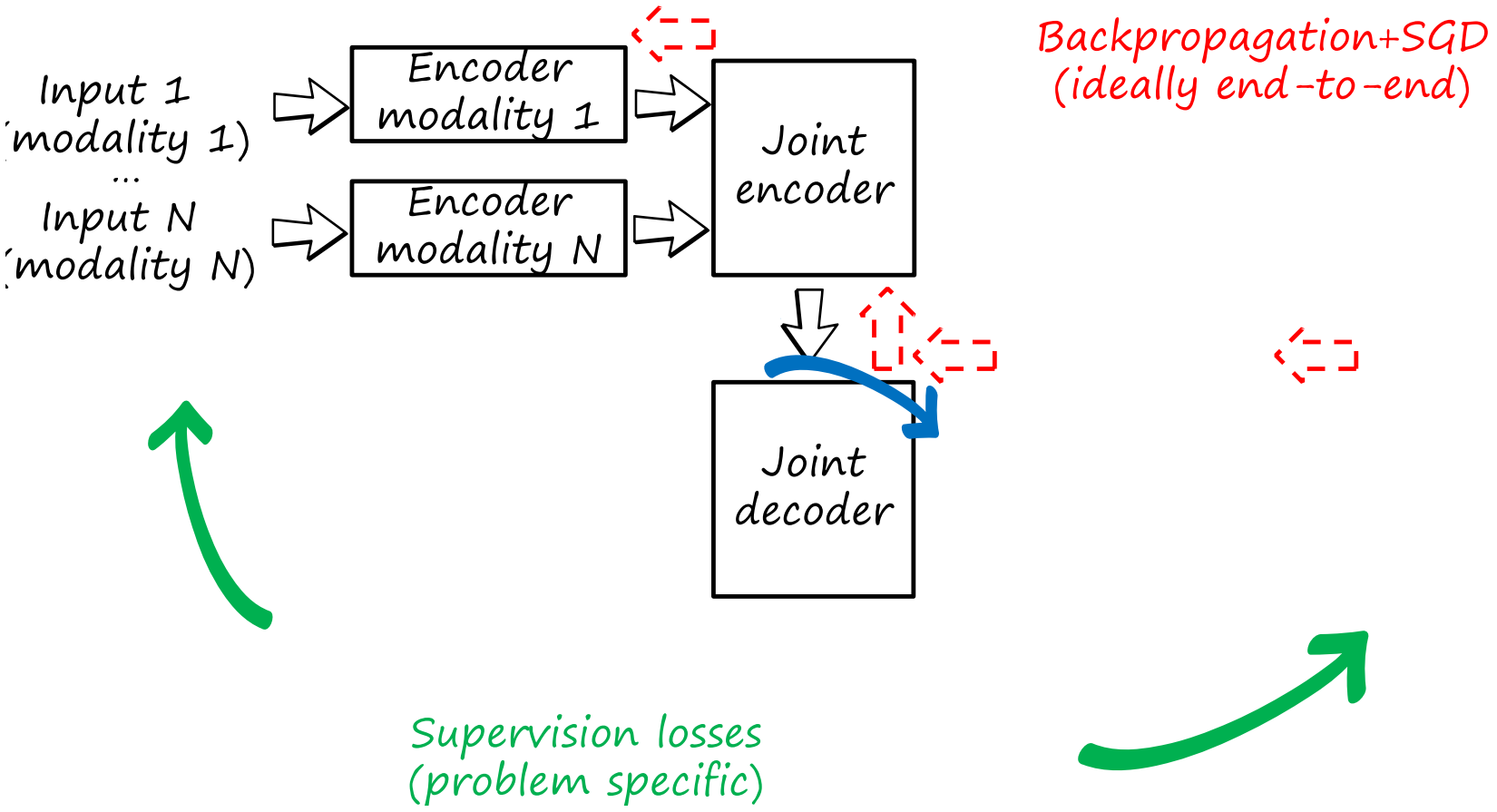
General multimodal multitask encoder-decoder



e information)

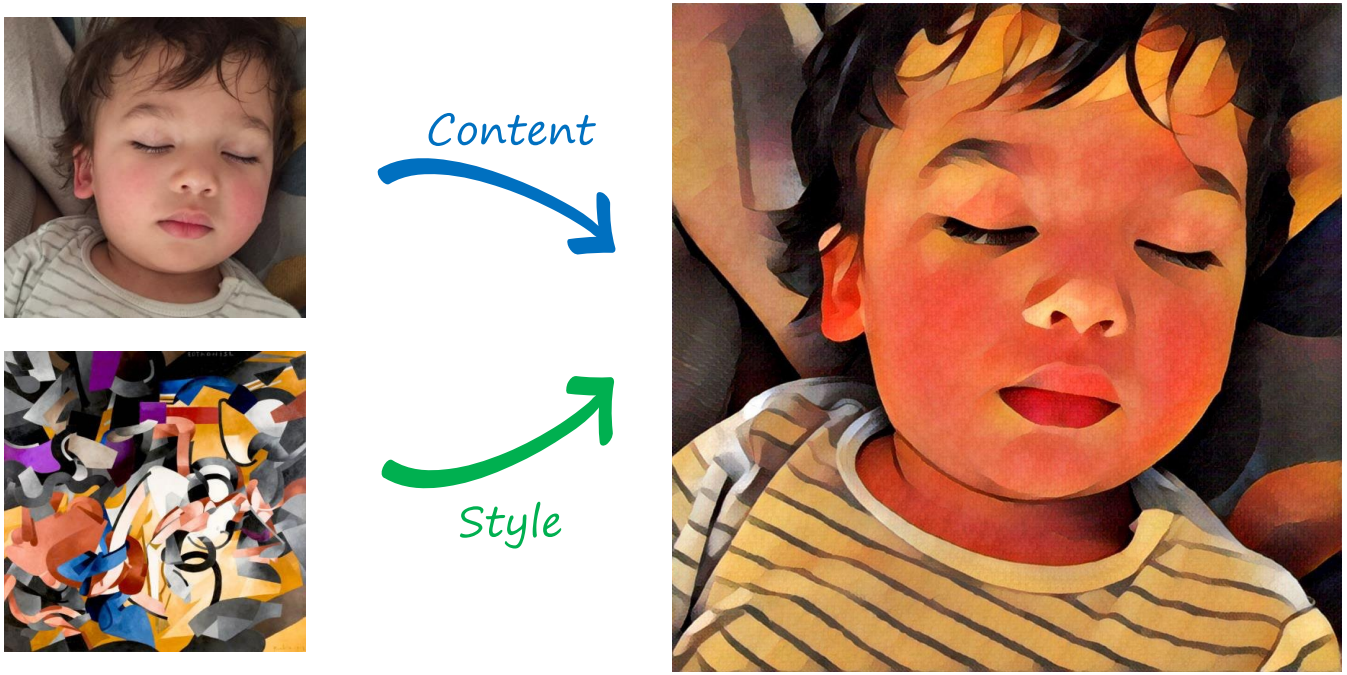
Deep Autoencoders

Training encoder-decoders



Deep Autoencoders

Application: Style transfer



Gatys et al, A Neural Algorithm of Artistic Style, 2015
Images from https://harishnarayanan.org/writing/artistic-style-transfer

Deep Autoencoders

Fast neural style transfer

Style

Content

Johnson et al, "Perceptual losses for real-time style transfer and super-resolution", ECCV 2016

Examples: image+text tasks

Image captioning

Image encoder → Text decoder → A big four legged animal with black and white stripes

Visual question answering

Image encoder

What is the color of the animal?

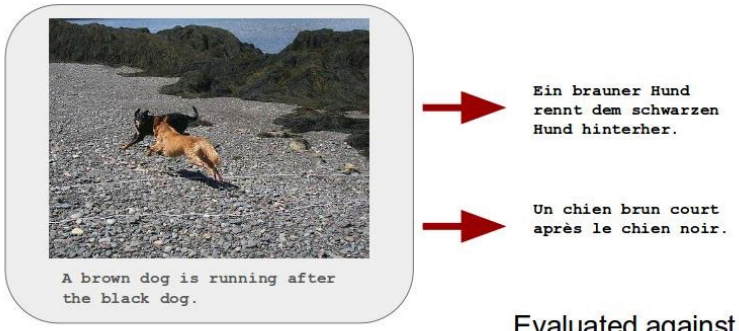
Text encoder

→ Text decoder or classifier → Yellow

Deep Autoencoders

Examples: image+text tasks

Multimodal translation

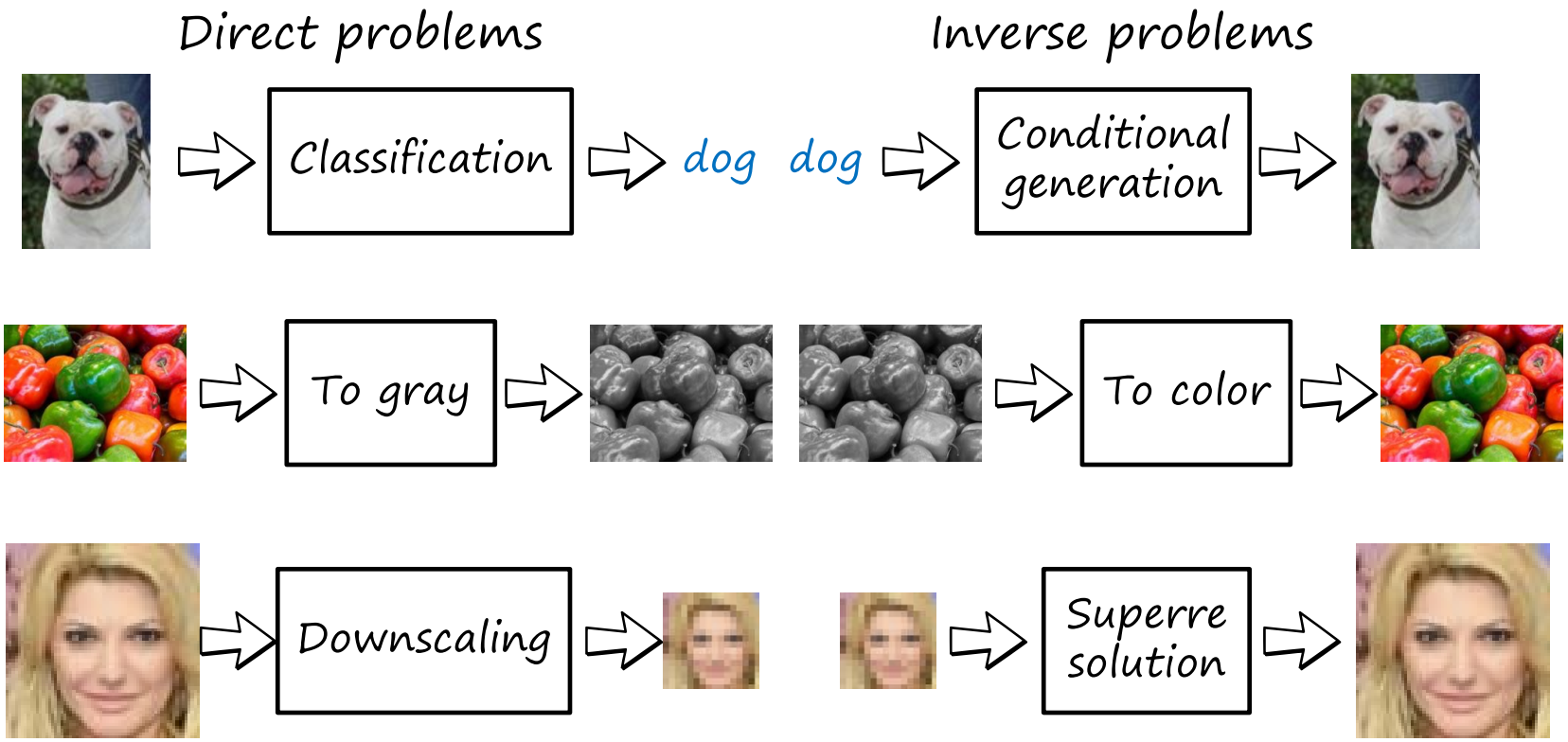


Text to image synthesis

Deep Autoencoders

Inverse Problems

Examples



Deep Autoencoders

Summary

Encoders

CNN
MLP
RNN (LSTM, GRU)
Word embeddings
Nearest neighbors
…

Decoders

Linear regression
Logistic regression
Multilayer perceptron
Deconv NN
RNN (LSTM, GRU)
Support Vector Machine
…

Losses

L1, L2
Cross-entropy
Contrastive
Triplet
Log-likelihood
KL, JS diverg.
Distillation
Wasserstein dist.
…

Training

Backpropagation
(Stage-wise)
(End-to-end)
…

Final Remarks

- Very active area of research with many industrial applications
- Deep learning is *literally replacing* existing ML methods in many domains
- Deep learning revolution can be explained because of:
  - Appearance of large, high-quality labeled datasets
  - Massively parallel computing with GPUs
  - Backprop-friendly activation
  - Improved architectures
  - Software platforms
  - New regularization techniques
  - Robust optimizers

Final Remarks

- Deep learning is empirically driven
  - *"Has Machine Learning Become Alchemy?"*
    (see LeCun vs Rahimi debate at NIPS 2017)

- Reproducibility crisis
  - Results are difficult to reproduce, mostly developed in industry
  - *"Deep Reinforcement Learning that Matters"*
    (example of initiatives mainly driven from Academia)

Final Remarks

- There are criticism about this methods:

"Such systems cannot reason about interventions and retrospection and, therefore, cannot serve as the basis for strong AI. To achieve human level intelligence, learning machines need the guidance of a model of reality, similar to the ones used in causal inference tasks".

Judea Pearl *(winner of Turing award 2011)*