Bil 470 / YAP 470

Introduction to Machine Learning (Yapay Öğrenme)

Batuhan Bardak

Introduction to Deep Learning

Date: 7.11.2022

Plan for today

- Convolutional Neural Network
- Word Embeddings
- Recurrent Neural Networks

Used everywhere for Vision

[Krizhevsky 2012]

[Ciresan et al. 2013]

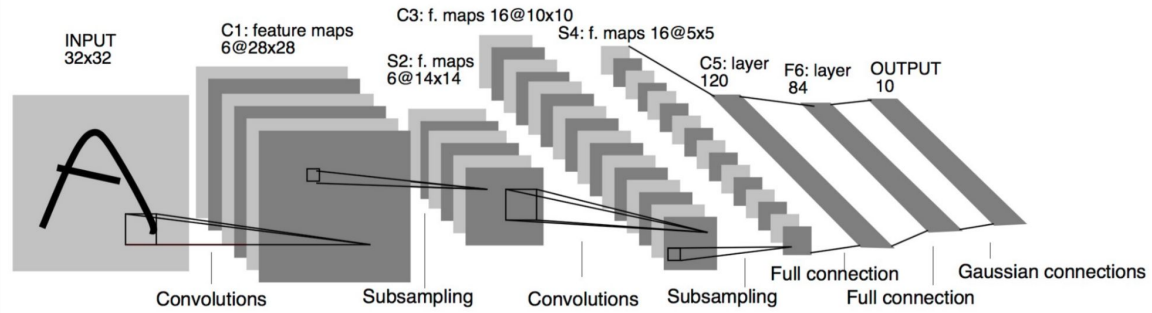
[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

Slide Credit: ip-paris, deep learning

ConvNets for image classification

CNN = Convolutional Neural Networks = ConvNet



LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition.

Convolutional Neural Networks, Motivation

Standard Dense Layer for an image input:

```
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

How many parameters in the Dense layer?

$$640 \times 480 \times 3 \times 1000 + 1000 = 922M!$$

Spatial organization of the input is destroyed by `Flatten`

We never use Dense layers directly on large images. Most standard solution is **convolution** layers

Slide Credit: ip-f

MLP vs CNN

Fully Connected Network: MLP

```
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```

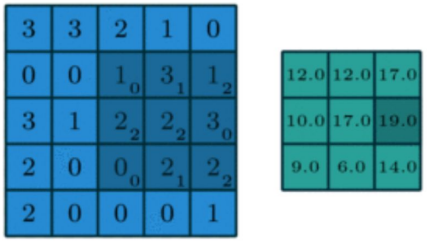
Convolutional Network

```
input_image = Input(shape=(28, 28, 1))
x = Conv2D(32, 5, activation='relu')(input_image)
x = MaxPool2D(2, strides=2)(x)
x = Conv2D(64, 3, activation='relu')(x)
x = MaxPool2D(2, strides=2)(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
convnet = Model(inputs=input_image, outputs=x)
```

2D spatial organization of features preserved untill `Flatten`.

Slide Credit: ip-paris, deep lea

Convolution in a neural network



- x is a 3×3 chunk (dark area) of the image *(blue array)*
- Each output neuron is parametrized with the 3×3 weight matrix $\mathbf{w}$ *(small numbers)*

The activation obtained by sliding the 3×3 window and computing:

$$z(x) = relu(\mathbf{w}^T x + b)$$

Slide Credit: ip-paris, deep learning

https://github.com/vdumoulin/conv_arithmetic

Motivations

Local connectivity

- A neuron depends only on a few local input neurons
- Translation invariance

Comparison to Fully connected

- Parameter sharing: reduce overfitting
- Make use of spatial structure: **strong prior** for vision!

Animal Vision Analogy

Hubel & Wiesel, RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX (1959)

Slide Credit: ip-paris, deep learning

Why Convolution

Discrete convolution (actually cross-correlation) between two functions f and g:

$$(f \star g)(x) = \sum_{a+b=x} f(a).\, g(b) = \sum_{a} f(a).\, g(x+a)$$

2D-convolutions (actually 2D cross-correlation):

$$(f \star g)(x, y) = \sum_{n} \sum_{m} f(n, m).\, g(x+n, y+m)$$

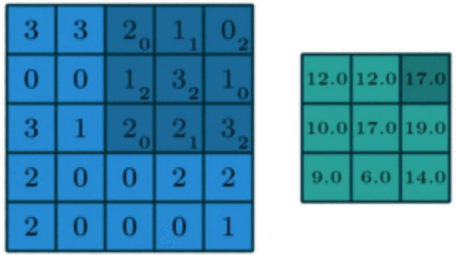
f is a convolution **kernel** or **filter** applied to the 2-d map g (our image)

Slide Credit: ip-paris, deep learning

Example: convolution image

- Image: im of dimensions 5×5
- Kernel: k of dimensions 3×3

$$(k \star im)(x, y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n, m).\, im(x + n - 1, y + m - 1)$$

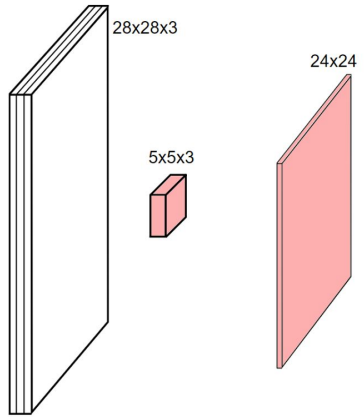


Slide Credit: ip-paris, deep learning

Channels

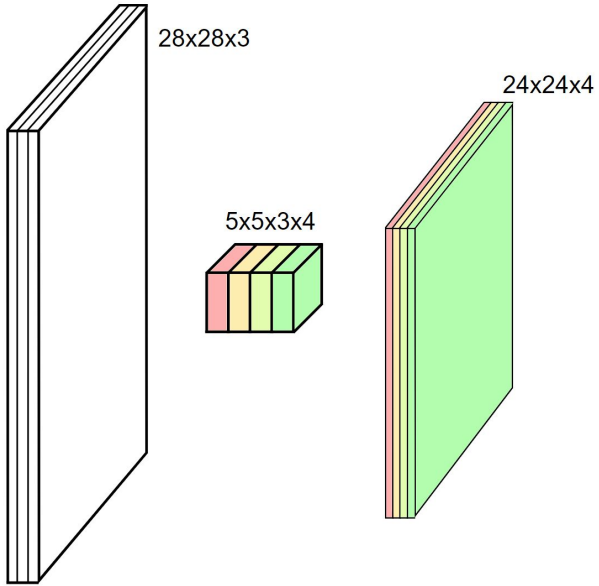
Colored image = tensor of shape `(height, width, channels)`

Convolutions are usually computed for each channel and summed:



$$(k \star im^{color}) = \sum_{c=0}^{2} k^c \star im^c$$

Multiple convolutions

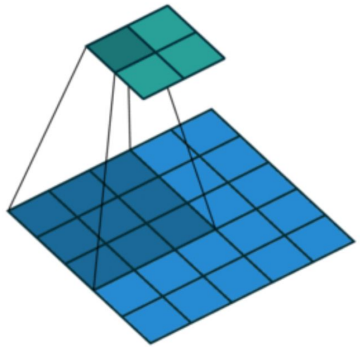


- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)
- Output dimension: `length - kernel_size + 1`

Slide Credit: ip-paris, deep learning

Strides

- Strides: increment step size for the convolution operator
- Reduces the size of the output map

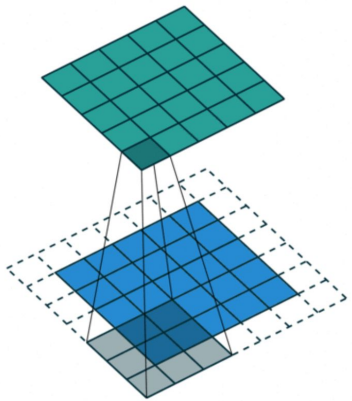


Example with kernel size 3×3 and a stride of 2 (image in blue)

Slide Credit: ip-paris, deep learning

Padding

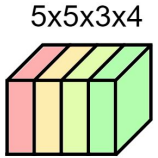
- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s



Slide Credit: ip-paris, deep learning

Dealing with shapes

Kernel or **Filter** shape (F, F, C^i, C^o)



- $F \times F$ kernel size,
- C^i input channels
- C^o output channels

Number of parameters: $(F \times F \times C^i + 1) \times C^o$

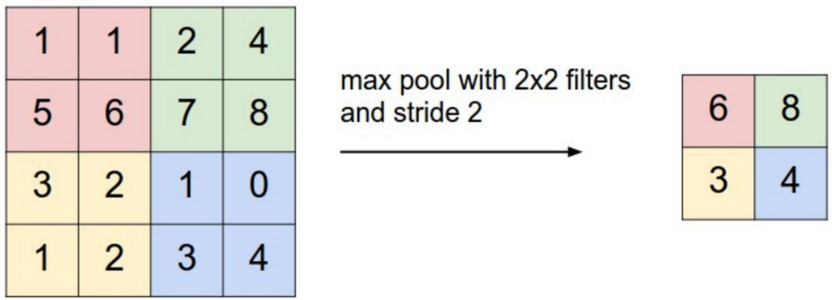
Activations or **Feature maps** shape:

- Input (W^i, H^i, C^i)
- Output (W^o, H^o, C^o)

$$W^o = (W^i - F + 2P)/S + 1$$

Pooling

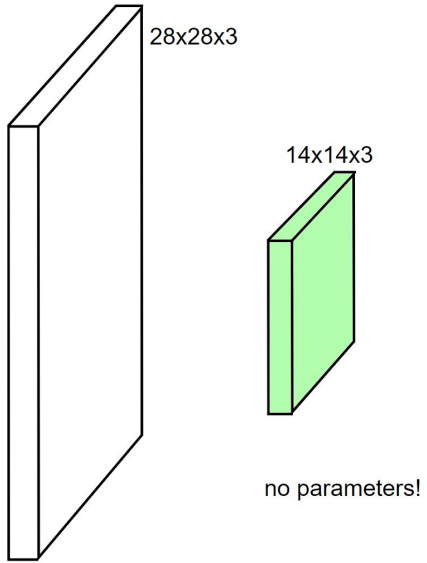
- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Slide Credit: ip-paris, deep learning

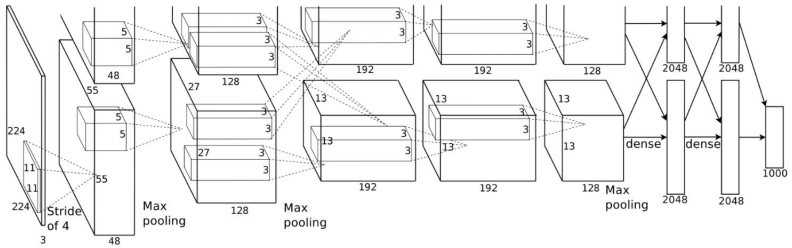
Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Slide Credit: ip-paris, deep learning

Example architecture, AlexNet



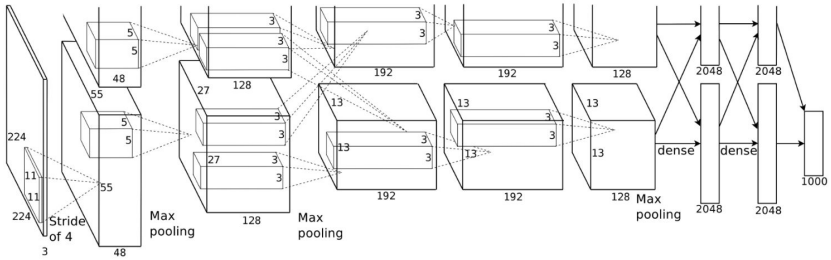
First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: `(11,11,3,96)`
- Output shape: `(55,55,96)`
- Number of parameters: `34,944`
- Equivalent MLP parameters: `43.7 x 1e9`

Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

Slide Credit: ip-paris, deep learning

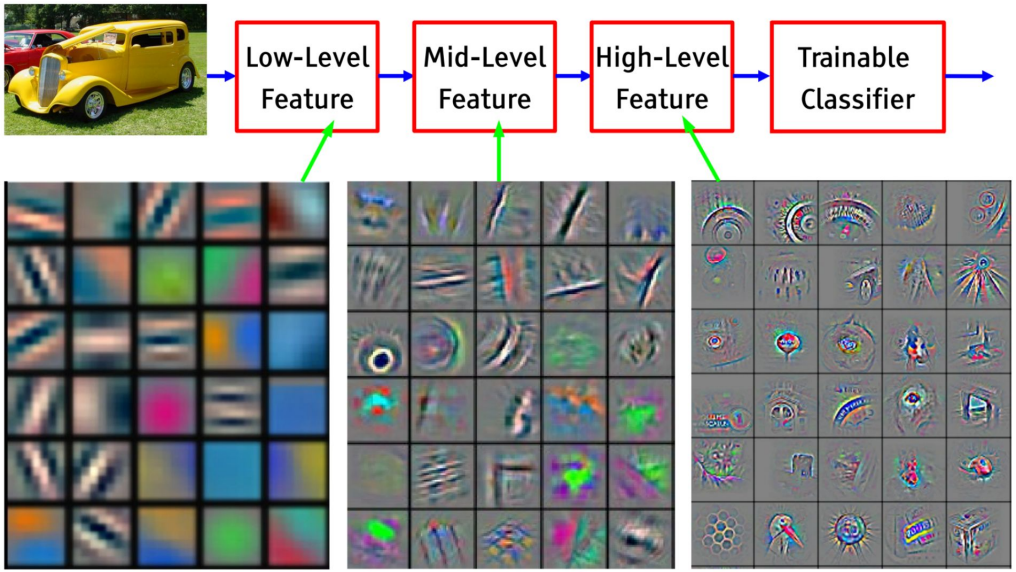
Example architecture, AlexNet




```
INPUT:     [227x227x3]
CONV1:     [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96]      3x3   filters at stride 2
CONV2:     [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2: [13x13x256]     3x3   filters at stride 2
CONV3:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV4:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV5:     [13x13x256] 256 3x3   filters at stride 1, pad 1
MAX POOL3: [6x6x256]       3x3   filters at stride 2
FC6:       [4096]      4096 neurons
FC7:       [4096]      4096 neurons
FC8:       [1000]      1000 neurons (softmax logits)
```

Slide Credit: ip-paris, deep learning

Hierarchical representation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide Credit: ip-paris, deep learning

Pretrained Models

Training a model on ImageNet from scratch takes **days or weeks**.

Many models trained on ImageNet and their weights are publicly available!

Fine-tuning

Retraining the (some) parameters of the network (given enough data)

- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning

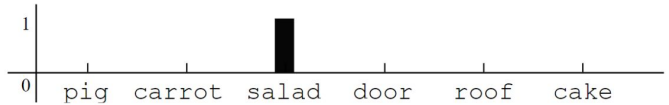
Slide Credit: ip-paris, deep learning

Word Embeddings

Slide Credit: ip-paris, deep learning

One-hot representation

$$onehot(\text{'salad'}) = [0, 0, 1, \ldots, 0] \in \{0, 1\}^{|V|}$$



- Sparse, discrete, large dimension $|V|$
- Each axis has a meaning
- Symbols are equidistant from each other:

$$\text{euclidean distance} = \sqrt{2}$$

Embedding

$$embedding(\text{'salad'}) = [3.28, -0.45, \dots 7.11] \in \mathbb{R}^d$$

- Continuous and dense
- Can represent a huge vocabulary in low dimension, typically: $d \in \{16, 32, \dots, 4096\}$
- Axis have no meaning *a priori*
- Embedding metric can capture semantic distance

Neural Networks compute transformations on continuous vectors

Distance and similarity in Embedding Space

Euclidean distance

$$d(x, y) = ||x - y||_2$$

- Simple with good properties
- Dependent on norm (embeddings usually unconstrained)

Cosine similarity

$$cosine(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

- Angle between points, regardless of norm
- $cosine(x, y) \in (-1, 1)$
- Expected cosine similarity of random pairs of vectors is 0

Slide Credit: ip-paris, deep learning

Visualizing Embeddings

- Visualizing requires a projection in 2 or 3 dimensions
- Objective: visualize which embedded symbols are similar

PCA

- Limited by linear projection, embeddings usually have complex high dimensional structure

t-SNE

Visualizing data using t-SNE, L van der Maaten, G Hinton, *The Journal of Machine Learning Research*, 2008

Slide Credit: ip-paris, deep learning

Word representation

Words are indexed and represented as 1-hot vectors

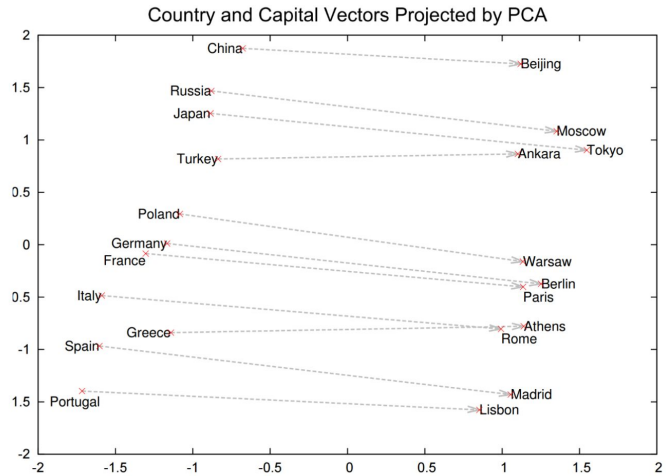
Large Vocabulary of possible words $|V|$

Use of **Embeddings** as inputs in all Deep NLP tasks

Word embeddings usually have dimensions 50, 100, 200, 300

Slide Credit: ip-paris, deep learning

Word2Vec



- Linear relations in Word2Vec embeddings
- Many come from text structure (e.g. Wikipedia)

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013

Slide Credit: ip-paris, deep learning

Self-supervised training

Distributional Hypothesis (Harris, 1954): *"words are characterised by the company that they keep"*

Main idea: learning word embeddings by **predicting word contexts**

Given a word e.g. "carrot" and any other word $w \in V$ predict probability $P(w|\text{carrot})$ that w occurs in the context of "carrot".

- **Unsupervised / self-supervised**: no need for class labels.
- (Self-)supervision comes from **context**.
- Requires a lot of text data to cover rare words correctly.

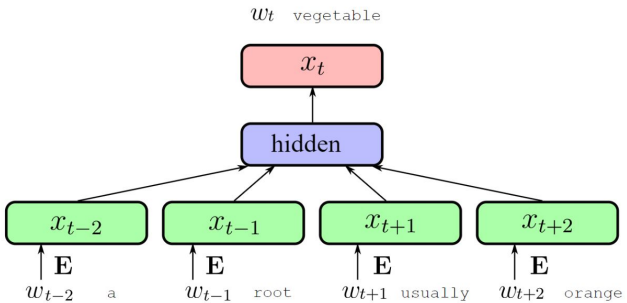
Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013

Word2Vec: CBoW

CBoW: representing the context as **Continuous Bag-of-Word**

Self-supervision from large unlabeled corpus of text: *slide* over an **anchor word** and its **context**:

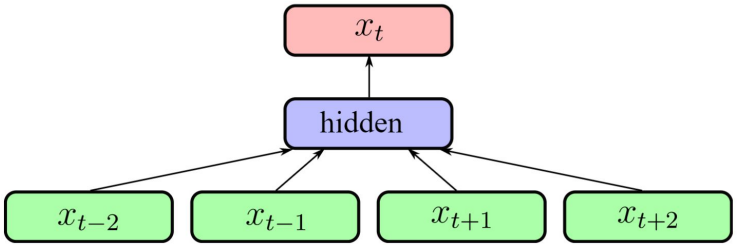
the carrot is a root vegetable, usually orange



Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013

Slide Credit: ip-paris, deep learning

Word2Vec: Details

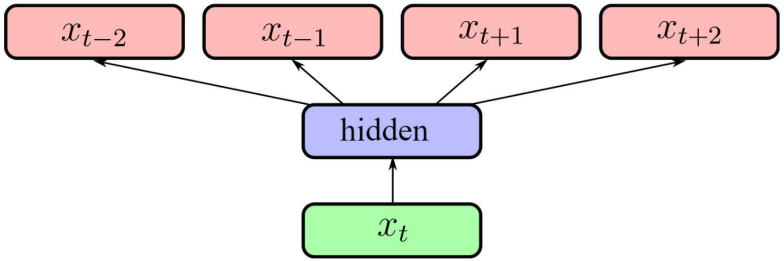


- Similar as supervised CBoW (e.g. fastText) with |V| classes
- Use **Negative Sampling**: sample *negative* words at random instead of computing the full softmax. See: http://sebastianruder.com/word-embeddings-softmax/index.html
- Large impact of **context size**

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013

Slide Credit: ip-paris, deep learning

Word2Vec: Skip Gram



- Given the central word, predict occurence of other words in its context.
- Widely used in practice
- Again **Negative Sampling** is used as a cheaper alternative to full softmax.

Slide Credit: ip-paris, deep learning

Take Away on Embeddings

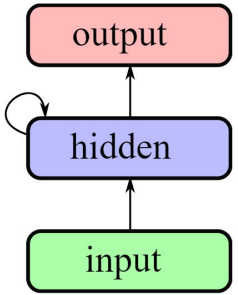
For text applications, inputs of Neural Networks are Embeddings

- If **little training data** and a wide vocabulary not well covered by training data, use **pre-trained self-supervised embeddings** (transfer learning from Glove, word2vec or fastText embeddings)
- If **large training data** with labels, directly learn task-specific embedding with methods such as **fastText in supervised mode**.
- These methods use **Bag-of-Words** (BoW): they **ignore the order** in word sequences
- Depth & non-linear activations on hidden layers are not that useful for BoW text classification.

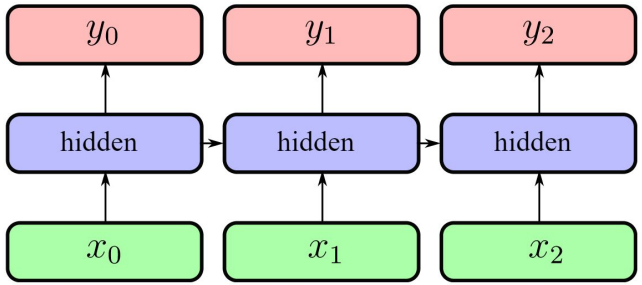
Word Embeddings no long state of the art for NLP tasks: BERT-style pretraining of deep transformers with sub-word tokenization is now used everywhere.

Slide Credit: ip-paris, deep learning

Recurrent Neural Network

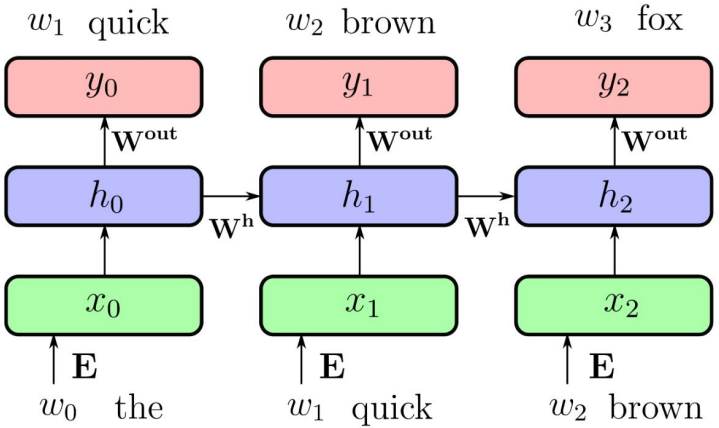


Unroll over a sequence (x_0, x_1, x_2):



Slide Credit: ip-paris, deep learning

Language Modelling

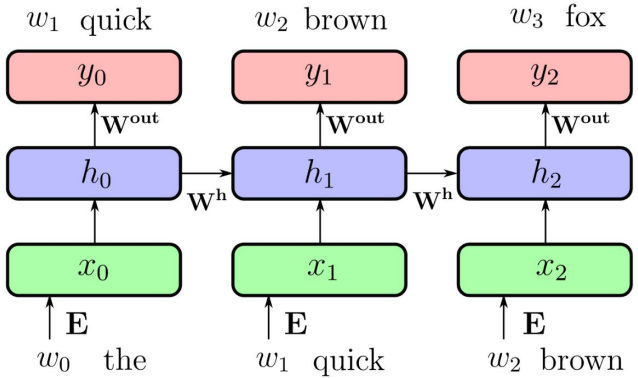


input $(w_0, w_1, \ldots, w_t)$ sequence of words (1-hot encoded)

output $(w_1, w_2, \ldots, w_{t+1})$ shifted sequence of words (1-hot encoded)

Slide Credit: ip-paris, deep learning

Language Modelling



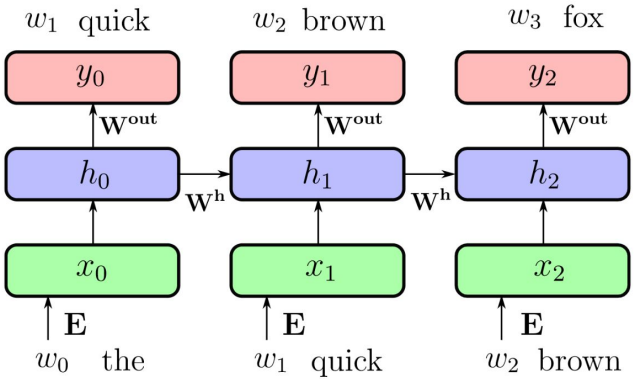
$$x_t = \text{Emb}(w_t) = \mathbf{E}w_t$$ input projection H

$$h_t = g(\mathbf{W^h}h_{t-1} + x_t + b^h)$$ recurrent connection H

$$y = \text{softmax}(\mathbf{W^o}h_t + b^o)$$ output projection K = |V|

Slide Credit: ip-paris, deep learning

Recurrent Neural Network



Input embedding $\mathbf{E}$ — |V| x H

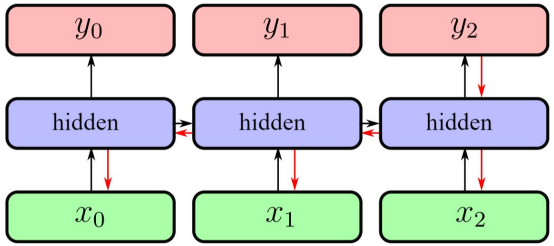
Recurrent weights $\mathbf{W}^{\text{h}}$ — H x H

Output weights $\mathbf{W}^{\text{out}}$ — H x K = H x |V|

Slide Credit: ip-paris, deep learning

Backpropagation through time

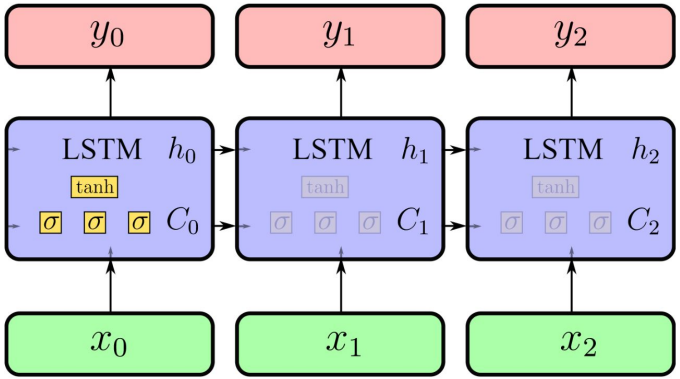
Similar as standard backpropagation on unrolled network



- Similar as training **very deep networks** with tied parameters
- Example between x_0 and y_2: W^h is used twice
- Usually truncate the backprop after T timesteps
- Difficulties to train long-term dependencies

Slide Credit: ip-paris, deep learning

LSTM



- 4 times more parameters than RNN
- Mitigates **vanishing gradient** problem through **gating**
- Widely used and SOTA in many sequence learning problems

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 1997

Slide Credit: ip-paris, deep learning

LSTM

$$\mathbf{u} = \sigma(\mathbf{W^u} \cdot h_{t-1} + \mathbf{I^u} \cdot x_t + b^u)$$ Update gate H

$$\mathbf{f} = \sigma(\mathbf{W^f} \cdot h_{t-1} + \mathbf{I^f} \cdot x_t + b^f)$$ Forget gate H

$$\mathbf{\tilde{c}_t} = \tanh(\mathbf{W^c} \cdot h_{t-1} + \mathbf{I^c} \cdot x_t + b^c)$$ Cell candidate H

$$\mathbf{c_t} = \mathbf{f} \odot \mathbf{c_{t-1}} + \mathbf{u} \odot \mathbf{\tilde{c}_t}$$ Cell output H

$$\mathbf{o} = \sigma(\mathbf{W^o} \cdot h_{t-1} + \mathbf{I^o} \cdot x_t + b^o)$$ Output gate H

$$\mathbf{h_t} = \mathbf{o} \odot \tanh(\mathbf{c_t})$$ Hidden output H

$$y = \text{softmax}(\mathbf{W} \cdot h_t + b)$$ Output K

$$W^u, W^f, W^c, W^o$$ Recurrent weights H x H

$$I^u, I^f, I^c, I^o$$ Input weights N x H

Slide Credit: ip-paris, deep learning

Vanishing / Exploding Gradients

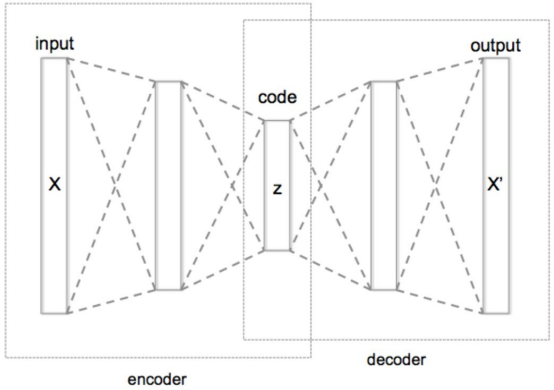
Passing through t time-steps, the resulting gradient is the **product** of many gradients and activations.

- Gradient messages close to 0 can shrink be 0
- Gradient messages larger than 1 can explode
- **LSTM / GRU** mitigate that in RNNs
- **Additive path** between c_t and c_{t-1}
- **Gradient clipping** prevents gradient explosion
- Well chosen **activation function** is critical (tanh)

Skip connections in ResNet also alleviate a similar optimization problem.

Slide Credit: ip-paris, deep learning

Autoencoder

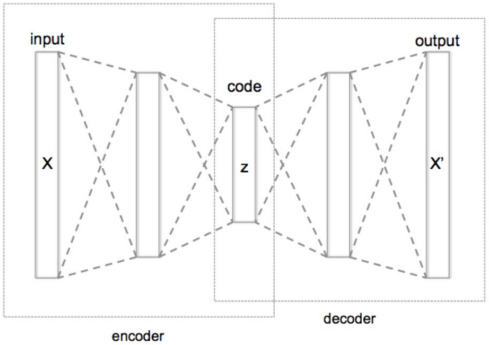


Supervision : reconstruction loss of the input, usually:

$$l(x, f(x)) = ||f(x) - x||_2^2$$

Slide Credit: ip-paris, deep learning

Autoencoder



Keeping the **latent code z** low-dimensional forces the network to learn a "smart" compression of the data, not just an identity function

Encoder and decoder can have arbritrary architecture (CNNs, RNNs...)

Slide Credit: ip-paris, deep learning

Use and Limitations

After **pre-training** use the latent code $\mathbf{z}$ as input to a classifier instead of $\mathbf{x}$

Semi-supervised learning simultaneous learning of the latent code (on a large, unlabeled dataset) and the classifier (on a smaller, labeled dataset)

Other use: Use decoder $D(x)$ as a **Generative model**: generate samples from random noise

Limitations :

- Direct autoencoder fails to capture good representations for complex data such as images
- The generative model is usually of very poor quality (very blurry for images for instance)

Slide Credit: ip-paris, deep learning

Next Class:

Unsupervised Learning