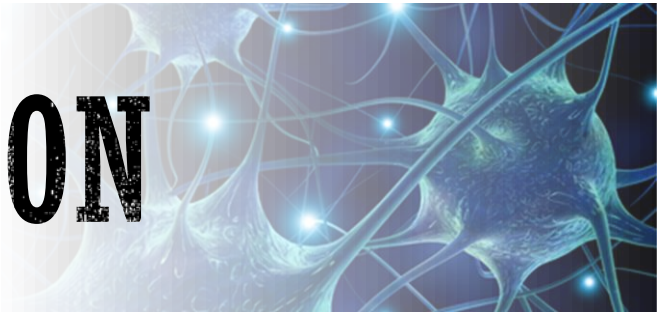
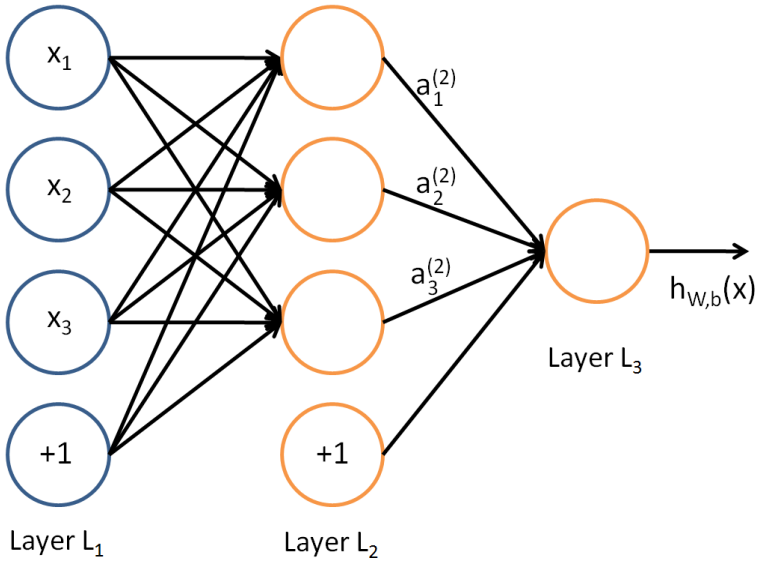
BACKPROPAGATION



MULTI-LAYER NEURAL NETWORK



Forward Propagation.

$$z^{(2)} = W^{(1)} x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

BACKPROPAGATION

Chain Rule for Neural Networks.

$$\frac{\partial}{\partial W_{ij}^{(l)}}\left(\frac{1}{2}\left\|a^{(n_l)}-y\right\|^2\right)=\left(a^{(n_l)}-y\right)\frac{\partial}{\partial W_{ij}^{(l)}}f\left(z^{(n_l)}\right)$$

$$=\left(a^{(n_l)}-y\right)f'\left(z^{(n_l)}\right)\frac{\partial}{\partial W_{ij}^{(l)}}\left(W^{(n_l-1)}a^{(n_l-1)}+b^{(n_l-1)}\right)$$

$$=\left(a^{(n_l)}-y\right)f'\left(z^{(n_l)}\right)W^{(n_l-1)}\frac{\partial}{\partial W_{ij}^{(l)}}a^{(n_l-1)}$$

$$=\underbrace{\underbrace{\left(a^{(n_l)}-y\right)f'\left(z^{(n_l)}\right)}_{\delta^{(n_l)}}W^{(n_l-1)}f'\left(z^{(n_l-1)}\right)}_{\delta^{(n_l-1)}}\frac{\partial}{\partial W_{ij}^{(l)}}z^{(n_l-1)}$$

Backpropagation algorithm

1. Perform forward propagation.
2. Compute δ for the output unit(s), e.g. $\delta = y - t$ in the case of MSE.
3. Backpropagate the δ's using

$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

   for each hidden unit in the network.
4. Compute the required derivatives using

$$\frac{\partial L}{\partial w_{ji}^{(k)}} = \delta_j z_i.$$

WHAT WAS WRONG WITH BACKPROPAGATION IN 1986?

1. Our labeled datasets were thousands of times too small.
2. Our computers were millions of times too slow.
3. We initialized the weights in a stupid way.
4. We used the wrong type of non-linearity.

Hinton, "Deep Learning," https://royalsociety.org/events/2015/05/breakthrough-science-technologies-machine-learning/

Xavier initialization

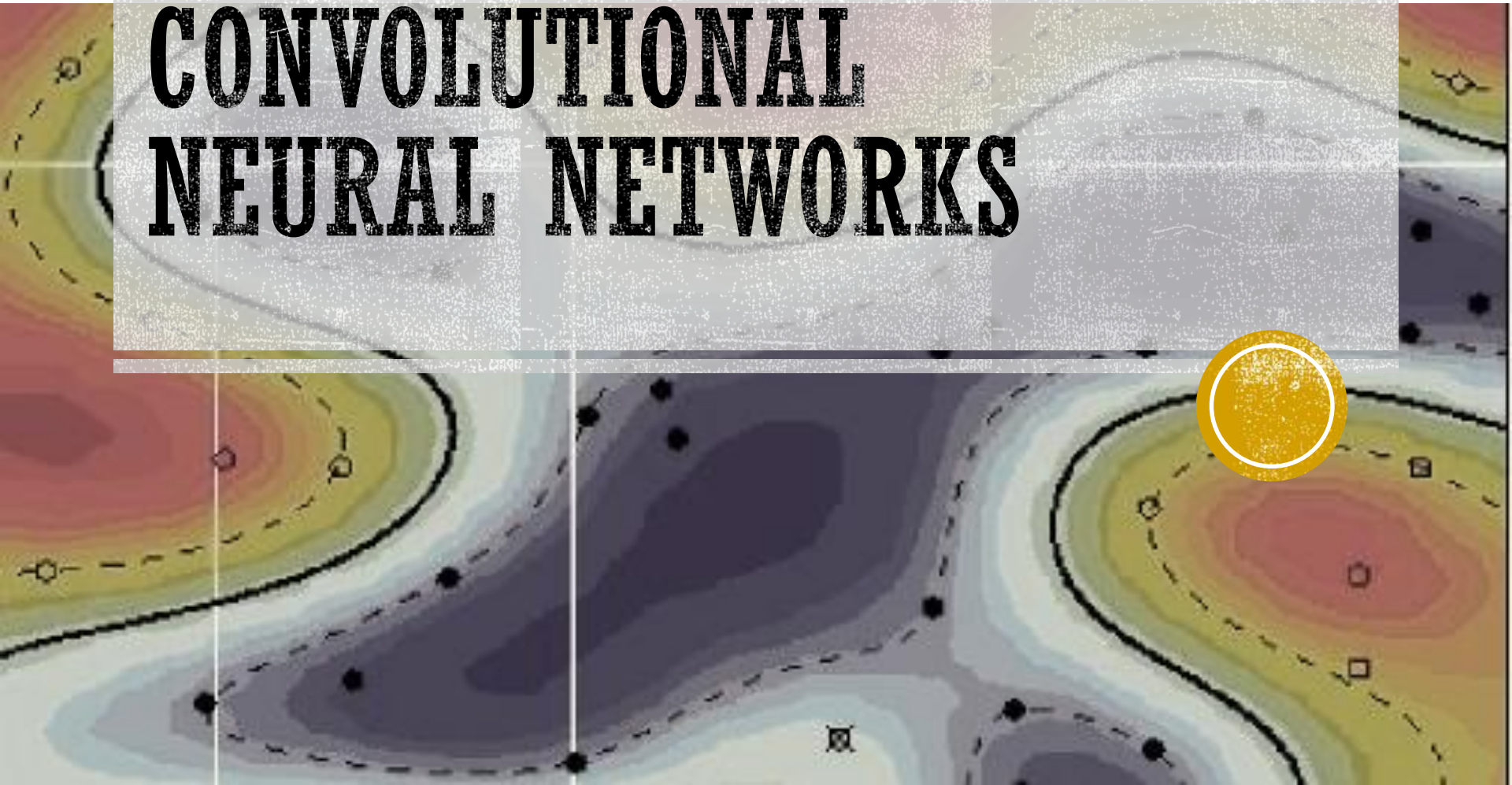
- Initizialize weights using

$$w^{(l)} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{(l)} + n^{(l-1)}}}\right),$$

  where $n^{(l)}$ is the number of neurons in layer l.

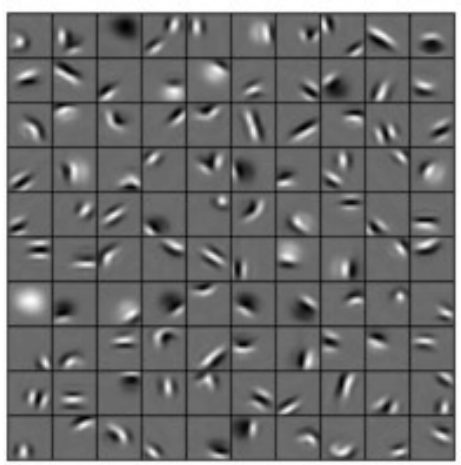
- This makes the variance of the activations in each layer similar to one another.

CONVOLUTIONAL NEURAL NETWORKS



What are convolutional neural networks (CNNs)?

- Category of neural networks particularly effective for image classification
- Deep CNNs extract a hierarchy of features from the input:
  - Lower layers extract local features such as lines and curves
  - Higher layers extract composites of local features; e.g. parts of a face



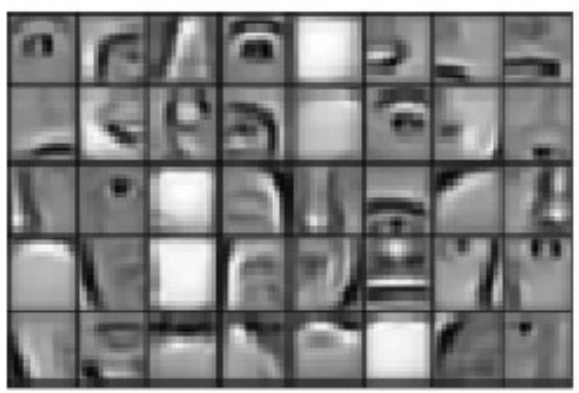
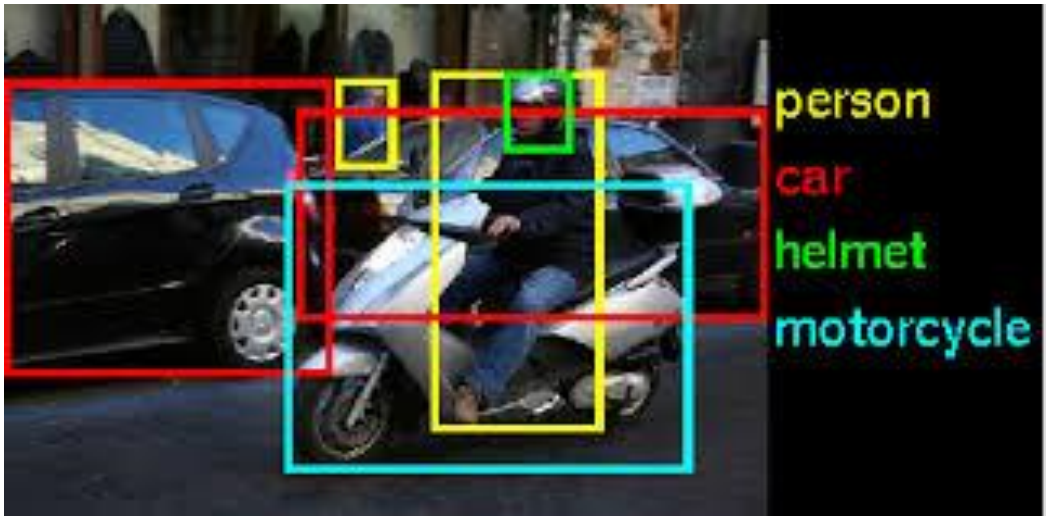


IMAGE RECOGNITION



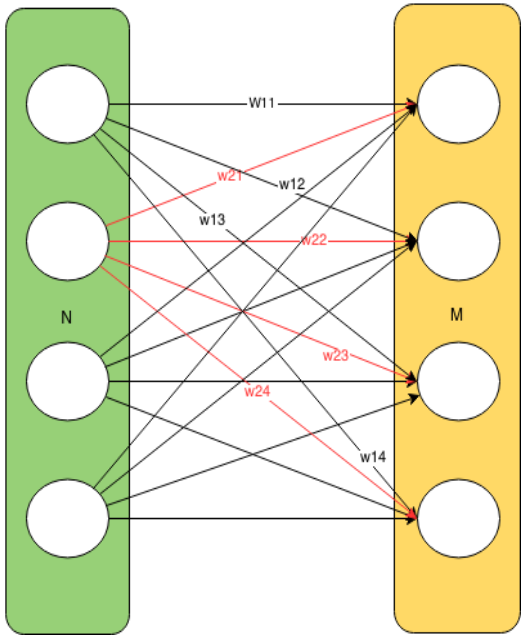
Too many parameters.

- (MNIST 28×28 pixels) * (100 features) = 78,400 params

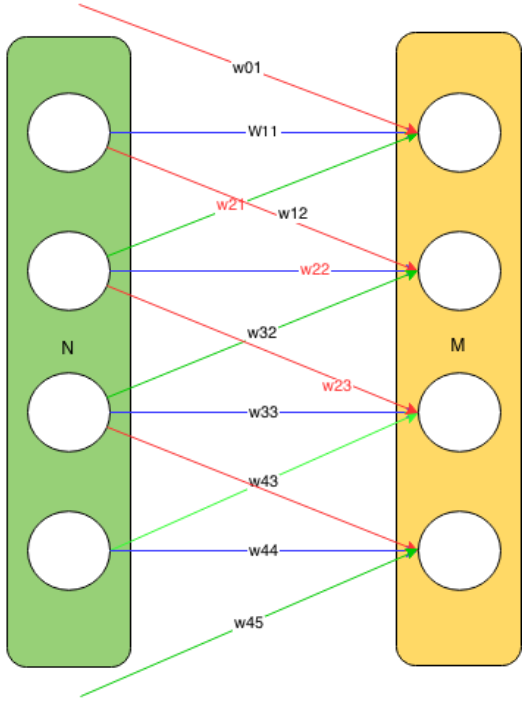
Translation invariance.

- Statistics of a patch (usually) does not depend on its location.

SHARED WEIGHTS



Fully-Connected (Dense)



Locally-Connected with Shared Weights

Edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

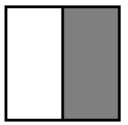
1	0	-1
1	0	-1
1	0	-1

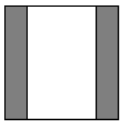
3 x 3

=

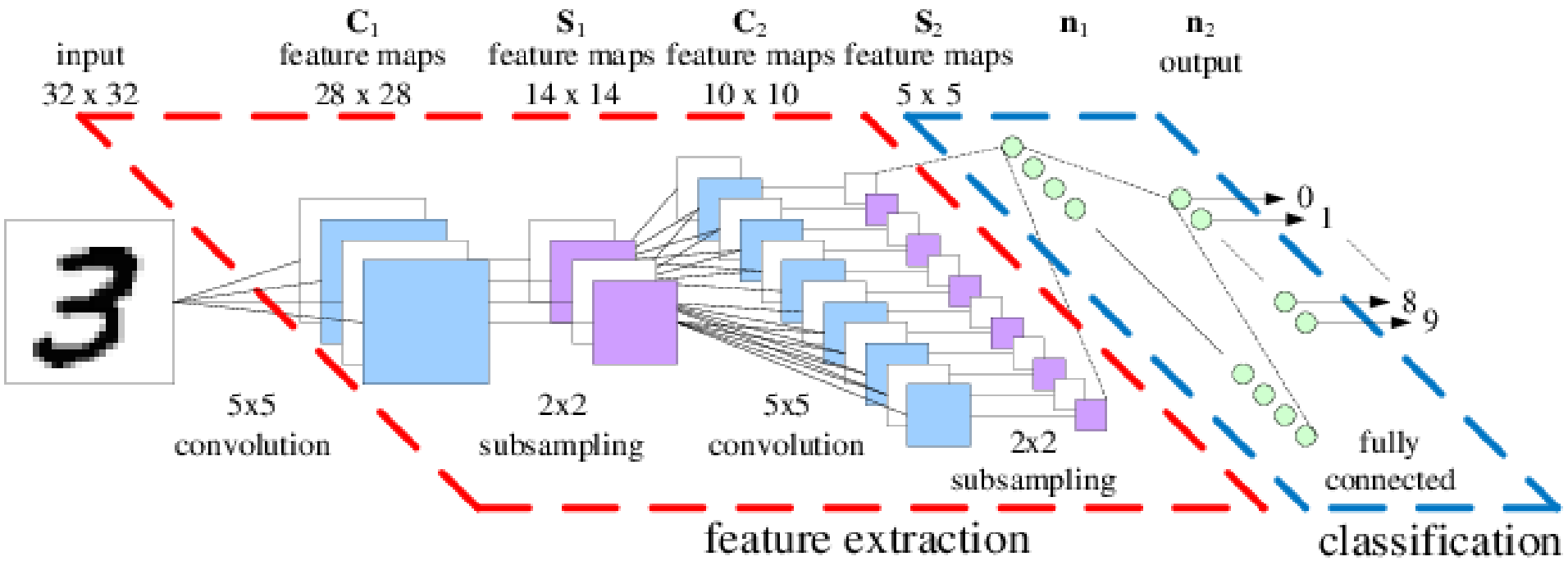
-0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

4 x 4





CNN architecture (e.g. LeNet)



PyTorch code

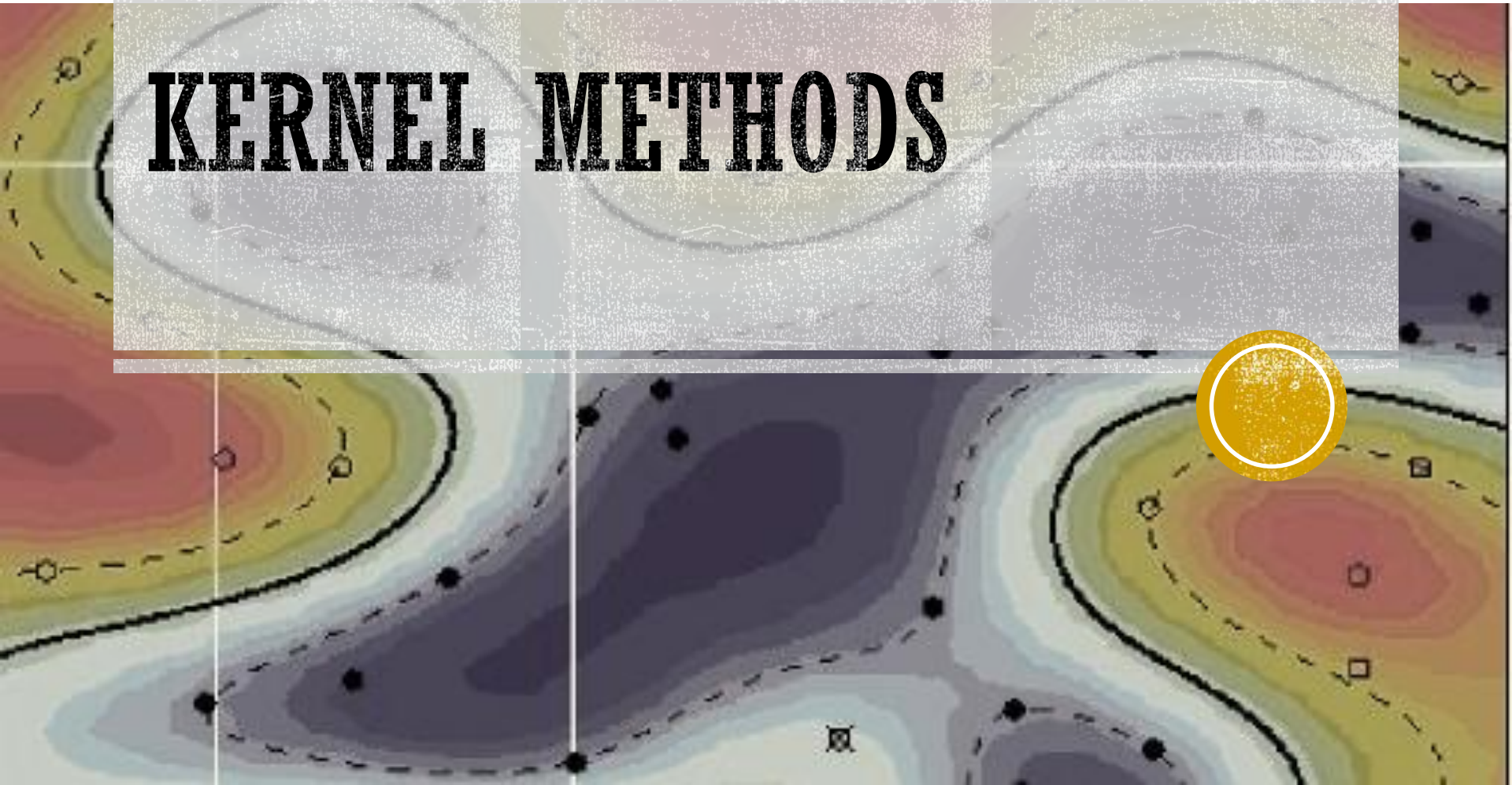
```
import torch.nn as nn
import torch.nn.functional as F

class convNet(nn.Module):
    def __init__(self):
        super(convNet, self).__init__()
        self.conv = nn.Conv2d(in_channels=1, out_channels=20,
kernel_size=5, stride=1, padding=1)

    def forward(self, x):
        x = F.relu(self.conv(x))
    return x
```

Input formats

- NCHW
  - "Number Channels Height Width" or "channels-first"
  - default on pyTorch
  - performs faster on GPUs
- NHWC
  - "Number Height Width Channels" or "channels-last"
  - default on Tensorflow, needed for CPUs

5 min break

KERNEL METHODS



FEATURE MAPS

Example. Non-linear classifiers.

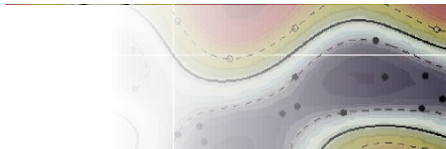
$$x = (x_1, x_2)$$

$$\phi(x) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}\, x_1 x_2, x_1^2, x_2^2\right)$$

$$h(x; \theta, \theta_0)$$

$$= \text{sign}\left(\theta \cdot \phi(x)\right)$$

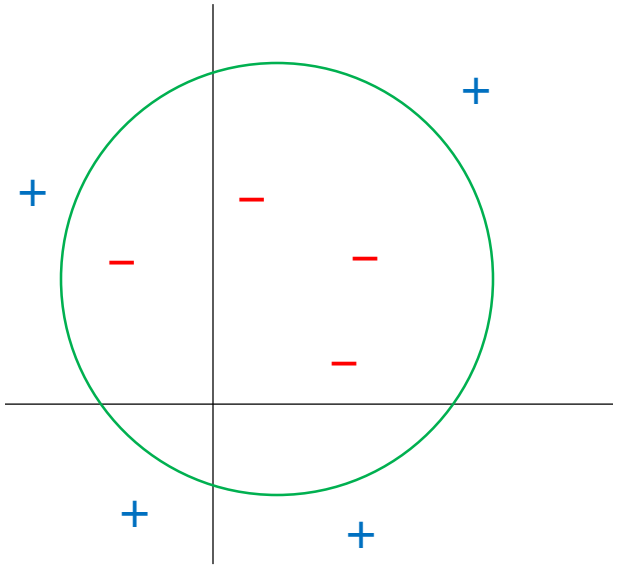
$$= \text{sign}\left(\theta_1 + \theta_2\sqrt{2}x_1 + \theta_3\sqrt{2}x_2 + \theta_4\sqrt{2}x_1 x_2 + \theta_5 x_1^2 + \theta_6 x_2^2\right)$$

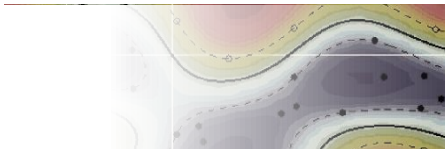


FEATURE MAPS

Example. Non-linear classifiers.

$$h(x;\theta,\theta_0)$$
$$= \text{sign}\left((x_1-1)^2+(x_2-2)^2-9\right)$$
$$= \text{sign}\left(-4-2x_1-4x_2+x_1^2+x_2^2\right)$$





CHALLENGES

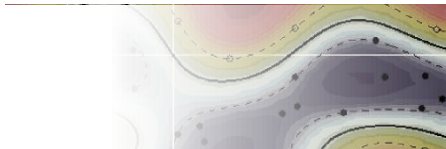
High-Dimensional Features.

$$x = (x_1, x_2, \dots, x_{1000}) \in \mathbb{R}^{1000}$$

$$\phi(x) = \left(1, \dots, \sqrt{2}x_i, \dots, \sqrt{2}x_i x_j, \dots, x_i^2, \dots\right) \in \mathbb{R}^{501501}$$

Inner Products.

Computing $\phi(x) \cdot \phi(x')$ for $x, x' \in \mathbb{R}^{1000}$ requires about 2,004,000 floating point operations.

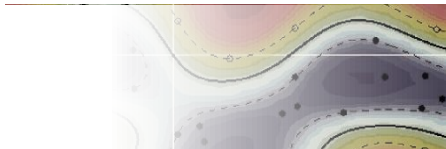


KERNEL FUNCTIONS

Fortunately, many inner products simplify nicely.

$$
\begin{aligned}
K(x, x') &= \phi(x) \cdot \phi(x') \\
&= 1 + 2\sum_i x_i x_i' + 2\sum_{i<j} x_i x_j x_i' x_j' + \sum_i x_i^2 x_i'^2 \\
&= 1 + 2\left(\sum_i x_i x_i'\right) + \left(\sum_i x_i x_i'\right)^2 \\
&= (x \cdot x' + 1)^2
\end{aligned}
$$

For $x, x' \in \mathbb{R}^{1000}$, computing this requires only about 2000 floating point operations, less than the 501,501 operations needed for $\phi(x)$.



Recall for linear regression with L2 regularization, we aim to minimize the loss function

$$\mathcal{L}(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(\left\langle \theta, \phi\left(x^{(i)}\right)\right\rangle - y^{(i)}\right)^2 + \frac{\lambda}{2}\|\theta\|^2 .$$

Taking the gradient and setting to zero, we have

$$\begin{aligned}\theta &= -\frac{1}{\lambda}\sum_{i=1}^{m}\left(\left\langle \theta, \phi\left(x^{(i)}\right)\right\rangle - y^{(i)}\right)\phi\left(x^{(i)}\right) \\ &= \sum_{i=1}^{m} a_i \phi\left(x^{(i)}\right),\end{aligned}$$

where $a_i = -\frac{1}{\lambda}\left(\left\langle \theta, \phi\left(x^{(i)}\right)\right\rangle - y^{(i)}\right)$.

Dual representation

We can write the expression above as $\mathbf{\Phi}^T \mathbf{a}$, where $\mathbf{\Phi}$ is the design matrix whose i^{th} row is given by $\phi\left(x^{(i)}\right)^T$, and where $a = (a_i, \ldots, a_m)^T$.
Substituting $\theta = \mathbf{\Phi}^T a$ into the loss function, we have

$$\mathcal{L}(a) = \frac{1}{2}\left\langle \mathbf{\Phi}\mathbf{\Phi}^T a, \mathbf{\Phi}\mathbf{\Phi}^T a \right\rangle - \left\langle \mathbf{\Phi}\mathbf{\Phi}^T a, y \right\rangle + \frac{1}{2}\|y\|^2 + \frac{\lambda}{2}\left\langle a, \mathbf{\Phi}\mathbf{\Phi}^T a \right\rangle,$$

where $y = \left(y^{(1)}, \ldots, y^{(m)}\right)^T$.

Now define $K = \mathbf{\Phi}\mathbf{\Phi}^T$, where the ij^{th} element of K is given by

$$\left\langle \phi\left(x^{(i)}\right), \phi\left(x^{(j)}\right)\right\rangle = k\left(x^{(i)}, x^{(j)}\right),$$

and we have

$$\mathcal{L}(a) = \frac{1}{2}\left\langle a, K^2 a\right\rangle - \langle Ka, y\rangle + \frac{1}{2}\|y\|^2 + \frac{\lambda}{2}\langle a, Ka\rangle.$$

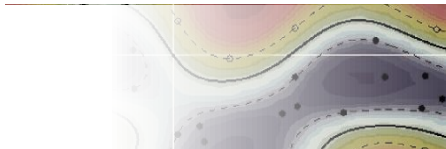
Taking the gradient with respect to a and setting it to zero, we obtain

$$a = (K + \lambda I)^{-1} y,$$

and we see that we can express the solution entirely in terms of the kernel.

KERNEL TRICK

The kernel trick refers to the strategy of converting a learning algorithm and the resulting predictor into ones that involve only the computation of the kernel $K(x, x') = \phi(x) \cdot \phi(x')$ but not of the feature map $\phi(x)$.



KERNEL FUNCTIONS

'Infinite dimensional' positive definite matrices

Definition.

A function $K: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel function* if

1. $K(x, y) = K(y, x)$ for all $x, y \in \mathbb{R}^d$,
2. given $n \in \mathbb{N}$ and $x^{(1)}, x^{(2)}, \ldots, x^{(n)} \in \mathbb{R}^d$,

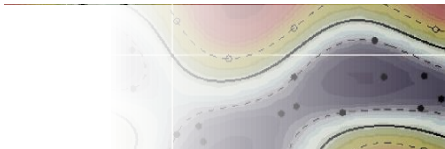
   the *Gram matrix* K with entries

$$K_{ij} = K\left(x^{(i)}, x^{(j)}\right)$$

   is positive semidefinite.

Example. $K(x, x') = \phi(x) \cdot \phi(x')$

Can be shown that all kernel functions are of this form!



EXAMPLES

Linear Kernel.

$$K(x, x') = x \cdot x'$$

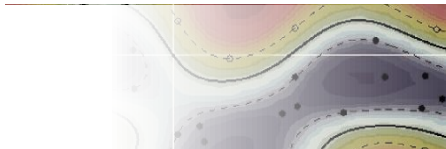
Polynomial Kernel.

$$K(x, x') = (x \cdot x' + 1)^k$$

Radial Basis Kernel.

$$K(x, x') = \exp\left(-\frac{1}{2}\|x - x'\|^2\right)$$

Feature map $\phi(x)$ is infinite dimensional!



Constructing kernels

Assuming $k_1(x, y)$ and $k_2(x, y)$ are valid kernels, the following kernels are also valid:

(i) $k(x, y) = ak_1(x, y) + bk(x, y), \quad a, b > 0$

(ii) $k(x, y) = k_1(x, y)k_2(x, y)$

(iii) $k(x, y) = p(k_1(x, y))$, where p is polynomial with positive coefficients

(iv) $k(x, y) = \exp(k_1(x, y))$

(v) $k(x, y) = f(x)k_1(x, y)f(y)$, for any function f

Why infinite dimensional?

Theorem (Mercer)

Let $\mathcal{X}$ be a closed and bounded subset of $\mathbb{R}^d$ and μ be a finite measure on $\mathcal{X}$. Let $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a continuous symmetric positive-definite kernel on $\mathcal{X}$. Define the operator

$$T_K f(x) = \int_{\mathcal{X}} K(x, t) f(t) \, \mathrm{d}\mu(t)$$

for all $f \in L^2(\mathcal{X}, \mu)$. Then there exists an orthonormal basis $\{\phi_i\}$ of eigenfunctions of T_K, with corresponding non-negative eigenvalues λ_i, such that

$$K(x, y) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(y)$$

where the convergence is absolute and uniform.

- Many feature mappings can be constructed without Mercer's theorem.
- However, Mercer's theorem guarantees a, possibly infinite-dimensional, feature vector $\tilde{\phi} = \left(\sqrt{\lambda_1}\phi_1(x), \sqrt{\lambda_2}\phi_2(x), \ldots\right)$ and

$$K(x, y) = \left\langle \tilde{\phi}(x), \tilde{\phi}(y) \right\rangle .$$

  (Note that if K is "simple" enough, it is possible that only a finite number of the eigenvalues are non-zero, making $\tilde{\phi}$ finite-dimensional.)
- Thus we get richer representations as we can work with arbitrarily high, and even infinite-dimensional, feature spaces.

Singular value decomposition (SVD)

Theorem

Given any real m x n matrix M, there exists a factorization of M of the form

$$M = U\Sigma V^T$$

where

- *U is an orthogonal m x m matrix,*
- *Σ is a m x n matrix with non-negative real numbers on the diagonal and zero elsewhere,*
- *V is an orthogonal n x n matrix.*

The diagonal entries of Σ are called the singular values of M.