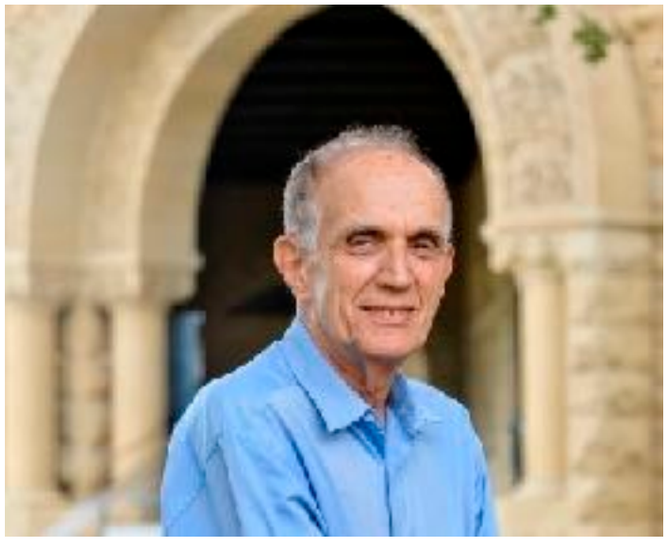
Ensemble methods

Bagging
(Bootstrap Aggregating)



Bradley Efron (1979)

Leo Breiman (1994)

Bootstrap

Why working with a single data set?

```
[[ 0.524]
 [ 0.4  ]
 [ 0.361]
 [ 0.078]
 [ 0.859]
 [ 0.654]
 [ 0.384]
 [ 0.738]
 [ 0.156]
 [ 0.572]]
```

Bootstrap

Why working with a single data set?

Create many data sets by sampling with replacement:

```
[[ 0.524]
 [ 0.4  ]
 [ 0.361]
 [ 0.078]
 [ 0.859]
 [ 0.654]
 [ 0.384]
 [ 0.738]
 [ 0.156]
 [ 0.572]]
```

```
[[ 0.361]
 [ 0.572]
 [ 0.156]
 [ 0.361]
 [ 0.4  ]
 [ 0.384]
 [ 0.572]
 [ 0.654]
 [ 0.4  ]
 [ 0.384]]
```

```
[[ 0.361]
 [ 0.156]
 [ 0.4  ]
 [ 0.654]
 [ 0.654]
 [ 0.859]
 [ 0.384]
 [ 0.4  ]
 [ 0.524]
 [ 0.859]]
```

```
[[ 0.156]
 [ 0.156]
 [ 0.524]
 [ 0.078]
 [ 0.524]
 [ 0.859]
 [ 0.361]
 [ 0.078]
 [ 0.654]
 [ 0.4  ]]
```

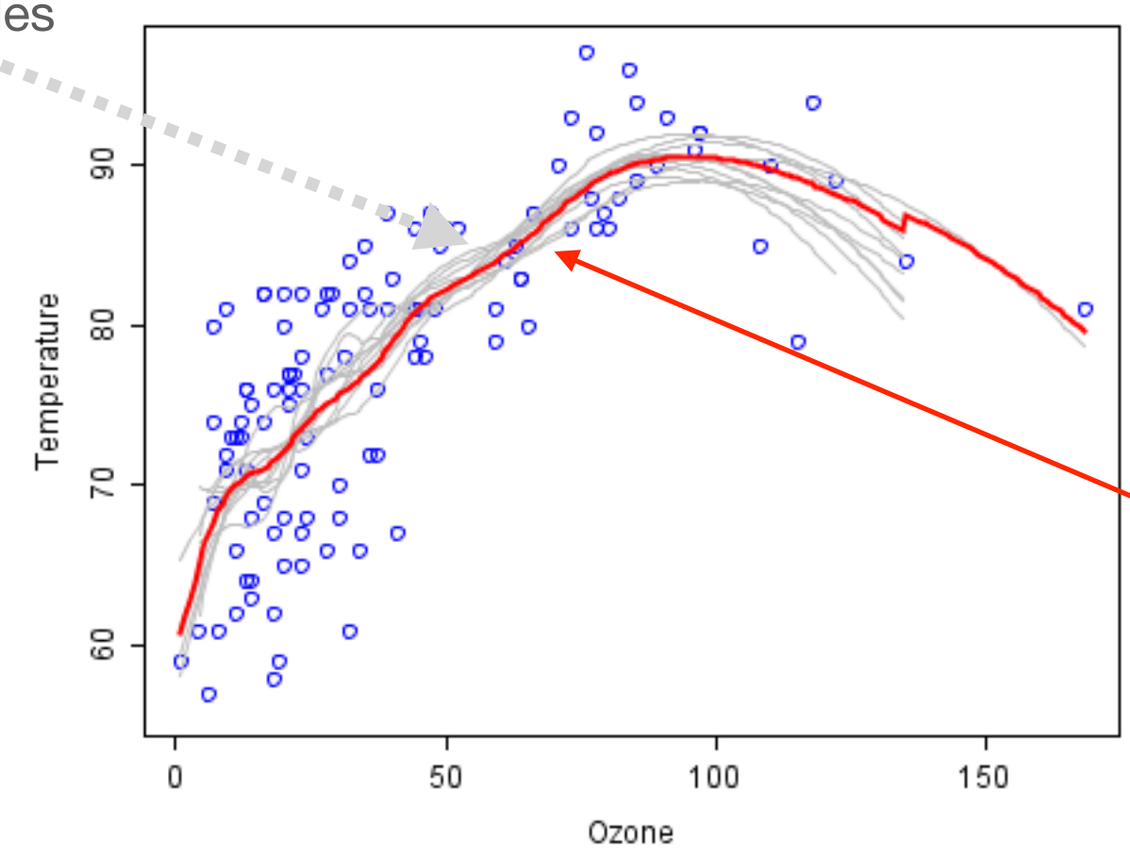
```
[[ 0.078]
 [ 0.524]
 [ 0.859]
 [ 0.384]
 [ 0.859]
 [ 0.361]
 [ 0.361]
 [ 0.738]
 [ 0.572]
 [ 0.156]]
```

Consider these datasets are equally valid

Many uses: error analysis, statistical tests, …

Bagging
(Bootstrap Aggregating)

Fits on bootstrap samples

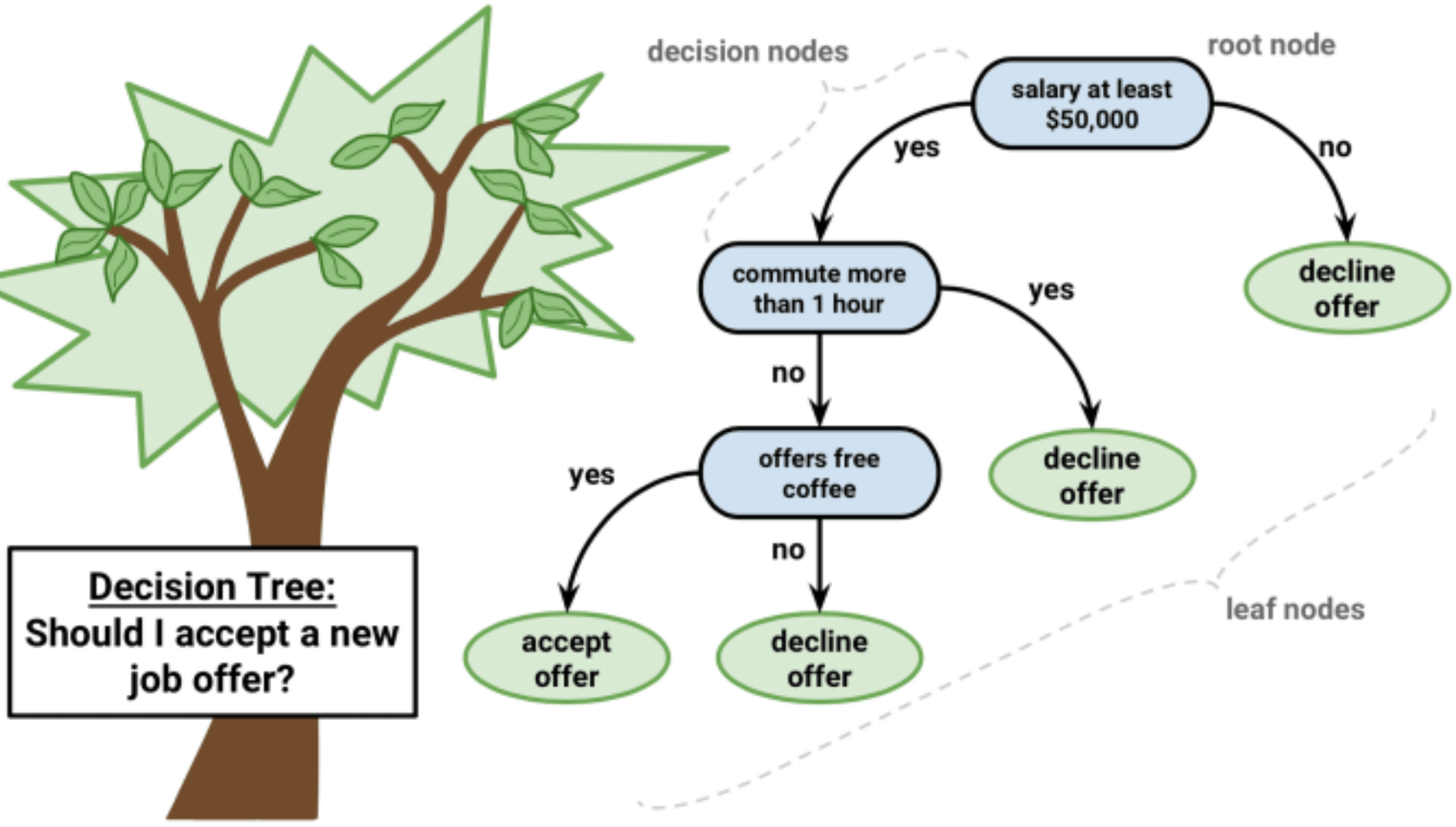


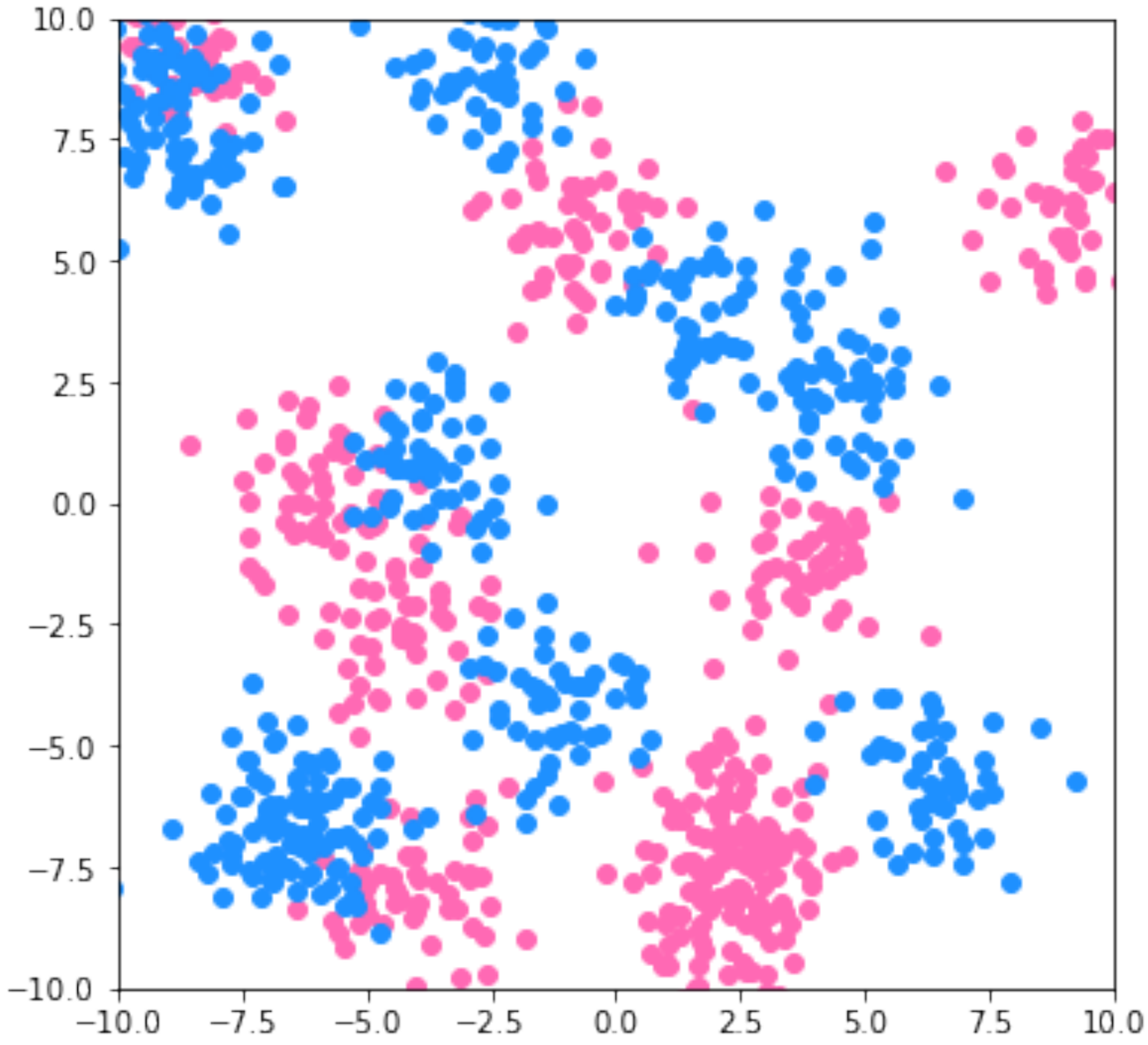
Ozone data

Averaging models reduces overfitting!

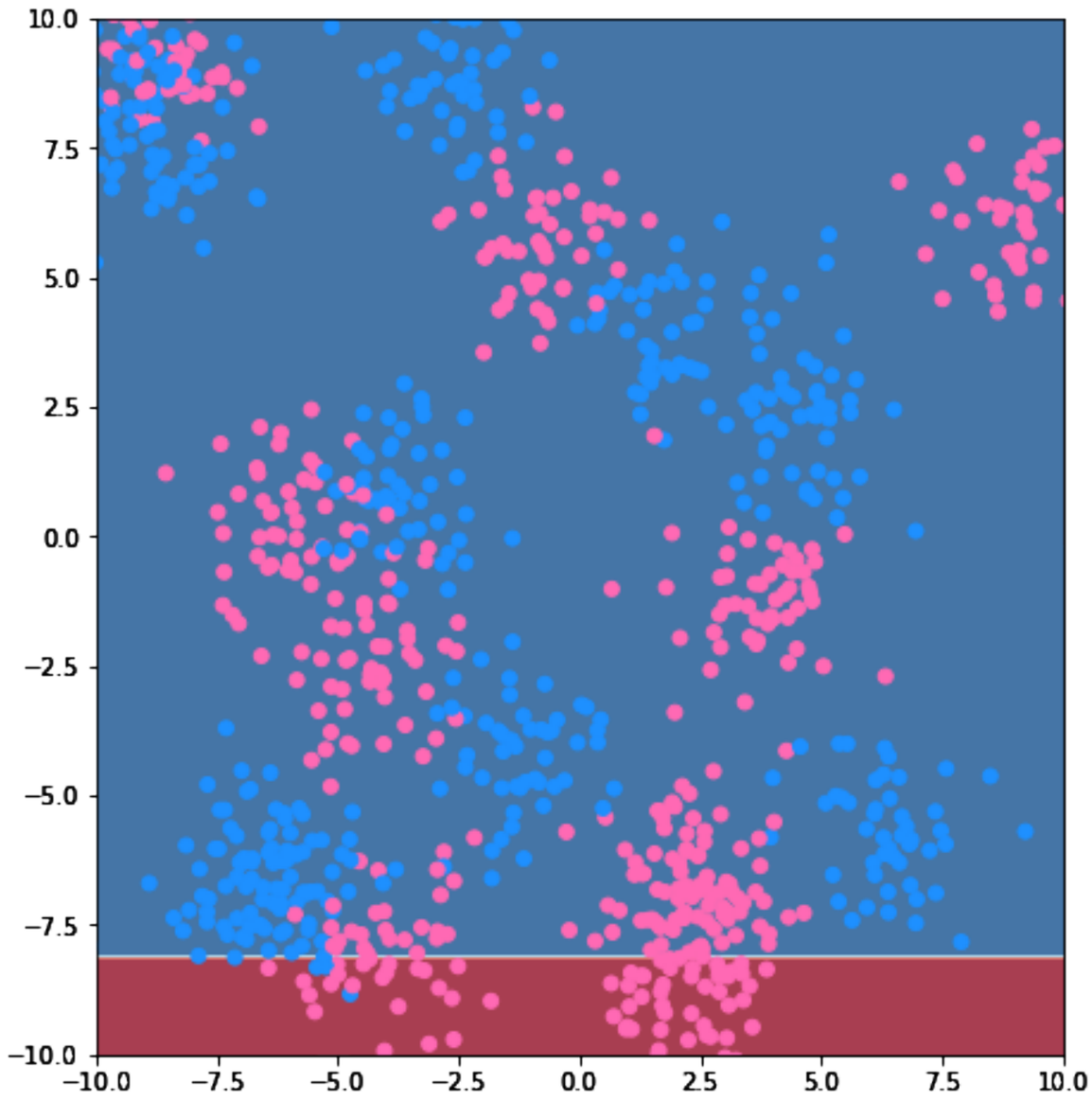
And for classification?

Decision trees!

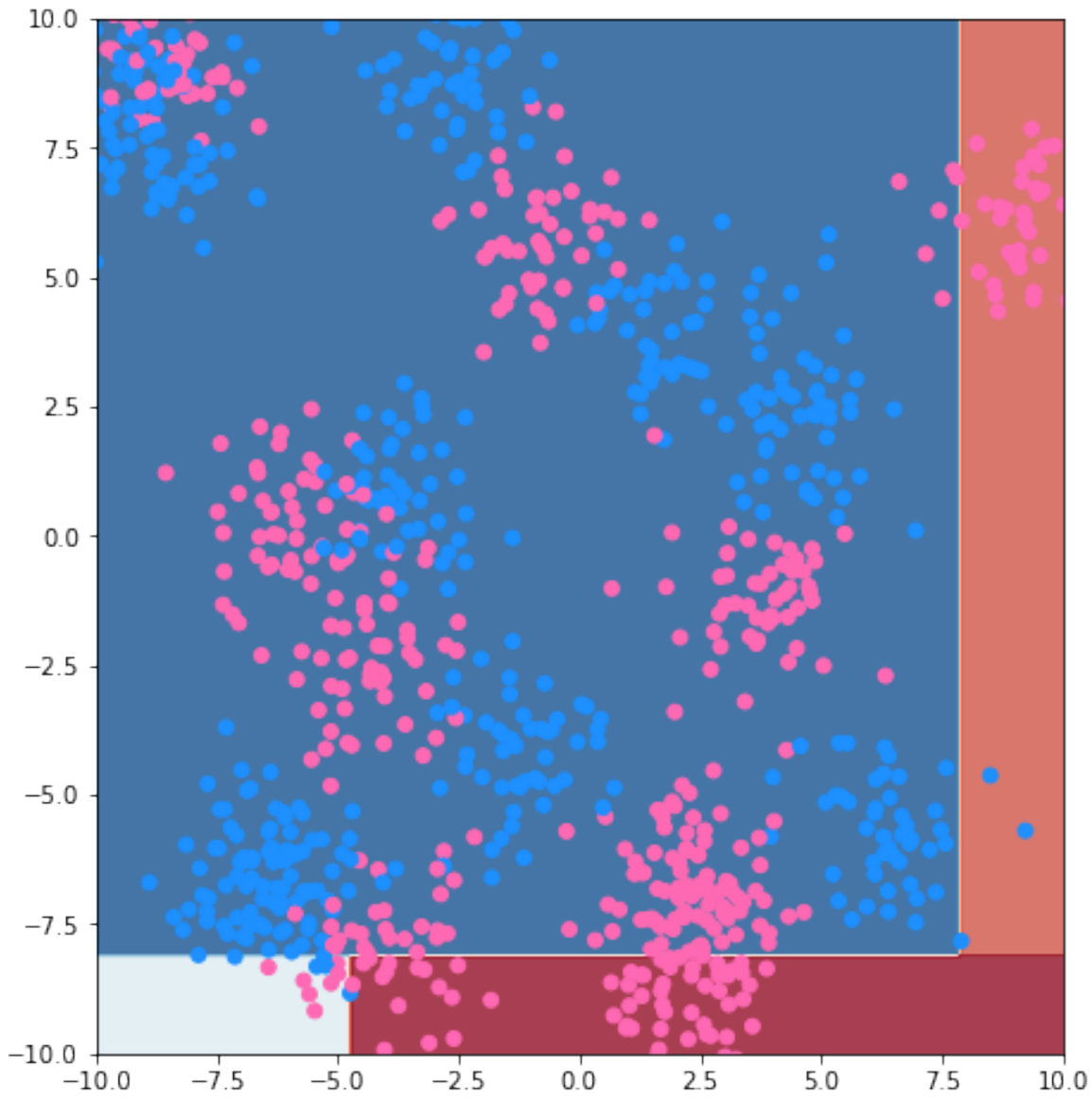



10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

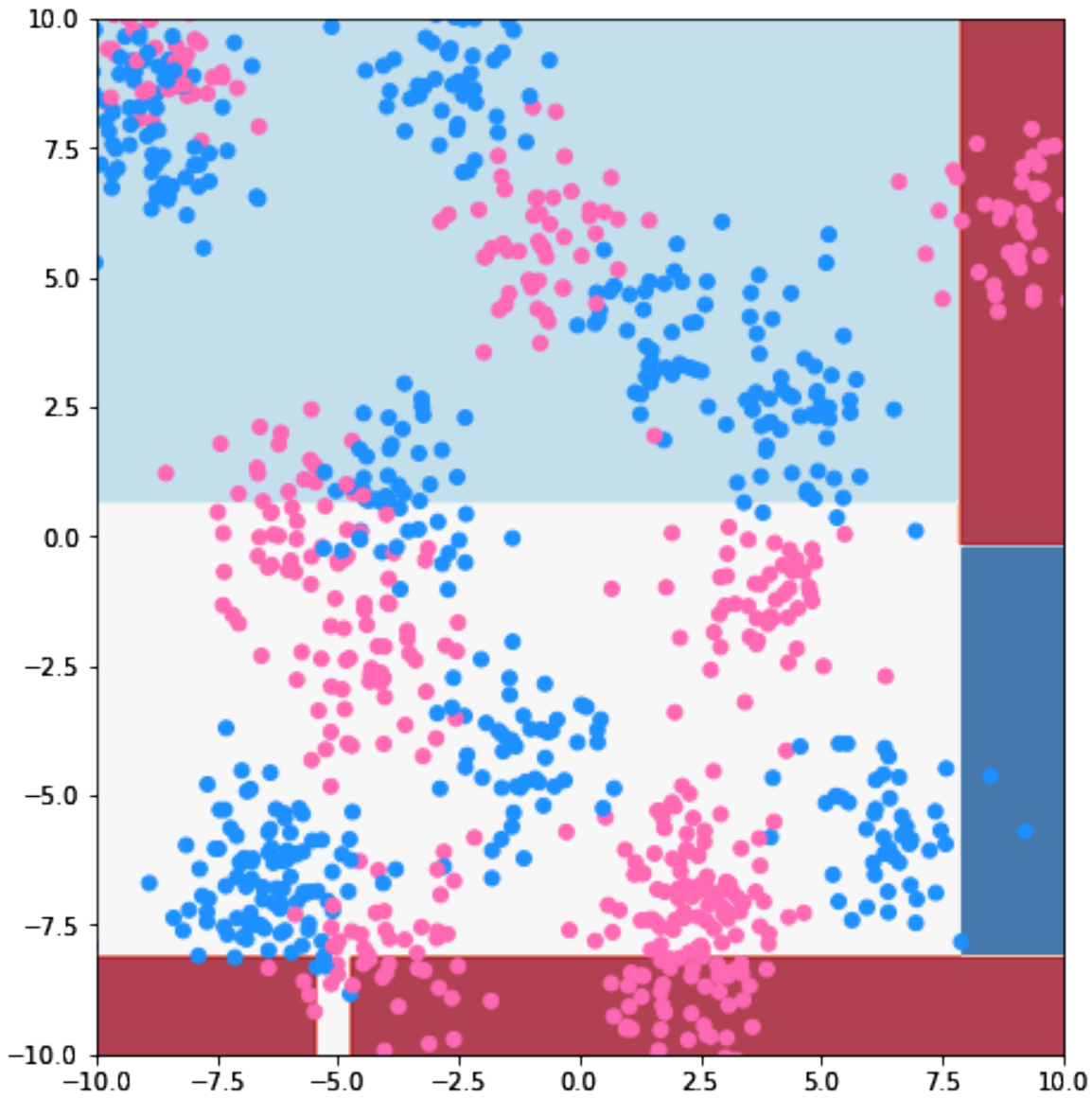
Decision tree, depth=1



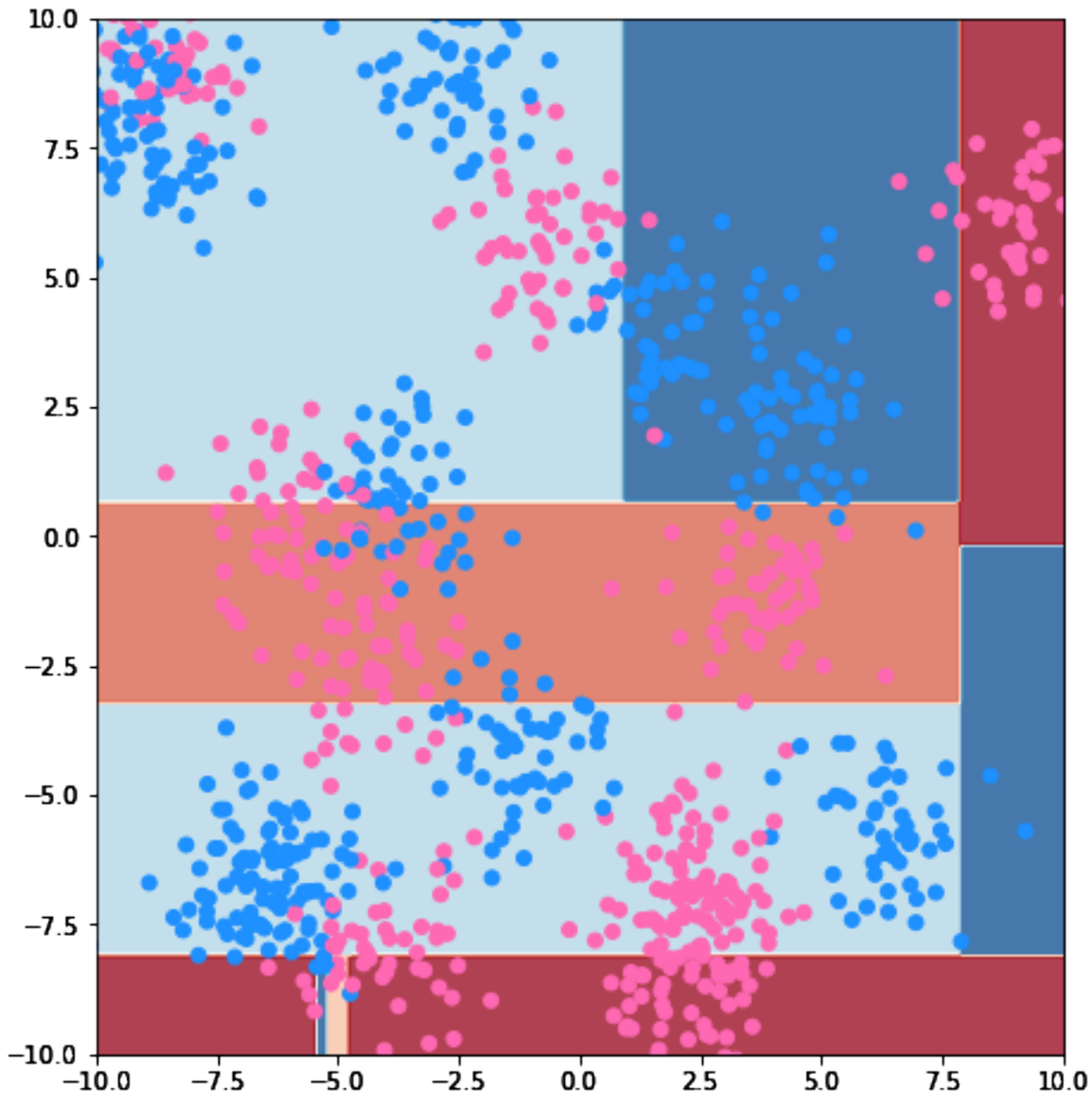
Decision tree, depth=2



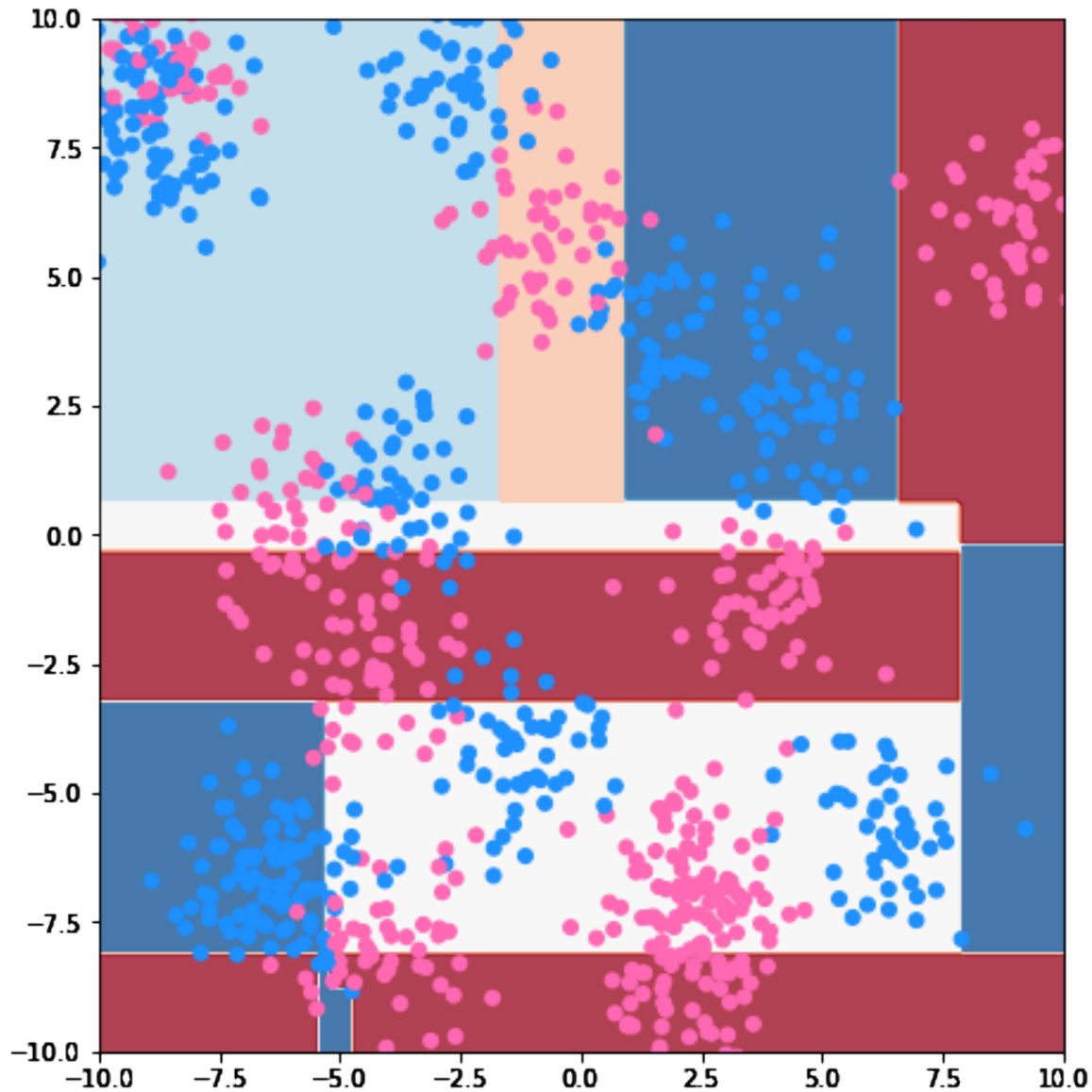
Decision tree, depth=3



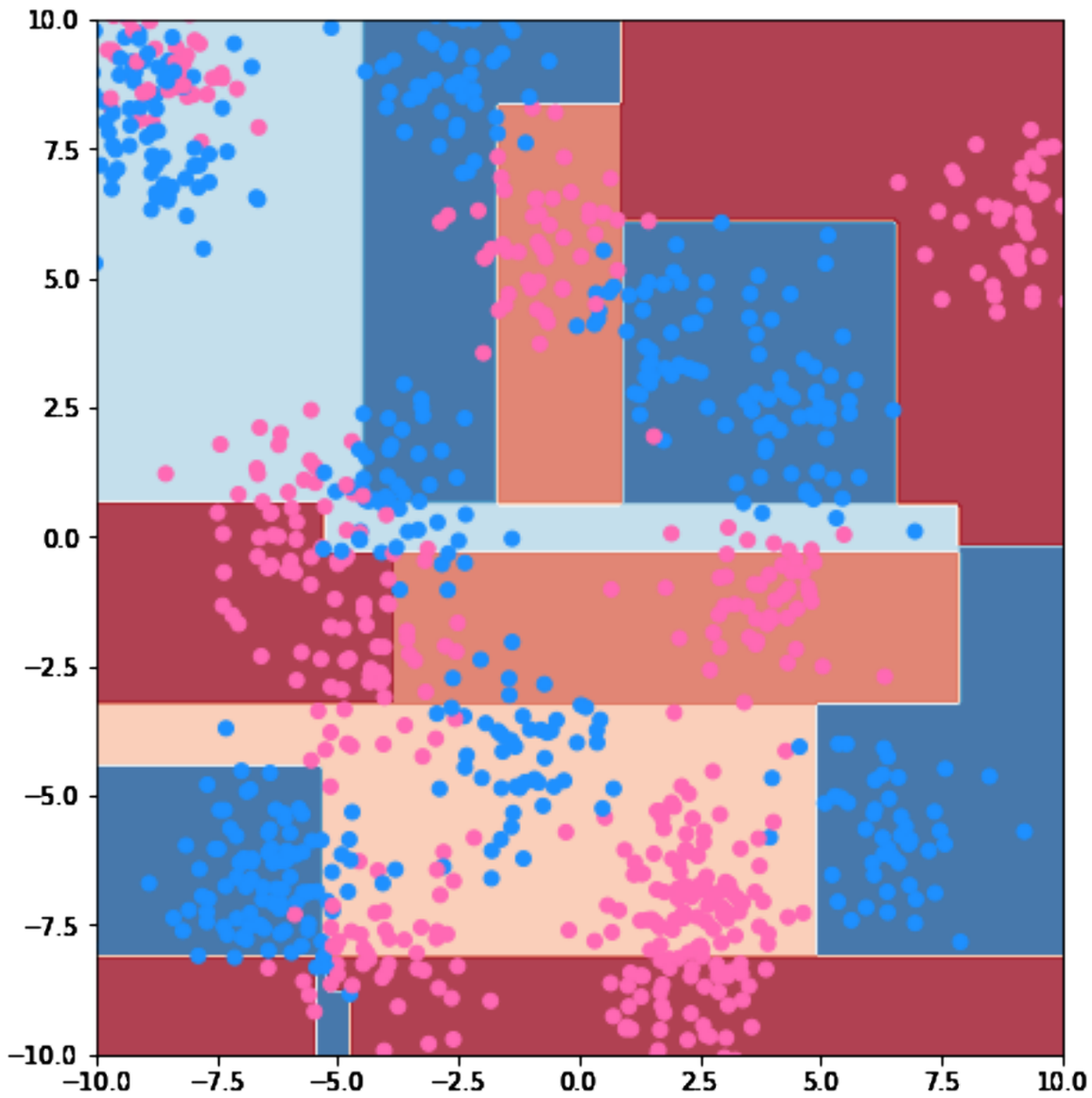
Decision tree, depth=4



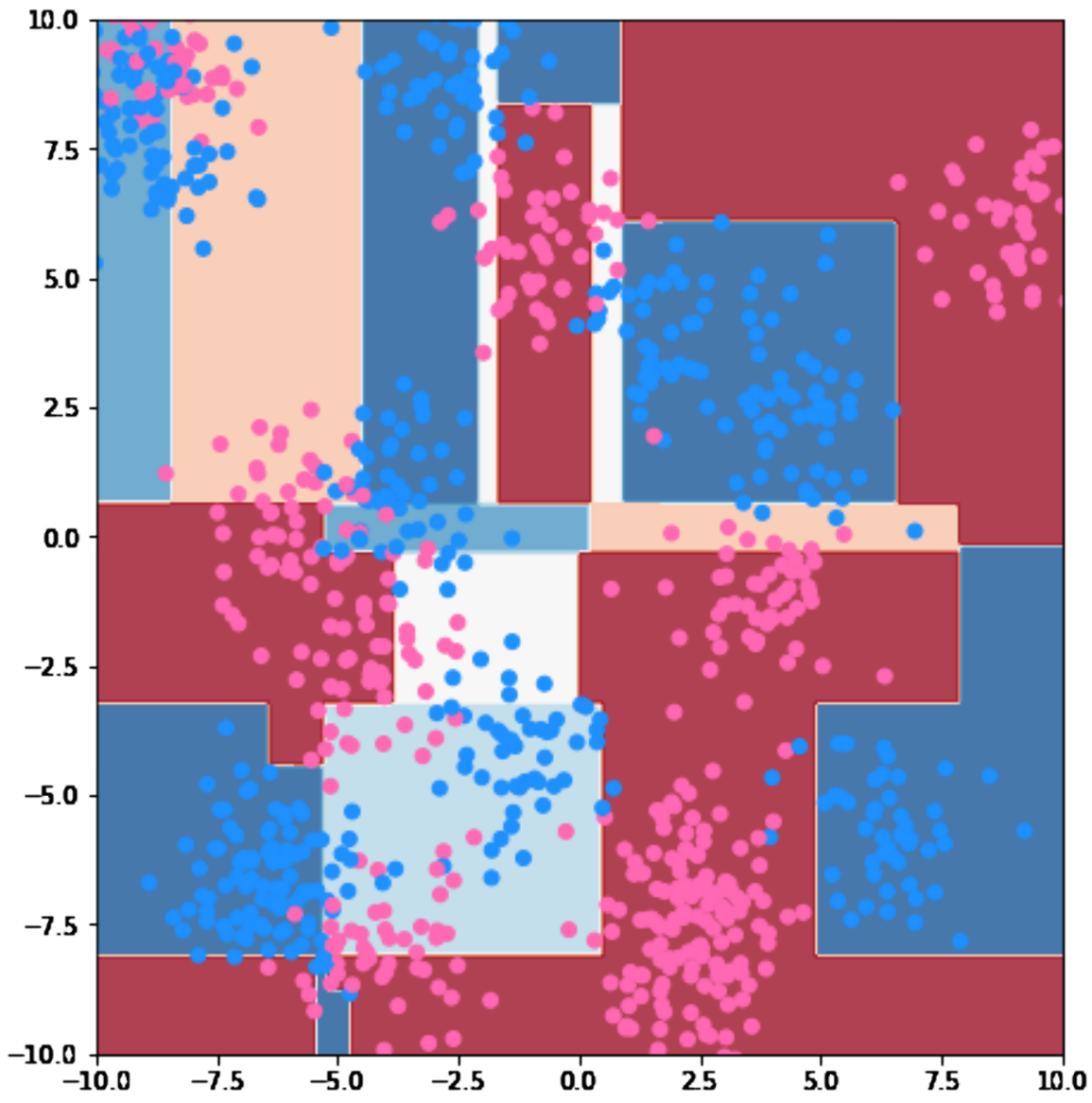
Decision tree, depth=5



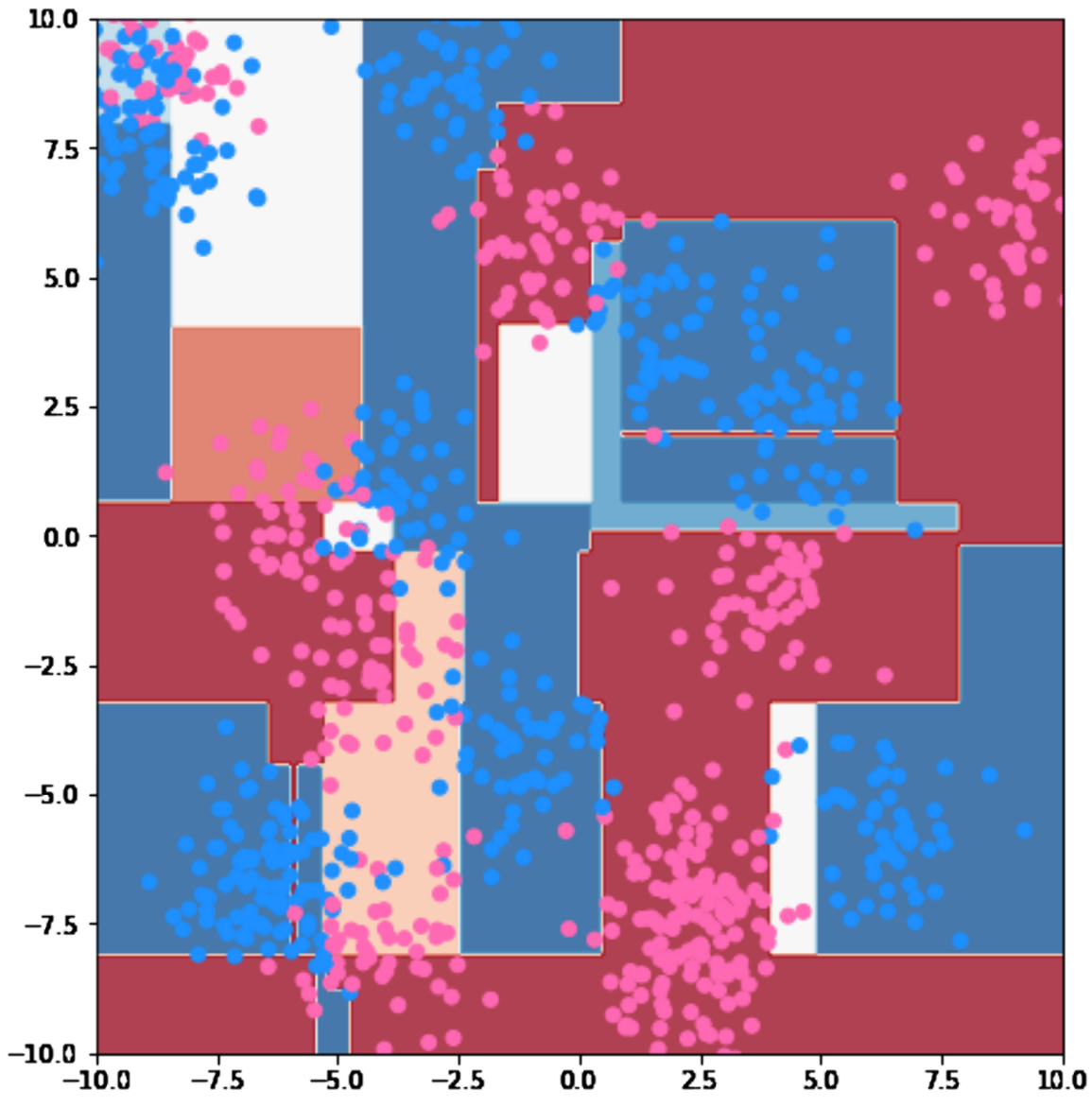
Decision tree, depth=6



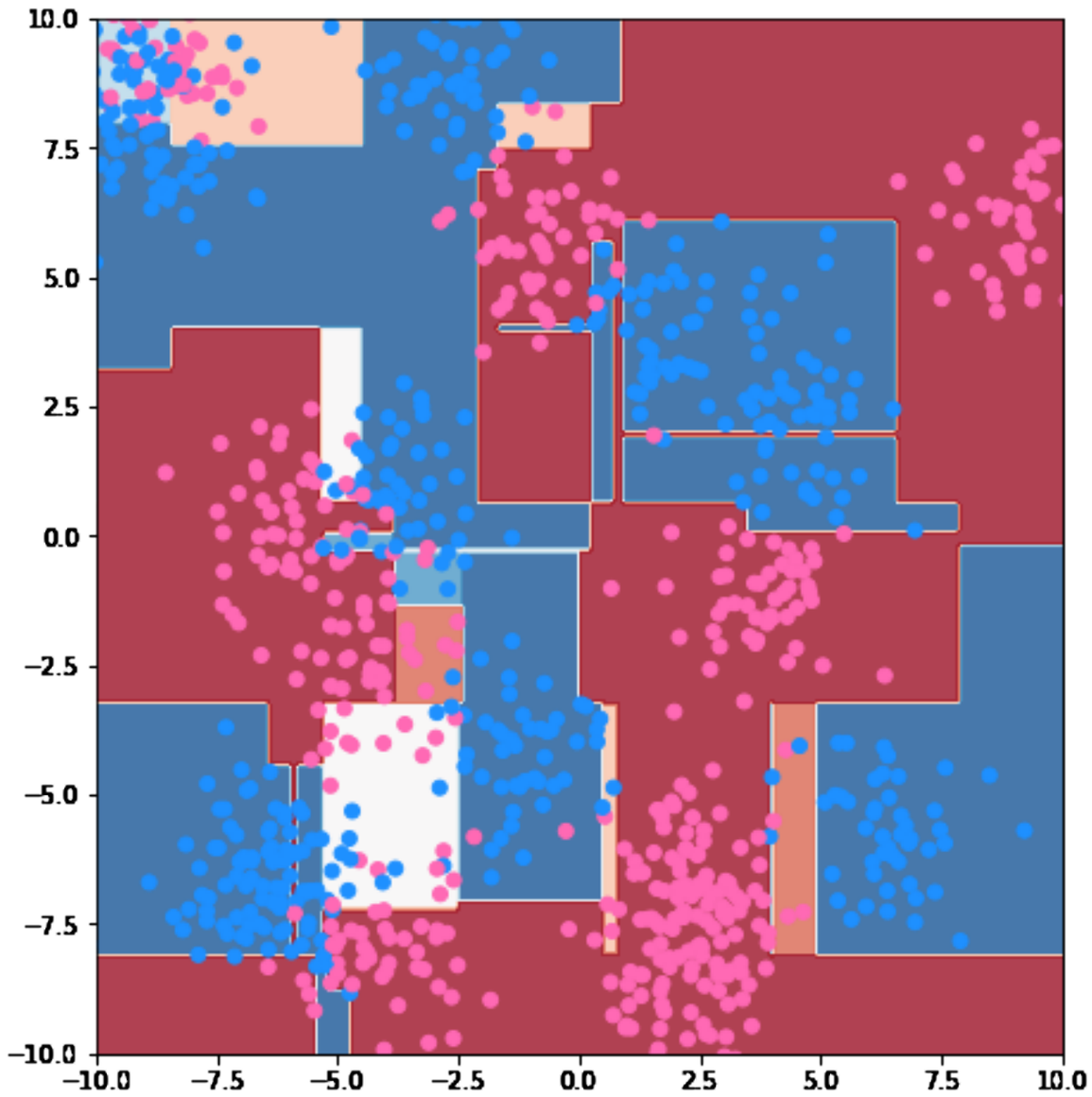
Decision tree, depth=7

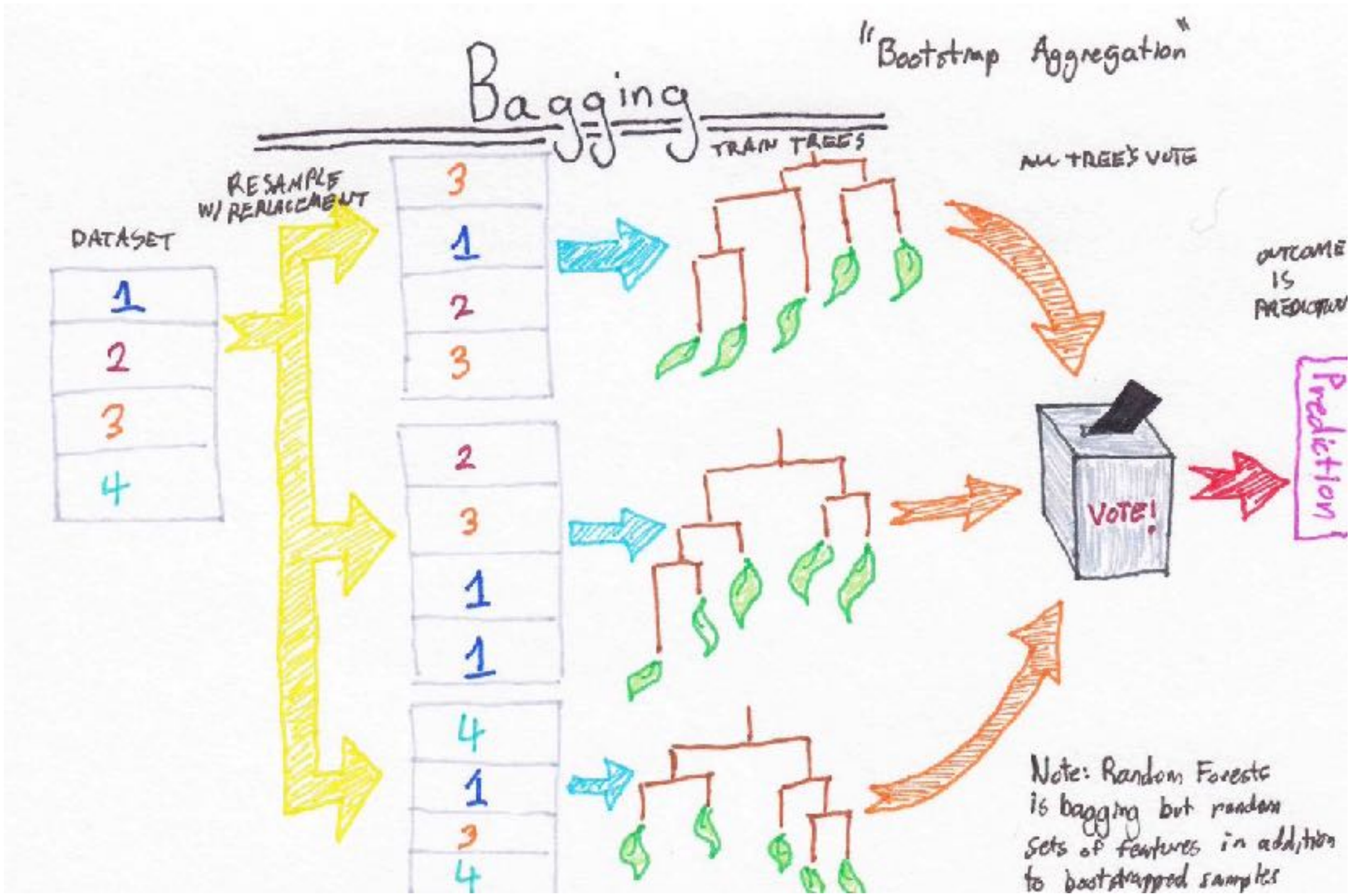


Decision tree, depth=8

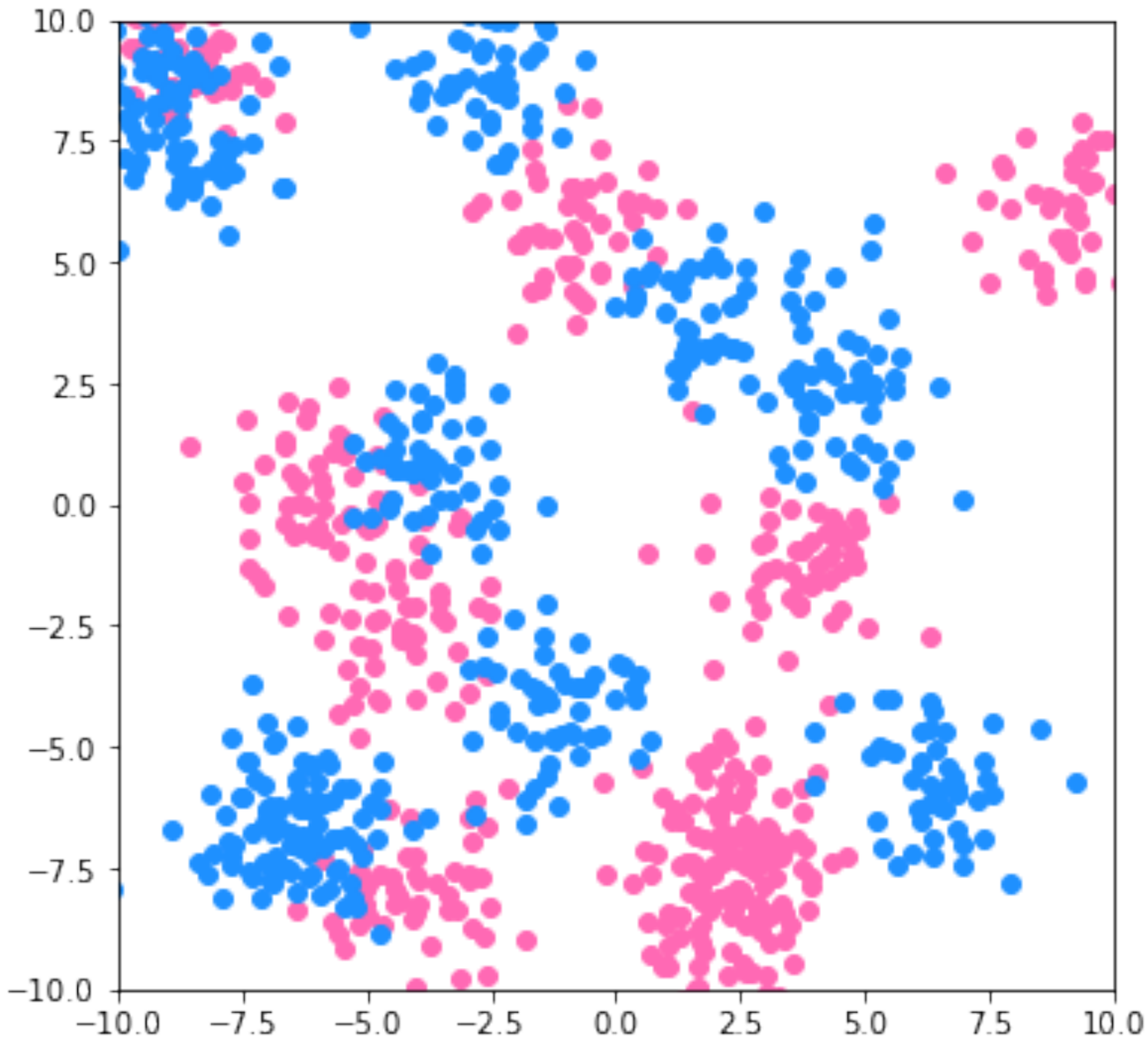


Decision tree, depth=9

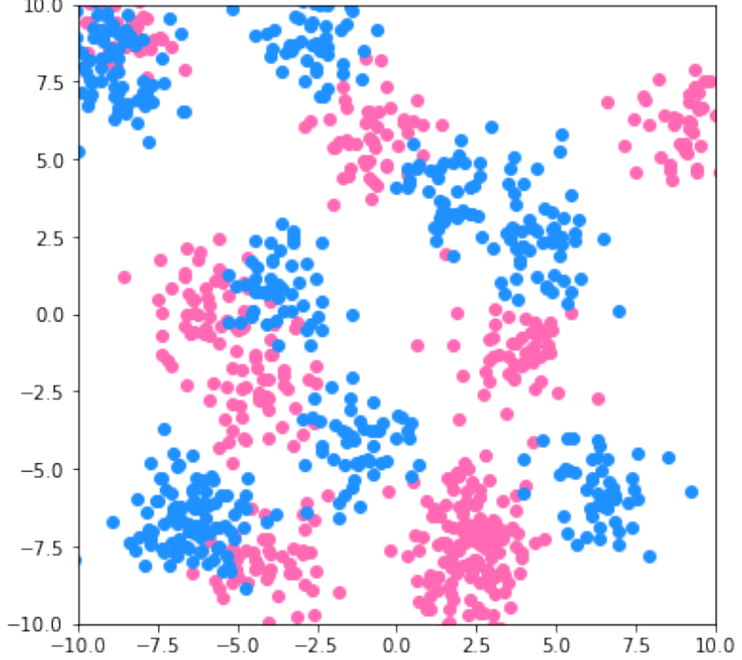



Bagging
"Bootstrap Aggregation"
DATASET
1
2
3
4
RESAMPLE W/ REPLACEMENT
3
1
2
3
2
3
1
1
4
1
3
4
TRAIN TREES
ALL TREES VOTE
VOTE!
OUTCOME IS PREDICTION
Prediction
Note: Random Forests is bagging but random sets of features in addition to bootstrapped samples

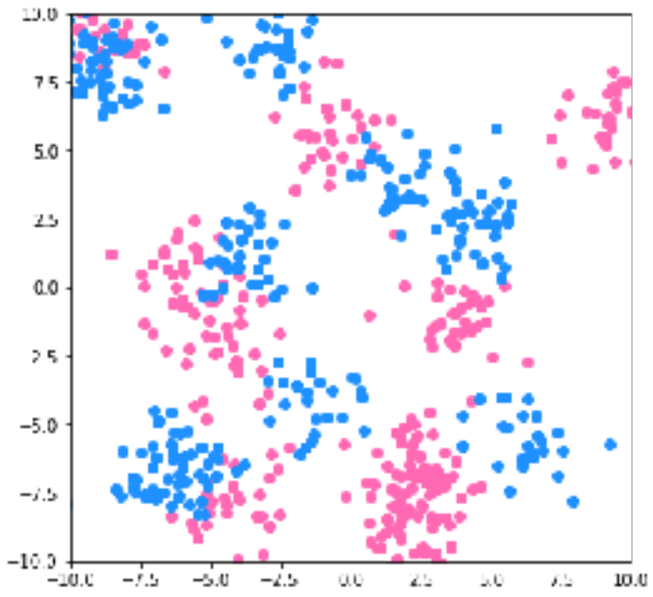
The original set

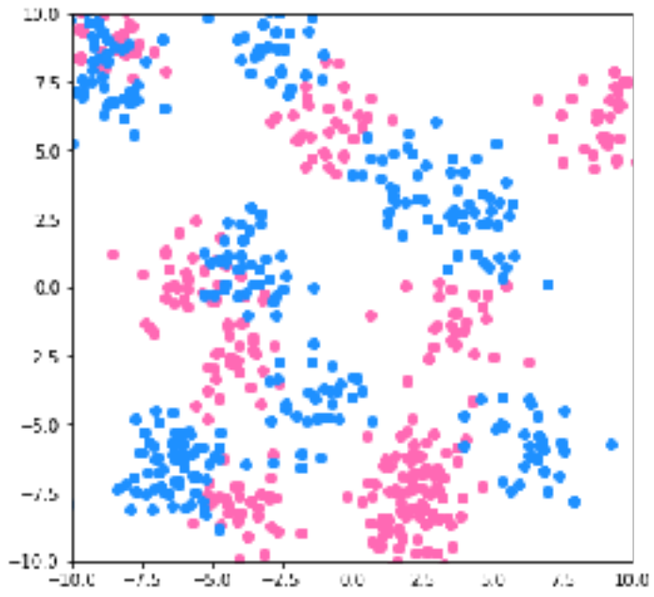


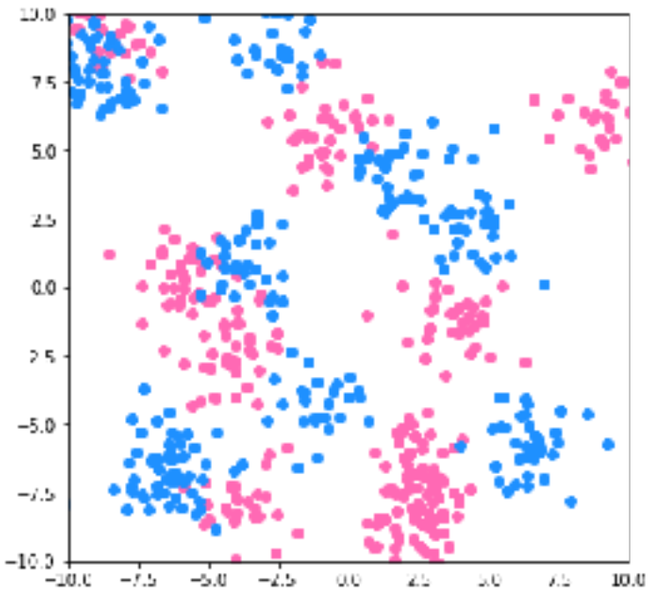
BOOSTRAP!!

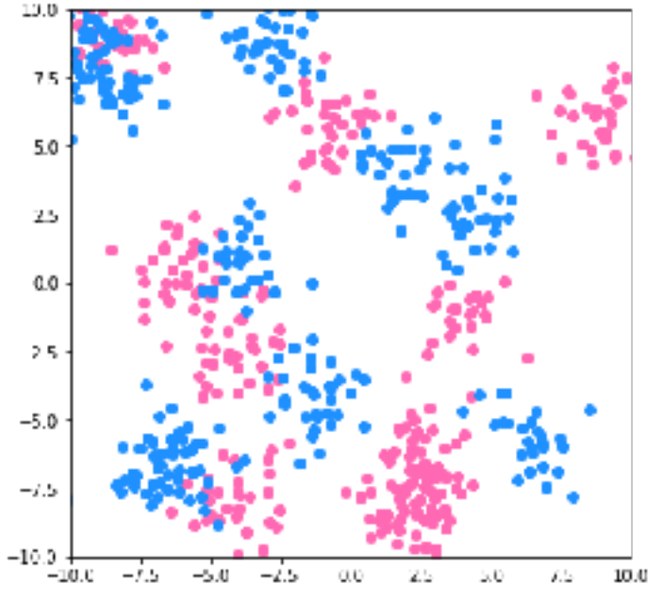


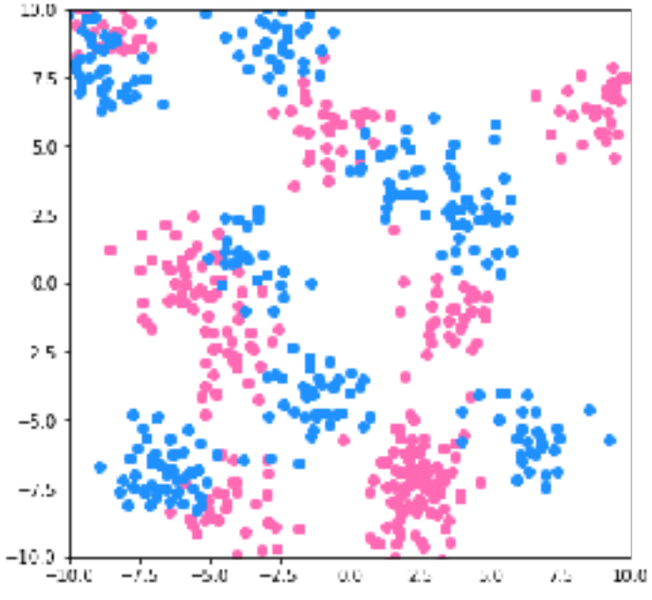
BOOSTRAP!!

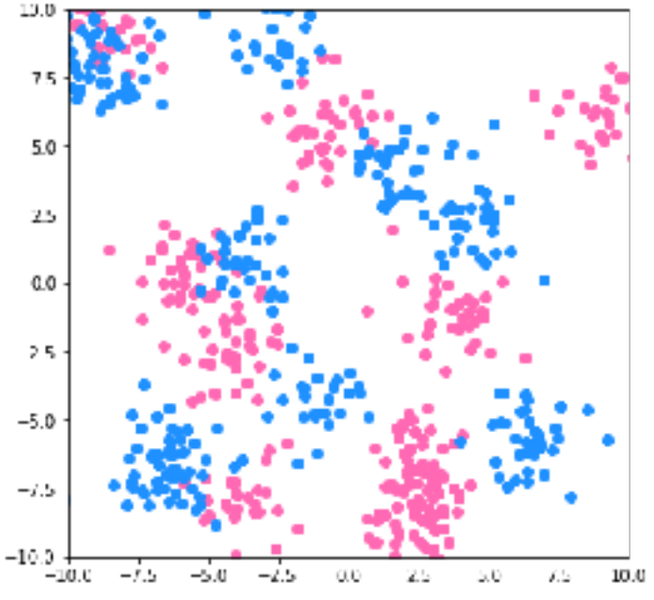




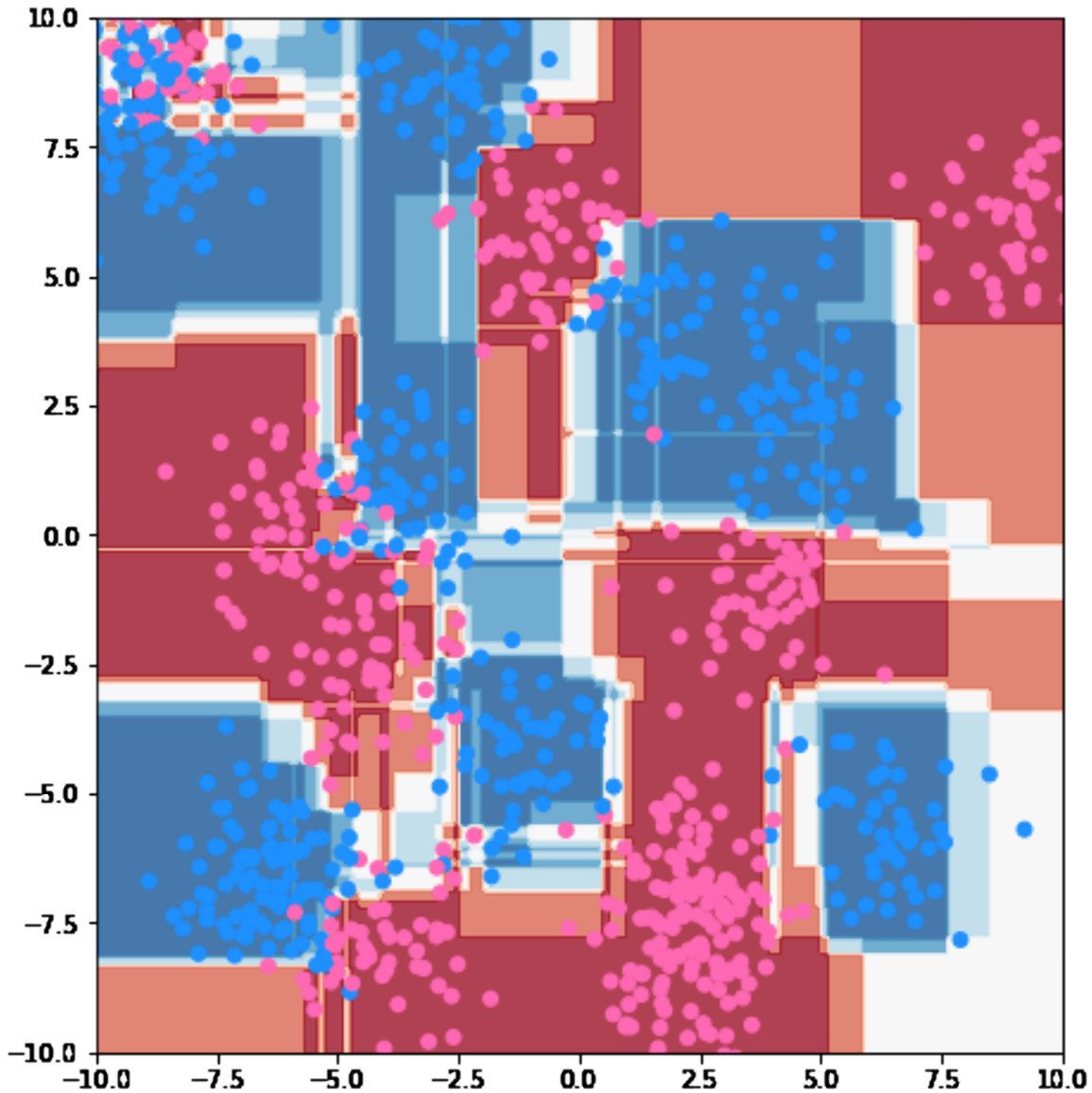




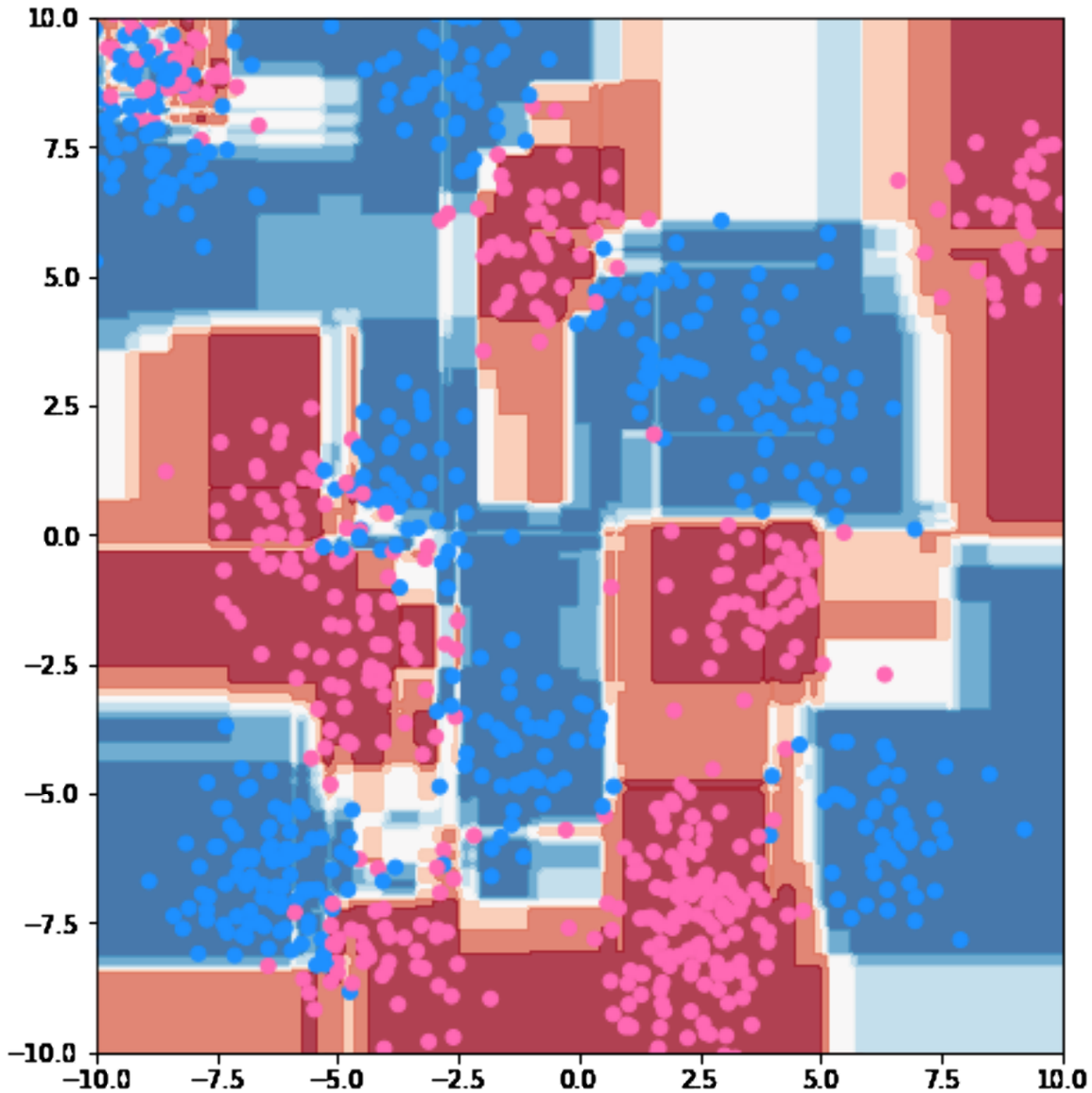




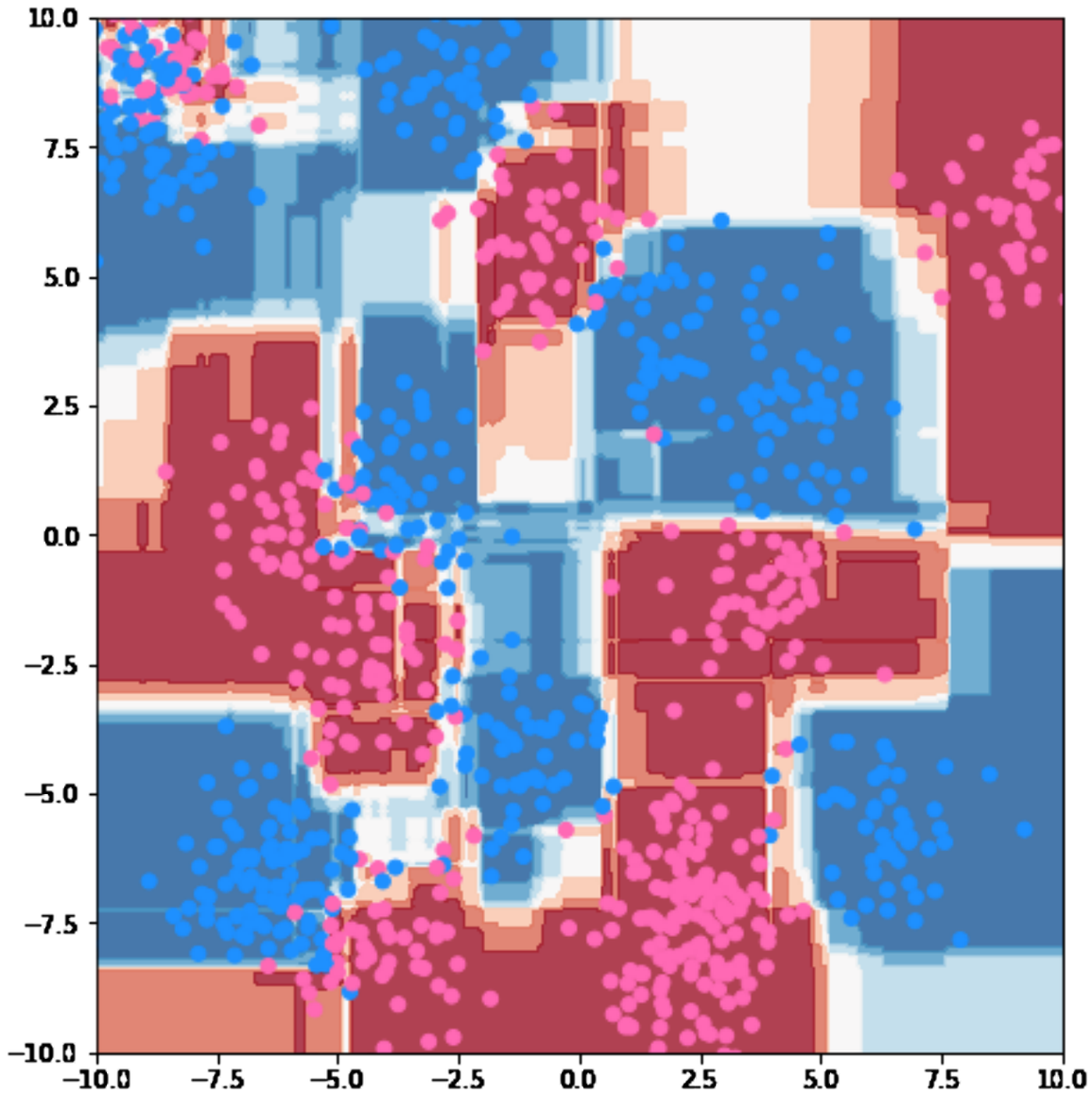
Bagging 5 Decision tree!



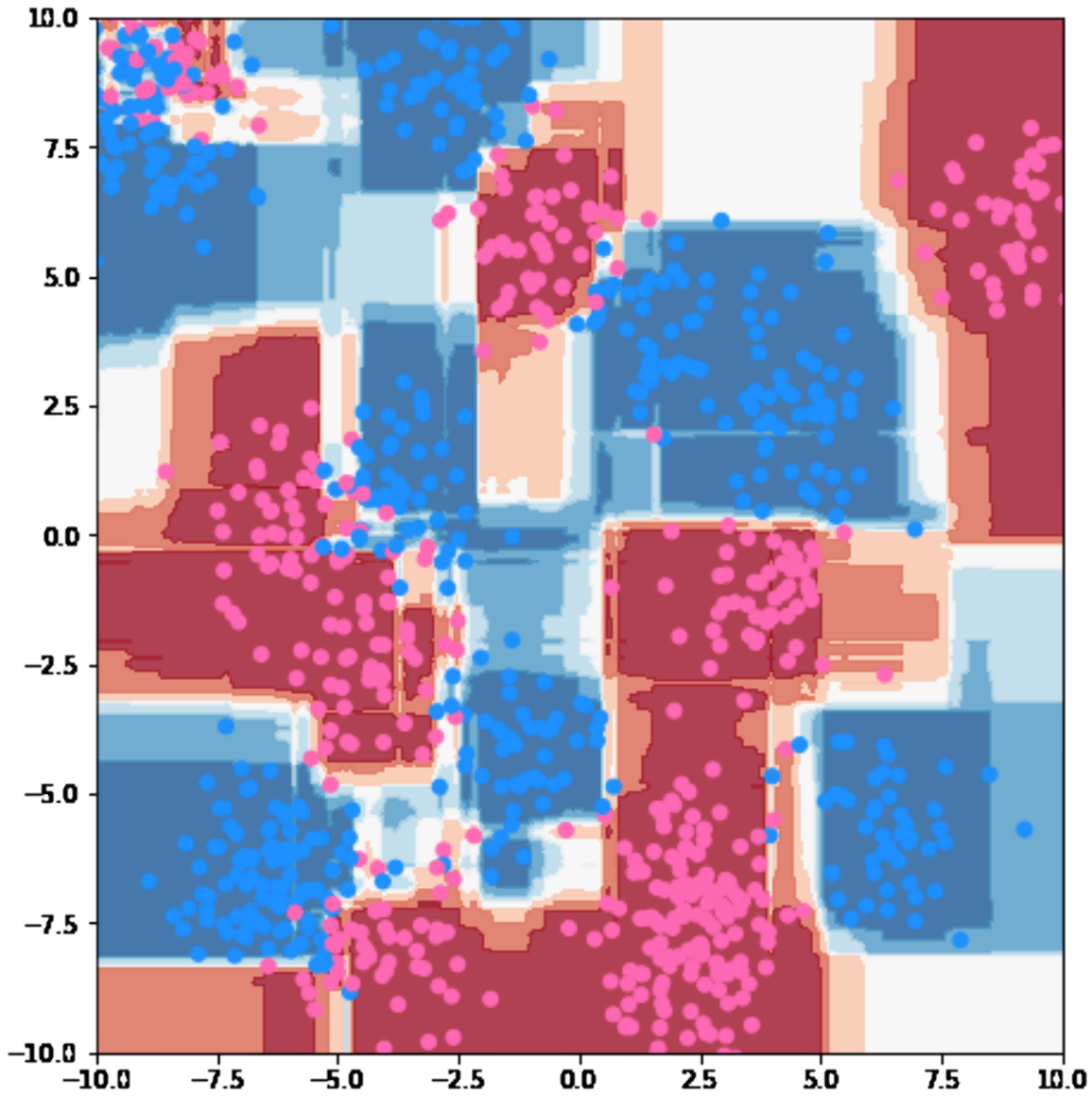
Bagging 10 Decision tree!



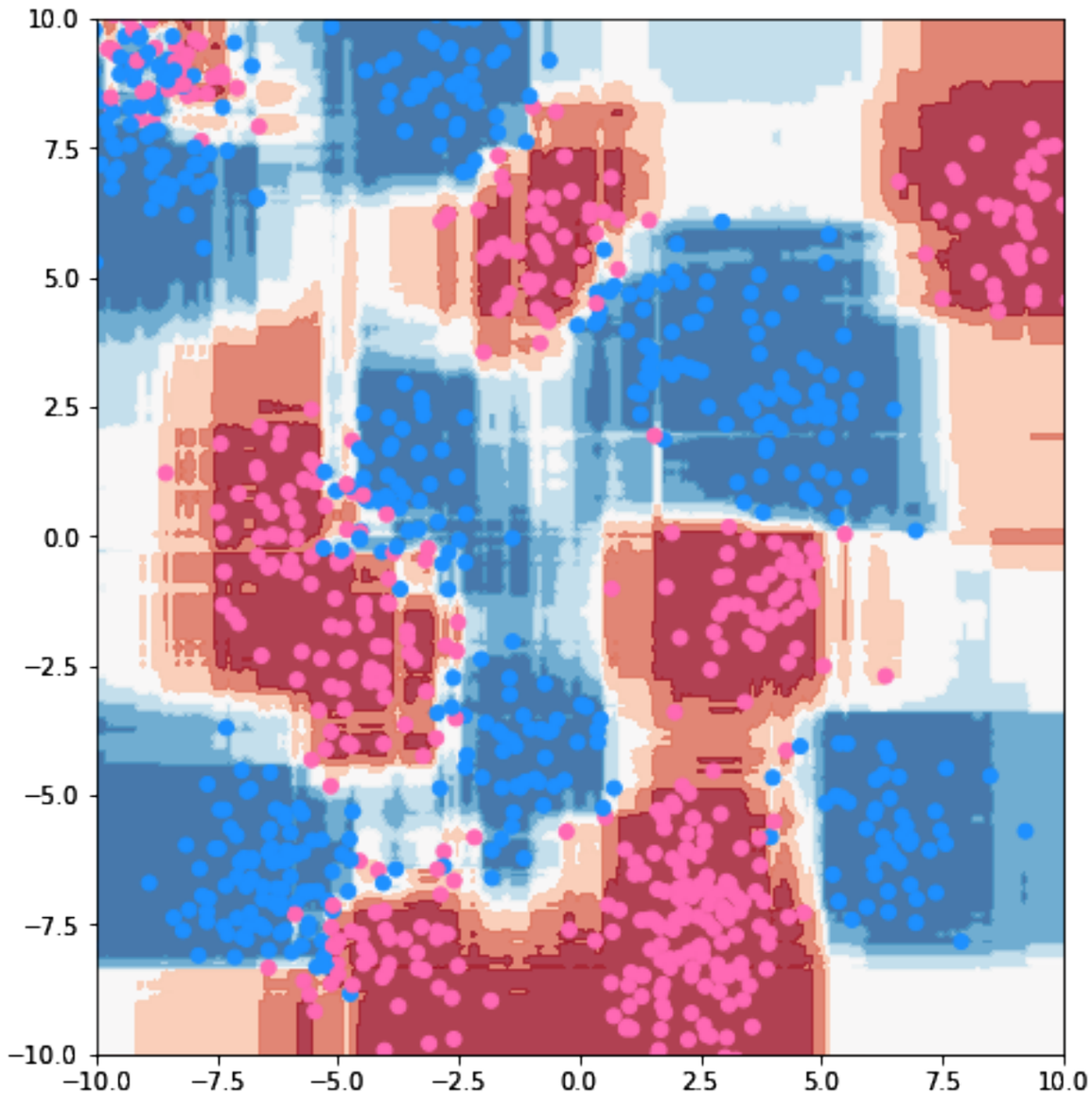
Bagging 20 Decision tree!



Bagging 20 Decision tree!



In practice: Random Forrest



Further randomization: at each candidate split in the learning process, a random subset of the features. For a classification problem with p features, $\sqrt{p}$ **(rounded down)** features are used in each split

Boosting

Can we make many dumb learners smart?

Yes (but you need a smart leader!)

Can we make many dumb learners smart?

Kearns and Valiant ’88:
- Does weak learnability imply strong learnability?
In other words, can we boost from weak to strong?

Schapire ’89
Freund and Schapire ’95
- Yes, with adaBoost

How to combine many weak classifiers?

Ingredients:

(i) T classifiers $h_t(.)$, each of them slightly better than random
(ii) A training set with N labeled examples

Adaptive linear combination of classifiers

$$H(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x})$$

ADABOOST

Exponential loss:

$$\mathcal{R} = \sum_i e^{-Y_i H(\vec{x}_i)}$$

Goal: start from t=0, add new classifiers and

(i) Adapt the weight alpha at each steps
(ii) re-weight the instances in the training set : more weight to ill-classified instances
(each new classifier concentrate on badly classified examples)

How does one set the weight and α at each time steps?

Assume we have done the job until time τ !

$$\mathcal{R} = \sum_i e^{-Y_i H(\vec{x}_i)} \qquad H(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x})$$

$$\mathcal{R} = \sum_i e^{-Y_i \overbrace{\sum_{t=1}^{\tau-1} \alpha_t h_t(\vec{x}_i)}^{\lambda_i^\tau} - Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

$$\mathcal{R} = \sum_i \underbrace{e^{-Y_i \lambda_i^\tau}}_{\omega_i^\tau} e^{-Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

$$\mathcal{R} = \sum_i \omega_i^\tau e^{-Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

ω_i^τ **= weights for each instances at time τ**

How does one set the weight and α at each time steps?

Assume we have done the job until time τ !

$$\mathcal{R} = \sum_i \omega_i^\tau e^{-Y_i \alpha_\tau h_\tau(\vec{x}_i)}$$

$$\mathcal{R} = e^{-\alpha_\tau} \sum_{i \in \mathrm{OK}} \omega_i^\tau + e^{\alpha_\tau} \sum_{i \in \mathrm{NOT\ OK}} \omega_i^\tau = e^{-\alpha_\tau} \sum_i \mathbf{1}(Y_i = h_i)\omega_i^\tau + e^{\alpha_\tau} \sum_i \omega_i^\tau \mathbf{1}(Y_i \neq h_i)$$

$$\mathcal{R} = e^{-\alpha_\tau} \sum_i (1 - \underbrace{\mathbf{1}(Y_i \neq h_i)}_{\epsilon_t})\omega_i^\tau + e^{\alpha_\tau} \sum_i \omega_i^\tau \mathbf{1}(Y_i \neq h_i)$$

ϵ_t **is what the classifier h_t(.) is trying to minimise**

Let us choose α in order to minize the global risk

$$\mathcal{R} = e^{-\alpha_\tau} - e^{-\alpha_\tau}\epsilon_t + e^{\alpha_\tau}\epsilon_t = e^{-\alpha_\tau}(1 - \epsilon_t) + e^{\alpha_\tau}\epsilon_t$$

$$\partial_{\alpha_t}\mathcal{R} = 0 \rightarrow \alpha_t = \frac{1}{2}\log\frac{1 - \epsilon_t}{\epsilon_t}$$

ADABOOST TRAINING

$$\forall\ i : \omega_i^{t=1} = \frac{1}{n}$$

$$\text{for t} = 1, \ldots, \text{T}_{\max} \qquad \textbf{Do}$$

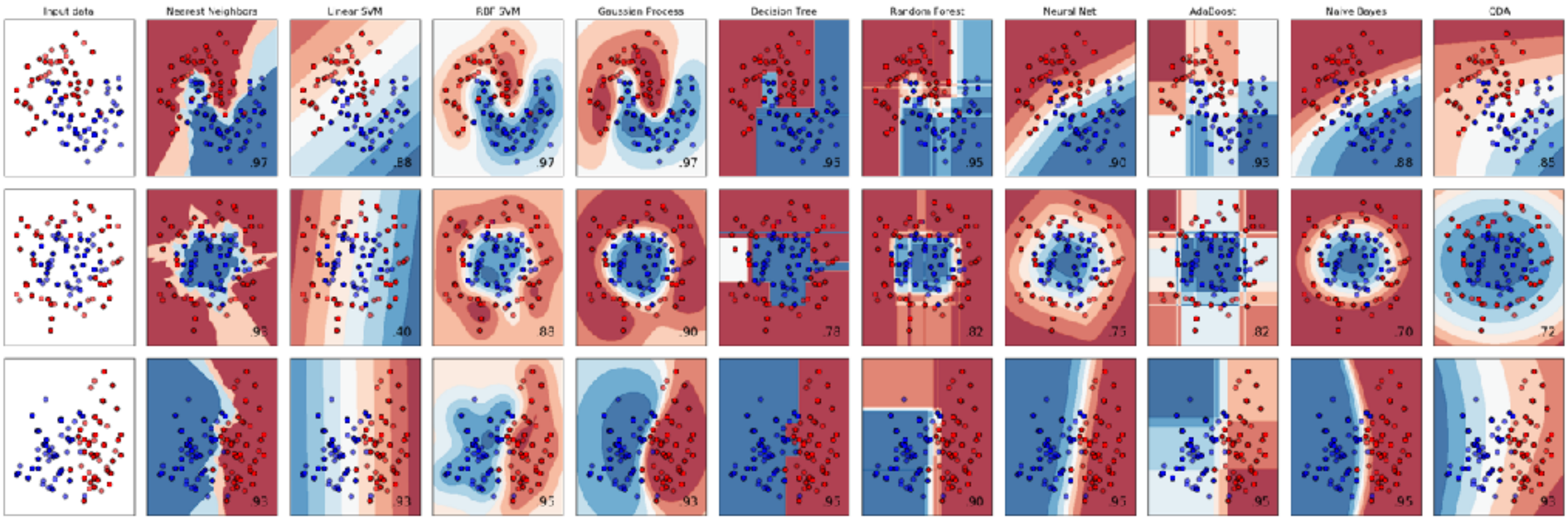
Run a classifier for : $\epsilon_t = \left[\sum_i \omega_i^{\tau} \mathbf{1}(Y_i \neq h_i))\right]$

Set: $\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$

Agregate classifier: $H(.) = H^{t-1}(.) + \alpha_t h_t(.)$

Update weights: $\omega_i^{t+1} = \frac{\omega_i^t e^{-Y_i \alpha_t h_t(\vec{x}_i)}}{\sum_i \omega_i^t e^{-Y_i \alpha_t h_t(\vec{x}_i)}}$

A tour on standard machine learning classifiers



From sk-learn

“Part I”

How I Learned to Stop Worrying and Love Kernel methods

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1,\ldots,n \qquad \mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

Ex: linear models

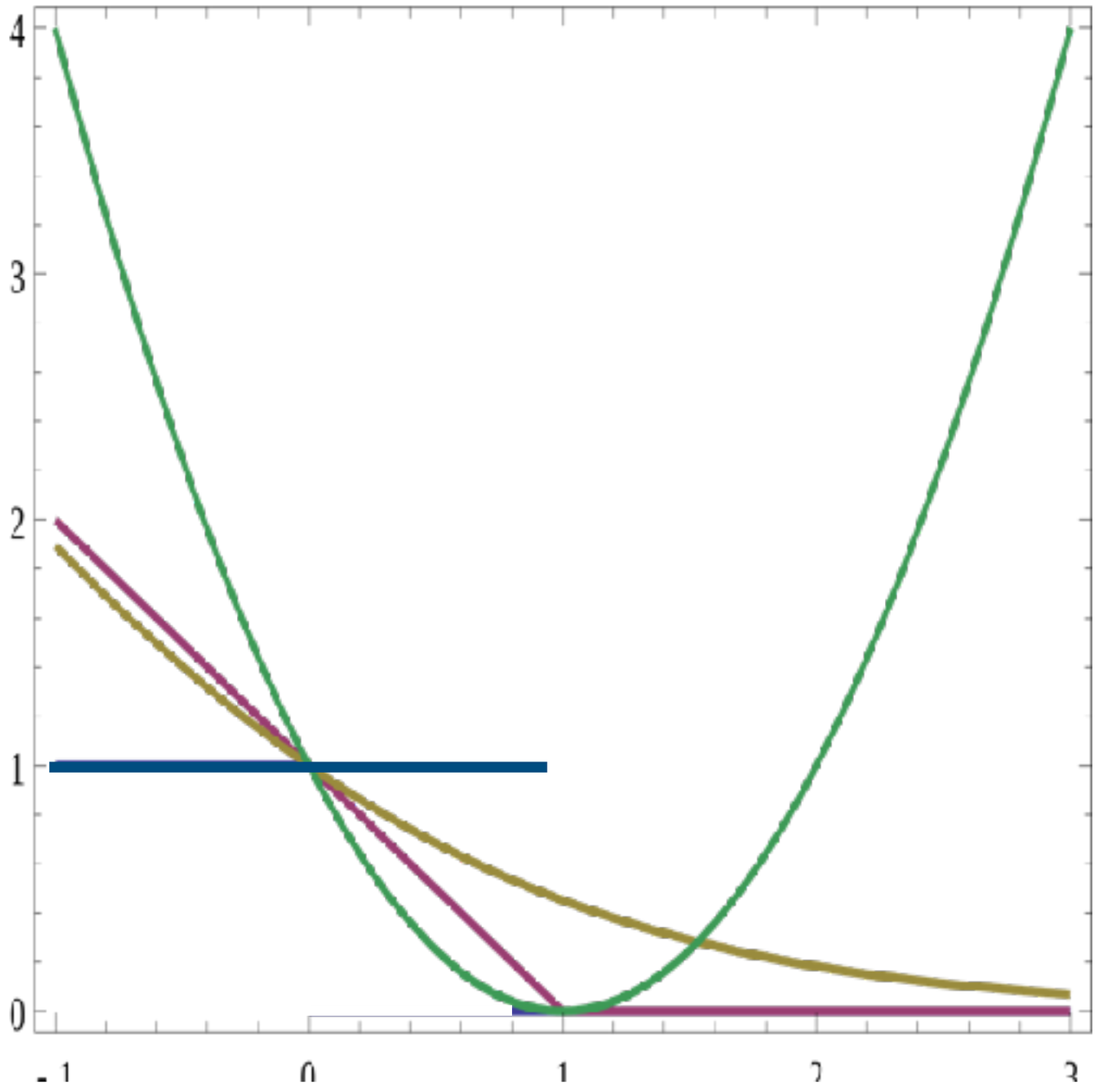
Model: $f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$

Loss : $\mathscr{L}(y, h)$ **Square Loss** $L(f(\vec{x}), y) = (1 - yf(\vec{x}))^2$

Logistic loss/Cross-entropy $L(f(\vec{x}), y) = \frac{1}{\ln 2}\ln(1 + e^{-yf(\vec{x})})$

Choices of loss

$$f(\vec{x}) = \theta \cdot \vec{x} + \alpha$$



Square Loss

$$L(f(\vec{x}), y) = (1 - yf(\vec{x}))^2$$

Hard margin

$$L(f(\vec{x}), y) = \mathbf{1}(yf(\vec{x}) > 1)$$

Hinge loss

$$L(f(\vec{x}), y) = \max(0, 1 - yf(\vec{x}))$$

Logistic loss/Cross-entropy

$$L(f(\vec{x}), y) = \frac{1}{\ln 2} \ln(1 + e^{-yf(\vec{x})})$$

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n$$

$$\mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

Ex: neural networks models

Model: $$f_\theta(\mathbf{X}) = \eta^{(0)} \left(\mathbf{W}^{(0)} \eta^{(1)} \left(\mathbf{W}^{(1)} \ldots \eta^{(L)} \left(\mathbf{W}^{(L)} \cdot \mathbf{X} \right) \right) \right)$$

$$\theta = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}\}$$

Loss : $\mathscr{L}(y, h)$

Just as before…

Linear models

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

$$f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$$

Gradient descent

$$\theta \in \mathbb{R}^d$$

Gradient flow

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Representer theorem

For any loss function such that

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}(y_i, \theta \cdot \mathbf{x}_i)$$

$$\hat{\theta} = \text{argmin}\mathcal{R}(\theta)$$

We can always write the minimiser as:

$$\hat{\theta} = \sum_{i=1}^{n}\beta_i \mathbf{X}_i$$

Proof

$$\hat{\theta} \in \mathbb{R}^d \qquad \mathbf{X}_i \in \mathbb{R}^d \forall i \qquad \mathbb{R}^d = \text{span}(\{X\}) + \text{null}(\{X\})$$

So we can write:

$$\hat{\theta} = \sum_{i=1}^{n}\beta_i \mathbf{X}_i + \overrightarrow{\mathscr{N}}$$

But:

$$\hat{\theta} \cdot \mathbf{X}_j = \left(\sum_{i=1}^{n}\beta_i \mathbf{X})_i\right) \cdot \mathbf{X}_j + \overrightarrow{\mathscr{N}} \cdot \mathbf{X}_j = \left(\sum_{i=1}^{n}\beta_i \mathbf{X})_i\right) \cdot \mathbf{X}_j$$

So we might as well write:

$$\hat{\theta} = \sum_{i=1}^{n}\beta_i \mathbf{X}_i \quad \blacksquare$$

Linear models

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

$$f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$$

Gradient descent

$$\theta \in \mathbb{R}^d$$

Gradient flow

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Linear models, dual…

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n}\beta_j\mathbf{X}_j \cdot \mathbf{X}$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta\nabla_\beta\mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta\mathscr{R}$$

Template matching

$$f_\beta(\mathbf{X}^{\text{new}}) = \sum_{j=1}^{n} \beta_j \mathbf{X}_j \cdot \mathbf{X}^{\text{new}}$$

Prediction for new sample are made just by comparing the scalar product of the new sample with those of the entire dataset!

If this is a good idea, why just using a scalar product as the “similarity” ?

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X})$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

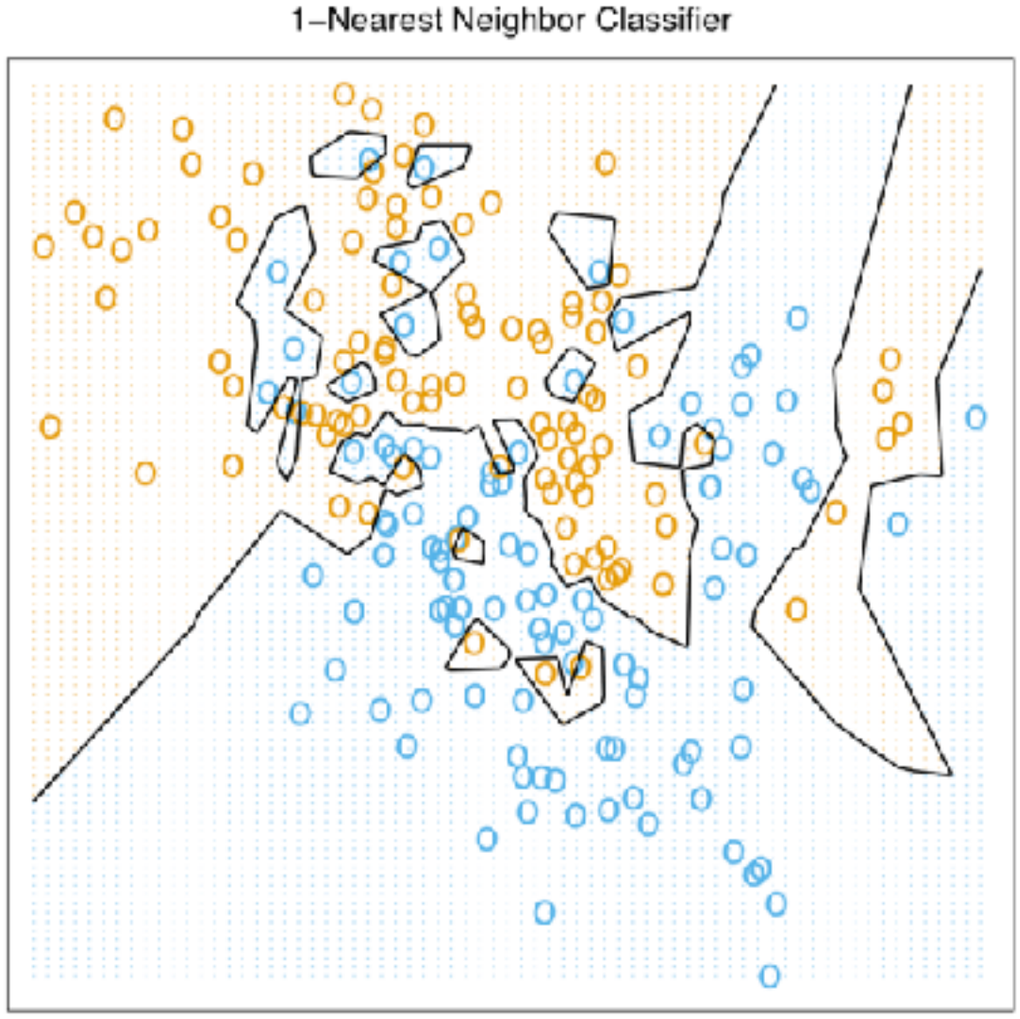
Depends only on the Gram matrix

$$K_{ij} = K(\mathbf{X}_i, \mathbf{X}_j)$$

Ex: Gaussian Kernel

$$K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\beta \|\mathbf{X}_i - \mathbf{X}_j\|_2^2}$$

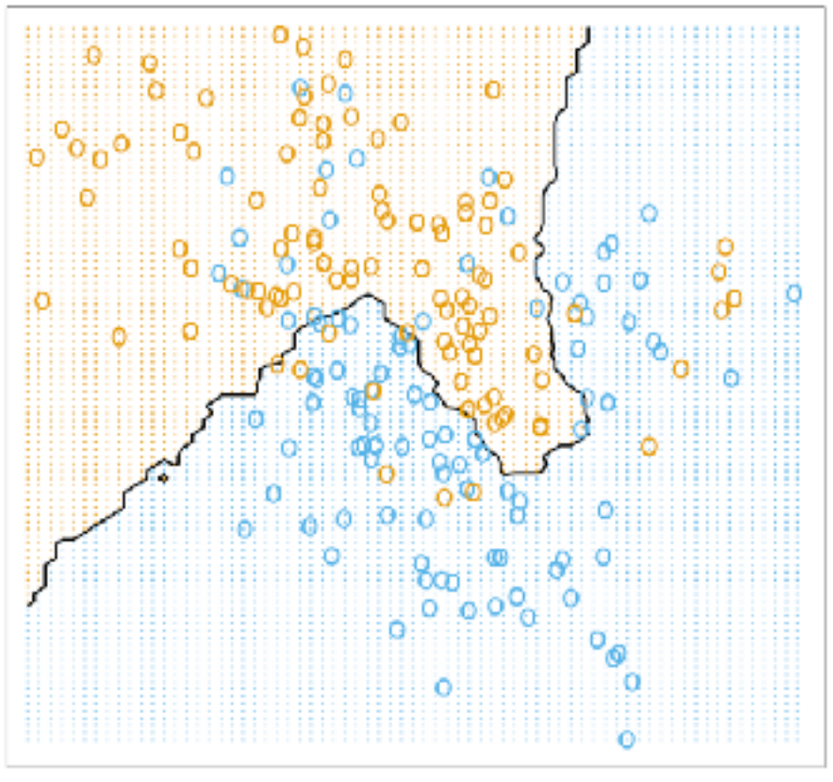
As $\beta \to \infty$ converges to 1NN methods

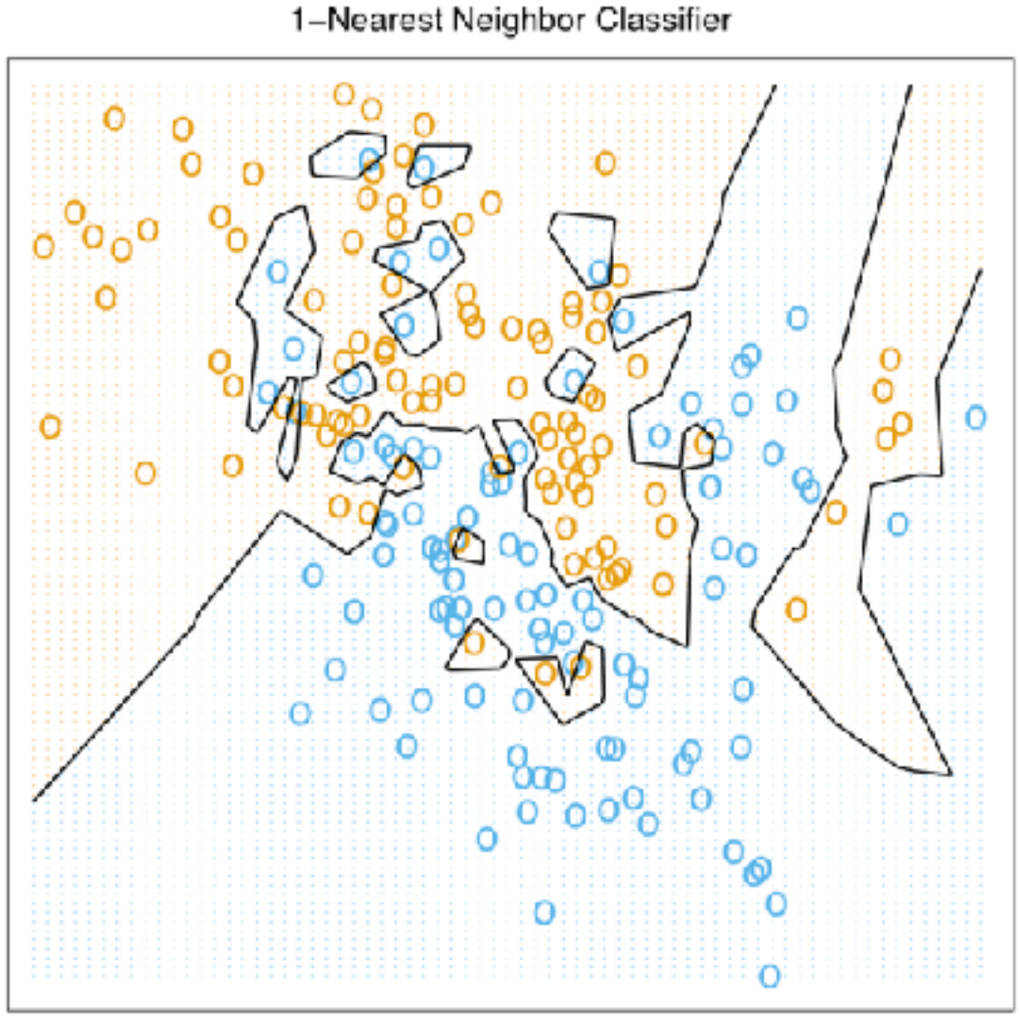


Ex: Gaussian Kernel

$$K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\beta \|\mathbf{X}_i - \mathbf{X}_j\|_2^2}$$

For lower values, interpolate between neighbours





Funes the memorious

Knn just memorise everything…
… no « understanding/learning »
whatsoever!

‘Not only was it difficult for him to understand that the generic term ‘dog’ could embrace so many disparate individuals of diverse size and shapes, it bothered him that the dog seen in profile at 3:14 would be called the same dog at 3:15 seen from the front.’

Without effort, he had learned English, French, Portuguese, Latin. I suspect, nevertheless, that he was not very capable of thought. To think is to forget a difference, to generalize, to abstract. In the overly replete world of Funes there were nothing but details, almost contiguous details.

Jorge Louis Borges, « *Funes the Memorious* » *1942*

More clever kernels?

$$K(\mathbf{X}_i, \mathbf{X}_j) = ?$$

Ideally, would like to implement a meaningful notion of similarity, and invariance for relevant symmetries

---

Convolutional Kernel Networks

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid
Inria*
firstname.lastname@inria.fr

Abstract

An important goal in visual recognition is to devise image representations that are invariant to particular transformations. In this paper, we address this goal with a new type of convolutional neural network (CNN) whose invariance is encoded by a reproducing kernel. Unlike traditional approaches where neural networks are learned either to represent data or for solving a classification task, our network learns to approximate the kernel feature map on training data.

Such an approach enjoys several benefits over classical ones. First, by teaching CNNs to be invariant, we obtain simple network architectures that achieve a similar accuracy to more complex ones, while being easy to train and robust to overfitting. Second, we bridge a gap between the neural network literature and kernels, which are natural tools to model invariance. We evaluate our methodology on visual recognition tasks where CNNs have proven to perform well, *e.g.*, digit recognition with the MNIST dataset, and the more challenging CIFAR-10 and STL-10 datasets, where our accuracy is competitive with the state of the art.

Method	[12]	[27]	[18]	[13]	[4]	[17]	[32]	CKN-GM	CKN-PM	CKN-CO
CIFAR-10	82.0	82.2	**88.32**	79.6	NA	83.96	84.87	74.84	78.30	82.18
STL-10	60.1	58.7	NA	51.5	**64.5**	62.3	NA	60.04	60.25	62.32

Table 2: Classification accuracy in % on CIFAR-10 and STL-10 without data augmentation.

Mercer's Theorem & the feature map

If K(s,t) is symmetric and positive-definite, then there is an orthonormal basis $\{e_i\}_i$ of $L^2[a, b]$ consisting of « **eigenfunctions** » such that the corresponding sequence of eigenvalues $\{\lambda_j\}_j$ is nonnegative.

$$K(s,t) = \sum_{j=1}^{\infty} \lambda_j \, e_j(s) \, e_j(t)$$

All symmetric positive define Kernels can be seen as a projection in an infinite dimensional space

Orignal space (data space) dimension d

Feature map

$$\Phi = g(X)$$

Features space (After projection) dimension D (*possibly infinite*)

$$X_i$$

$$K(\mathbf{X}_i, \mathbf{X}_j) = \mathbf{\Phi}_i \cdot \mathbf{\Phi}_j$$

$$\Phi_i = \begin{pmatrix} \sqrt{\lambda_1} e_1(X_i) \\ \sqrt{\lambda_2} e_2(X_i) \\ \sqrt{\lambda_3} e_4(X_i) \\ \cdots \\ \sqrt{\lambda_D} e_D(X_i) \end{pmatrix}$$

Example: Gaussian Kernel

$$K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\frac{1}{2\sigma^2}\|\mathbf{X}_i - \mathbf{X}_j\|_2^2}$$

$$e^{\frac{-1}{2\sigma^2}(x_i - x_j)^2} = e^{\frac{-x_i^2 - x_j^2}{2\sigma^2}} \left(1 + \frac{2x_i x_j}{1!} + \frac{(2x_i x_j)^2}{2!} + \ldots\right) \tag{1.25}$$

$$= e^{\frac{-x_i^2 - x_j^2}{2\sigma^2}} \left(1 \cdot 1 + \sqrt{\frac{2}{1!}}x_i \cdot \sqrt{\frac{2}{1!}}x_j + \sqrt{\frac{(2)^2}{2!}}(x_i)^2 \cdot \sqrt{\frac{(2)^2}{2!}}(x_j)^2 + \ldots\right)$$

$$= \phi(x_i)^T \phi(x_j)$$

$$\text{where, } \phi(x) = e^{\frac{-x^2}{2\sigma^2}} \left(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \ldots\right)$$

Infinite dimensional feature (polynomial) map!

Kernel methods (i)

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X})$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

Depends only on the Gram matrix

$$K_{ij} = K(\mathbf{X}_i, \mathbf{X}_j)$$

Kernel methods (ii)

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{\Phi}_i))$$

$$f_\beta(\mathbf{\Phi}) = \sum_{j=1}^{n} \beta_j \mathbf{\Phi}_j, \mathbf{\Phi}_i$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

Depends only on the Gram matrix

$$K_{ij} = \mathbf{\Phi}_i \cdot \mathbf{\Phi}_j$$

Kernel methods (iii)
Linear model (in feature space!)

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i))$$

$$f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi}$$

$$\theta \in \mathbb{R}^{D} \quad (D = \infty)$$

Gradient descent

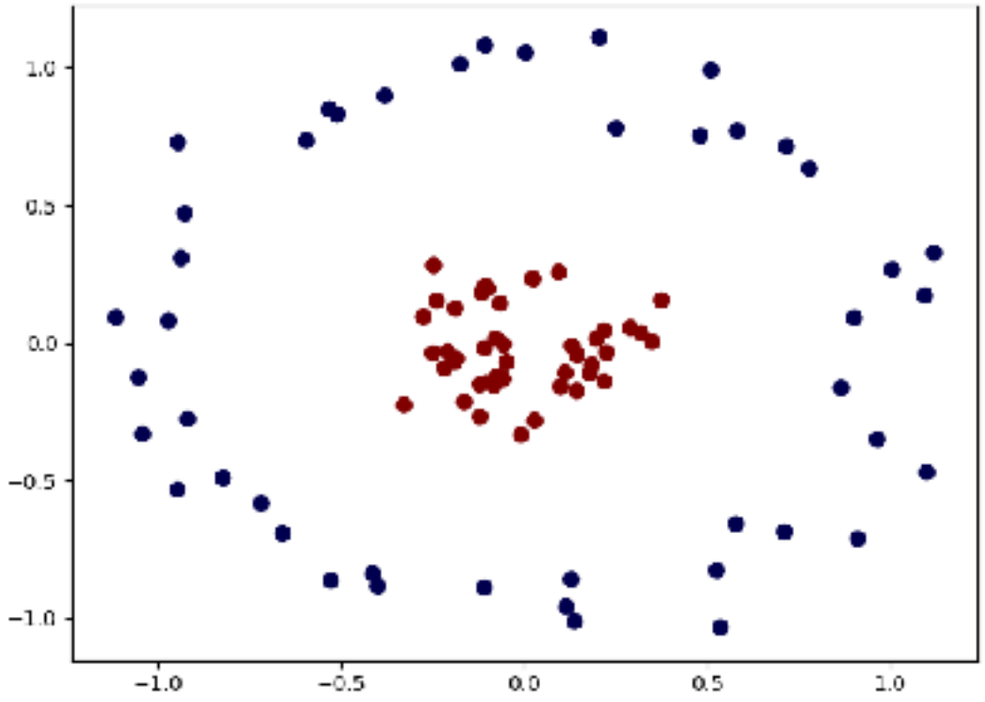
$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

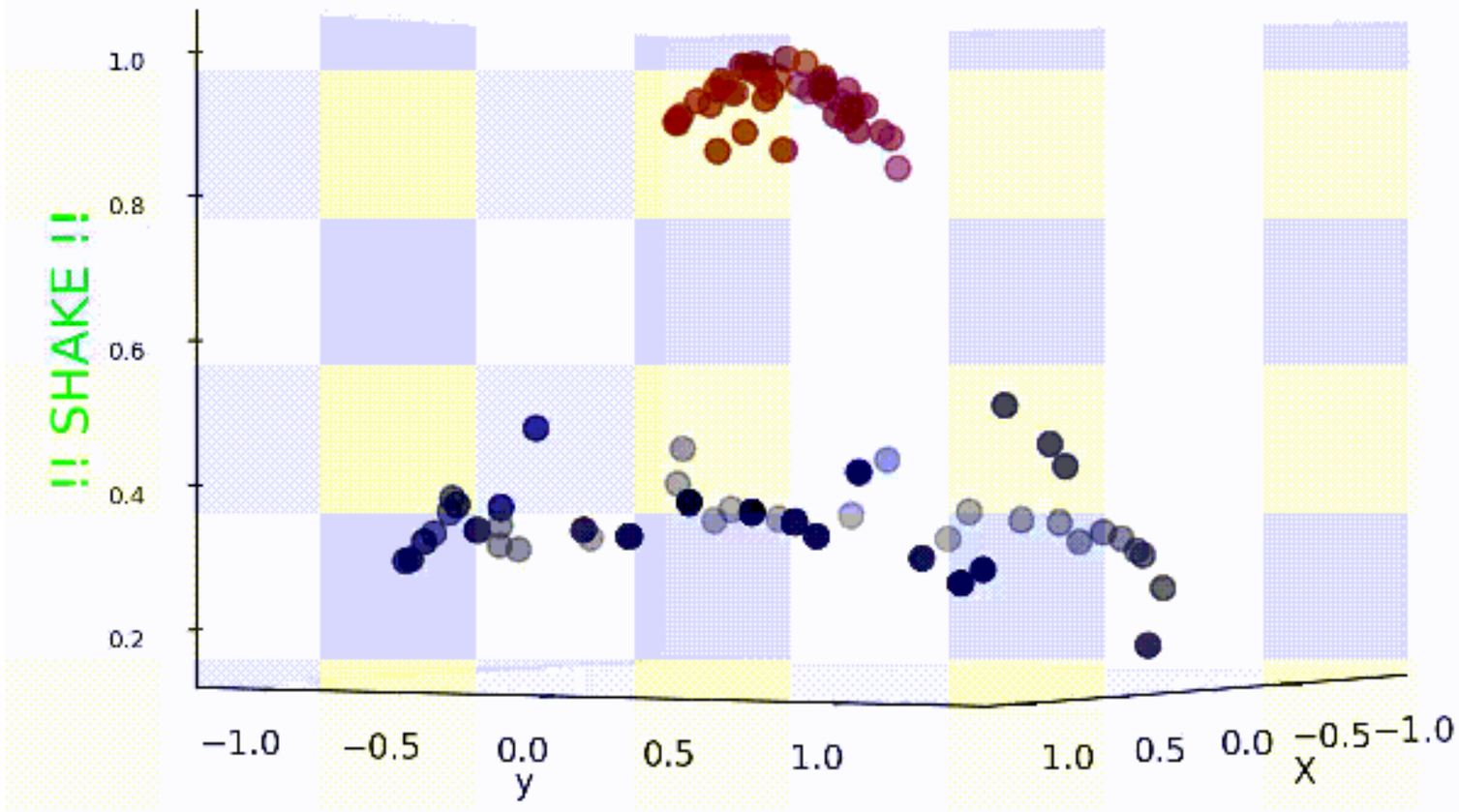
Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Mapping to large dimension

Can separate arbitrary complicated functions!





The Kernel Trick!

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X}) \quad \beta \in \mathbb{R}^n$$

Gradient descent

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

Gradient flow

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i))$$

Feature map $\Phi = g(X)$

$$f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi} \qquad \theta \in \mathbb{R}^{D=\infty}$$

Gradient descent

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

“Part II”

The problem with Kernels and the solution

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X}) \quad \beta \in \mathbb{R}^n$$

Gradient descent

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

Gradient flow

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i))$$

Feature map $\Phi = g(X)$

$$f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi} \qquad \theta \in \mathbb{R}^{D=\infty}$$

Gradient descent

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

$$K(X_i, X_j) = \Phi_i \cdot \Phi_j$$

$$\mathbf{K} = \begin{pmatrix} K(X^1, X^1) & K(X^1, X^2) & \dots & K(X^1, X^N) \\ K(X^2, X^1) & K(X^2, X^2) & \dots & K(X^2, X^N) \\ \dots & \dots & \dots & \dots \\ K(X^N, X^1) & K(X^N, X^2) & \dots & K(X^N, X^N) \end{pmatrix}$$

Say you have one million examples….

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X}) \qquad \beta \in \mathbb{R}^n$$

Gradient descent

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

Gradient flow

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i))$$

Feature map $\Phi = g(X)$

$$f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi} \qquad \theta \in \mathbb{R}^{D=\infty}$$

Gradient descent

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i)) \qquad f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi}$$

Feature map $\Phi = g(X)$

$$\theta \in \mathbb{R}^{D=\infty}$$

Gradient descent

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Idea 1: truncate the expansion of the feature map (e.g. polynomial features)

Idea 2: approximate the feature map by sampling

Random Fourrier features

(Recht-Rahimi '07)

$$\mathbf{\Phi} = g(\mathbf{x})$$

$$\mathbf{\Phi} \in \mathbb{R}^D \qquad \mathbf{x} \in \mathbb{R}^d$$

$$\mathbf{\Phi} = \frac{1}{\sqrt{D}} e^{iF\mathbf{x}}$$

F a Dxd matrix,
Coefficients i.i.d. random from P(F)

$$\Phi_i = \frac{1}{\sqrt{\mathbf{D}}} \begin{pmatrix} e^{i2\pi \vec{F}_1 \cdot X_i} \\ e^{i2\pi \vec{F}_2 \cdot X_i} \\ \cdots \\ e^{i2\pi \vec{F}_D \cdot X_i} \end{pmatrix}$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}_i \cdot \mathbf{\Phi}_j = \frac{1}{D} \sum_{k=1}^{D} e^{i \vec{F}_k (\mathbf{x_i} - \mathbf{x_j})} \xrightarrow[D \to \infty]{\lim} \mathbb{E}_{\vec{F}} \left[e^{i \vec{F} (\mathbf{x_i} - \mathbf{x_j})} \right] = \int d\vec{F} P(\vec{F}) e^{i \vec{F} (\mathbf{x_i} - \mathbf{x_j})}$$

$$K(X_i, X_j) = K(||X_i - X_j||) = \mathrm{TF}(P(\vec{F}))$$

Kernel Name	$k(\Delta)$	$p(\omega)$
Gaussian	$e^{-\frac{\|\Delta\|_2^2}{2}}$	$(2\pi)^{-\frac{D}{2}} e^{-\frac{\|\omega\|_2^2}{2}}$
Laplacian	$e^{-\|\Delta\|_1}$	$\prod_d \frac{1}{\pi(1+\omega_d^2)}$
Cauchy	$\prod_d \frac{2}{1+\Delta_d^2}$	$e^{-\|\Delta\|_1}$

Random erf features

(Williams '07)

$$\Phi_i = \frac{1}{\sqrt{\mathbf{D}}} \begin{pmatrix} \mathrm{erf}(\tilde{\mathrm{F}}_1 \cdot \mathrm{X_i}) \\ \mathrm{erf}(\tilde{\mathrm{F}}_2 \cdot \mathrm{X_i}) \\ \cdots \\ \mathrm{erf}(\tilde{\mathrm{F}}_\mathrm{D} \cdot \mathrm{X_i}) \end{pmatrix}$$

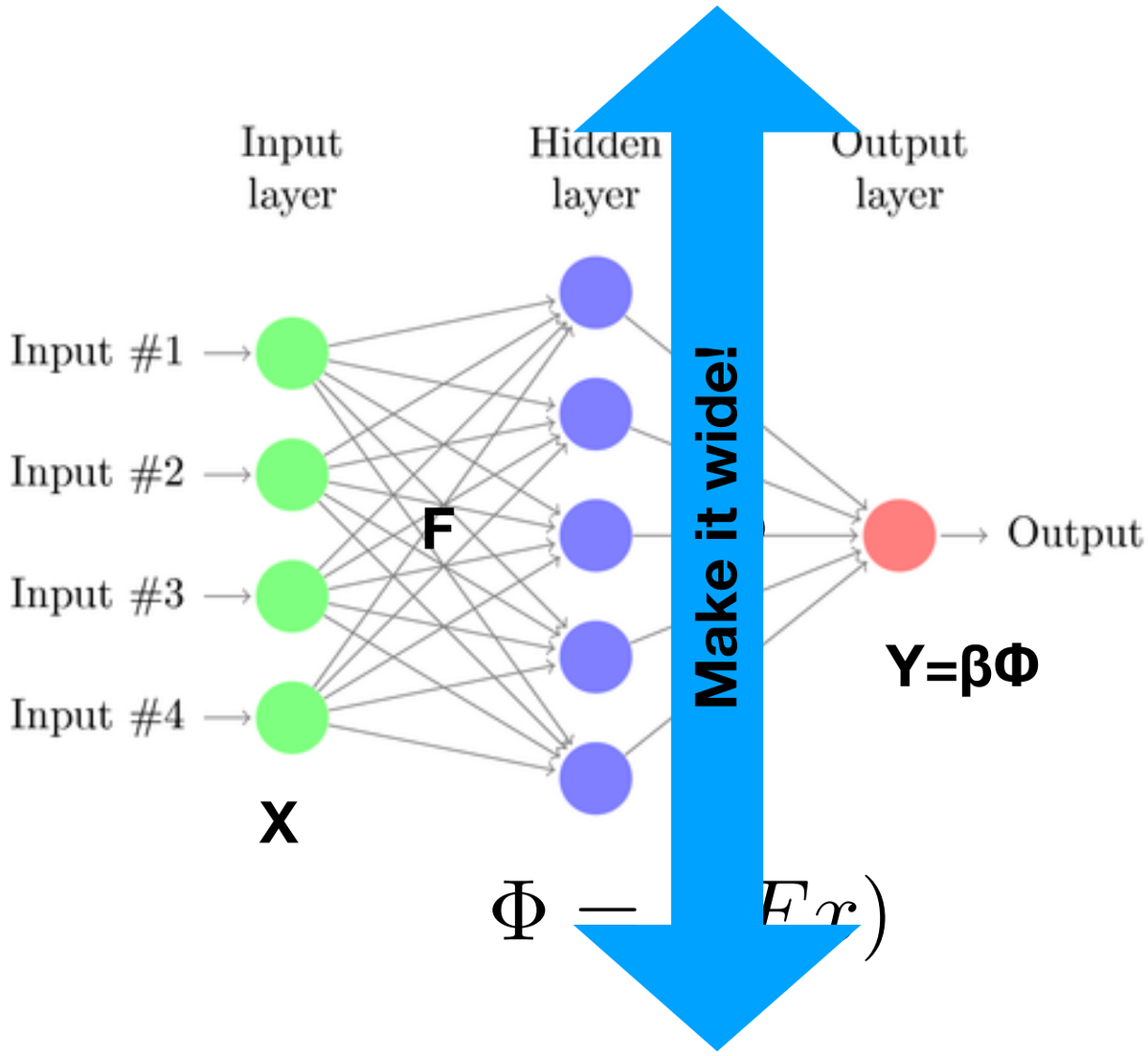
$$\lim_{D \to \infty} \Phi_i \cdot \Phi_j \ \textbf{?}$$

After a bit of work (Williams, 98)

$$K(X_i, X_j) = \frac{2}{\pi} \arcsin \left(\frac{2X_i \cdot X_j}{\sqrt{1 + 2X_i \cdot X_i}\sqrt{1 + 2X_j \cdot X_j}} \right)$$

Here the kernel depends on the angle….

Equivalent representation:
A WIIIIIIIDE random 2-layer neural network



Fix the « weights » in the first layer randomly…
… and to learn only the weights in the second layer

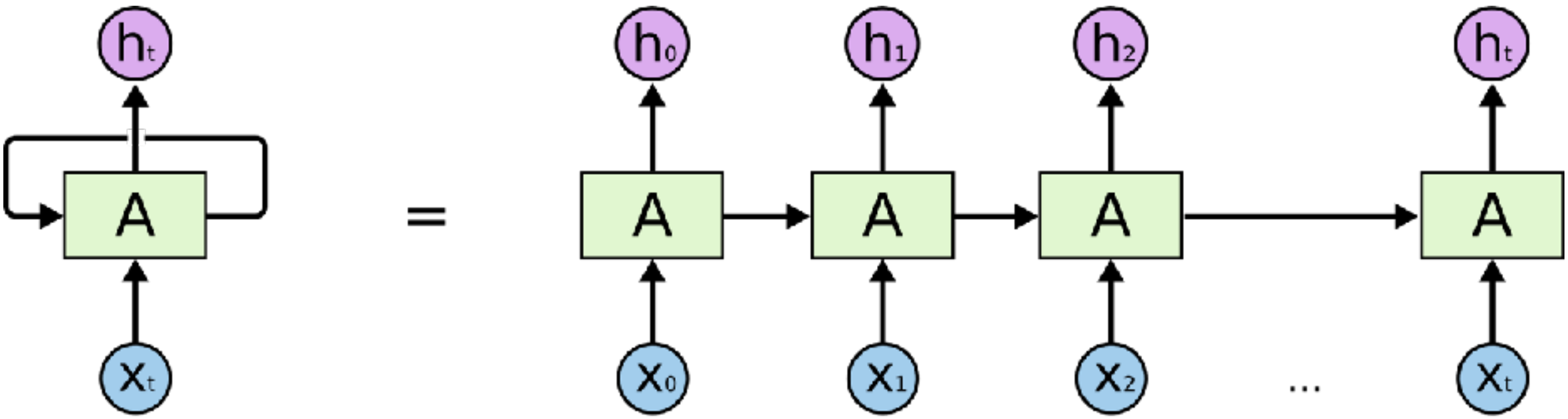
Infinitly wide neural net with random weights converges to kernel methods
(Neal ’96, Williams 98, Recht-Rahimi ‘07)

Achitectures

Recurrent Neural Networks

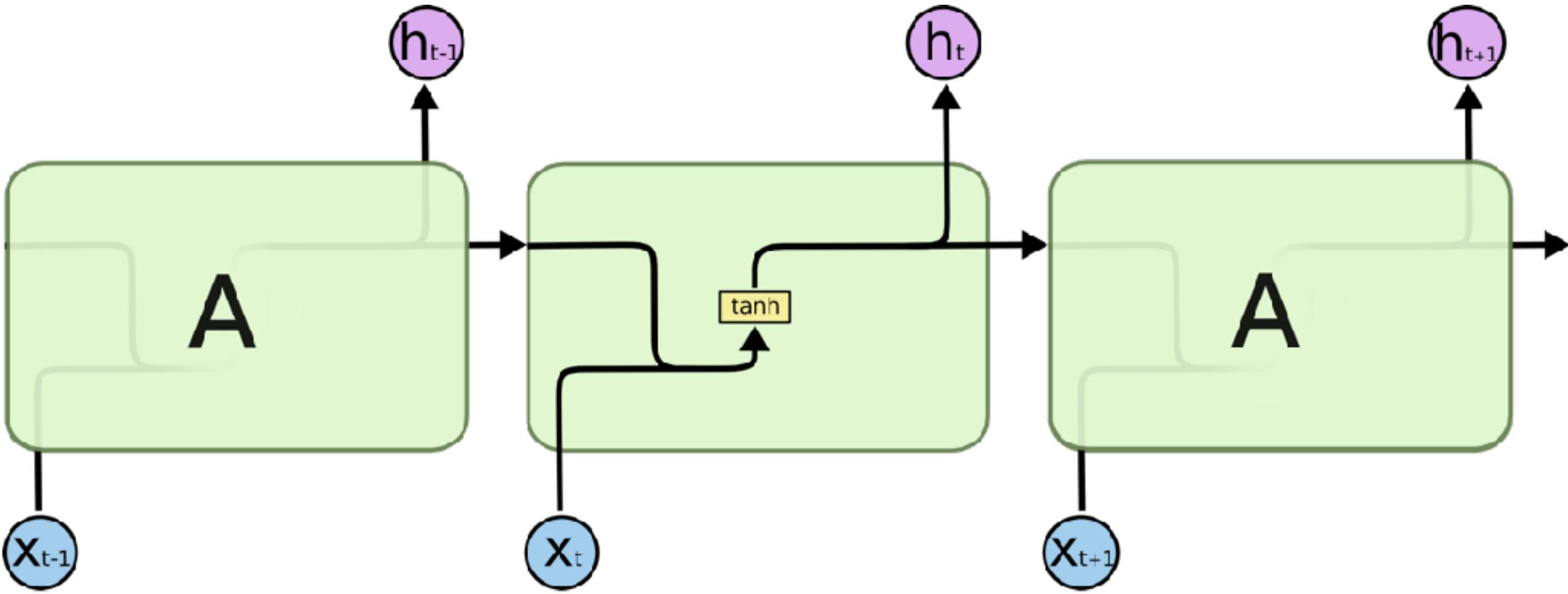
Times series

Recurent Neural Networks



Slide credit: Christopher Olah

Recurent Neural Networks



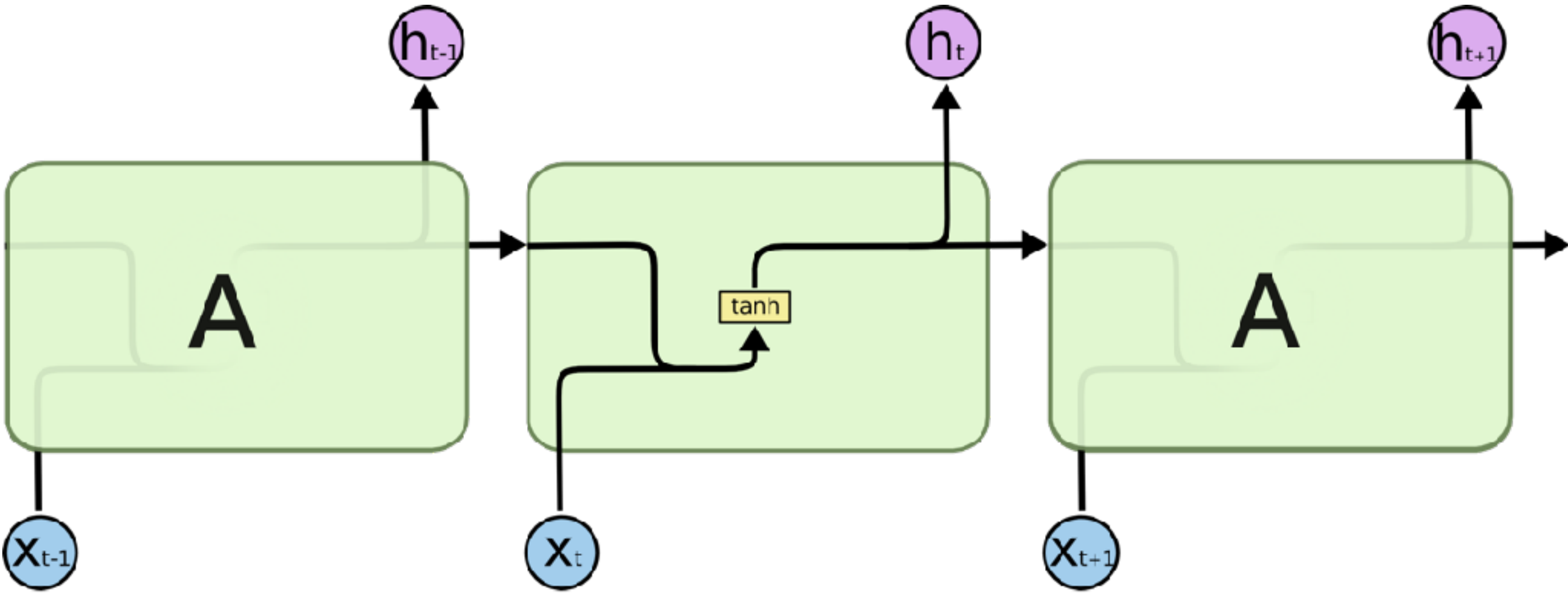
$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

Slide credit: Christopher Olah

Echo State Networks

Use random matrices, fit on h!

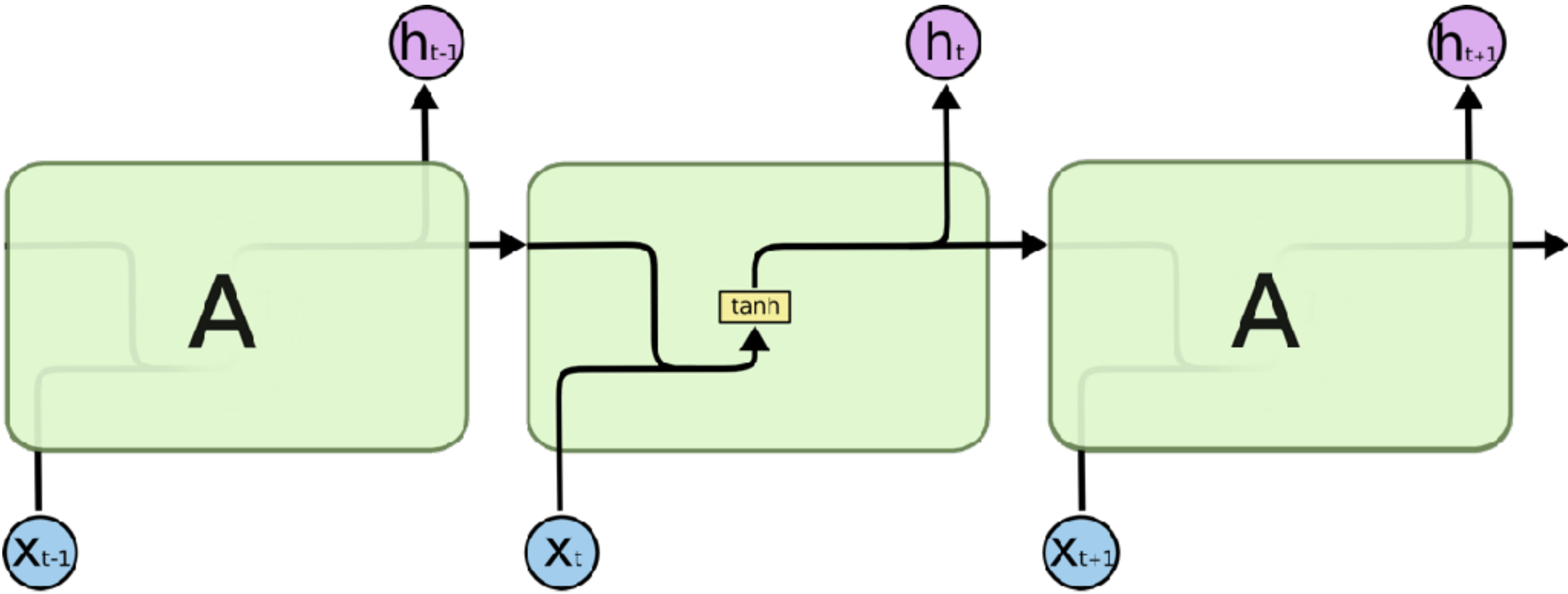
« Reccurent » equivalent to the random projection of lecture II



$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

Slide credit: Christopher Olah

Recurent Neural Networks

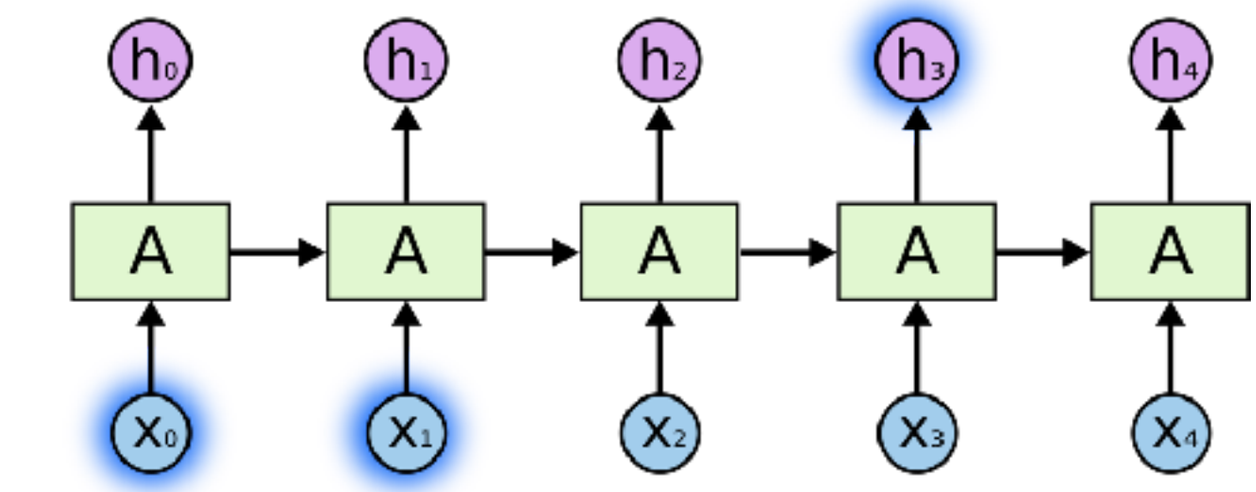


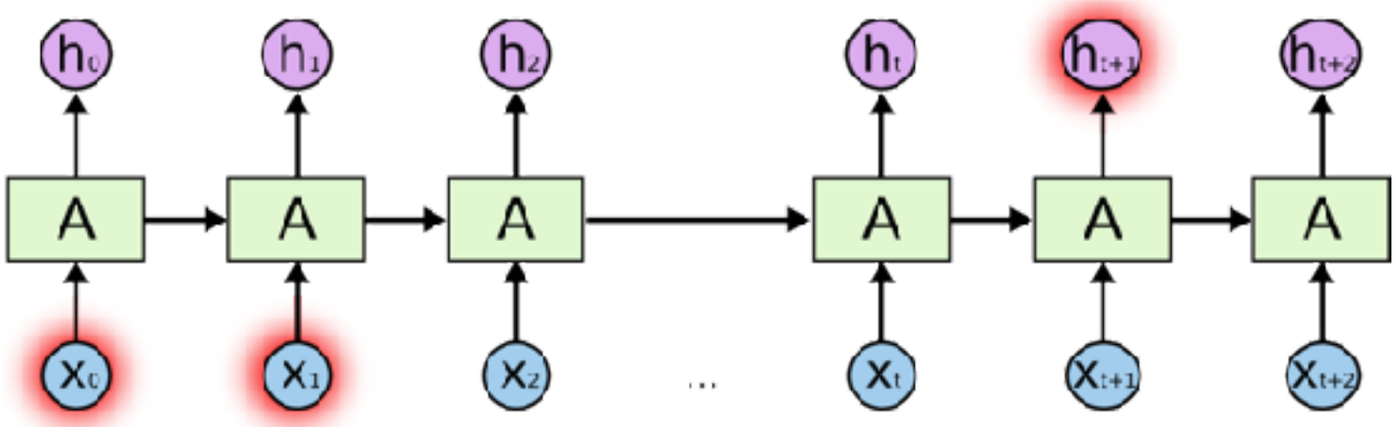
$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

Slide credit: Christopher Olah

Recurent Neural Networks

The Problem of Long-Term Dependencies

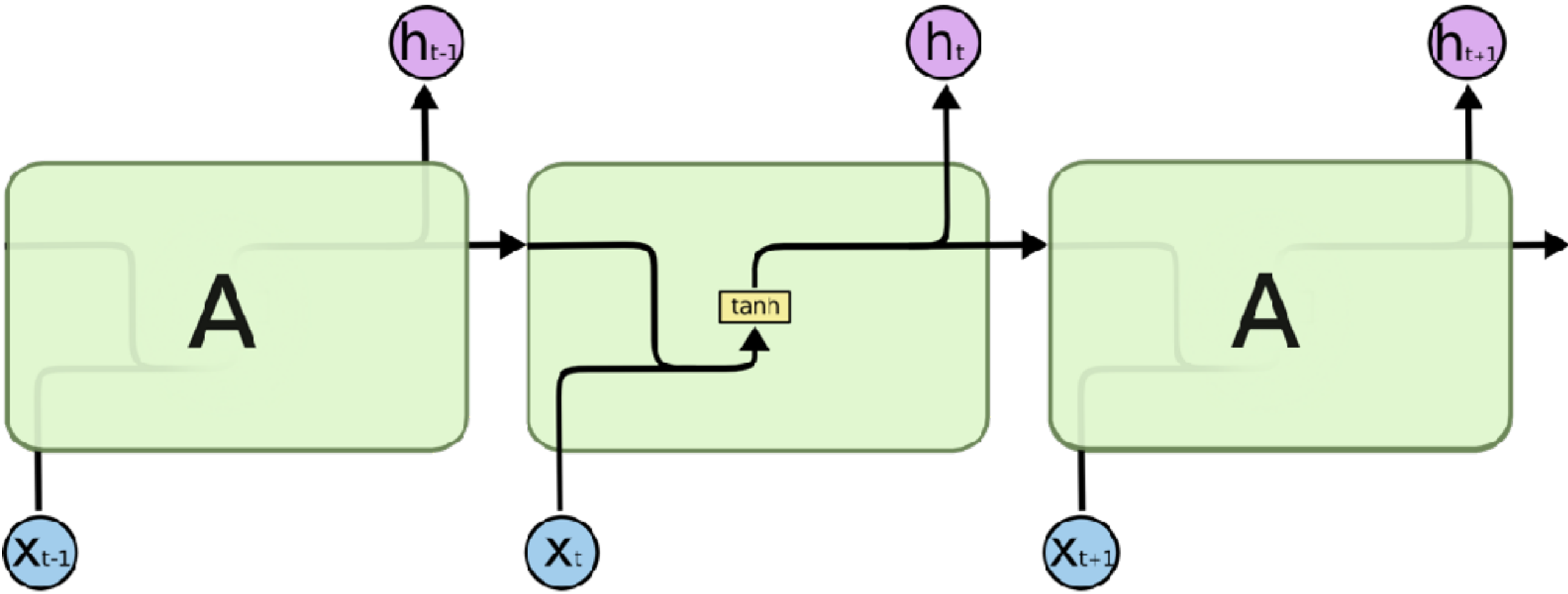




Slide credit: Christopher Olah

LSTMs!

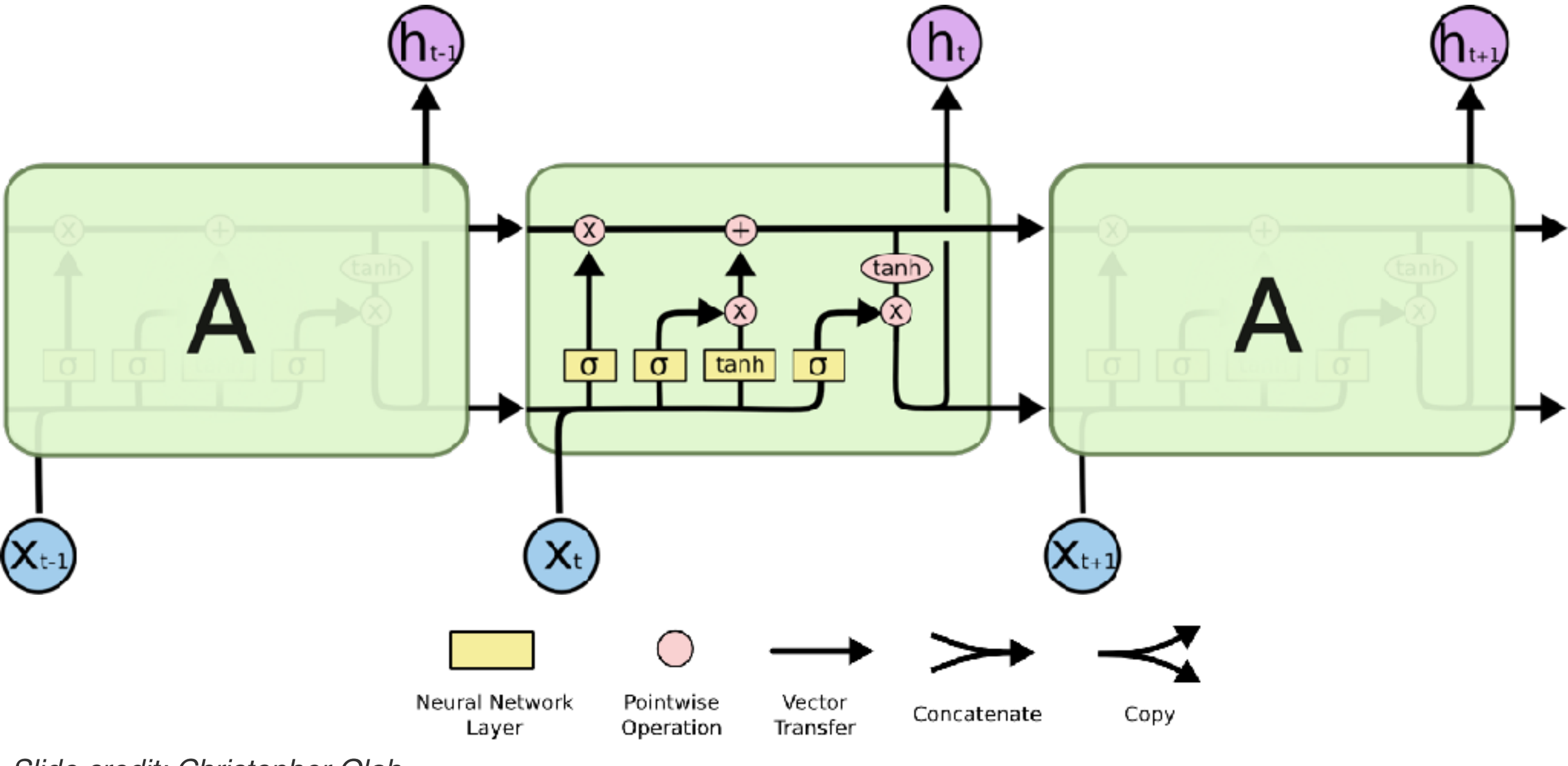
Long Short Term Memory networks



Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

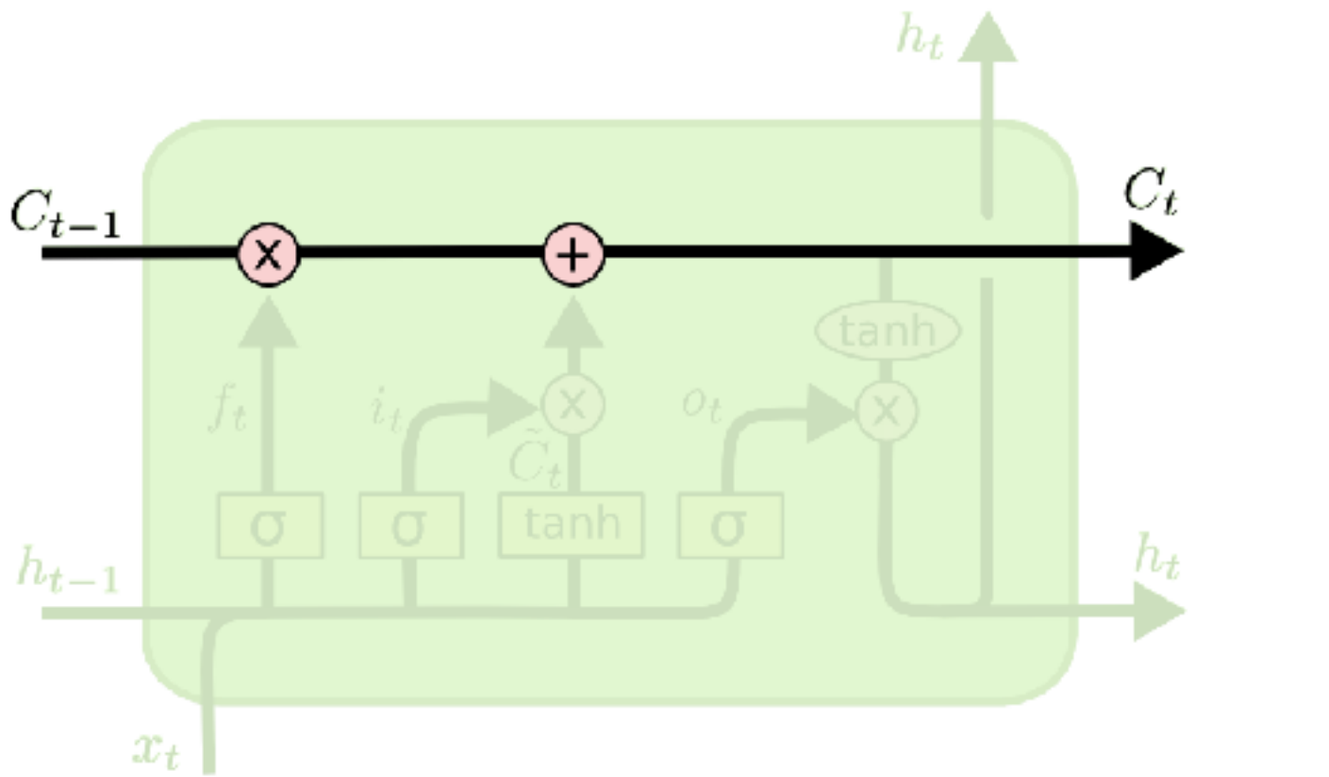


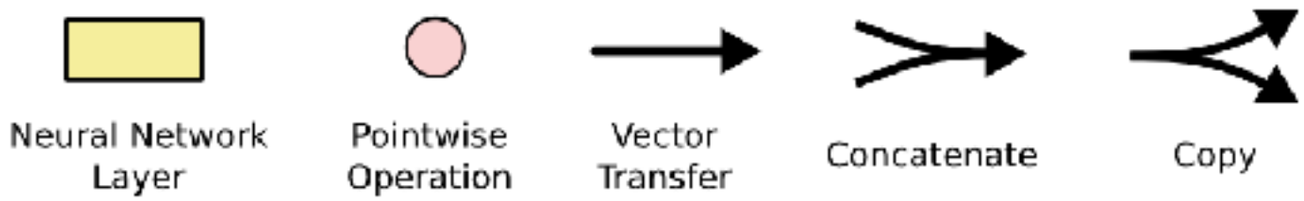
Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

The CELL GATE! The Core Idea Behind LSTMs



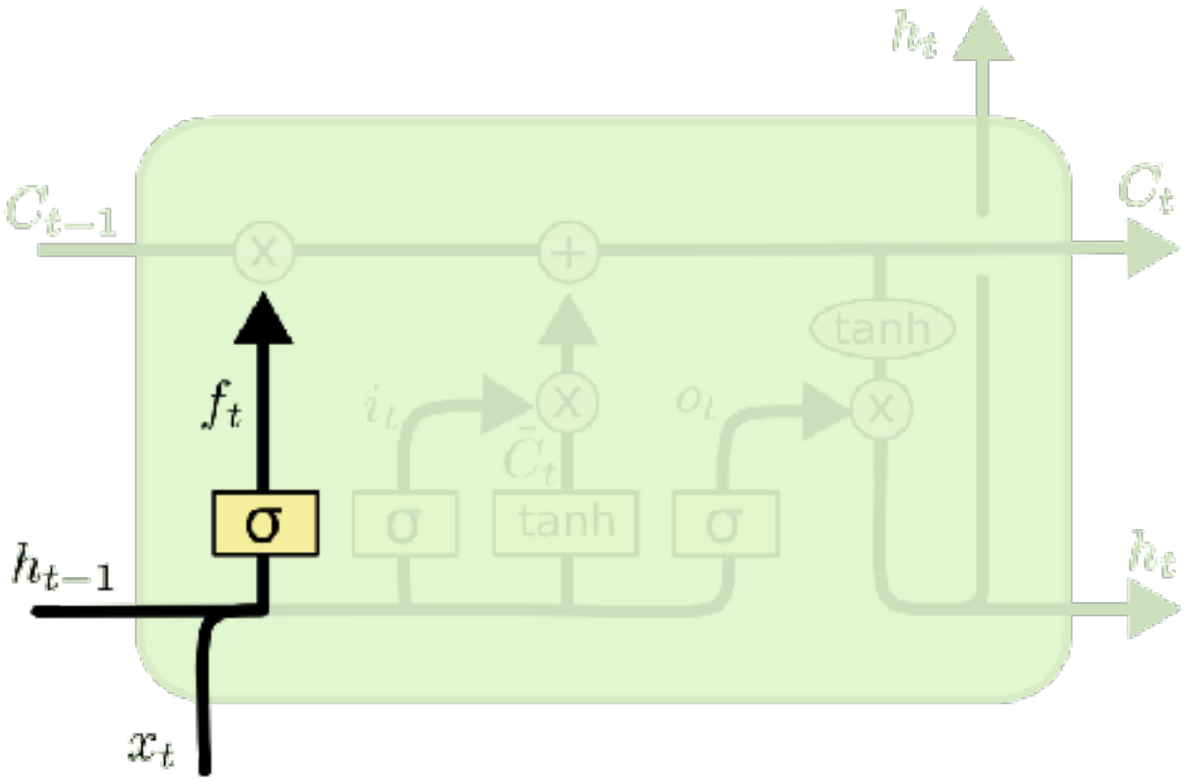


Slide credit: Christopher Olah

LSTMs!

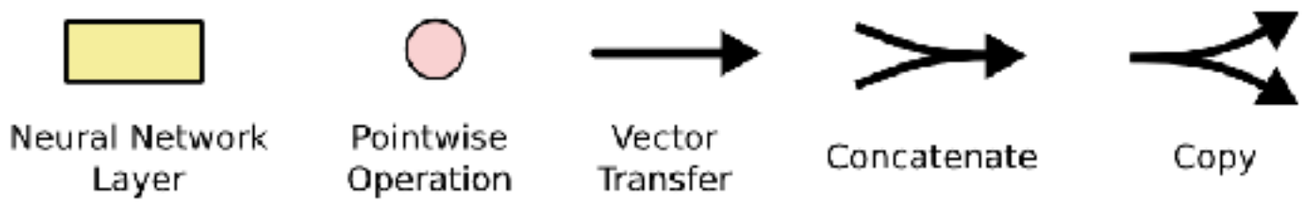
Long Short Term Memory networks

First step: decide what information we're going to throw away from the cell state.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Sigmoid!

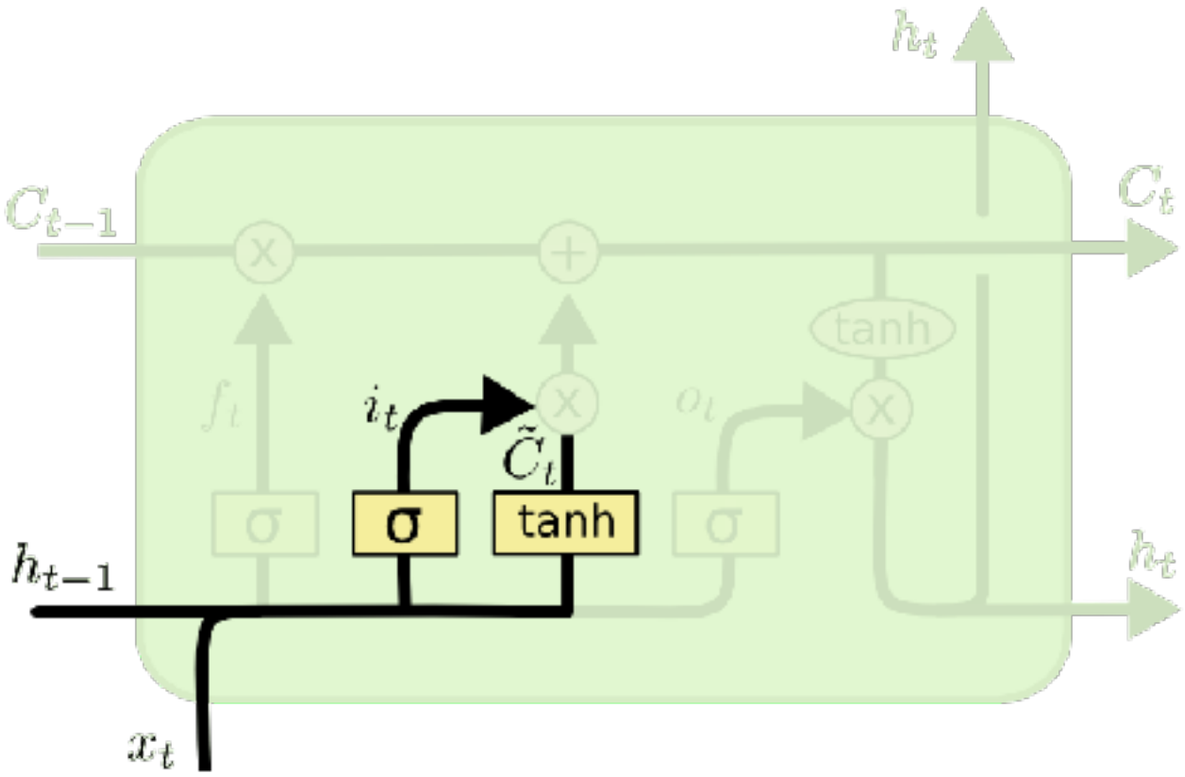


Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

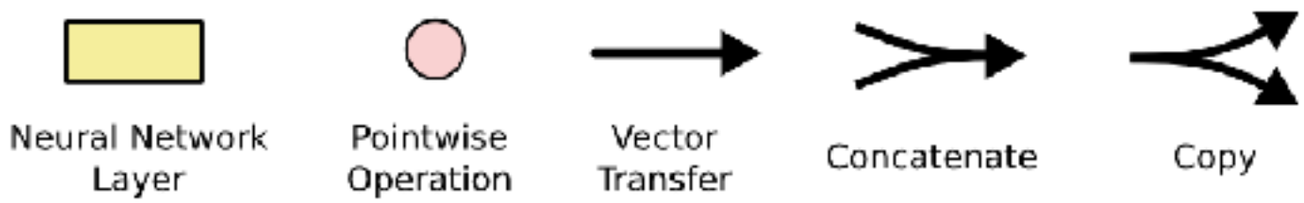
Second step: decide what information we're going to store in the cell state.



Sigmoid!

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C$$

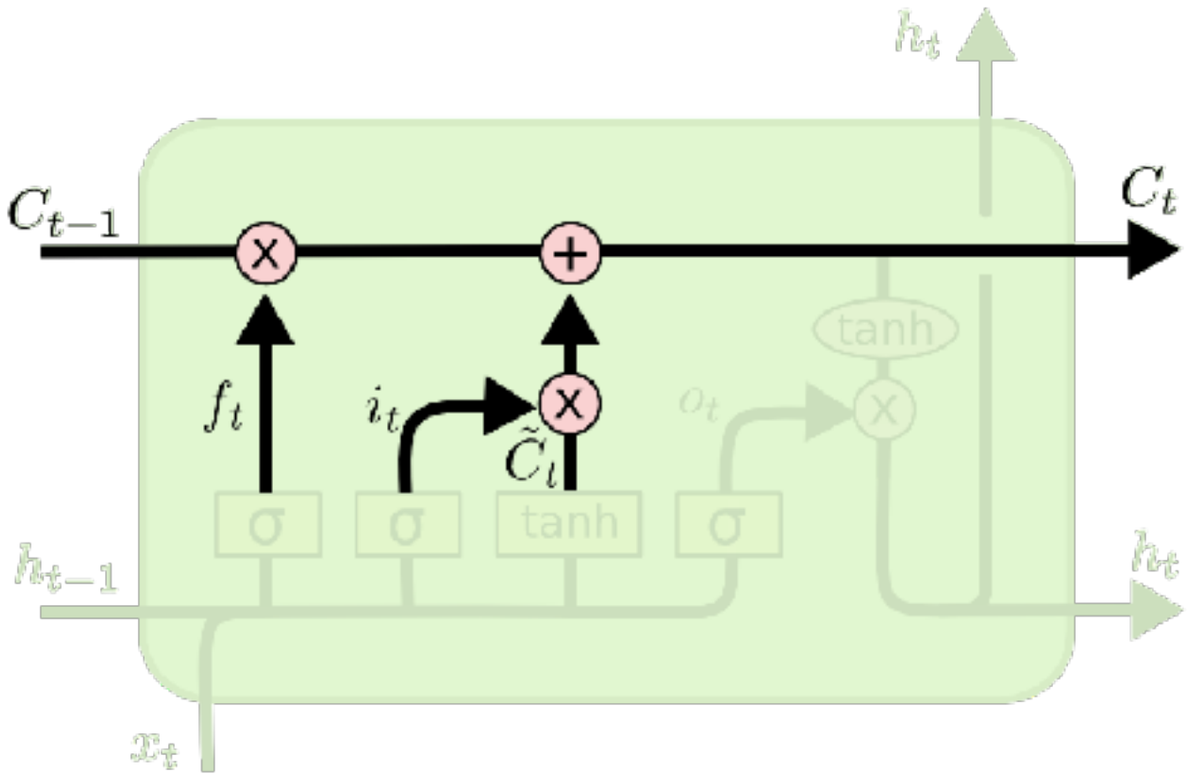


Slide credit: Christopher Olah

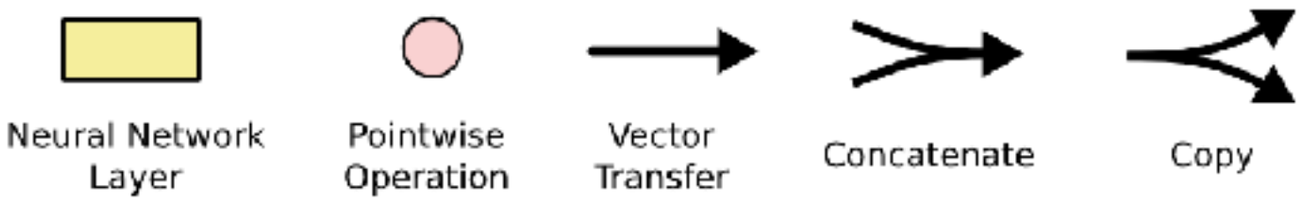
LSTMs!

Long Short Term Memory networks

Merge the last two steps in the cell gate



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

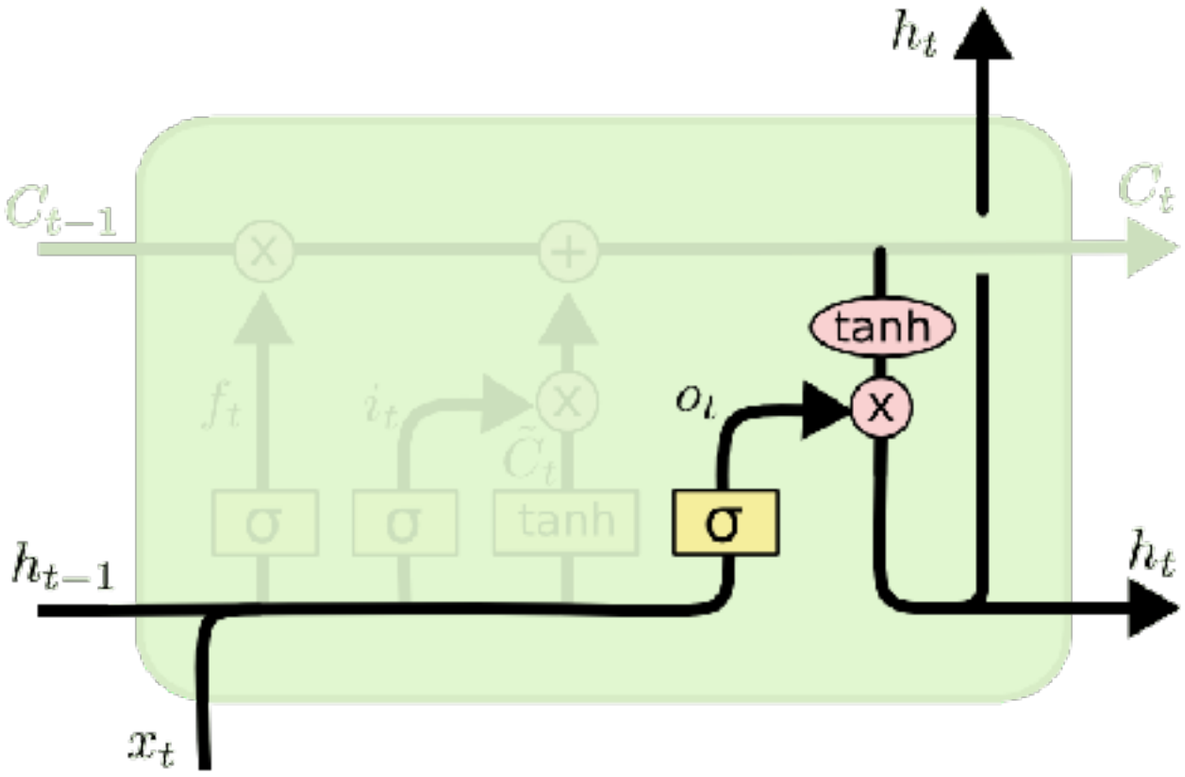


Slide credit: Christopher Olah

LSTMs!

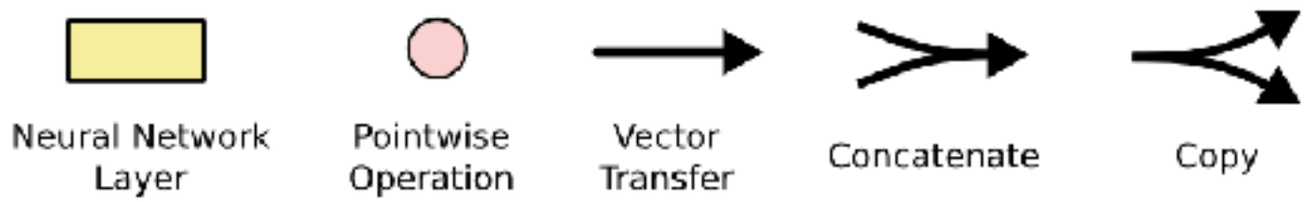
Long Short Term Memory networks

Finally, a more standard output (weighted by the cell gate!)



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

RNN Bible
@RNN_Bible

Suivre

3:11 The LORD is my strength and glory,
strifes of corrupt things: and thou wilt be thy
supplications.

LSTMs!

Long Short Term Memory networks

SEED: Jobs

Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going to take out the fact that the American people have fought to make sure that they have to be able to protect our part. It was a chance to stand together to completely look for the commitment to borrow from the American people. And the fact is the men and women in uniform and the millions of our country with the law system that we should be a strong stretcks of the forces that we can afford to increase our spirit of the American people and the leadership of our country who are on the Internet of American lives.

Thank you very much. God bless you, and God bless the United States of America.