THE UNIVERSITY OF BRITISH COLUMBIA

Topics in AI (CPSC 532L):
Multimodal Learning with Vision, Language and Sound

Lecture 3: Introduction to Deep Learning (continued)

Course **Logistics**

- Update on **course registrations** - 6 seats left now
- Microsoft **Azure** credits and tutorial
- **Assignment 1** … any questions?
- Question about **constants in a computational graph**

Course **Logistics**

- Update on **course registrations** - 6 seats left now
- Microsoft **Azure** credits and tutorial
- **Assignment 1** … any questions?
- Question about **constants in a computational graph**

$$f(x) = a \cdot x + b$$

Course **Logistics**

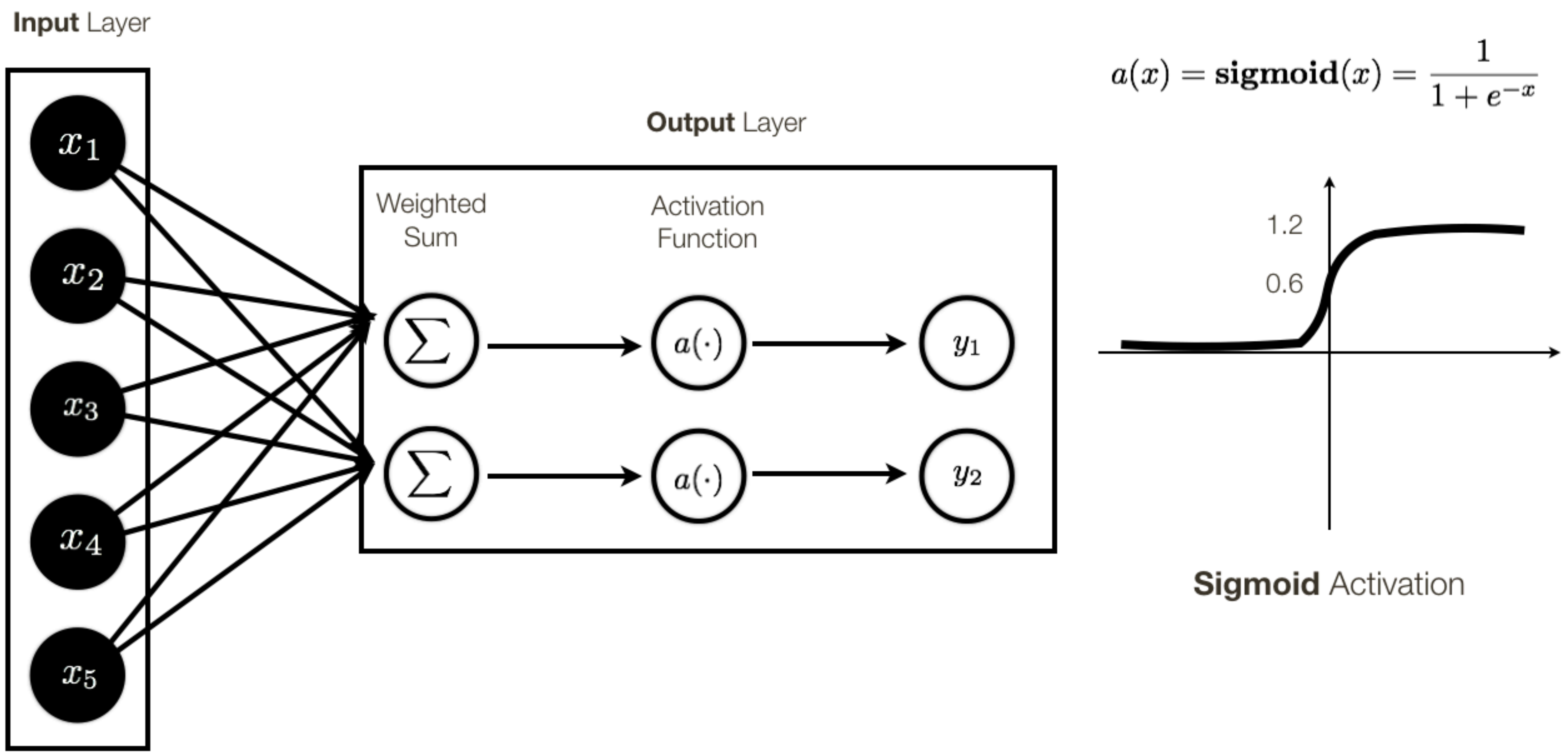
- Update on **course registrations** - 6 seats left now
- Microsoft **Azure** credits and tutorial
- **Assignment 1** … any questions?
- Question about **constants in a computational graph**

$$f(x) = a \cdot x + b$$

$$f(x, a, b) = a \cdot x + b$$

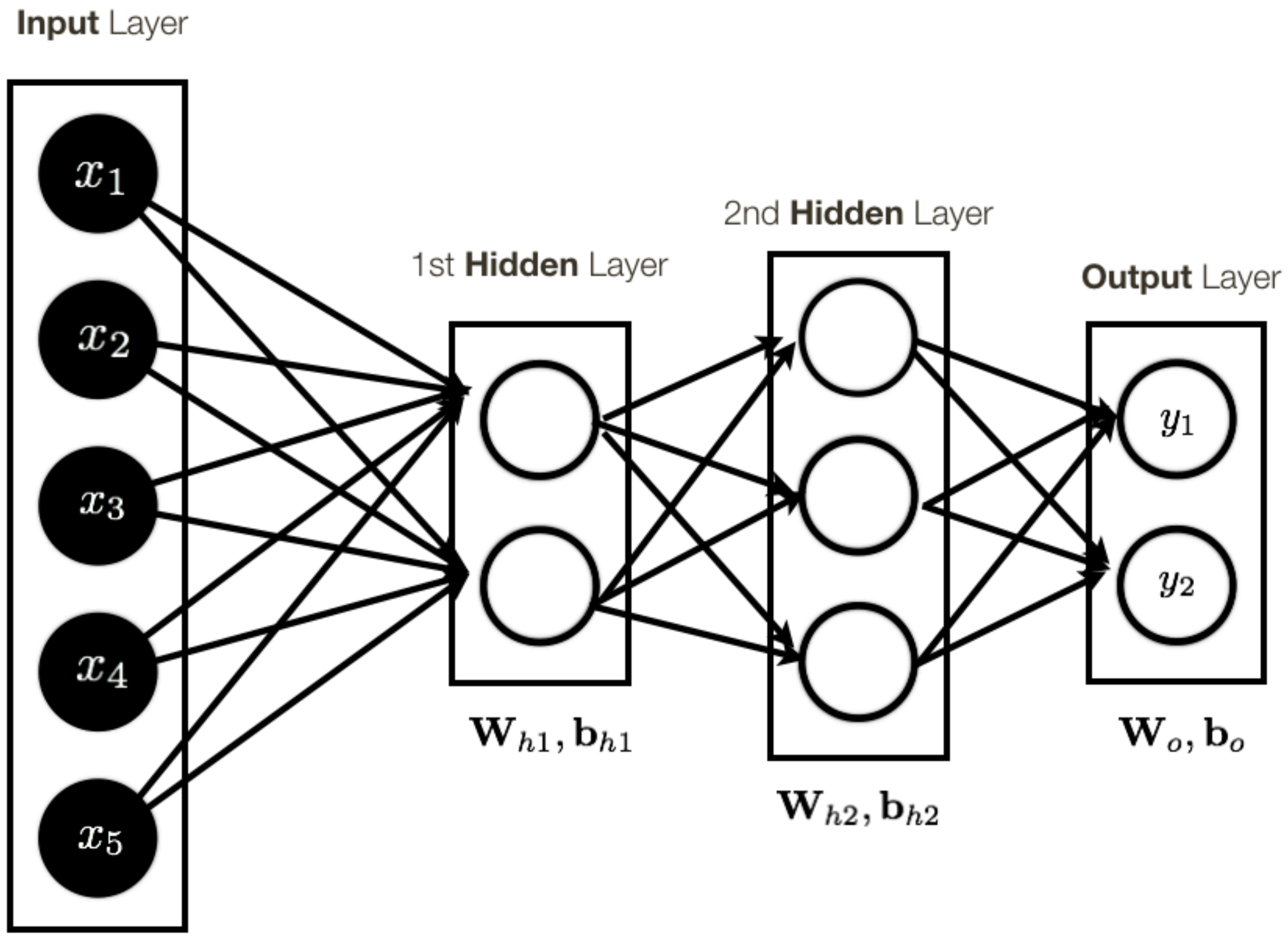
Short **Review** …

- Introduced the basic building block of Neural Networks **(MLP/FC) layer**



Short **Review** …

- Introduced the basic building block of Neural Networks **(MLP/FC) layer**
- How do we **stack these layers** up to make a Deep NN

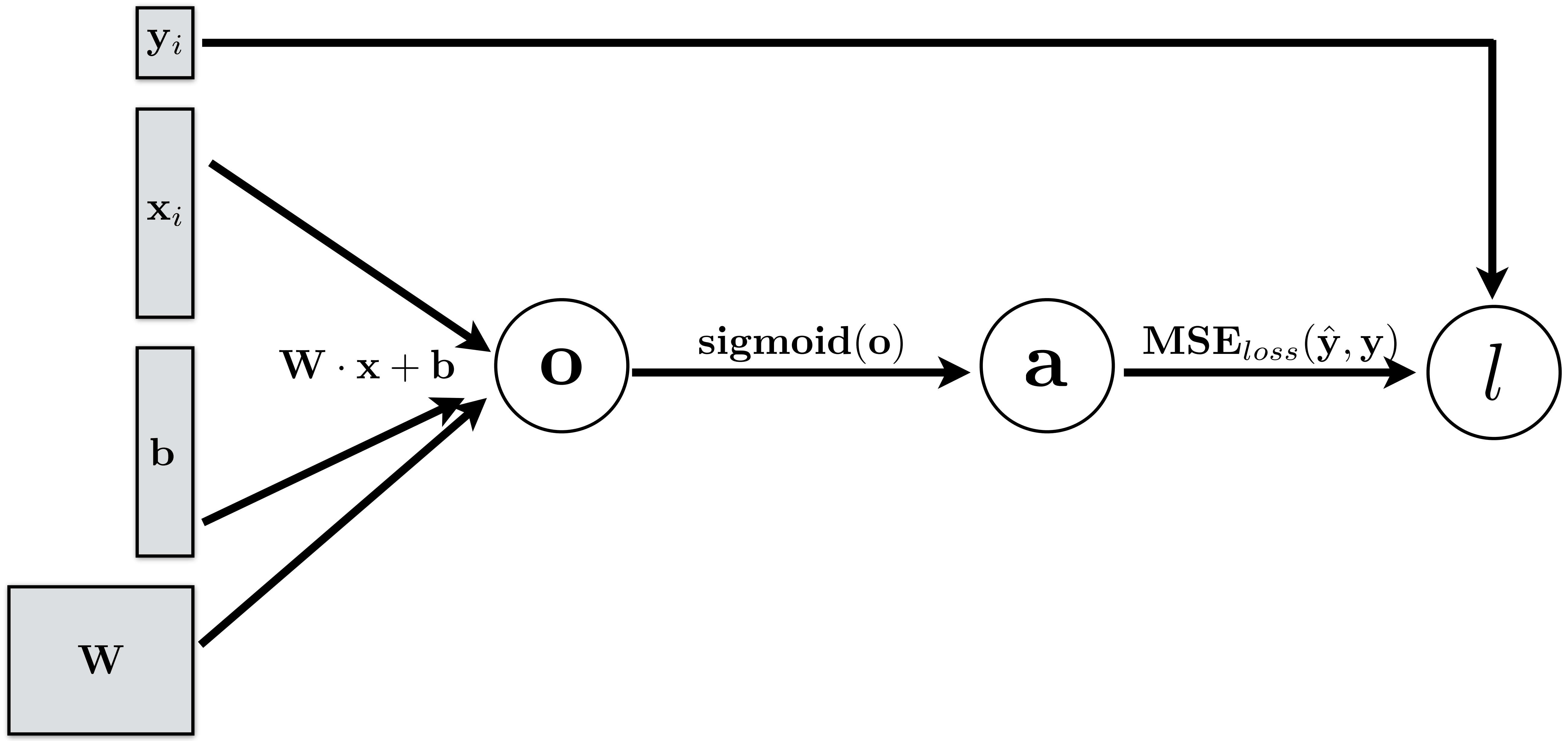


Short **Review** ...

- Introduced the basic building block of Neural Networks **(MLP/FC) layer**

- How do we **stack these layers** up to make a Deep NN

- Basic **NN operations** (implemented using **computational graph**)



Short **Review** …

- Introduced the basic building block of Neural Networks **(MLP/FC) layer**
- How do we **stack these layers** up to make a Deep NN
- Basic **NN operations** (implemented using **computational graph**)

Prediction / Inference

Function evaluation

(a.k.a. **ForwardProp**)

Short **Review** …

- Introduced the basic building block of Neural Networks **(MLP/FC) layer**
- How do we **stack these layers** up to make a Deep NN
- Basic **NN operations** (implemented using **computational graph**)

Prediction / Inference	Parameter **Learnings**
Function evaluation (a.k.a. **ForwardProp**)	(Stochastic) Gradient Descent (needs **derivatives**)

Short **Review** …

- Introduced the basic building block of Neural Networks **(MLP/FC) layer**
- How do we **stack these layers** up to make a Deep NN
- Basic **NN operations** (implemented using **computational graph**)

Prediction / Inference

Function evaluation

(a.k.a. **ForwardProp**)

Parameter **Learnings**

(Stochastic) Gradient Descent (needs **derivatives**)

- **Numerical** differentiation (not accurate)
- **Symbolic** differential (intractable)
- AutoDiff **Forward** (computationally expensive)
- AutoDiff **Backward / BackProp**

Short **Review** …

- Introduced the basic building block of Neural Networks **(MLP/FC) layer**
- How do we **stack these layers** up to make a Deep NN
- Basic **NN operations** (implemented using **computational graph**)

Prediction / Inference

Function evaluation

(a.k.a. **ForwardProp**)

Parameter **Learnings**

(Stochastic) Gradient Descent (needs **derivatives**)

- **Numerical** differentiation (not accurate)
- **Symbolic** differential (intractable)
- AutoDiff **Forward** (computationally expensive)
- AutoDiff **Backward / BackProp**

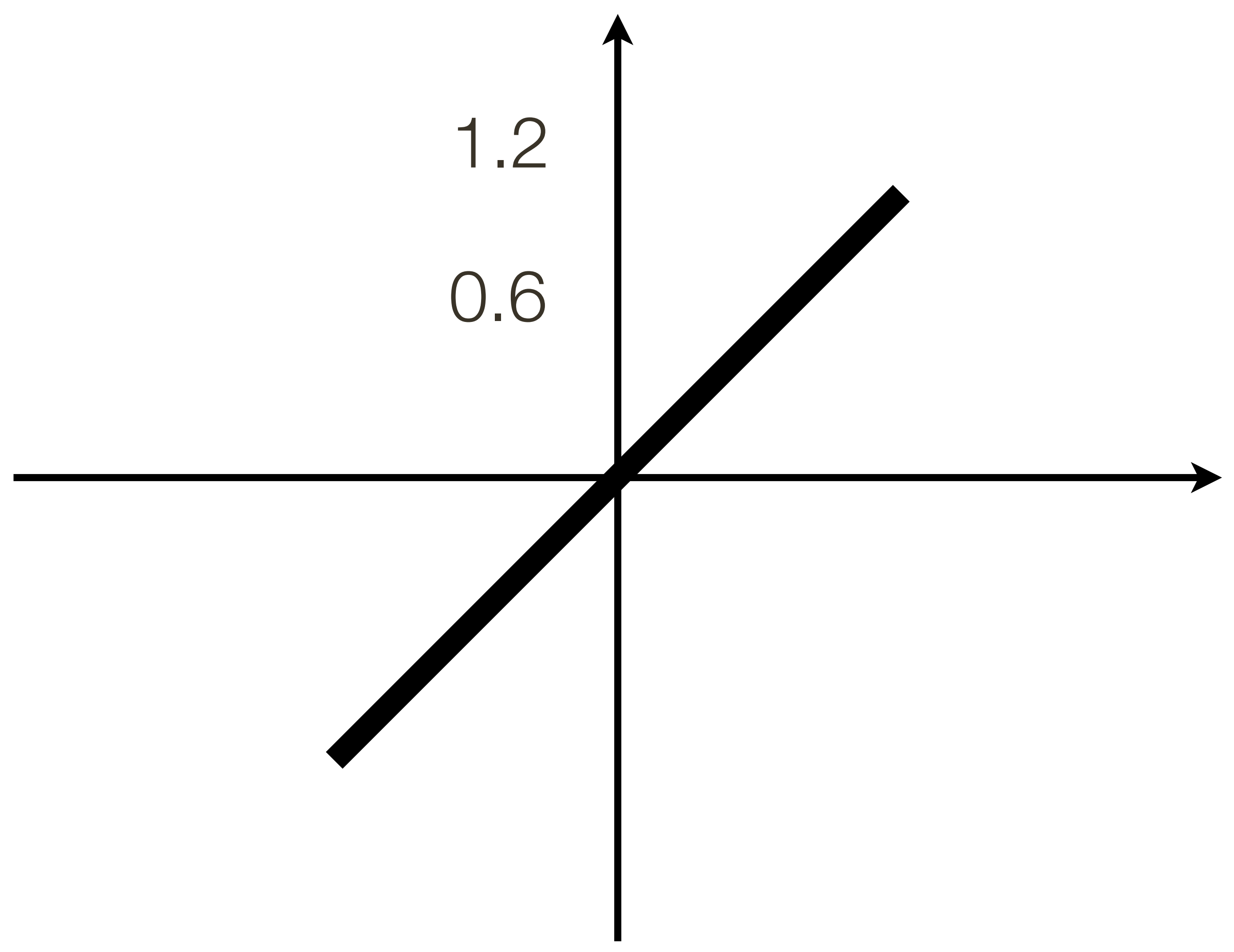
- Different **activation functions** and saturation problem

Follow up (some intuition)

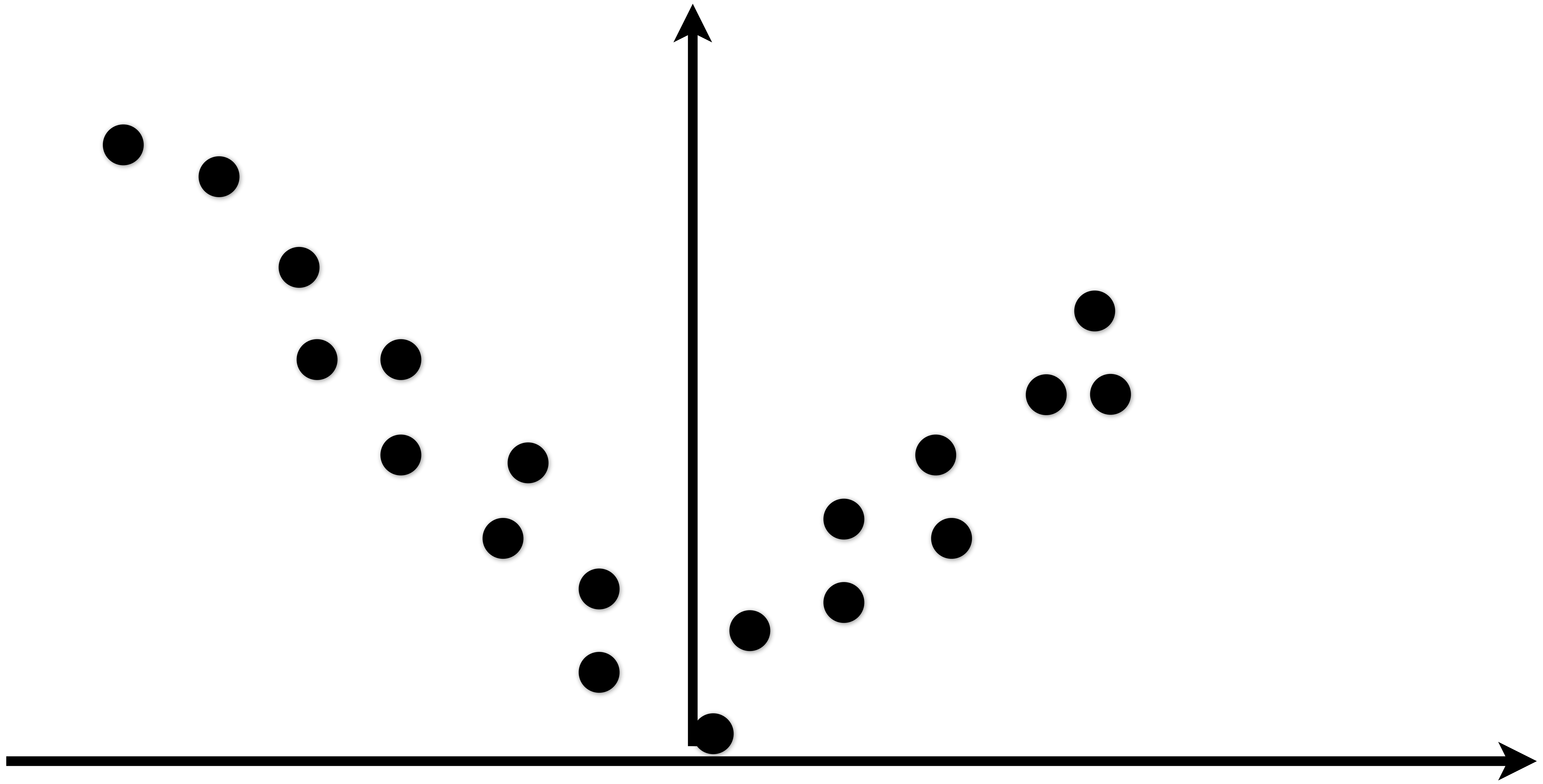
- Affine transformation of the input, followed by an activation

- Another interpretation of the NN, affine combination of activation functions

$$a(x) = x$$



Linear Activation

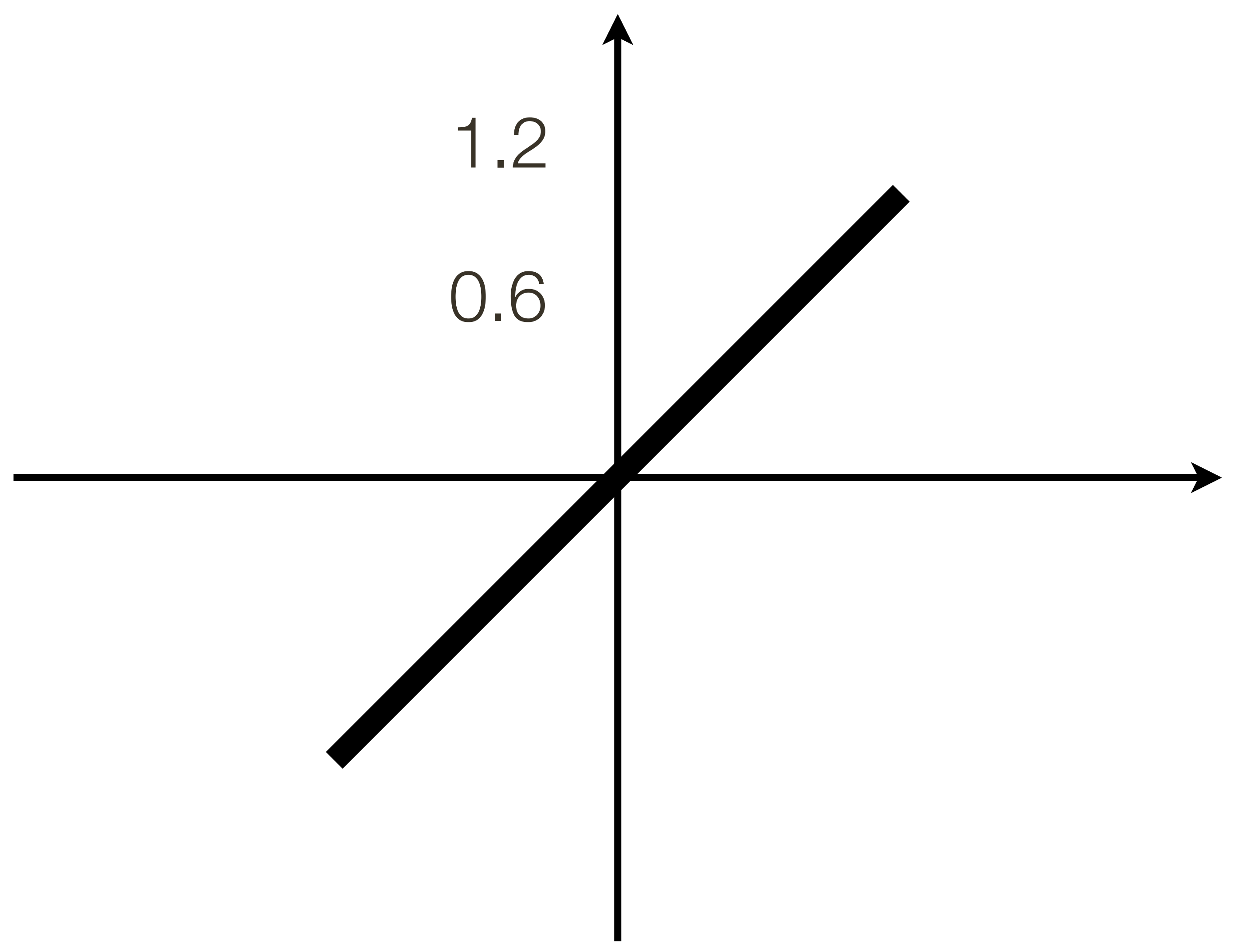


Follow up (some intuition)

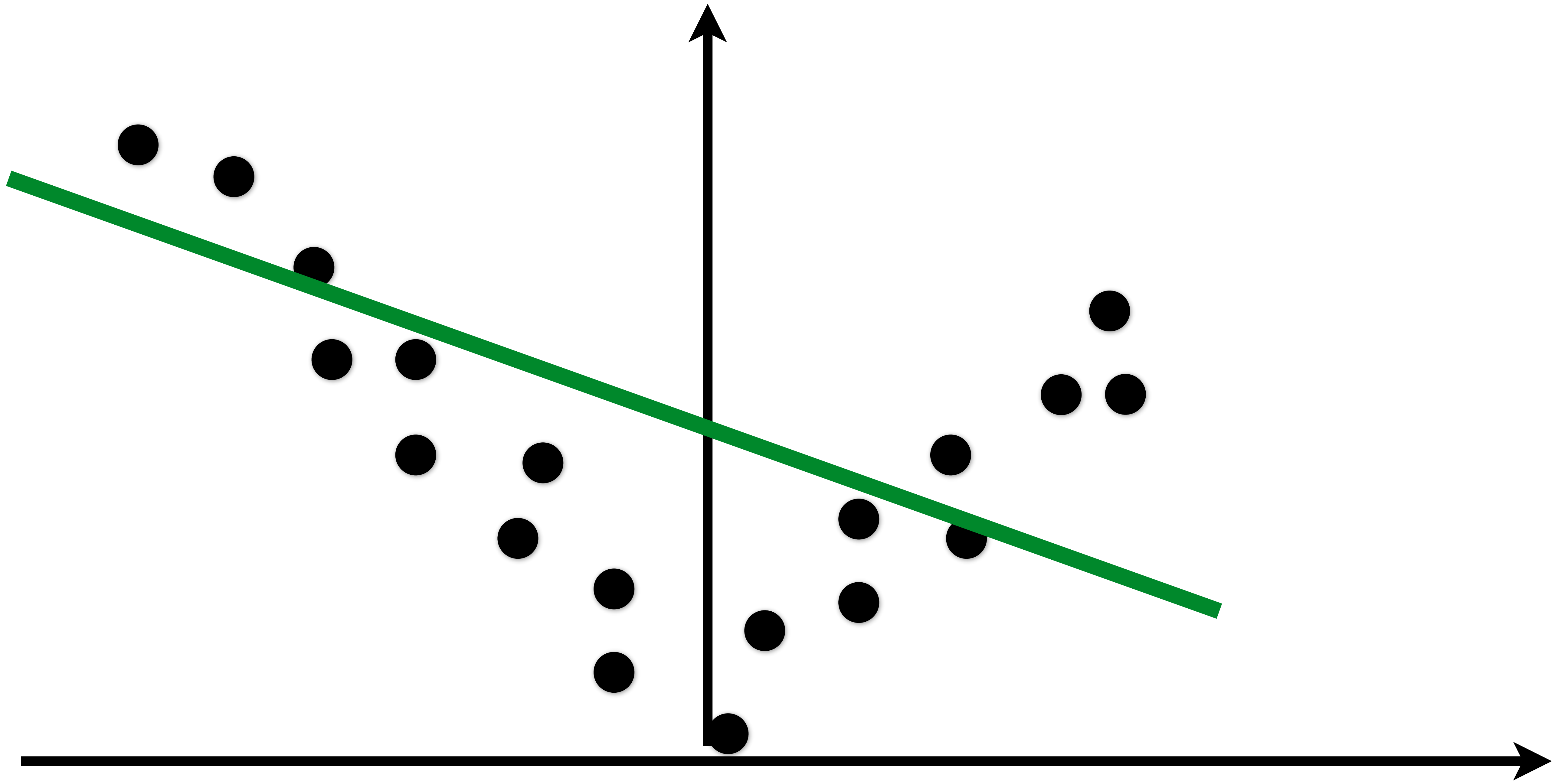
- Affine transformation of the input, followed by an activation

- Another interpretation of the NN, affine combination of activation functions

$$a(x) = x$$



Linear Activation

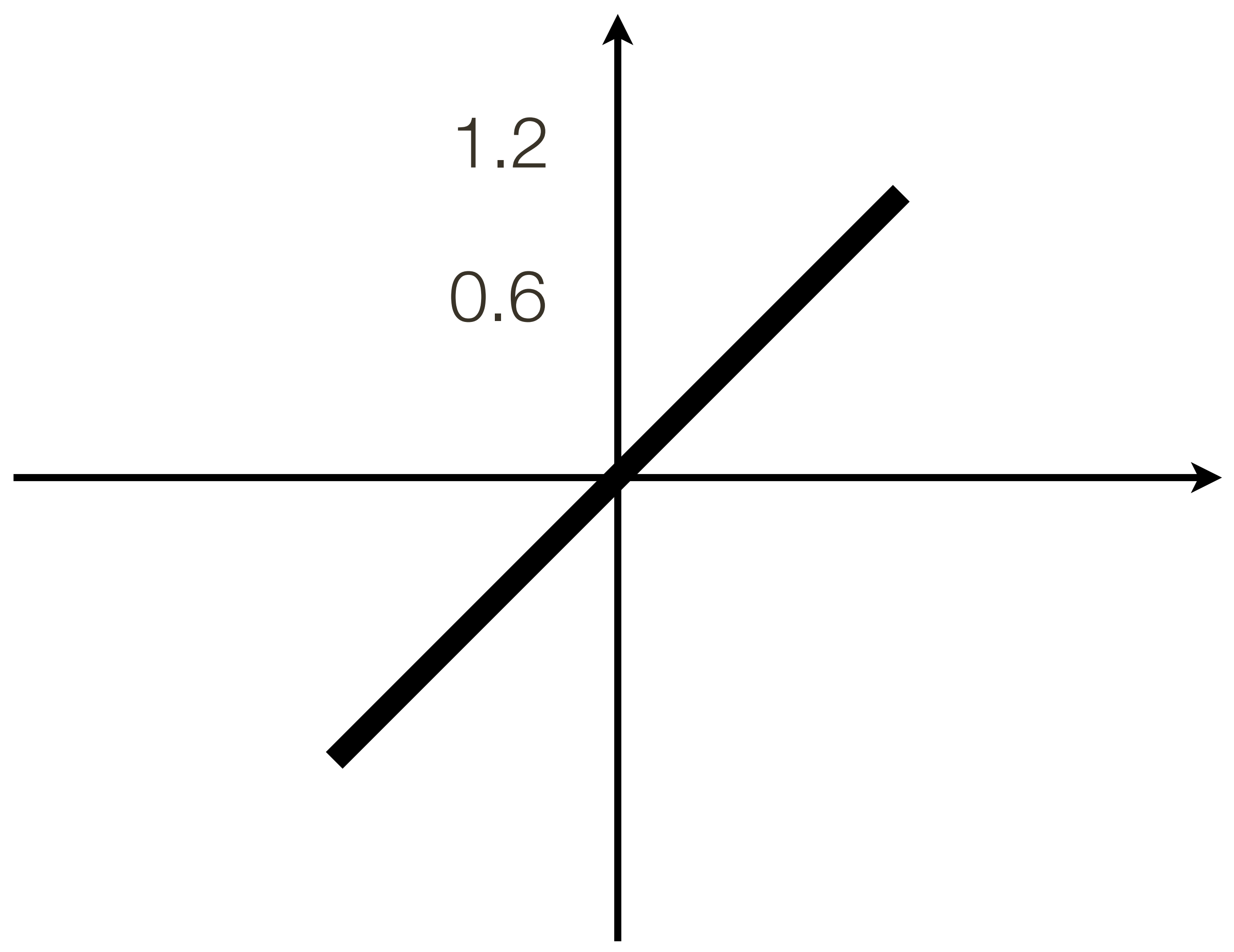


Follow up (some intuition)

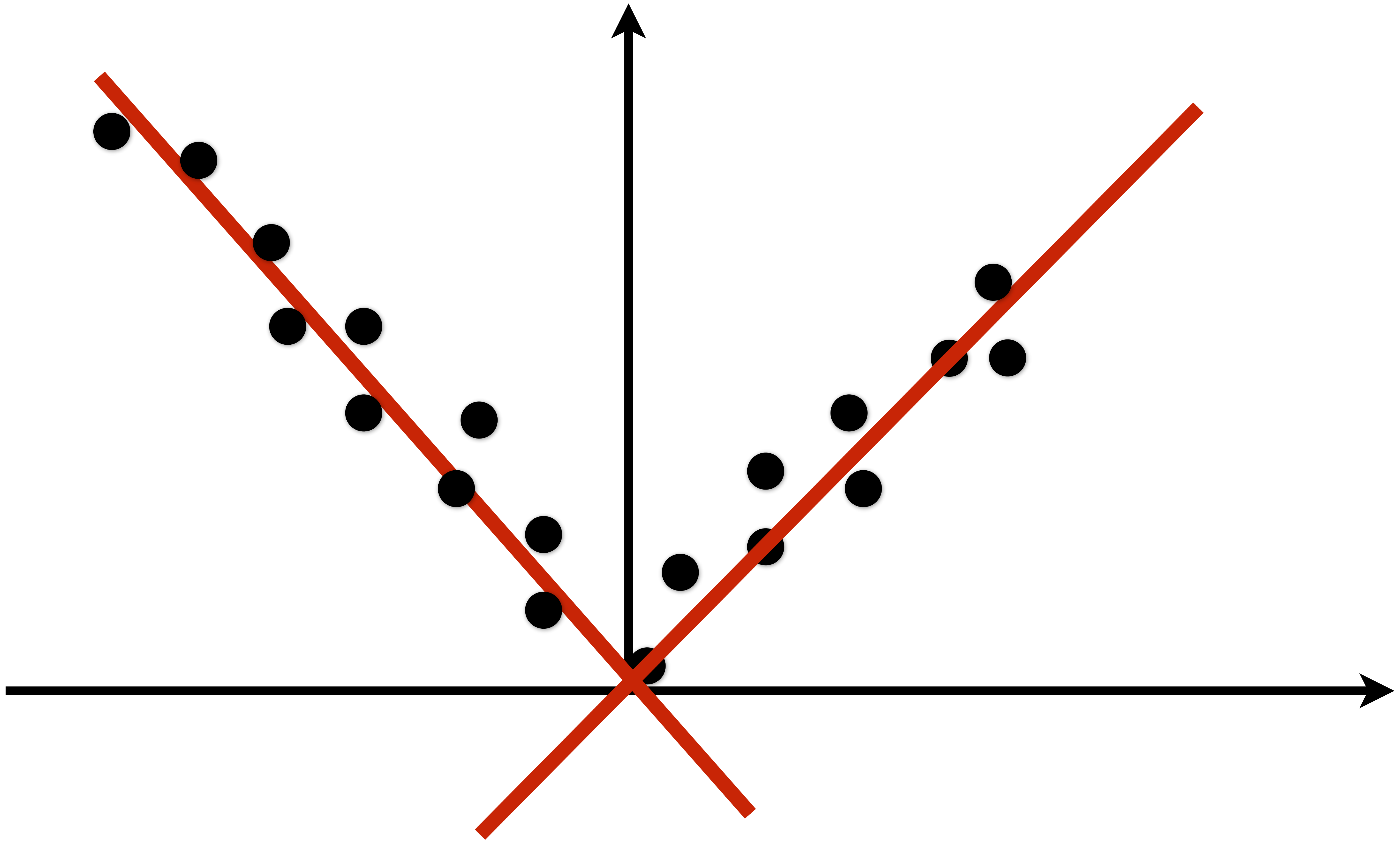
- Affine transformation of the input, followed by an activation

- Another interpretation of the NN, affine combination of activation functions

$$a(x) = x$$



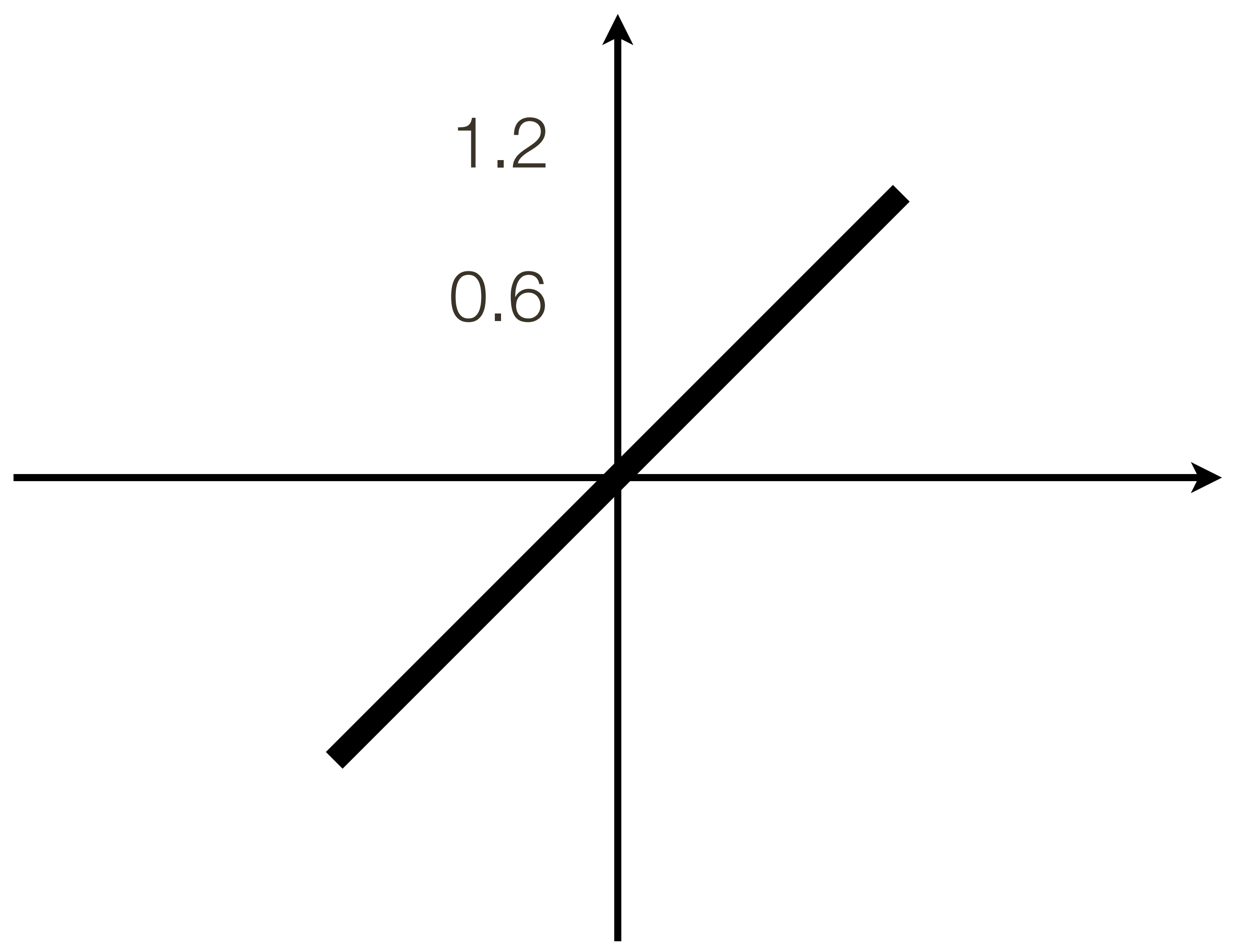
Linear Activation



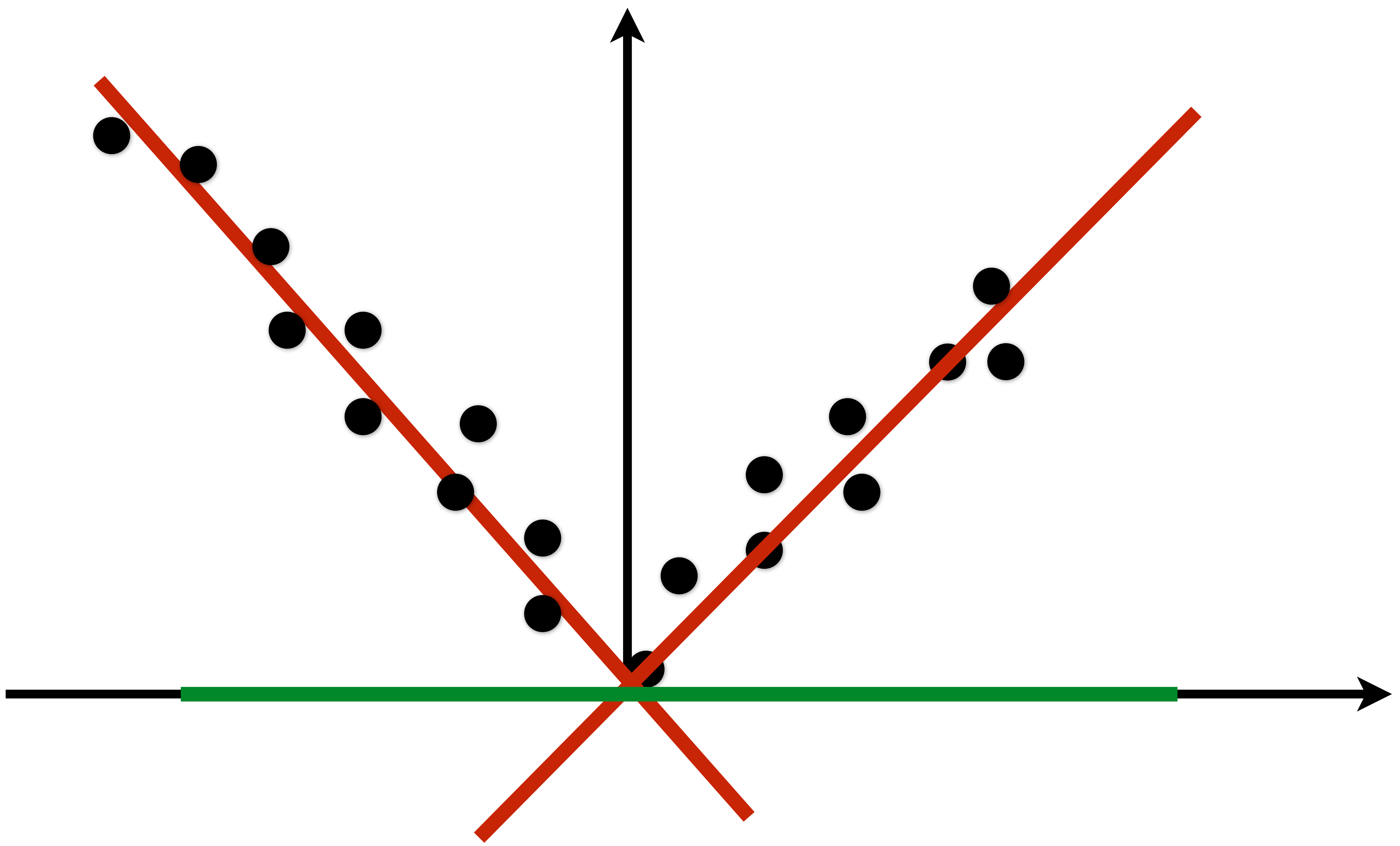
Follow up (some intuition)

- Affine transformation of the input, followed by an activation
- Another interpretation of the NN, affine combination of activation functions

$$a(x) = x$$



Linear Activation

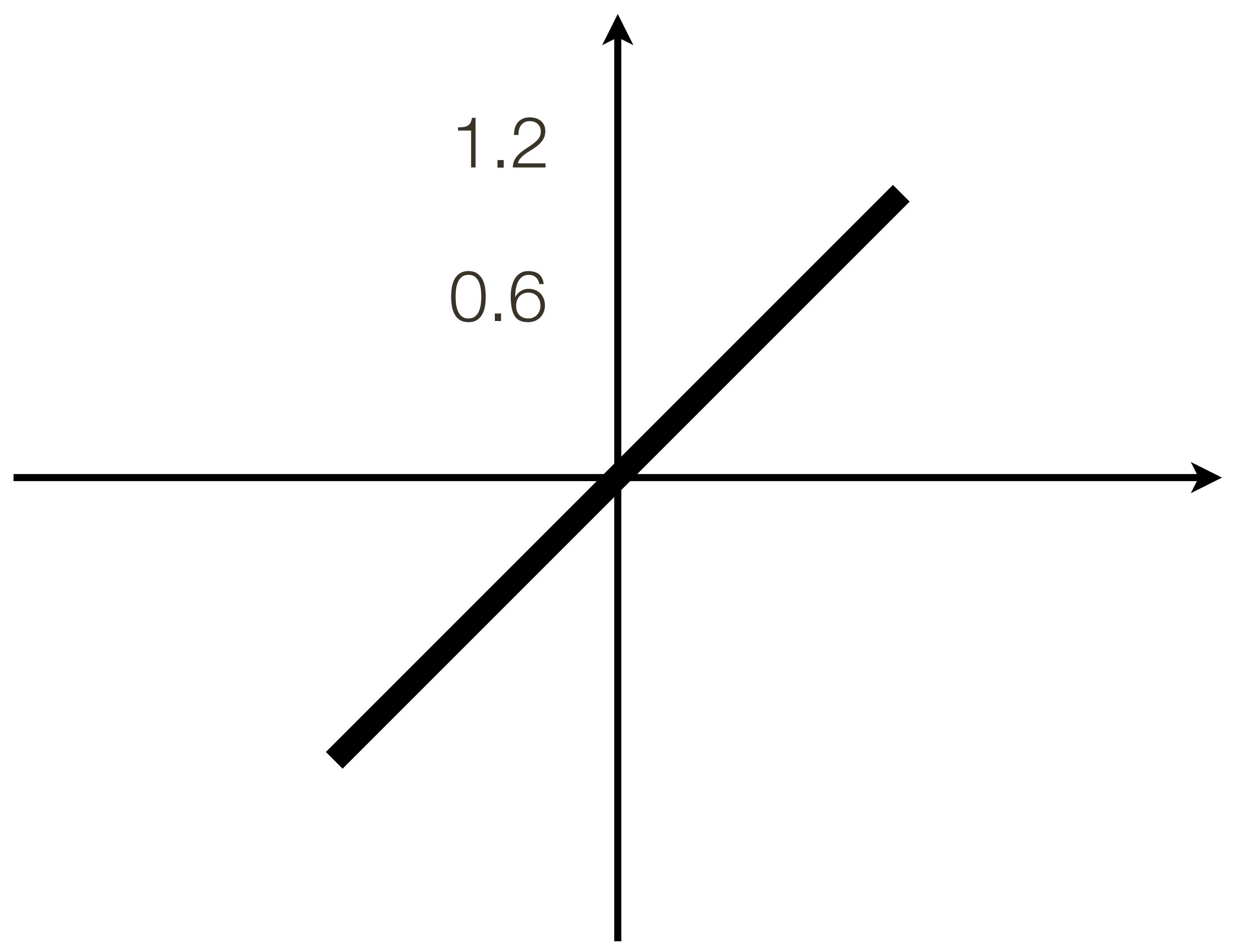


Follow up (some intuition)

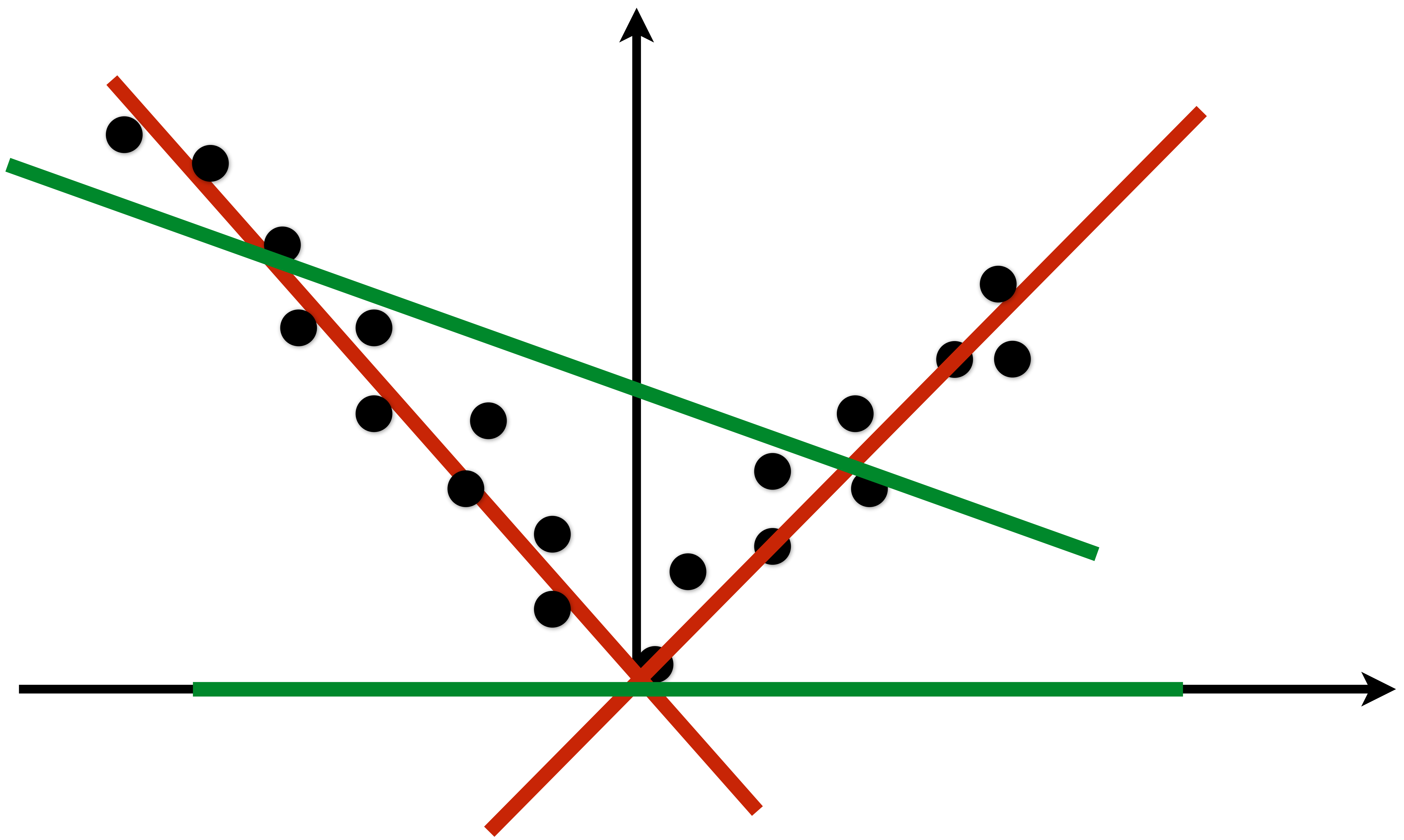
- Affine transformation of the input, followed by an activation

- Another interpretation of the NN, affine combination of activation functions

$$a(x) = x$$



Linear Activation

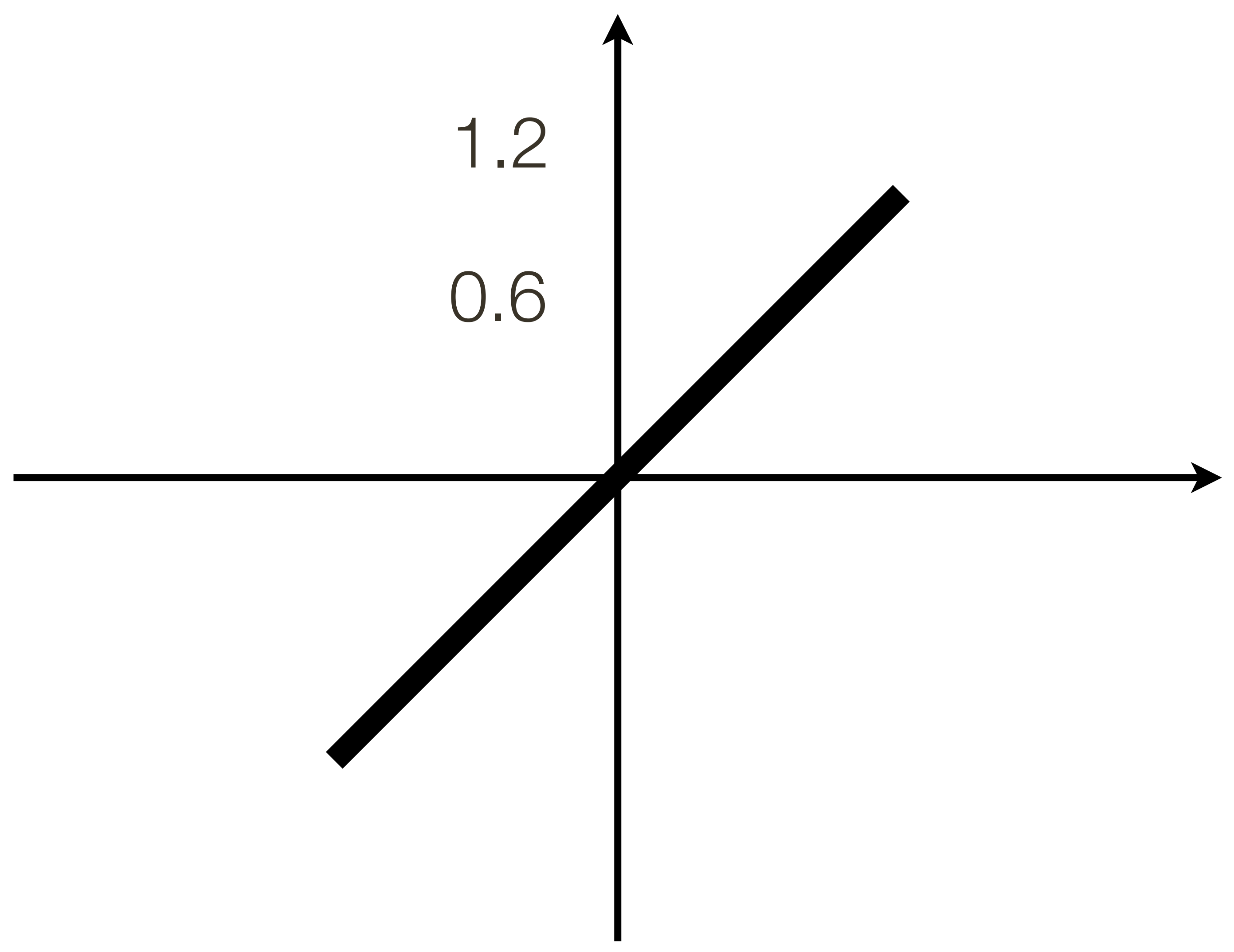


Follow up (some intuition)

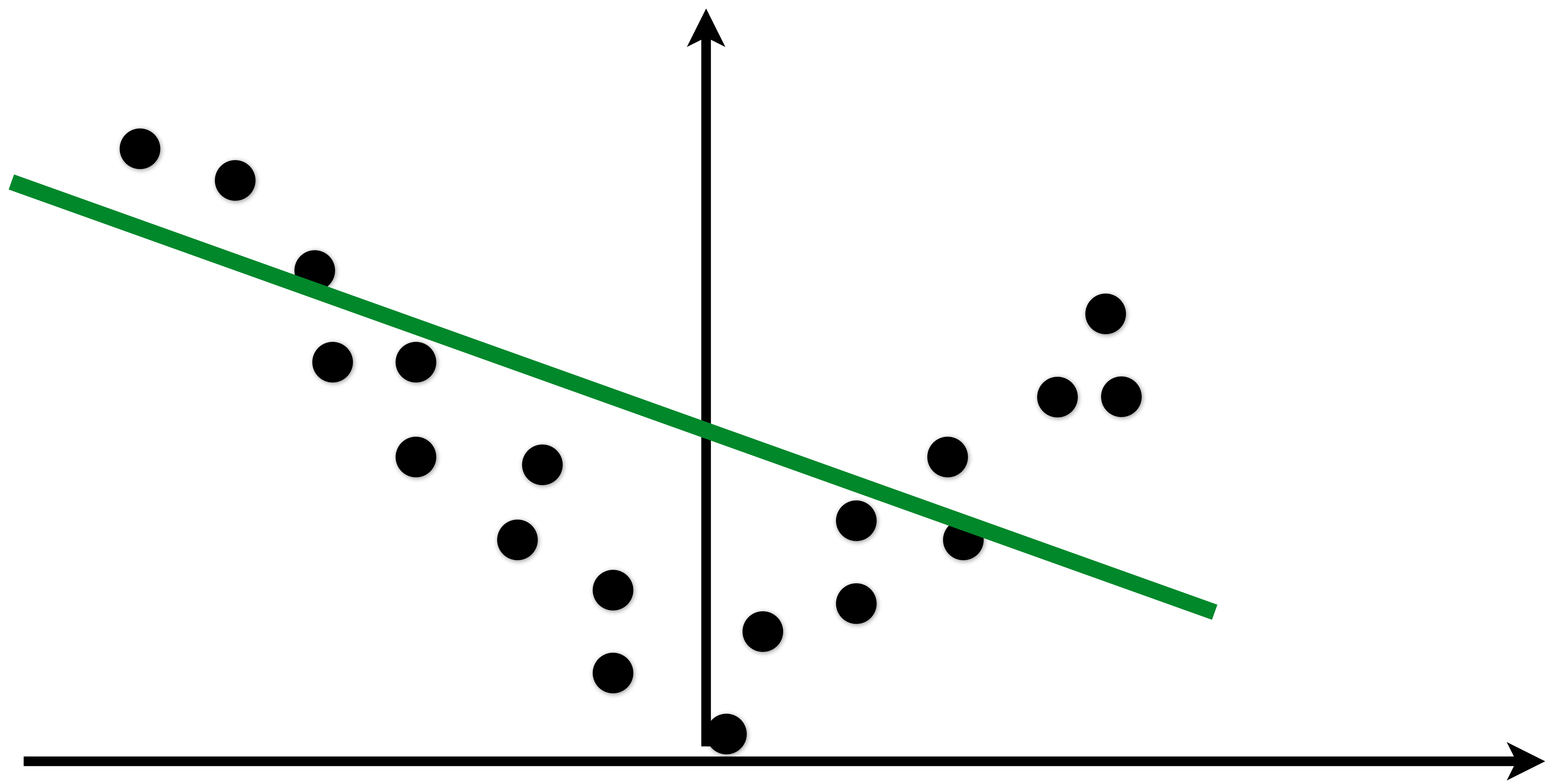
- Affine transformation of the input, followed by an activation

- Another interpretation of the NN, affine combination of activation functions

$$a(x) = x$$



Linear Activation

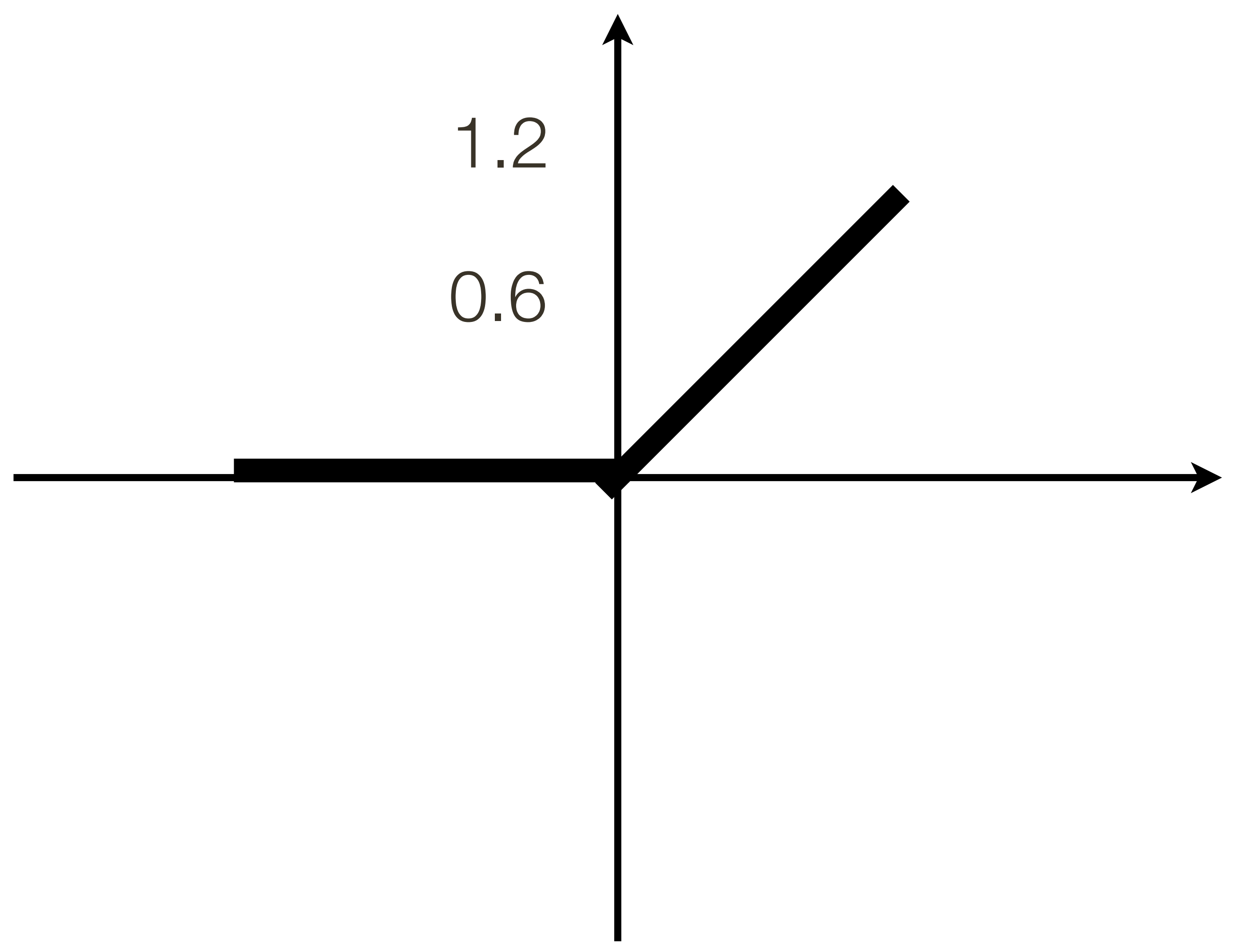


Follow up (some intuition)

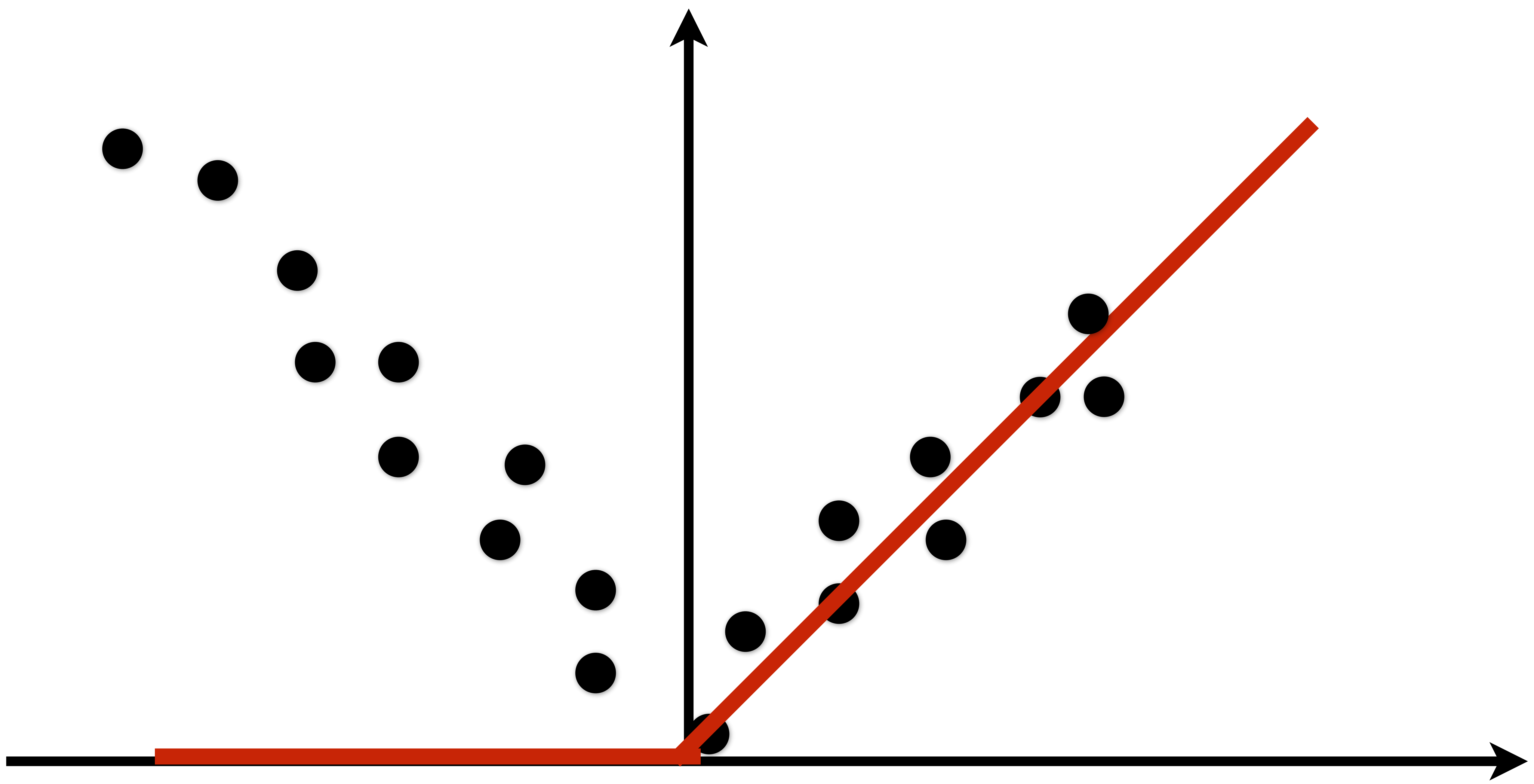
- Affine transformation of the input, followed by an activation

- Another interpretation of the NN, affine combination of activation functions

$$a(x) = max(0, x)$$



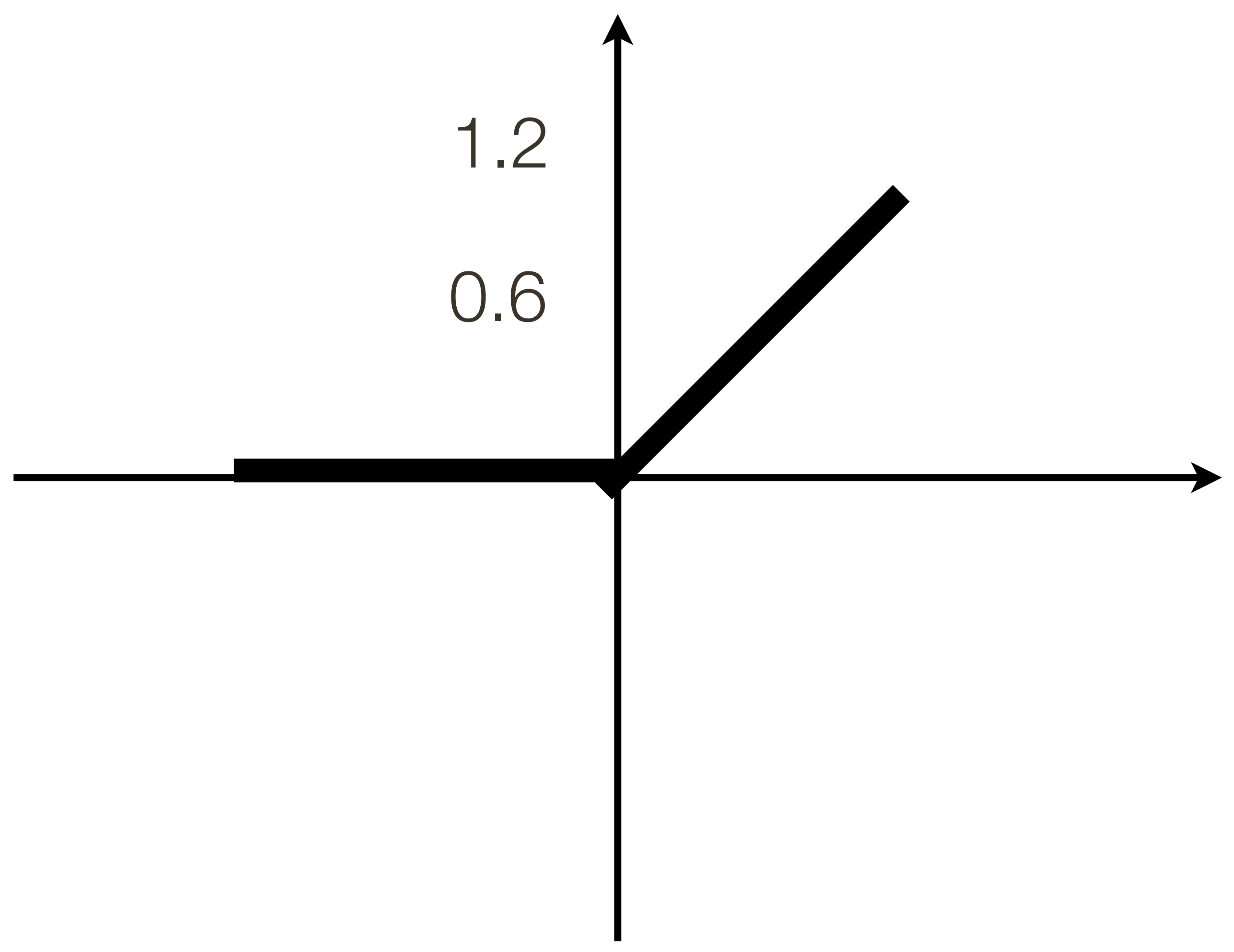
ReLU Activation



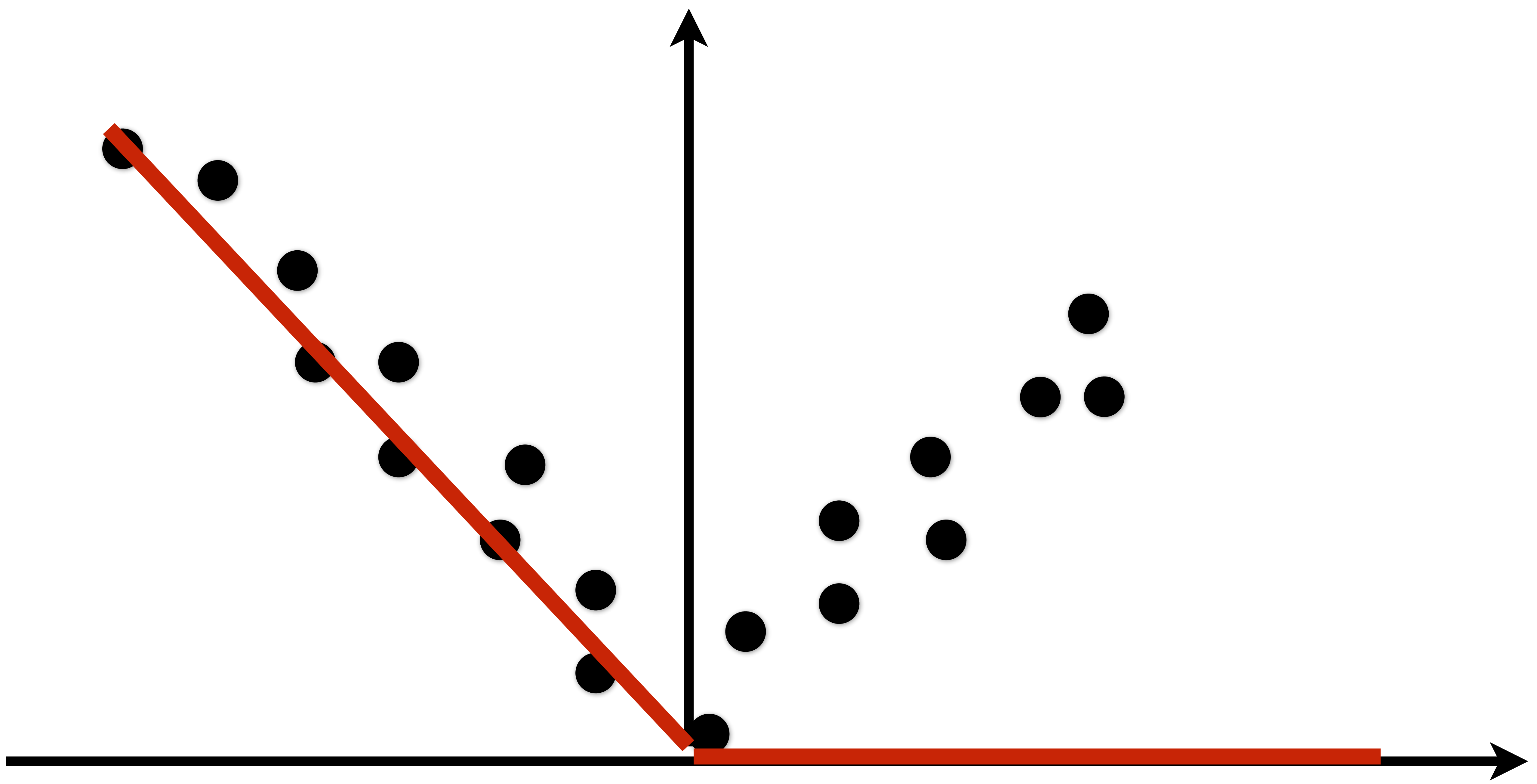
Follow up (some intuition)

- Affine transformation of the input, followed by an activation
- Another interpretation of the NN, affine combination of activation functions

$$a(x) = max(0, x)$$



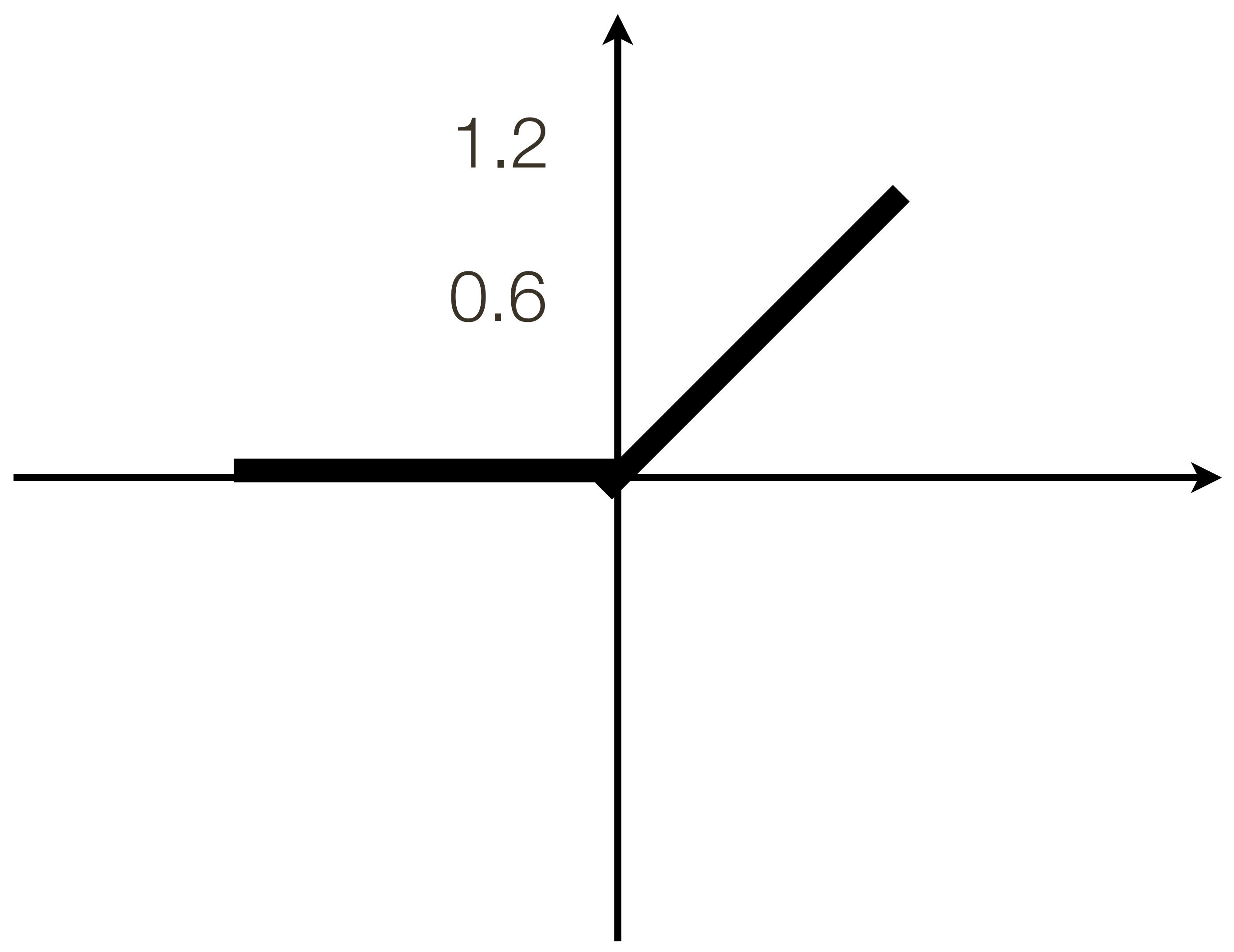
ReLU Activation



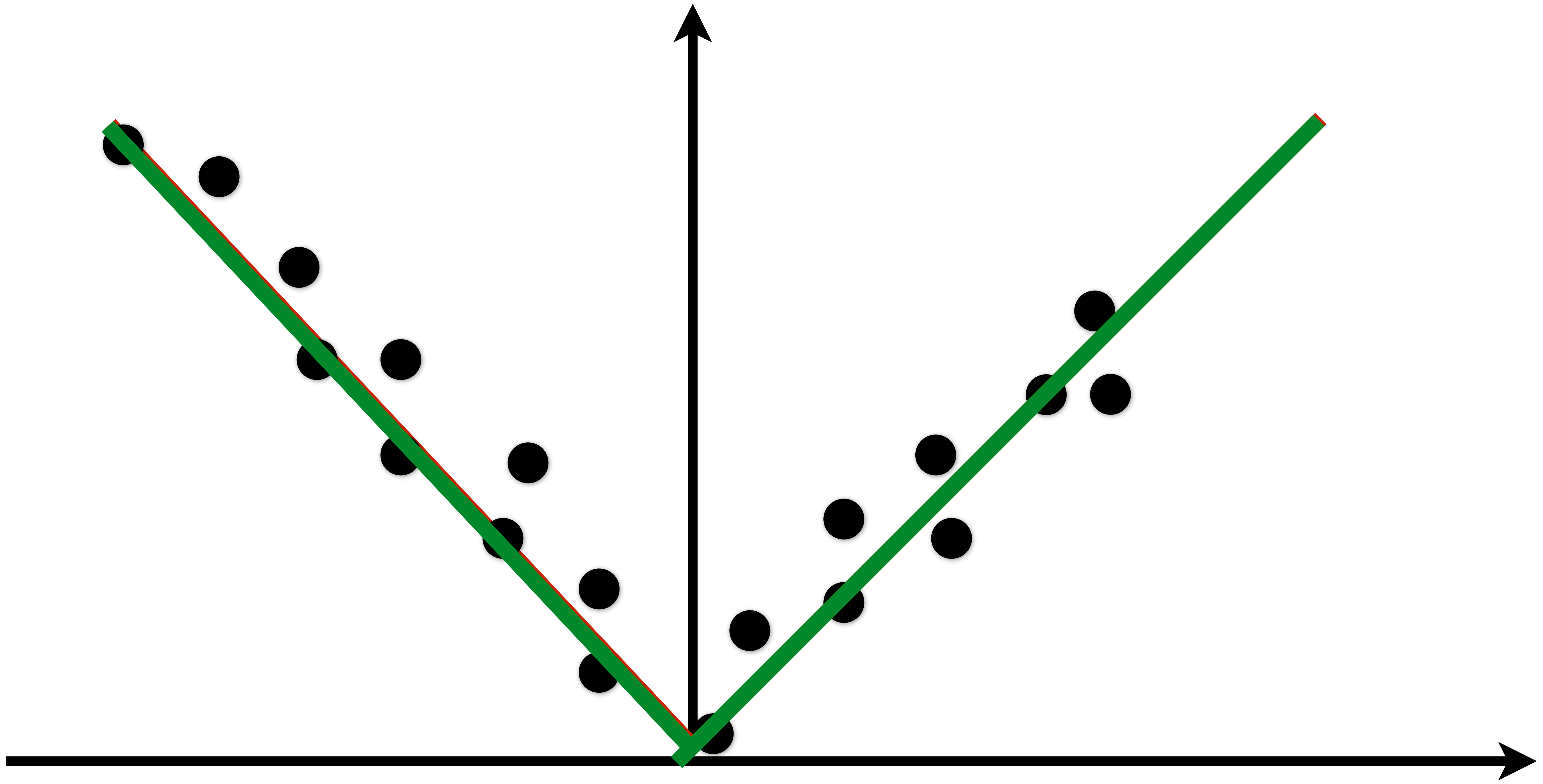
Follow up (some intuition)

- Affine transformation of the input, followed by an activation
- Another interpretation of the NN, affine combination of activation functions

$$a(x) = max(0, x)$$



ReLU Activation



Regularization: L2 or L1 on the weights

L2 Regularization: Learn a more (dense) distributed representation

$$R(\mathbf{W}) = ||\mathbf{W}||_2 = \sum_i \sum_j \mathbf{W}_{i,j}^2$$

L1 Regularization: Learn a sparse representation (few non-zero wight elements)

$$R(\mathbf{W}) = ||\mathbf{W}||_1 = \sum_i \sum_j |\mathbf{W}_{i,j}|$$

(others regularizers are also possible)

Example:

$$\mathbf{x} = [1, 1, 1, 1]$$

$$\mathbf{W}_1 = [1, 0, 0, 0]$$

$$\mathbf{W}_2 = \left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right]$$

$$\mathbf{W}_1 \cdot \mathbf{x}^T = \mathbf{W}_2 \cdot \mathbf{x}^T$$

two networks will have identical loss

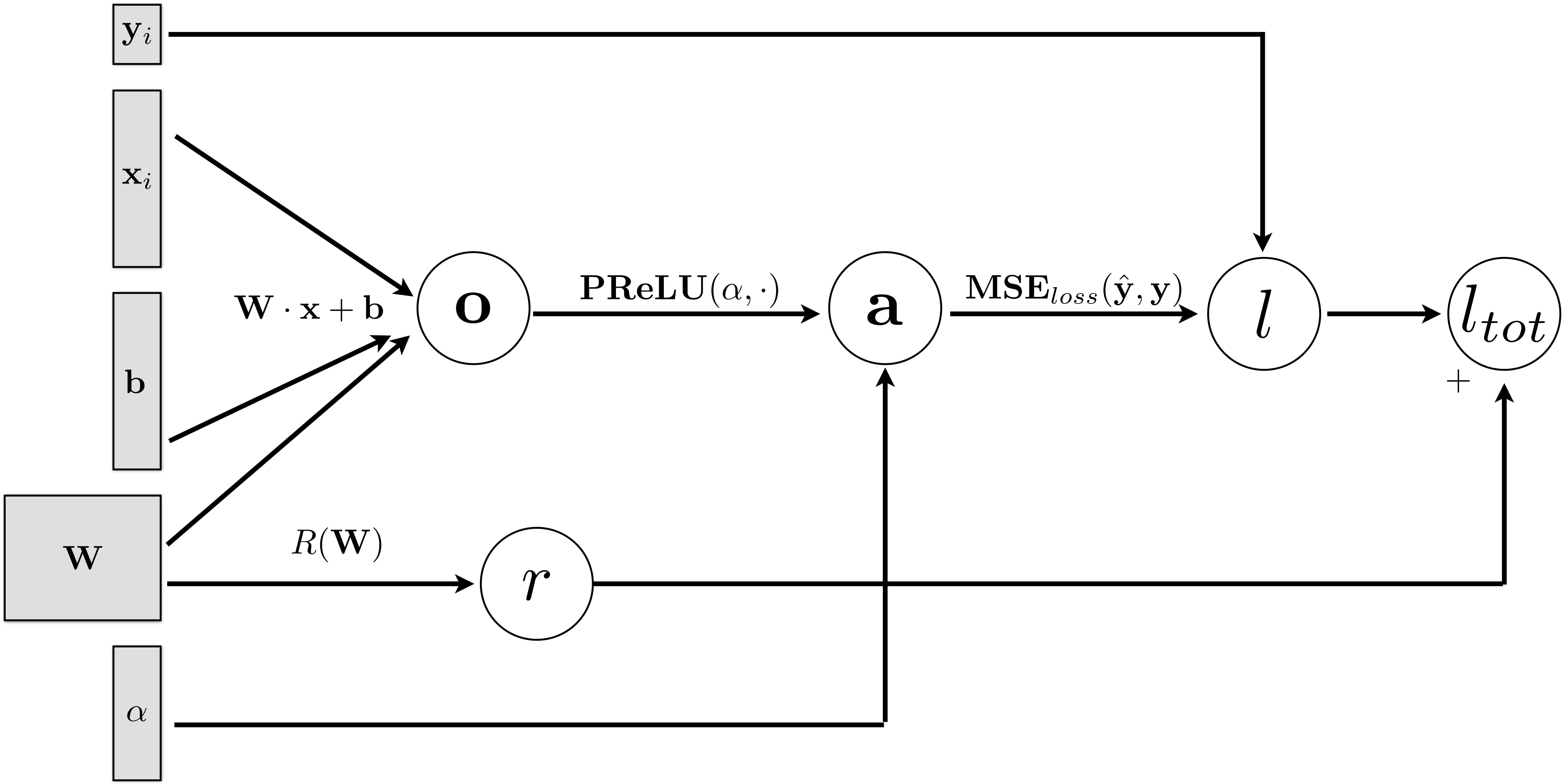
L2 Regularizer:

$$R_{L2}(\mathbf{W}_1) = 1$$
$$R_{L2}(\mathbf{W}_2) = 0.25 \leftarrow$$

L1 Regularizer:

$$R_{L1}(\mathbf{W}_1) = 1 \leftarrow$$
$$R_{L1}(\mathbf{W}_2) = 1 \leftarrow$$

Computational Graph: 1-layer with PReLU + Regularizer



Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

$$\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Benefit:

Improves learning (better gradients, higher learning rate)

[Ioffe and Szegedy, NIPS 2015]

Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

$$\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Benefit:

Improves learning (better gradients, higher learning rate)

Why?

[Ioffe and Szegedy, NIPS 2015]

Activation Function: Sigmoid

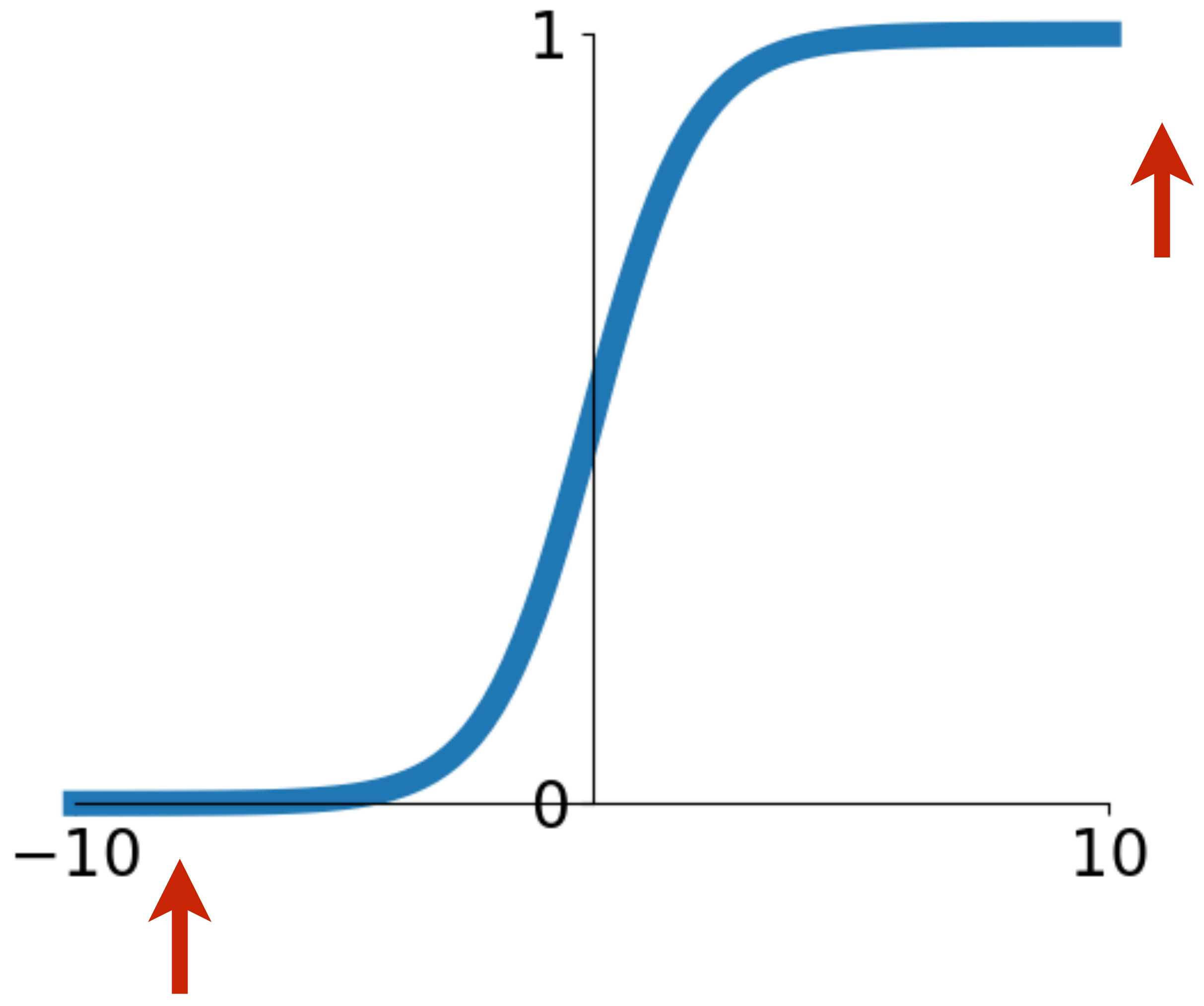
$$x$$

$$a = \textbf{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial\ \textbf{sigmoid}(x)}{\partial x}\frac{\partial \mathcal{L}}{\partial a}$$

$$\frac{\partial \mathcal{L}}{\partial a}$$

$$a(x) = \textbf{sigmoid}(x) = \frac{1}{1+e^{-x}}$$



Sigmoid Activation

Cons:

- Saturated neurons **“kill” the gradients**
- Non-zero centered
- Could be expensive to compute

* slide adopted from Li, Karpathy, Johnson’s **CS231n at Stanford**

Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

$$\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Benefit:

Improves learning (better gradients, higher learning rate)

Typically inserted **before** activation layer

[Ioffe and Szegedy, NIPS 2015]

Activation Function: Sigmoid vs. Tanh

$$a(x) = \mathbf{tanh}(x) = 2 \cdot \mathbf{sigmoid}(2x) - 1$$

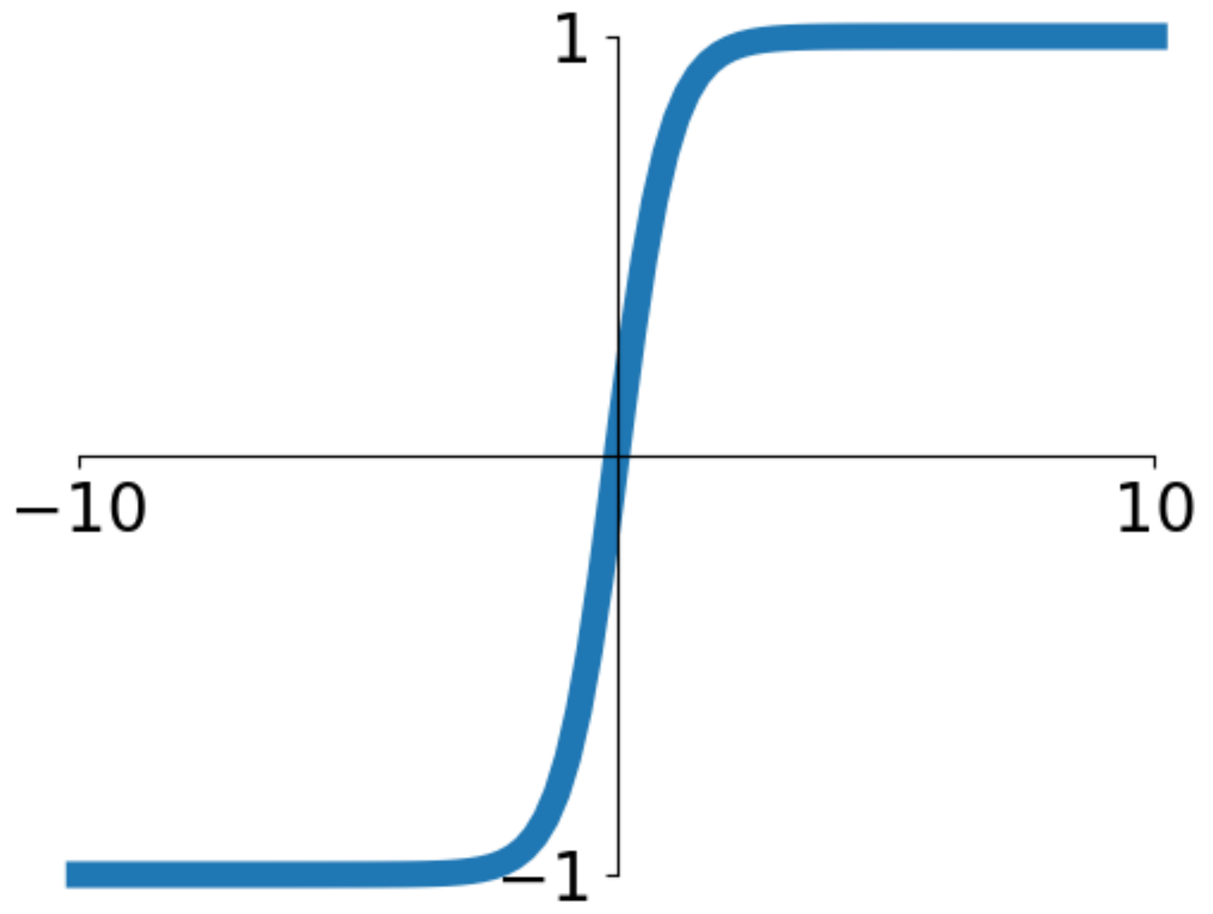
$$a(x) = \mathbf{tanh}(x) = \frac{2}{1 + e^{-2x}} - 1$$

Pros:

- Squishes everything in the range [-1,1]
- Centered around zero
- Has well defined gradient everywhere

Cons:

- Saturated neurons “kill” the gradients



Tanh Activation

* slide adopted from Li, Karpathy, Johnson’s **CS231n at Stanford**

Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

$$\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

In practice, also learn how to scale and offset:

$$y^{(k)} = \gamma^{(k)} \bar{x}^{(k)} + \beta^{(k)}$$

BN layer parameters

Benefit:

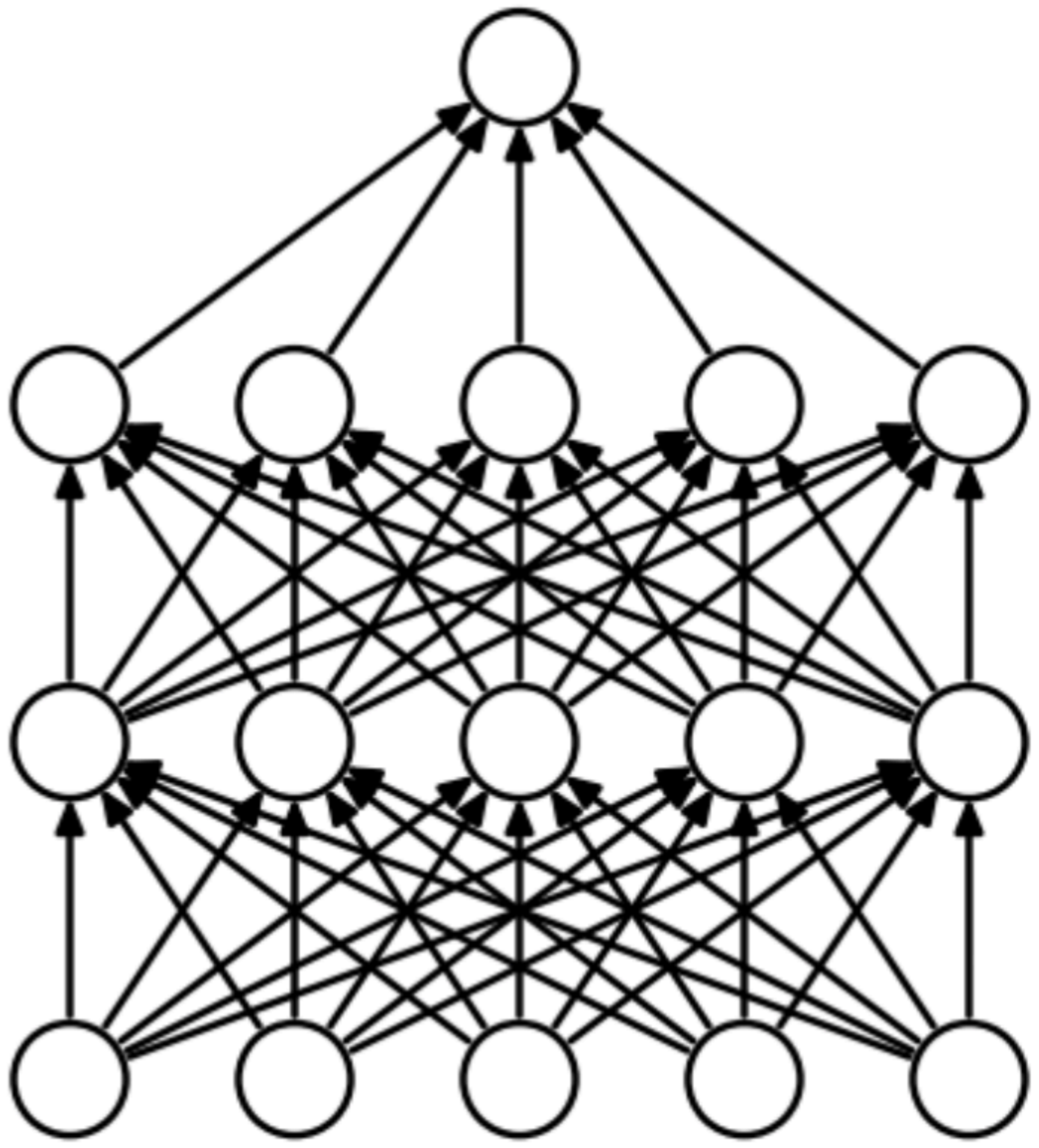
Improves learning (better gradients, higher learning rate)

Typically inserted **before** activation layer

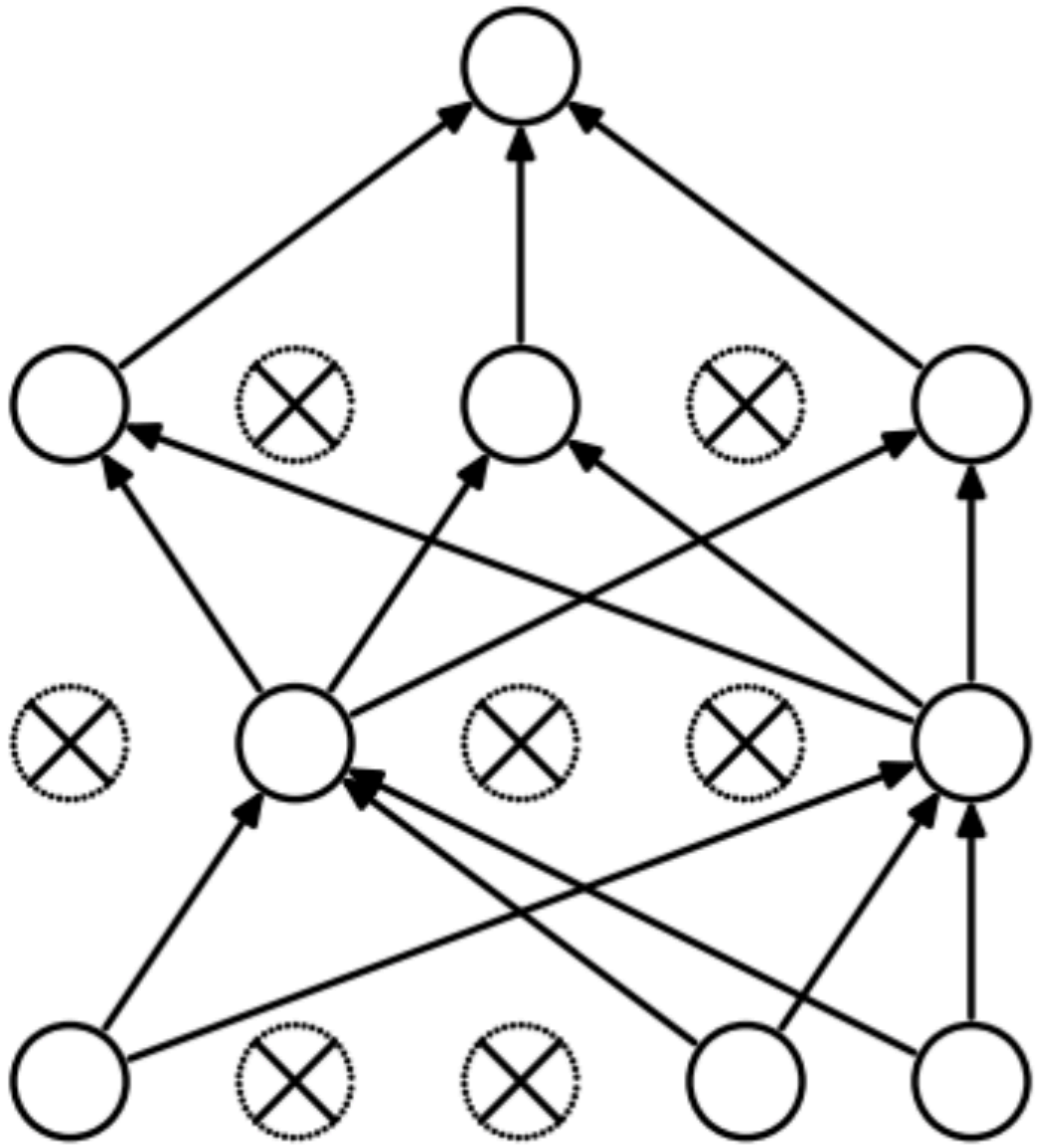
[Ioffe and Szegedy, NIPS 2015]

Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)



Standar Neural Network



After Applying **Dropout**

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

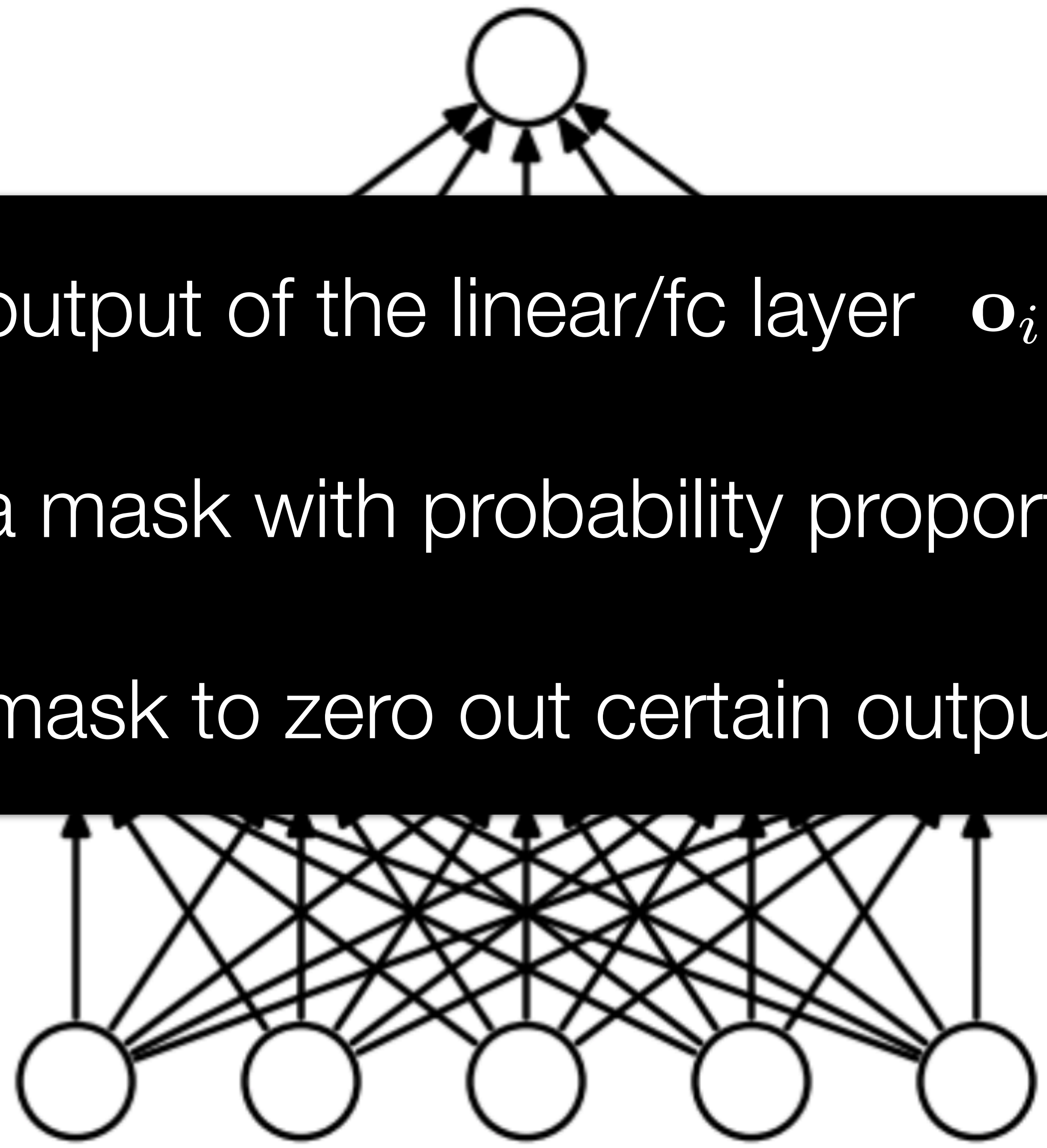
Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)

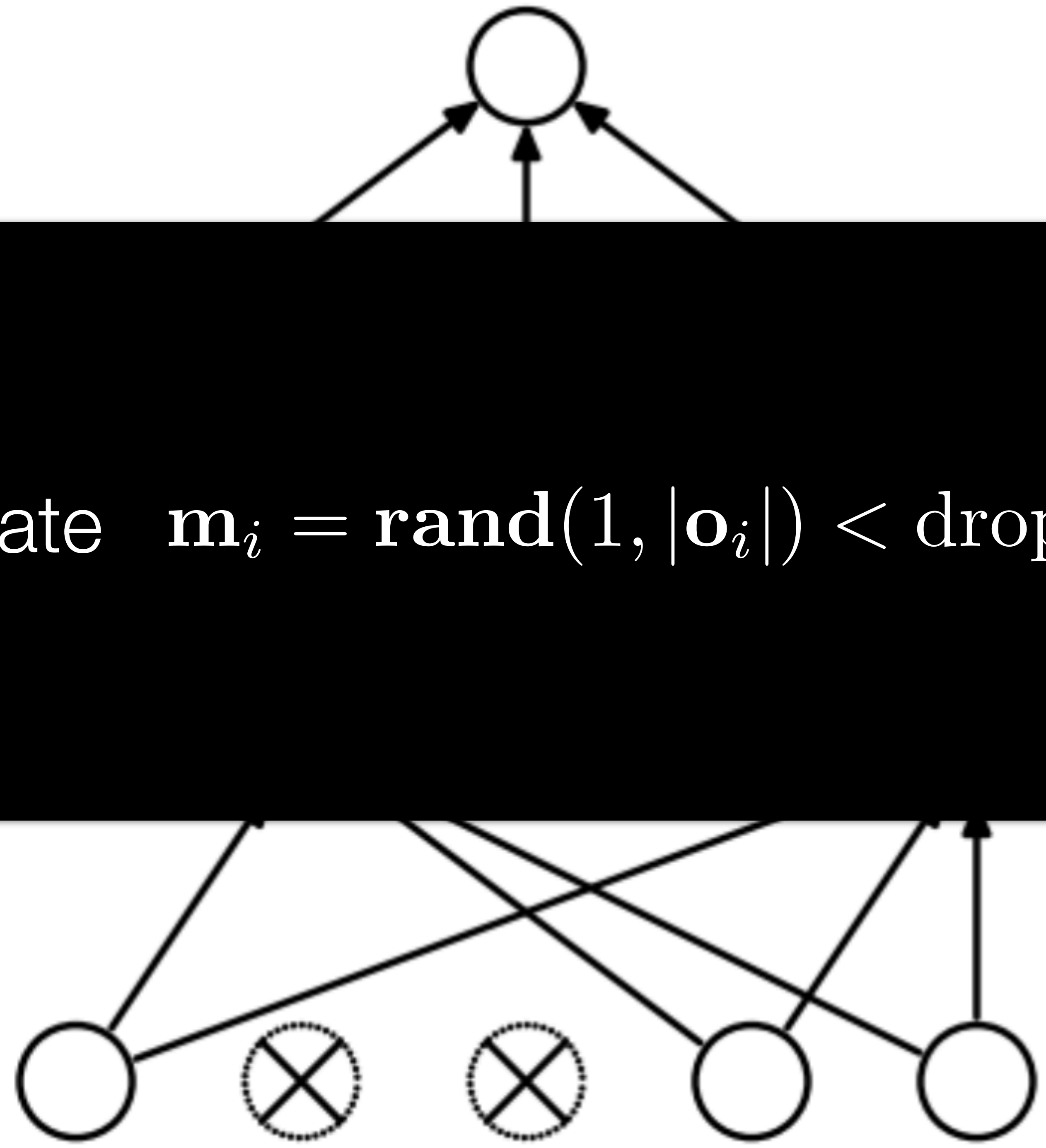
1. Compute output of the linear/fc layer $\mathbf{o}_i = \mathbf{W}_i \cdot \mathbf{x} + \mathbf{b}_i$

2. Compute a mask with probability proportional to dropout rate $\mathbf{m}_i = \mathbf{rand}(1, |\mathbf{o}_i|) < \text{dropout rate}$

3. Apply the mask to zero out certain outputs $\mathbf{o}_i = \mathbf{o}_i \odot \mathbf{m}_i$



Standar Neural Network



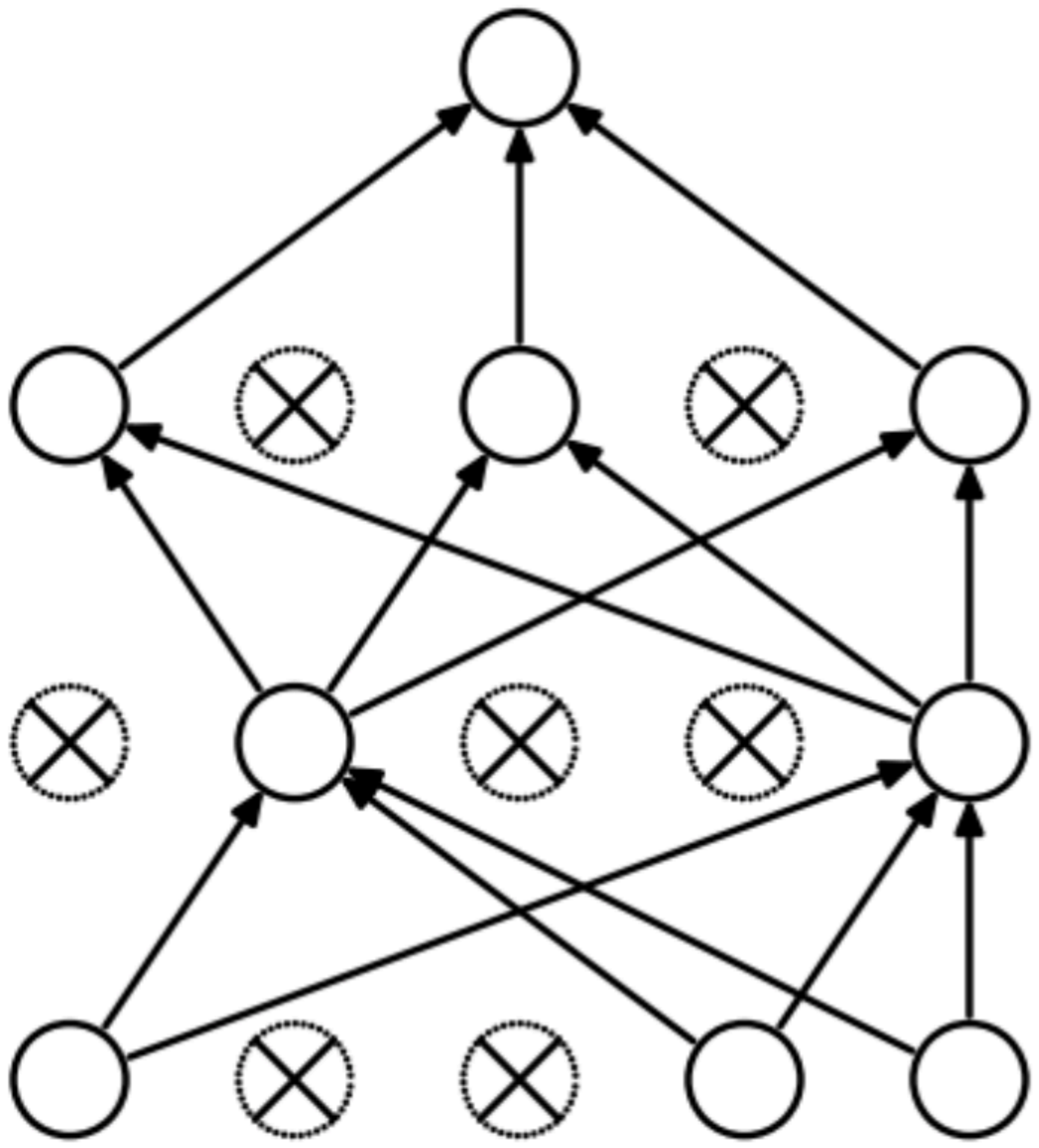
After Applying **Dropout**

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)



After Applying **Dropout**

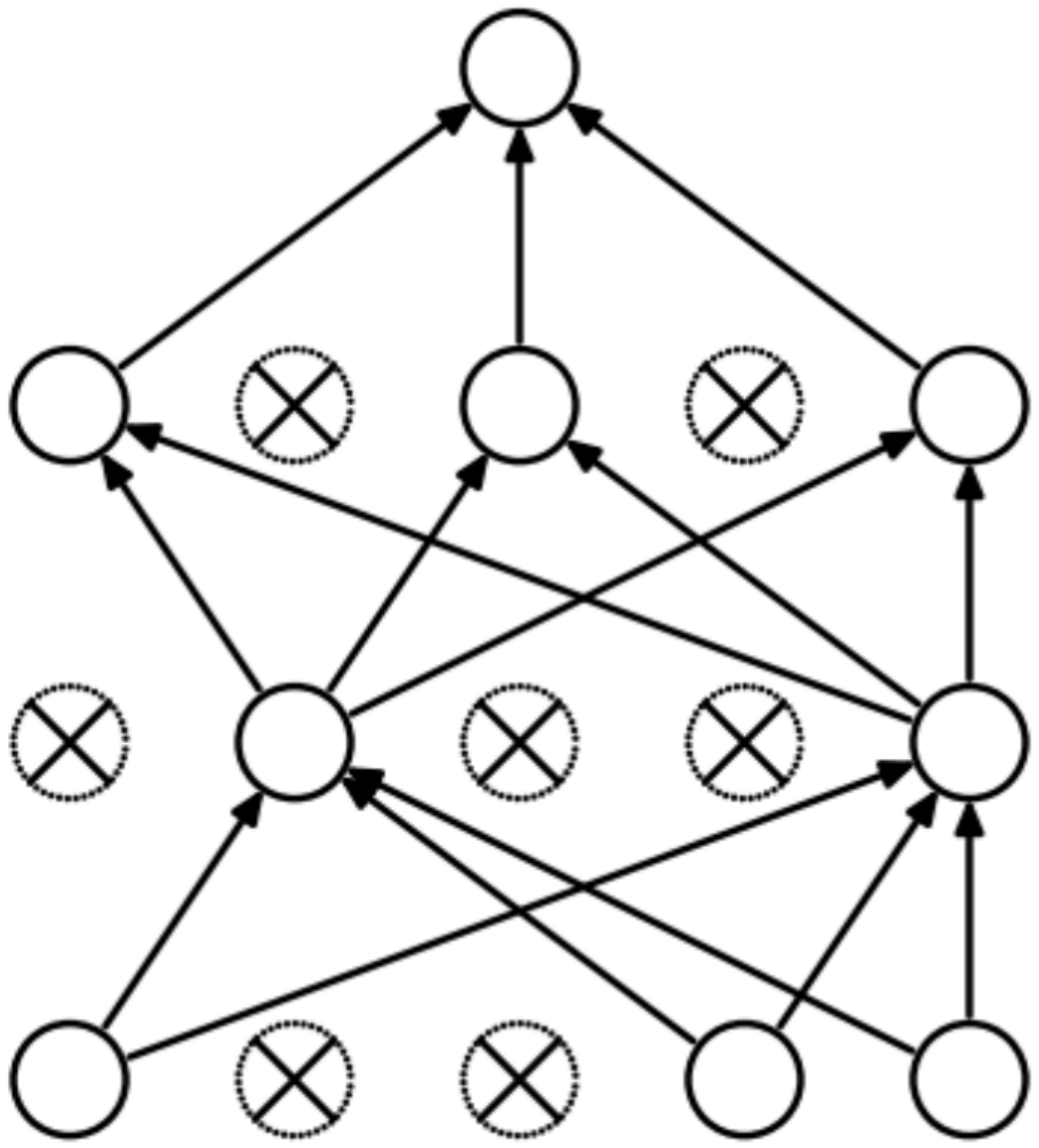
Why is this a good idea?

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)



After Applying **Dropout**

Why is this a good idea?

Dropout is training an **ensemble of models** that share parameters

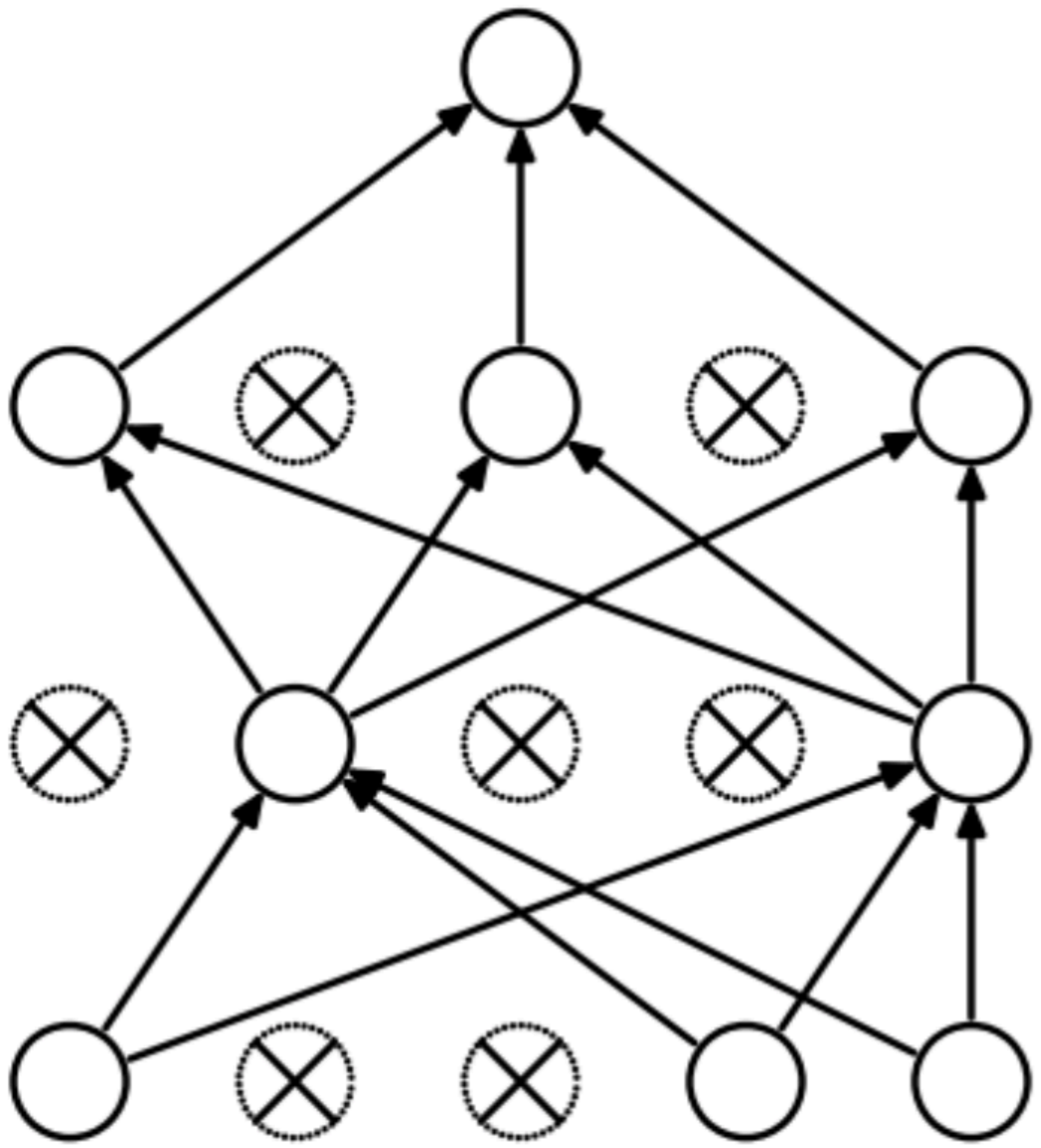
Each binary mask (generated in the forward pass) is one model that is trained on (approximately) one data point

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout (at test time)

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)



After Applying **Dropout**

At test time, **integrate out all the models** in the ensemble

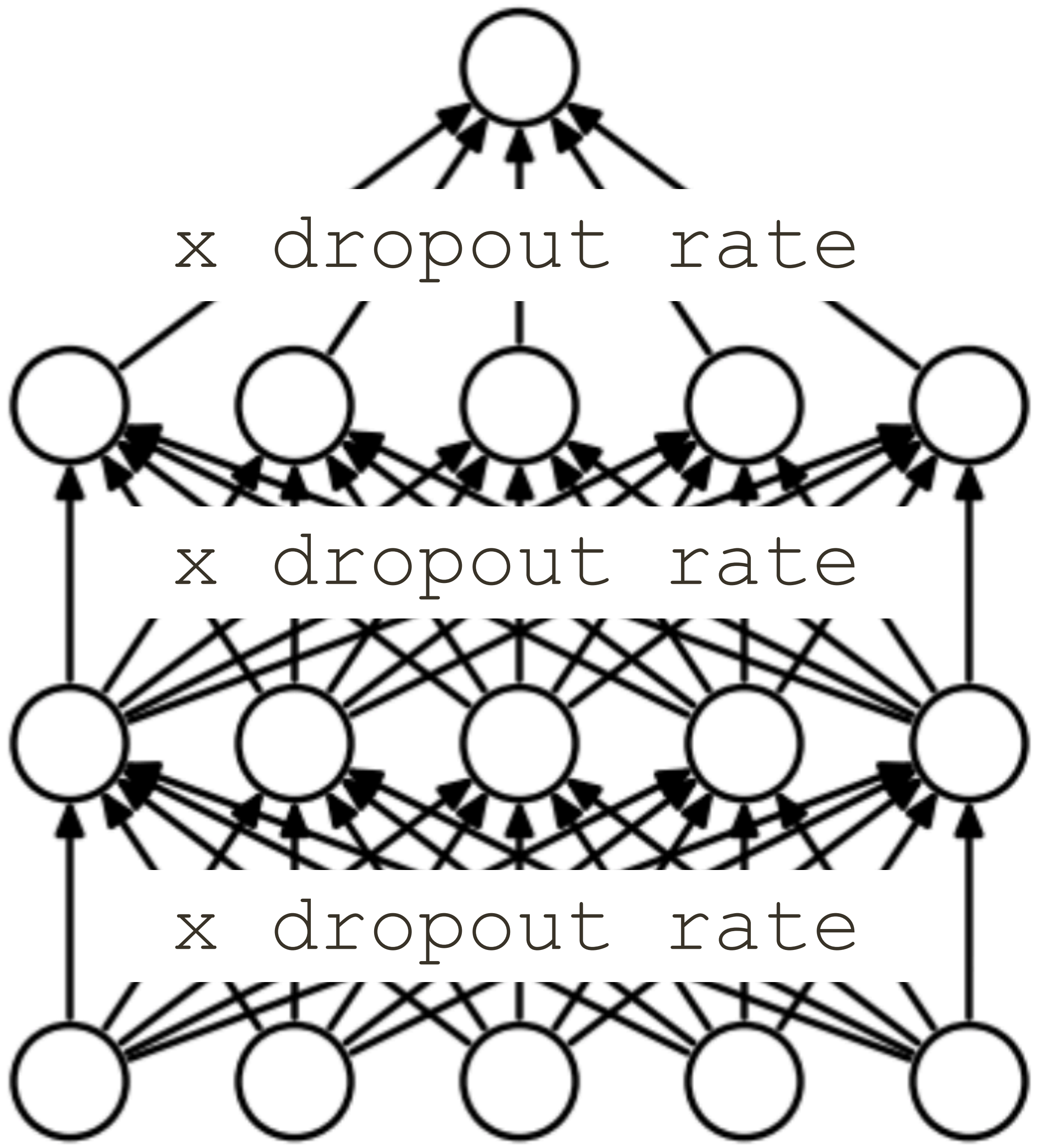
Monte Carlo approximation: many forward passes with different masks and average all predictions

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Regularization: Dropout (at test time)

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)



At test time, **integrate out all the models** in the ensemble

Monte Carlo approximation: many forward passes with different masks and average all predictions

Equivalent to forward pass with all connections on and **scaling of the outputs** by dropout rate

[Srivastava et al, JMLR 2014]

For derivation see Lecture 6 of **CS231n at Stanford**

* adopted from slides of **CS231n at Stanford**

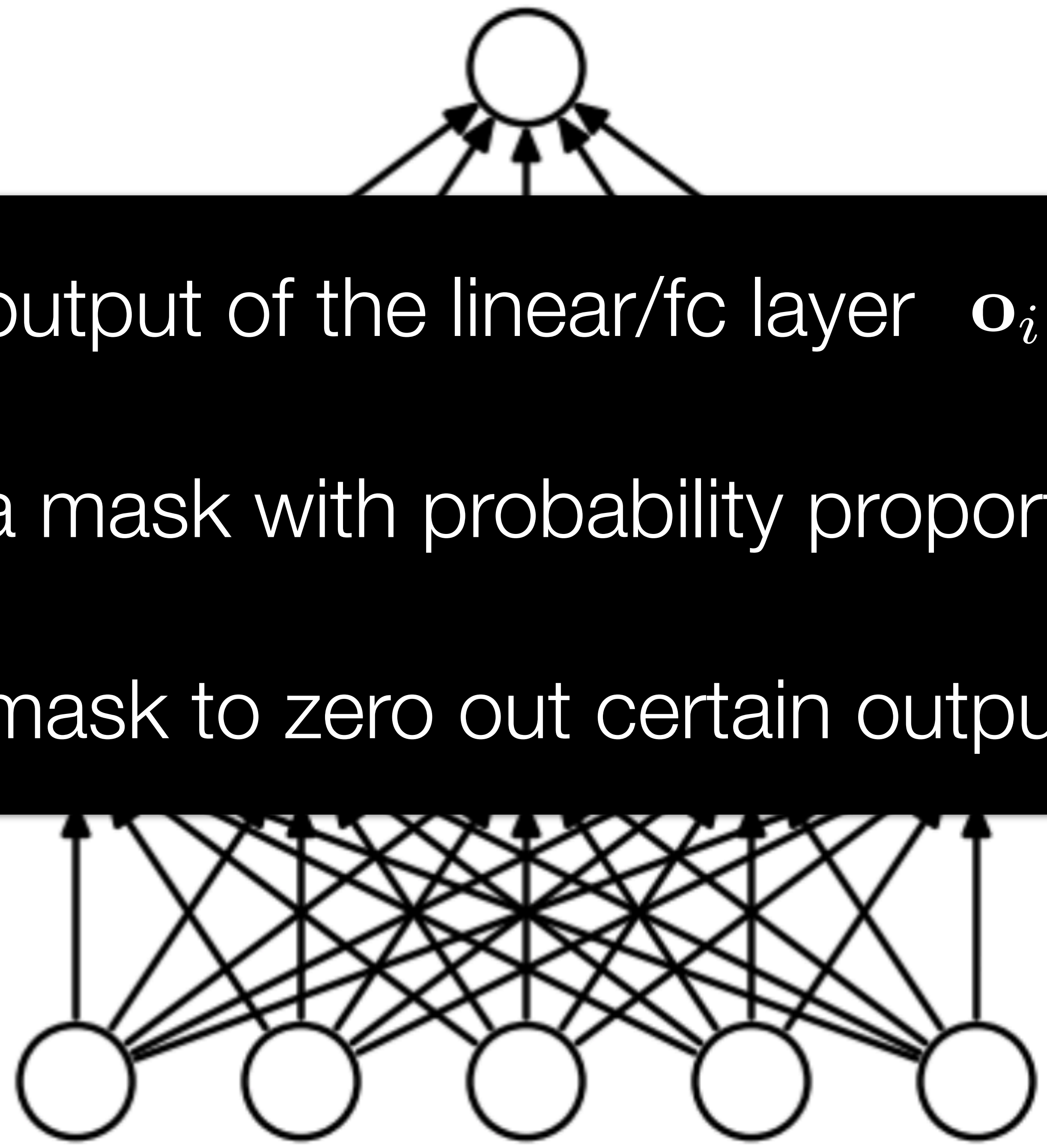
Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)

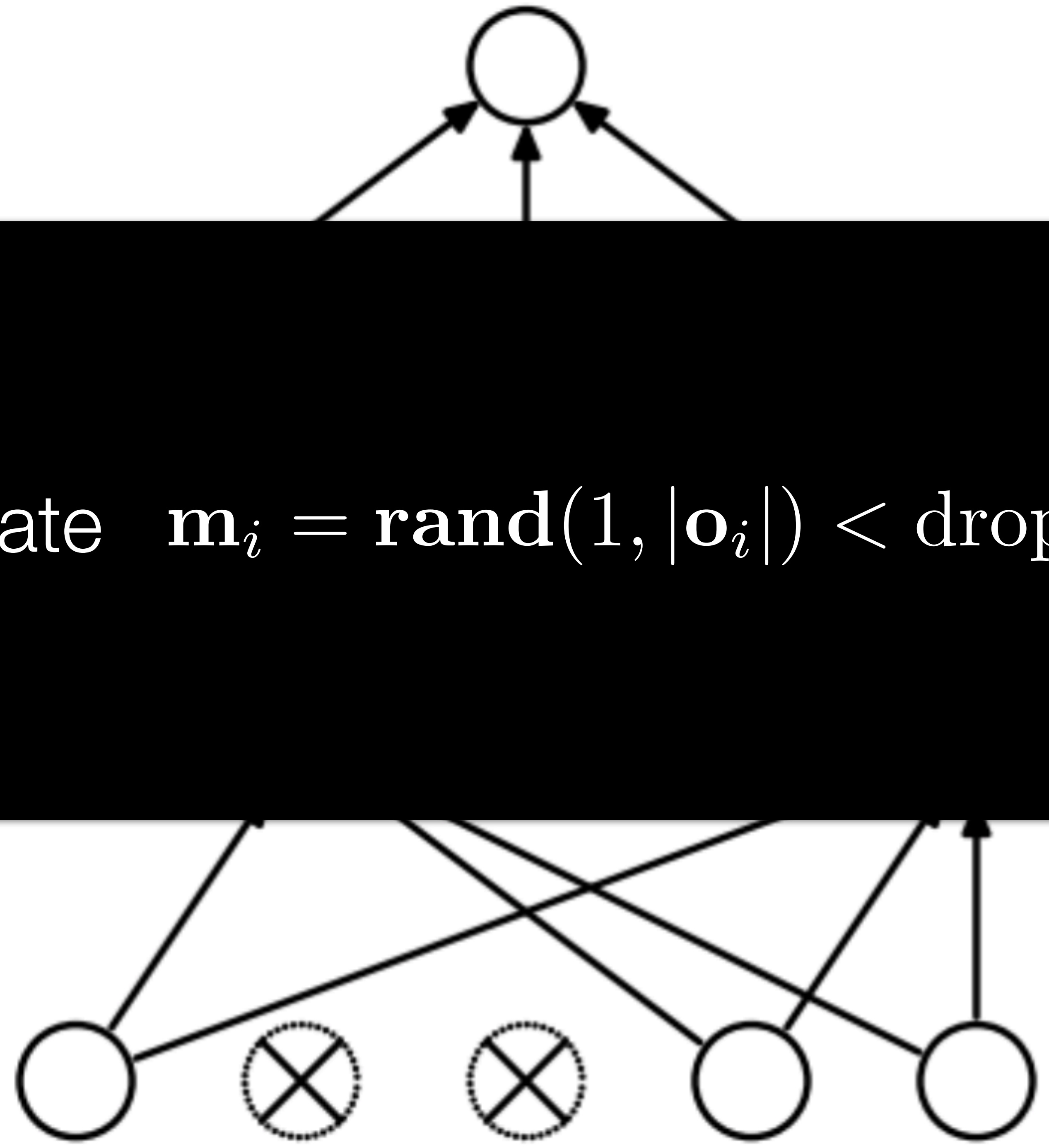
1. Compute output of the linear/fc layer $\mathbf{o}_i = \mathbf{W}_i \cdot \mathbf{x} + \mathbf{b}_i$

2. Compute a mask with probability proportional to dropout rate $\mathbf{m}_i = \mathbf{rand}(1, |\mathbf{o}_i|) < \text{dropout rate}$

3. Apply the mask to zero out certain outputs $\mathbf{o}_i = \mathbf{o}_i \odot \mathbf{m}_i$



Standar Neural Network

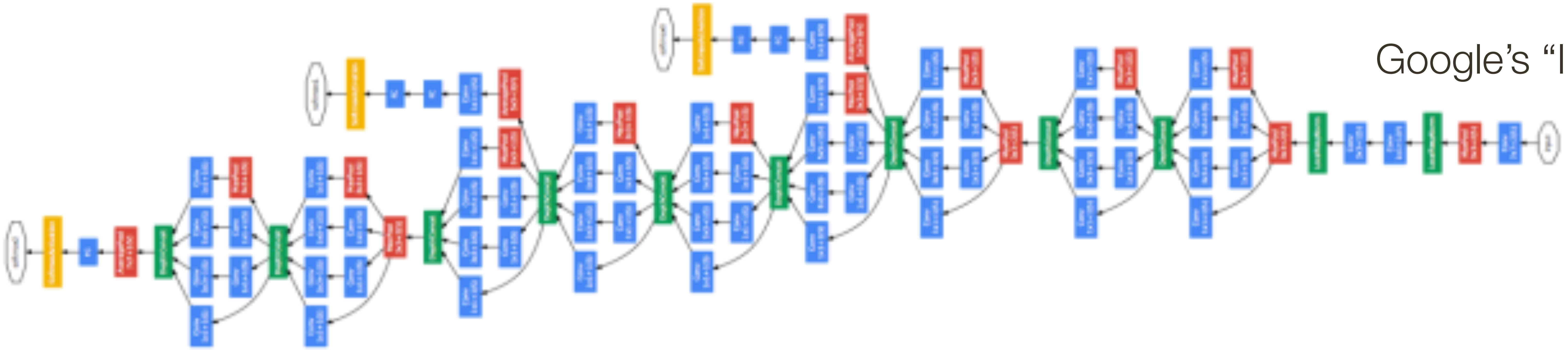


After Applying **Dropout**

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

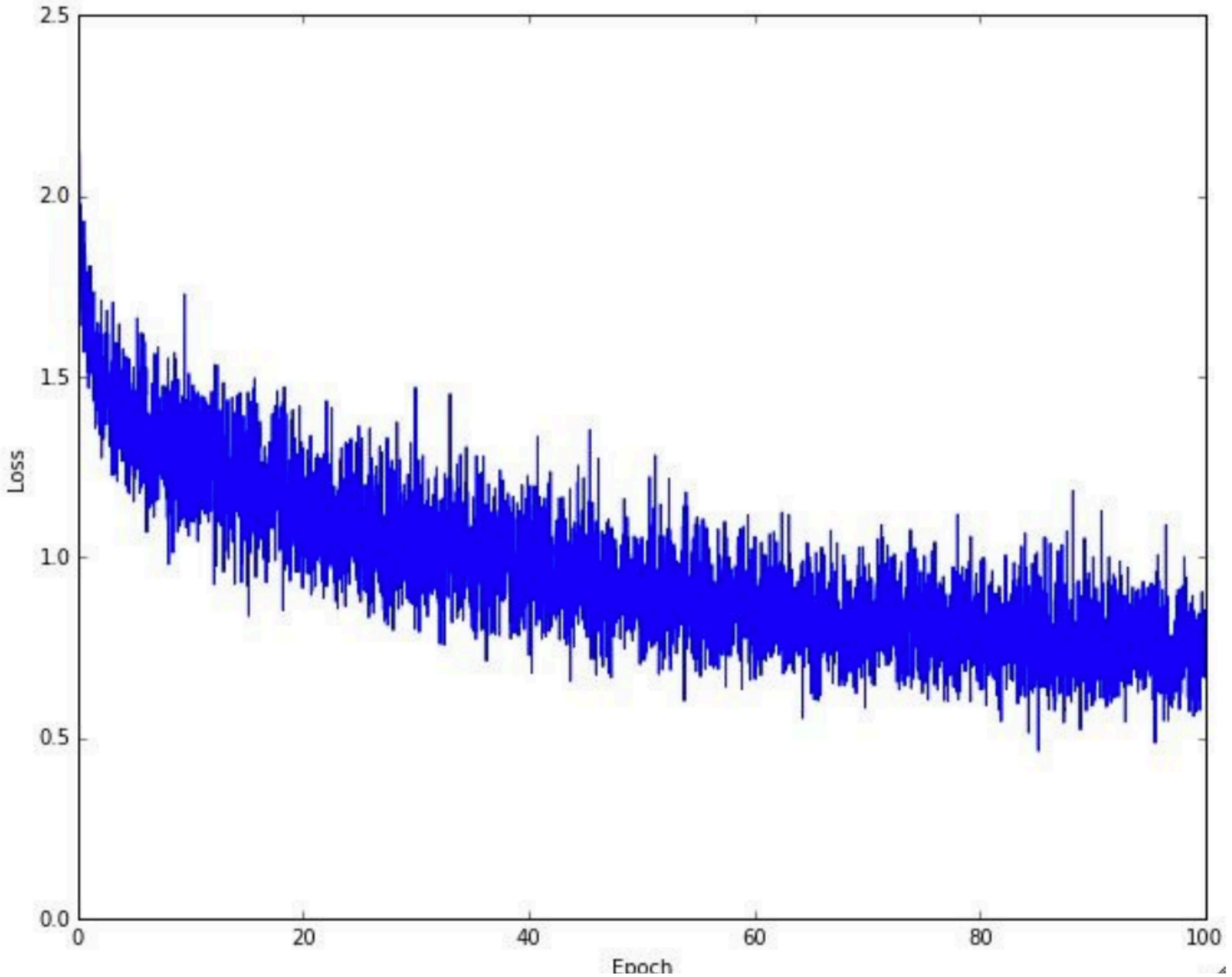
- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

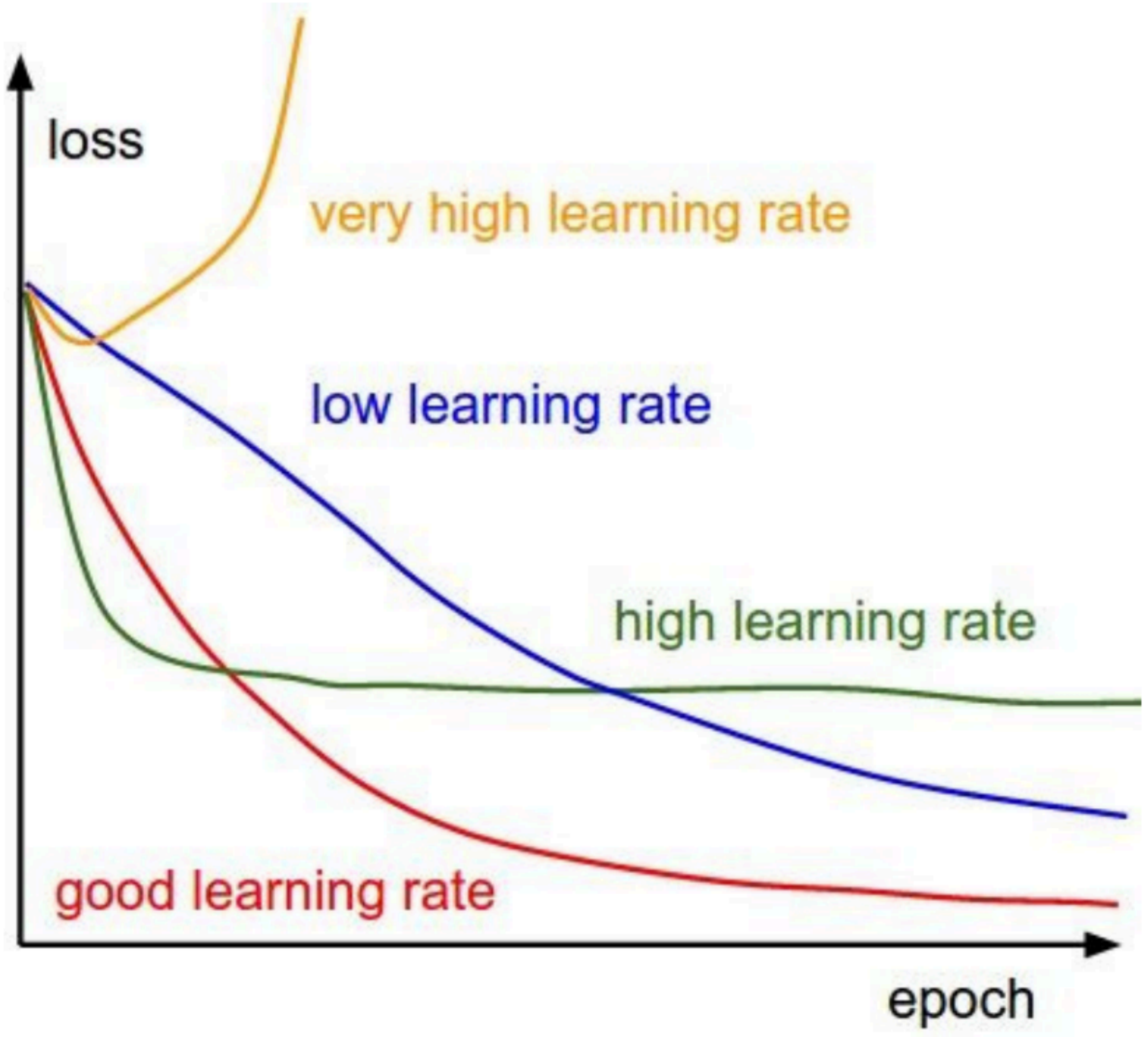
grid search

Loss Functions …

This is where all the **fun** is … but later … too many to cover …

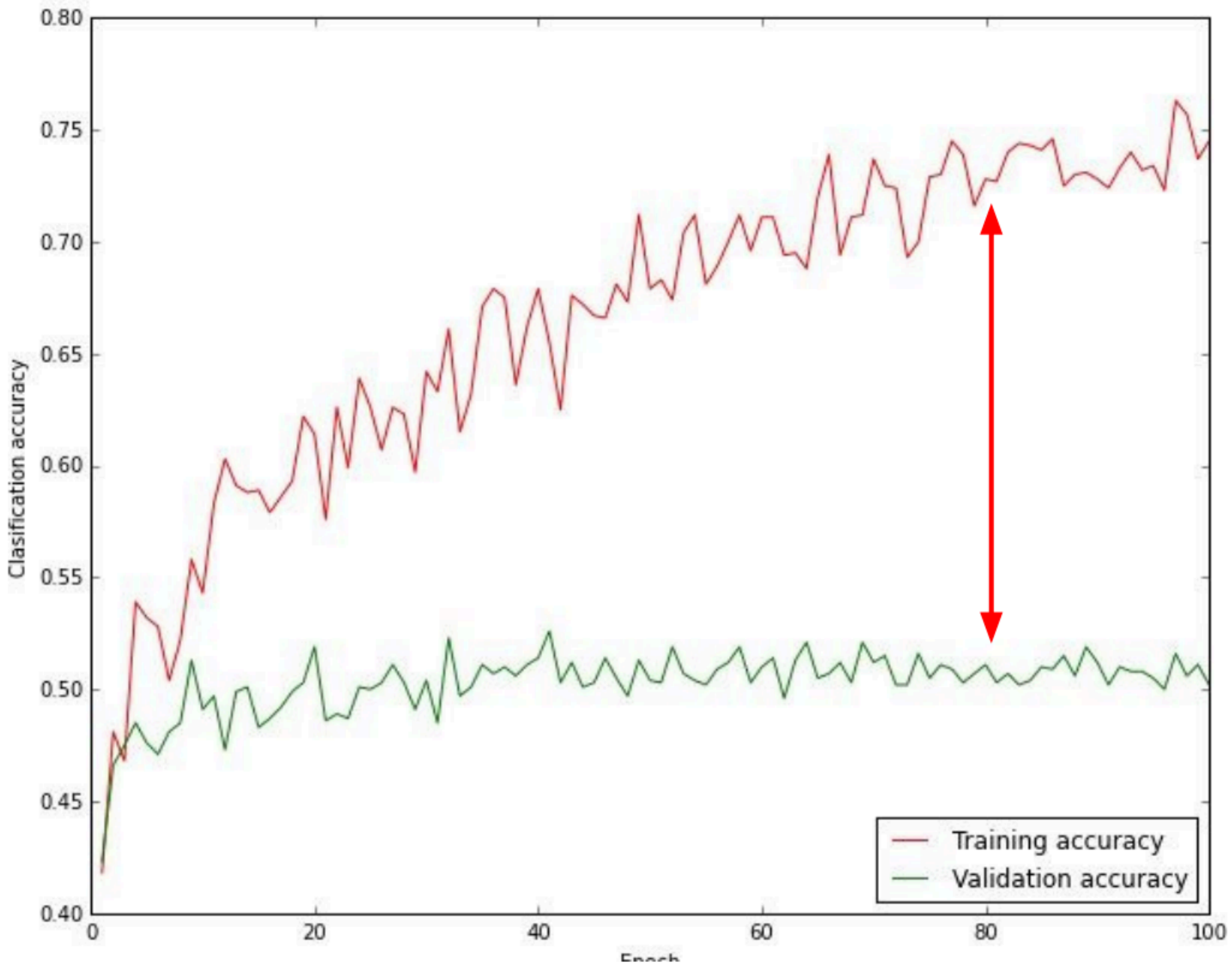
Monitoring Learning: Visualizing the (training) loss





* slide from Li, Karpathy, Johnson's **CS231n at Stanford**

Monitoring Learning: Visualizing the (training) loss



Big gap = overfitting

Solution: increase regularization

No gap = undercutting

Solution: increase model capacity

Small gap = **ideal**

* slide from Li, Karpathy, Johnson's **CS231n at Stanford**