Tutorial: Auto-Rotating Perceptrons (ARP)

Ing. Daniel Saromo Mori

22-03-2021

This work presents an improved design of the perceptron unit for deep neural networks:
The Auto-Rotating Perceptron (ARP) [1].

Outline

Perceptrons: Base units for Feedforward Neural Networks

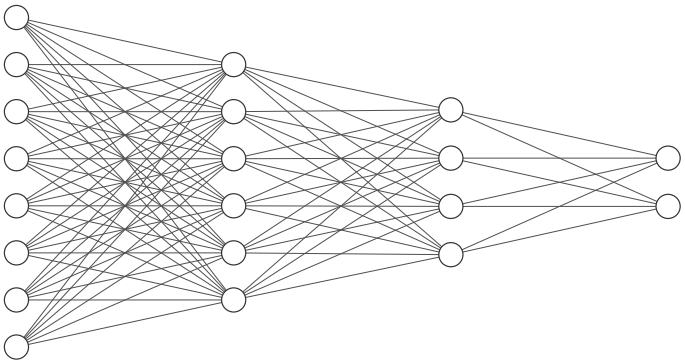


Figure 1: Perceptrons in a FNN.

Perceptrons: Base units for Feedforward Neural Networks

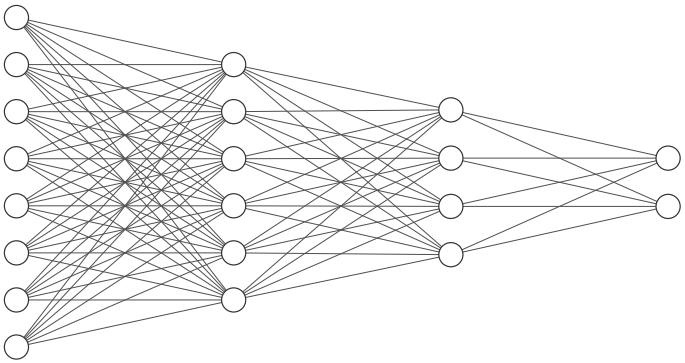
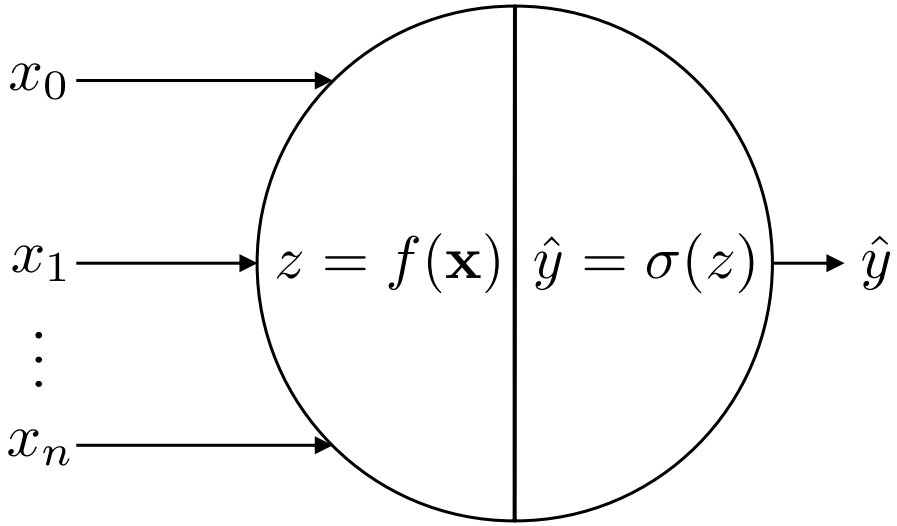


Figure 1: Perceptrons in a FNN.



$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0\, x_0 \qquad x_0 = 1$$

Figure 2: Perceptron structure: **two** phases.

Perceptrons: Base units for Feedforward Neural Networks

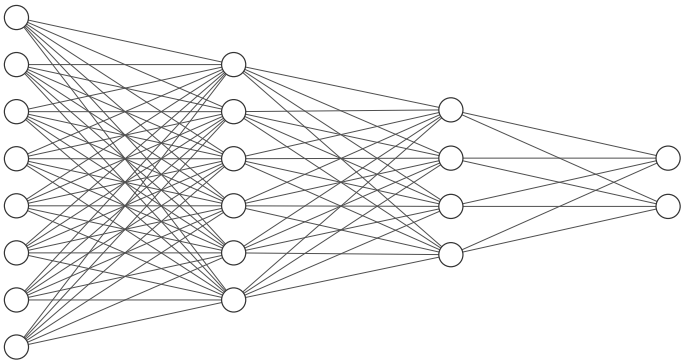
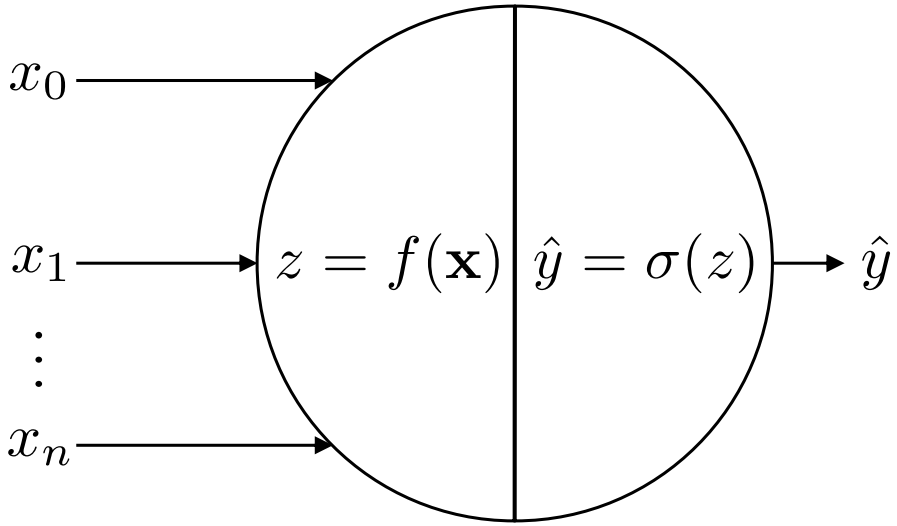


Figure 1: Perceptrons in a FNN.



$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0\, x_0 \qquad x_0 = 1$$

Figure 2: Perceptron structure: **two** phases.

- Each perceptron maps an input vector $\mathbf{x} \in \mathbb{R}^n$ to a scalar value $\hat{y}$, after passing through z.
- $\mathbf{x} \rightarrow$ (Linear Transformation) $\rightarrow z \rightarrow$ (Non-linear Activation) $\rightarrow \hat{y}$
- Neurons as hierarchical feature extractors: Layered arrangement.

Outline

Vanishing Gradient Problem (VGP)

- By stacking more layers, the neural network can learn more complex patterns from the input data.
- VGP appears when training deep multilayer perceptrons (MLP) with **saturated activation functions** (e.g., **sigmoid**).
- In a feedforward neural network, layers closer to the input learn slower than those near the output [2].
  - Reason: Exponential **decrease of the error gradients** as we get closer to the input layers during backpropagation learning.
- Error gradient depends on the **derivative of the activation function** $\sigma(z)$.

- **TLDR**: DNN's classic perceptrons suffer from VGP when $\sigma(z)$ is bounded.

Outline

Methodology — Dynamic region of the neurons

- e.g., $\sigma(z)$: Unipolar sigmoid. If $\sigma'(z) \approx 0 \rightarrow$ Unwanted node saturation.
- Nodes need to be in their **dynamic region** $\mathcal{L}$. ARP let us control that.

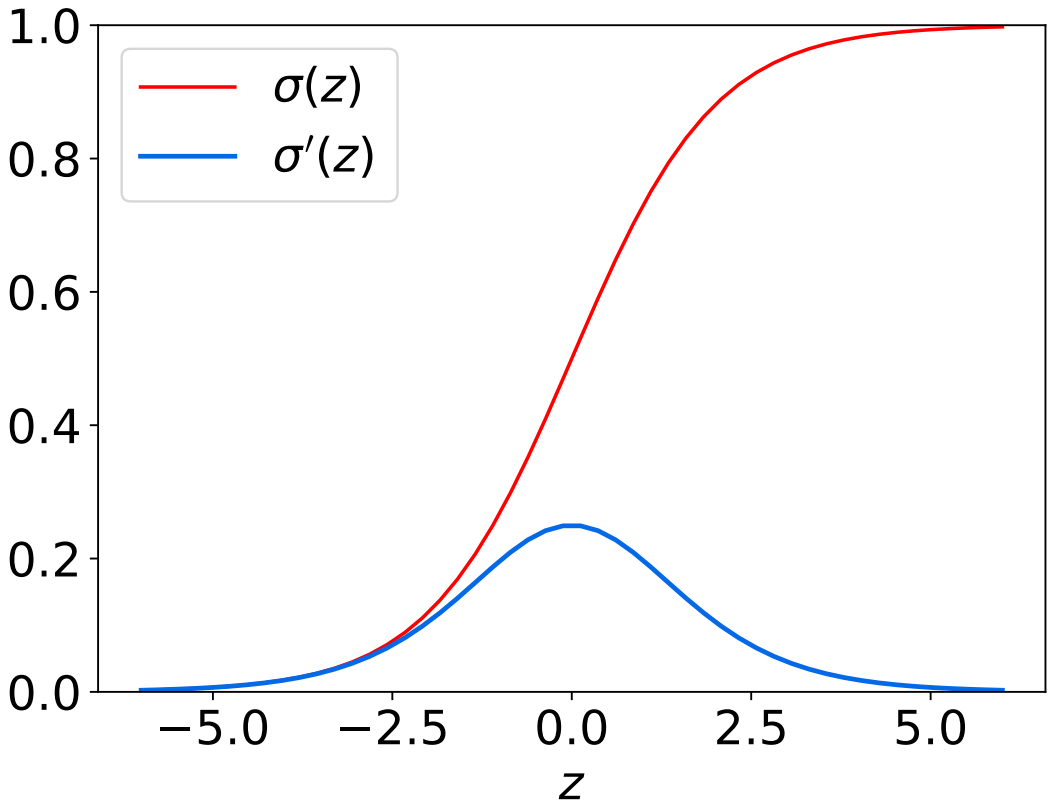


Figure 3: Sigmoid activation function $\sigma(z)$ and its derivative.

Methodology — Dynamic region of the neurons

- e.g., $\sigma(z)$: Unipolar sigmoid. If $\sigma'(z) \approx 0 \rightarrow$ Unwanted node saturation.
- Nodes need to be in their **dynamic region $\mathcal{L}$**. ARP let us control that.

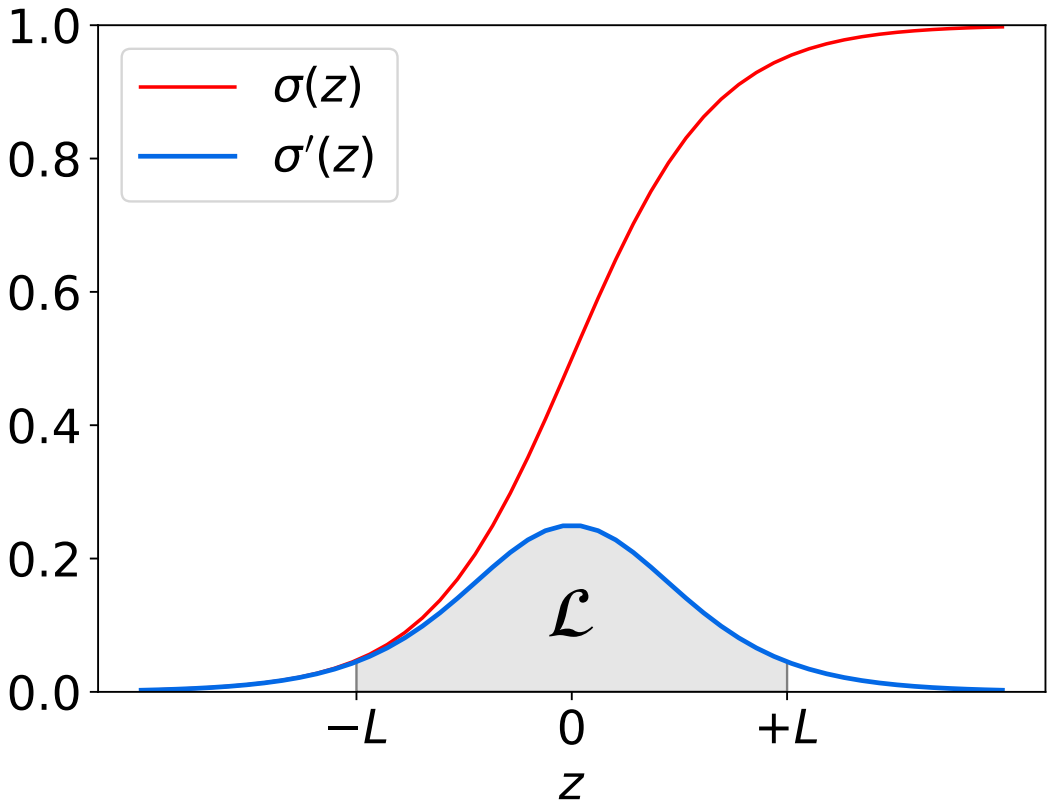


Figure 3: Sigmoid activation function $\sigma(z)$ and its derivative.

Outline

How to avoid node saturation?

- Classic solution: Change the activation function (e.g., ReLU).

How to avoid node saturation?

- Classic solution: Change the activation function (e.g., ReLU).
- Our solution:
  Modify the pre-activation mechanism of the perceptron.

How to avoid node saturation?

- Classic solution: Change the activation function (e.g., ReLU).
- Our solution:

  Modify the pre-activation mechanism of the perceptron. How?

How to avoid node saturation?

- Classic solution: Change the activation function (e.g., ReLU).
- Our solution:

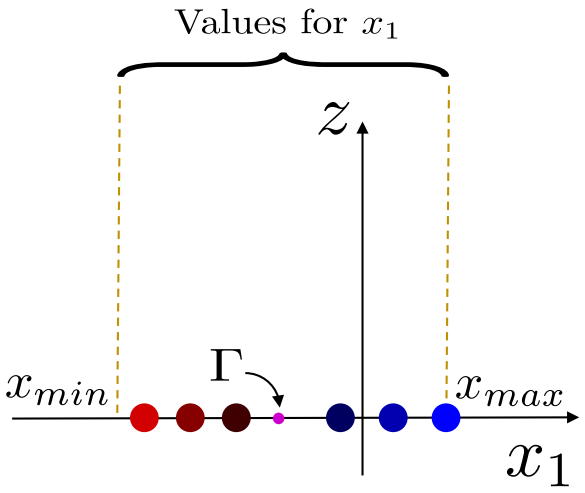
  Modify the pre-activation mechanism of the perceptron.
  How? Using Auto-Rotating Perceptrons (ARP).

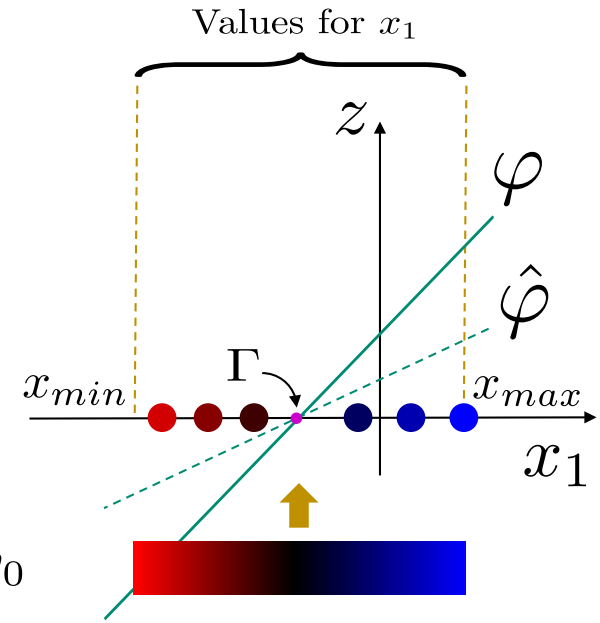
Outline

Graphical interpretation of the pre-activation phase (at 1D)

Consider: $x_i \in [x_{min}, x_{max}]$, $n = 1$, $\mathbf{x} = x_1$, $\langle \mathbf{x} \rangle \subset \mathbb{R}^n$ and $z \perp x_i$.

Graph: Some input vectors $\mathbf{x}$ VS axis z.





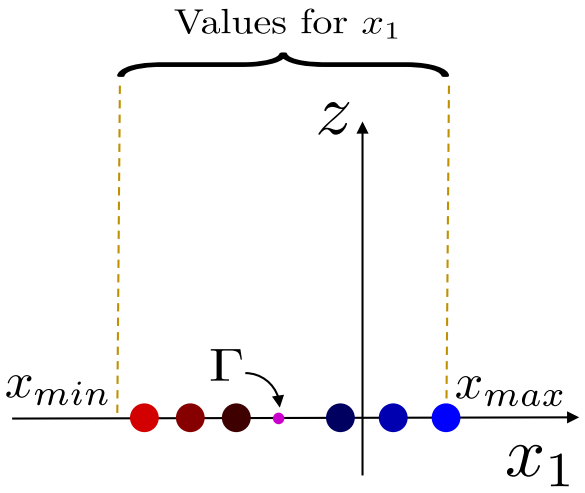
At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

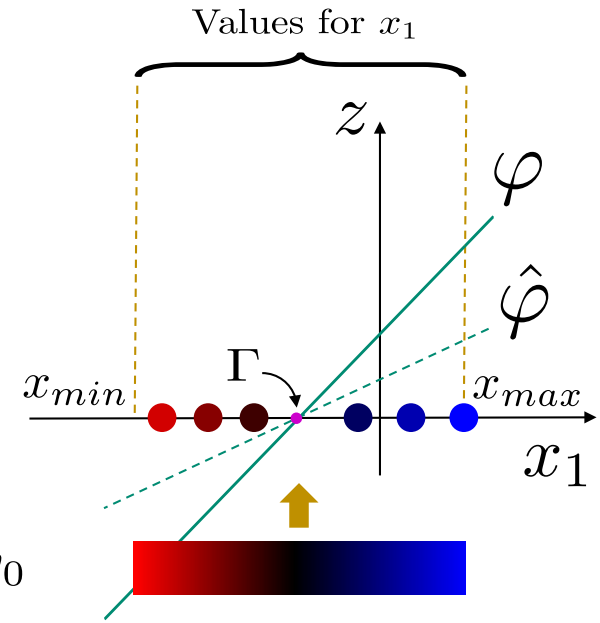
Literature says that each perceptron:

Graphical interpretation of the pre-activation phase (at 1D)

Consider: $x_i \in [x_{min}, x_{max}]$, $n = 1$, $\mathbf{x} = x_1$, $\langle \mathbf{x} \rangle \subset \mathbb{R}^n$ and $z \perp x_i$.

Graph: Some input vectors $\mathbf{x}$ VS axis z.





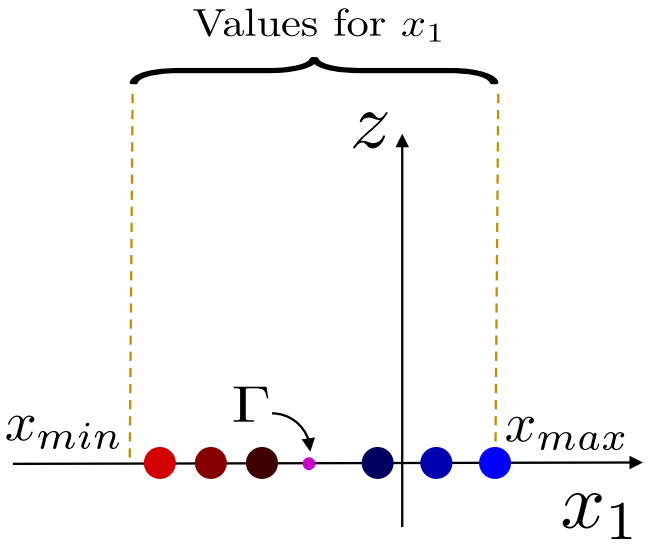
At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

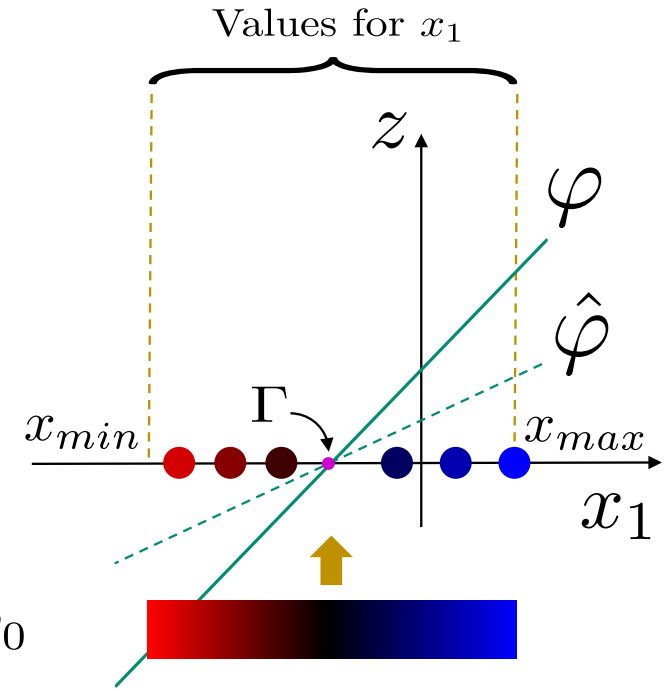
Literature says that each perceptron:

1. Extracts a *dual* abstract feature from the input $\mathbf{x} \in \mathbb{R}^n$.
2. Creates a new *feature axis* z orthogonal to the n original ones.
3. Weight adjustment finds the boundary Γ that best divides $\mathbb{R}^n$ (□/⌀/■).
4. $\Gamma \subset \mathbb{R}^n$ is used to define φ. $\Gamma = \langle \mathbf{x}, 0 \rangle \cap \varphi$.
5. The hyperplane $\varphi = \langle \mathbf{x}, f(\mathbf{x}) \rangle \subset \mathbb{R}^{n+1}$ divides the augmented space.

Graphical interpretation of the pre-activation phase (at 1D)

- $\mathbf{w} \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ generate the n-dimensional hyperplane φ.



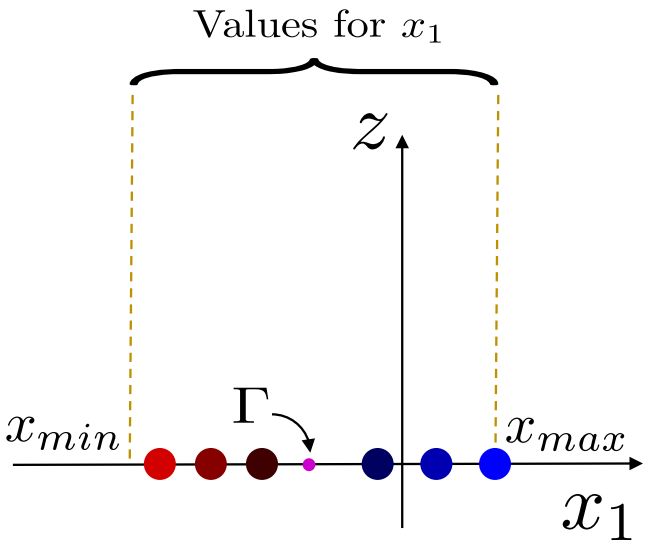


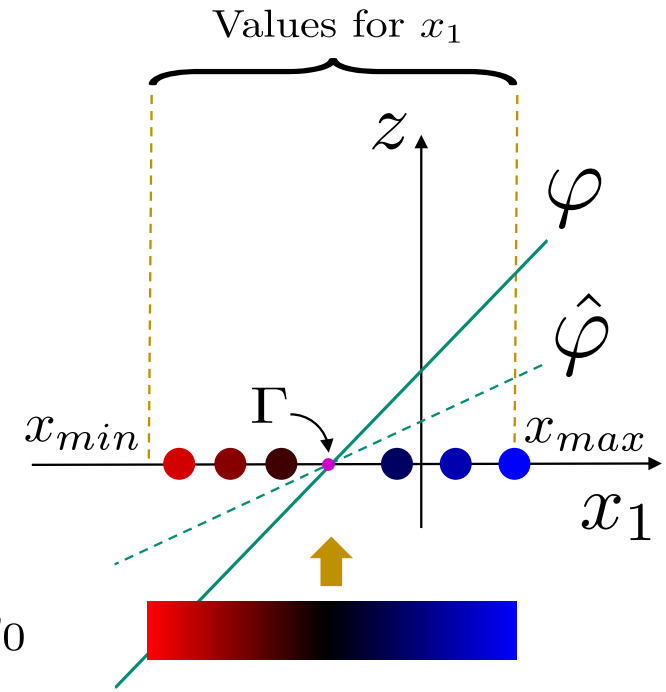
At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

Individual perceptrons create $\varphi = \langle \mathbf{x}, f(\mathbf{x}) \rangle \subset \mathbb{R}^{n+1}$:

Graphical interpretation of the pre-activation phase (at 1D)

- $\mathbf{w} \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ generate the n-dimensional hyperplane φ.





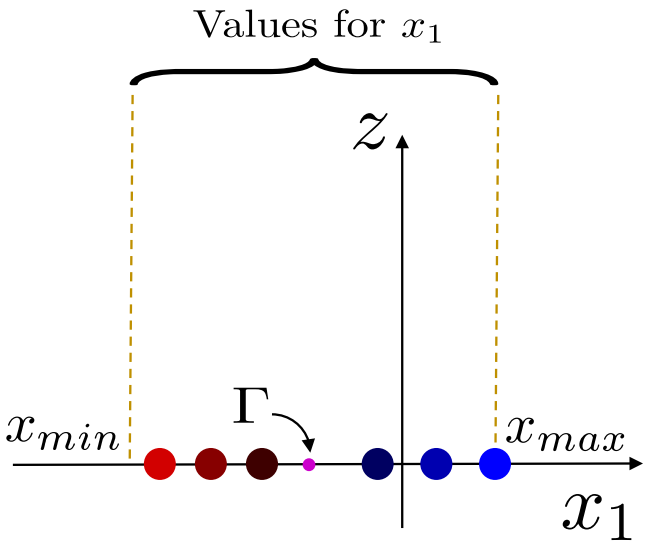
At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

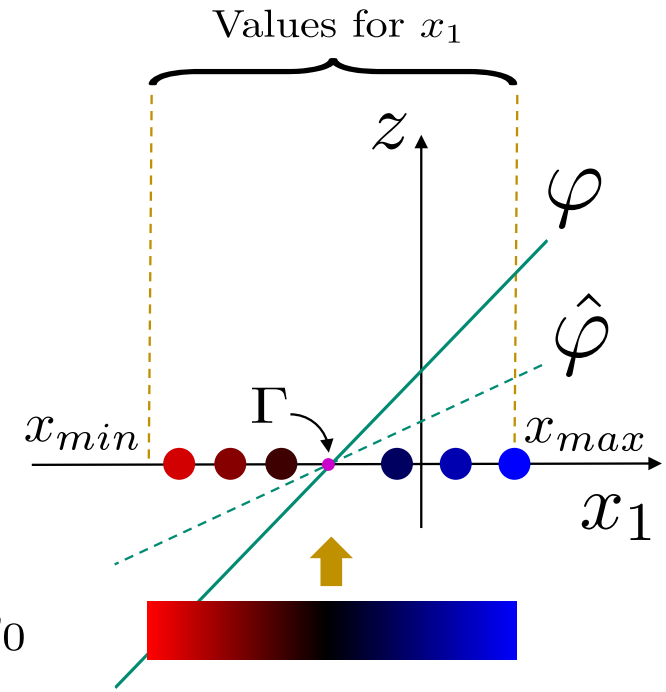
Individual perceptrons create $\varphi = \langle \mathbf{x}, f(\mathbf{x}) \rangle \subset \mathbb{R}^{n+1}$:

- Hierarchical feature extractors.
- Dual abstract feature z.
- When $z = 0 : \Gamma$. $\quad \Gamma \supset \varphi$.

Graphical interpretation of the pre-activation phase (at 1D)

- $\mathbf{w} \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ generate the n-dimensional hyperplane φ.





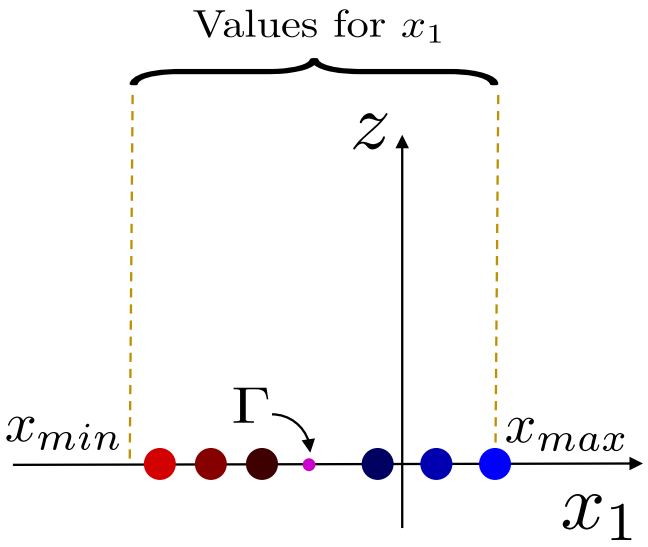
At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

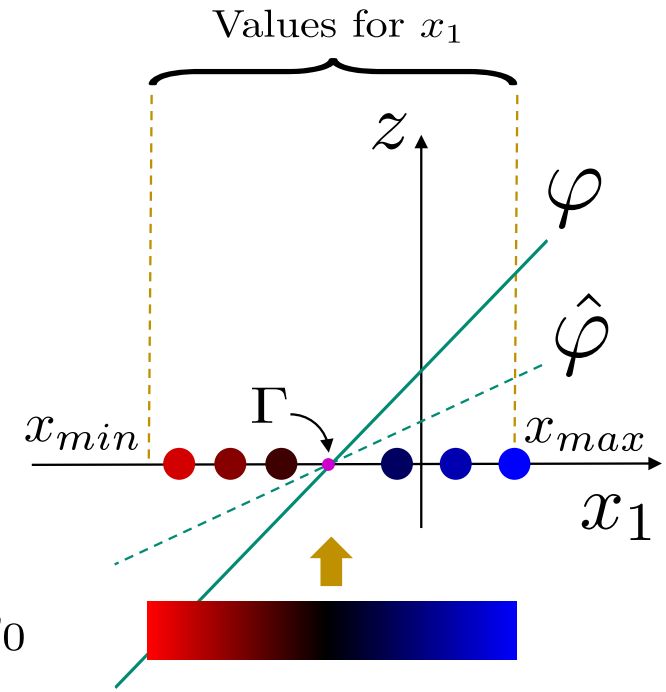
Individual perceptrons create $\varphi = \langle \mathbf{x}, f(\mathbf{x}) \rangle \subset \mathbb{R}^{n+1}$:

- Hierarchical feature extractors. $\Rightarrow \varphi$ is *unique*?
- Dual abstract feature z.
- When $z = 0 : \Gamma$. $\Gamma \supset \varphi$.

Graphical interpretation of the pre-activation phase (at 1D)

- $\mathbf{w} \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ generate the n-dimensional hyperplane φ.





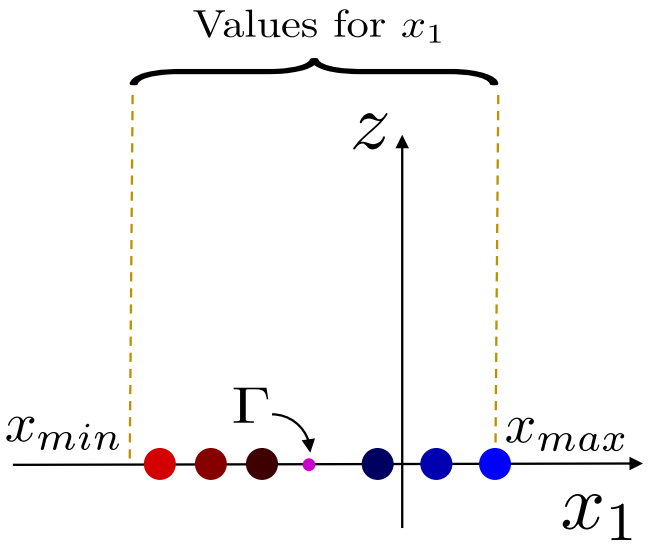
At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

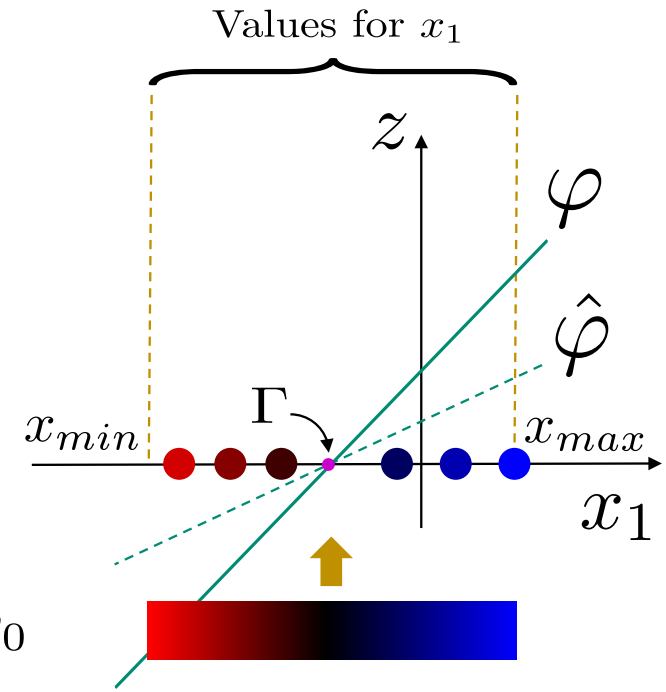
Individual perceptrons create $\varphi = \langle \mathbf{x}, f(\mathbf{x}) \rangle \subset \mathbb{R}^{n+1}$:

- Hierarchical feature extractors. $\Rightarrow \varphi$ is *unique*? Look at $\hat{\varphi}$.
- Dual abstract feature z.
- When $z = 0 : \Gamma$. $\Gamma \supset \varphi$.

Graphical interpretation of the pre-activation phase (at 1D)

- $\mathbf{w} \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ generate the n-dimensional hyperplane φ.





At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

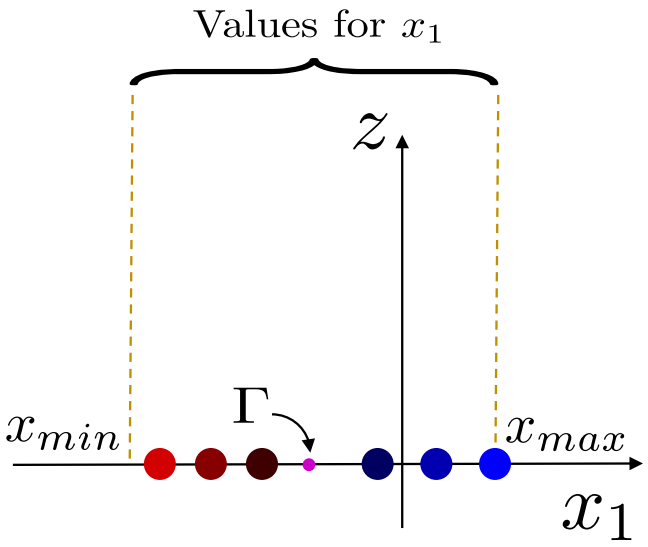
Individual perceptrons create $\varphi = \langle \mathbf{x}, f(\mathbf{x}) \rangle \subset \mathbb{R}^{n+1}$:

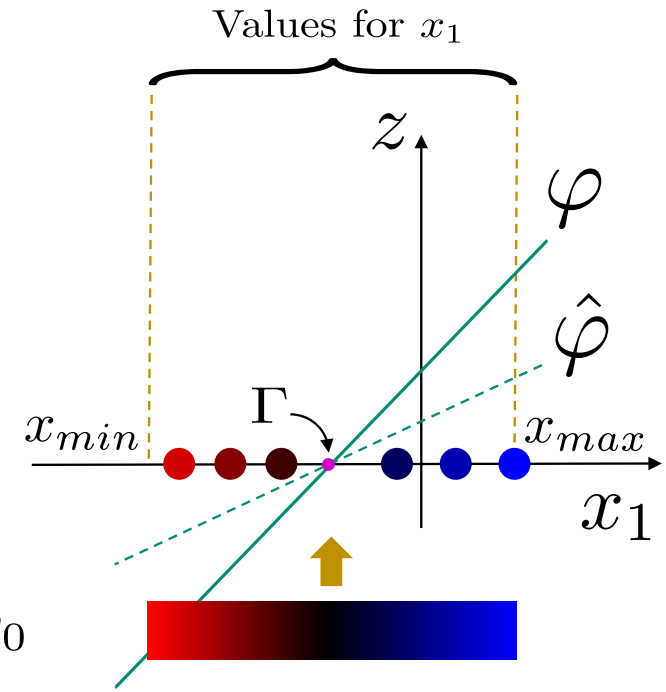
- Hierarchical feature extractors.
- Dual abstract feature z.
- When $z = 0 : \Gamma$. $\quad \Gamma \supset \varphi$.

$\Rightarrow \varphi$ is *unique*? Look at $\hat{\varphi}$.
No!

Graphical interpretation of the pre-activation phase (at 1D)

- $\mathbf{w} \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ generate the n-dimensional hyperplane φ.





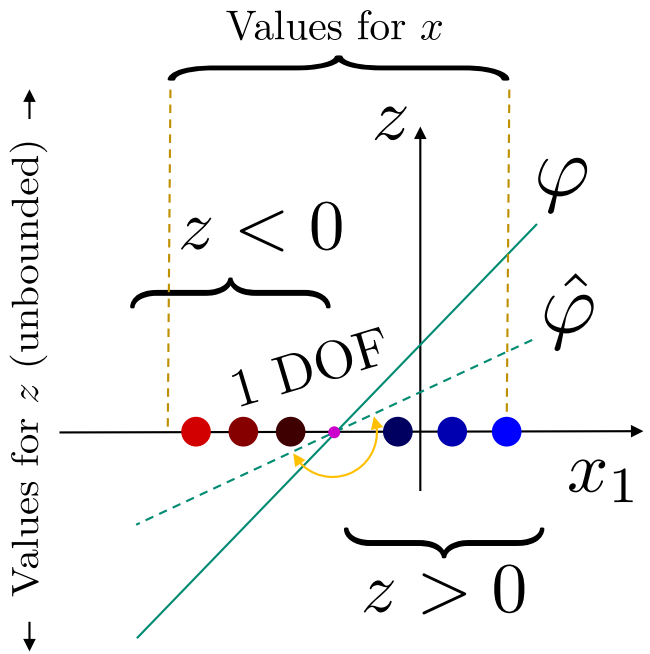
At 1D: $z_\varphi = f(\mathbf{x}) = w_1 x_1 + w_0$

Individual perceptrons create $\varphi = \langle \mathbf{x}, f(\mathbf{x}) \rangle \subset \mathbb{R}^{n+1}$:

- Hierarchical feature extractors.
- Dual abstract feature z.
- When $z = 0 : \Gamma$. $\quad \Gamma \supset \varphi$.

$\Rightarrow \varphi$ is *unique*? Look at $\hat{\varphi}$.
No! : Classic perceptrons find one of many possible *rotated* hyperplanes.

Graphical interpretation of the pre-activation phase (at 1D)



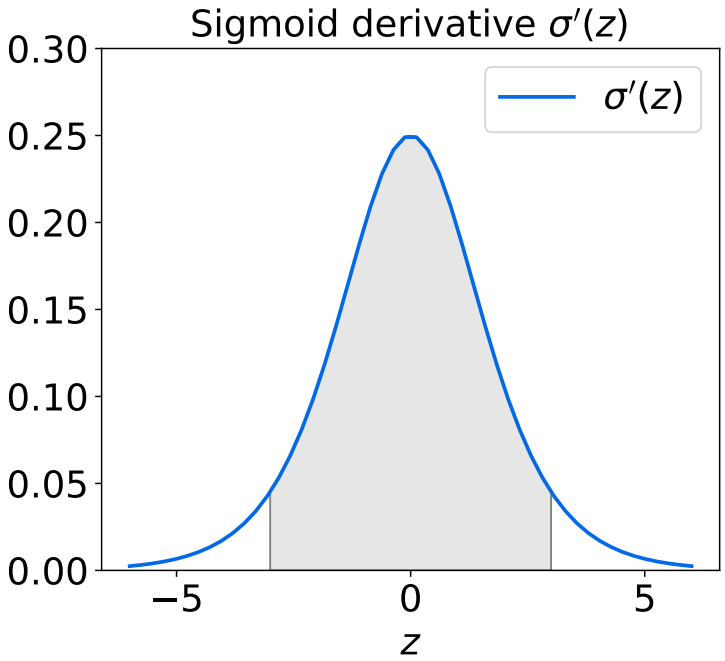
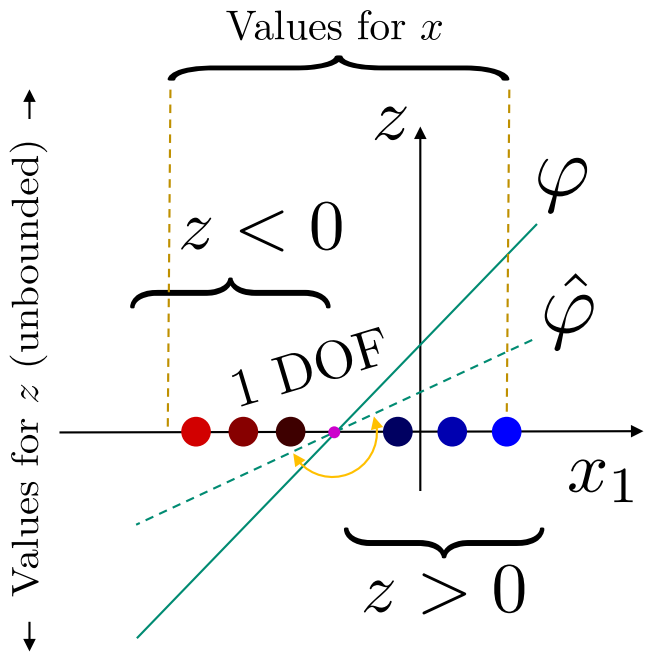


Figure 4: **DOF** at classic perceptrons. Figure 5: Dynamic region: $z \in [-L, +L]$.

- z is unbounded. We need to avoid node saturation.
- Notice: Rotation axis Γ holds inference structure of the perceptron.

Graphical interpretation of the pre-activation phase (at 1D)



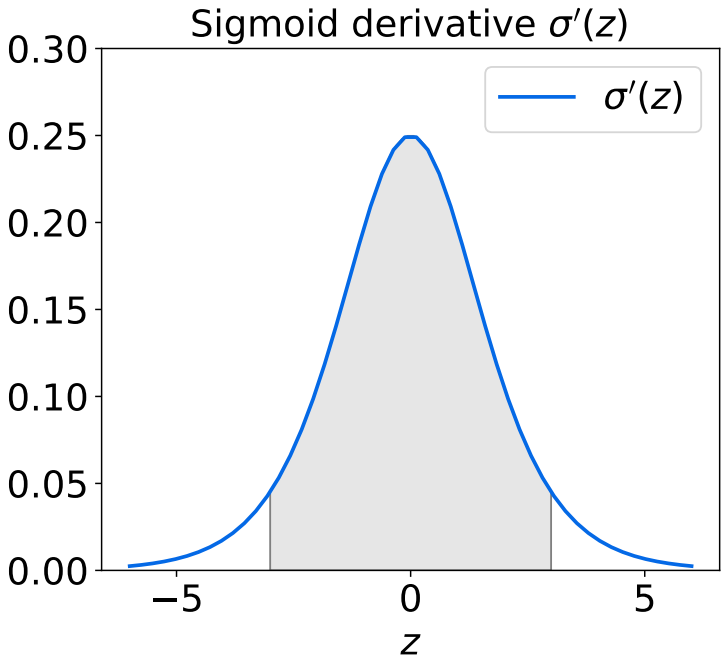
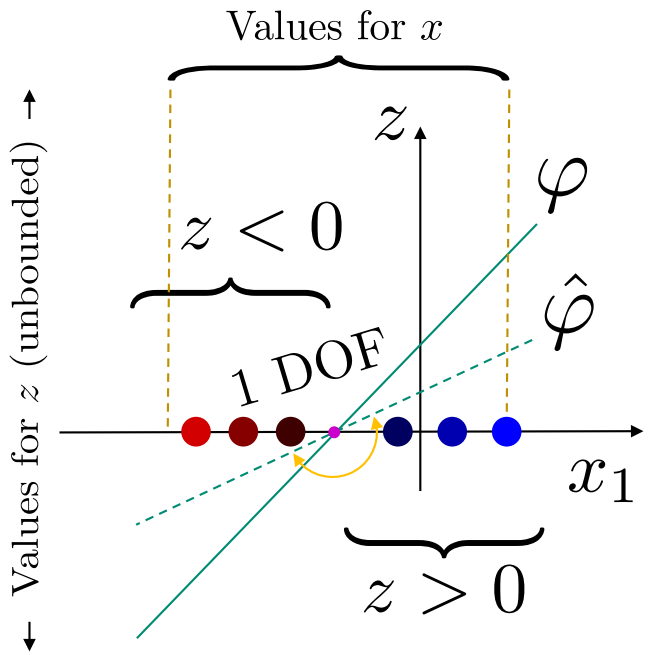


Figure 4: **DOF** at classic perceptrons. Figure 5: Dynamic region: $z \in [-L, +L]$.

- z is unbounded. We need to avoid node saturation.
- Notice: Rotation axis Γ holds inference structure of the perceptron.
- Nice couple: Something not controlled and something to control.

Graphical interpretation of the pre-activation phase (at 1D)



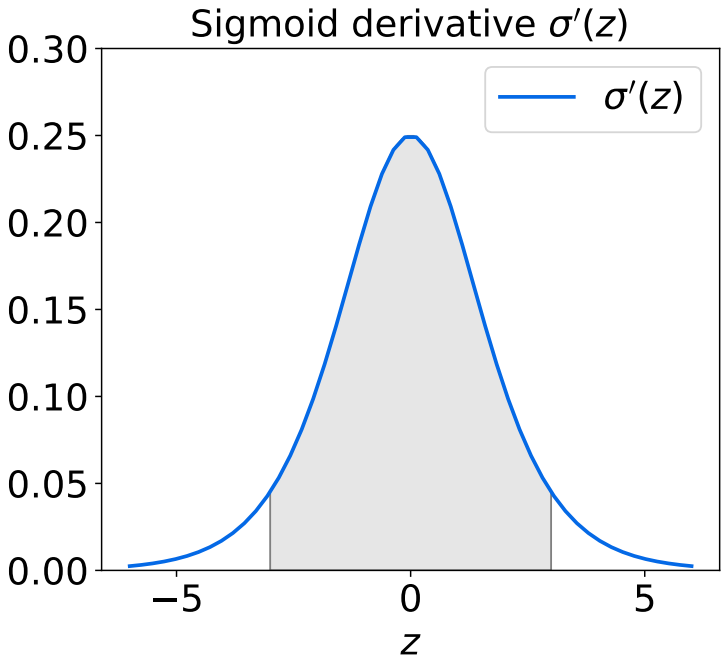
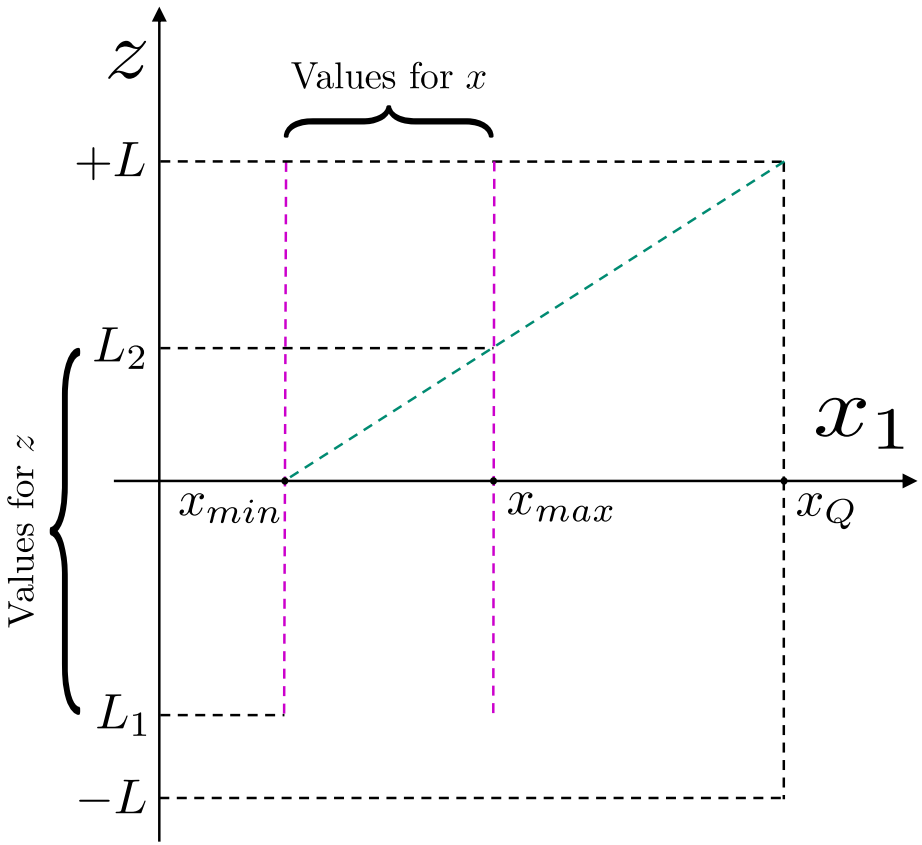


Figure 4: **DOF** at classic perceptrons. Figure 5: Dynamic region: $z \in [-L, +L]$.

- z is unbounded. We need to avoid node saturation.
- Notice: Rotation axis Γ holds inference structure of the perceptron.
- Nice couple: Something not controlled and something to control.
- ARP wisely chooses $\hat{\varphi}$.

Outline

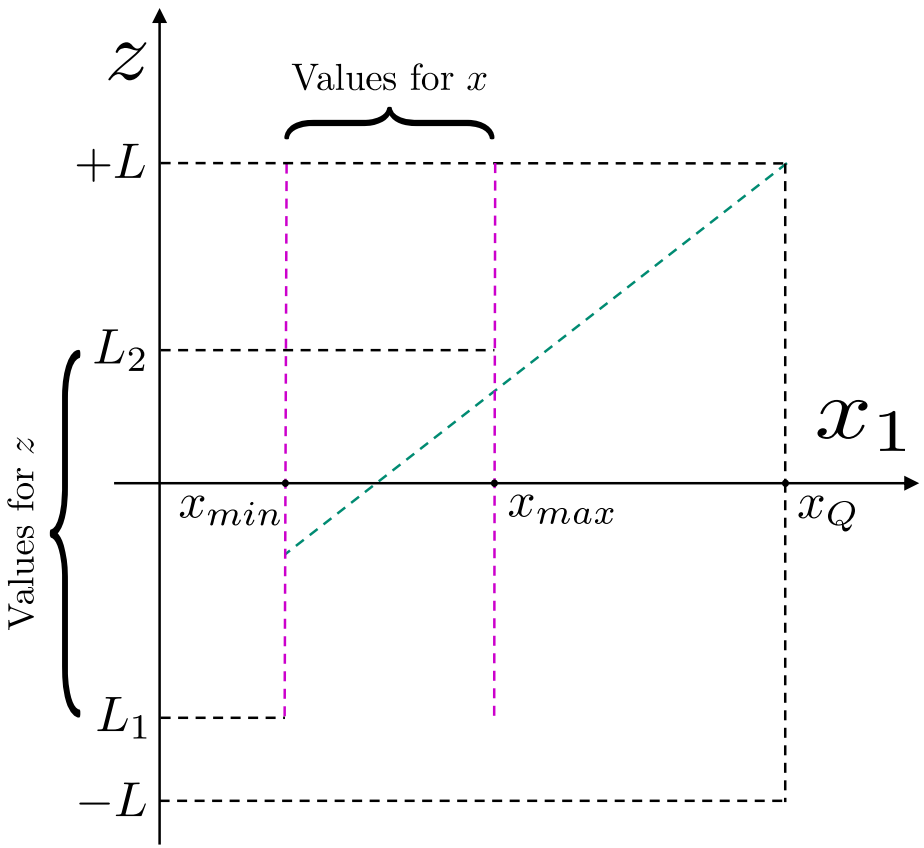
Controlling the rotation (at 1D)



- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.

Figure 6: Rotation bounds z values.

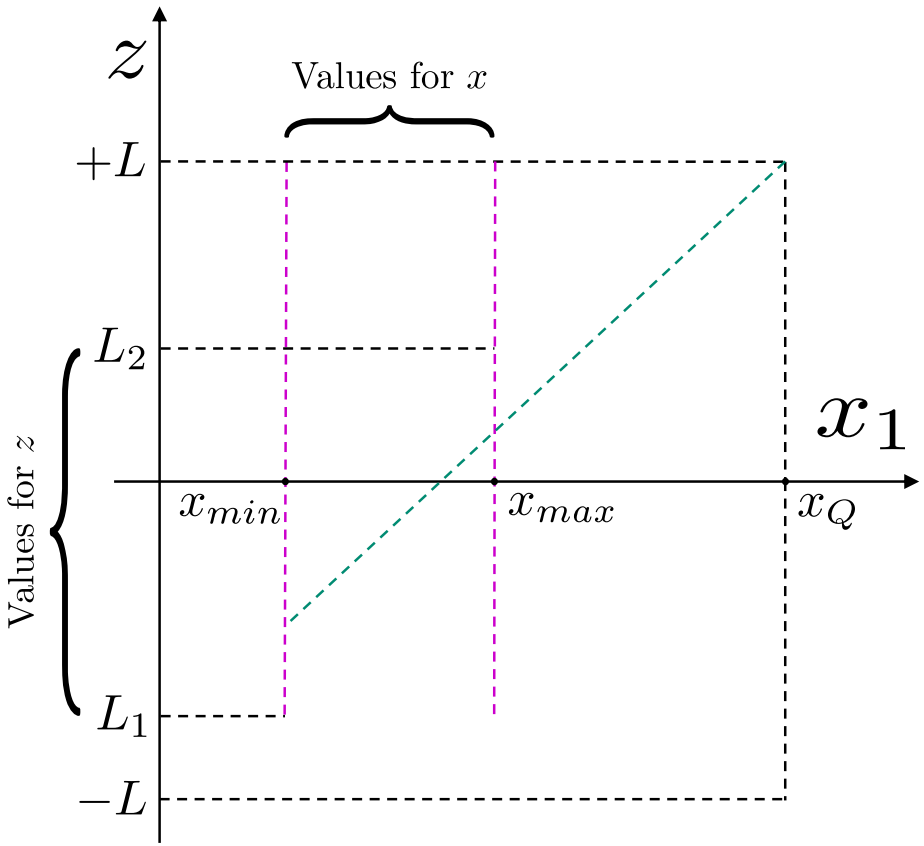
Controlling the rotation (at 1D)



- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.

Figure 6: Rotation bounds z values.

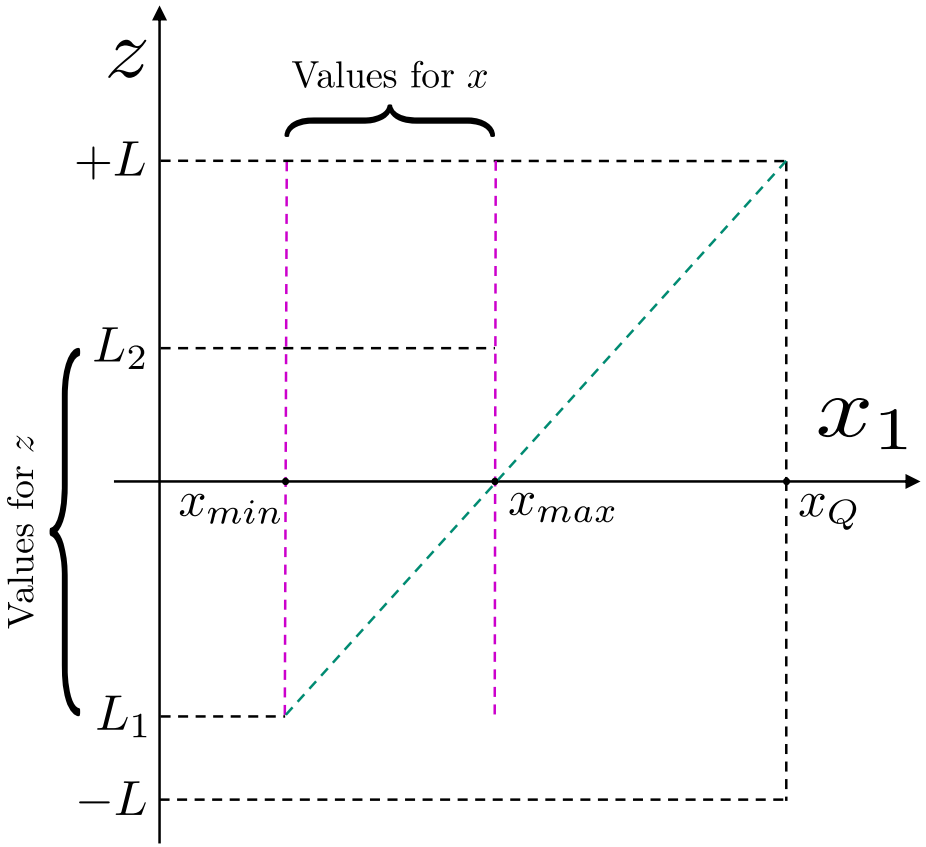
Controlling the rotation (at 1D)



- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.

Figure 6: Rotation bounds z values.

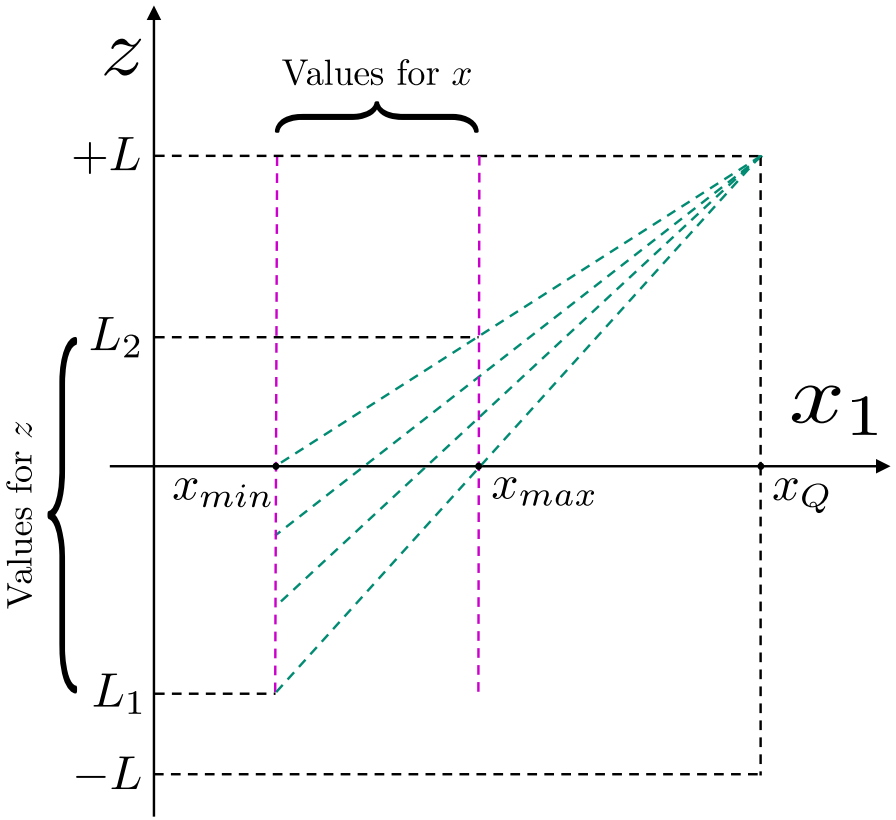
Controlling the rotation (at 1D)



- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.

Figure 6: Rotation bounds z values.

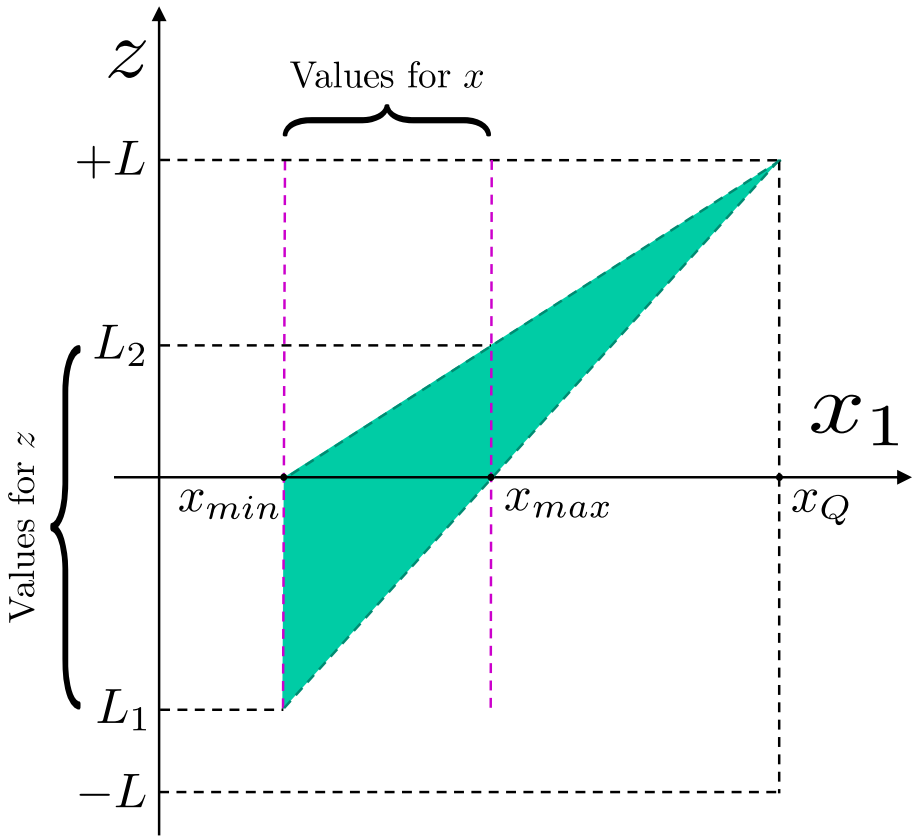
Controlling the rotation (at 1D)



- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.

Figure 6: Rotation bounds z values.

Controlling the rotation (at 1D)



- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.

Figure 6: Rotation bounds z values.

Controlling the rotation (at 1D)

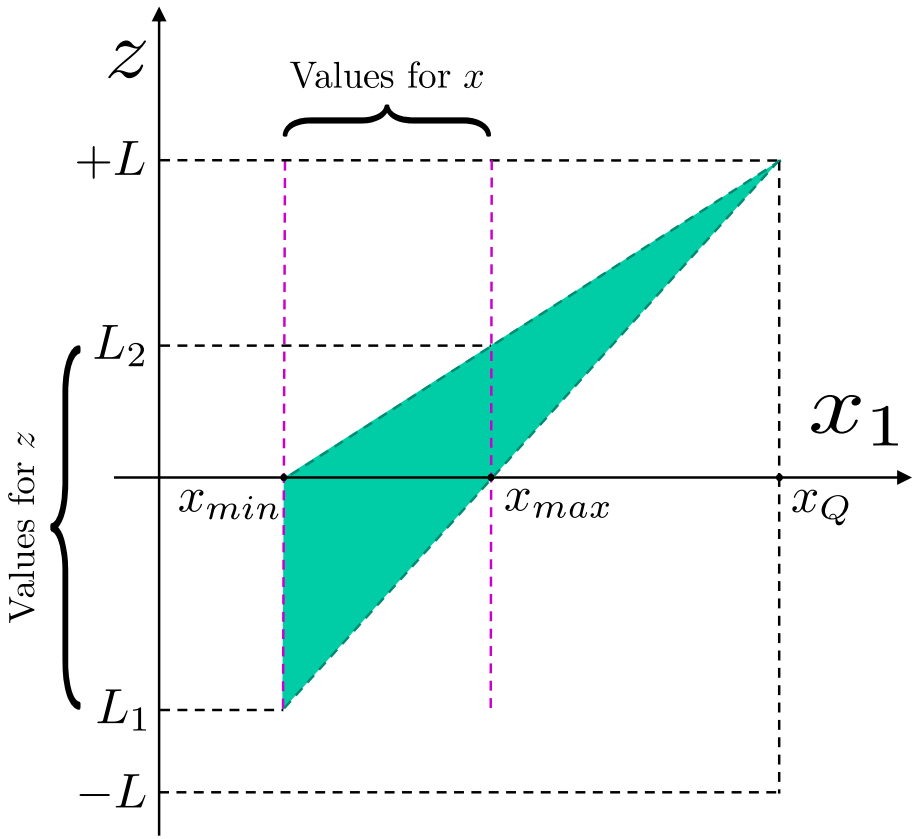


Figure 6: Rotation bounds z values.

- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.
- Green region: all possible positions for $\hat{\varphi} \supset \Gamma$.

Controlling the rotation (at 1D)

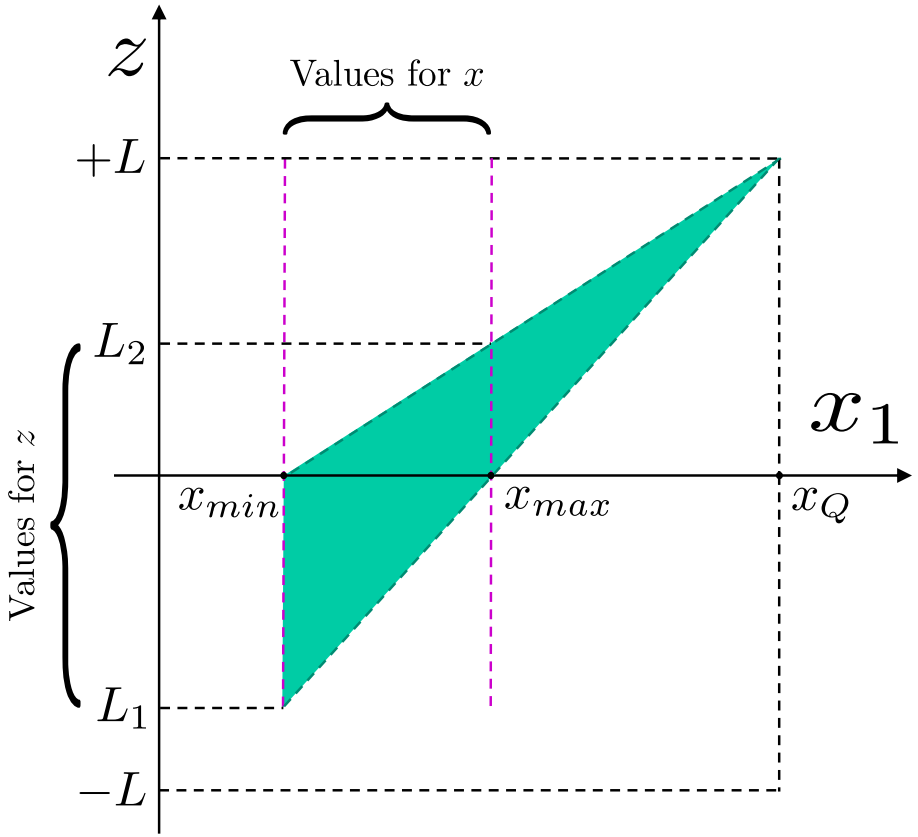


Figure 6: Rotation bounds z values.

- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.
- Green region: all possible positions for $\hat{\varphi} \supset \Gamma$.
- In reality: $z \in [L_1, L_2]$.
- Consider: $z \in [-L, +L]$.

Controlling the rotation (at 1D)

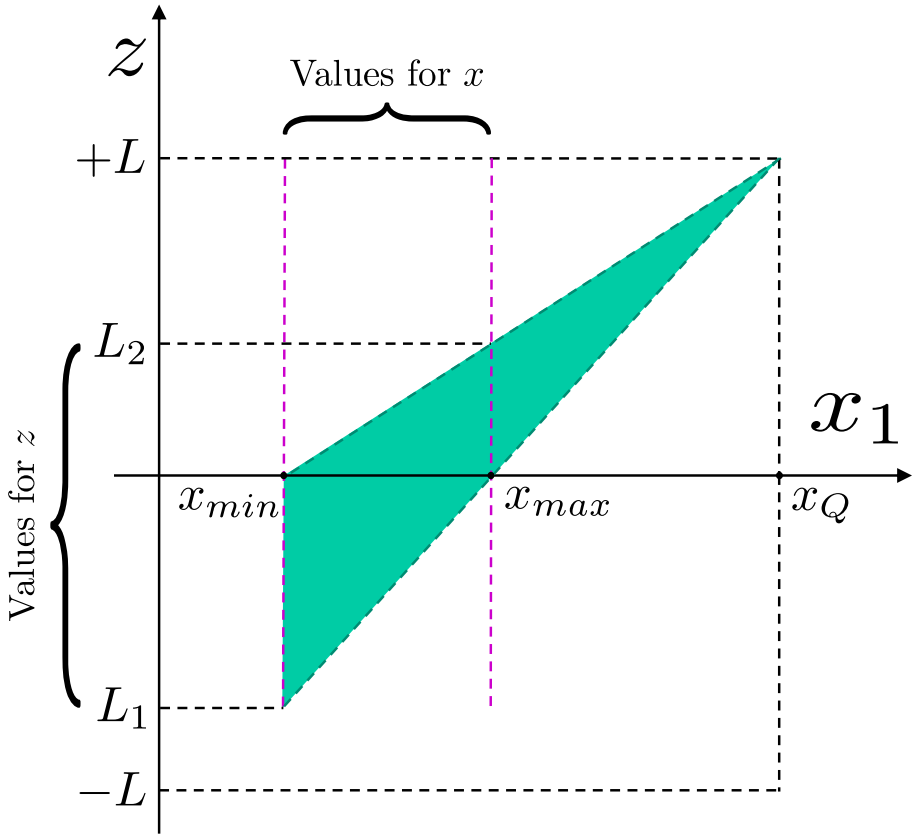


Figure 6: Rotation bounds z values.

- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.
- Green region: all possible positions for $\hat{\varphi} \supset \Gamma$.
- In reality: $z \in [L_1, L_2]$.
- Consider: $z \in [-L, +L]$.
- Rotation depends on $\rho(L, \mathbf{x}_Q)$.
- Consequence: Limit the z values that will enter $\sigma(z)$.

Controlling the rotation (at 1D)

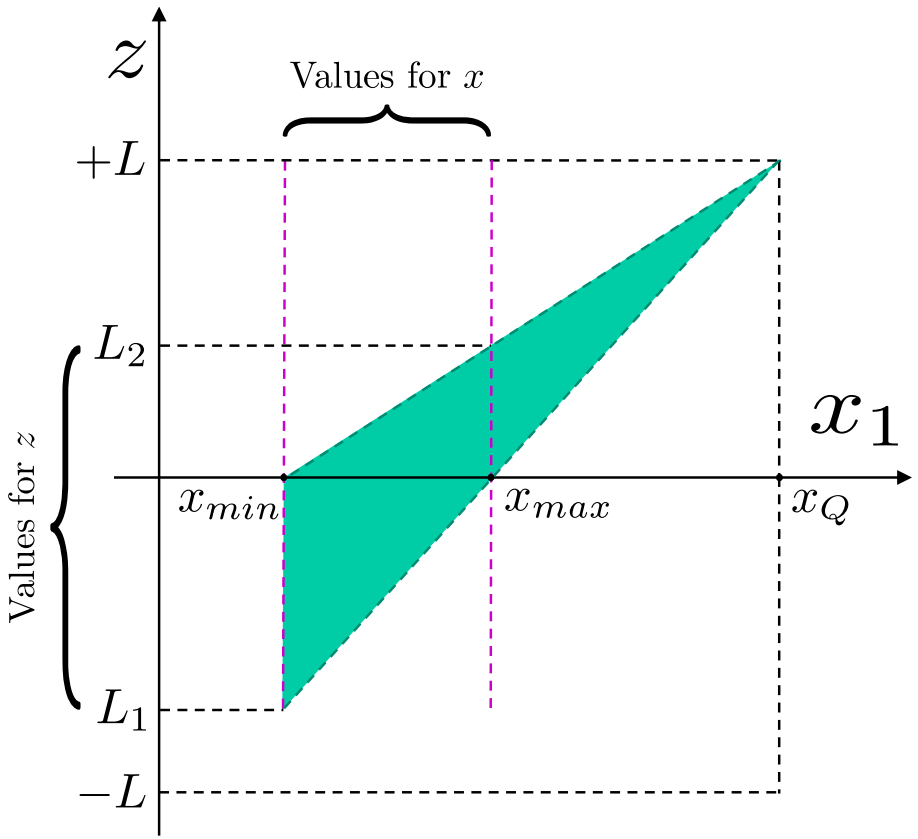


Figure 6: Rotation bounds z values.

- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.
- Green region: all possible positions for $\hat{\varphi} \supset \Gamma$.
- In reality: $z \in [L_1, L_2]$.
- Consider: $z \in [-L, +L]$.
- Rotation depends on $\rho(L, \mathbf{x}_Q)$.
- Consequence: Limit the z values that will enter $\sigma(z)$.
- Formulation is extrapolated to the n-dimensional case.

Controlling the rotation (at 1D)

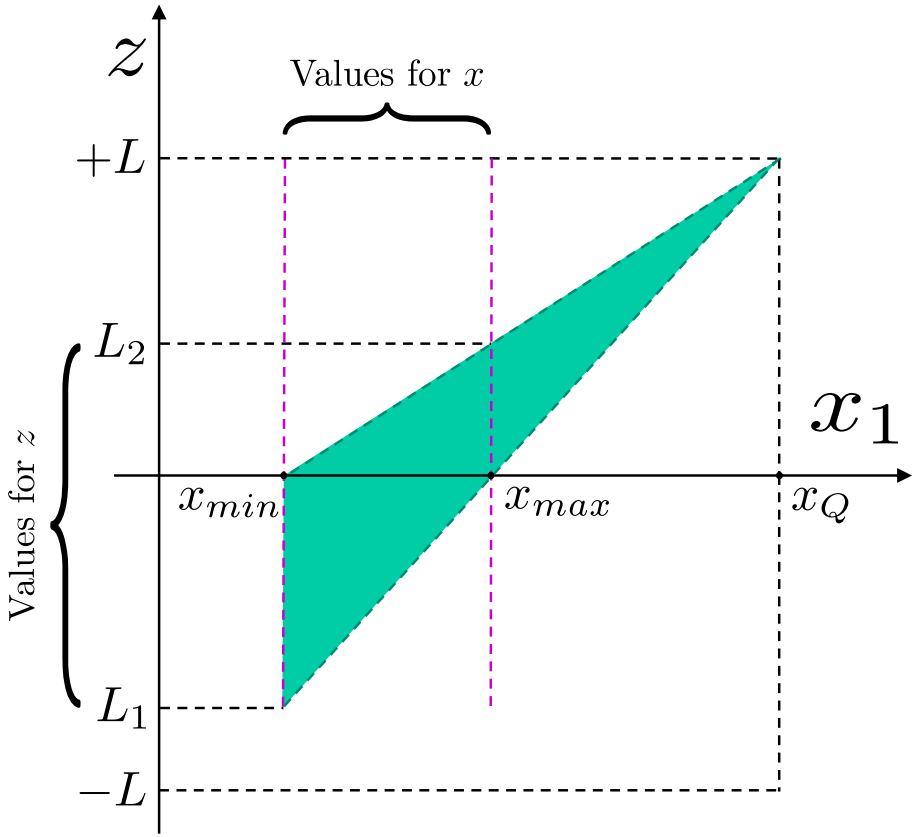


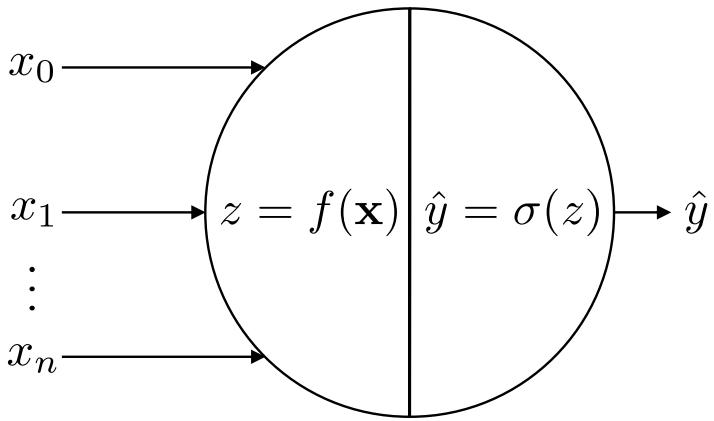
Figure 6: Rotation bounds z values.

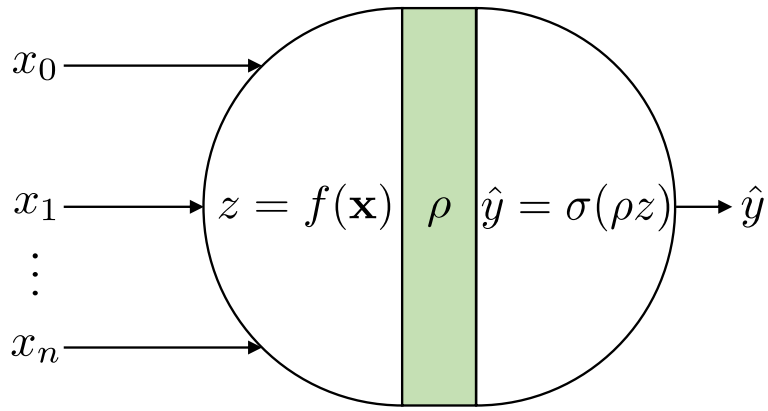
- **Hyperparameters**: L and $\mathbf{x}_Q$.
- $\mathbf{x}_Q = \langle x_Q, \cdots, x_Q \rangle$.
- **Condition 1:** $\langle \mathbf{x}_Q, L \rangle \in \hat{\varphi}$.
- **Condition 2:** $\Gamma \subset \hat{\varphi}$.
- Green region: all possible positions for $\hat{\varphi} \supset \Gamma$.
- In reality: $z \in [L_1, L_2]$.
- Consider: $z \in [-L, +L]$.
- Rotation depends on $\rho(L, \mathbf{x}_Q)$.
- Consequence: Limit the z values that will enter $\sigma(z)$.
- Formulation is extrapolated to the n-dimensional case.
- With $L_1 = -L \rightarrow \mathbf{x}_Q$ ✓

Outline

Auto-Rotating Perceptrons (ARP)

- Avoid the activation function derivative $\sigma'(z)$ to be very small.
- Keep z value that enters to $\sigma(z)$ near zero.
- Rotation does **not** change the inference structure of the perceptron.
- Hyperparameters: L and $\mathbf{x}_Q$. Note: $\mathbf{x}_Q$ can be automatically calculated.
- *Auto*: Each perceptron rotates independently of the others.
- Rotation around the $(n-1)$-dimensional axis Γ.





$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0\, x_0 \qquad x_0 = 1$$

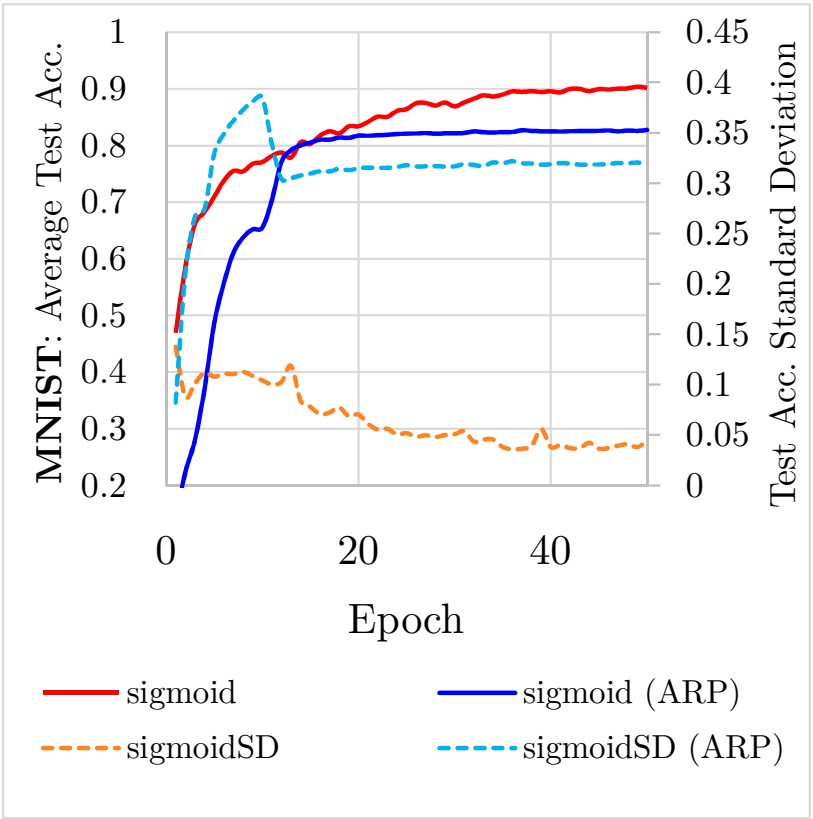
$$g(\mathbf{x}) = \rho\, f(\mathbf{x}) \qquad \rho = \frac{L}{|f(\mathbf{x}_Q)|}$$

Figure 7: Classic perceptron (left) and ARP (right).

Outline

Experimental results [1]

- 50 pairs of neural networks: (image), 50, 50, 40, 30, 30, 20, 10.
- Only difference: One with classic perceptrons and the other with ARP.
- Goal: High average test accuracy with low standard deviation (SD).



- Red curve above the blue one.
- High standard deviation.
- Very scattered results.
- We cannot say which perceptron type is better.

Figure 8: Test accuracy throughout the epochs (MNIST).

Experimental results [1]

- 50 pairs of neural networks: (image), 50, 50, 40, 30, 30, 20, 10.
- Only difference: One with classic perceptrons and the other with ARP.
- Goal: High average test accuracy with low standard deviation (SD).

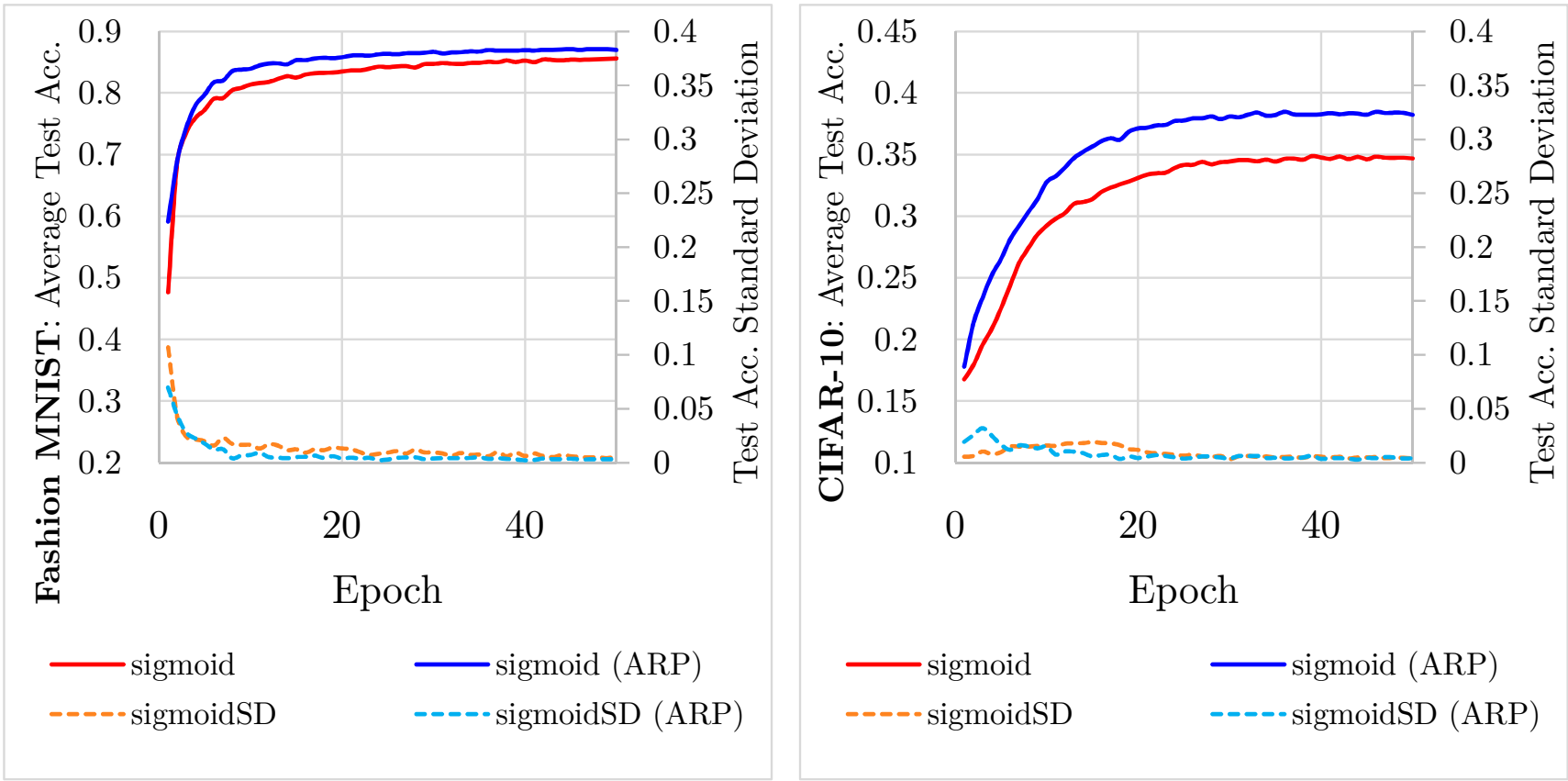


Figure 9: Test accuracy throughout the epochs (Fashion MNIST and CIFAR-10).

Outline

ARP used in a physical problem — WEVES calibration [3]

- An insole-type WEarable VErtical Sensor system (WEVES) for GRF measurement was developed at GIRAB PUCP laboratory, see Figure 10.

(a) (b)

Figure 10: **(a)** WEVES with insole detail. **(b)** Stand-up straight position test with AMTI (red) and WEVES (green) sensors. Physical system developed by Leonardo Bravo Thais.

ARP used in a physical problem — WEVES calibration [3]

- An insole-type WEarable VErtical Sensor system (WEVES) for GRF measurement was developed at GIRAB PUCP laboratory, see Figure 10.

(a) (b)

Figure 10: **(a)** WEVES with insole detail. **(b)** Stand-up straight position test with AMTI (red) and WEVES (green) sensors. Physical system developed by Leonardo Bravo Thais.

- WEVES measurement signal $\mathbf{w}$ must be as close as possible to the AMTI reference signal $\mathbf{p}$. Regression neural model predicts the calibration factor.

Experimentation — Learning results [3]

- ARP hyperparameters: $x_Q = 2.6$, since the maximum input value is $1 < x_Q$; and $L = 3$, because the sigmoid derivative isn't very small for $|z| \leq 3$.

Experimentation — Learning results [3]

- ARP hyperparameters: $x_Q = 2.6$, since the maximum input value is $1 < x_Q$; and $L = 3$, because the sigmoid derivative isn't very small for $|z| \leq 3$.

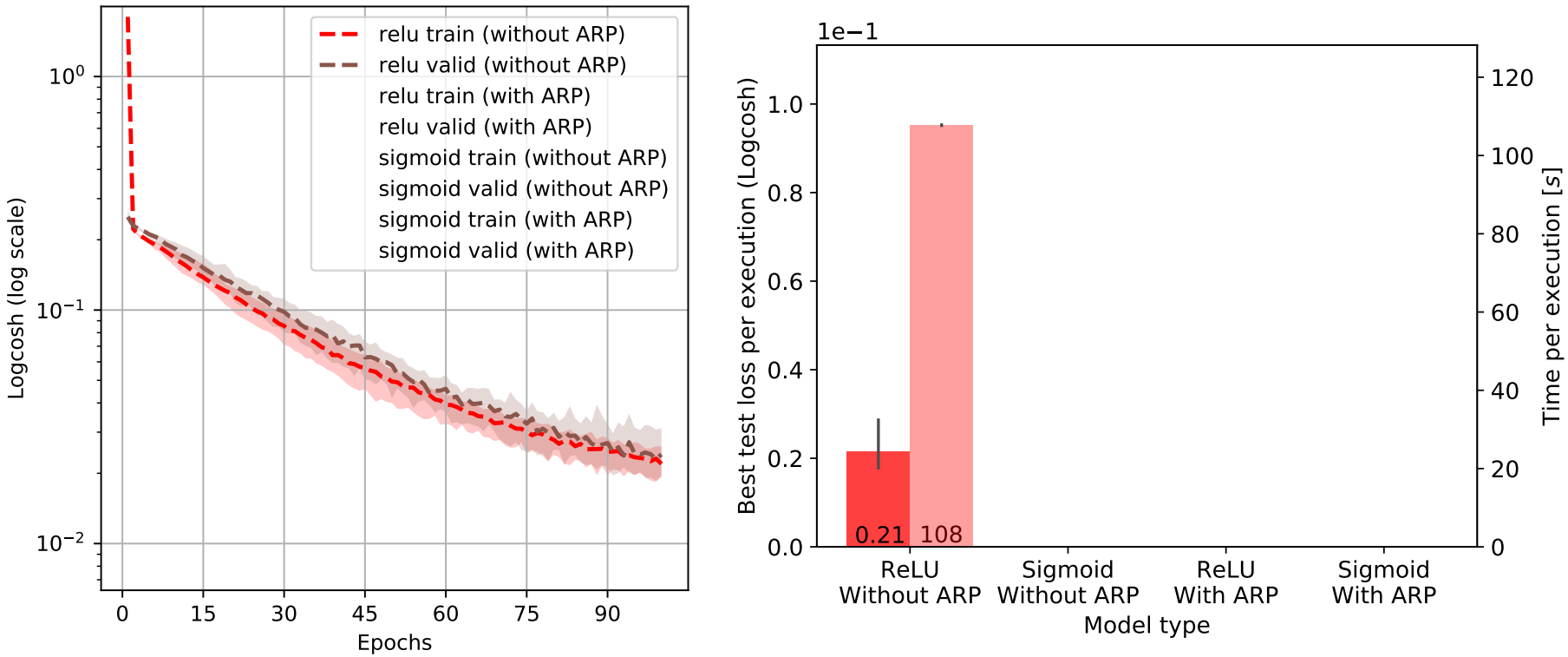


Figure 11: Comparison of the four model types tested. For each model family: 50 executions with 100 epochs per execution. ARP + sigmoid: loss reduction!

Experimentation — Learning results [3]

- ARP hyperparameters: $x_Q = 2.6$, since the maximum input value is $1 < x_Q$; and $L = 3$, because the sigmoid derivative isn't very small for $|z| \leq 3$.

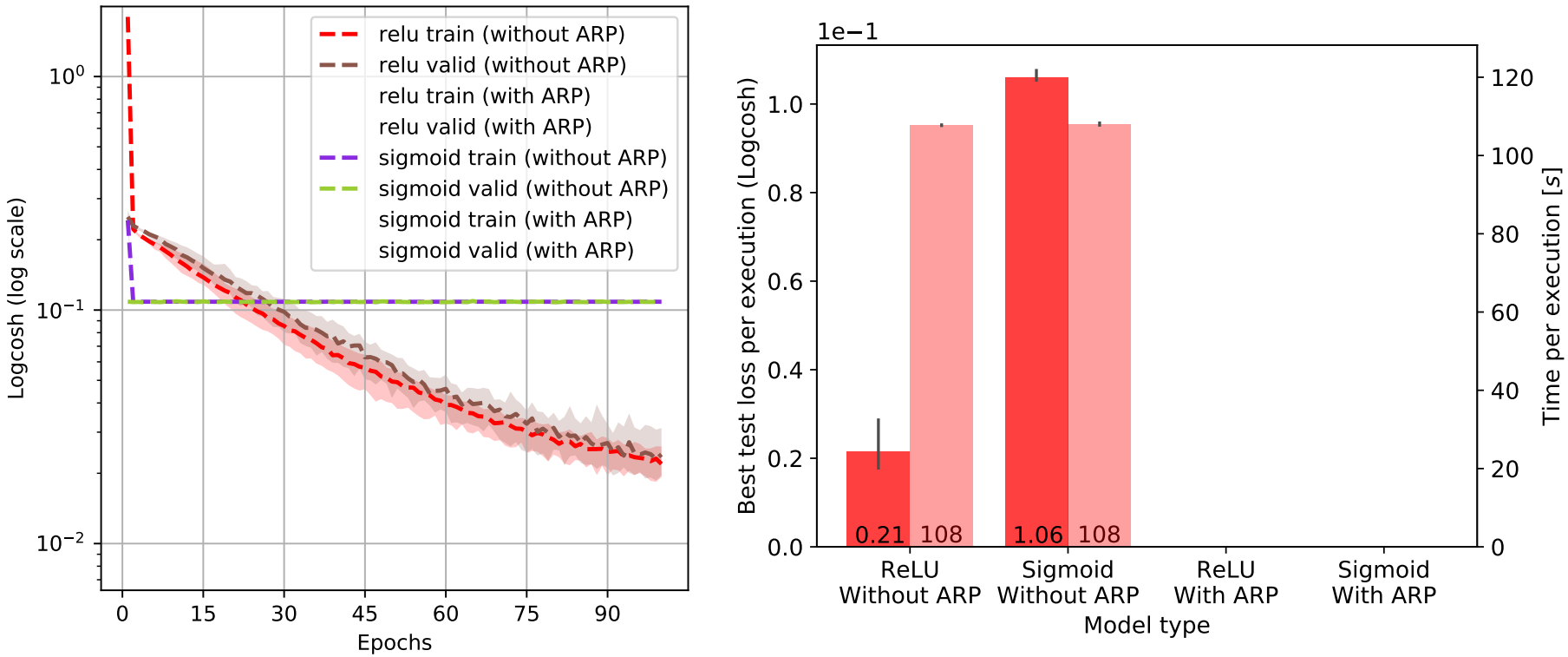
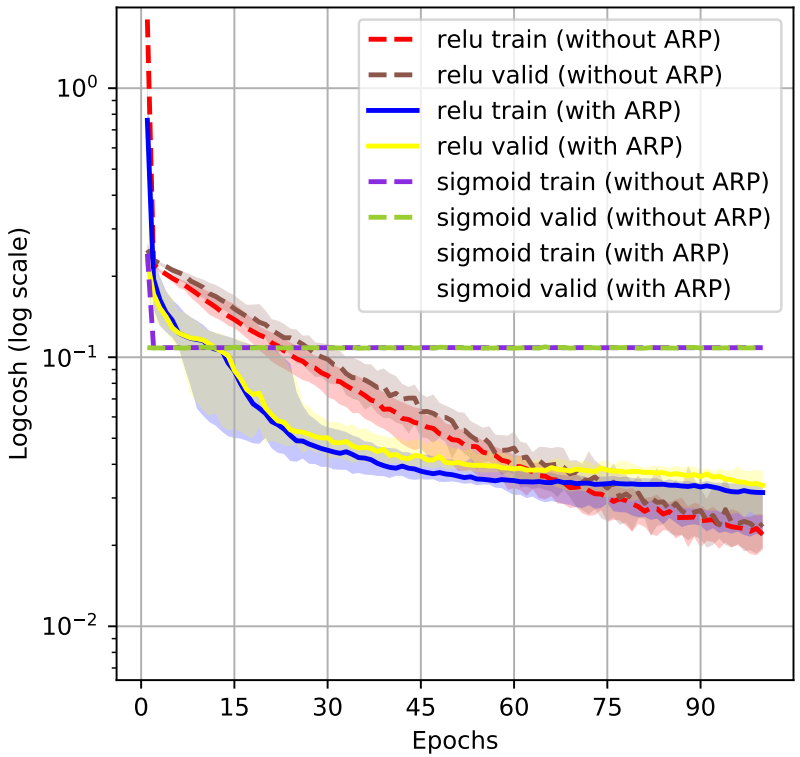


Figure 11: Comparison of the four model types tested. For each model family: 50 executions with 100 epochs per execution. ARP + sigmoid: loss reduction!

Experimentation — Learning results [3]

- ARP hyperparameters: $x_Q = 2.6$, since the maximum input value is $1 < x_Q$; and $L = 3$, because the sigmoid derivative isn't very small for $|z| \leq 3$.



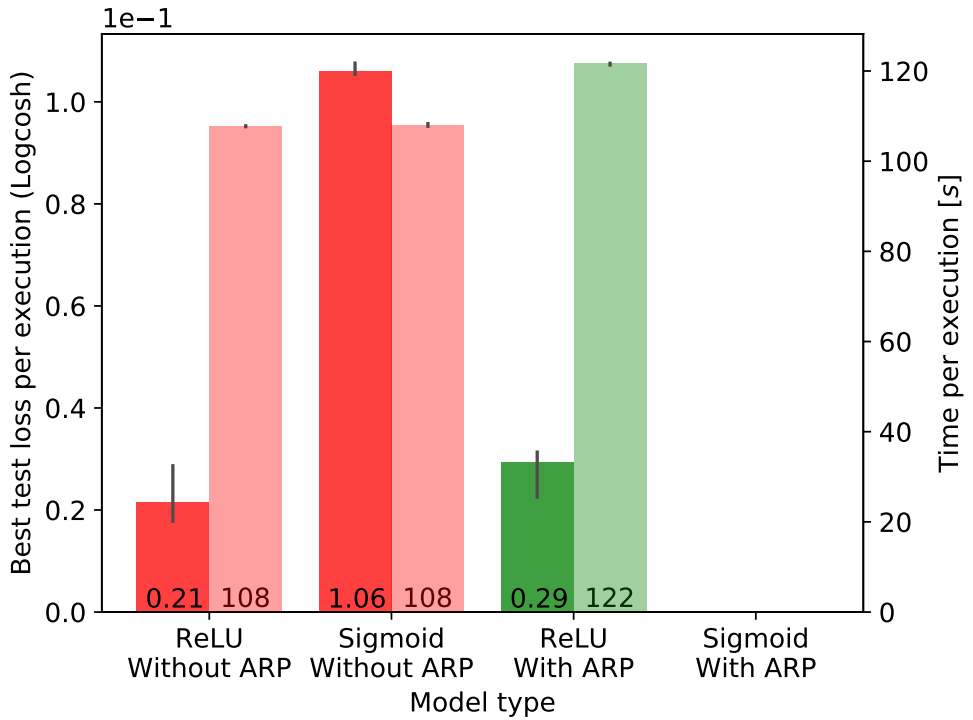


Figure 11: Comparison of the four model types tested. For each model family: 50 executions with 100 epochs per execution. ARP + sigmoid: loss reduction!

Experimentation — Learning results [3]

- ARP hyperparameters: $x_Q = 2.6$, since the maximum input value is $1 < x_Q$; and $L = 3$, because the sigmoid derivative isn't very small for $|z| \leq 3$.

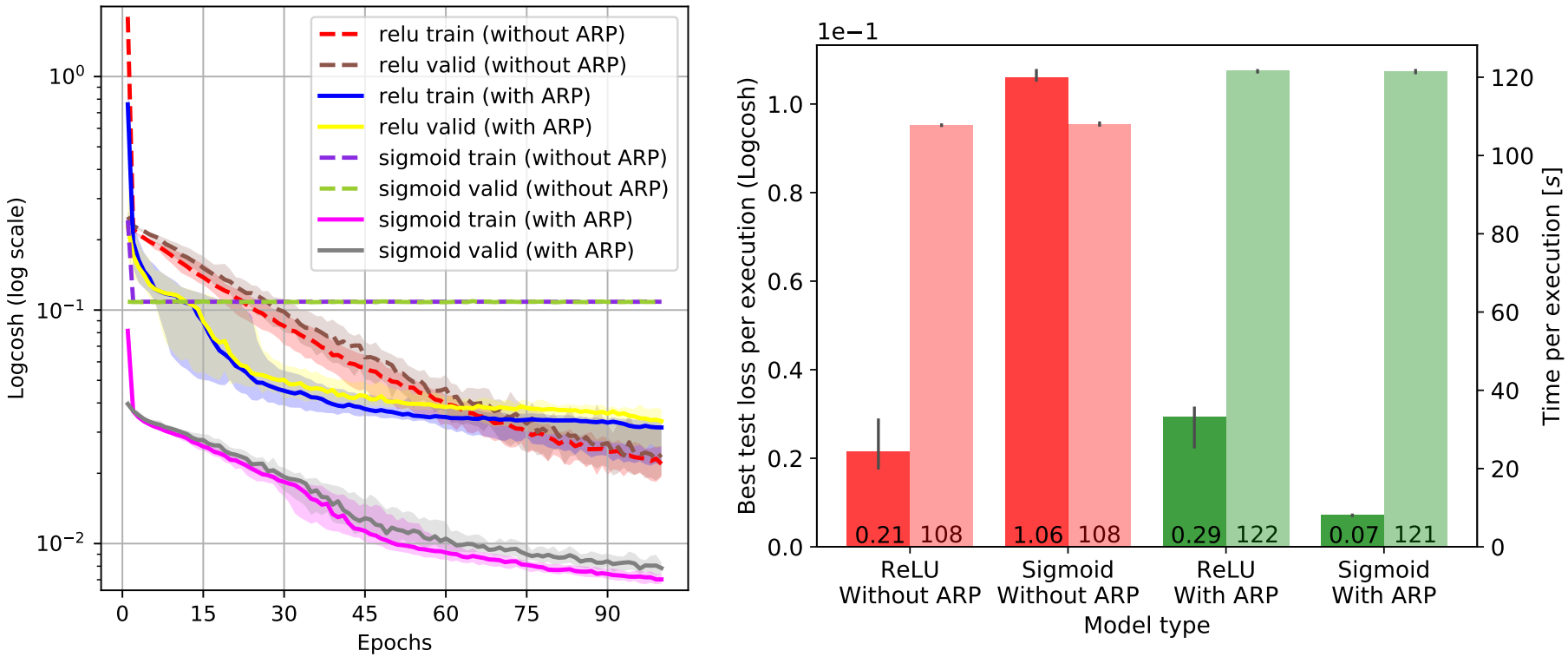


Figure 11: Comparison of the four model types tested. For each model family: 50 executions with 100 epochs per execution. ARP + sigmoid: loss reduction!

Outline

Extending the Auto-Rotating Operation: ARNN

Extending the Auto-Rotating Operation: ARNN

Keras implementations

Keras layers implemented with the Auto-Rotating Operation (**Tip**: Just add `AutoRot` before the layer name):

Keras Original Layer	Auto-Rotating Implementation	Basic version	With refactoring	Tested
`Dense`	`AutoRotDense`	✅	✅	✅
`SimpleRNN`	`AutoRotSimpleRNN`	✅	☐	☐
`LSTM`	`AutoRotLSTM`	✅	☐	☐
`GRU`	`AutoRotGRU`	✅	☐	☐
`Conv1D`	`AutoRotConv1D`	✅	✅	✅
`Conv2D`	`AutoRotConv2D`	✅	✅	✅
`Conv3D`	`AutoRotConv3D`	✅	✅	✅
`Conv2DTranspose`	`AutoRotConv2DTranspose`	✅	☐	☐
`Conv3DTranspose`	`AutoRotConv3DTranspose`	✅	☐	☐
`SeparableConv`	`AutoRotSeparableConv`	✅	☐	☐
`SeparableConv1D`	`AutoRotSeparableConv1D`	✅	☐	☐
`SeparableConv2D`	`AutoRotSeparableConv2D`	✅	☐	☐
`DepthwiseConv2D`	`AutoRotDepthwiseConv2D`	✅	☐	☐

Extending the Auto-Rotating Operation: ARNN

- ARNN can be seen as a generalization of classical neural networks.

```
model_base = Sequential()
model_base.add(AutoRotConv2D(32, (3, 3), padding='same',
    ... input_shape=x_train.shape[1:], **dict_AutoRot_params))
model_base.add(Activation(activFun))
model_base.add(AutoRotConv2D(32, (3, 3), xmin_lim=xmin_lim,
    ... xmax_lim=xmax_lim, L=L))
model_base.add(Activation(activFun))
model_base.add(MaxPooling2D(pool_size=(2, 2)))
model_base.add(Dropout(0.25))

# ...  Similar layer block with 64-sized convolutional kernels

model_base.add(Flatten())
model_base.add(AutoRotDense(512, **dict_AutoRot_params))
model_base.add(Activation(activFun))
model_base.add(Dropout(0.5))
model_base.add(Dense(num_classes)) #AutoRot not needed at last layer
model_base.add(Activation('softmax'))
```

Outline

Experimental results 2020/2021

Why using Auto-Rotating Perceptrons?

- There is evidence that using the Auto-Rotation with a sigmoid-activated neural network can lead to a **huge boosting performance** (15x, see [3]).
- ARP-sigmoid networks can have a better performance than ReLU networks with classic neurons **without altering the inference structure** learned by the perceptron.
- Auto-Rotating layers can be **easily inserted** into your models. Wherever you put a `Dense` layer, you can replace it with an `AutoRotDense` layer.

Let's see another examples analyzing the effects of changing from classic layers to Auto-Rotating layers.

Experiment A - CIFAR-10 using only dense layers

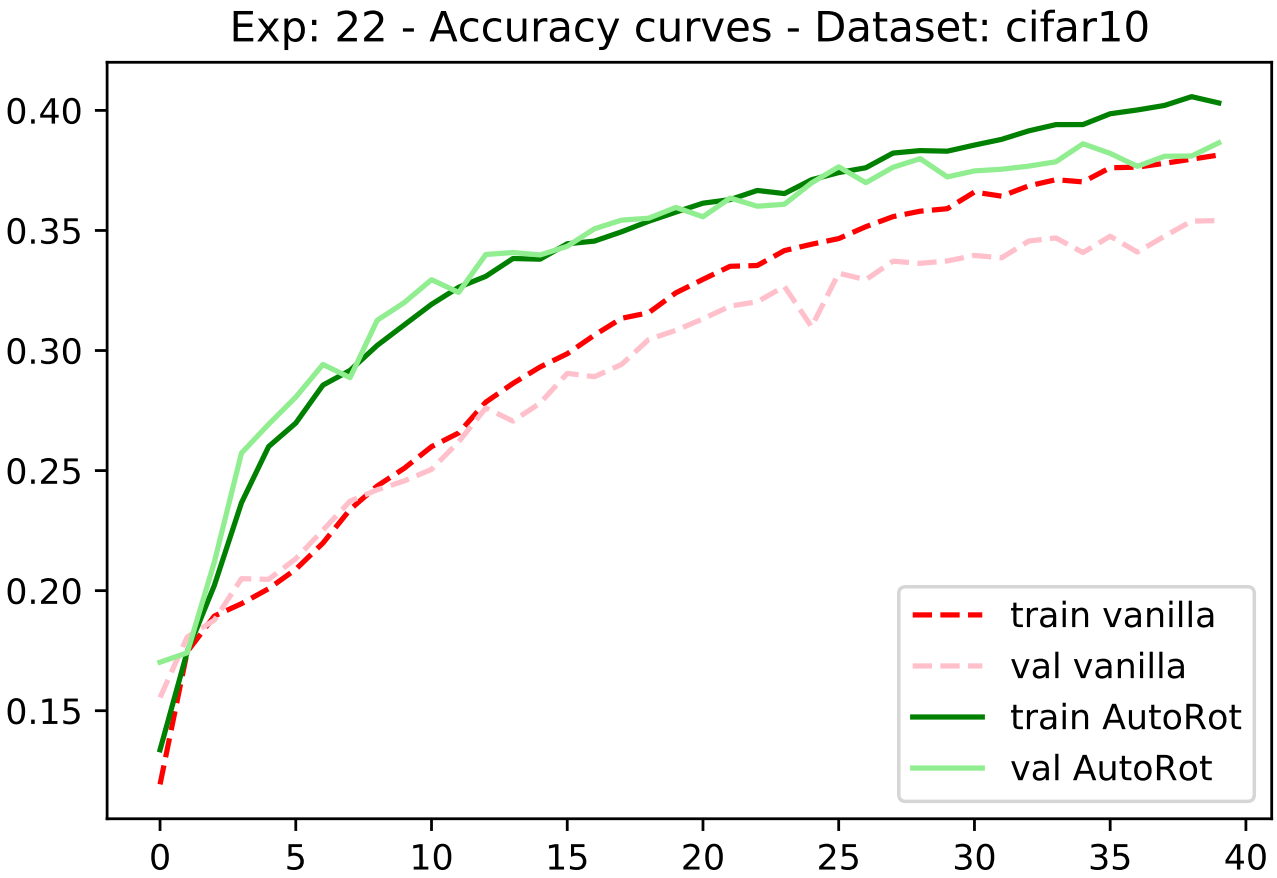


Figure 13: Accuracy curves: Architecture: 1024, 50, 50, 40, 40, 30, 10.

Experiment B - CIFAR-10 using dense and conv. layers

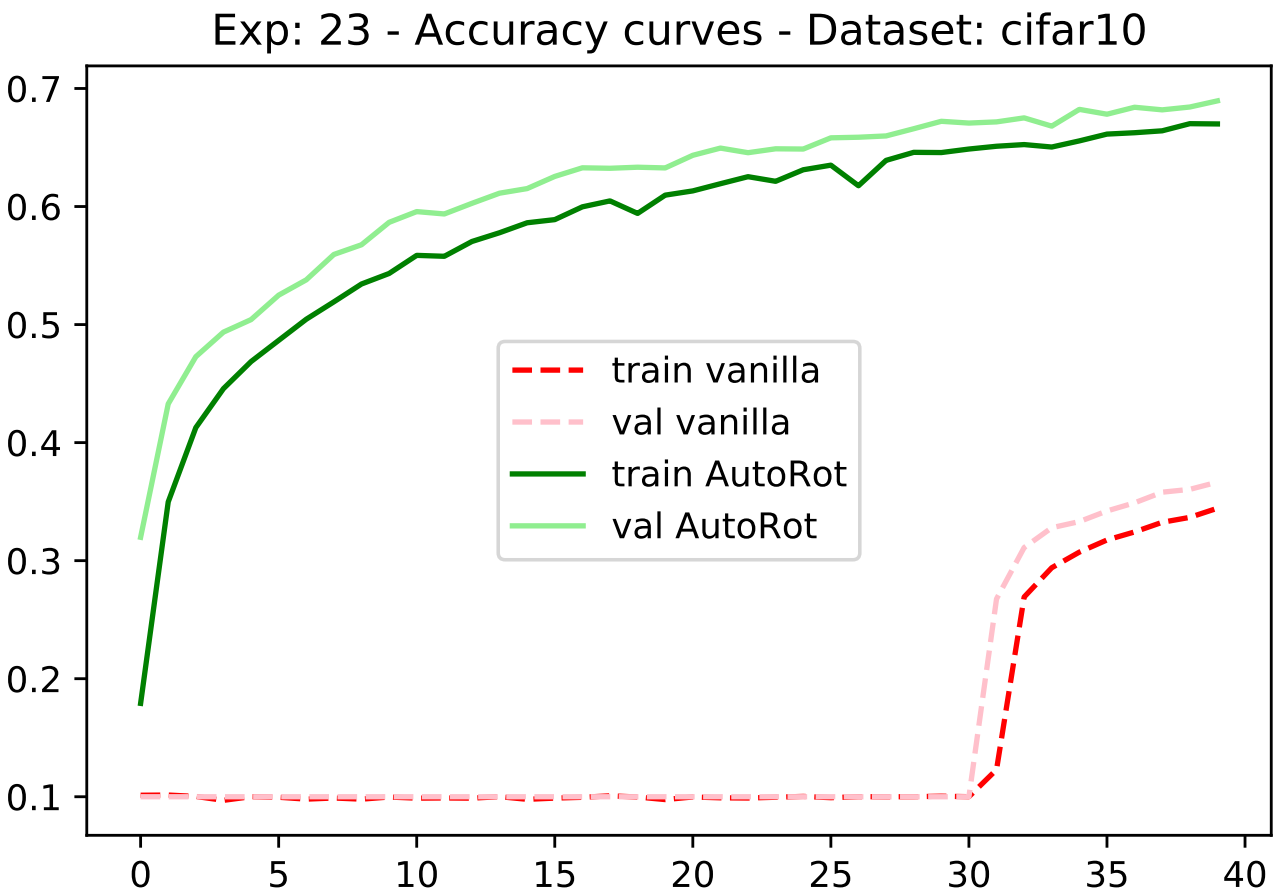


Figure 14: Accuracy curves. Arch.: VGG-based with sigmoid activation (slide 26).

Experiment B - CIFAR-10 using dense and conv. layers

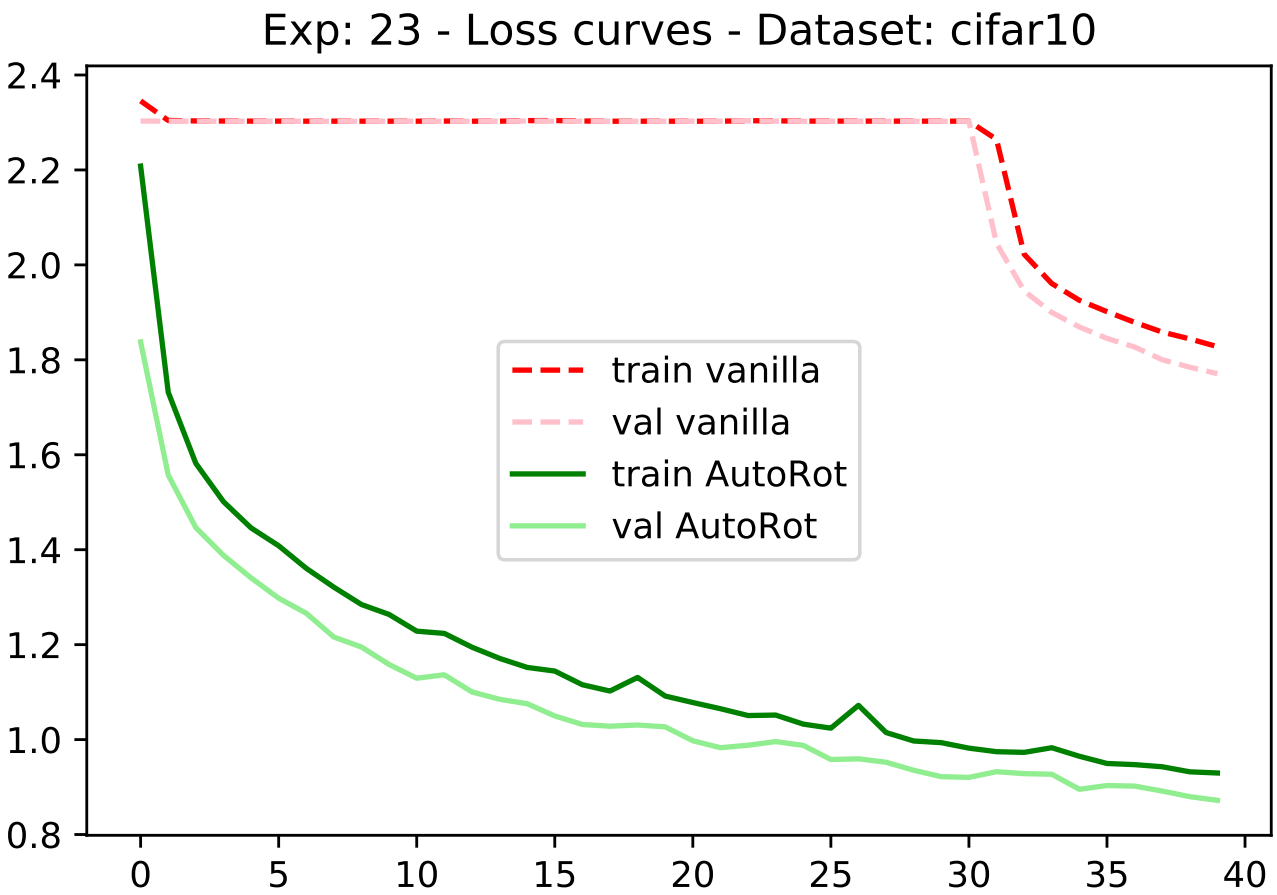


Figure 15: Loss curves. Arch.: VGG-based with sigmoid activation (slide 26).

Experiment C - CIFAR-10: Shallower dense + conv. net

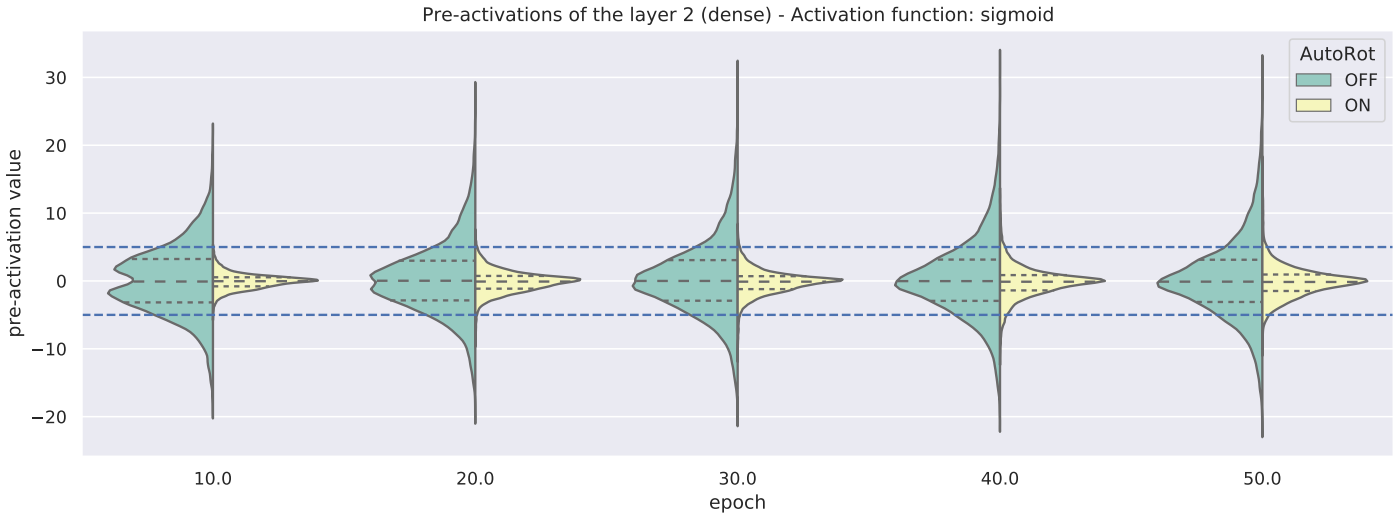
- We implemented a shallower network to see the pre-activations and activations. We present the plot results from the first dense layer.

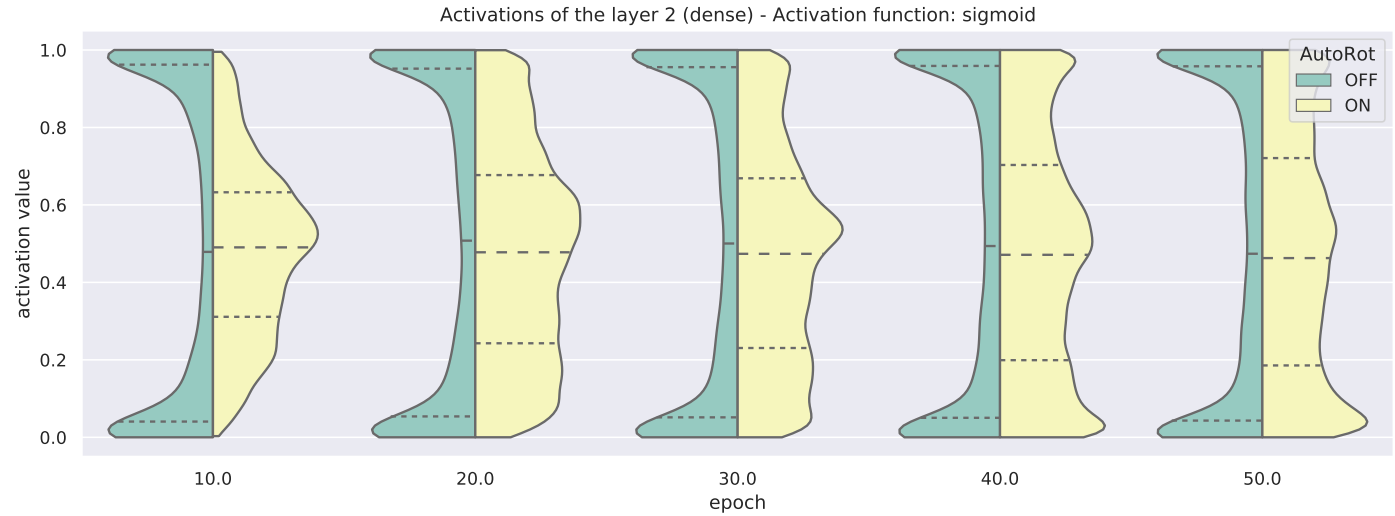
```
model_ARP = keras.Sequential([

keras.Input(shape=input_shape),
AutoRotConv2D(32, (6, 6), xmin_lim, xmax_lim, L, activation="linear"),
layers.Activation('sigmoid'),
layers.MaxPooling2D(pool_size=(2, 2)),
AutoRotConv2D(16, (3, 3), xmin_lim, xmax_lim, L, activation="linear"),
layers.Activation('sigmoid'),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Flatten(),
layers.Dropout(0.5),
AutoRotDense(30, xmin_lim, xmax_lim, L, activation="linear"),
layers.Activation('sigmoid'),
AutoRotDense(20, xmin_lim, xmax_lim, L, activation="linear"),
layers.Activation('sigmoid'),
AutoRotDense(num_classes, xmin_lim, xmax_lim, L, activation="linear"),
layers.Activation(activations.softmax),
])

model_ARP.summary()
```

Experiment C - CIFAR-10: Shallower dense + conv. net





Experiment C - CIFAR-10: Shallower dense + conv. net

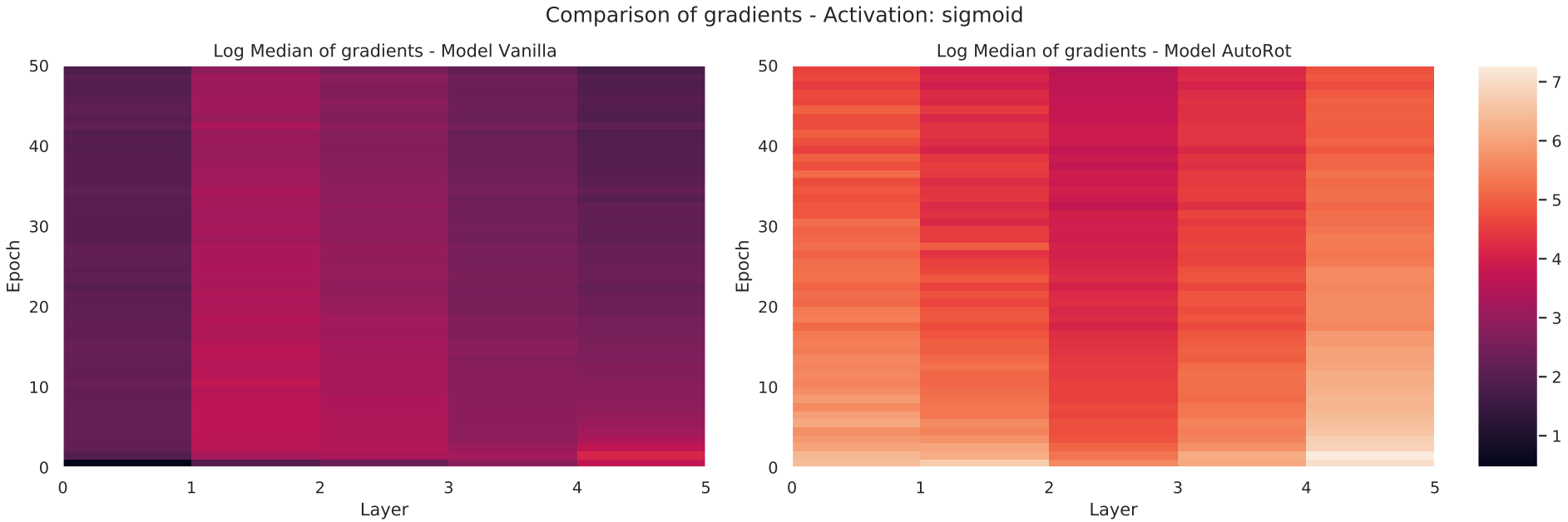


Figure 16: Gradients of the models' trainable weights after 50 epochs.

Experiment C - CIFAR-10: Shallower dense + conv. net

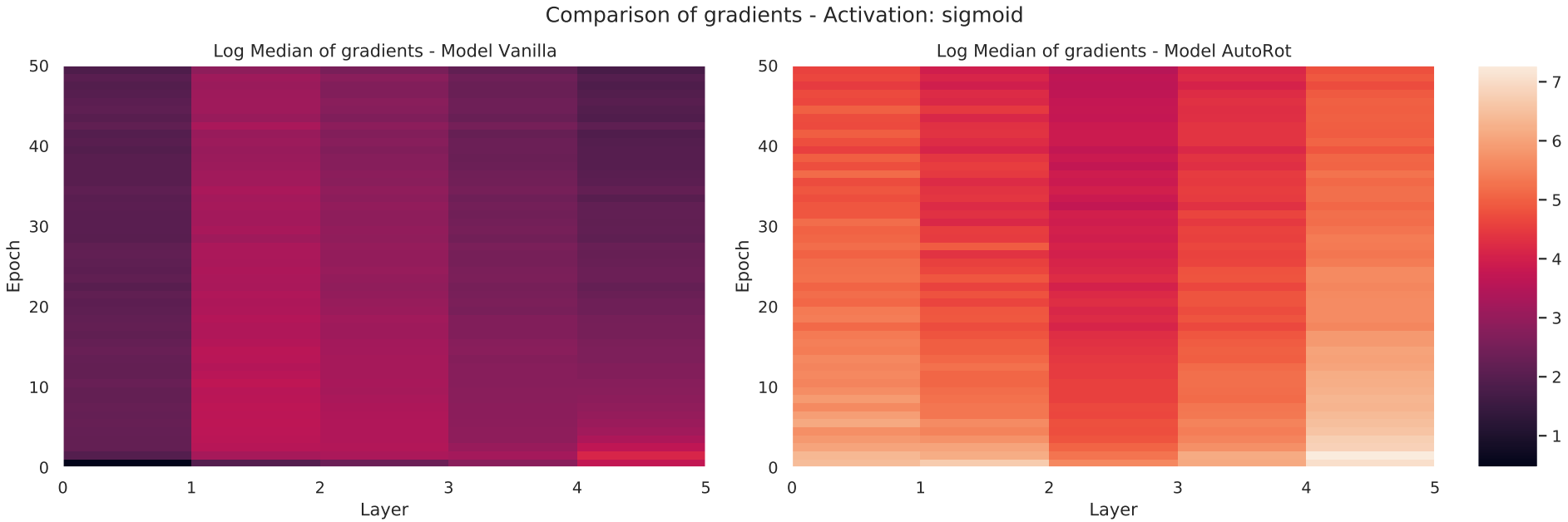


Figure 16: Gradients of the models' trainable weights after 50 epochs.

- Noticeable diminish of the Vanishing Gradient Problem (VGP)!

Experiment C - CIFAR-10: Shallower dense + conv. net

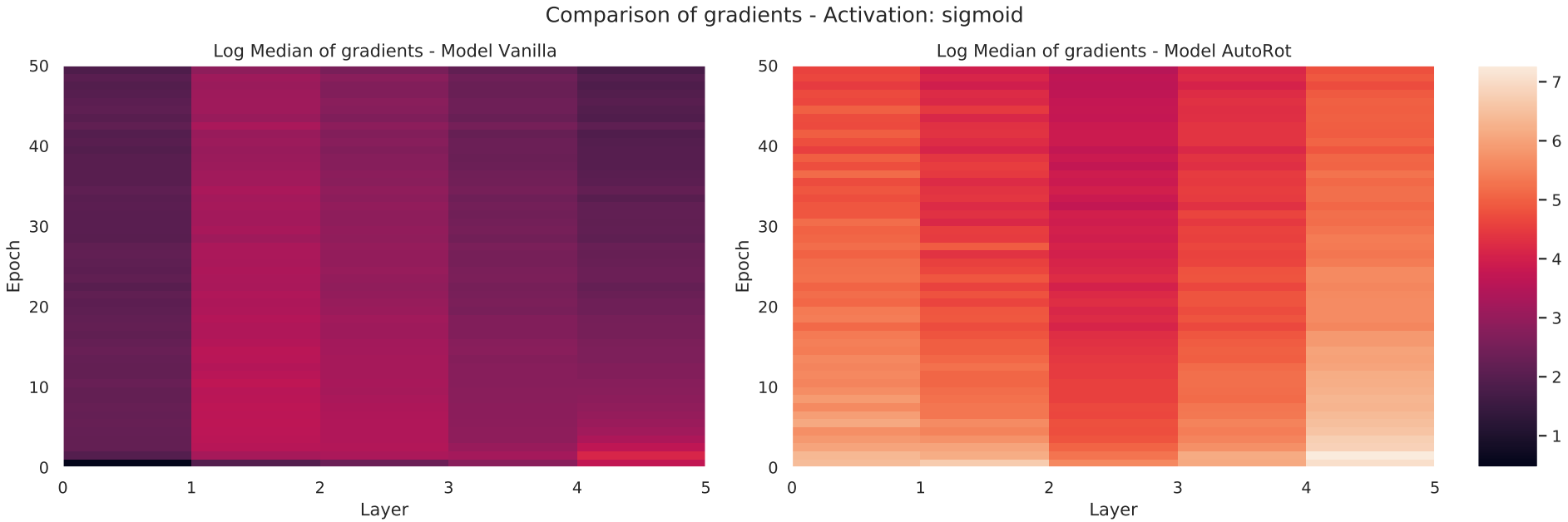


Figure 16: Gradients of the models' trainable weights after 50 epochs.

- Noticeable diminish of the Vanishing Gradient Problem (VGP)!
- How does this affect the final performance of the models?

Experiment C - CIFAR-10: Shallower dense + conv. net

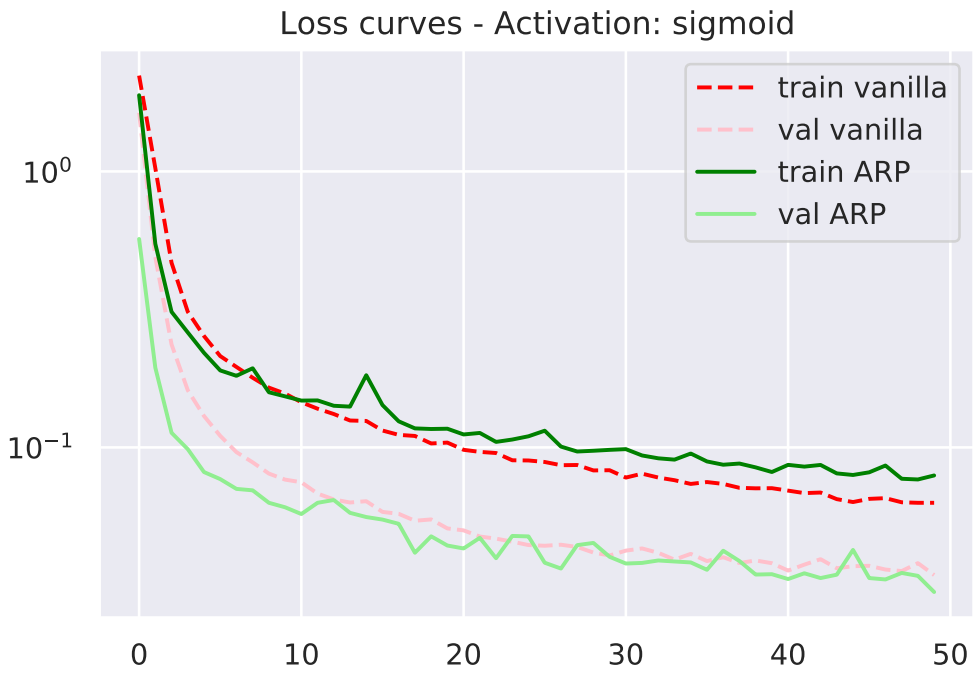


Figure 17: Loss curves after 30 executions. Architecture: See slide 33.

- A bit better performance on the validation test.
- Improvement from Auto-Rotation is stronger for deeper networks.
- Neurons work in their dynamic region.

Experiment D - Regression: Rastrigin function

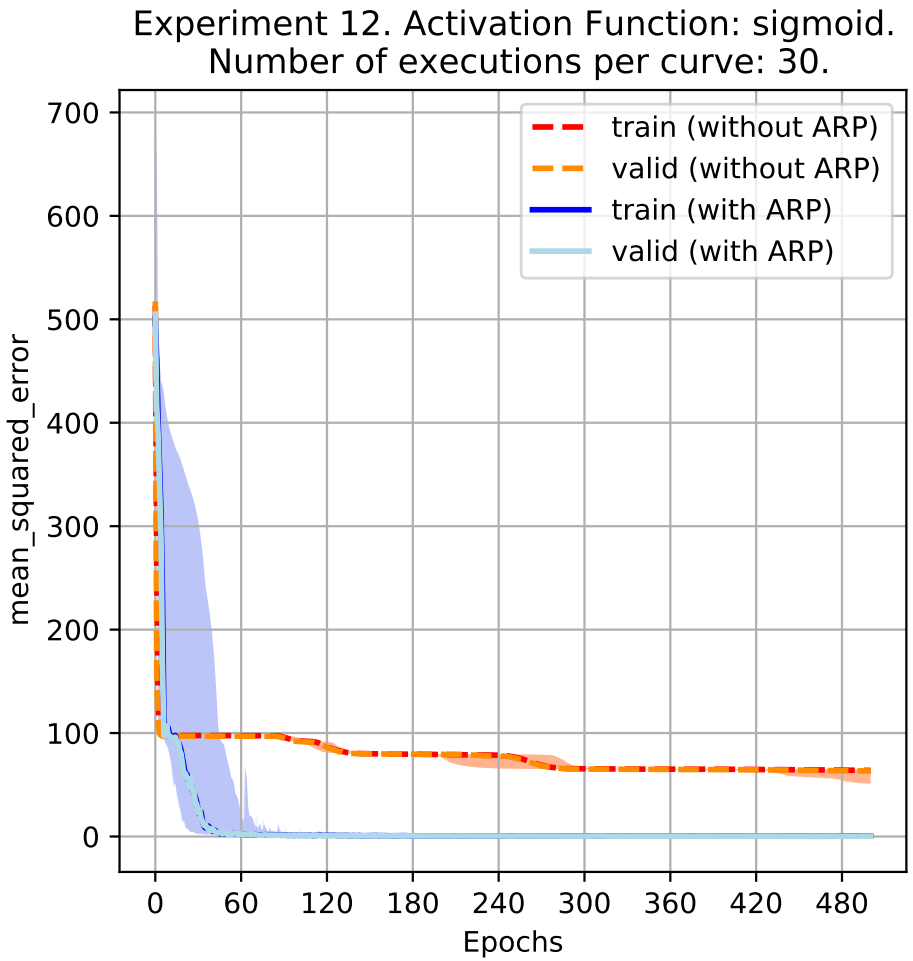


Figure 18: Loss curves after 30 executions. Architecture: 2, 50, **50**, 50, 1.

Experiment D - Regression: Rastrigin function

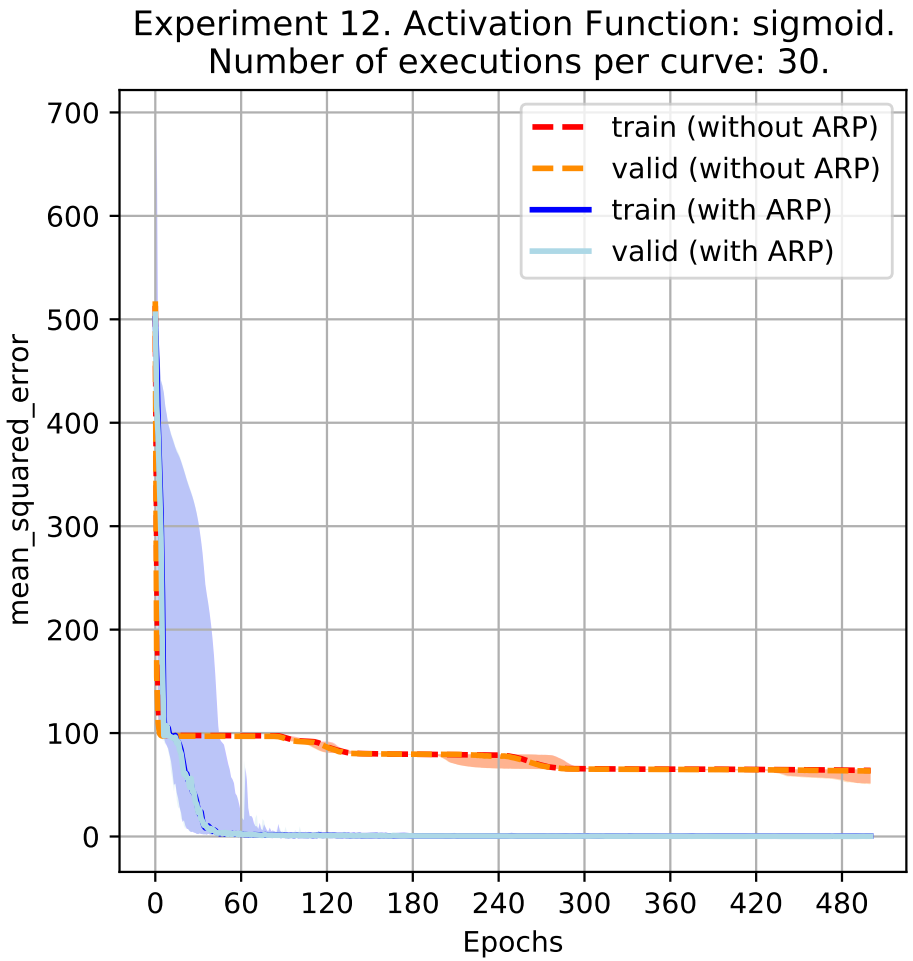
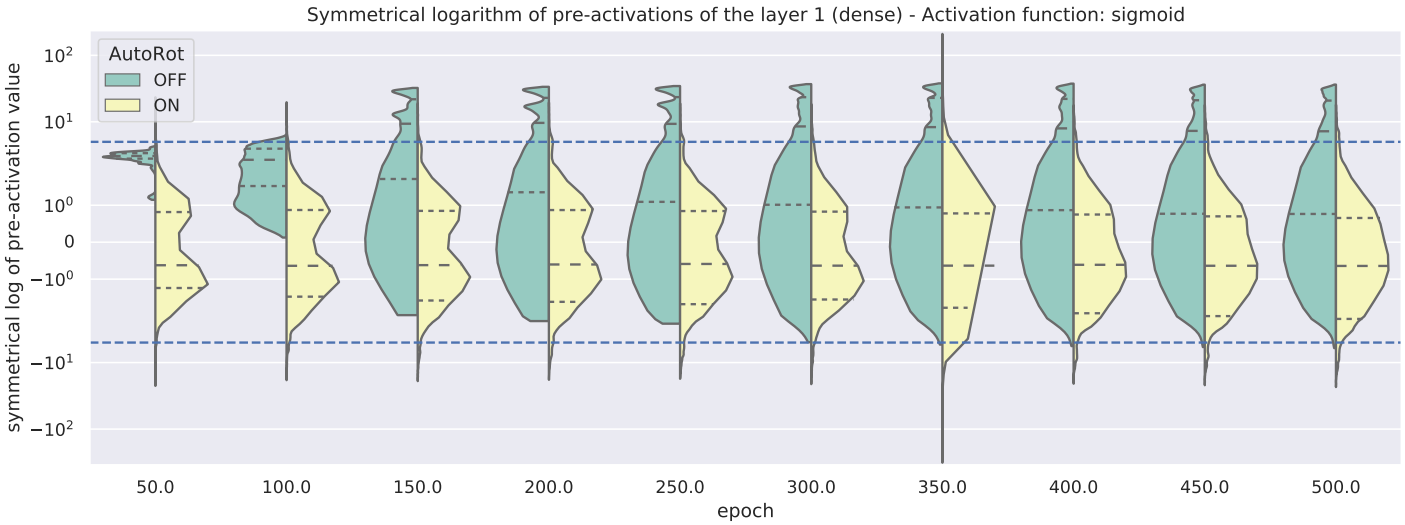
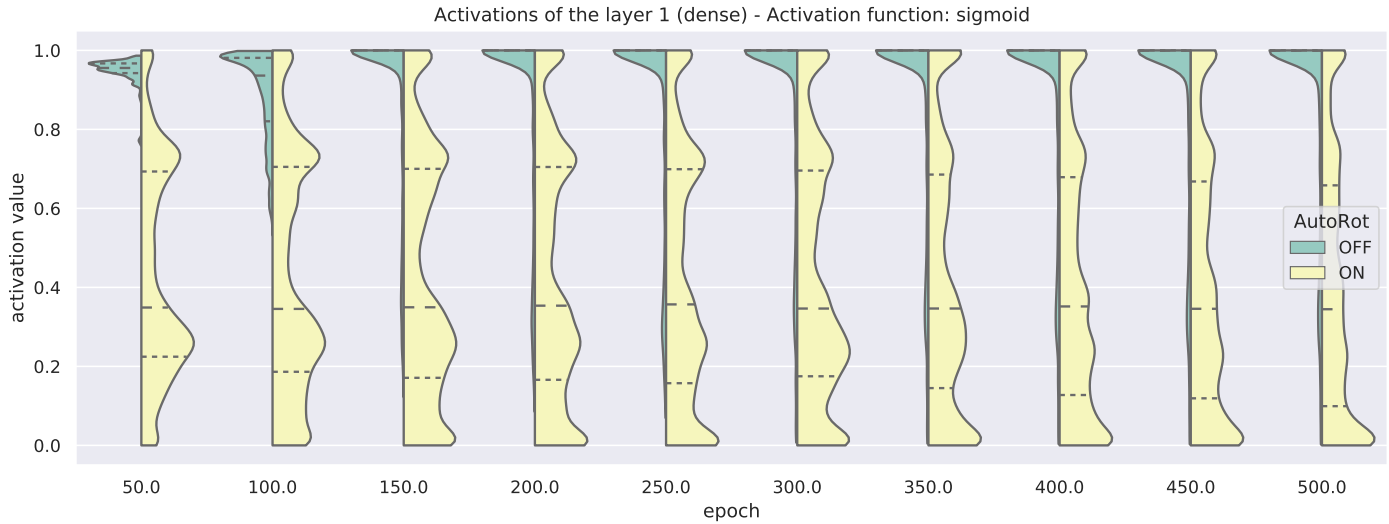


Figure 18: Loss curves after 30 executions. Architecture: 2, 50, **50**, 50, 1.

- Do we actually have more dynamic neurons? Let's delve into the layer 1.

Experiment D - Regression: Rastrigin function





Experiment D - Regression: Rastrigin function

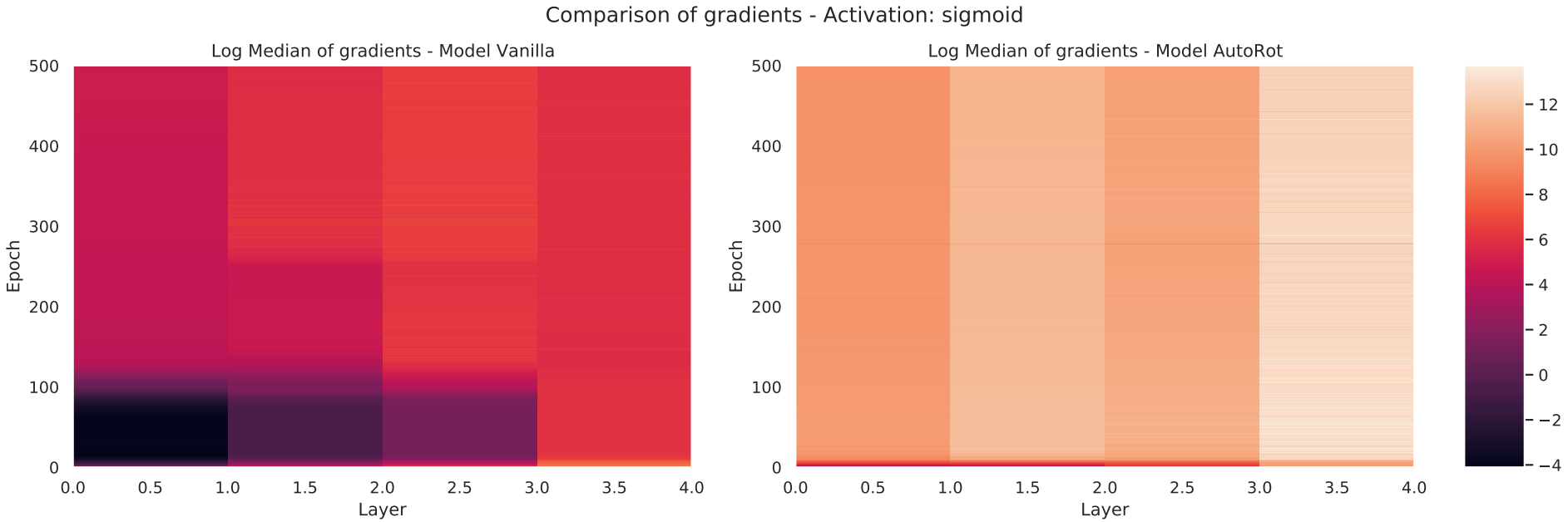


Figure 19: Gradients of the models' trainable weights after 500 epochs.

Experiment D - Regression: Rastrigin function

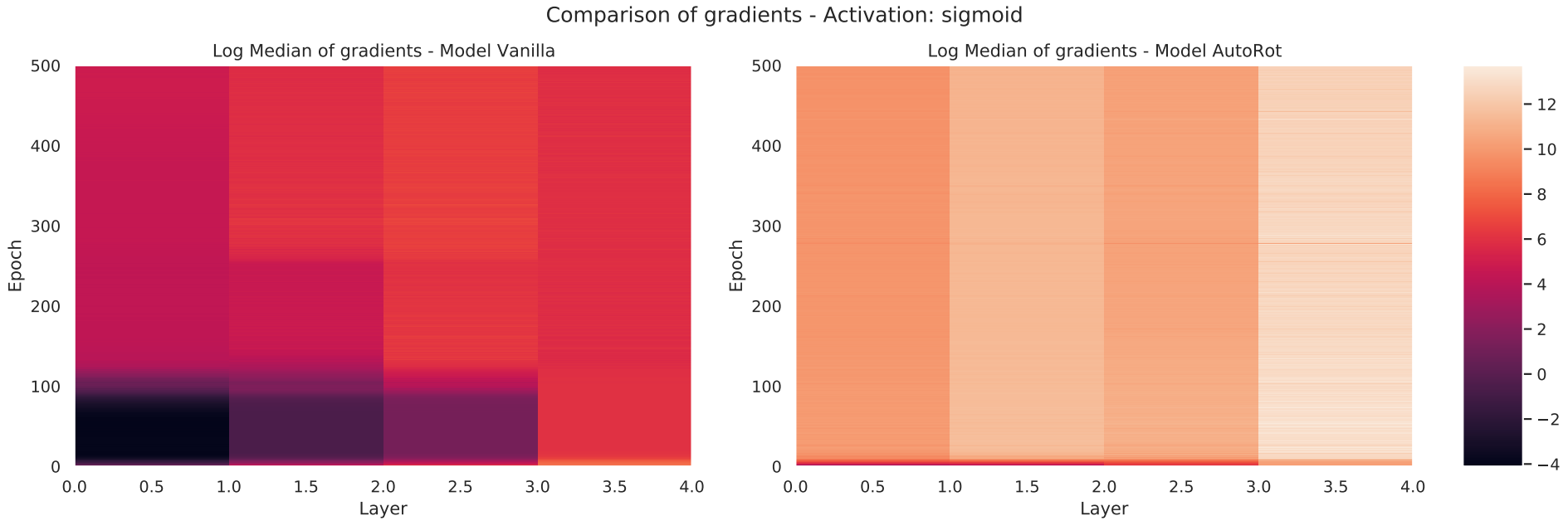


Figure 19: Gradients of the models' trainable weights after 500 epochs.

- Noticeable diminish of the Vanishing Gradient Problem (VGP)!
- How does this affect the final performance of the models?

Experiment D - Regression: Rastrigin function

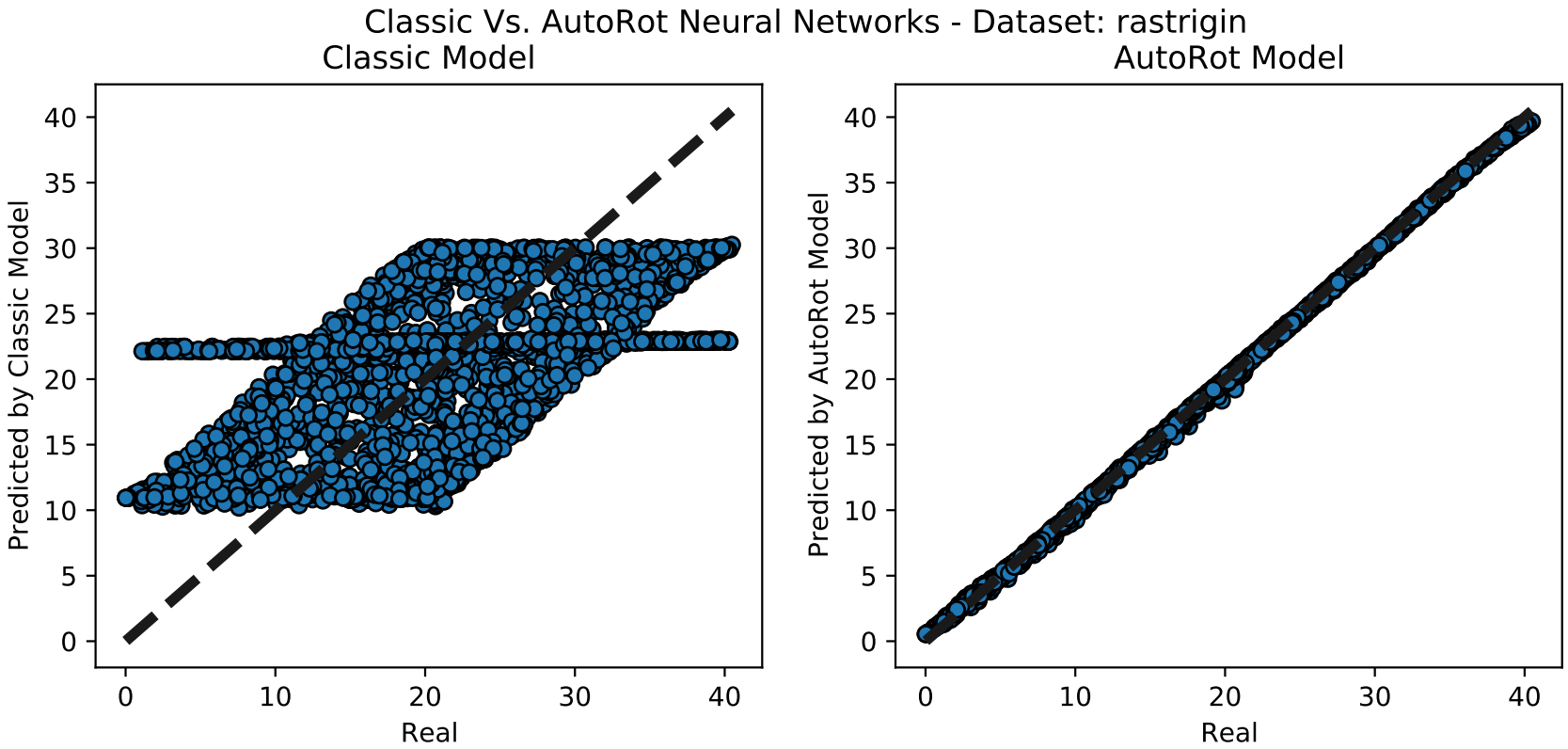


Figure 20: Prediction comparison on test samples. Scatter plots show the median prediction of 30 models after 500 epochs.

Outline

How to use ARP models?

1. Install the Keras ARP Library using: `pip install arpkeras`.
2. Import the Keras Auto-Rotating Dense layer definition using: `from ARPkeras import AutoRotDense`.
3. Choose the scalar value `L`. This is the **only hyperparameter** needed for the Auto-Rotating layers.
4. Set `xmin_lim` and `xmax_lim` according to your activation function and your data preprocessing.
5. You can define your Keras ARP layer using:

```
AutoRotDense(10, xmin_lim, xmax_lim, L, activation='sigmoid')
```

How to use ARP models?

1. Install the Keras ARP Library using: `pip install arpkeras`.
2. Import the Keras Auto-Rotating Dense layer definition using: `from ARPkeras import AutoRotDense`.
3. Choose the scalar value `L`. This is the **only hyperparameter** needed for the Auto-Rotating layers.
4. Set `xmin_lim` and `xmax_lim` according to your activation function and your data preprocessing.
5. You can define your Keras ARP layer using:

```
AutoRotDense(10, xmin_lim, xmax_lim, L, activation='sigmoid')
```

Now that we got the know the ARP theory, let's dive into the code!

Source code available at: `https://github.com/DanielSaromo/ARP`

Outline

Acknowledgments

- The ARNN paper is being developed with the strong support from Dr. Edwin Villanueva Talavera and Dr. Matias Valdenegro Toro.

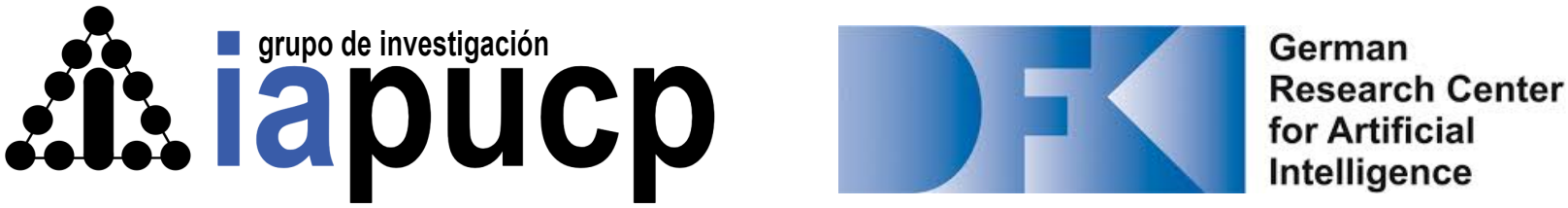


Figure 21: Research groups involved in the project: IA-PUCP and DFKI.

- Experiments executed at their GPU clusters.

Outline

References

[1] SAROMO, Daniel; VILLOTA, Elizabeth; VILLANUEVA, Edwin.
Auto-Rotating Perceptrons.
LXAI Workshop at NeurIPS. Vancouver, 2019.
Paper and Oral presentation.

[2] NIELSEN, Michael A.
Neural Networks and Deep Learning.
Determination Press, 2015.

[3] SAROMO, Daniel; BRAVO, Leonardo; VILLOTA, Elizabeth.
Smart Sensor Calibration with Auto-Rotating Perceptrons.
LXAI Workshop at ICML. Vienna, 2020.
Paper and Oral presentation.