Microsoft Research

Microsoft

Conditional Networks

Unifying Deep Neural Networks and Decision Forests

Y. Ioannou, D. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, **A. Criminisi**

The Decision Jungle Special Project,
a.k.a. Project WIN

Outline

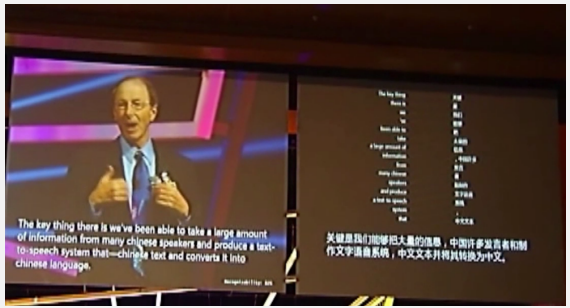
- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

Outline

- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

Deep learning and jungles in Microsoft products

Deep neural networks in speech recognition



… in Skype universal translator …

… see also work on ImageNet, project Adam …

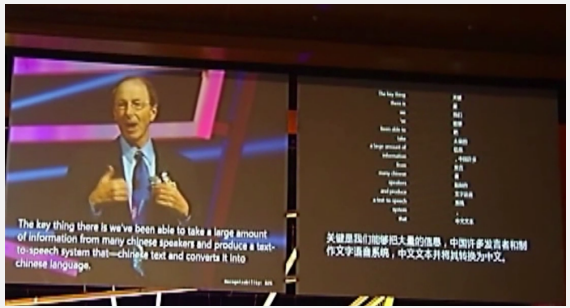
Decision forests in Kinect's body tracking

Decision **jungles** in HoloLens' gesture recognition

… also in Azure ML

Deep learning and jungles in Microsoft products

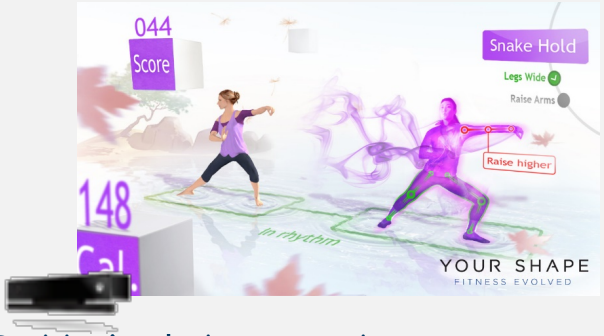
Deep neural networks in speech recognition



… in Skype universal translator …

… see also work on ImageNet, project Adam …

Decision forests in Kinect's body tracking

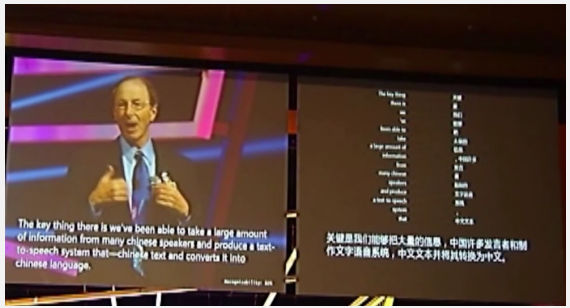


Decision **jungles** in secret projects

… also in Azure ML

Deep learning and jungles in Microsoft products

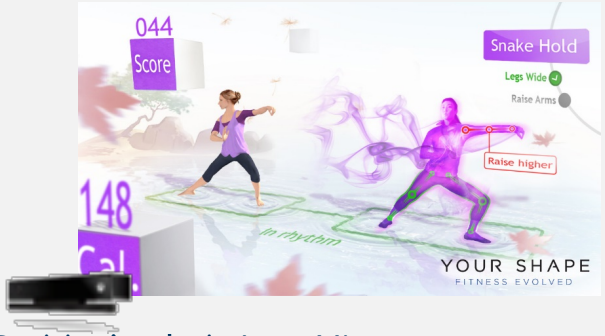
Deep neural networks in speech recognition



… in Skype universal translator …

… see also work on ImageNet, project Adam …

Decision forests in Kinect's body tracking

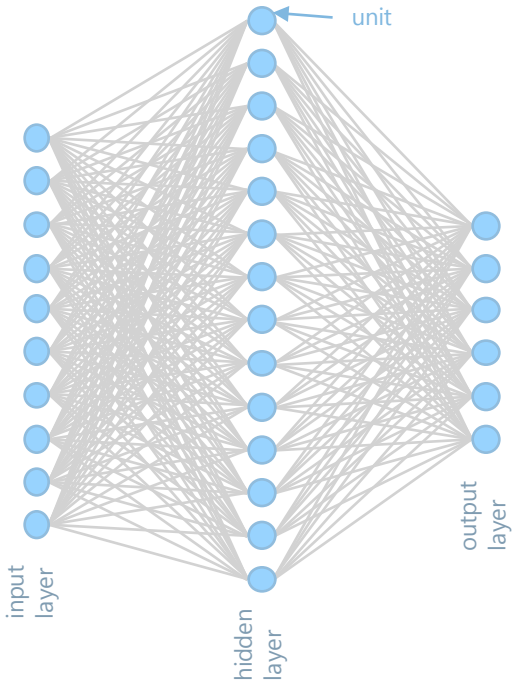


Decision **jungles** in Azure ML

… also in secret projects

Neural networks vs decision trees

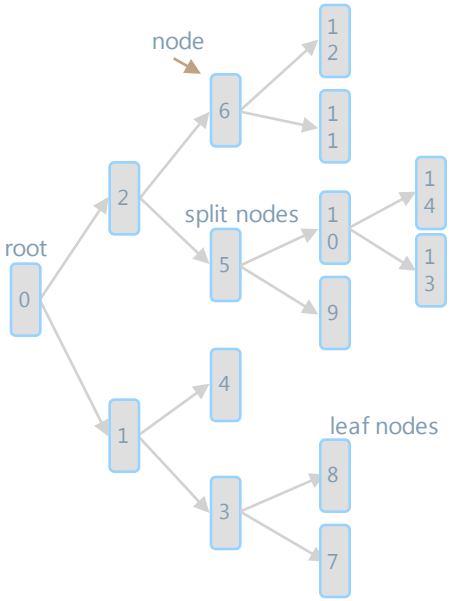
Multi-Layer Perceptron



Some practical differences

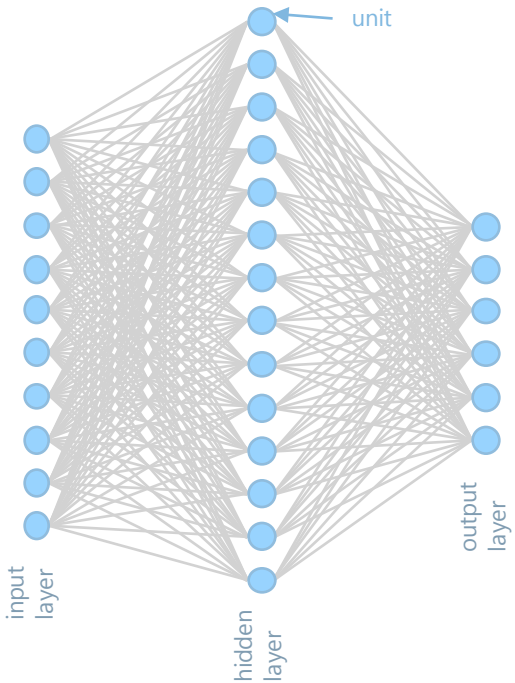
	ML Perceptrons	Decision Trees/DAGs
locality of computation	All "neurons" involved	Conditional computation
data transformation	Projections, non-linearities	Identity, copy
training algorithm	Gradient descent, back-prop	Greedy, randomized
structure	fixed ahead of time	learned and optimized
output	soft-max probabilities	histograms, probabilities
…	…	…

Decision Tree



Neural networks vs decision trees

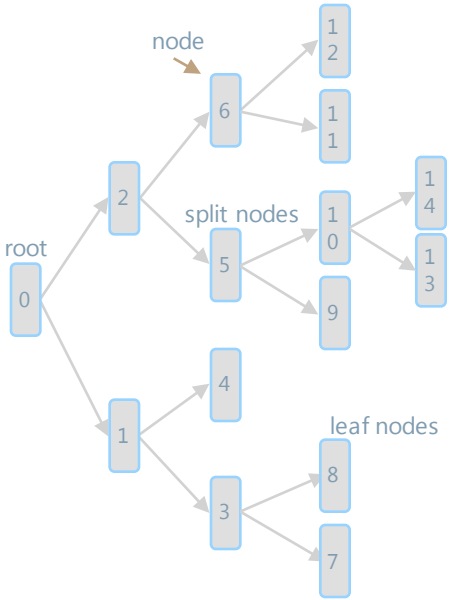
Multi-Layer Perceptron



Some practical differences

	ML Perceptrons	Decision Trees/DAGs
locality of computation	All "neurons" involved	Conditional computation
data transformation	Projections, non-linearities	Identity, copy
training algorithm	Gradient descent, back-prop	Greedy, randomized
structure	fixed ahead of time	learned and optimized
output	soft-max probabilities	histograms, probabilities
…	…	…

Decision Tree



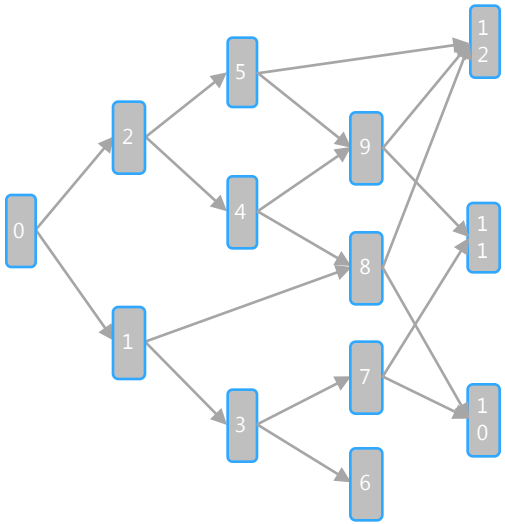
Outline

- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

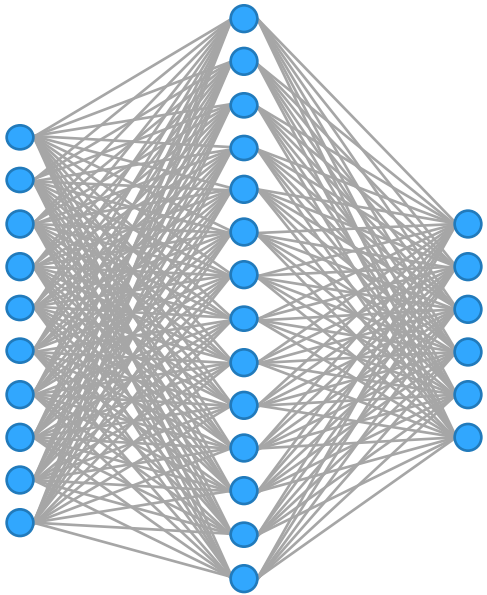
Representing trees as perceptrons

Proposition. *Any DAG can be represented as a two-layer perceptron.*

Directed Acyclical Graph (DAG)



Two-Layer Perceptron

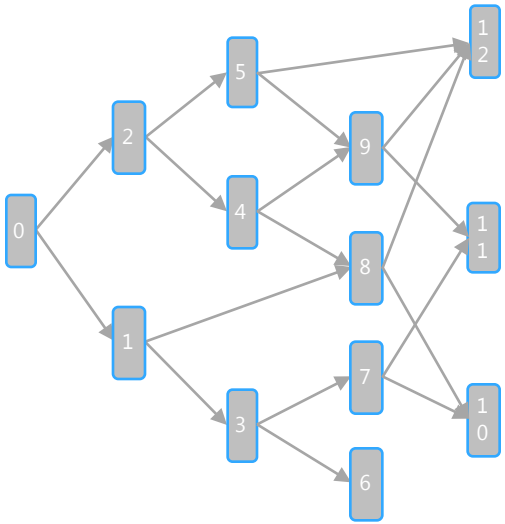


Representing trees as perceptrons

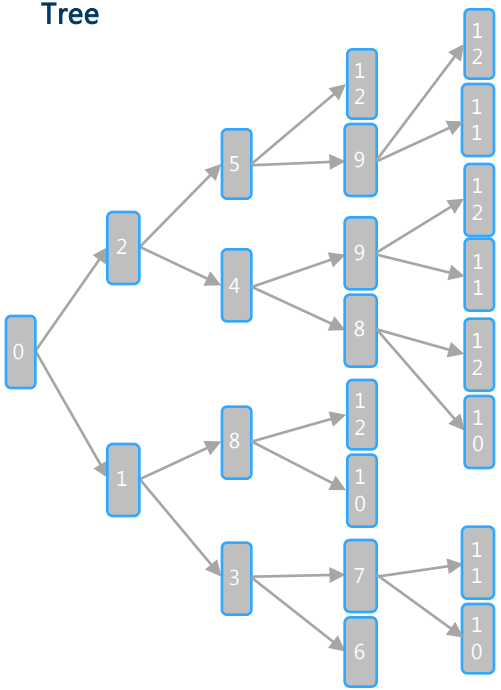
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 1 – any DAG can be represented as a tree

Directed Acyclical Graph



Tree



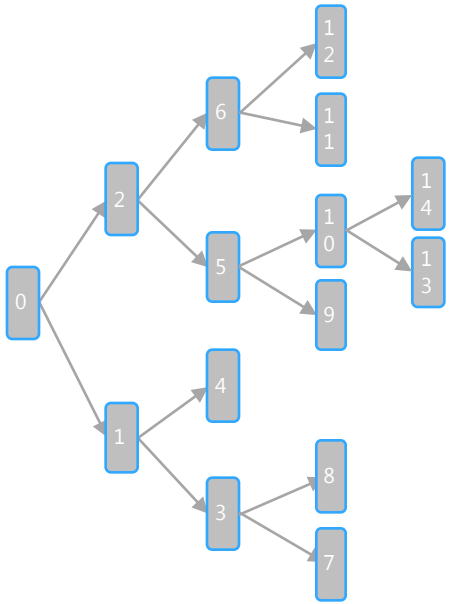
Similarly, n-ary trees can be transformed into binary trees.

Representing trees as perceptrons

Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree

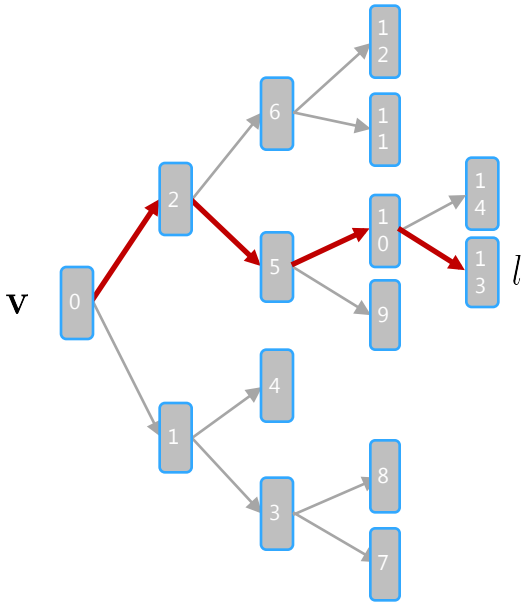


Representing trees as perceptrons

Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



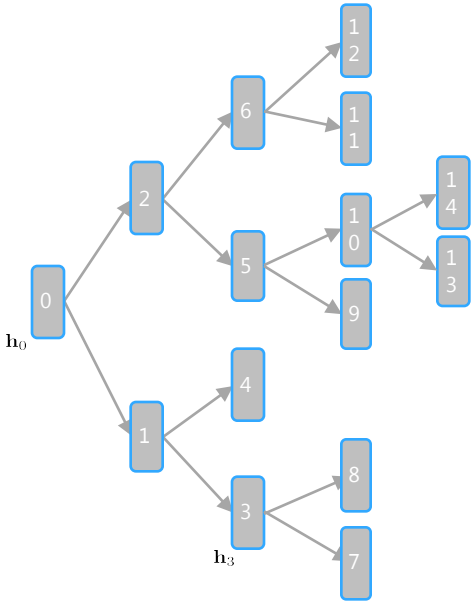
$$l = tree(\mathbf{v})$$

Representing trees as perceptrons

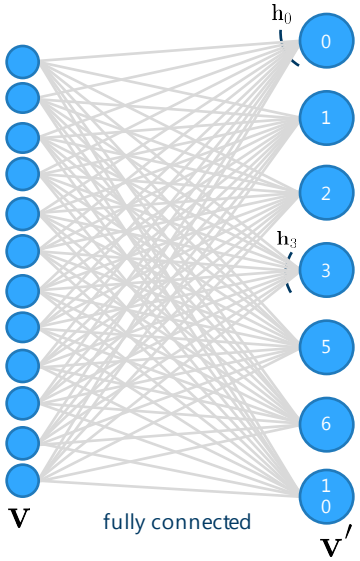
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



One-layer Perceptron

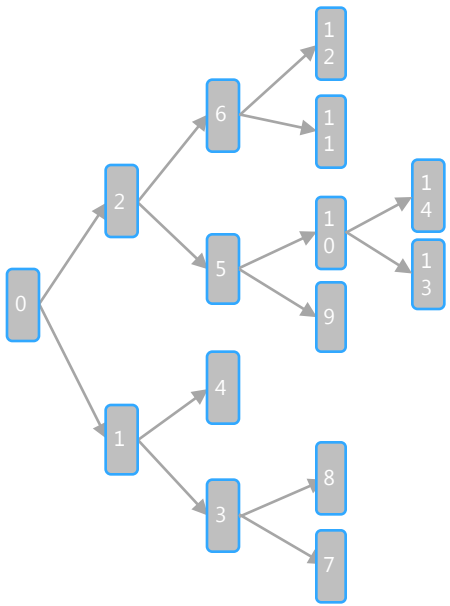


Representing trees as perceptrons

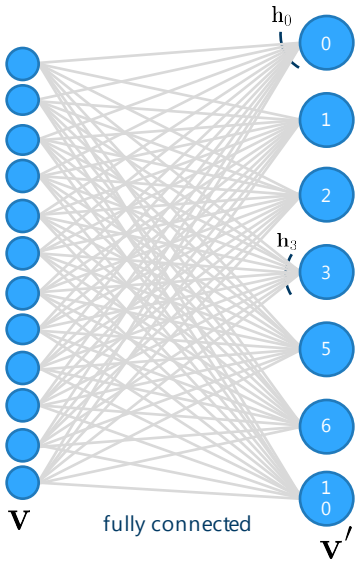
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



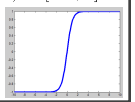
One-layer Perceptron



Hidden activations

$$v'_j = 2\,(\sigma(\mathbf{v}\cdot\mathbf{h}_j;0,\beta)-0.5)\in[-1,1]$$

$$\sigma(x;\mu,\beta)=\frac{1}{1+e^{\frac{\mu-x}{\beta}}}$$

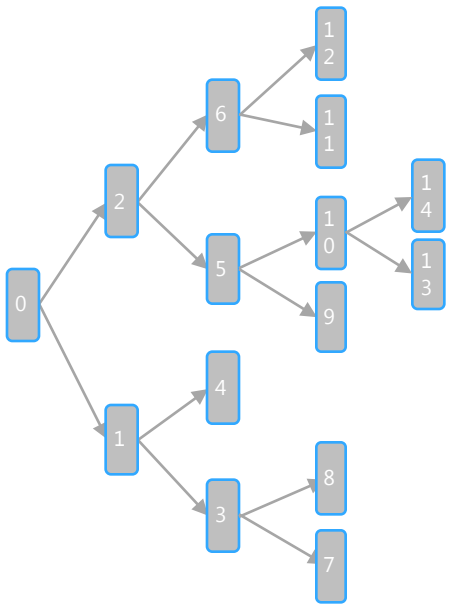


Representing trees as perceptrons

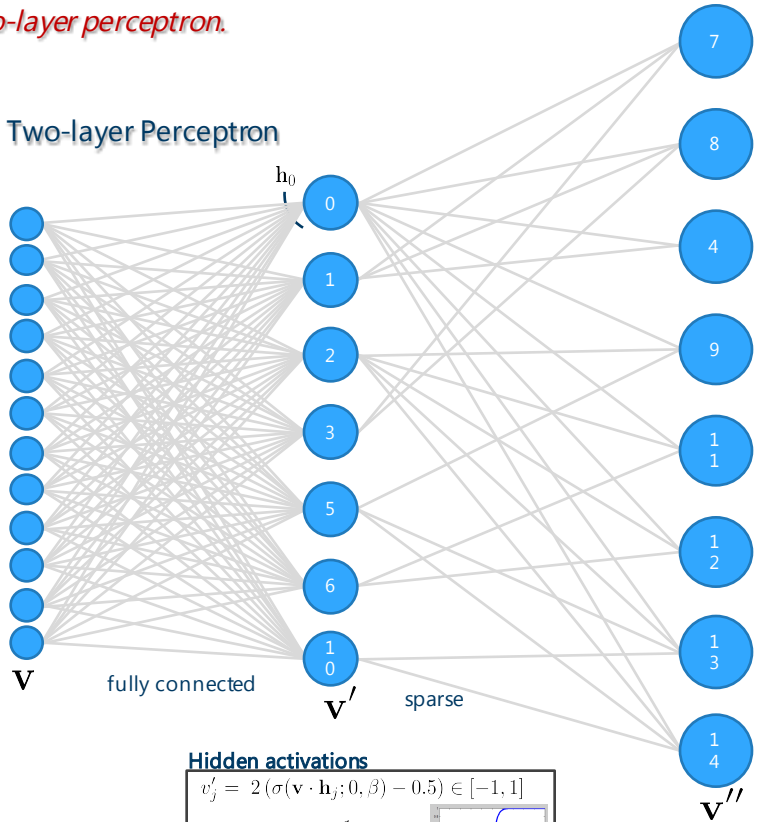
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



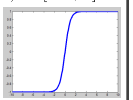
Two-layer Perceptron



Hidden activations

$$v'_j = 2\,(\sigma(\mathbf{v}\cdot\mathbf{h}_j;0,\beta) - 0.5) \in [-1,1]$$

$$\sigma(x;\mu,\beta) = \frac{1}{1+e^{\frac{\mu-x}{\beta}}}$$

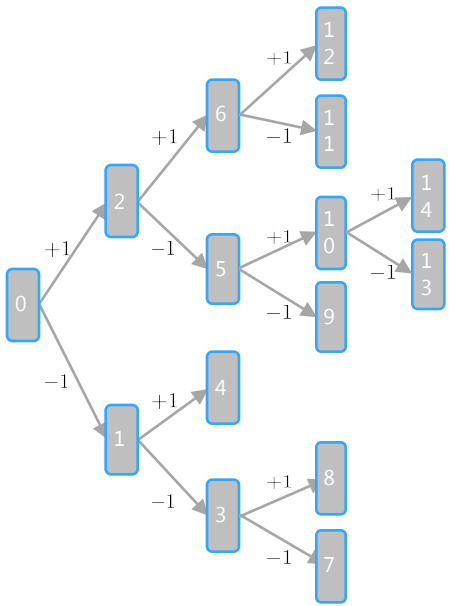


Representing trees as perceptrons

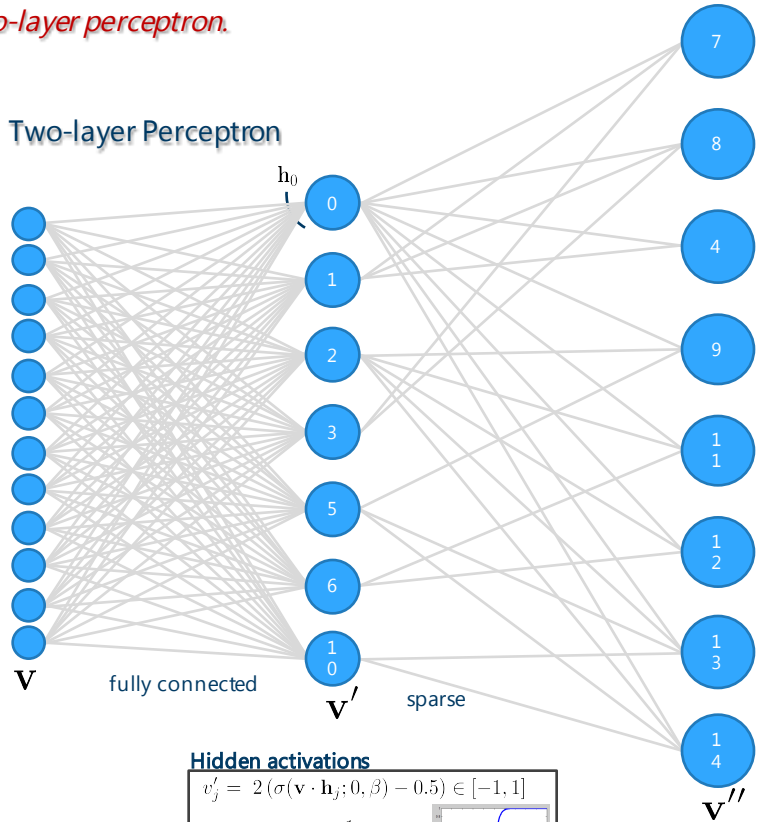
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



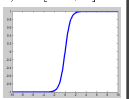
Two-layer Perceptron



Hidden activations

$$v'_j = 2\,(\sigma(\mathbf{v}\cdot\mathbf{h}_j;0,\beta) - 0.5) \in [-1,1]$$

$$\sigma(x;\mu,\beta) = \frac{1}{1+e^{\frac{\mu-x}{\beta}}}$$

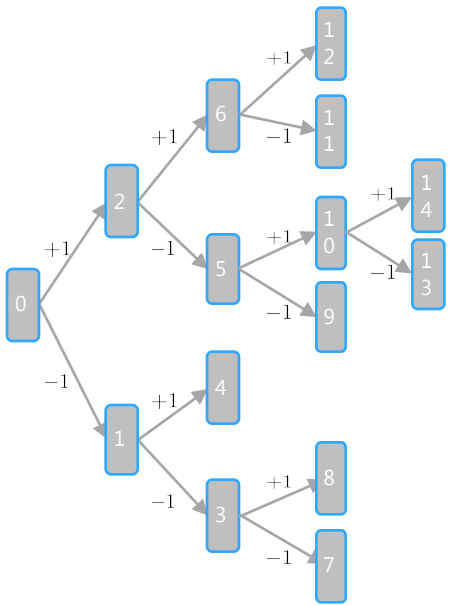


Representing trees as perceptrons

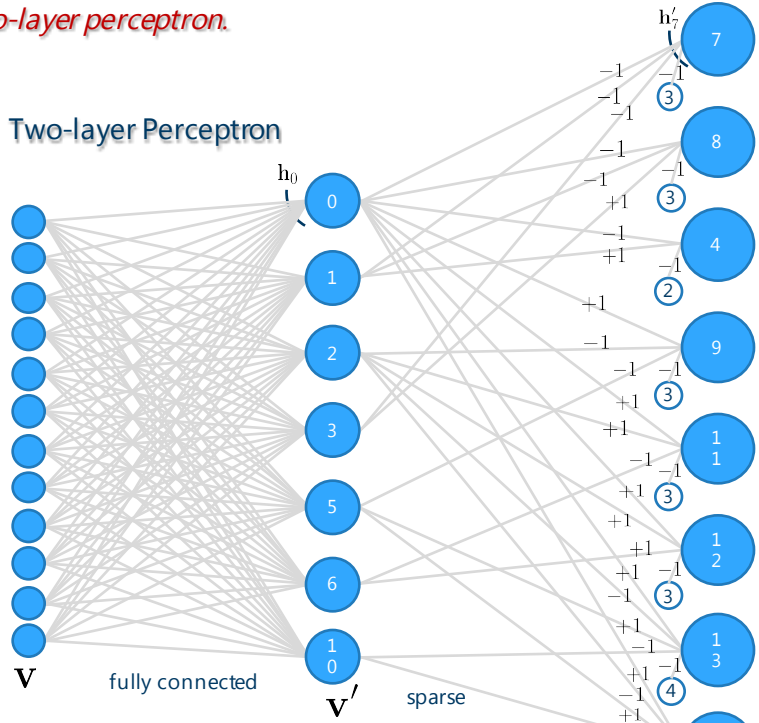
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



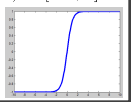
Two-layer Perceptron



Hidden activations

$$v'_j = 2\left(\sigma(\mathbf{v}\cdot\mathbf{h}_j;0,\beta)-0.5\right)\in[-1,1]$$

$$\sigma(x;\mu,\beta)=\frac{1}{1+e^{\frac{\mu-x}{\beta}}}$$



Output activations

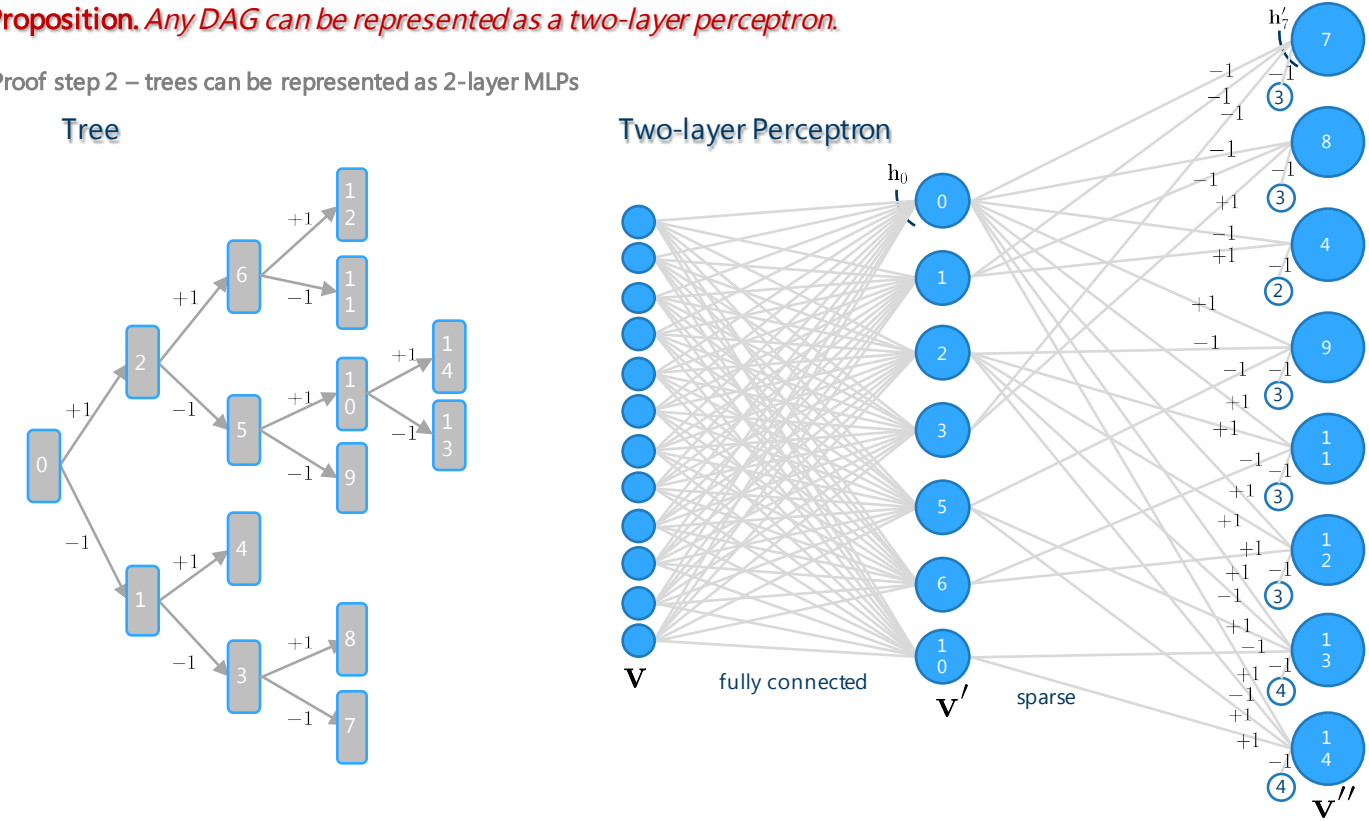
$$v''_j = \sigma(\mathbf{v}'\cdot\mathbf{h}'_j;0,\beta')\in[0,1]$$

$d_j = d(l_j)$ Bias is depth of leaf

Representing trees as perceptrons

Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

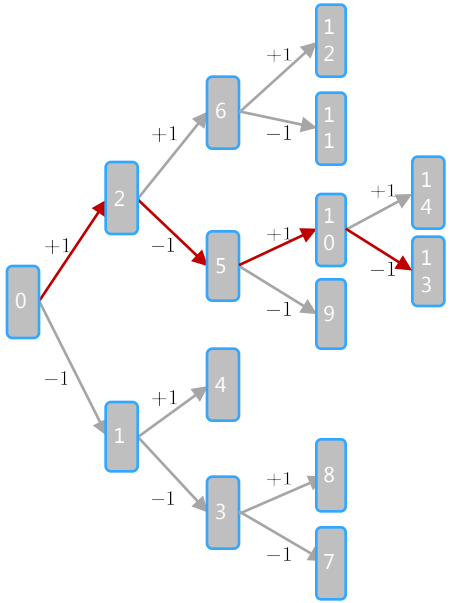


Representing trees as perceptrons

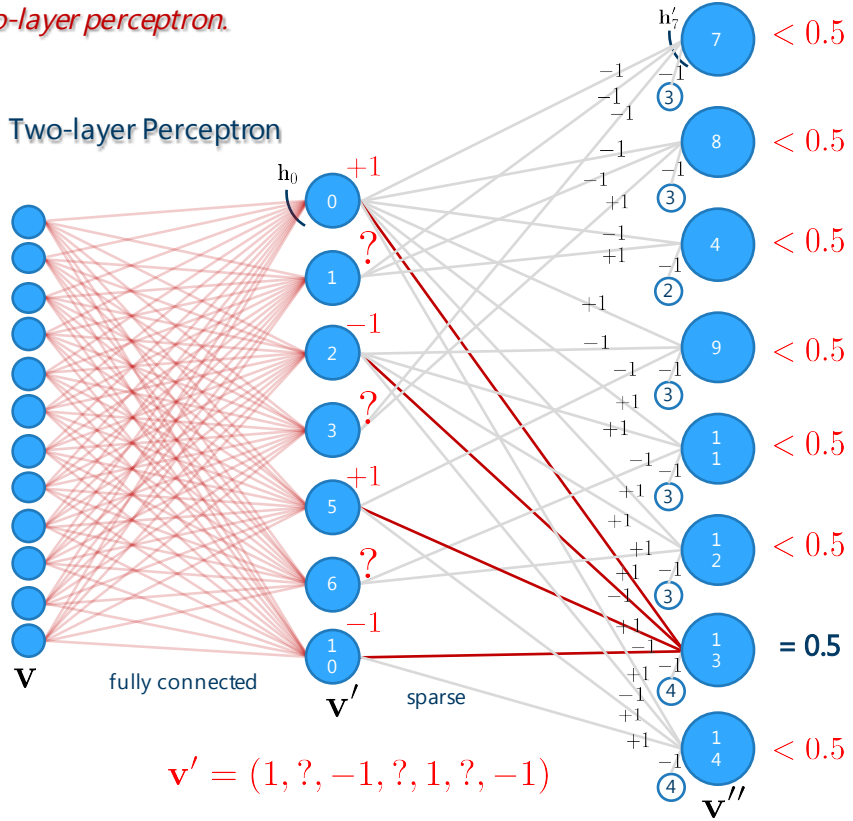
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



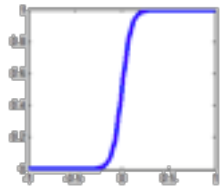
Two-layer Perceptron



$$\mathbf{v}' = (1, ?, -1, ?, 1, ?, -1)$$

$$v''_j = \sigma(\mathbf{v}' \cdot \mathbf{h}'_j; 0, \beta') \in [0, 1]$$

$$\leq \frac{1}{1 + \exp(-\frac{1}{b})}$$



$$\sigma(x; \mu, \beta) = \frac{1}{1 + e^{\frac{\mu - x}{\beta}}}$$

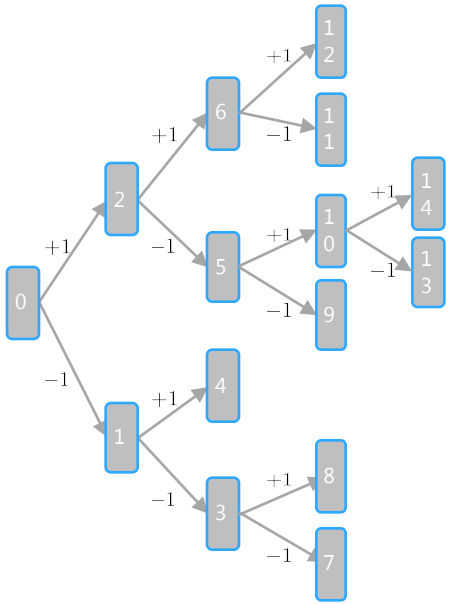
Representing trees as perceptrons

(script_equivalence_test_4.m)

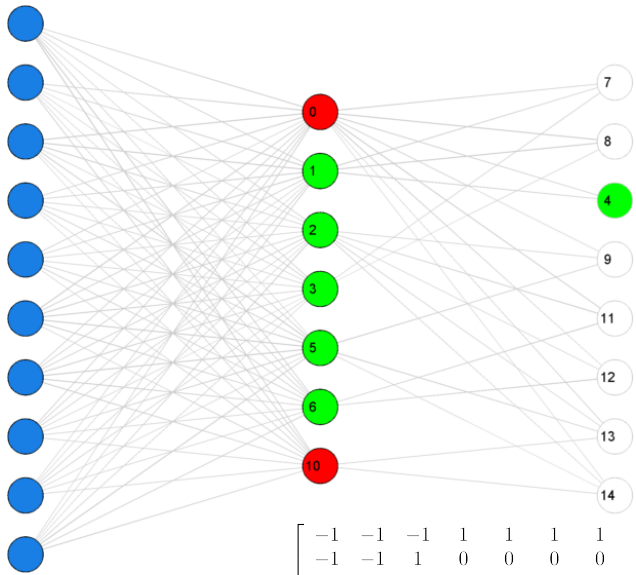
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



Conditional Perceptron (i.e. a special 2-layer MLP)



$$v'_j = 2\left(\sigma(\mathbf{v}\cdot\mathbf{h}_j;0,\beta)-0.5\right)\in[-1,1]$$

$$\beta = 0.01$$

$$v''_j = \sigma(\mathbf{v}'\cdot\mathbf{h}'_j;0,\beta')\in[0,1]$$

$$\beta' = 0.01$$

$$\mathrm{P}'' = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ -3 & -3 & -2 & -3 & -3 & -3 & -4 & -4 \end{bmatrix}^\top$$

A tree/Dag can be thought of as a sparsified version of a 2-layer perceptron

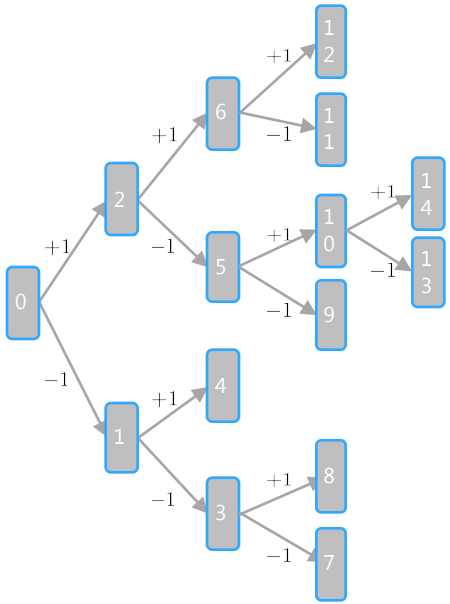
Representing trees as perceptrons

(script_equivalence_test_4.m)

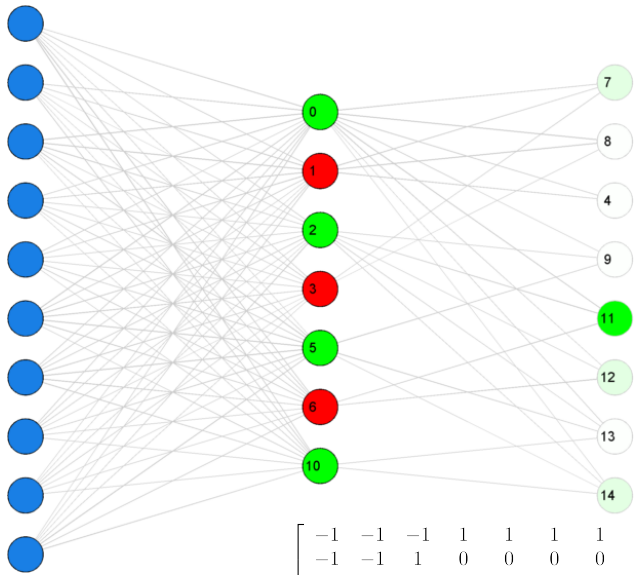
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



Conditional Perceptron (i.e. a special 2-layer MLP)



$$v'_j = 2\left(\sigma(\mathbf{v}\cdot\mathbf{h}_j;0,\beta)-0.5\right)\in[-1,1]$$

$$\beta = 0.01$$

$$v''_j = \sigma(\mathbf{v}'\cdot\mathbf{h}'_j;0,\beta')\in[0,1]$$

$$\beta' = 0.7$$

$$\mathrm{P}'' = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ -3 & -3 & -2 & -3 & -3 & -3 & -4 & -4 \end{bmatrix}^\top$$

A tree/Dag can be thought of as a sparsified version of a 2-layer perceptron

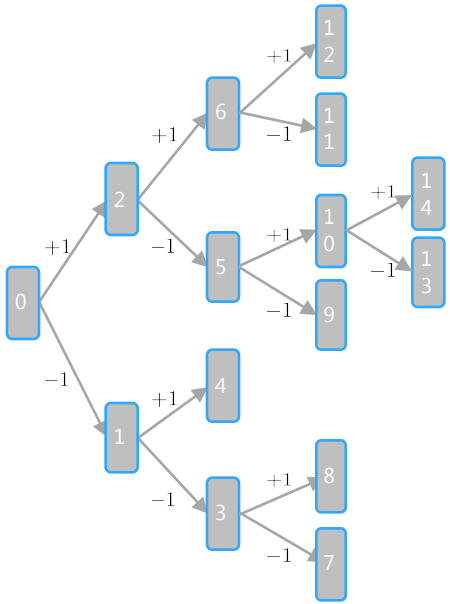
Representing trees as perceptrons

(script_equivalence_test_4.m)

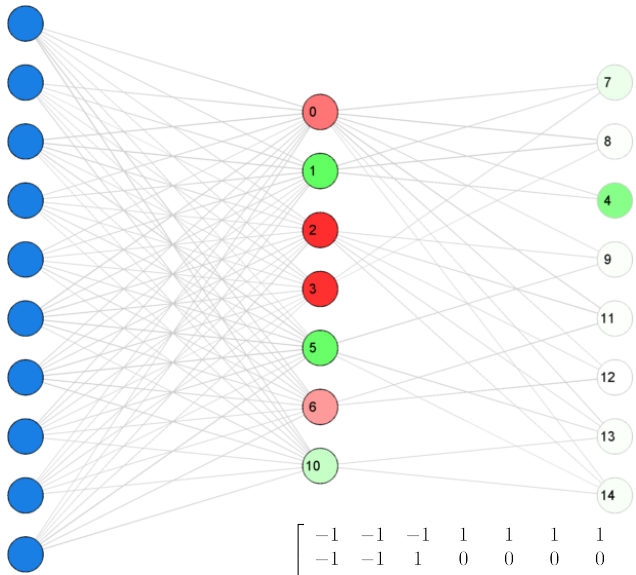
Proposition. *Any DAG can be represented as a two-layer perceptron.*

Proof step 2 – trees can be represented as 2-layer MLPs

Tree



Conditional Perceptron (i.e. a special 2-layer MLP)



$$v'_j = 2\,(\sigma(\mathbf{v}\cdot\mathbf{h}_j;0,\beta)-0.5)\in[-1,1]$$

$$\beta = 0.7$$

$$v''_j = \sigma(\mathbf{v}'\cdot\mathbf{h}'_j;0,\beta')\in[0,1]$$

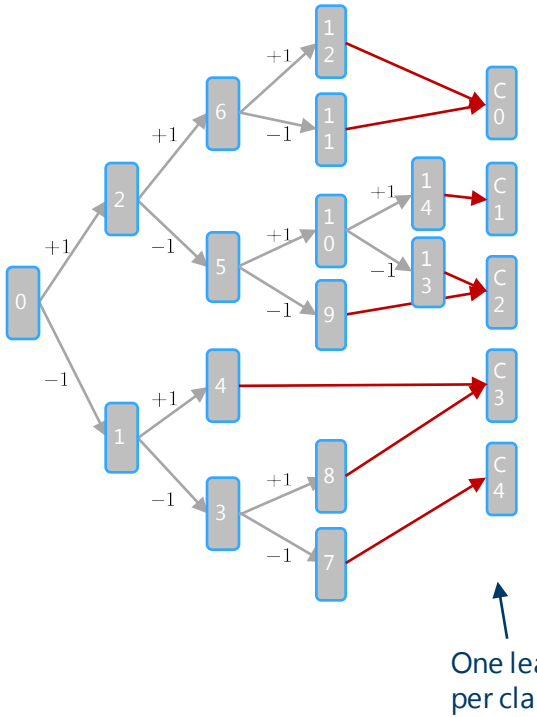
$$\beta' = 0.7$$

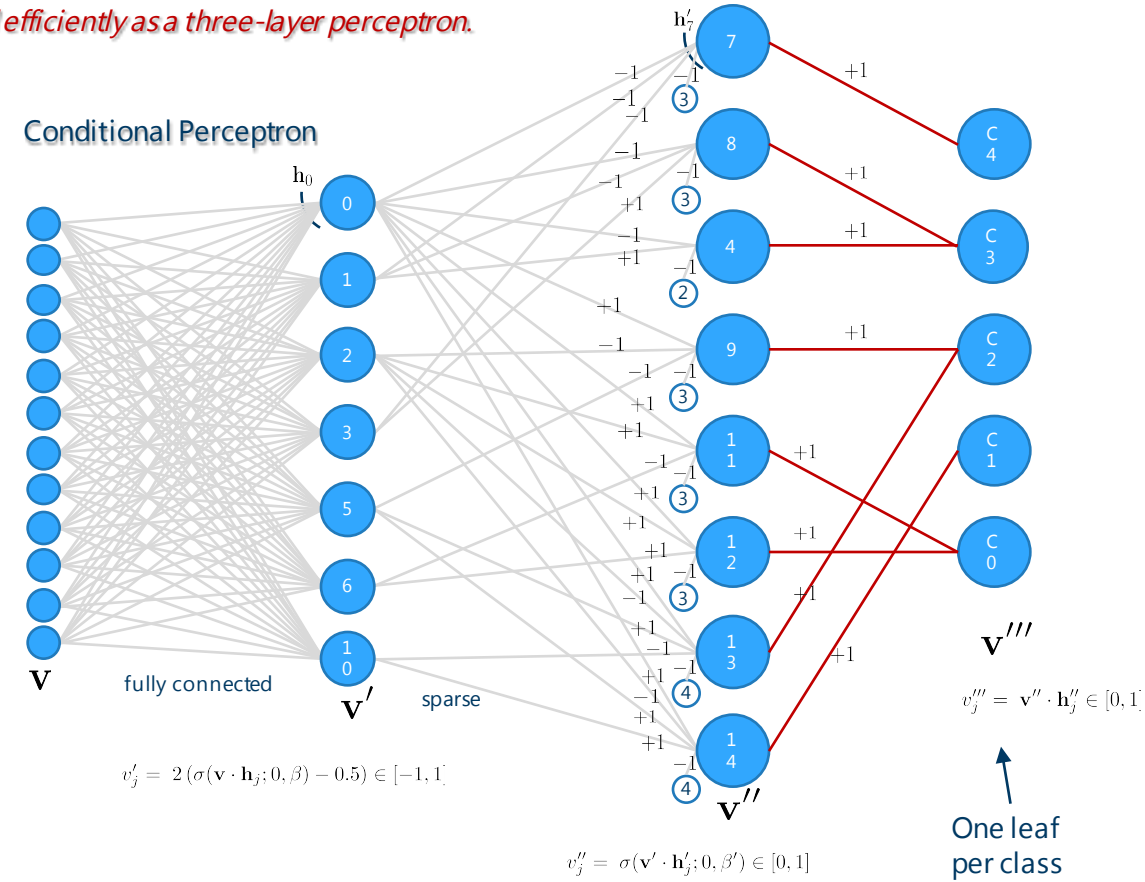
$$\mathrm{P}'' = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ -3 & -3 & -2 & -3 & -3 & -3 & -4 & -4 \end{bmatrix}^{\top}$$

A tree/Dag can be thought of as a sparsified version of a 2-layer perceptron

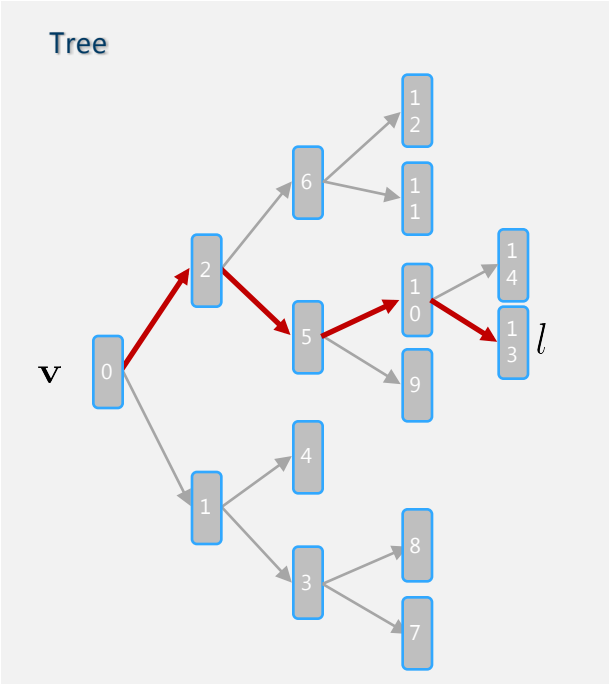
Representing trees as perceptrons

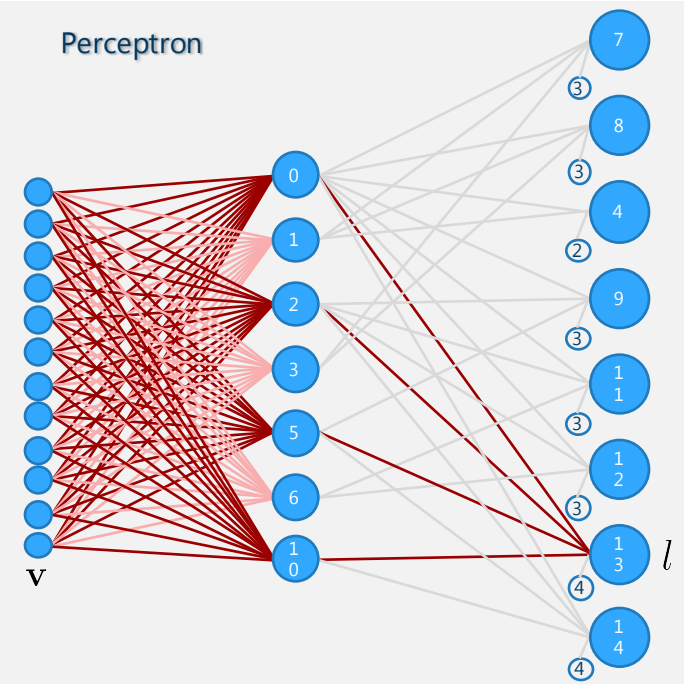
Corollary. *Some useful DAGs can be represented efficiently as a three-layer perceptron.*





Representing trees as perceptrons





Functional equivalence

No computational equivalence

$l = tree(\mathbf{v})$

Conditional computation

$l = perceptron(\mathbf{v})$

No conditional computation

Goal of conditional networks: adding conditional computation to neural networks

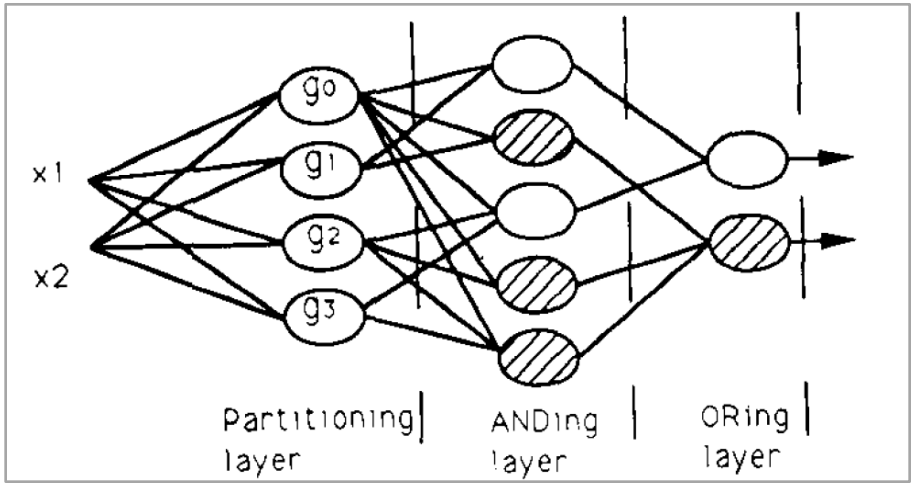
Representing trees as perceptrons

Proposition. *Any DAG can be represented as a two-layer perceptron.*

Not a new idea. Rediscovering and revisiting the work in

[Entropy Nets: From Decision Trees to Neural Networks.
I. K. Sethi. Proc. of the IEEE (Vol. 78, Issue 10). Oct 1990]

[Casting Random Forests as Artificial Neural Networks
(and Profiting from It). J. Welbl. Proc. of GCPR. 2014]



"restructuring a decision tree as a multilayer neural network"
"...automatic tree generation...incremental learning..."
"neural network design through decision tree mappings..."
"soft decision making"
Training an MLP via entropy-based loss. What about training a tree with back-prop instead?
"...**far fewer connections**..." -> greater efficiency

"initializing an MLP from a random forest works better than initializing at random..."

Representing trees as perceptrons

Tree / DAG $\Rightarrow$ (=) Sparse Neural Network 

Deep Neural Network $\Rightarrow$ (≈) Tree / DAG

approximate

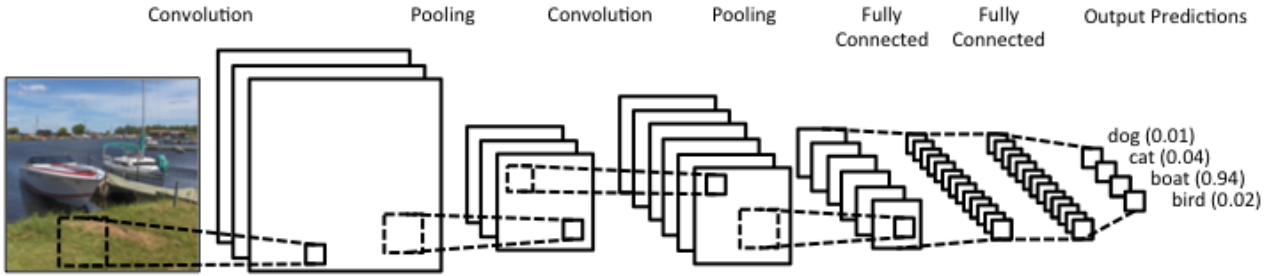
What do we gain?
What do we lose?

Outline

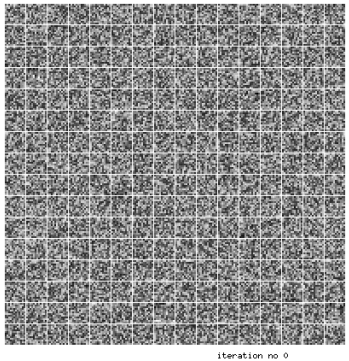
- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

Structured activation sparsity and data routing

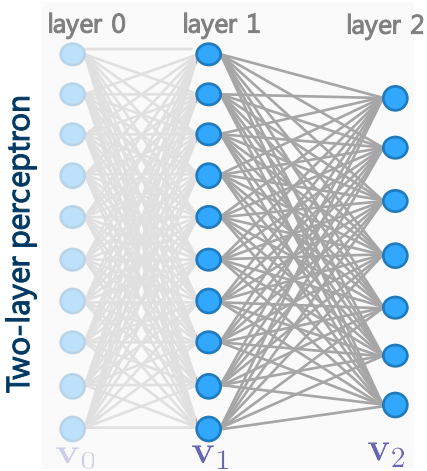
A deep convolutional neural network

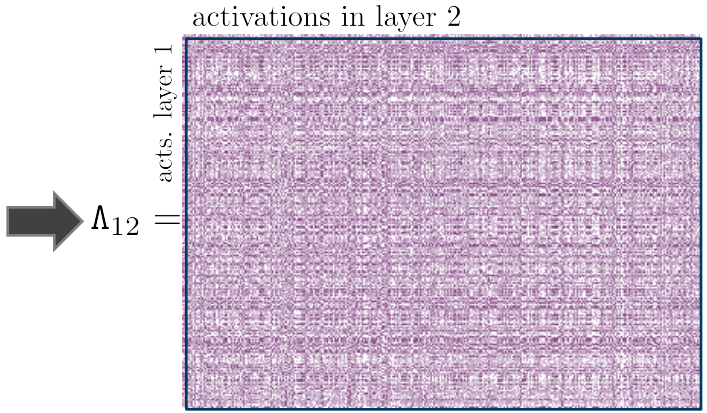


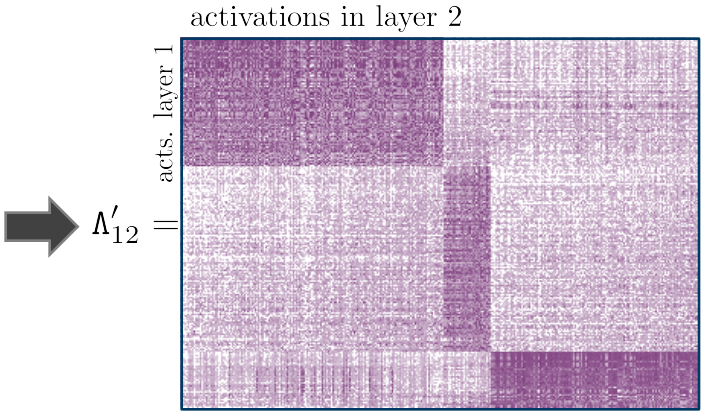
Representation learning: selecting optimal convolution kernels

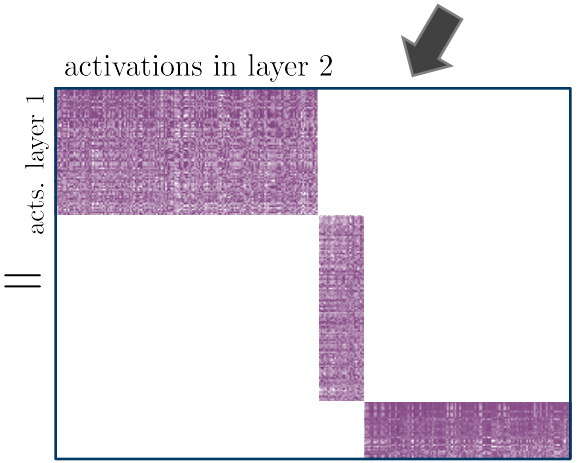


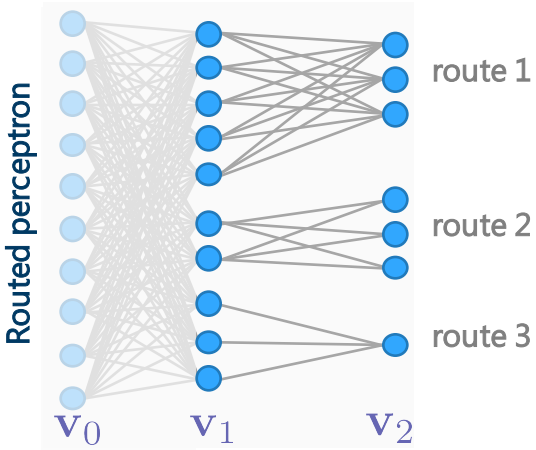
Structured activation sparsity and data routing











"Cells that fire together, wire together". [D. Hebb. **The organization of Behavior.** Wiley and Sons. 1949]

Trees and neural networks

What we have learnt so far

- A decision tree/DAG is a **special subclass** of multi-layer perceptrons.
- In general, at test time decision trees/DAGs perform **fewer computations** than MLPs because of data routing and conditional computation.
- However, hard data routing may yield **lower accuracy**. finally,
- Trees are **hard to train** in a globally optimal manner.

Therefore we introduce **conditional networks**

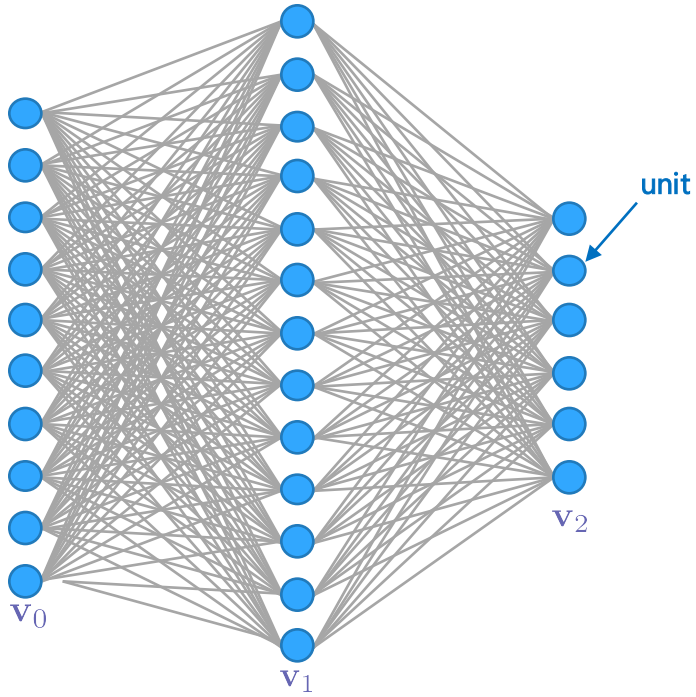
A *conditional network* is a new model that generalizes both (deep) neural networks and decision forests/jungles. Conditional networks marry efficient test-time **conditional computation** with the **accuracy** and **global trainability** of neural networks.

Outline

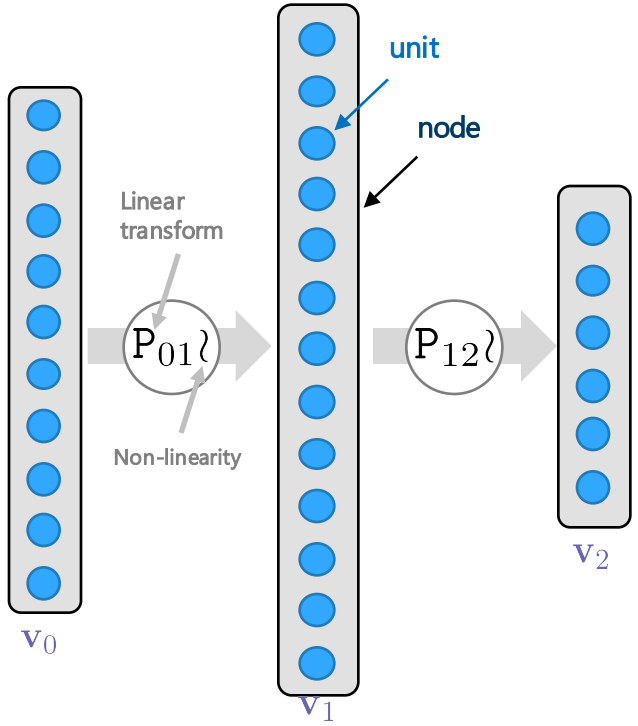
- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

A new, compact graphical notation - MLP

Old notation for a 2-layer perceptron



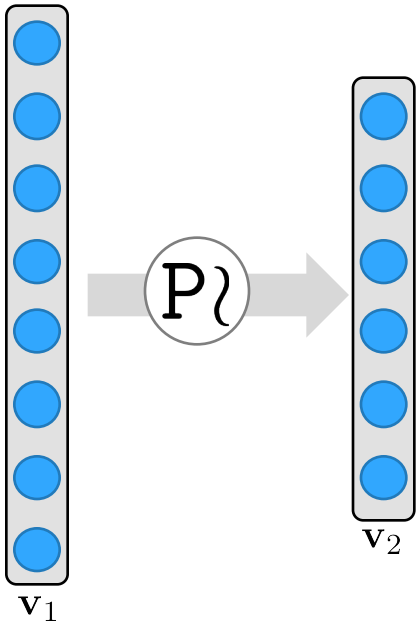
New notation



The non-linearity could be a ReLU, sigmoid, tanh…

A new, compact graphical notation - MLP

New notation



Algebraically

$$\mathbf{v}_2 = \sigma\left(\mathrm{P}\mathbf{v}_1\right)$$

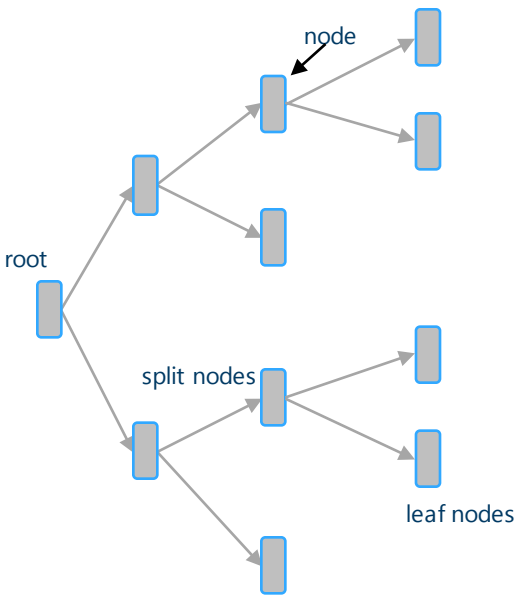
Generic non-linearity

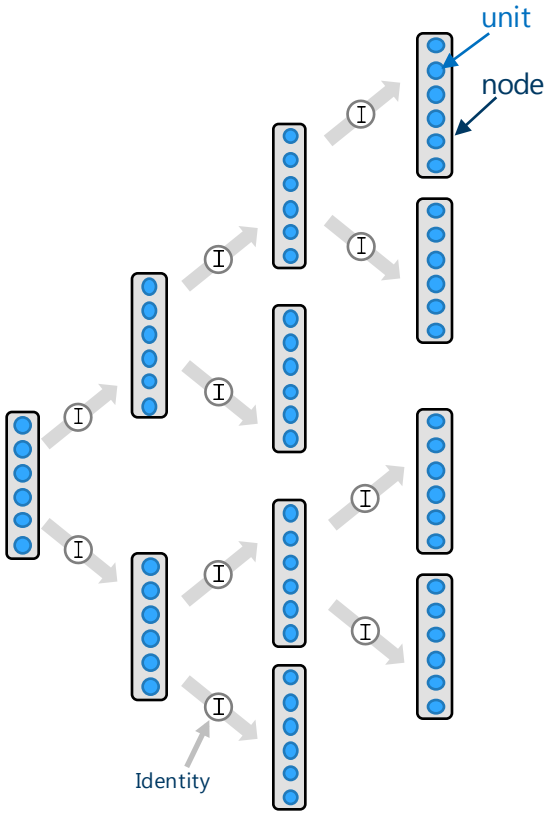
$$\mathrm{P} = \begin{bmatrix} - & \mathbf{h}_1 & - \\ - & \mathbf{h}_2 & - \\ - & \mathbf{h}_3 & - \\ - & \mathbf{h}_4 & - \\ - & \mathbf{h}_5 & - \\ - & \mathbf{h}_6 & - \end{bmatrix}$$

The bias is incorporated within the Homogeneous notation for simplicity

Old notation

A new, compact graphical notation - Tree



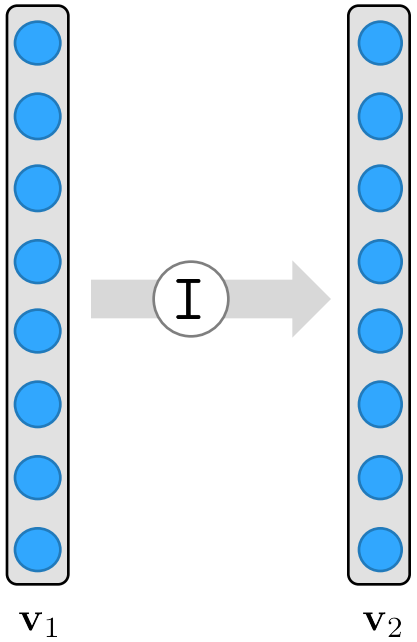


In a tree/DAG data is **moved** about, unchanged

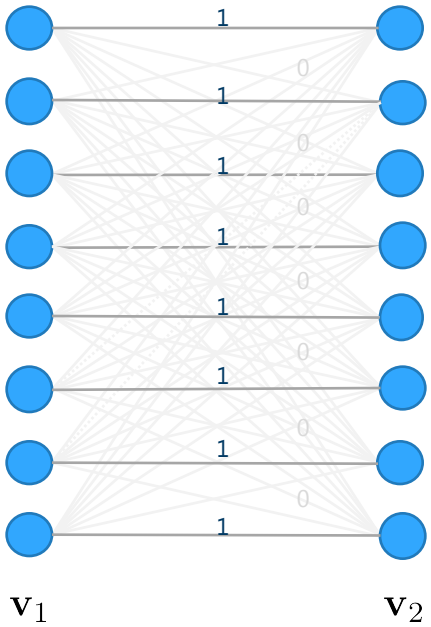
We need one more ingredient to make it all work…

A new, compact graphical notation

New notation



Old notation

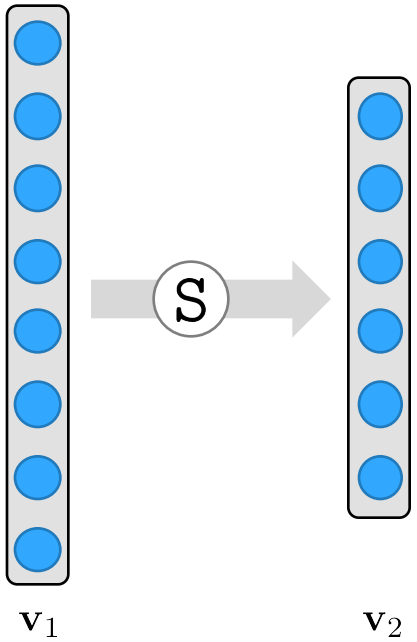


Algebraically

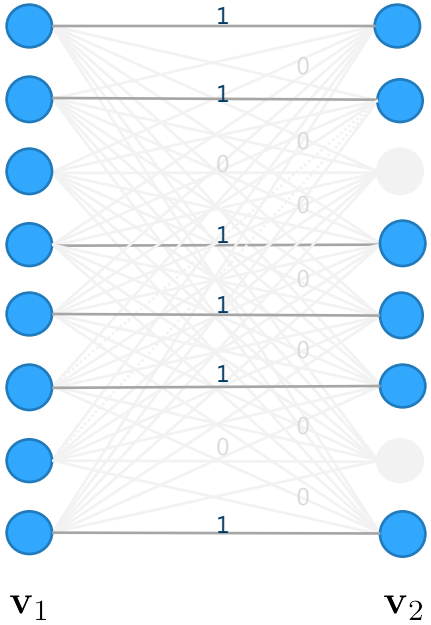
$$\mathbf{v}_2 = \mathbf{I}\mathbf{v}_1$$

A new, compact graphical notation

New notation



Old notation

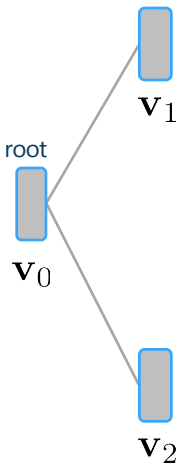


Algebraically

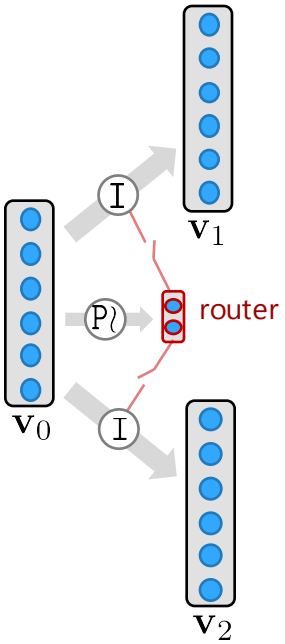
$$\mathbf{v}_2 = \mathrm{S}\mathbf{v}_1$$

A new, compact graphical notation - tree

Old notation



New notation



Algebraically

Data transformation

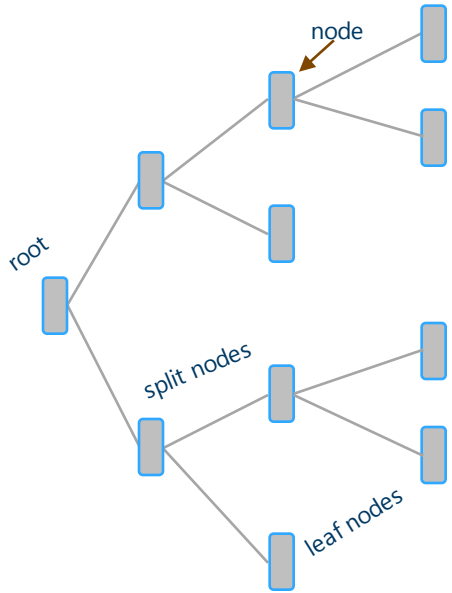
$$\mathbf{v}_i = r_i \, \mathtt{I} \, \mathbf{v}_0 \qquad i \in \{1, 2\}$$

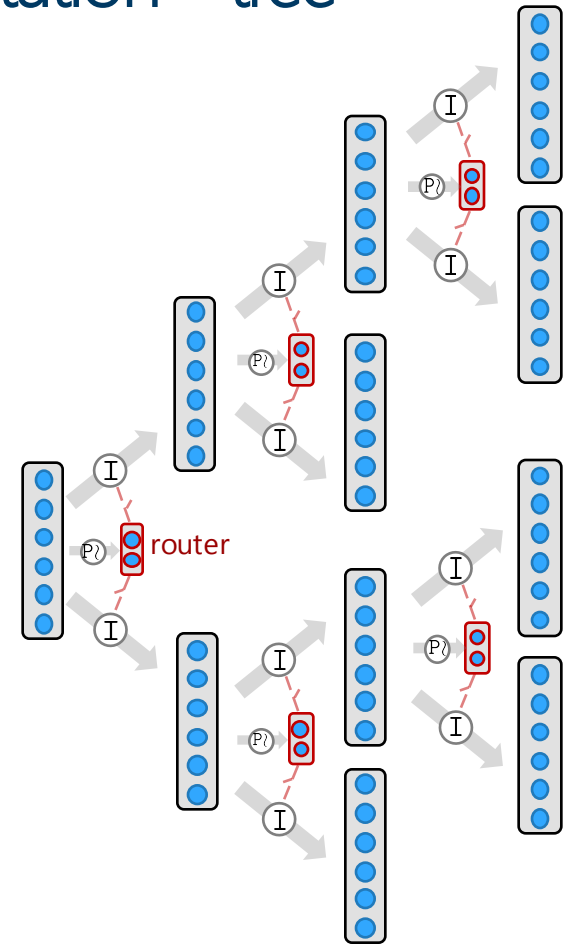
Routing weights (with ReLU non-linearity)

$$\mathbf{r} = \sigma(\mathtt{P} \, \mathbf{v}_0) \qquad r_i \in [0, 1]$$

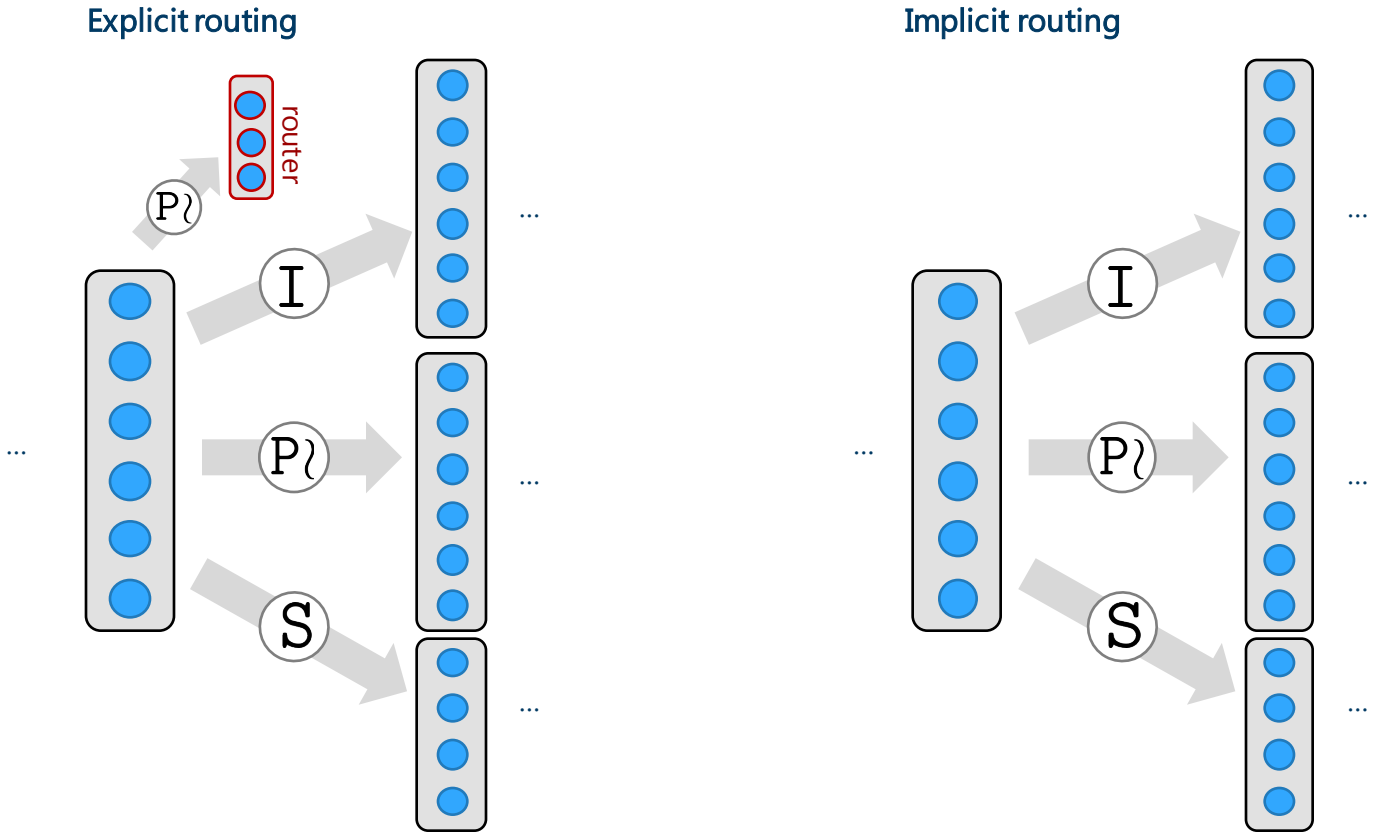
The router can produce N>=2 outputs.

A new, compact graphical notation - tree

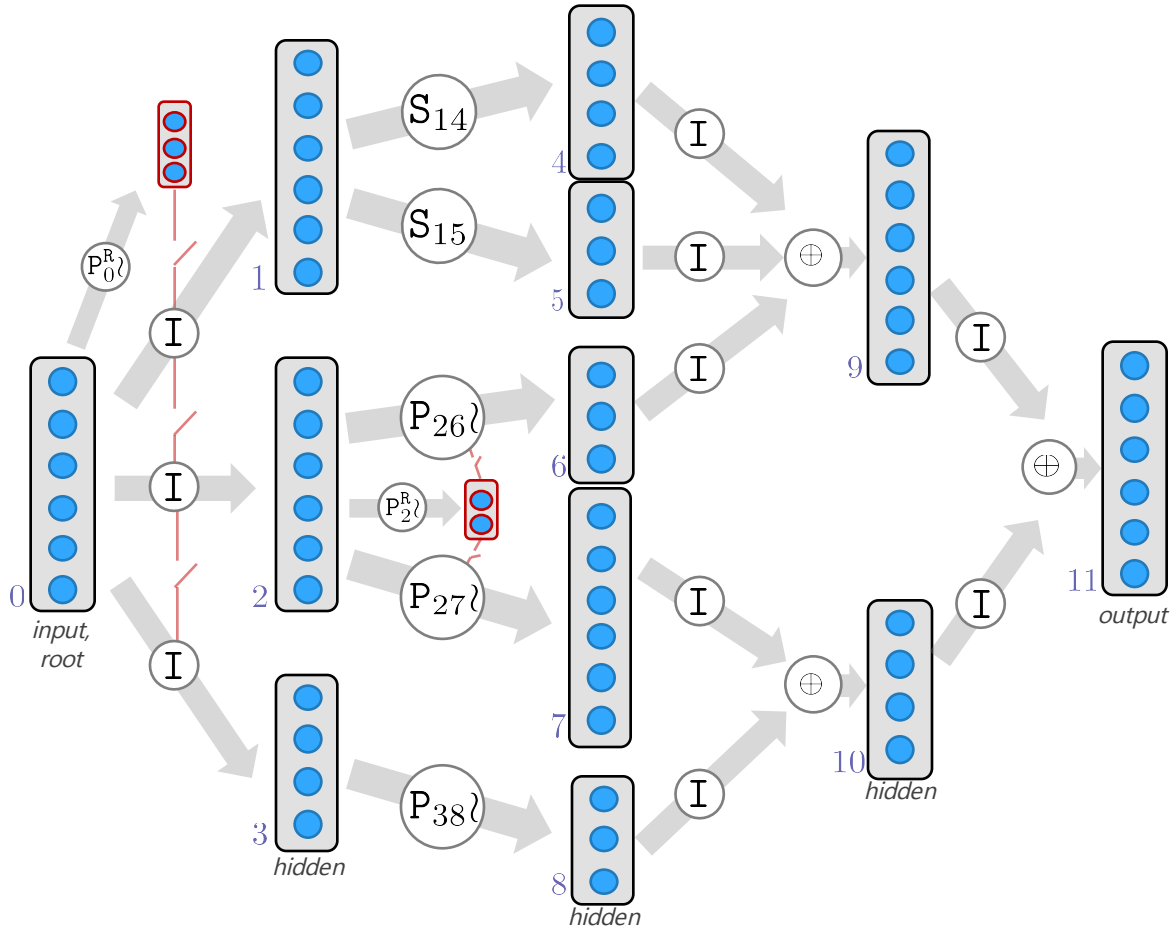




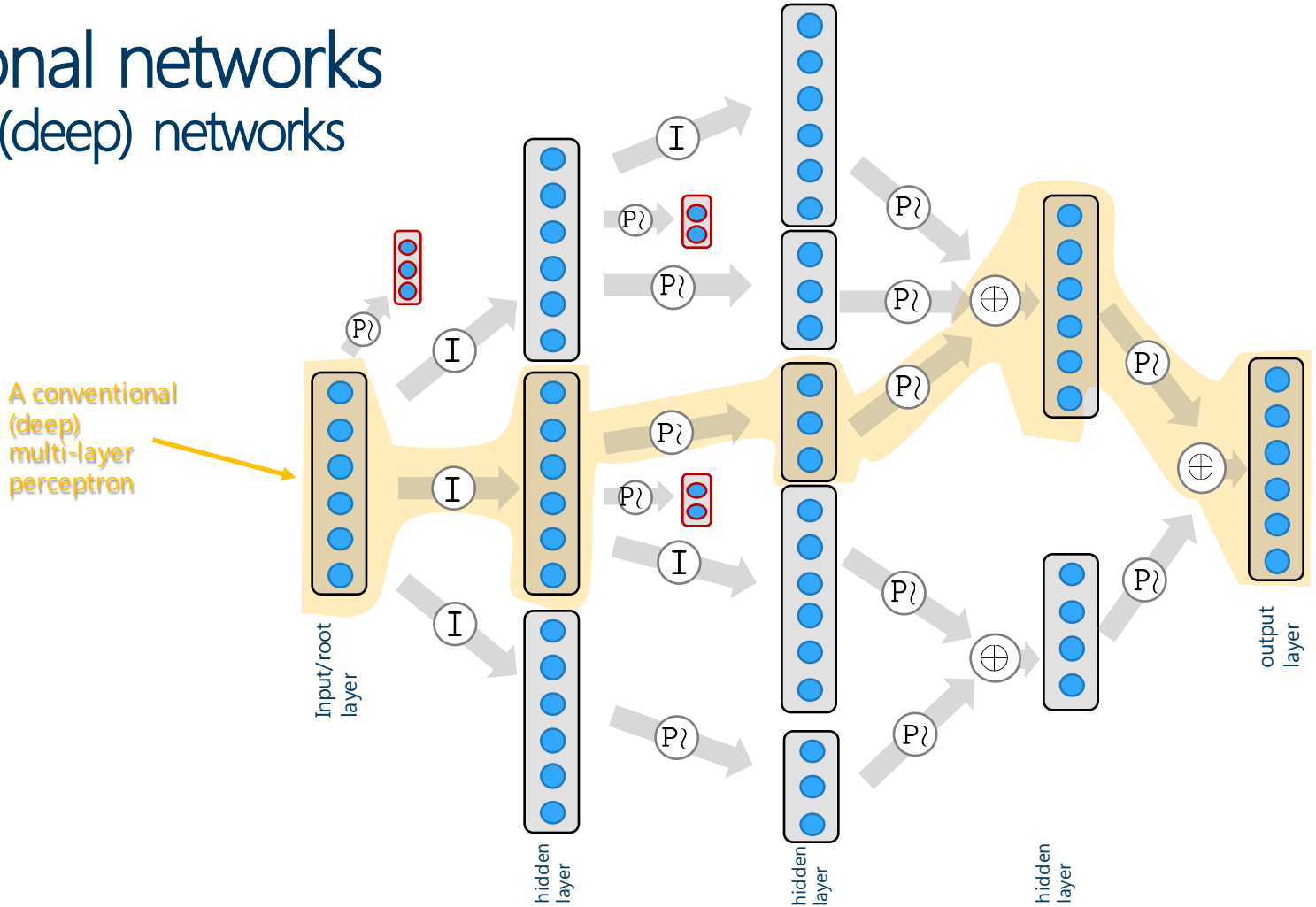
Explicit vs Implicit Data Routing



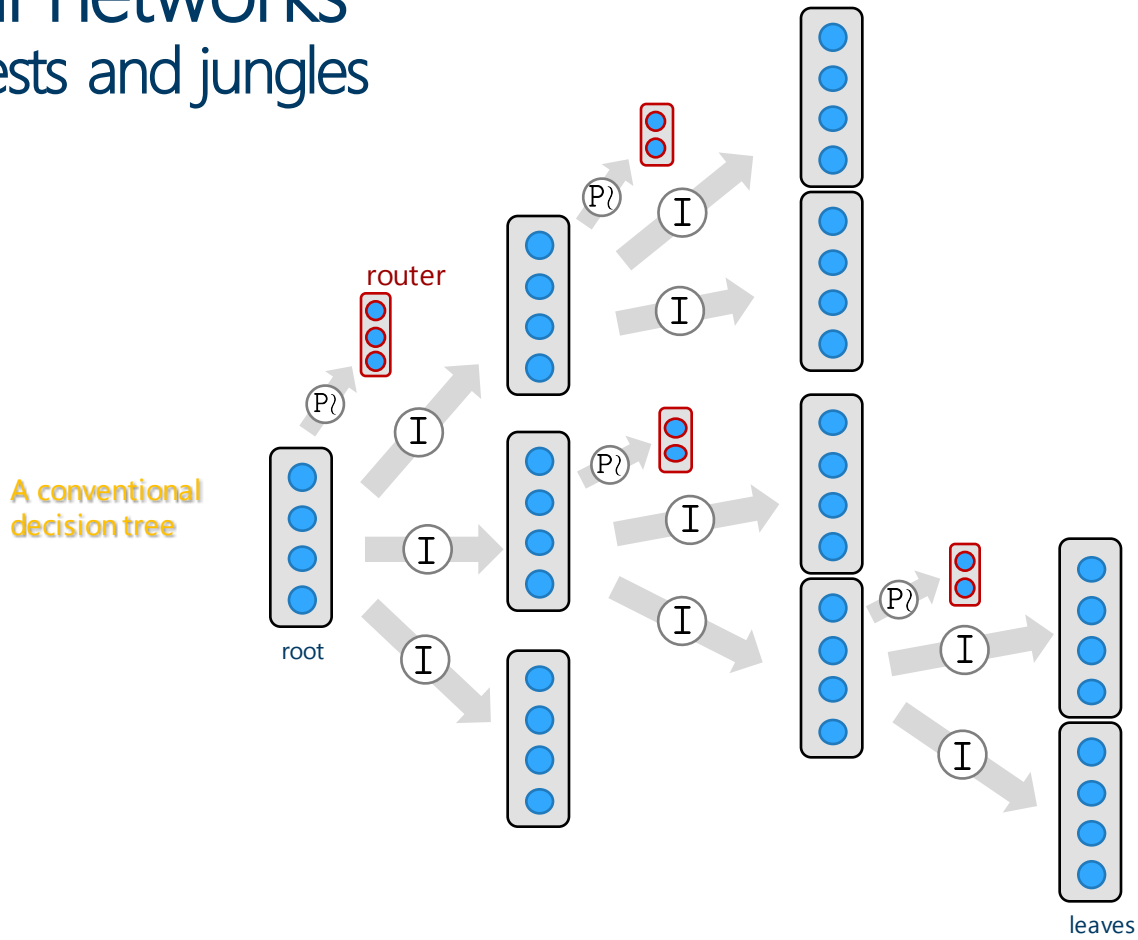
A conditional network



Conditional networks
generalize (deep) networks



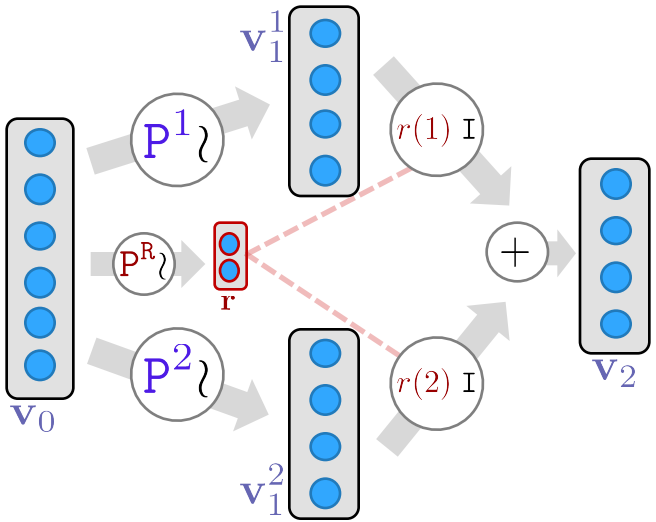
Conditional networks
generalize forests and jungles



Outline

- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

Training via gradient-descent back-propagation



Valid in general for **soft routing weights r.**

All implemented in unmodified CAFFE (GPU/CPU).

$$E(\boldsymbol{\theta}) = \frac{1}{2}\sum_{\mathcal{T}} \left(\mathbf{v}_2(\boldsymbol{\theta}) - \mathbf{v}_2^*\right)^\top \left(\mathbf{v}_2(\boldsymbol{\theta}) - \mathbf{v}_2^*\right)$$ The training loss to be minimized.

$$\boldsymbol{\theta} := \{\mathrm{P}^{\mathrm{R}}, \{\mathrm{P}^j\}\}$$ The parameters to optimize over

$$\mathcal{T} = \{\ldots, (\mathbf{v}_0, \mathbf{v}_2^*), \ldots\}$$ The labelled training set

$$\mathbf{v}_2(\boldsymbol{\theta}) = \mathbf{r}(\boldsymbol{\theta})\ \mathrm{V}_1(\boldsymbol{\theta})$$ Network's forward mapping

$$\mathrm{V}_1 = \begin{bmatrix} \vdots & \vdots & \vdots \\ - & (\mathbf{v}_1^j)^\top & - \\ \vdots & \vdots & \vdots \end{bmatrix}$$ Matrix of outputs for all routes

$$\mathbf{v}_1^j = \sigma\left(\mathrm{P}^j \mathbf{v}_0\right)$$ Intermediate output for j-th route

$$\mathbf{r} = \sigma\left(\mathrm{P}^{\mathrm{R}} \mathbf{v}_0\right)$$ **Soft** routing weights $\mathbf{r} \in \mathbb{R}^R$

$$\Delta\boldsymbol{\theta}_{t+1} := -\rho \left.\frac{\partial E}{\partial \boldsymbol{\theta}}\right|_t$$ The parameter update rule

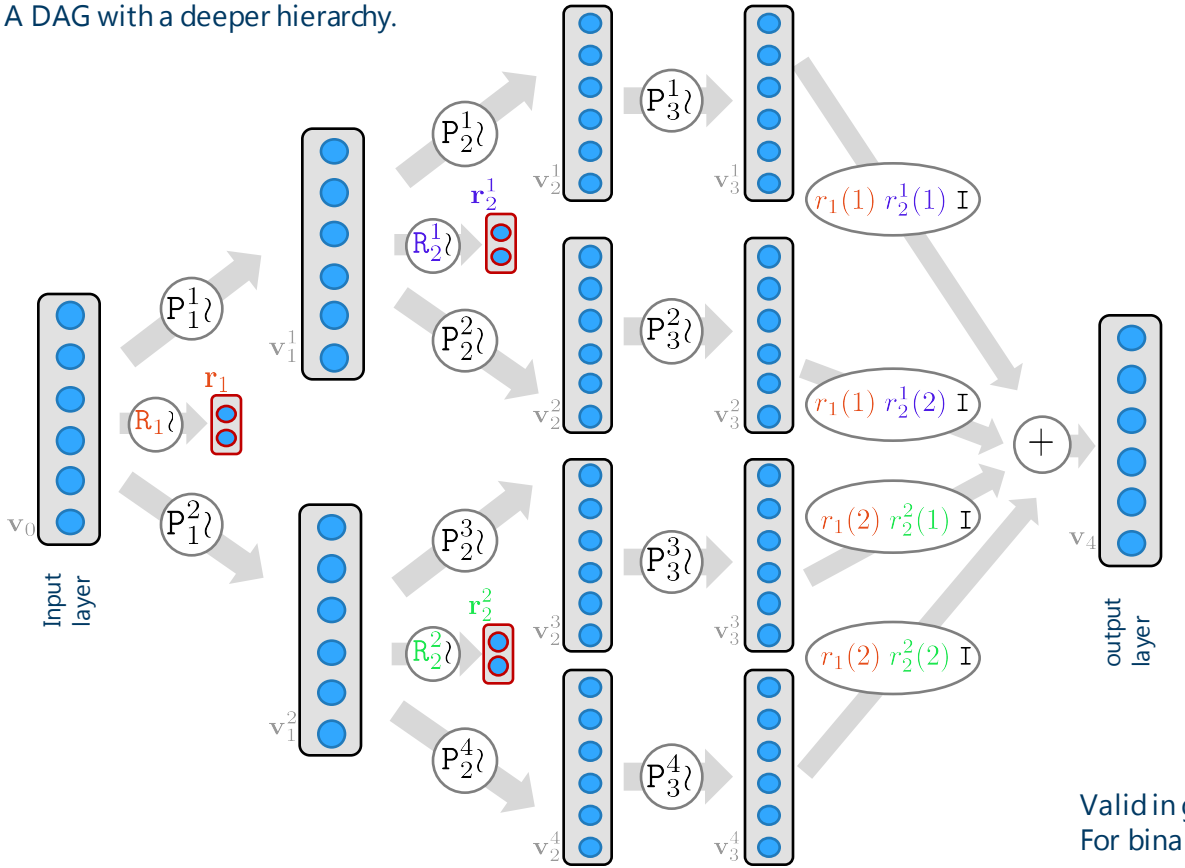
$$\frac{\partial E}{\partial \boldsymbol{\theta}} = \frac{\partial E}{\partial \mathbf{v}_2}\frac{\partial \mathbf{v}_2}{\partial \boldsymbol{\theta}} = \frac{\partial E}{\partial \mathbf{v}_2}\left(\frac{\partial \mathbf{r}}{\partial \mathrm{P}^{\mathrm{R}}}\mathrm{V}_1 + \sum_j r(j)\frac{\partial \mathbf{v}_1^j}{\partial \boldsymbol{\phi}^j}\frac{\partial \boldsymbol{\phi}^j}{\partial \mathrm{P}^j}\right)$$ Chain rule

$$\boldsymbol{\phi}^j := \mathrm{P}^j \mathbf{v}_0$$

Routing weights influence the back-propagating errors differently for different routes

Training a conditional network via back-propagation

A DAG with a deeper hierarchy.

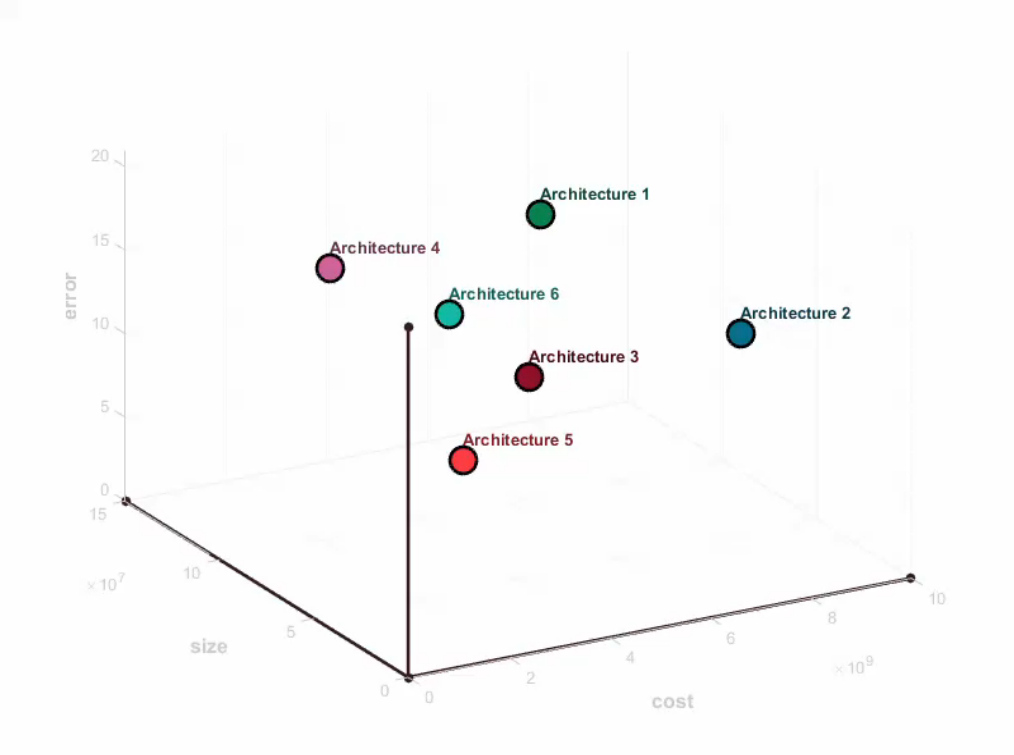


Valid in general for **soft routing weights r**.
For binary, hard weights the network structure simplifies.

Outline

- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional deep ensembles
- Conclusion

How to measure the efficacy of a DNN architecture ?



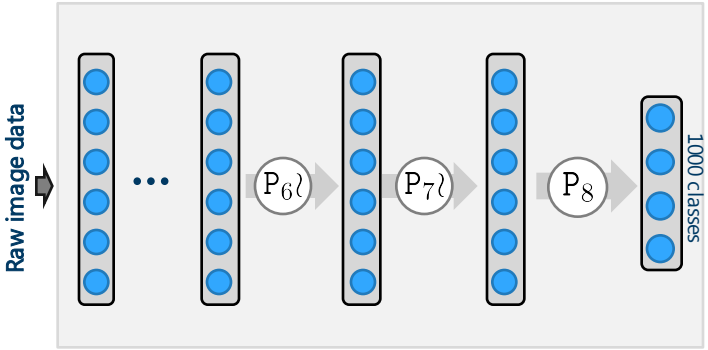
- Test accuracy
- Test-time compute cost
- Model size (num. params)
- …
- Training-time compute cost
- Ease of parallelization
- Generalization to other datasets
- …

Outline

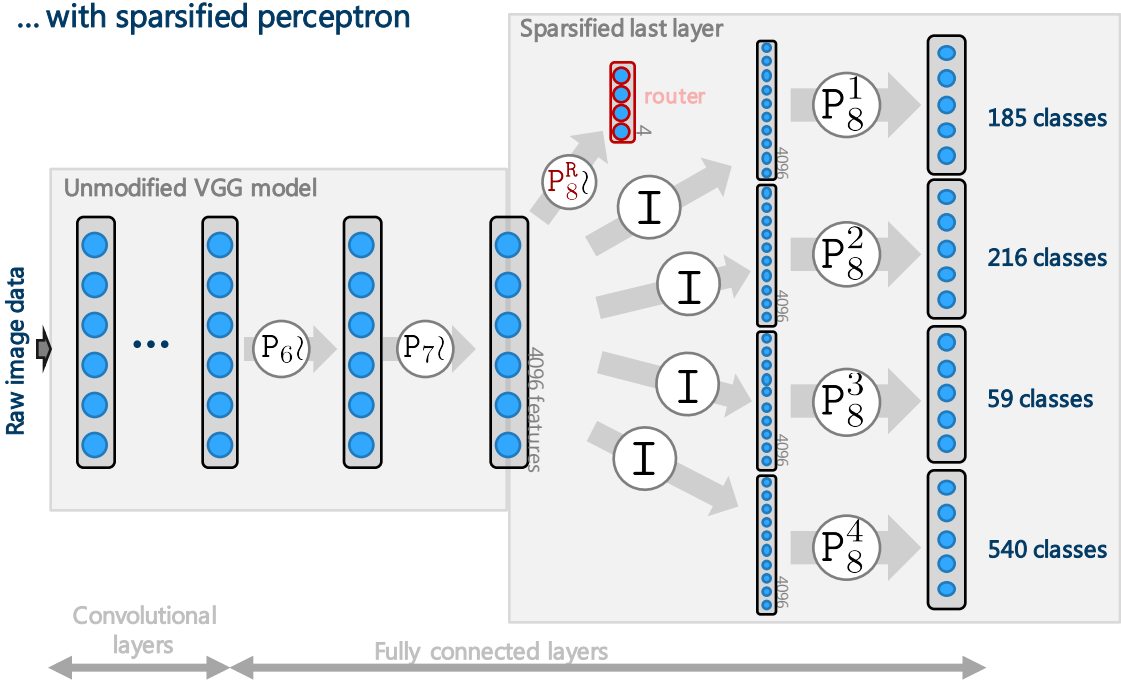
- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

Conditional sparsification of a perceptron

Trained convolutional DNN



... with sparsified perceptron

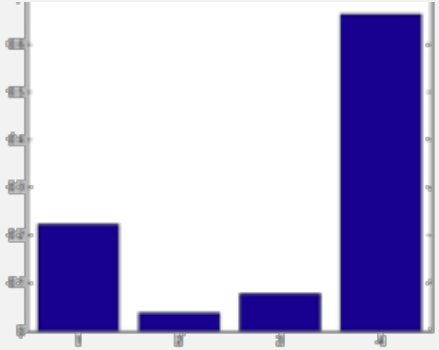


[K. Simonyan, A. Zisserman. **Very Deep Convolutional Networks for Large-Scale Image Recognition** arXiv, 2014]

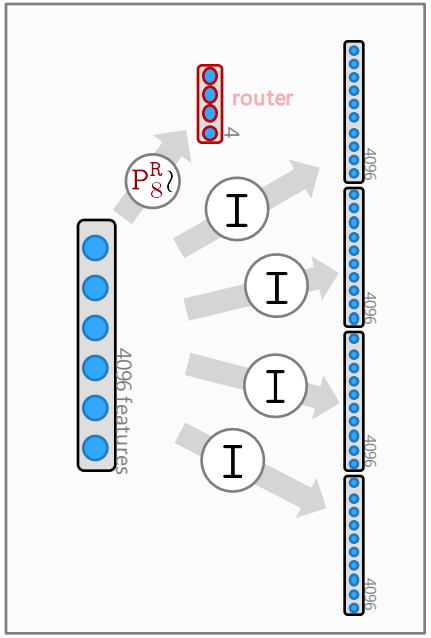
Conditional sparsification of a perceptron

Testing with multiple (hard) routes per input image

$p_1(r)$

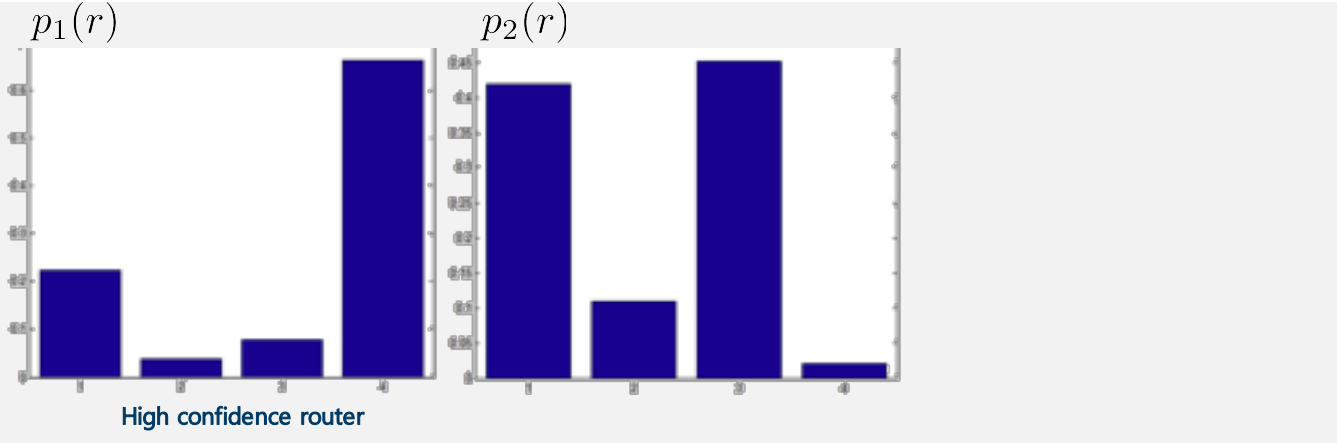


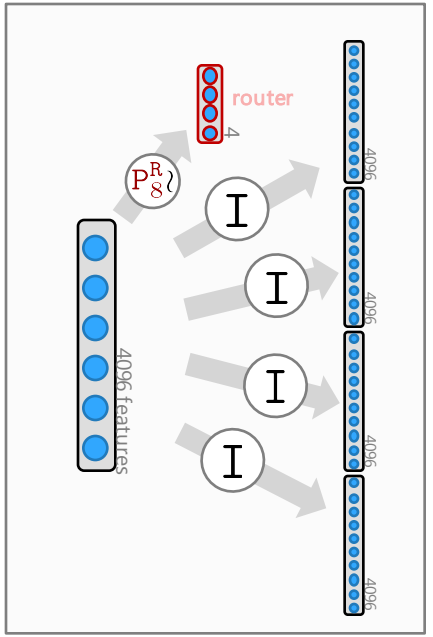
High confidence router



Conditional sparsification of a perceptron

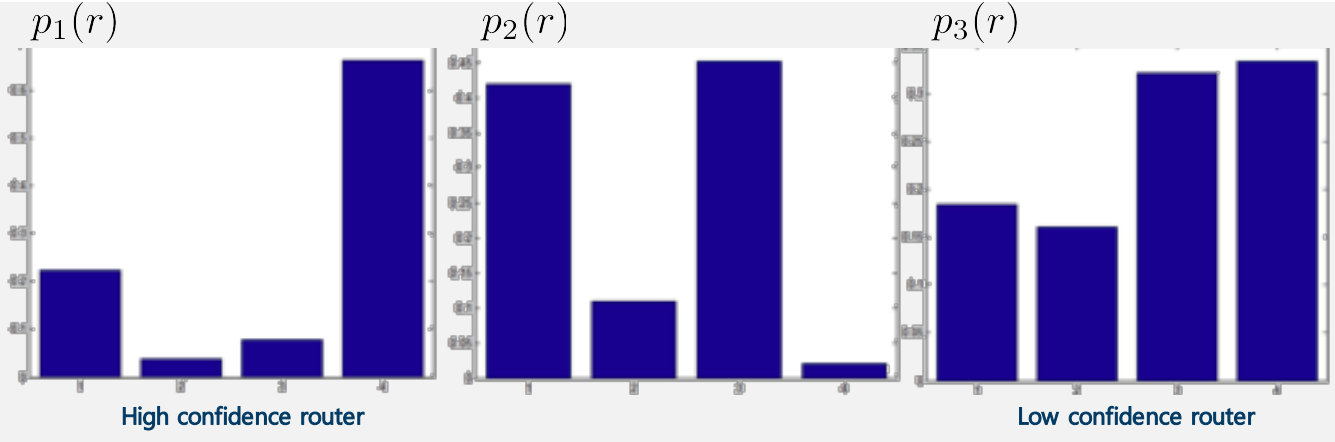
Testing with multiple (hard) routes per input image

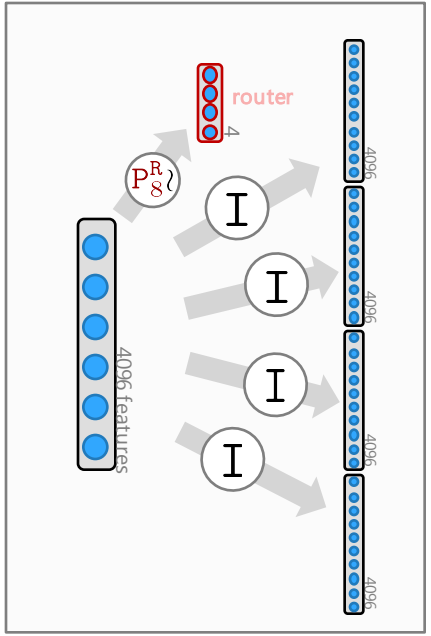




Conditional sparsification of a perceptron

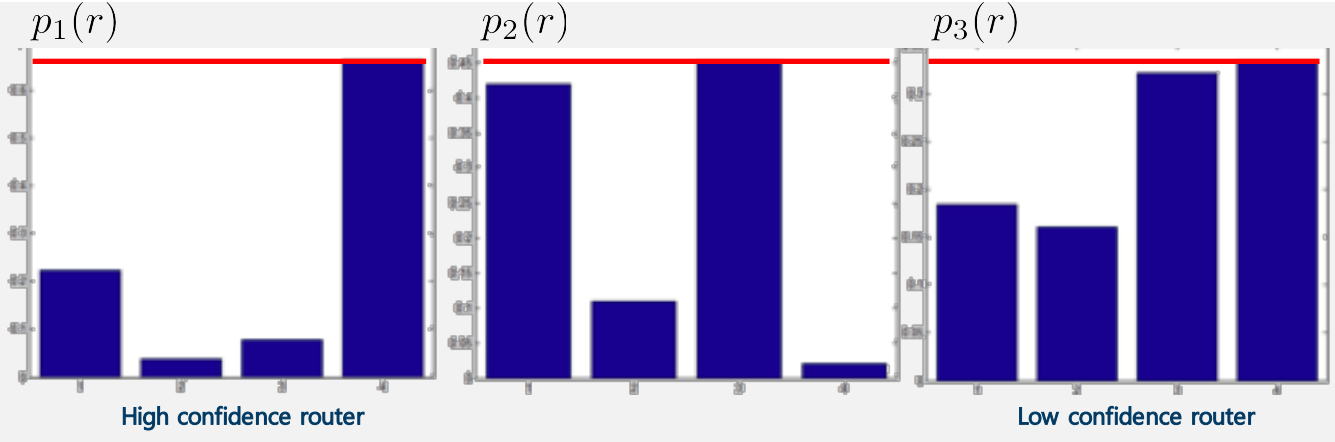
Testing with multiple (hard) routes per input image

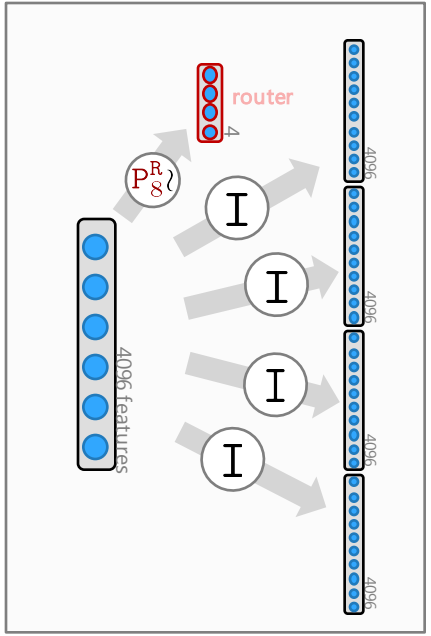




Conditional sparsification of a perceptron

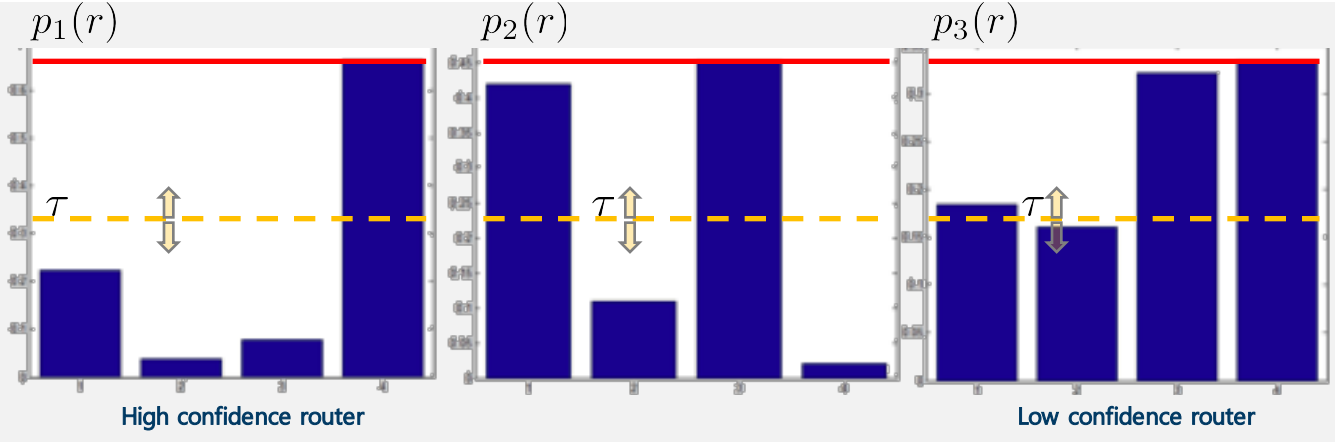
Testing with multiple (hard) routes per input image

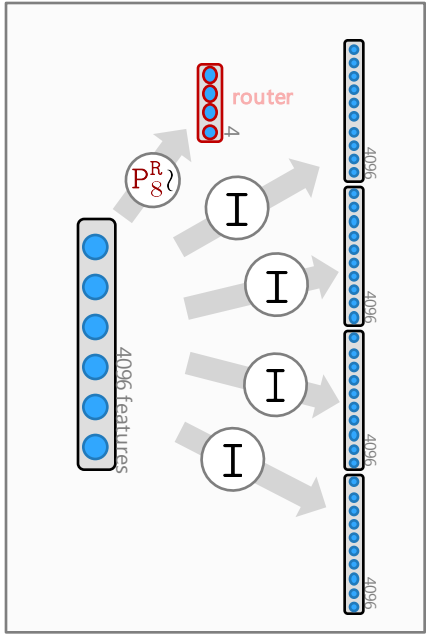




Conditional sparsification of a perceptron

Testing with multiple (hard) routes per input image

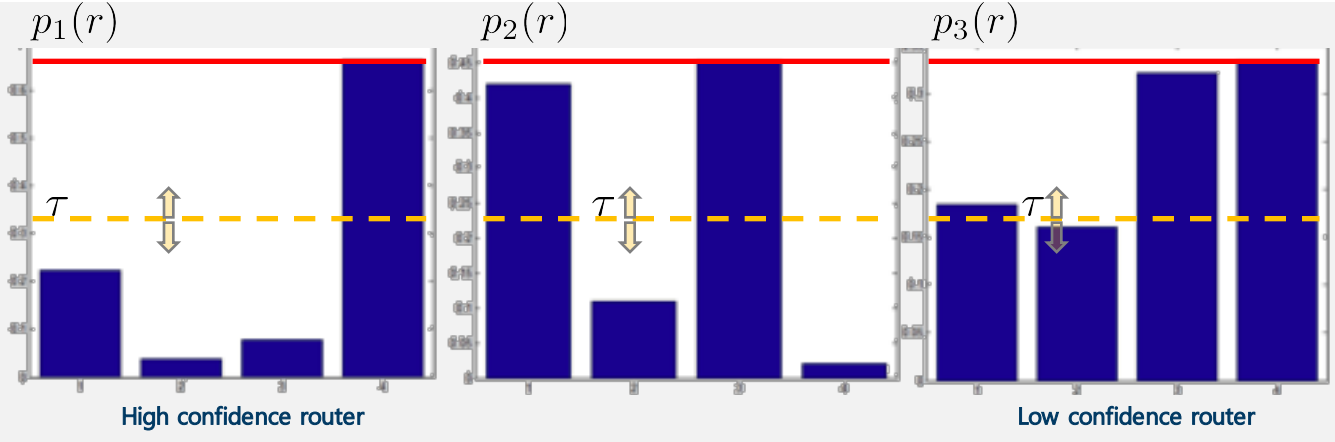


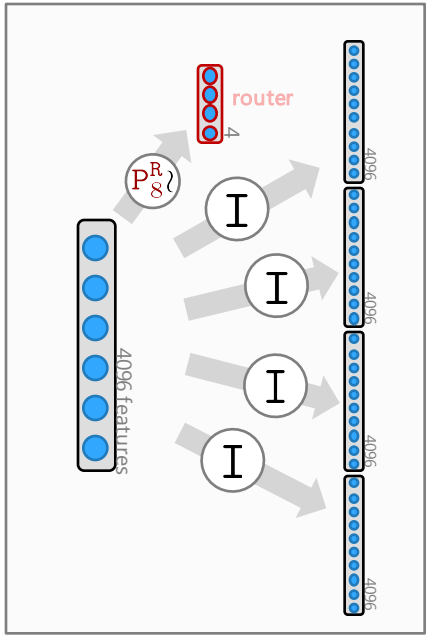


$$\tau = \alpha \max_r p(r) \qquad \alpha \in [0, 1]$$

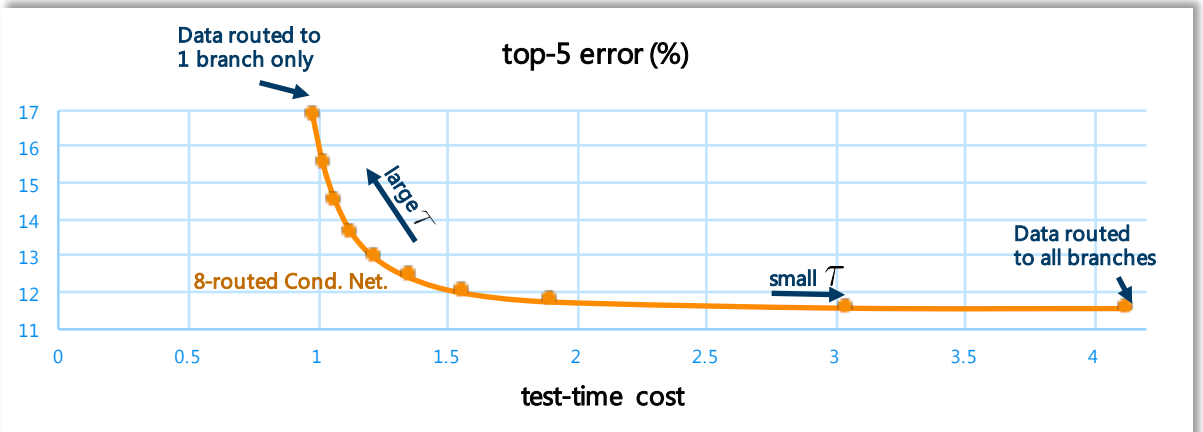
Conditional sparsification of a perceptron

Testing with multiple (hard) routes per input image





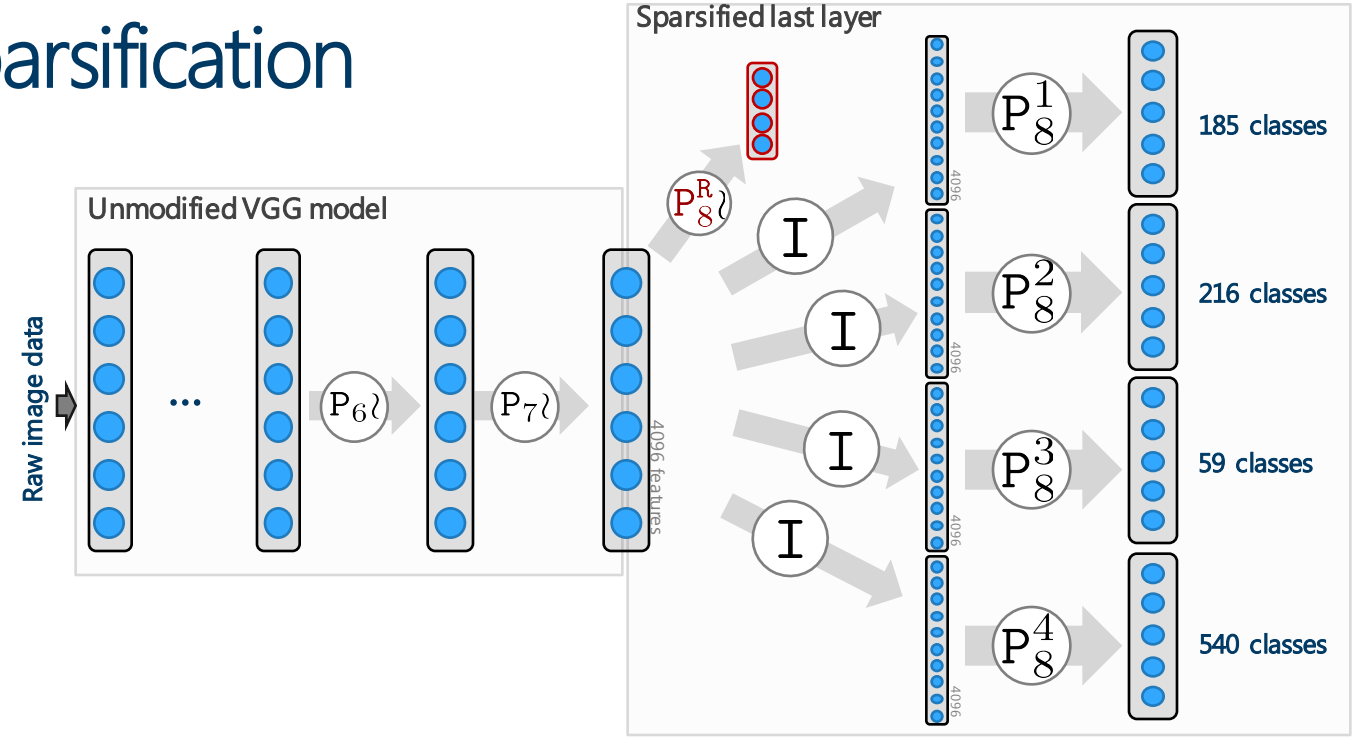
$$\tau = \alpha \max_r p(r) \qquad \alpha \in [0, 1]$$



On-demand accuracy-compute trade-off

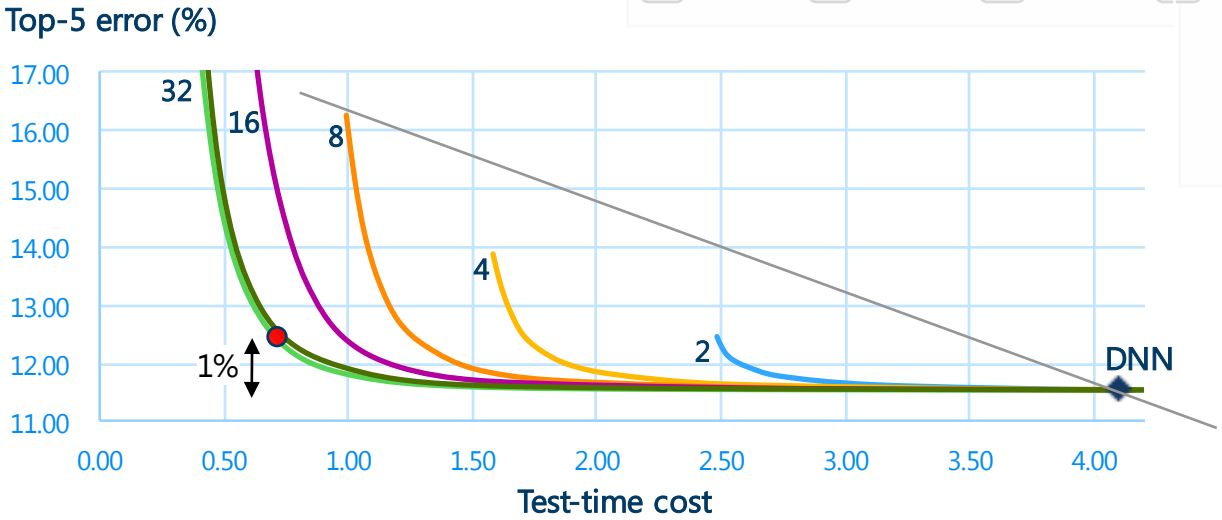
Conditional sparsification

On-demand accuracy-compute trade-off



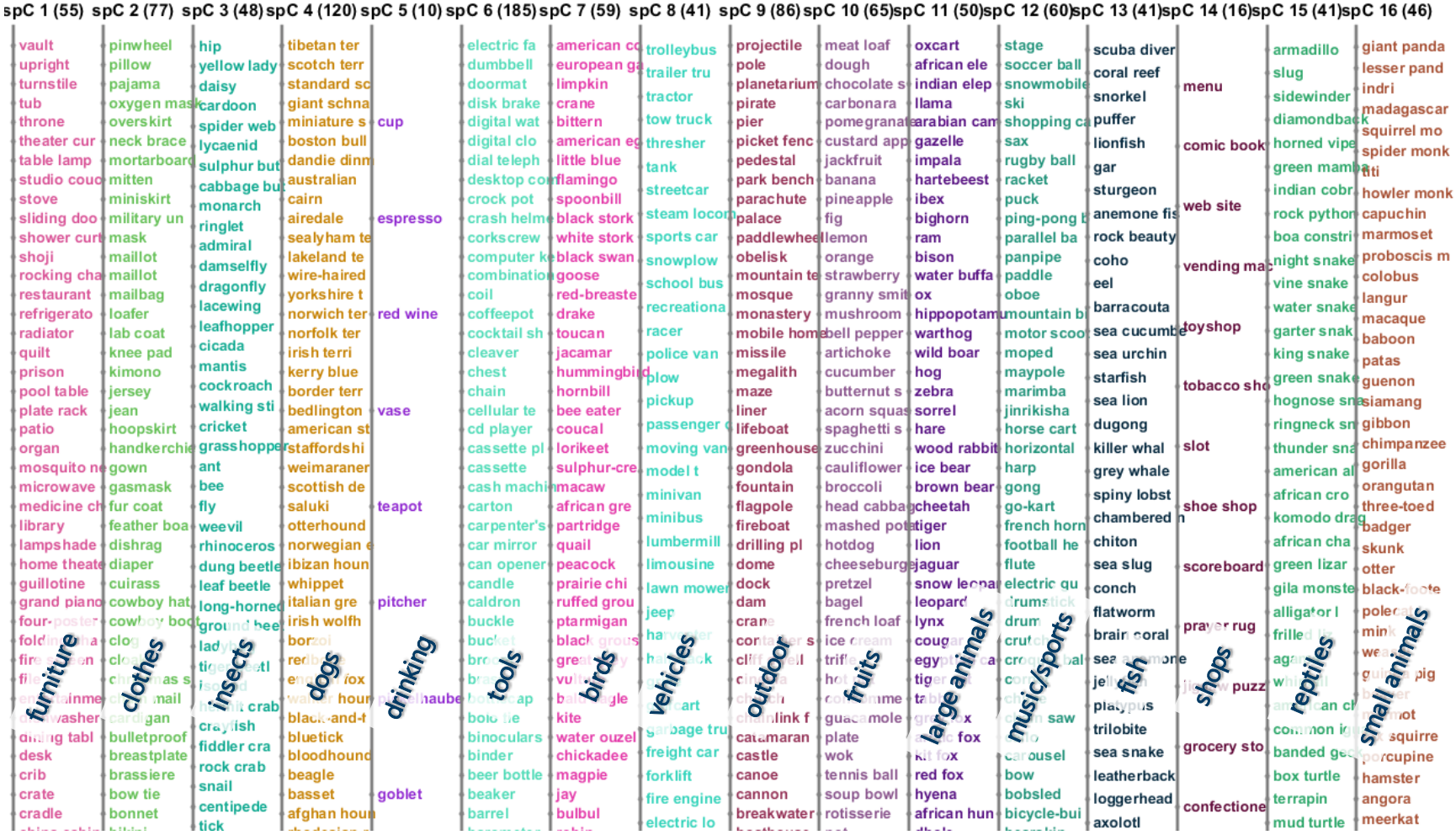
Conditional sparsification

On-demand accuracy-compute trade-off



- For 1% increase of T5E:
  - ~ X 6 speed up
  - ~ same model size

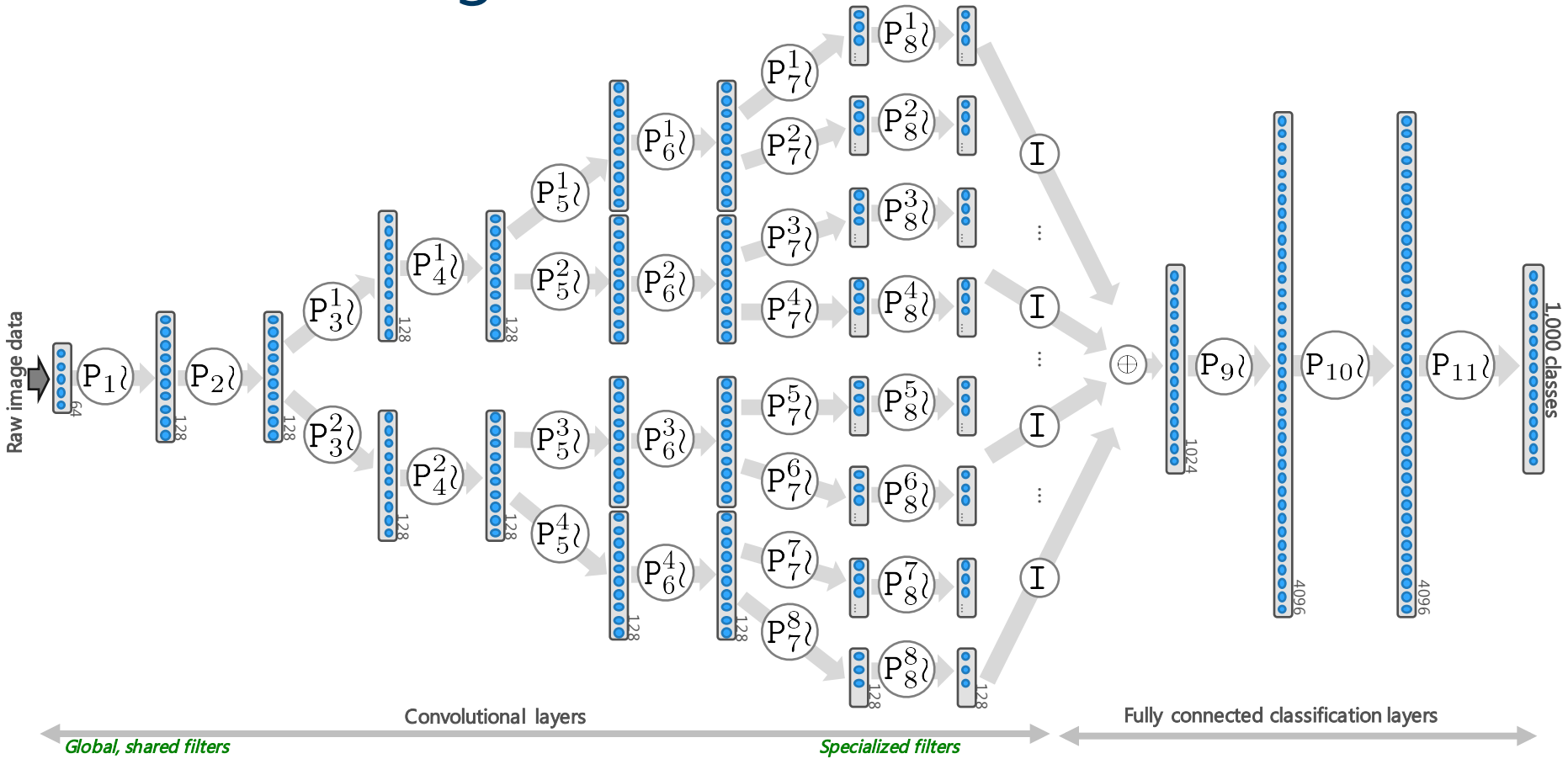
Conditional sparsification: learned super classes for a 16-ary CONNET



Outline

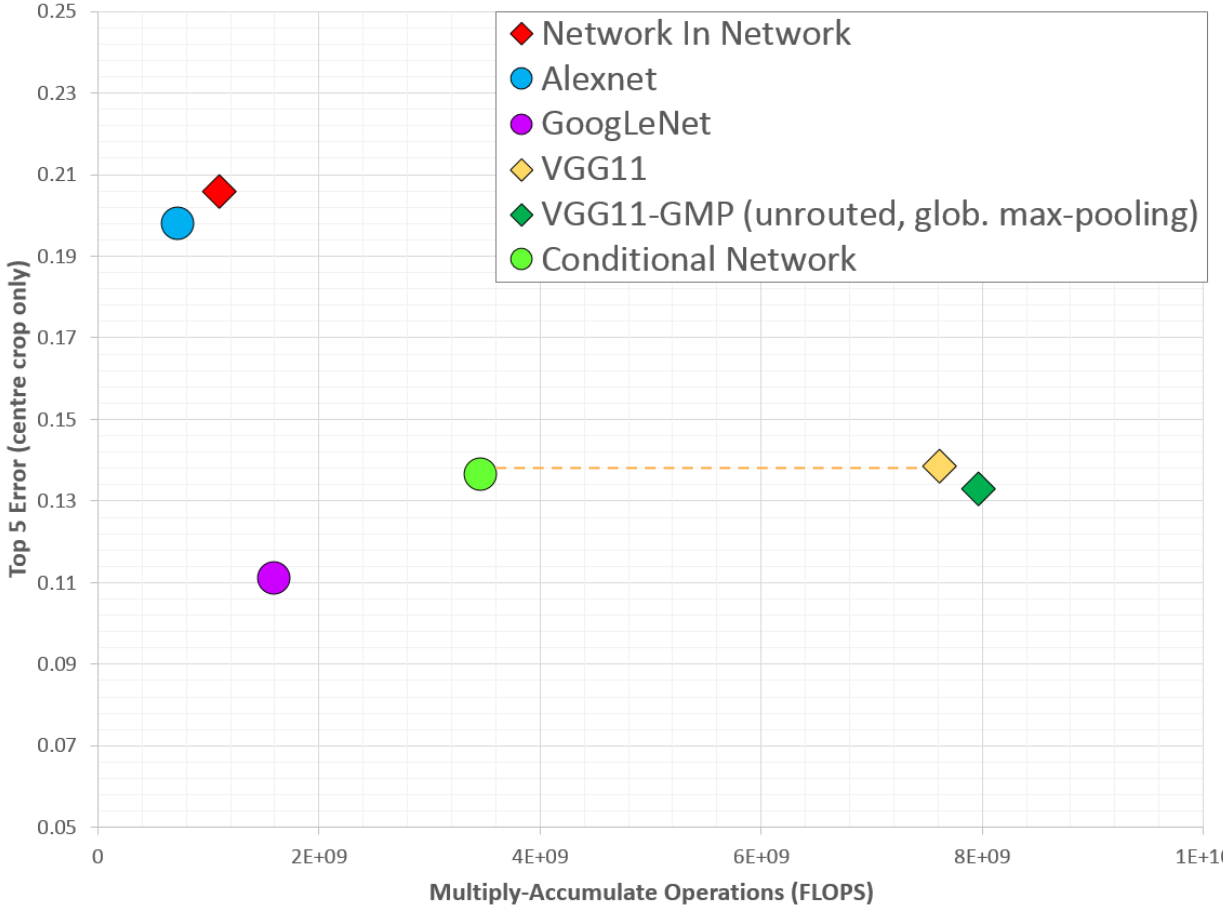
- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

Results on ImageNet-1000



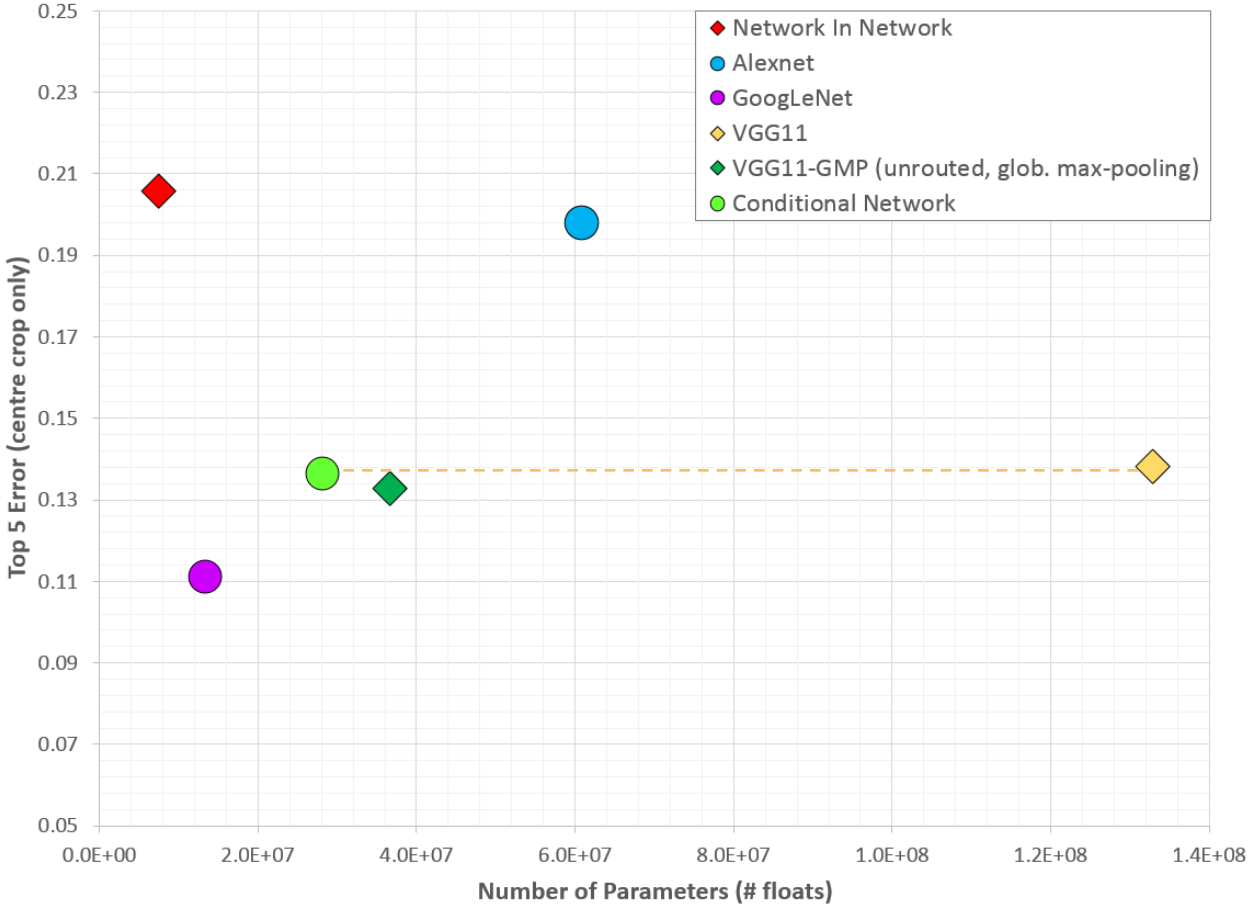
Results on ImageNet-1000

accuracy vs compute

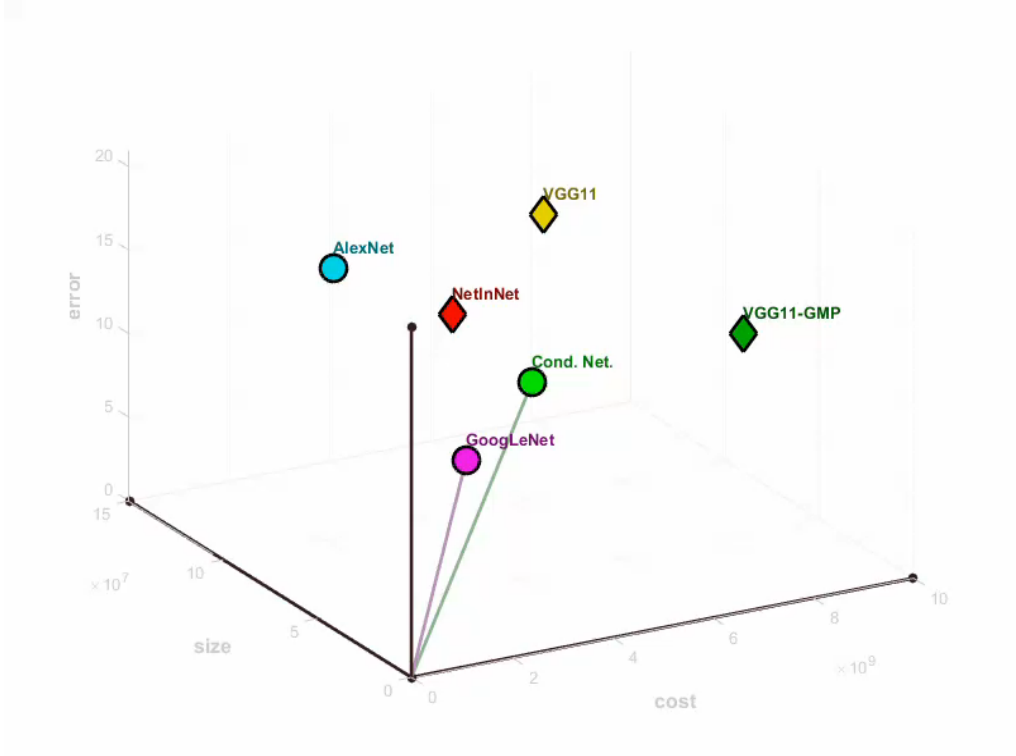


Results on ImageNet-1000

accuracy vs model size



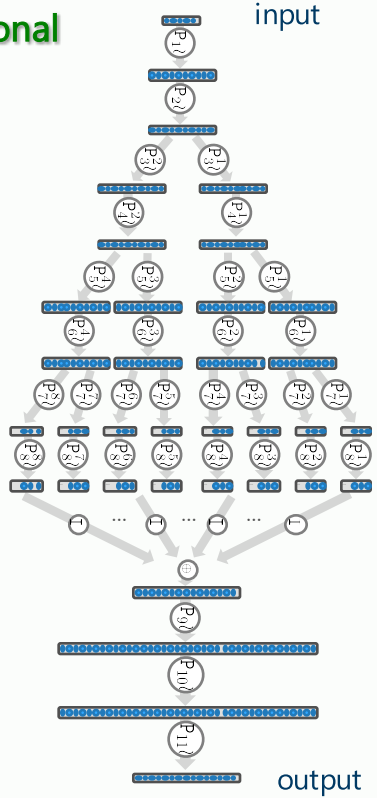
Results on ImageNet-1000



- Top-5 error
- Test-time compute cost (num. mult - add ops)
- Model size (num. params)

Results on ImageNet-1000

Our conditional network



11 layers

Implicitly-routed conditional networks

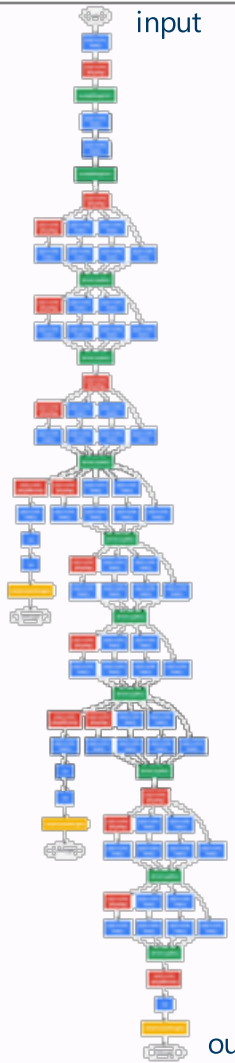
Simpler architecture

Intermediate, manually-weighted losses

Easier to train

Low-dimension embeddings

GoogLeNet



22 layers (~100 blocks)

Results on ImageNet-1000

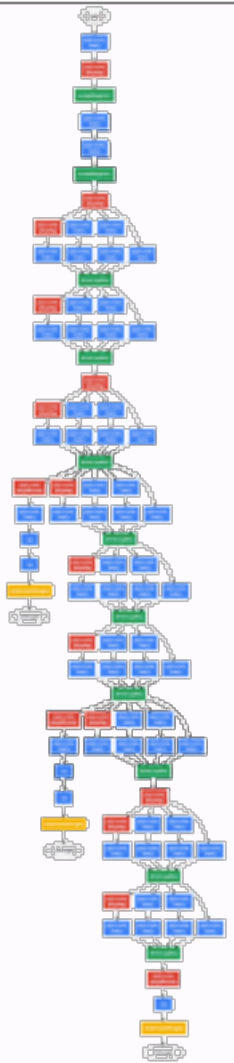
6 Training Methodology

Our networks were trained using the DistBelief [4] distributed machine learning system using modest amount of model and data-parallelism. Although we used CPU based implementation only, a rough estimate suggests that the GoogLeNet network could be trained to convergence using few high-end GPUs within a week, the main limitation being the memory usage. Our training used asynchronous stochastic gradient descent with 0.9 momentum [17], fixed learning rate schedule (decreasing the learning rate by 4% every 8 epochs). Polyak averaging [13] was used to create the final model used at inference time.

Our image sampling methods have changed substantially over the months leading to the competition, and already converged models were trained on with other options, sometimes in conjunction with changed hyperparameters, like dropout and learning rate, so it is hard to give a definitive guidance to the most effective single way to train these networks. To complicate matters further, some of the models were mainly trained on smaller relative crops, others on larger ones, inspired by [8]. Still, one prescription that was verified to work very well after the competition includes sampling of various sized patches of the image whose size is distributed evenly between 8% and 100% of the image area and whose aspect ratio is chosen randomly between 3/4 and 4/3. Also, we found that the photometric distortions by Andrew Howard [8] were useful to combat overfitting to some extent. In addition, we started to use random interpolation methods (bilinear, area, nearest neighbor and cubic, with equal probability) for resizing relatively late and in conjunction with other hyperparameter changes, so we could not tell definitely whether the final results were affected positively by their use.

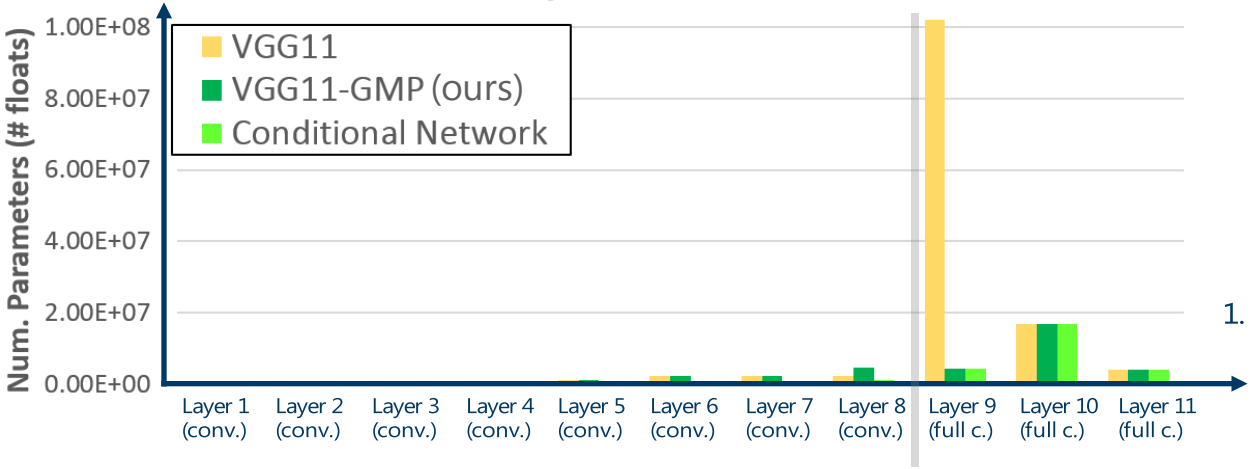
[C.Szegedy et al. **Going Deeper with Convolutions.** arXiv. Sep 2014]

GoogLeNet



22 layers (~100 blocks)

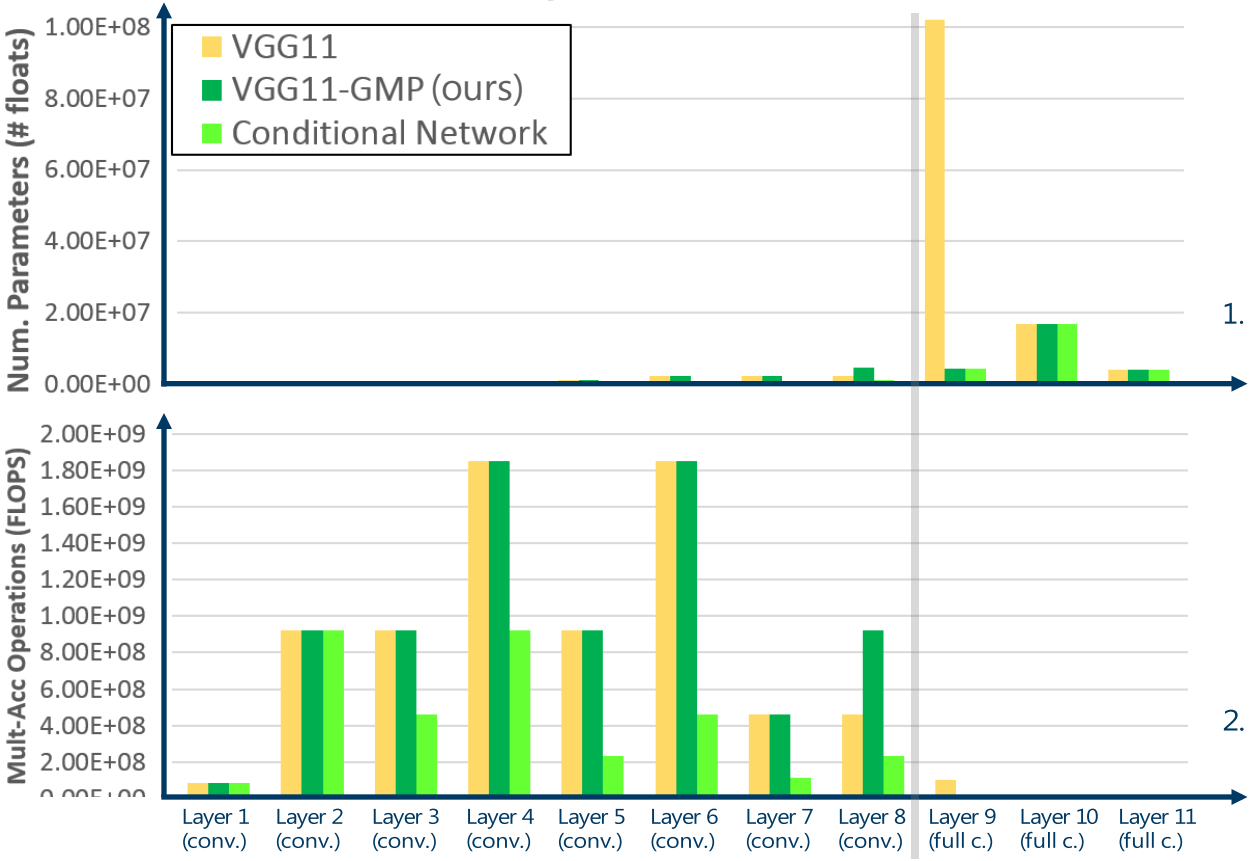
Results on ImageNet-1000



Layer by layer improvements in model size and compute efficiency

1. Global-max pooling reduces the model size

Results on ImageNet-1000



Layer by layer improvements in model size and compute efficiency

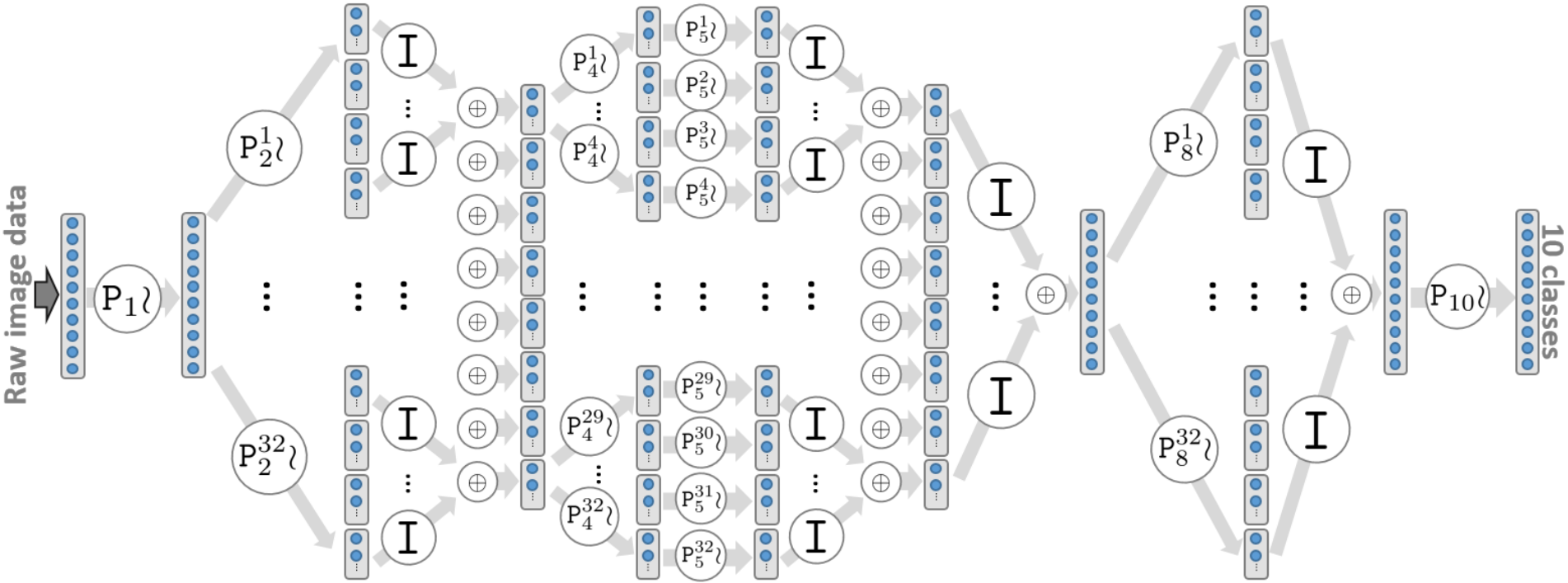
1. Global-max pooling reduces the model size

2. Data routing reduces the compute cost

Outline

- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

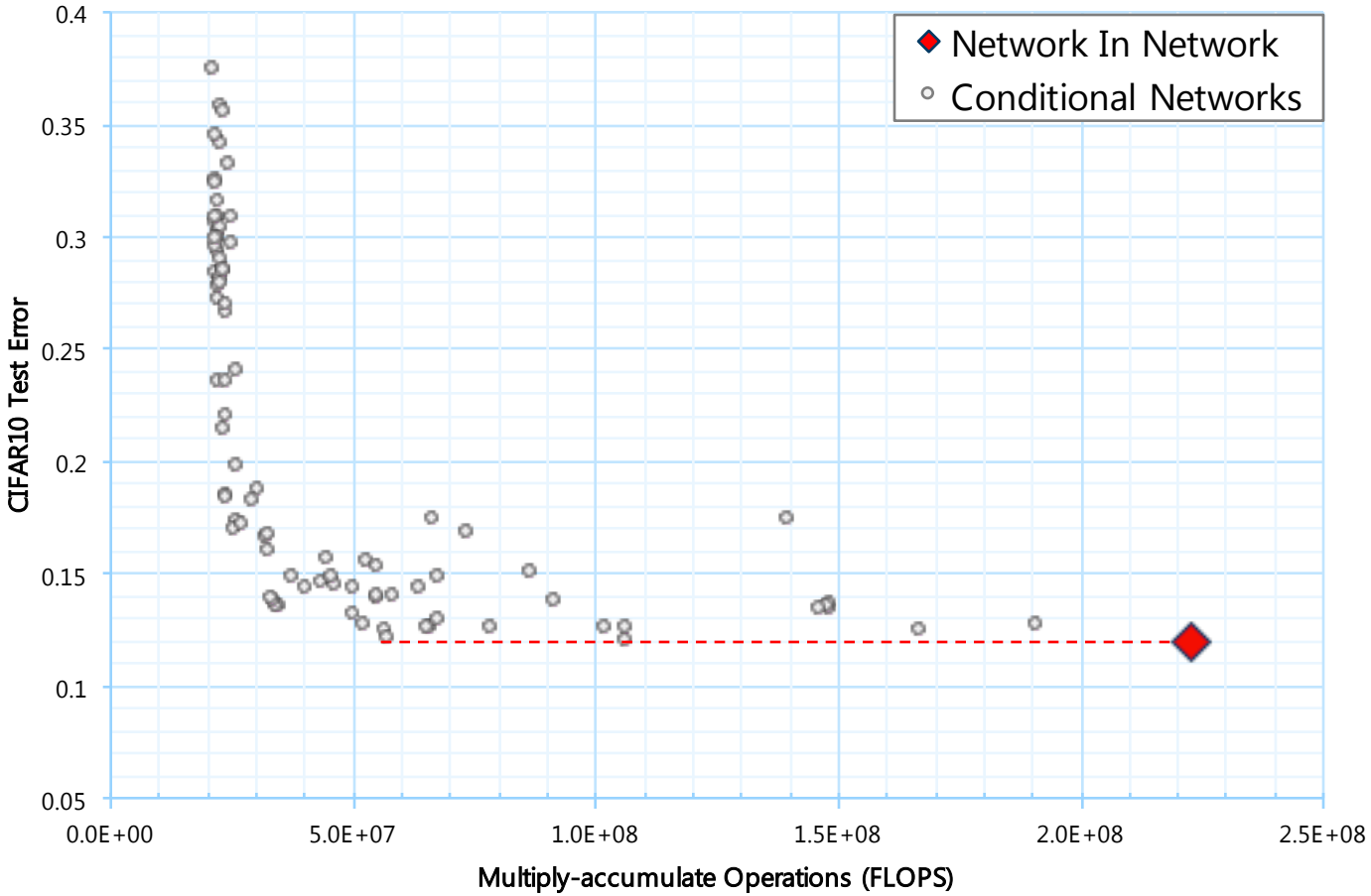
Results on CIFAR-10



Network **parameters and architecture have been learned** via ***Bayesian optimization*** (https://www.whetlab.com/)

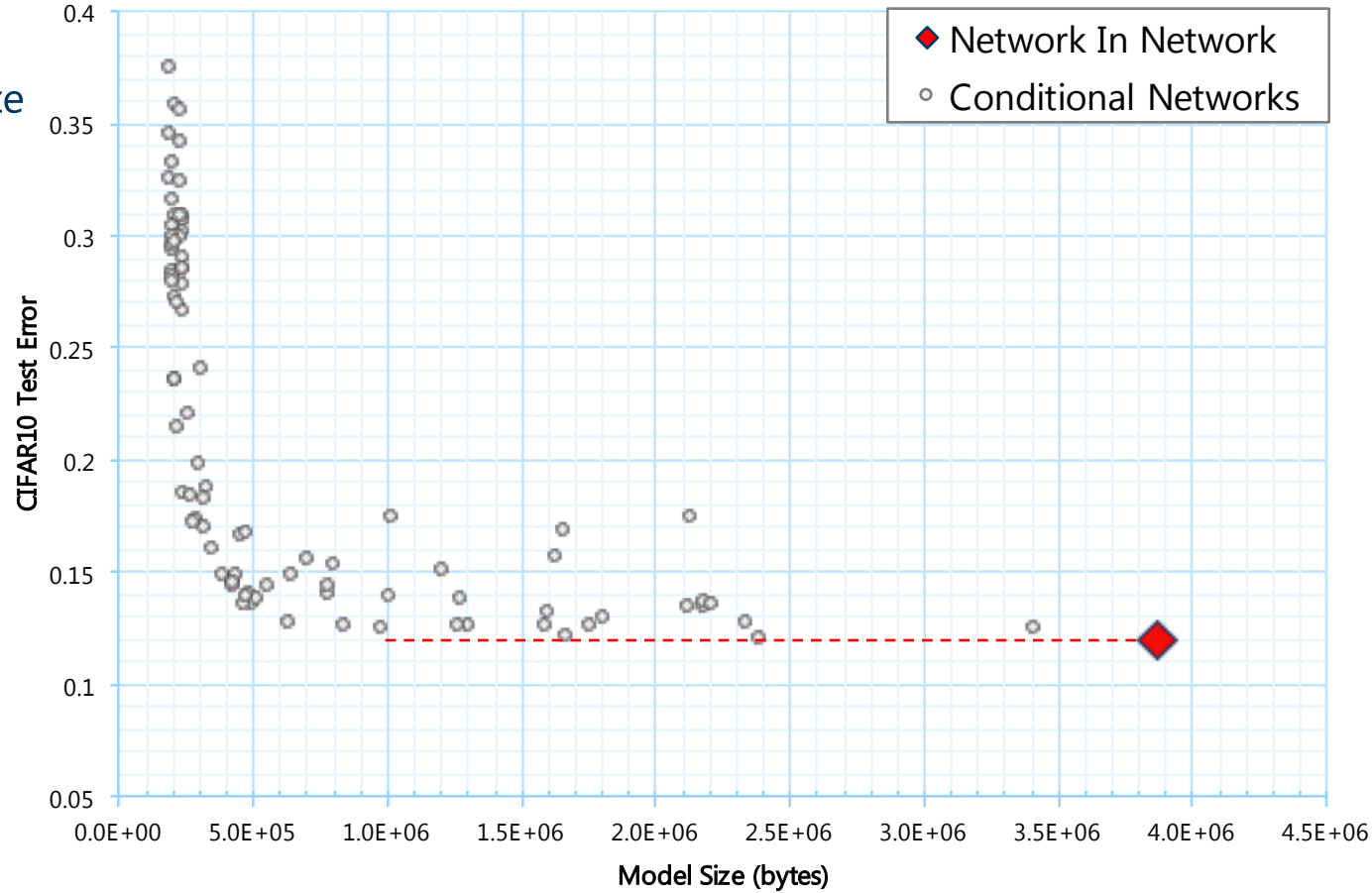
Results on CIFAR-10

accuracy vs compute

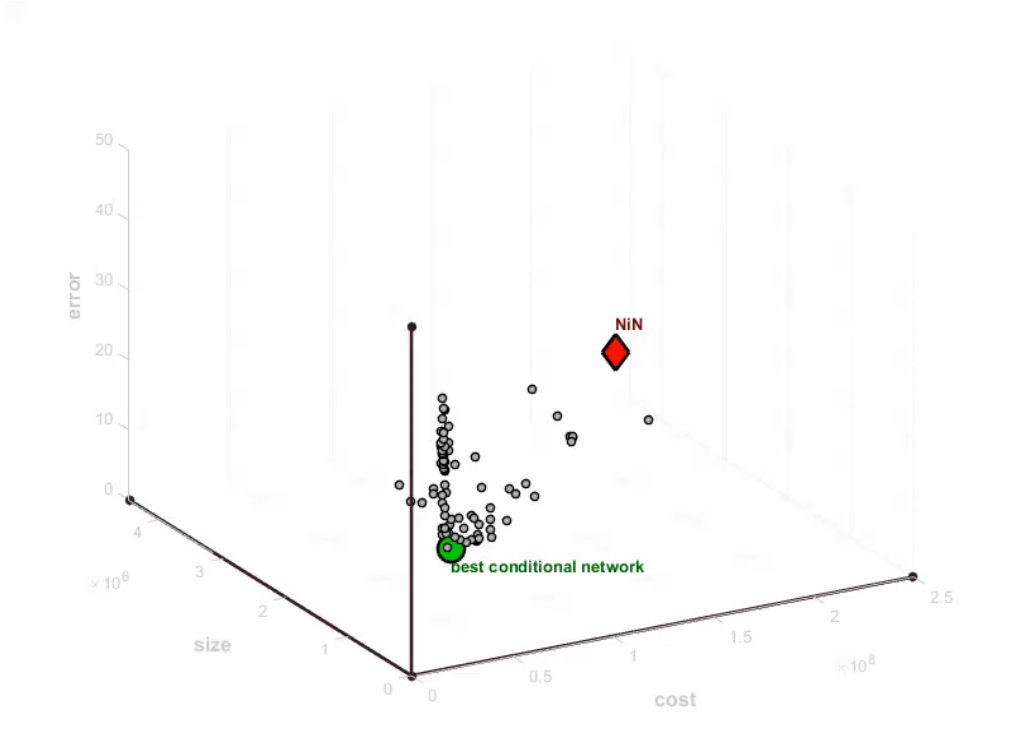


Results on CIFAR-10

accuracy vs model size



Results on CIFAR-10



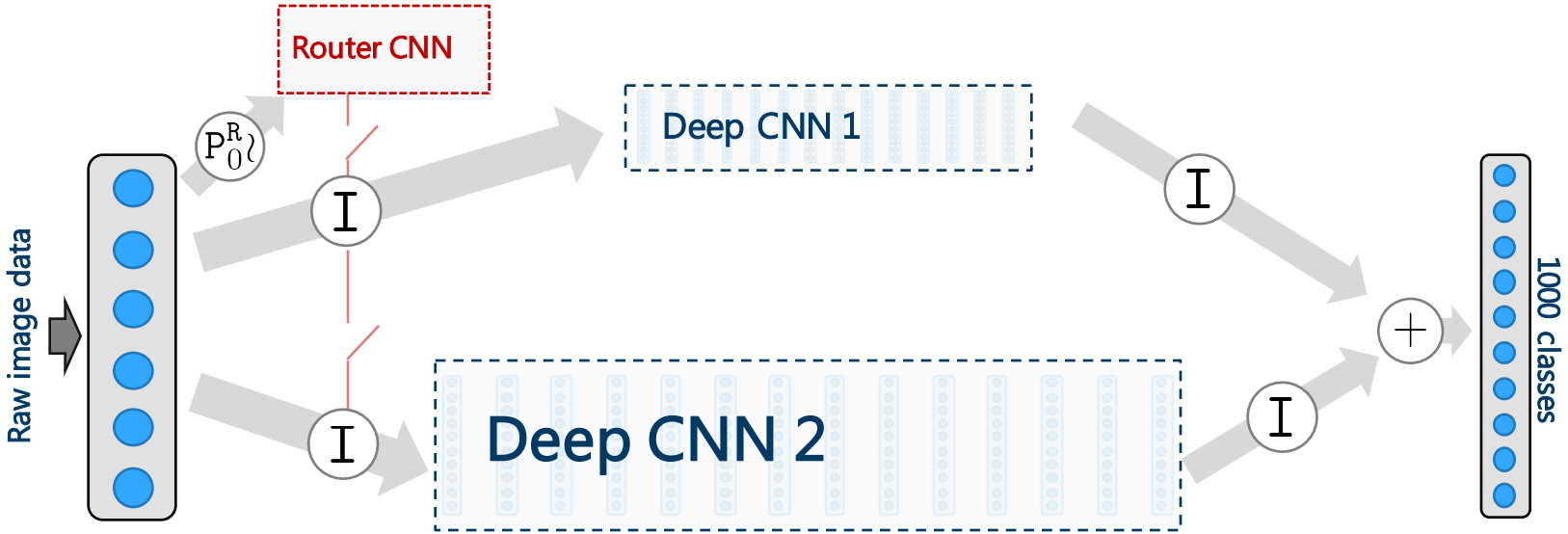
- Top-1 error
- Test-time compute cost (num. mult - add ops)
- Model size (bytes)

1/8 size, 1/7 cost, identical accuracy

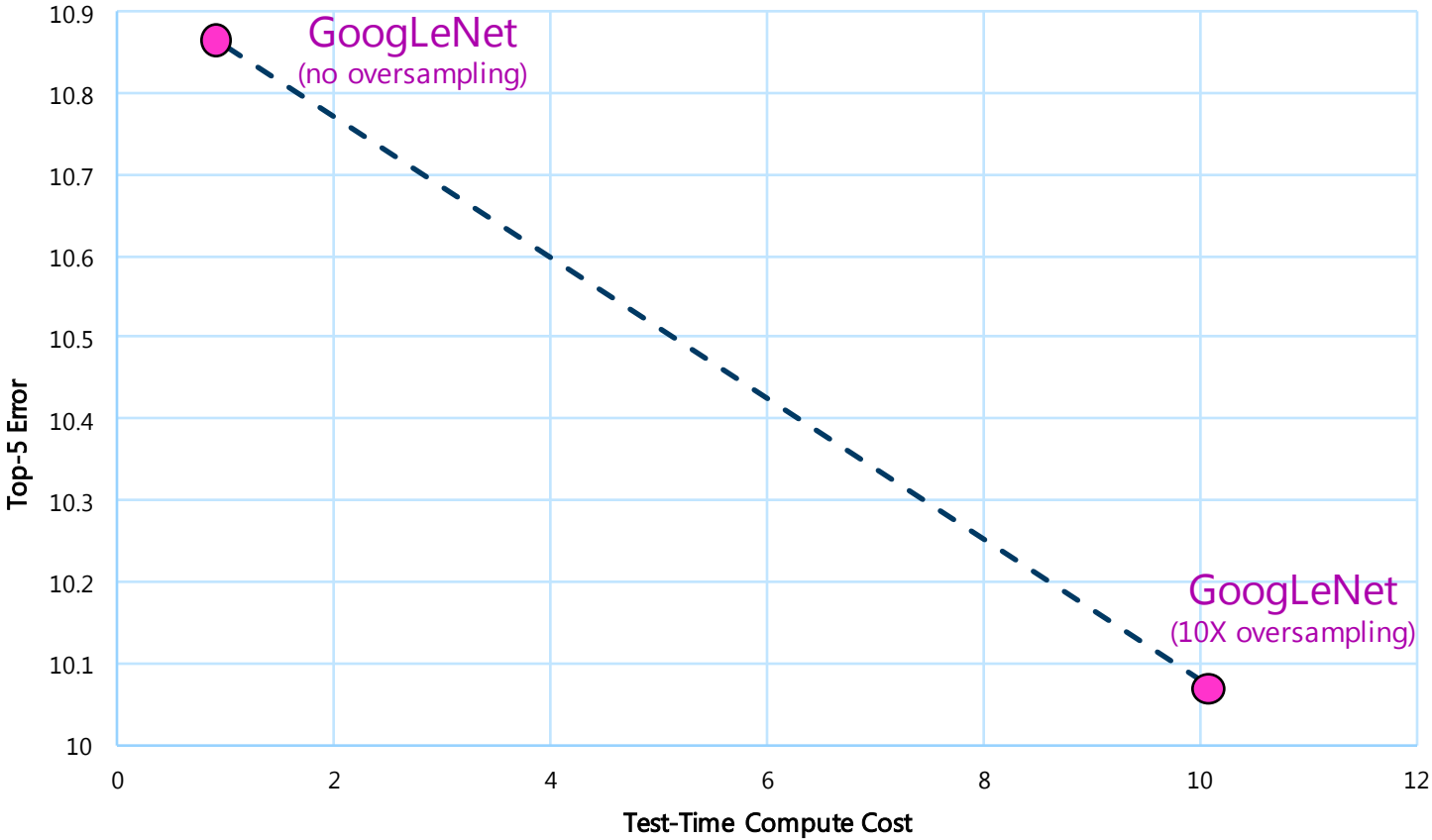
Outline

- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

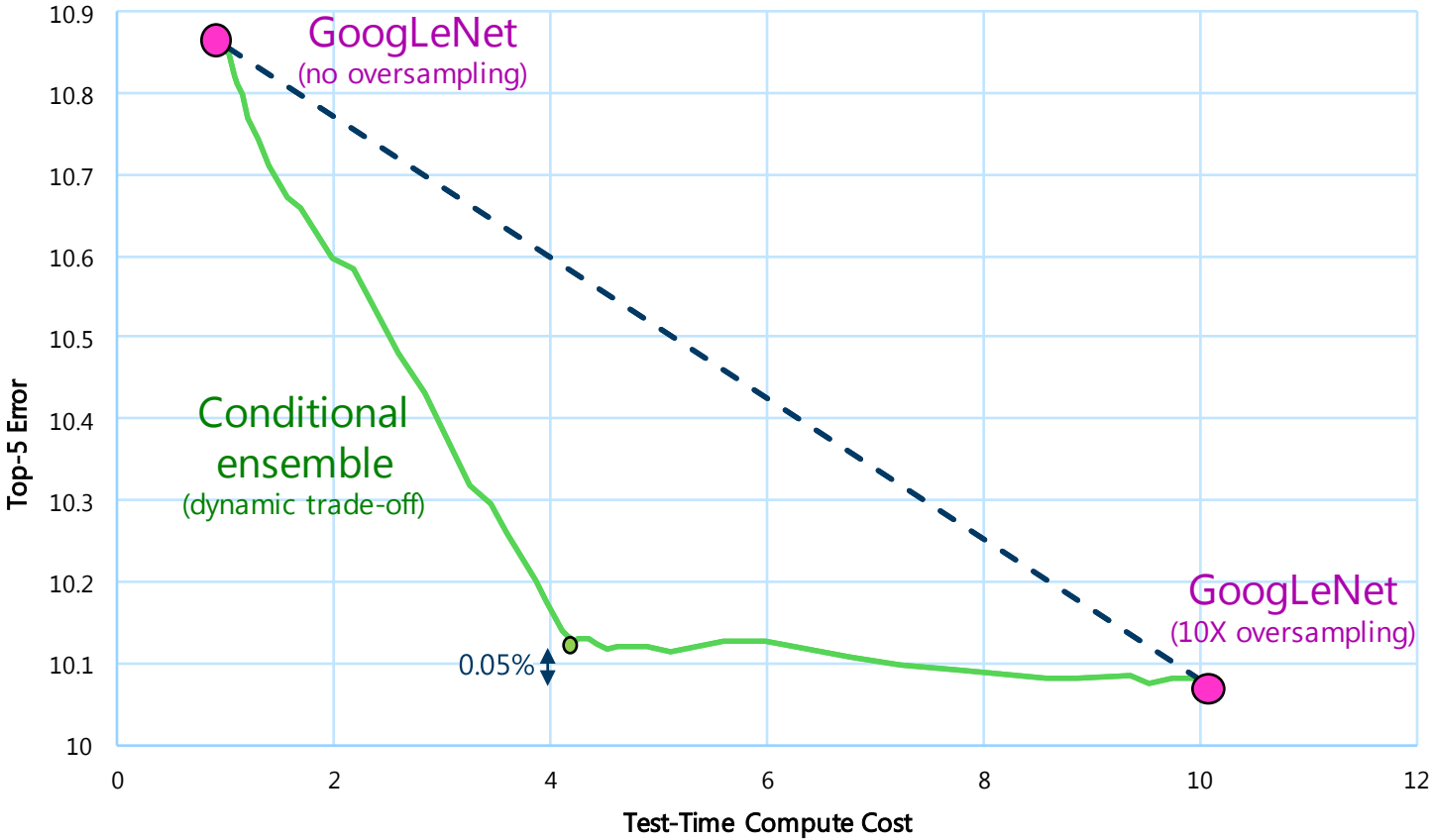
Conditional ensembles



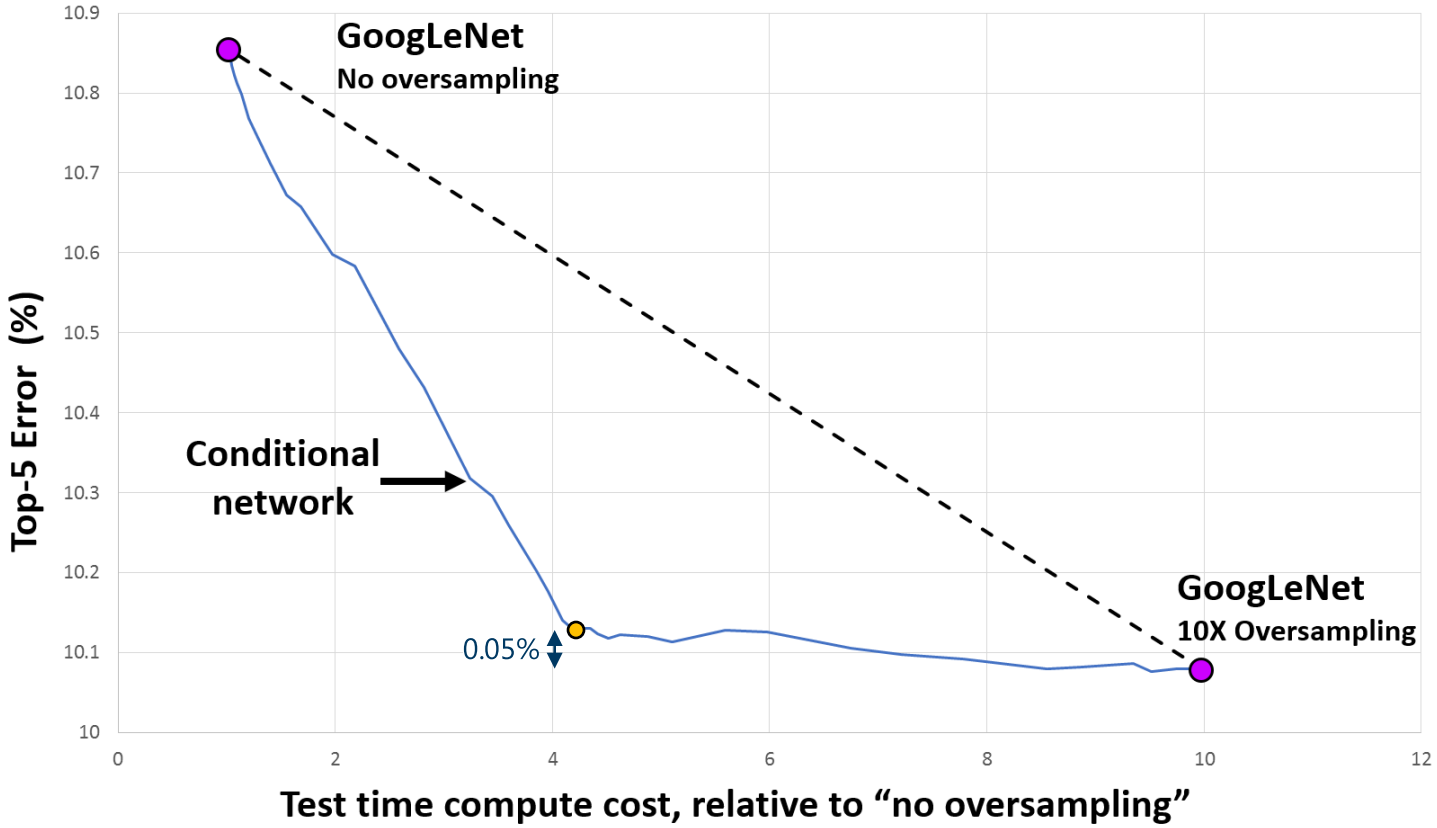
Conditional ensembles



Conditional ensembles



Conditional ensembles



Outline

- Introduction
  - Trees, DAGs and deep neural networks (DNNs)
  - *Motivation:* Representing trees/DAGs as MLPs
  - *Motivation:* ReLUs and data routing
- Conditional networks
  - The model
  - Training via back-propagation
- Experiments and results
  - Conditional sparsification of a perceptron
  - Comparing architectures on ImageNet
  - Comparing architectures on CIFAR
  - Conditional ensembles
- Conclusion

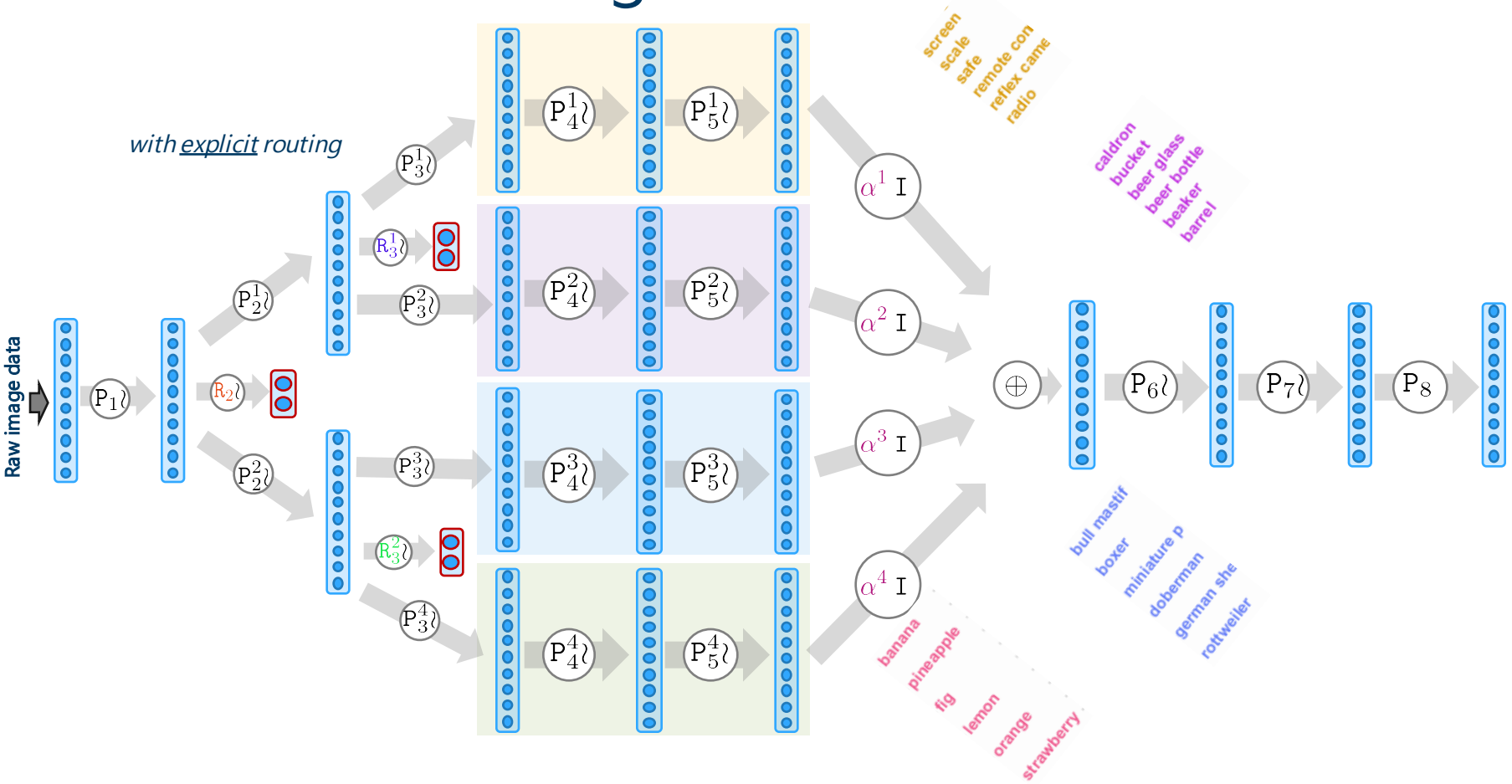
Conclusion

Conditional networks. A new, deep learning model with the following properties

- **Generalizes** decision forests, decision jungles and deep neural networks
- Can be **trained** entirely via back-propagation.
- **On-demand** selection of operating point = *f(accuracy, efficiency, size)*
- Higher **interpretability** than deep neural networks

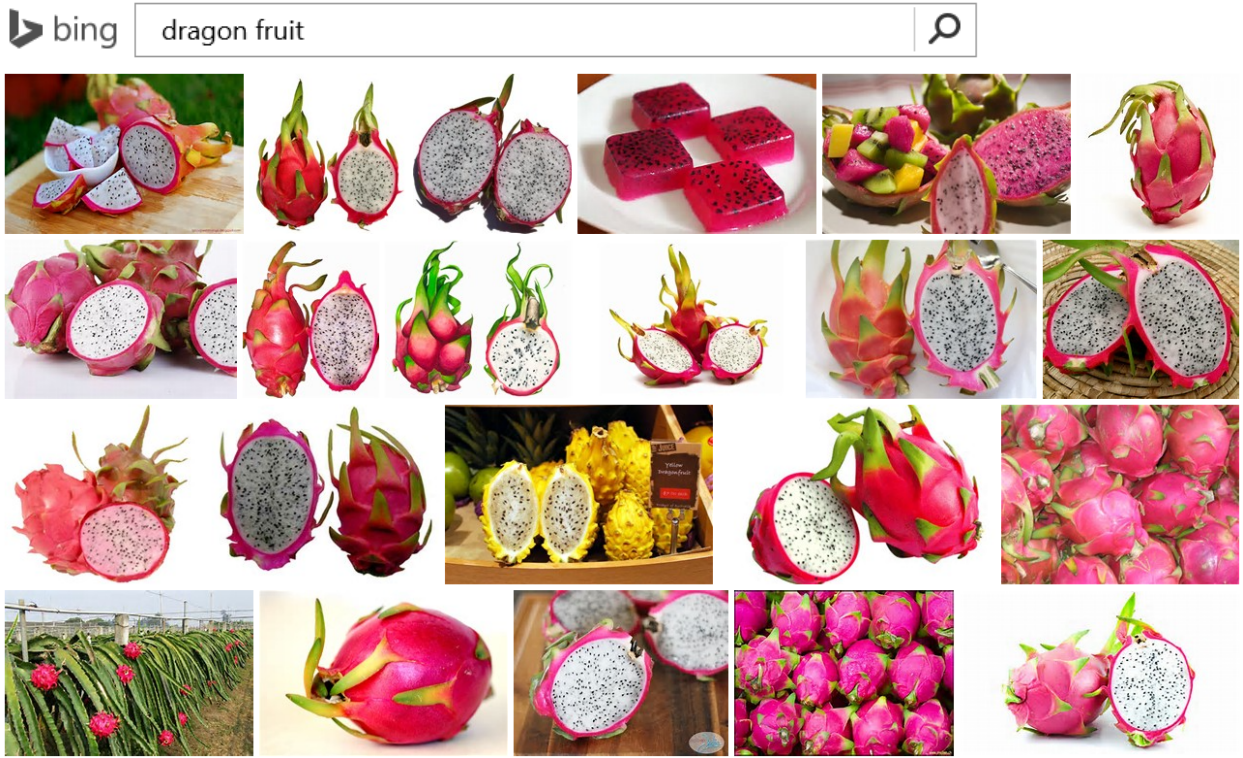
Appendix

On-demand learning of additional classes

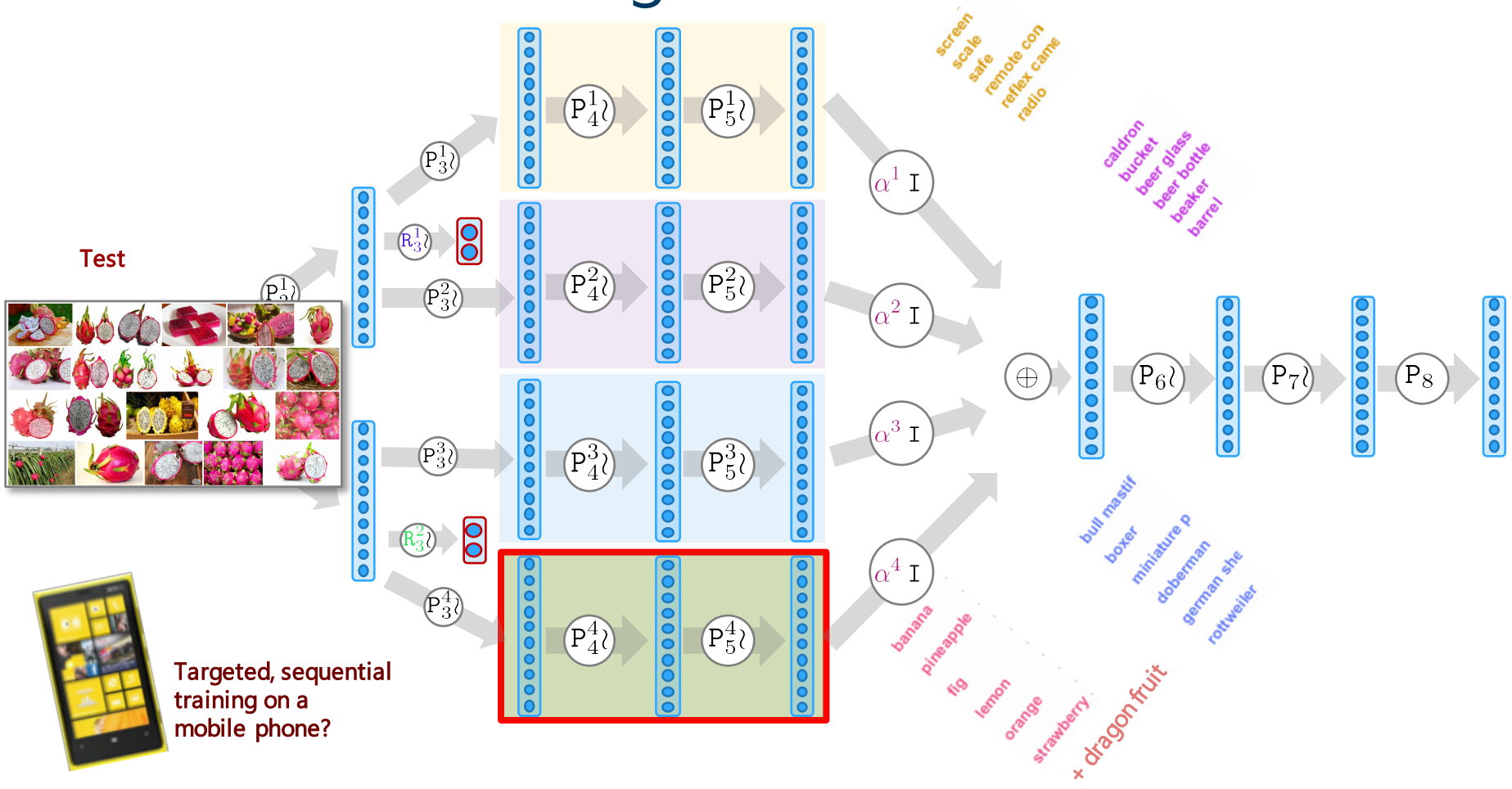


On-demand learning of additional classes

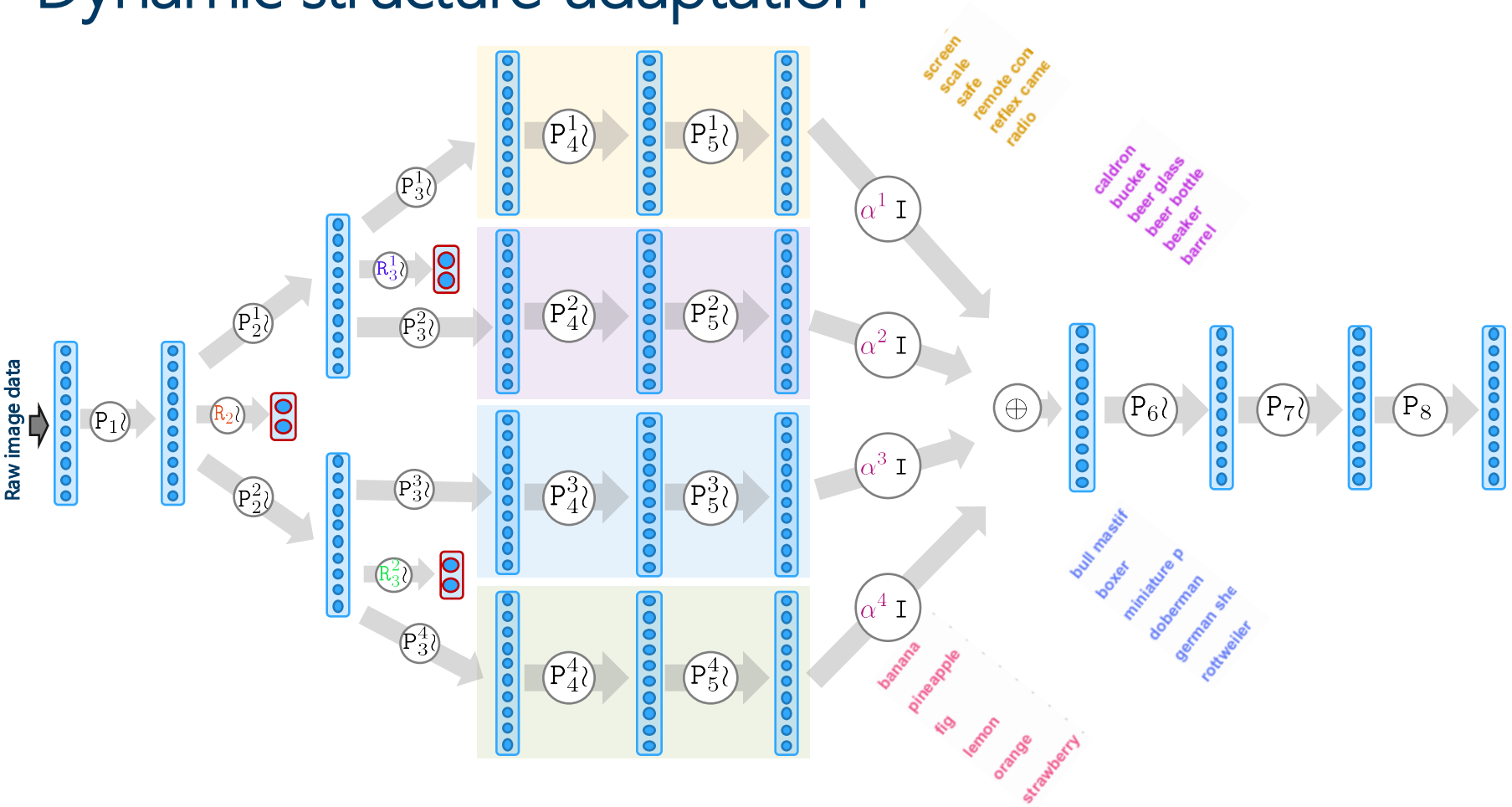
"Now I am also interested in dragon fruit"



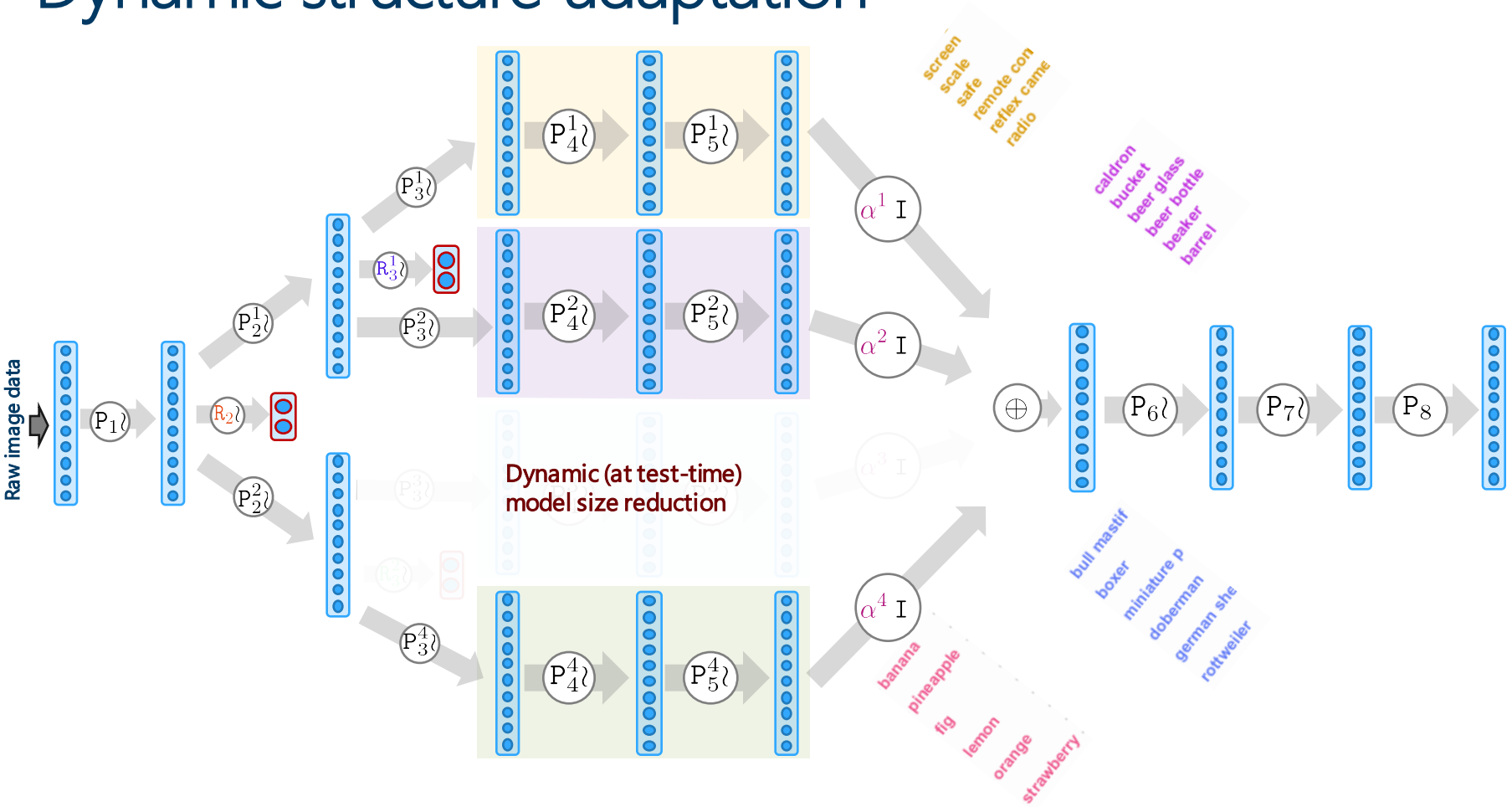
On-demand learning of additional classes



Dynamic structure adaptation



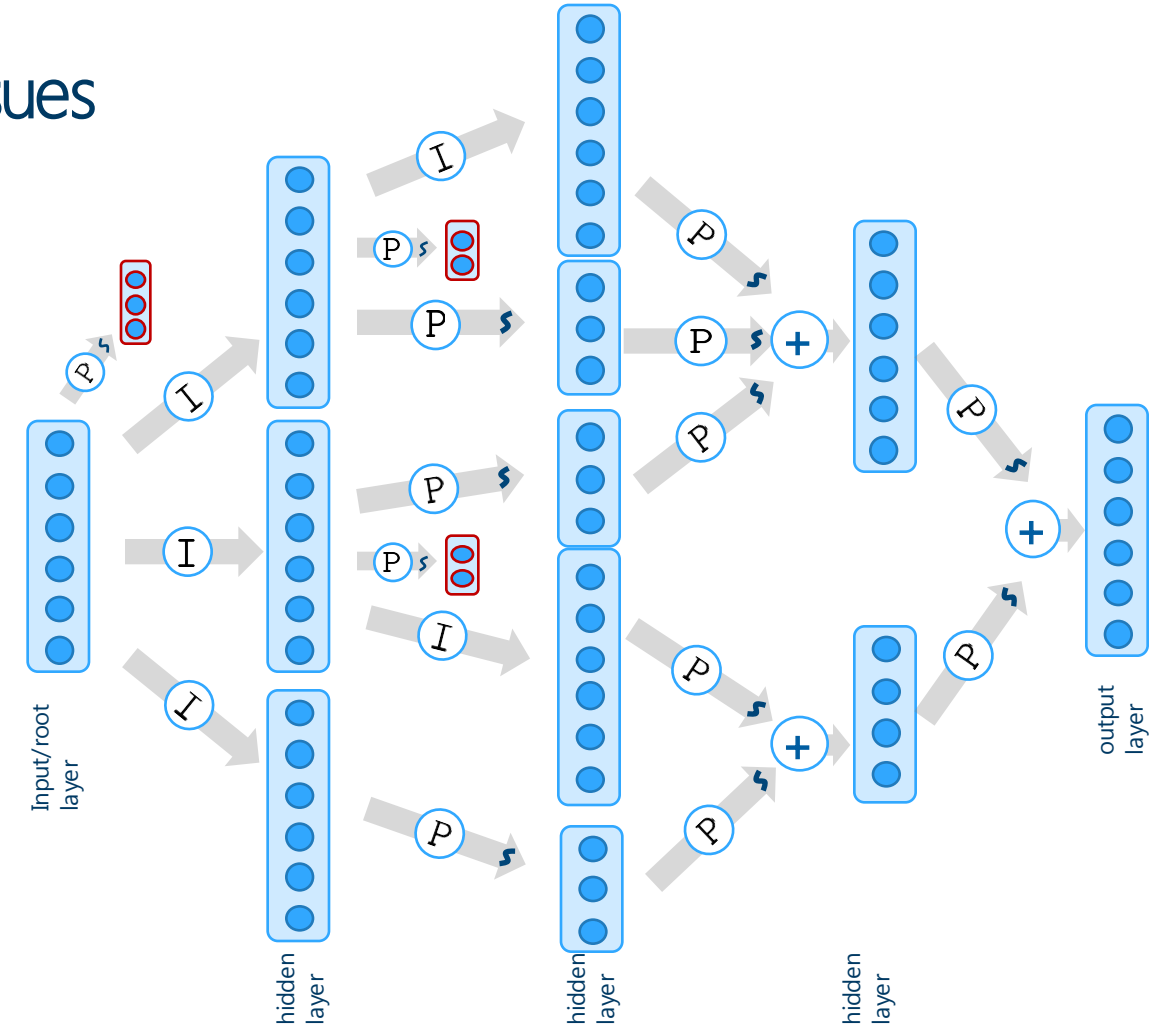
Dynamic structure adaptation



OLD UNUSED

Conditional networks some properties and issues

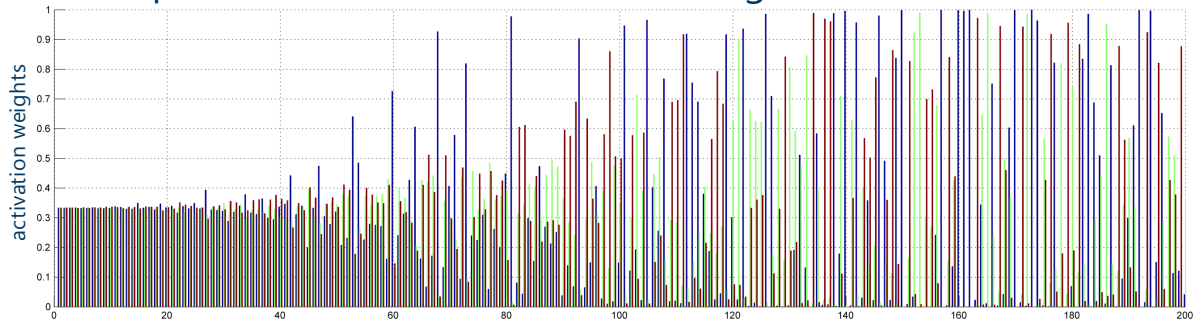
- If there is inherent structure in the input data we can exploit it and achieve saving. CuP can be seen as a way of sparsifying the weight (transformation) matrices
- Being able to learn the structure of the model is new compared to commonly used DNNs. Can we actually do it well?
- Cup vs MLP-forest.
  - In MLP forests we train a whole MLP at each node for data splitting. Here the MLP is distributed, a single layer at each node. It is trained directly for class ranking instead.
- Questions:
  - Can it all be learned via back-propagation? Perhaps the n-ary routing weights can be initialized greedily and then refined via gradient descent?
  - Can we / does it make sense to keep track of fractional class counts through nodes and empirical class distributions?
  - Is this way of achieving sparsification better than setting the lowest weights to 0 and use sparse matrix algorithms ?



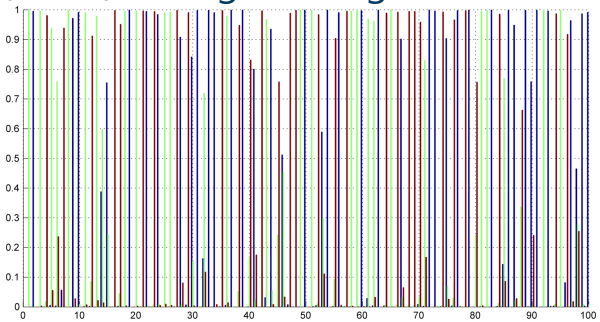
Training a classification-only CONNECT globally

From Darko

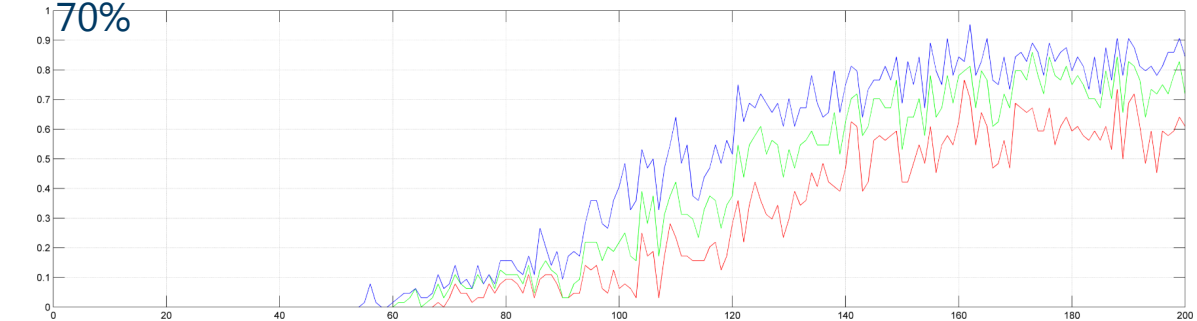
Examples of evolution of activation weights for the 3 different routes (R,G,B) during training



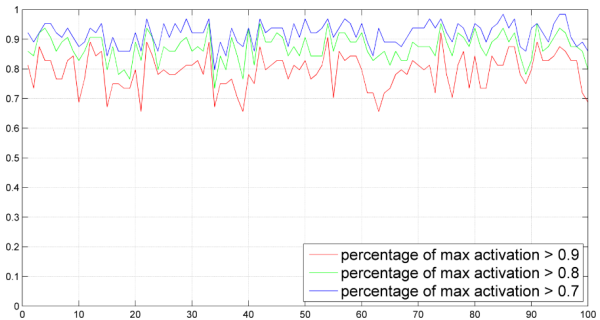
…



Influence of Individual Route: Percentage of maximum activation larger than thresholds of 90%, 80%, 70%



…



Final percentage of max activation > 0.90 = 0.7927
Final percentage of max activation > 0.80 = 0.8725
Final percentage of max activation > 0.70 = 0.9184

Training the router

Surrogate training energy

$$E = \sum_{i=1}^{>10^6} \sum_{j=1}^{S} w_j \; [s_i^* = s_j(\mathbf{v}_i)]$$

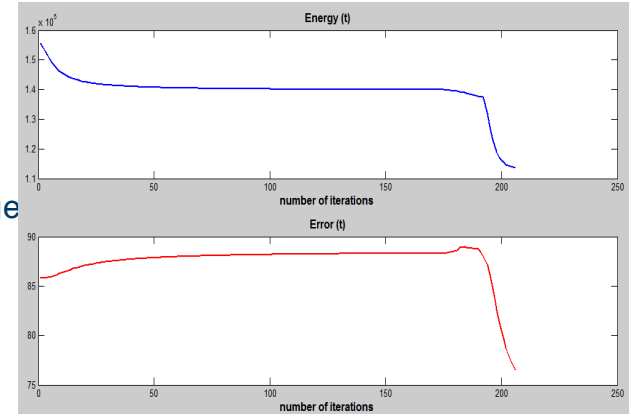
$\mathbf{v}_i$ is the 4096-dimensional descriptor for image i

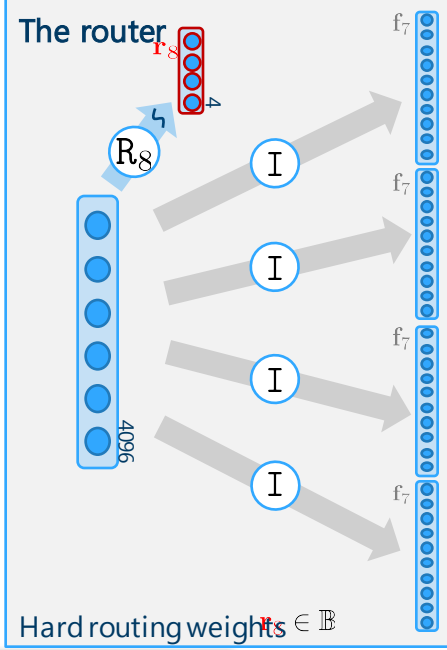
s_j denotes a super-class, out of S of them. $(S = 8)$

$s_j(i) \;\; s.t. \;\; a_{j-1} > a_j > a_{j+1}$

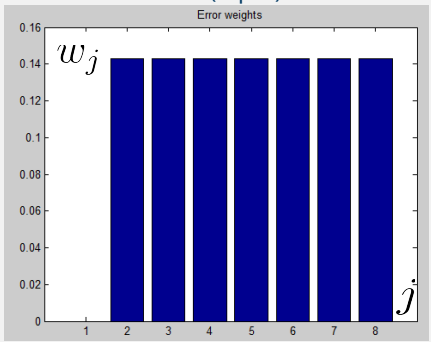
a_j denotes activation for superclass j

Key to have randomized, numerical gradient descent work is to have a smooth surrogate training energy.

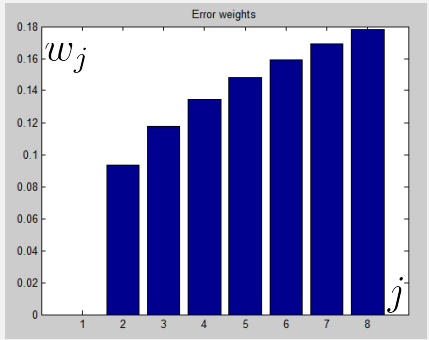




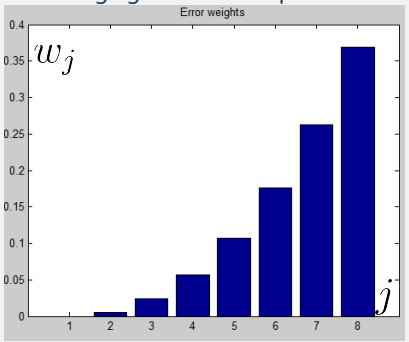
Conventional super-class classification error (top-1)



Smoother energy



Smoother energy
encouraging mass into top-ranked branches

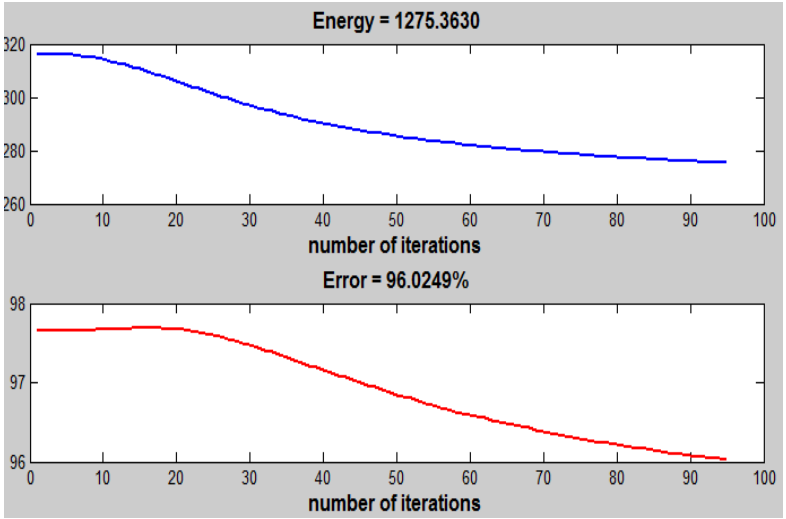


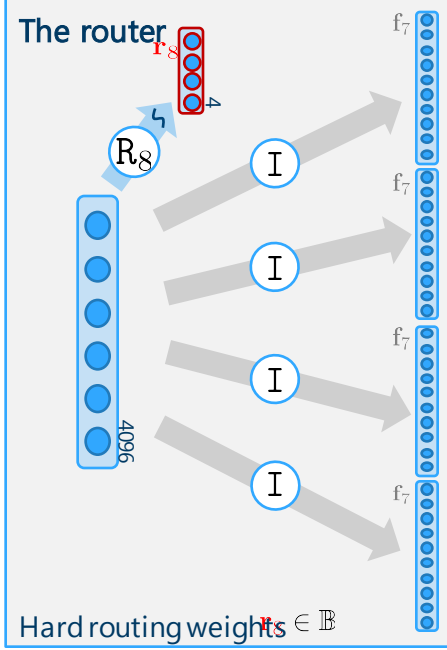
Slower convergence but lower local minimum?

Conditional specialization of an existing DNN

Training the router

Iterating
- PSO, Particle swarm optimization
(on a smoother surrogate energy), and
- Class to superclass reassignment





The K-Hyperplanes optimization algorithm:

1. Initialize class-to-superclass assignments, e.g. via Kmeans on fc7. (The Kmeans centres are not used, just the class assignments
2. Initialize router projection with random values
3. numerical gradient descent on the router matrix to minimize soft energy for ranking branches (in the spirit of random forests).
4. reassign classes to superclasses using current router projection and go to 3.

The above works well, but initializing the router projection (point 1.) from the already available P8 matrix leads to faster convergence.

Conditional Networks: summary

Contributions

- Demonstrated **equivalence** of trees, dags and two-layer perceptrons.

Rewrite this

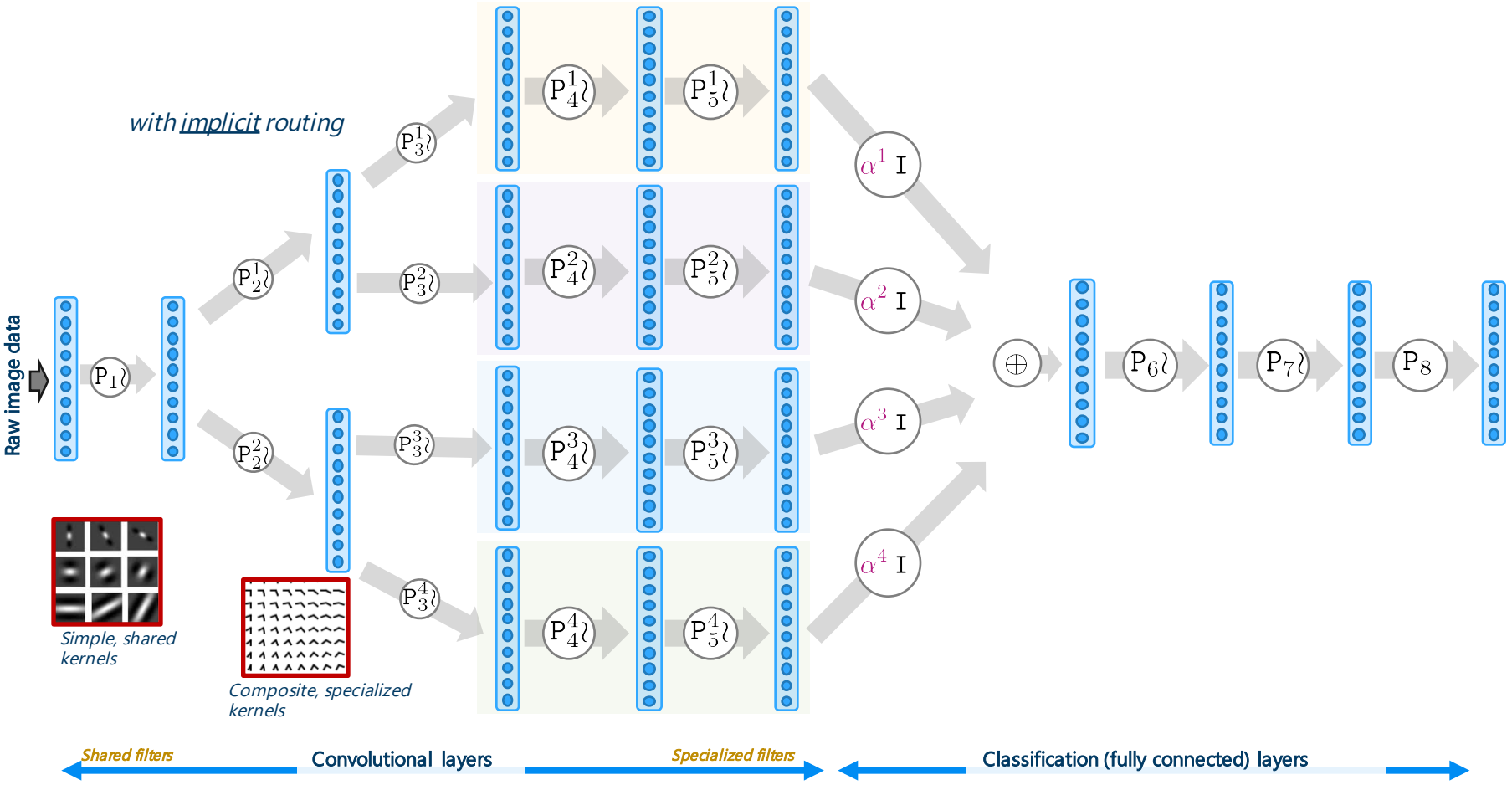
- **Conditional Networks generalize** neural networks and n-ary decision forests and jungles.
- With the right choices of model parameters we can potentially obtain **saving in terms of compute and memory.**
- We should be able to train the whole model globally via **back-propagation.**
- We now have a means of **regularizing** MLP training so as to prefer a particular kind of efficient structure, via **lateral inhibition.**
- We should be able to also learn (part of) the network **structure** rather than defining it all by hand.
- Combining this with the use of **separable filters** promises further gains
- Easily implementable within existing GPU-based codebases such as **Caffe**.

Summary 2

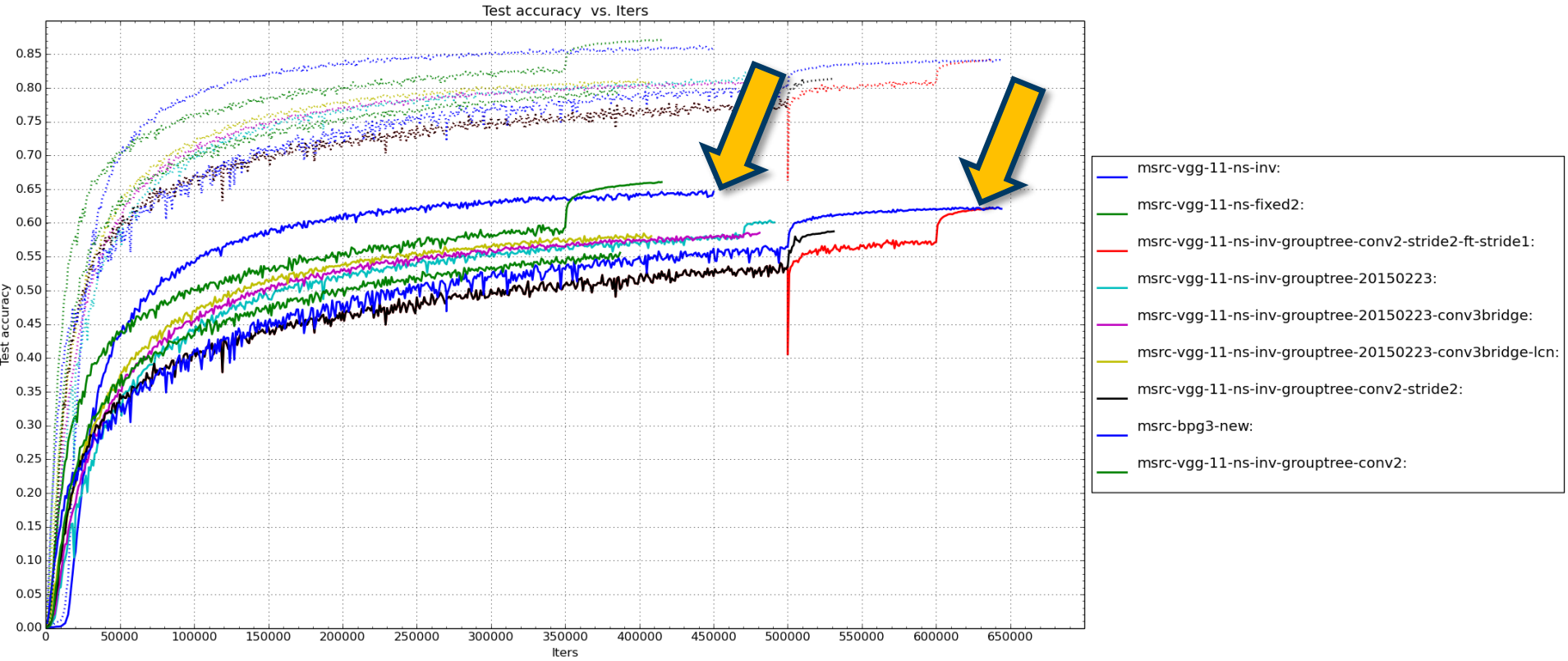
Rewrite this

- Decision trees/DAGS can be represented as a special form of 2-layer perceptrons and thus they can be trained via back-propagation.
- Deep Networks (with ReLu activations) implement a special routing mechanism. Such routing can be made explicit in the model architecture to achieve higher efficiency. This gives rise to a new learning model that we call "Conditional Networks".
- In summary. Conditional Networks represent a new model which bridges the gap between decision trees/DAGs and (deep) neural networks. They can represent both. Some model choices allow us to get the best in terms of efficiency-vs-accuracy.
-

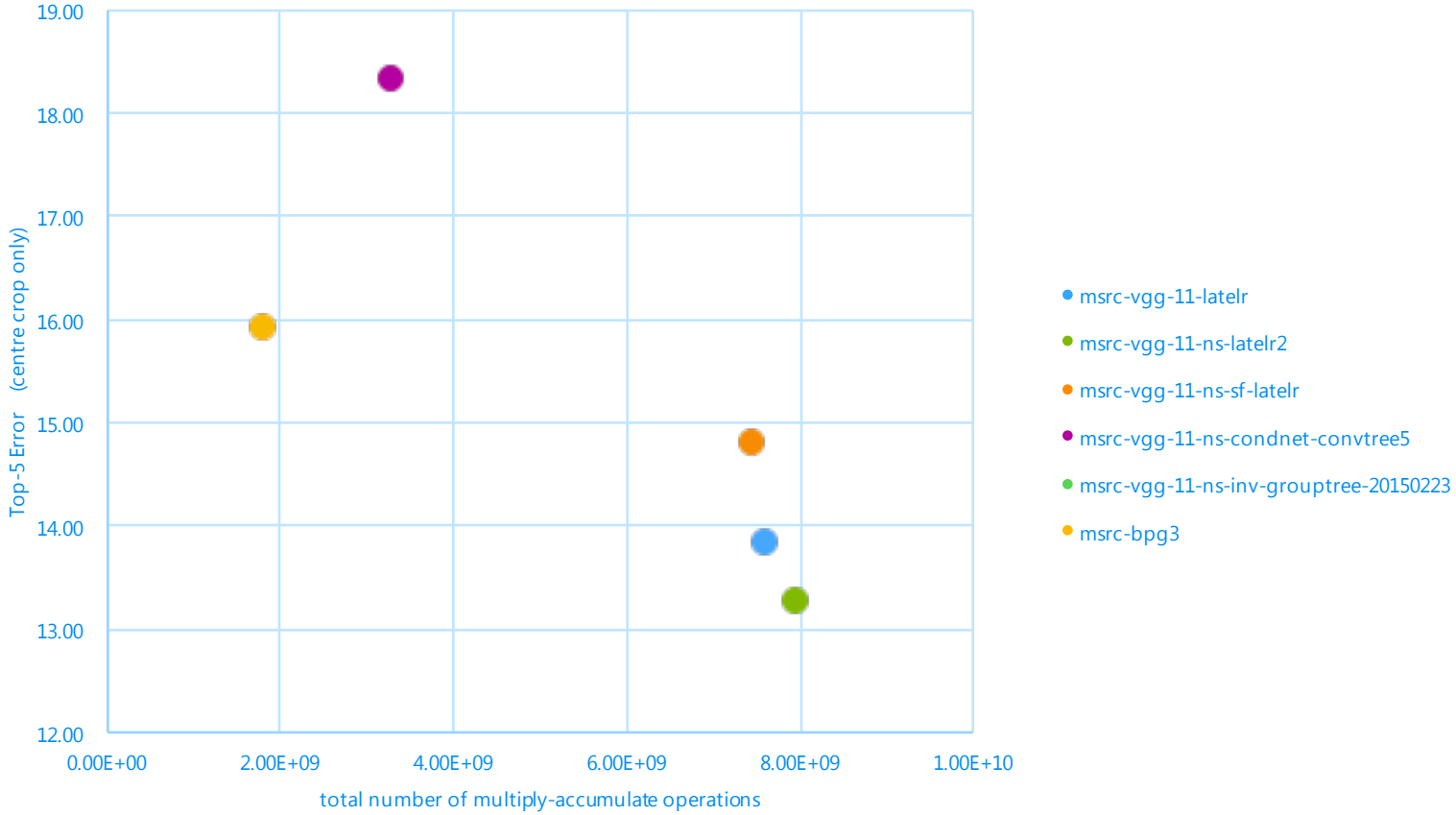
Results on ImageNet (1,000 classes)



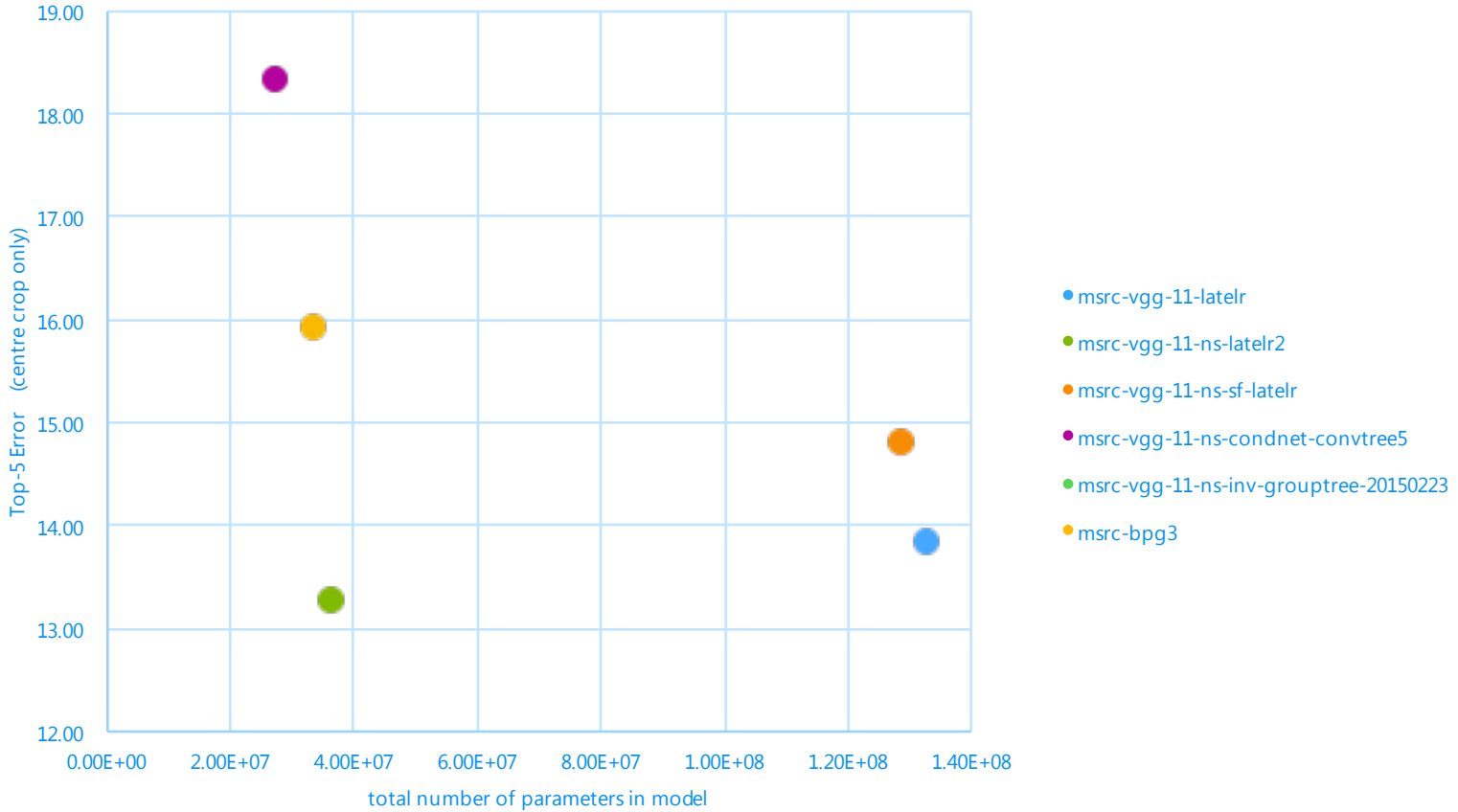
Results on ImageNet (1,000 classes)



Results on ImageNet (1,000 classes)



Results on ImageNet (1,000 classes)



Main results

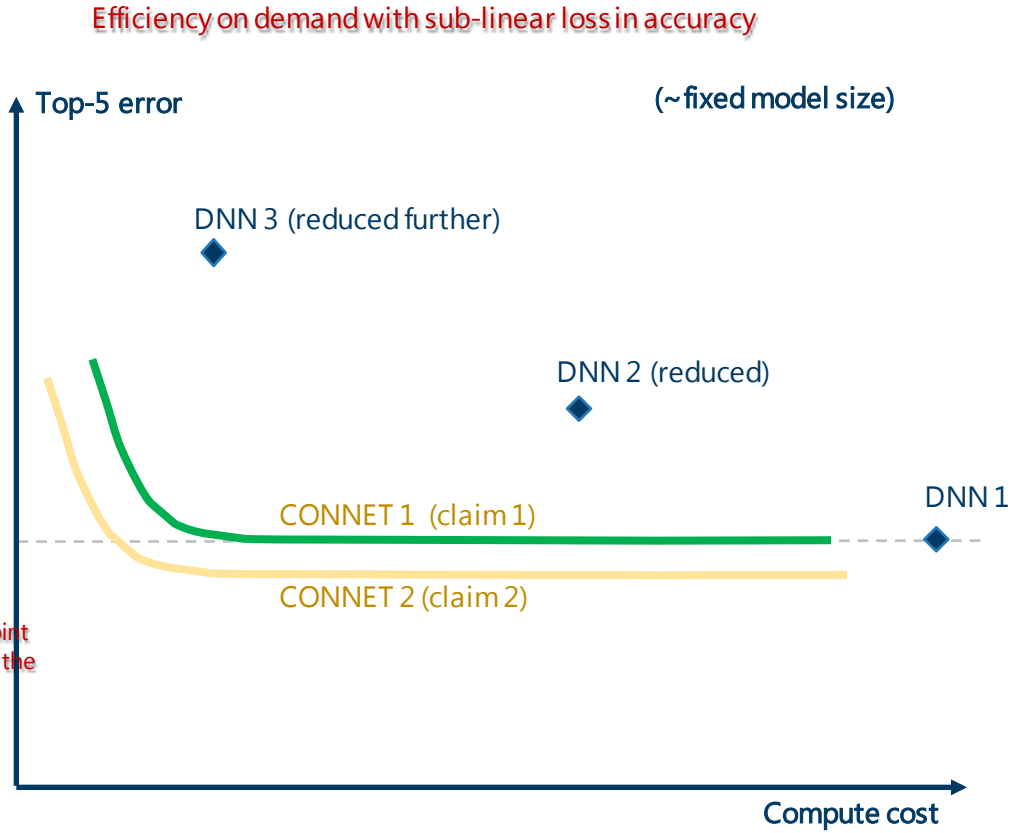
- ***Claim 1.*** **Greater efficiency, on the fly**
  - In a DNN the point (accuracy,efficency) is fixed and unchangeable unless we train a new DNN.
  - In a CONNET, a whole curve is available for a single trained CONNET. The optimal (accuracy,efficiency) can be changed on the fly, at TEST time, based on the dynamically changing application requirements.

- ***Claim 2.*** **Greater efficiency and accuracy**
  - training a CONNECT from scratch also achieves **better generalization?**

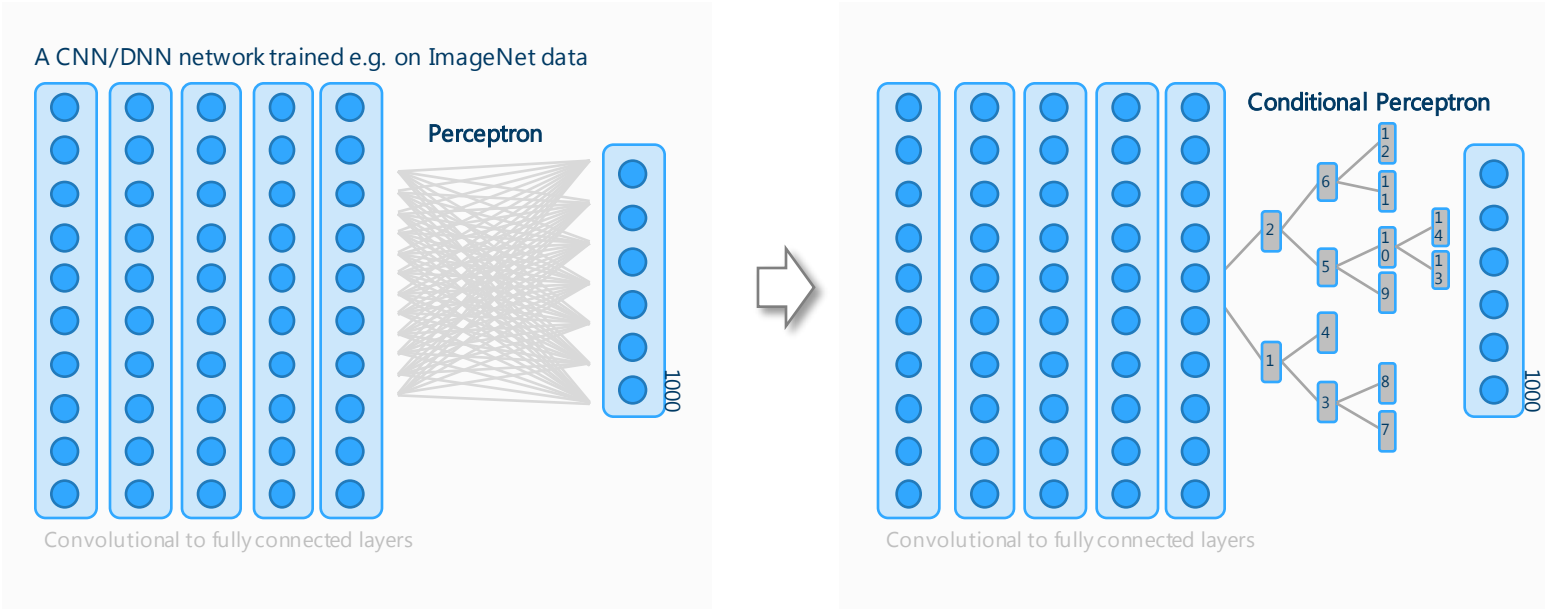
Efficiency on demand with sub-linear loss in accuracy

example in a phone app, as day progresses the battery is drained. At some point we automatically adjust the operating point so as to reduce the accuracy and the test time cost, thus reducing the speed of battery consumption, on the fly.

We need to make sure we speed up also the early convolutional layers!



Conditional sparsification of a perceptron

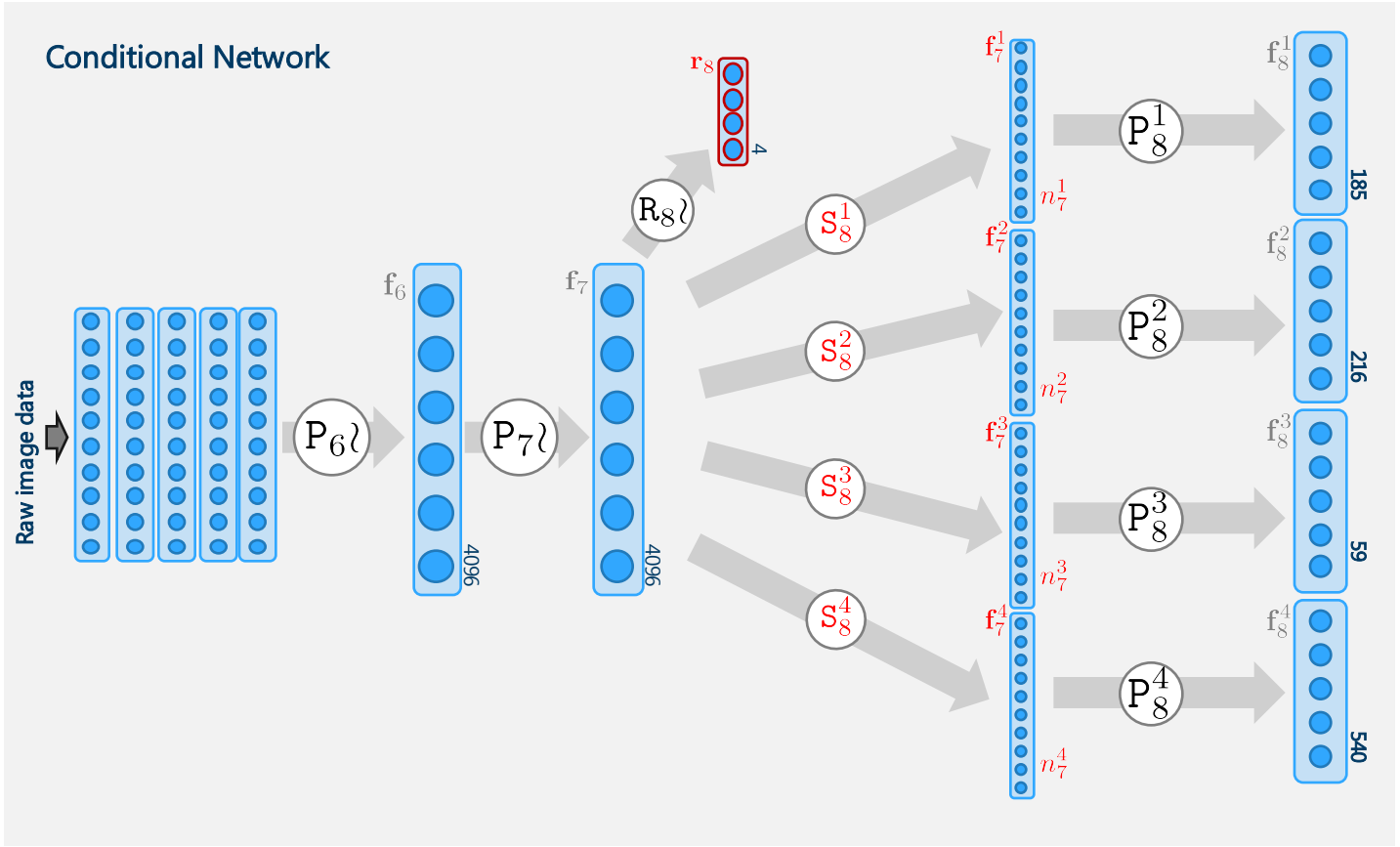


Conditional sparsification of a perceptron

Results 2. model compression

Experiment demonstrate effective sparsification of a single perceptron. No deep architecture yet.

Now reducing model complexity by removing unnecessary features from the various tree branches



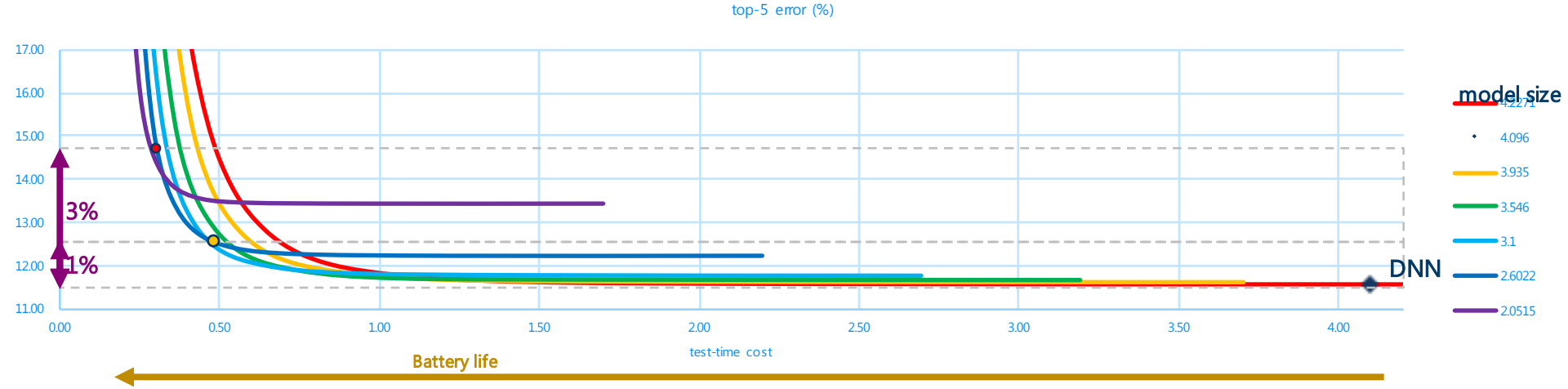
Hard routing weights $r_8 \in \mathbb{B}$

$\cup f_7^j = f_7 \quad n_7^j \leq 4096 \quad \sum_j n_7^j \geq 4096 \quad \sum_j n_8^j = 1000$

Conditional sparsification of a perceptron

Results 2. model compression (final layer only)

Experiment demonstrate effective sparsification of a single perceptron. No deep architecture yet.



Summary of results

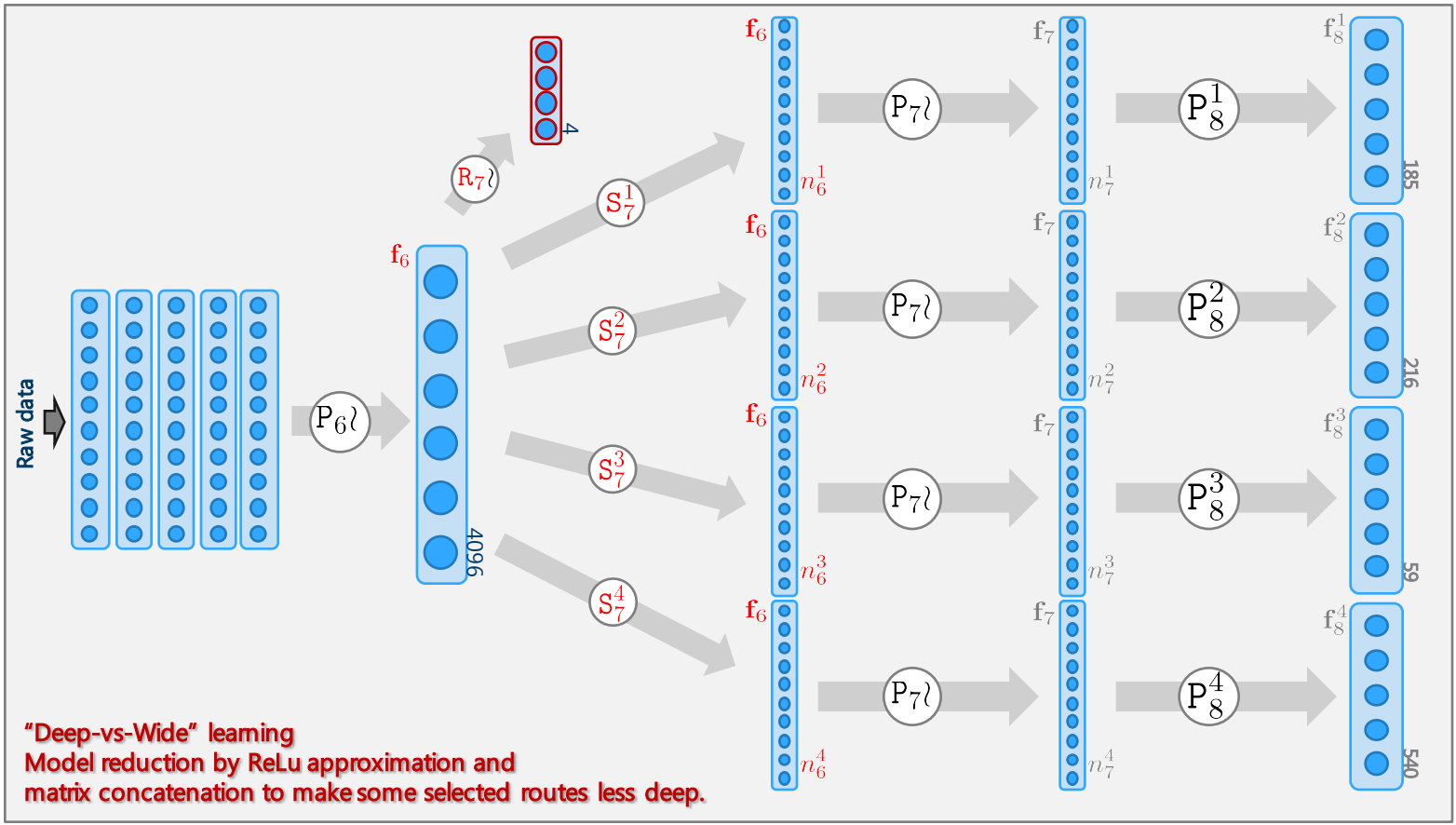
- For 1% increase of T5E: ~ X 9 speed up ~2/3 of model size
- For 3% increase of T5E: ~ X 11 speed up ~0.5 of model size

Conditional sparsification of a multi-layer perceptron

Unzipping the net

$$\mathbf{f}_8^i = \mathbf{f}_7 \mathrm{P}_8$$
$$\mathbf{f}_7 = \sigma(\mathbf{f}_6 \mathrm{P}_7)$$
$$\mathbf{f}_8^i = \sigma(\mathbf{f}_6 \mathrm{P}_7)\mathrm{P}_8^i$$
$$\mathbf{b}_7^i = [(\mathbf{f}_6 \mathrm{P}_7) > 0] \in \mathbb{B}^{4K}$$
$$\tilde{\mathrm{P}}_t^i = \mathrm{P}_7 \ \ \mathrm{diag}(\mathbf{b}_7^i)$$
$$\mathbf{f}_7^i := \mathbf{f}_6 \tilde{\mathrm{P}}_7^i \approx \mathbf{f}_7$$
$$\mathbf{f}_8^i = \mathbf{f}_6 \ \ \tilde{\mathrm{P}}_7^i \mathrm{P}_8^i$$
$$\tilde{\mathrm{P}}_7^i \mathrm{P}_8^i \in \mathbb{R}^{4K \times N_t}$$

(transposed)



"Deep-vs-Wide" learning
Model reduction by ReLu approximation and
matrix concatenation to make some selected routes less deep.

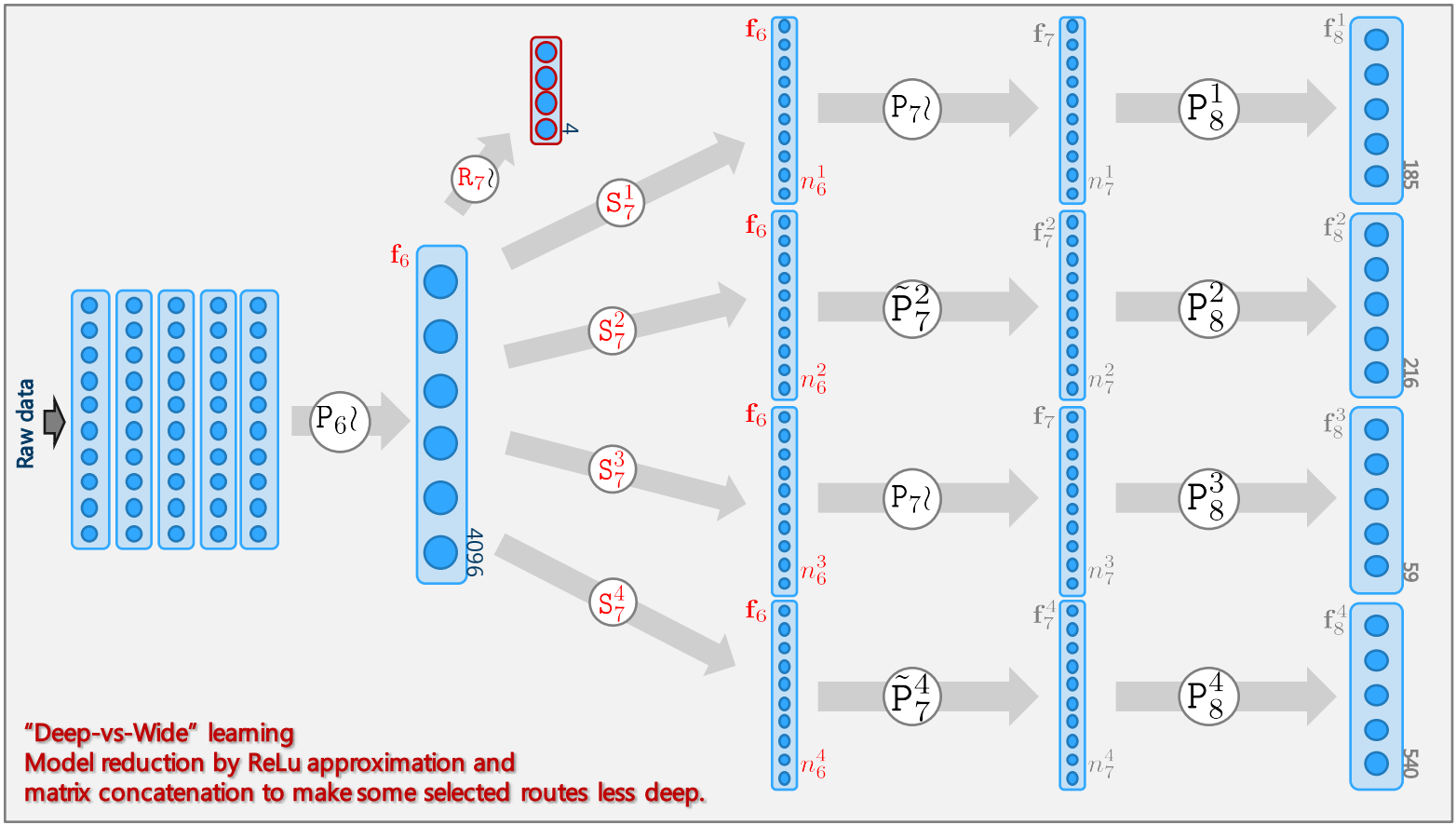
$$\cup \mathbf{f}_7^j = \mathbf{f}_7 \qquad \cap \mathbf{f}_7^j = \emptyset \quad n_7^j \le 4096 \qquad \sum_j n_7^j \ge 4096 \qquad \sum_j n_8^j = 1000$$

Conditional sparsification of a multi-layer perceptron

Unzipping the net

$$\mathbf{f}_8^i = \mathbf{f}_7 \mathrm{P}_8$$
$$\mathbf{f}_7 = \sigma(\mathbf{f}_6 \mathrm{P}_7)$$
$$\mathbf{f}_8^i = \sigma(\mathbf{f}_6 \mathrm{P}_7) \mathrm{P}_8^i$$
$$\mathbf{b}_7^i = [(\mathbf{f}_6 \mathrm{P}_7) > 0] \in \mathbb{B}^{4K}$$
$$\tilde{\mathrm{P}}_t^i = \mathrm{P}_7 \ \ \mathrm{diag}(\mathbf{b}_7^i)$$
$$\mathbf{f}_7^i := \mathbf{f}_6 \tilde{\mathrm{P}}_7^i \approx \mathbf{f}_7$$
$$\mathbf{f}_8^i = \mathbf{f}_6 \ \ \tilde{\mathrm{P}}_7^i \mathrm{P}_8^i$$
$$\tilde{\mathrm{P}}_7^i \mathrm{P}_8^i \in \mathbb{R}^{4K \times N_t}$$

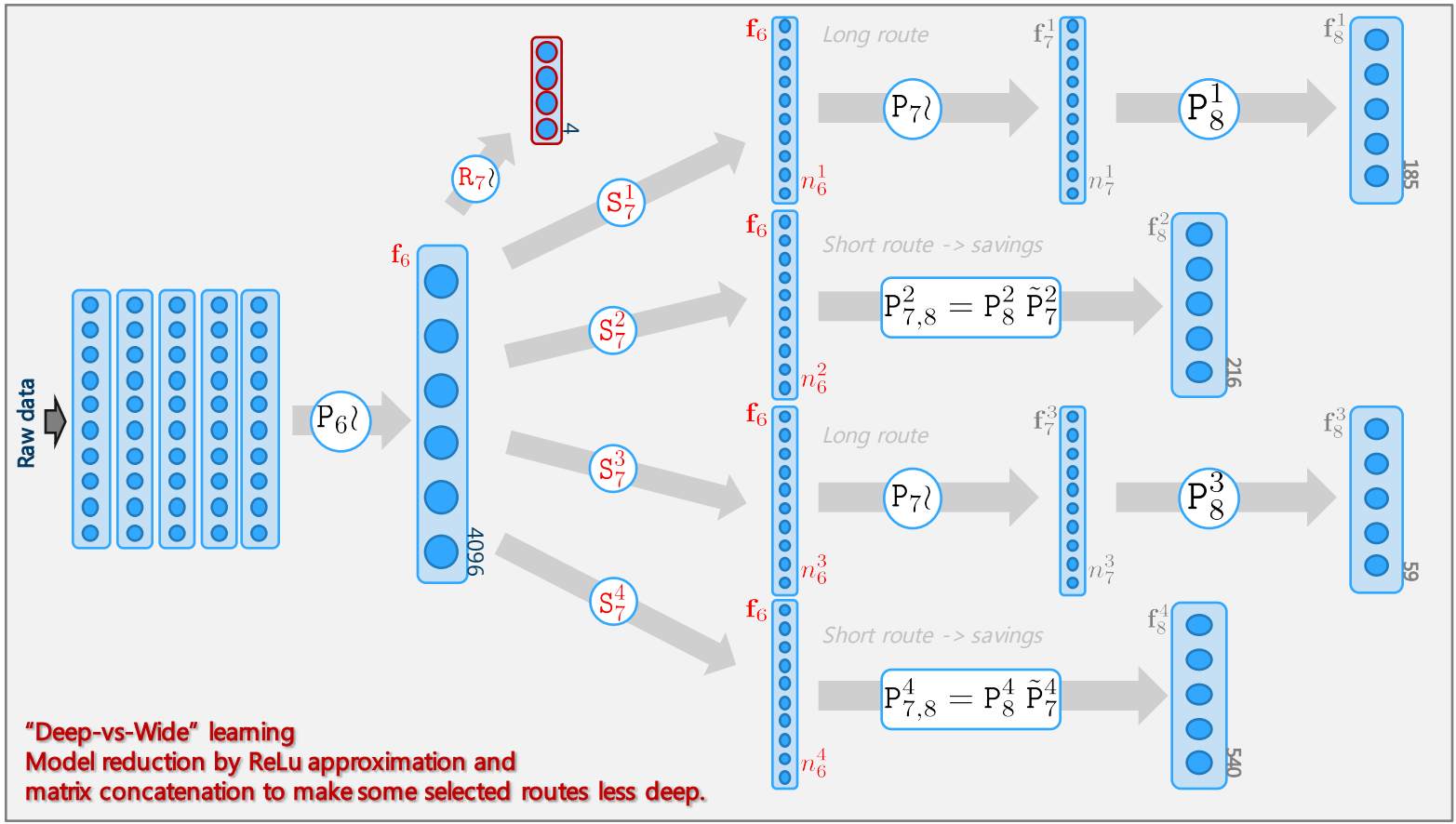
(transposed)



"Deep-vs-Wide" learning
Model reduction by ReLu approximation and
matrix concatenation to make some selected routes less deep.

Conditional sparsification of a multi-layer perceptron

Unzipping the net



"Deep-vs-Wide" learning
Model reduction by ReLu approximation and
matrix concatenation to make some selected routes less deep.

$$\mathbf{f}_8^i = \mathbf{f}_7 \mathrm{P}_8$$
$$\mathbf{f}_7 = \sigma(\mathbf{f}_6 \mathrm{P}_7)$$
$$\mathbf{f}_8^i = \sigma(\mathbf{f}_6 \mathrm{P}_7)\mathrm{P}_8^i$$
$$\mathbf{b}_7^i = [(\mathbf{f}_6 \mathrm{P}_7) > 0] \in \mathbb{B}^{4K}$$
$$\tilde{\mathrm{P}}_t^i = \mathrm{P}_7\ \ \mathrm{diag}(\mathbf{b}_7^i)$$
$$\mathbf{f}_7^i := \mathbf{f}_6 \tilde{\mathrm{P}}_7^i \approx \mathbf{f}_7$$
$$\mathbf{f}_8^i = \mathbf{f}_6\ \ \tilde{\mathrm{P}}_7^i \mathrm{P}_8^i$$
$$\tilde{\mathrm{P}}_7^i \mathrm{P}_8^i \in \mathbb{R}^{4K \times N_t}$$

(transposed)

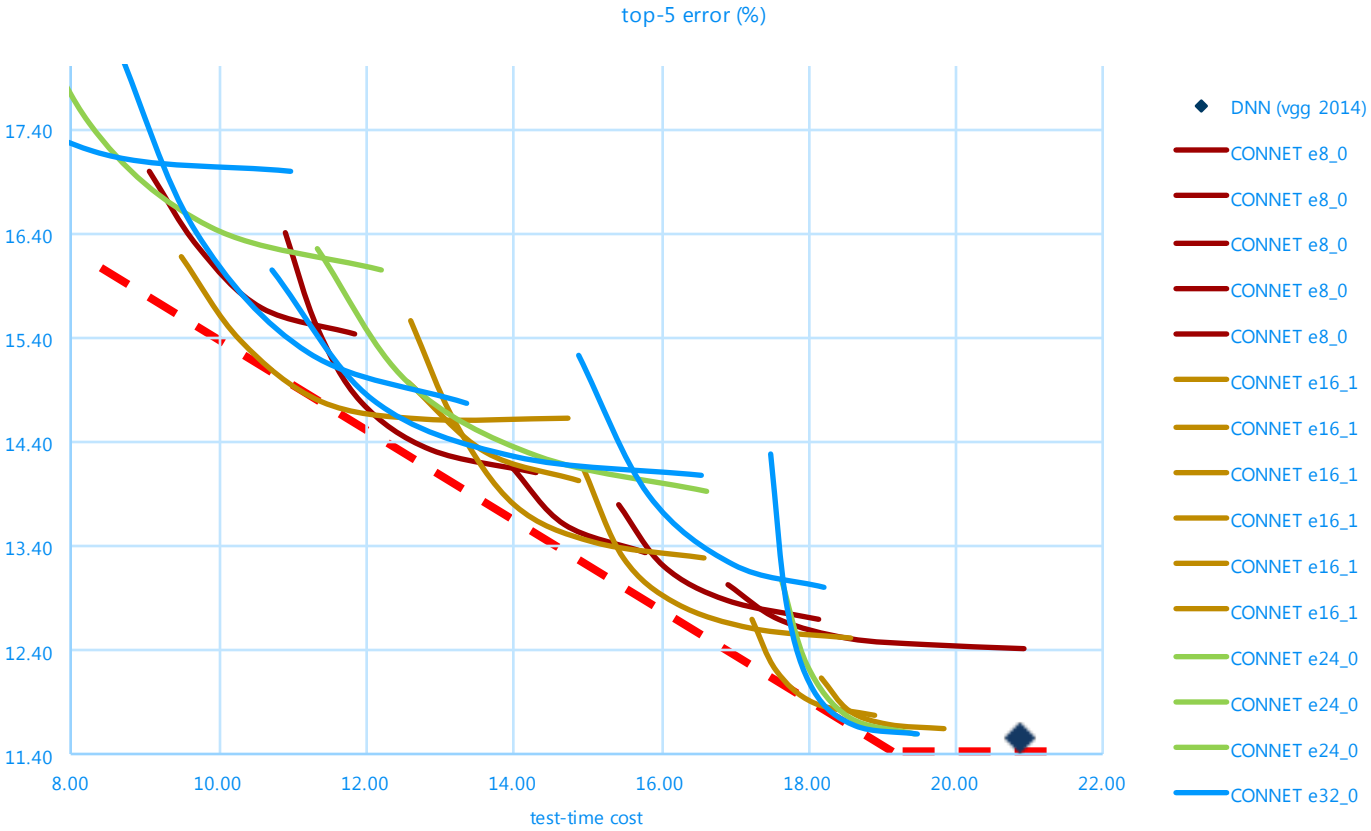
Learning the structure of the conditional network automatically, from data.

Conditional sparsification of a multi-layer perceptron

Unzipping the net: results

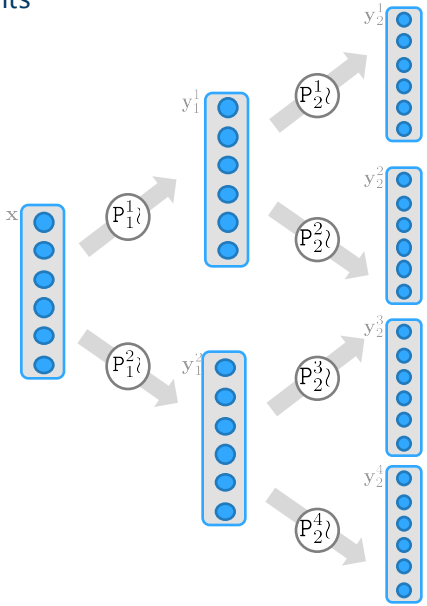
Some savings, but not great savings here.

We need to train the CONNET globally, rather than in a post-hoc fashion as done so far.

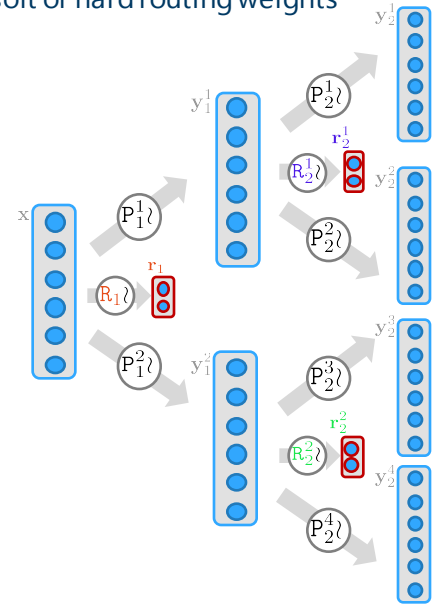


Training a conditional network via back-propagation

Implicit routing
- Routes do not represent super-classes
- Routers trained with pure back-propagation, for class discriminability.
- Using soft weights



Explicit routing
- Routes represent sets of classes (super-classes)
- Routers trained with priors to "group" similar classes together
- Using soft or hard routing weights



Training in practice
1. We train an implicit routed conditional network, and
2. Train explicit routers afterwards.

This seems to work better in practice than training everything globally via back-prop.