Topics:

- Backpropagation
- Matrix/Linear Algebra view

CS 4644-DL / 7643-A
ZSOLT KIRA

- **Assignment 1 out!**
  - **Due June 5th (with grace period June 7th)**
  - Start now, start now, start now!
  - Start now, start now, start now!
  - Start now, start now, start now!

- Resources:
  - These lectures
  - Matrix calculus for deep learning
  - Gradients notes and MLP/ReLU Jacobian notes.
  - **Topic OH:** Assignment 1 and matrix calculus (TBA)

- Piazza: Project teaming thread
  - **Project Proposal: June 15th, Project Check-in: July 1st .**
  - Project proposal overview during my OH (Thursday 12:30pm ET, recorded)

The main goal of the project proposal is to:

1. Clearly state the problem you aim to investigate.
2. Explain why it is interesting or important (the motivation).
3. Describe how you plan to address it (methods, experiments, datasets).
4. Provide a short overview of related work and background reading.

Your proposal is a chance for us to give you feedback on your project direction, so don't worry if all your ideas are not fully concrete yet. Think of it as a condensed version of the "Introduction" and "Related Work" sections of a typical Computer Vision, NLP, or Robotics conference paper.

Some important dates: the project proposal is due on Feb 14th (11:59pm EST)

Your proposal should address the following questions and points:

1. What: What problem are you investigating?
2. So What: Why is it interesting or significant?
3. Now What: What course of action or methodology do you propose?
4. Related Work & Background: Summarize the readings or papers you have examined that provide context.

Additionally, please include *initial thoughts* on the following, even if they are not fully concrete:

- Uniqueness: How does your approach differ from (or build upon) prior works?
- Benchmark/Dataset/Setup: What benchmark, dataset, or experimental setup will you use?
- Algorithms: What class of algorithm(s) will you propose or implement?
- Evaluation: How will you measure performance (quantitative) and analyze results (qualitative/visualization)? What kinds of analysis do you expect to run?
- Risks/Roadblocks: What potential challenges do you foresee, and how might you address them?

You are free to choose any topic you would like. Below are some suggestions if you are having trouble:

FAQs:

I don't have a group. Can I do my project solo?
No, you will be penalized for not gaining approval for a solo group.

It's the week/day/hour before the proposal deadline, and I couldn't find a group. Can I do my project solo?
This is insufficient justification. You will be penalized for not gaining approval for a solo group.

I don't have a group. When should I start panicking?
We are releasing this info now, but note the proposal is not due until Feb 14. Expect churn in groups until early Feb (students will drop, find other topics of interest, etc.). Ideally, you want to start locking down groups after assignment 2 is due and begin experimenting with ideas before finalizing a project topic.

Can we use existing implementations?
Yes, you can use existing implementations (do not forget to cite). However, remember that your project must be comprehensive per the project guidelines; don't provide just the results and shallow analysis. If you use existing models/code, this is even more important, as we can't judge how much you have learned from that. The key is to iterate between running, making hypotheses/claims about what's happening (with evidence, which you should show), and documenting the decisions you made in response to improving things. Also, be cautious about unofficial implementations. Often, they have bugs; we typically try to avoid using implementations that don't reproduce the results from the paper. You can often tell as the good ones have tables comparing their performance to the paper in the README.

Changing topic after the proposal deadline?
It depends. It is a higher risk further in the term. Feel free to attend office hours to ask or make a private post if you encounter this. Ultimately, it's up to you and your group. The proposal doesn't lock you into anything; however, the expectations for the final report do not change, even if you changed topics the week before.

Can we use pre-trained models?
You are allowed to use pre-trained models. However, just running open-source code is not sufficient. What we're looking for in the project is making novel contributions in the field (can be novel in both algorithms and applications). In that case, using a pre-trained model and fine-tuning on a dataset is insufficient by itself. However, using a pre-trained model and doing some non-trivial DL as part of the pipeline is acceptable. Do not forget to cite.

Can we use libraries other than Pytorch?
Yes.

What makes a good project?
Usually, projects where the dataset is readily available and existing benchmarks are solid options. However, you are free to experiment.

Projects where you collect your own data take longer, require more infrastructure, and generally are not good candidates. We don't discourage passion projects, so if you have something in mind, go for it! Just be forewarned that you should likely start looking into the infrastructure now to ensure you have both the right data and enough data amenable to DL. An additional challenge here is that you will have to create baseline models (naive or otherwise) to show DL shows improvement over something.

Deep reinforcement learning (RL) projects are tough for the same reason. In deep RL, you'd be using a simulator that generates the data you train on. Often, these simulators require significant debugging to work, so be forewarned. Again, if you're passionate, don't let this stop you from pursuing projects in this space.

What type of resources are required to train deep neural networks (DNNs) for the project? Depending on the dataset, it can take hours or days for models to train on non-trivial datasets. If you haven't used GPUs or cloud computing before and plan to use them (often necessary for the Facebook projects or non-trivial datasets), plan on accounting for time to learn how to use these technologies.

An Example Project Proposal:

A sample will be posted on Canvas. We will make a post with a sample final project report after the proposal deadline.

Learning from Demonstrations using Parameter-efficient Finetuning of Language Models

Team Name: Precision Pilots

Team Members: Neelabh Sinha, Snigdha Verma, Ananya Sharma

January 26, 2025

1 Project Summary

The knowledge of Language Models (LMs) as next-word predictors have been proven to be very powerful recently. However, there is still a challenge in aligning their responses to user objectives. The ideal way is to finetune them for a downstream task, but that is inefficient and resource-consuming due to the scale of these models. Recently, in-context learning, which is prompting LMs with in-context examples to get results for a task under zero-shot settings, has also produced great results. It stands out for its efficiency; however, for long sequence tasks, it encounters limitations due to restrictive context size of LMs, i.e., a maximum cap of sequence length under which they produce good results. When providing many in-context examples, it is natural that the prompt size for long-sequence tasks can exceed this limit. We aim to bridge this gap between finetuning and in-context learning by building a technique that is efficient and improves outputs without expensive finetuning/training these large LMs on large datasets.

2 Approach

To address the above problem, we propose **Parameter-Efficient Fine-Tuning (PEFT)** of LMs on these downstream tasks with prompts containing in-context examples. This involves exploring efficient methodologies to enhance their performance in downstream tasks. These adapters are methods/architectures to use with Transformer-based models that can improve performance on downstream tasks by training none or significantly less number of parameters. This direction is promising, primarily because of two reasons:

1. It fine-tunes a much smaller number of parameters, solving the limitation of inefficiency of end-to-end supervised finetuning.
2. It can compensate for context size limitation with in-context learning as the finetuning will be done using the specific downstream tasks.

As in the paper [16], we will focus on two tasks to start: Natural Language Inference and Paraphrase Identification. The hypothesis is that the finetuning with prompts containing in-context examples will improve the context-size limitation of these models and the examples themselves will provide the LM with the information it needs to perform a new task.

We will start from pre-trained OPT [24] models from smallest to largest scale and experiment with different kinds of PEFT techniques. We have categorized those techniques into multiple types as listed in Table 1 and will try to implement at least one from each category to do a comparative study of how they generalize. Some of them are also illustrated in Figure 1.

The pre-trained models are available on HuggingFace[1].

Type	PEFT Techniques
Prompt-based Techniques	Prompt-tuning [12], Prefix-tuning [13], SPOT [20], IPT [17]
Representation-based Techniques	LoRA [11], KronA [4]
Series-based Techniques	Series Adapters [9], Adamix [21], Sparse Adapters [8], LeTS [5]
Parallel Techniques	Parallel Adapters [6]
Hybrid Techniques	MAM Adapters [7], UniPelt [15], Compacter [14], S4 [1]

Table 1: Existing Techniques for PEFT

[1] https://huggingface.co/docs/transformers/en/model_doc/opt

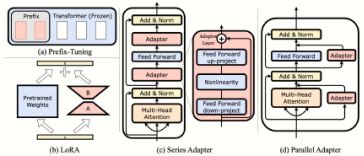


Figure 1: Some types of PEFT techniques

We will also study how these adapters behave with increasing scale of LMs. Since OPT provide range of scaled LMs from 125 million to 30 billion, we can do this easily. At last, we will analyze the in-domain and out-of-domain accuracy along with memory and execution time for all the models across the range of experiments designed above. We will compare our methods against end-to-end supervised finetuning and in-context learning as baselines.

Tentative steps are described as follows -

1. Devise methods to get in-context examples: We will explore using cosine similarity or SBERT similarity, as well as look into more sophisticated methods.
2. Pick a Language Model and Adapter: We will use one of the OPT variants with an adapter from Table 1. The choice will be based on types, usability, etc.
3. Finetune: We can either do vanilla finetuning [10] of pattern-based finetuning [16] based on experimentation results that we get.
4. Evaluate: We will be evaluating our models through the following metrics:
   - In-Distribution and Out-of-Distribution Accuracies
   - Execution Time
   - Memory
5. Experiment and Ablation Study: Observe the change in performance with:
   - Scale of LM
   - across different adapters and their variants
   - Baseline comparisons with SFT and ICL

3 Related Work

Supervised finetuning of LMs fine-tune a pretrained language model on a specific task using a randomized classification head on top. A variation of it is pattern-based finetuning [16] where there is a verbalizer [18] that maps LM output logits to desired answer, and rest of the finetuning is similar. In-context learning [2], on the other hand, is a method that doesn't update model's weights but provides examples similar to the desired task to instruct LMs how to respond to a given task.

We are exploring PEFT, which is a technique to improve LM performance by not modifying its original weights but adding additional elements and finetune those to allow the LM to adapt to these new tasks. There are multiple such methods that are presented and cited in Table 1.

4 Datasets

As defined before, we will focus on two tasks: Natural Language Inference (NLI) and Paraphrase Identification. For NLI, we can use the MNLI [23] and RTE [3] dataset. And for paraphrase identification, we can use the QQP [19] dataset. NLIV2 [22] is also a great dataset which contains many of such tasks with given in-context examples. So, this will remove our need for pre-processing and finding adequate examples and prompts.

We can also look at some generation tasks for our proposed methodology later on, based on timeline. Text summarization is one good example. We can use the NLIV2 [22] for that also, as it contains summarization examples from multiple other datasets.

All these datasets are publicly available through the website of the authors of the paper, or HuggingFace[2].

[2] https://huggingface.co/docs/datasets/en/index

5 Group Members

Neelabh Sinha, Snigdha Verma, Ananya Sharma

6 Ethical Implications

Ethical considerations accompany the initiative to fine-tune language models through parameter-efficient fine-tuning (PEFT). Efficient fine-tuning could lessen obstacles to unethical use, such as customizing language models to produce misleading, harmful, or biased content. Additionally, biases inherent in pre-trained language models could be exacerbated by fine-tuning, potentially resulting in discriminatory or unfair results in later applications. Privacy issues might emerge if sensitive data unintentionally becomes part of the fine-tuning datasets, risking exposure through model outputs. Although PEFT seeks to reduce resource consumption compared to traditional methods, even training smaller adapters demands considerable computational power, posing environmental and economic challenges. Furthermore, tasks like Natural Language Inference and Paraphrase Identification could spread misinformation if outputs aren't carefully scrutinized or if biases in datasets are perpetuated. Advances in PEFT may disproportionately benefit organizations with robust computational access, potentially widening gaps between well-resourced and less-resourced communities. To counteract these problems, we commit to conducting comprehensive audits of training datasets, incorporating fairness and robustness into our evaluations, documenting our models' limitations and intended purposes, and adhering to ethical standards to promote responsible development and use of our approaches.

References

[1] Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. Parameter-efficient fine-tuning design spaces. In *The Eleventh International Conference on Learning Representations*, 2023.

[2] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[3] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

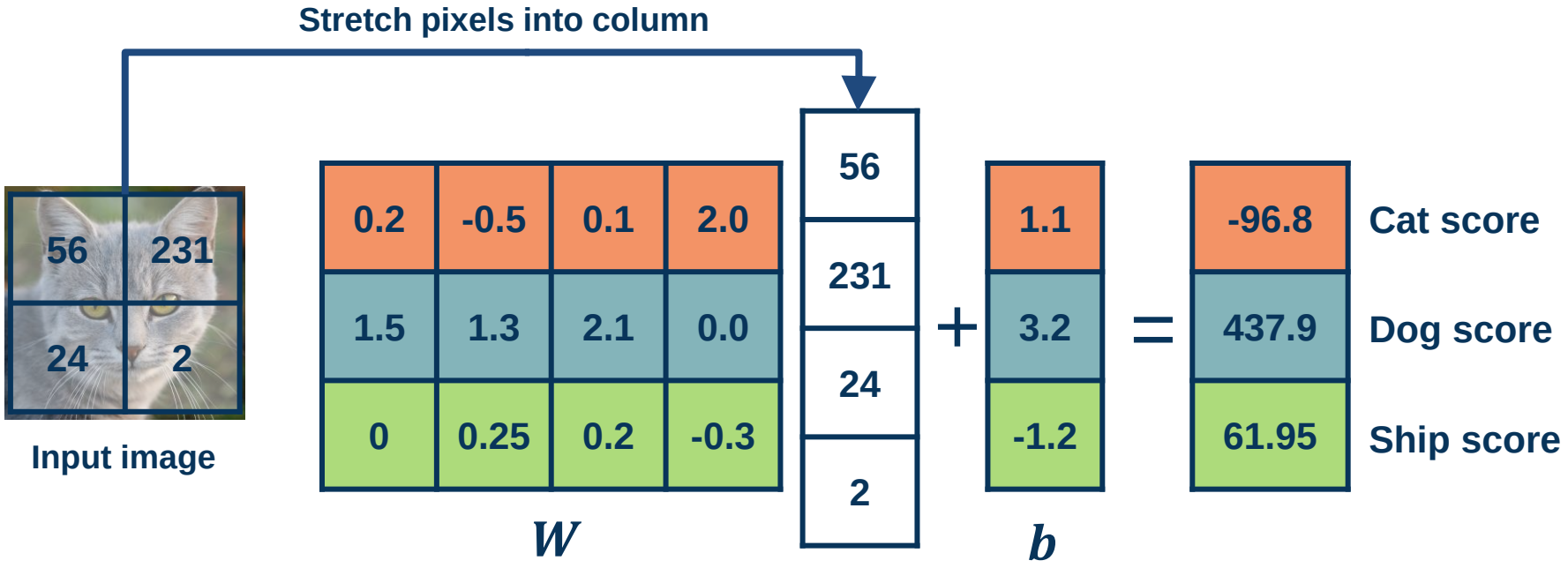
[4] Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J. Clark, and Mehdi Rezagholizadeh. Krona: Parameter efficient tuning with kronecker adapter, 2022.

[5] Cheng Fu, Hanxian Huang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. Learn-to-share: A hardware-friendly transfer learning framework exploiting computation and parameter sharing. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3469–3479. PMLR, 18–24 Jul 2021.

[6] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022.

[7] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning, 2022.

Example with an image with **4 pixels**, and **3 classes (cat/dog/ship)**



Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

- We can find the steepest descent direction by computing the **derivative (gradient):**

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

- Steepest descent direction is the **negative gradient**
- **Intuitively:** Measures how the function changes as the argument a changes by a small step size
  - As step size goes to zero
- **In Machine Learning:** Want to know how the **loss function** changes **as weights** are varied
  - Can consider each parameter separately by taking **partial derivative** of loss function with respect to that parameter

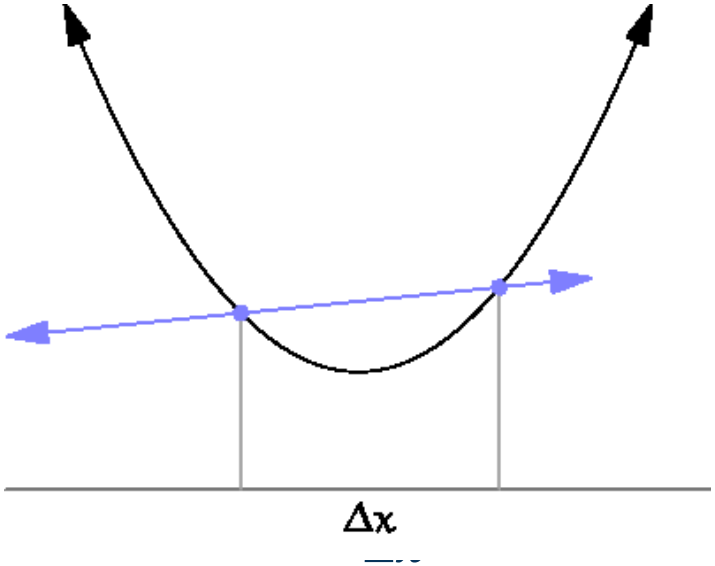
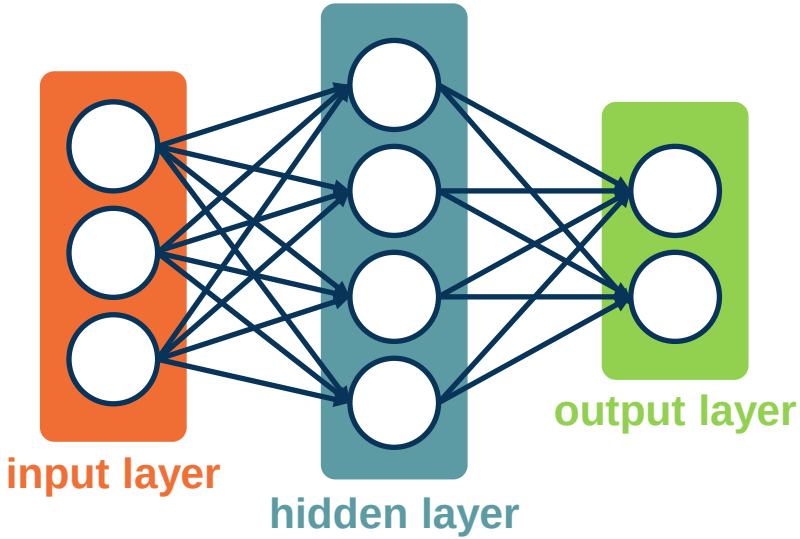


Image and equation from: https://en.wikipedia.org/wiki/Derivative#/media/File:Tangent_animation.gif

The same two-layered neural network **corresponds to adding another weight matrix**

- We will prefer the linear algebra view, but use some terminology from neural networks (& biology)



$$x \quad W_1 \quad W_2$$

$$=$$

$$f(x, W_1, W_2) = \sigma(W_2 \sigma(W_1 x))$$

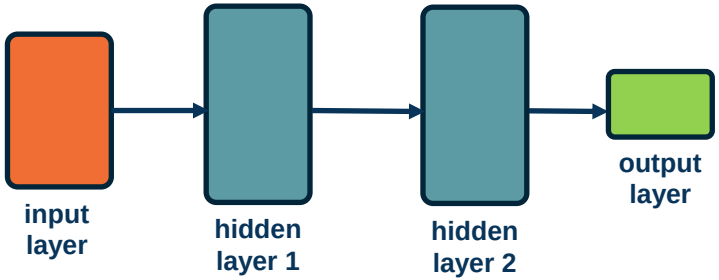
Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

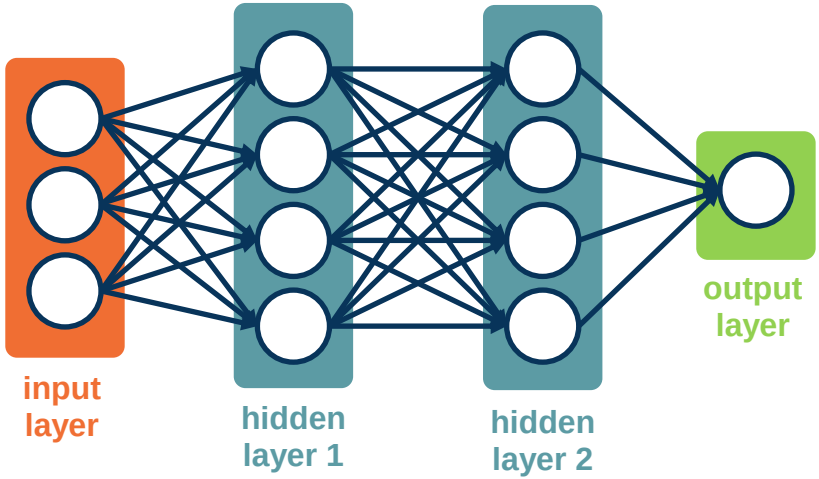
Large (deep) networks can be built by adding more and more layers

Three-layered neural networks can represent **any function**

- The number of nodes could grow unreasonably (exponential or worse) with respect to the complexity of the function

We will show them **without edges**:



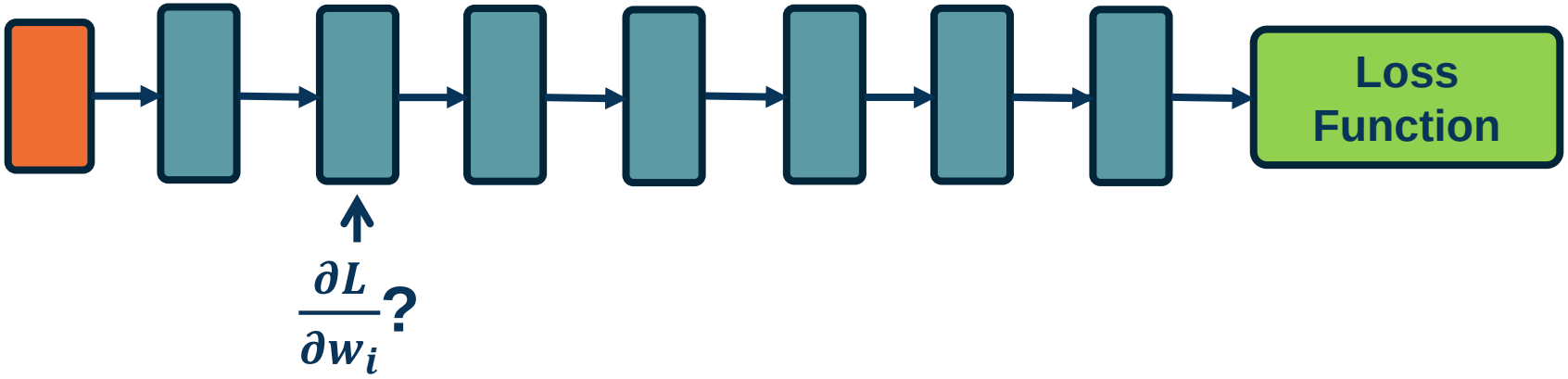


$$f(x, W_1, W_2, W_3) = \sigma(W_2\sigma(W_1 x))$$

Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Adding More Layers!

- We are learning **complex models** with significant amount of parameters (millions or billions)
- How do we compute the gradients of the **loss** (at the end) with respect to **internal** parameters?
- Intuitively, want to understand how **small changes** in weight deep inside **are propagated** to affect the **loss function** at the end

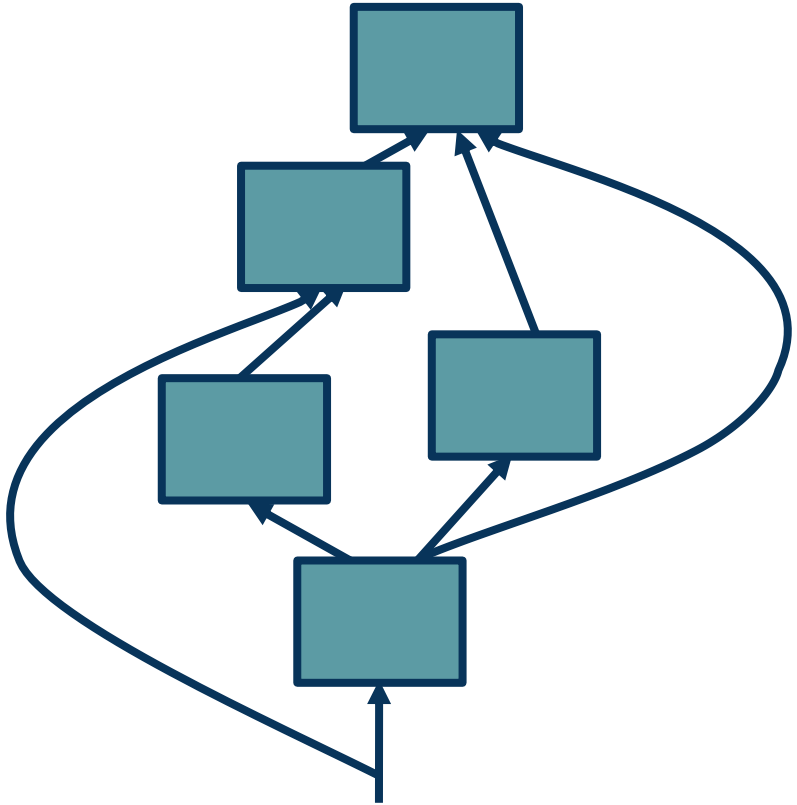


To develop a general algorithm for this, we will view the function as a **computation graph**

Graph can be any **directed acyclic graph (DAG)**

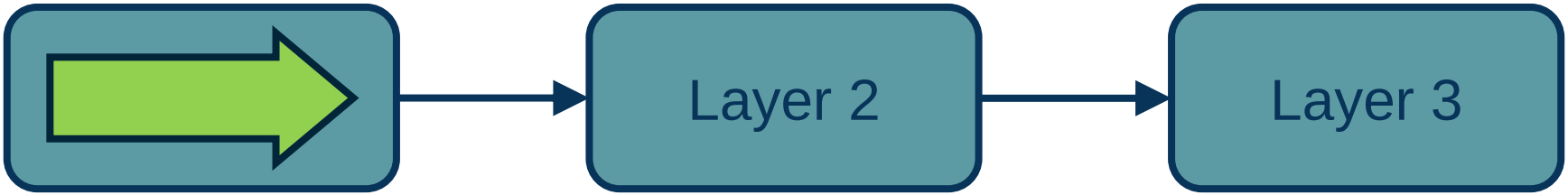
- Modules must be differentiable to support gradient computations for gradient descent

A **training algorithm** will then process this graph, **one module at a time**



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

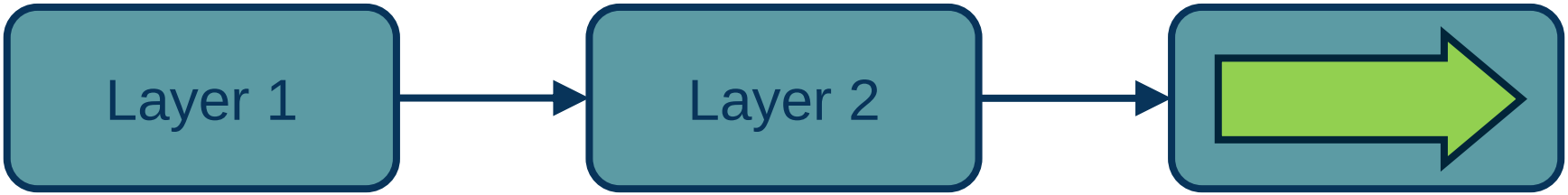


Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**



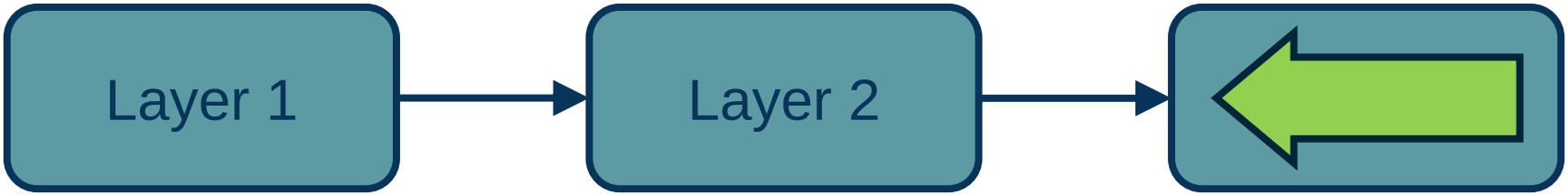
Note that we must store the **intermediate outputs of all layers**!

- This is because we will need them to **compute the gradients** (the gradient equations will have terms with the output values in them)

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

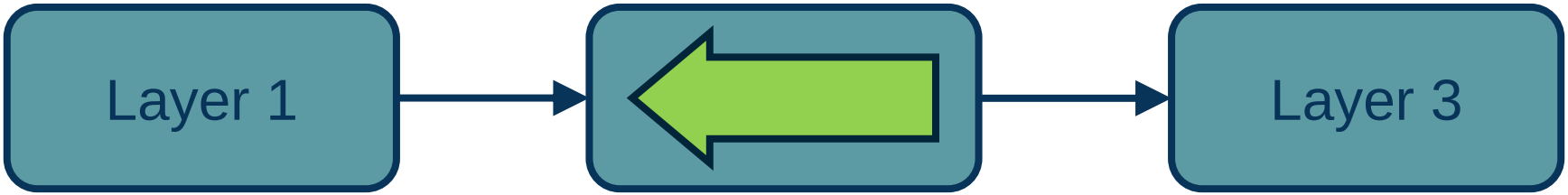
Step 2: Compute Gradients wrt parameters: **Backward Pass**



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

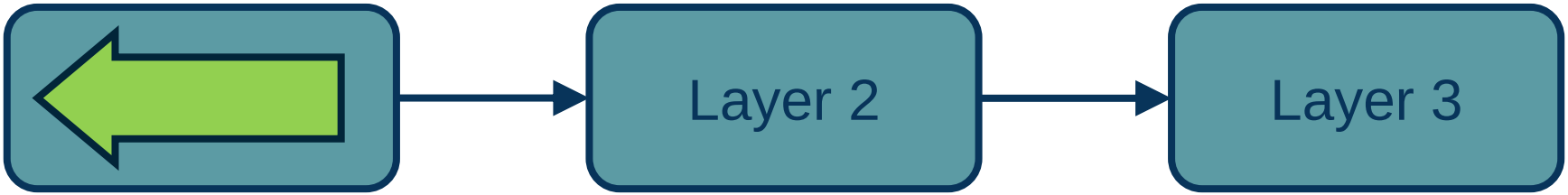
Step 2: Compute Gradients wrt parameters: **Backward Pass**



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

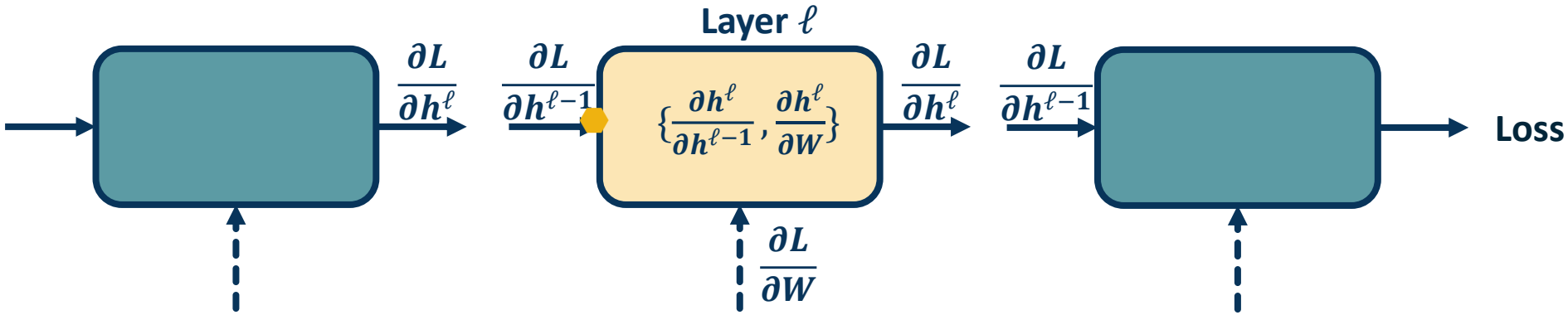
Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**



Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

- We want to compute: $\left\{\frac{\partial L}{\partial h^{\ell-1}}, \frac{\partial L}{\partial W}\right\}$



- We will use the *chain rule* to do this:

Chain Rule: $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$

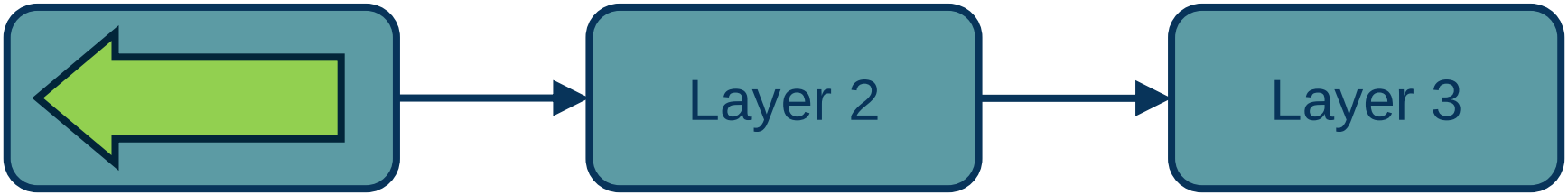
$$\frac{\partial L}{\partial h^{\ell-1}} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial h^{\ell-1}}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial W}$$

Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**

Step 3: Use **gradient** to update **all parameters** at the end



$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

Backpropagation is the application of gradient descent to a computation graph via the chain rule!

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun

Backpropagation: a simple example

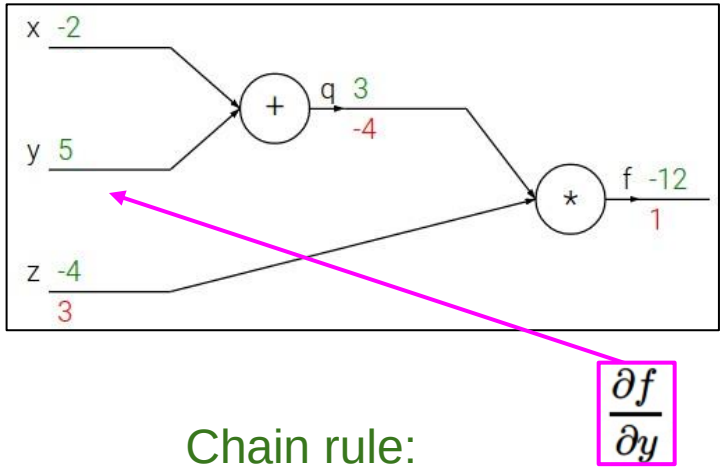
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

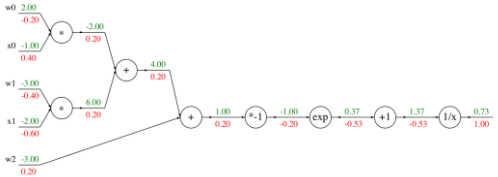
Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Deep Learning = Differentiable Programming

- Computation = Graph
  - Input = Data + Parameters
  - Output = Loss
  - Scheduling = Topological ordering

- What do we need to do?
  - Generic code for representing the graph of modules
  - Specify modules (both forward and backward function)
  - Backpropagation implementation on the graph

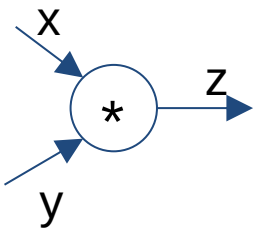
Modularized implementation: forward / backward API



Graph (or Net) object *(rough psuedo code)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Modularized implementation: forward / backward API



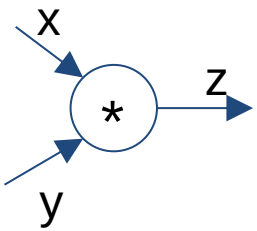
(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$\frac{\partial L}{\partial z}$

$\frac{\partial L}{\partial x}$

Modularized implementation: forward / backward API



(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Example: Caffe layers

Branch: master ▾ **caffe** / src / caffe / **layers** / Create new file | Upload files | Find file | History

shelhamer committed on GitHub Merge pull request #4630 from BlGene/load_hdf5_fix ... Latest commit e687a71 21 days ago

..		
absval_layer.cpp	dismantle layer headers	a year ago
absval_layer.cu	dismantle layer headers	a year ago
accuracy_layer.cpp	dismantle layer headers	a year ago
argmax_layer.cpp	dismantle layer headers	a year ago
base_conv_layer.cpp	enable dilated deconvolution	a year ago
base_data_layer.cpp	Using default from proto for prefetch	3 months ago
base_data_layer.cu	Switched multi-GPU to NCCL	3 months ago
batch_norm_layer.cpp	Add missing spaces besides equal signs in batch_norm_layer.cpp	4 months ago
batch_norm_layer.cu	dismantle layer headers	a year ago
batch_reindex_layer.cpp	dismantle layer headers	a year ago
batch_reindex_layer.cu	dismantle layer headers	a year ago
bias_layer.cpp	Remove incorrect cast of gemm int arg to Dtype in BiasLayer	a year ago
bias_layer.cu	Separation and generalization of ChannelwiseAffineLayer into BiasLayer	a year ago
bnll_layer.cpp	dismantle layer headers	a year ago
bnll_layer.cu	dismantle layer headers	a year ago
concat_layer.cpp	dismantle layer headers	a year ago
concat_layer.cu	dismantle layer headers	a year ago
contrastive_loss_layer.cpp	dismantle layer headers	a year ago
contrastive_loss_layer.cu	dismantle layer headers	a year ago
conv_layer.cpp	add support for 2D dilated convolution	a year ago
conv_layer.cu	dismantle layer headers	a year ago
crop_layer.cpp	remove redundant operations in Crop layer (#5138)	2 months ago
crop_layer.cu	remove redundant operations in Crop layer (#5138)	2 months ago
cudnn_conv_layer.cpp	dismantle layer headers	a year ago
cudnn_conv_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_lcn_layer.cpp	dismantle layer headers	a year ago
cudnn_lcn_layer.cu	dismantle layer headers	a year ago
cudnn_lrn_layer.cpp	dismantle layer headers	a year ago
cudnn_lrn_layer.cu	dismantle layer headers	a year ago
cudnn_pooling_layer.cpp	dismantle layer headers	a year ago
cudnn_pooling_layer.cu	dismantle layer headers	a year ago
cudnn_relu_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_relu_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_softmax_layer.cpp	dismantle layer headers	a year ago
cudnn_softmax_layer.cu	dismantle layer headers	a year ago
cudnn_tanh_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_tanh_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
data_layer.cpp	Switched multi-GPU to NCCL	3 months ago
deconv_layer.cpp	enable dilated deconvolution	a year ago
deconv_layer.cu	dismantle layer headers	a year ago
dropout_layer.cpp	supporting N-D Blobs in Dropout layer Reshape	a year ago
dropout_layer.cu	dismantle layer headers	a year ago
dummy_data_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cu	dismantle layer headers	a year ago
elu_layer.cpp	ELU layer with basic tests	a year ago
elu_layer.cu	ELU layer with basic tests	a year ago
embed_layer.cpp	dismantle layer headers	a year ago
embed_layer.cu	dismantle layer headers	a year ago
euclidean_loss_layer.cpp	dismantle layer headers	a year ago
euclidean_loss_layer.cu	dismantle layer headers	a year ago
exp_layer.cpp	Solving issue with exp layer with base e	a year ago
exp_layer.cu	dismantle layer headers	a year ago

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Caffe Sigmoid Layer

```cpp
#include <cmath>
#include <vector>

#include "caffe/layers/sigmoid_layer.hpp"

namespace caffe {

template <typename Dtype>
inline Dtype sigmoid(Dtype x) {
  return 1. / (1. + exp(-x));
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}

#ifdef CPU_ONLY
STUB_GPU(SigmoidLayer);
#endif

INSTANTIATE_CLASS(SigmoidLayer);


}  // namespace caffe
```


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$(1-\sigma(x))\,\sigma(x)$$ * top_diff (chain rule)

- Neural networks involves composing simple functions into a **computation graph**

- Optimization (updating weights) of this graph is through backpropagation
  - Recursive algorithm: Gradient descent (partial derivatives) plus chain rule

- Remaining questions:
  - How does this work with vectors, matrices, tensors?
    - Across a composed function?
  - How can we implement this algorithmically to make these calculations automatic? **Automatic Differentiation**

Linear Algebra View: Vector and Matrix Sizes

Georgia Tech

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b2 \\ w_{31} & w_{32} & \cdots & w_{3m} & b3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ 1 \end{bmatrix}$$

$$\boldsymbol{W} \qquad\qquad \boldsymbol{x}$$

Sizes: $[c \times (m+1)]$ $\quad$ $[(m+1) \times 1]$

Where c is number of classes

m is dimensionality of input

Conventions:

- Size of derivatives for scalars, vectors, and matrices:
  Assume we have scalar $\boldsymbol{s} \in \mathbb{R}^{\mathbf{1}}$, vector $\boldsymbol{v} \in \mathbb{R}^{\boldsymbol{m}}$, i.e. $\boldsymbol{v} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_m}]^{\boldsymbol{T}}$ and matrix $\boldsymbol{M} \in \mathbb{R}^{\boldsymbol{m_1} \times \boldsymbol{m_2}}$

	$\boldsymbol{S}$ [ ]	$\boldsymbol{V}$ [ ]	$\boldsymbol{M}$ [ ]
$\boldsymbol{S}$	$\frac{\partial s_1}{\partial s_2}$ [ ]	$\frac{\partial s}{\partial v}$ [ ]	$\frac{\partial s}{\partial M}$ [ ]
$\boldsymbol{V}$	$\frac{\partial v}{\partial s}$ [ ]	$\frac{\partial v_1}{\partial v_2}$ [ ]	
$\boldsymbol{M}$	$\frac{\partial M}{\partial s}$ [ ]		

Tensors

Conventions:

- Size of derivatives for scalars, vectors, and matrices:
  Assume we have scalar $\boldsymbol{s} \in \mathbb{R}^{\boldsymbol{1}}$, vector $\boldsymbol{v} \in \mathbb{R}^{\boldsymbol{m}}$, i.e. $\boldsymbol{v} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_m}]^{\boldsymbol{T}}$ and matrix $\boldsymbol{M} \in \mathbb{R}^{\boldsymbol{m_1 \times m_2}}$

- What is the size of $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{s}}$? $\mathbb{R}^{\boldsymbol{m \times 1}}$ (column vector of size $\boldsymbol{m}$)

$$\begin{bmatrix} \frac{\partial \boldsymbol{v_1}}{\partial \boldsymbol{s}} \\ \frac{\partial \boldsymbol{v_2}}{\partial \boldsymbol{s}} \\ \vdots \\ \frac{\partial \boldsymbol{v_m}}{\partial \boldsymbol{s}} \end{bmatrix}$$

- What is the size of $\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{v}}$? $\mathbb{R}^{\boldsymbol{1 \times m}}$ (row vector of size $\boldsymbol{m}$)

$$\begin{bmatrix} \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{v_1}} & \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{v_1}} & \cdots & \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{v_m}} \end{bmatrix}$$

Conventions:

- What is the size of $\frac{\partial v^1}{\partial v^2}$? A matrix:

Col j

Row i

$$\begin{bmatrix} \frac{\partial v_1^1}{\partial v_1^2} & \cdots & \cdots & \cdots & \cdots & \\ \cdots & & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial v_i^1}{\partial v_1^2} & \cdots & \frac{\partial v_i^1}{\partial v_j^2} & \cdots & \frac{\partial v_i^1}{\partial v_{m_2}^2} & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \end{bmatrix}$$

$m_1 \times m_2$

- This matrix of partial derivatives is called a **Jacobian**

(Note this is slightly different convention than on Wikipedia). Also, computationally other conventions are used.

Conventions:

- What is the size of $\frac{\partial s}{\partial M}$? A matrix:

$$\begin{bmatrix} \frac{\partial s}{\partial m_{[1,1]}} & \cdots & \cdots & \cdots & \cdots & \\ \cdots & & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \frac{\partial s}{\partial m_{[i,j]}} & \cdots & \cdots & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \end{bmatrix}$$

(Note this is slightly different convention than on Wikipedia). Also, computationally other conventions are used.

Example 1:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x \\ x^2 \end{bmatrix} \qquad \frac{\partial y}{\partial x} = \begin{bmatrix} 1 \\ 2x \end{bmatrix}$$

Example 2:

$$y = w^T x = \sum_k w_k x_k$$

$$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x_1}, \ldots, \frac{\partial y}{\partial x_m} \right]$$

$$= [w_1, \ldots, w_m] \quad \text{because} \quad \frac{\partial (\sum_k w_k x_k)}{\partial x_i} = w_i$$

$$= w^T$$

Example 3:

$$y = Wx \qquad \frac{\partial y}{\partial x} = W$$

Col j

Row i

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \cdots & \cdots & \cdots \\ \cdots & & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \frac{\partial y_i}{\partial x_j} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & w_{ij} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

$$y_i = \sum_j w_{ij} x_j$$

Example 4:

$\frac{\partial (wAw)}{\partial w} = 2w^T A$ (assuming A is symmetric)

- What is the size of $\frac{\partial L}{\partial W}$?
  - Remember that loss is a **scalar** and W is a matrix:

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b2 \\ w_{31} & w_{32} & \cdots & w_{3m} & b3 \end{bmatrix}$$

Jacobian is also a matrix:

$$\begin{matrix} & & & & W & \\ \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \cdots & \frac{\partial L}{\partial w_{1m}} & \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial w_{21}} & \cdots & \cdots & \frac{\partial L}{\partial w_{2m}} & \frac{\partial L}{\partial b_2} \\ \cdots & \cdots & \cdots & \frac{\partial L}{\partial w_{3m}} & \frac{\partial L}{\partial b_3} \end{bmatrix} \end{matrix}$$

Batches of data are **matrices** or **tensors** (multi-dimensional matrices)

Examples:

- Each instance is a vector of size $\boldsymbol{m}$, our batch is of size $[\boldsymbol{B} \times \boldsymbol{m}]$
- Each instance is a matrix (e.g. grayscale image) of size $\boldsymbol{W} \times \boldsymbol{H}$, our batch is $[\boldsymbol{B} \times \boldsymbol{W} \times \boldsymbol{H}]$
- Each instance is a multi-channel matrix (e.g. color image with R,B,G channels) of size $\boldsymbol{C} \times \boldsymbol{W} \times \boldsymbol{H}$, our batch is $[\boldsymbol{B} \times \boldsymbol{C} \times \boldsymbol{W} \times \boldsymbol{H}]$

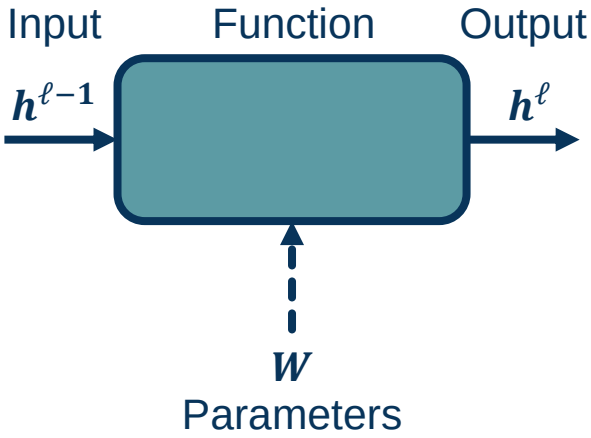
Jacobians become tensors which is complicated

- Instead, flatten input to a vector and get a vector of derivatives!
- This can also be done for partial derivatives between two vectors, two matrices, or two tensors

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix}$$

Flatten

$$\begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{21} \\ x_{22} \\ \vdots \\ x_{n1} \\ \vdots \\ x_{nn} \end{bmatrix}$$



Define:

$$h_i^{\ell} = w_i^T h^{\ell-1}$$

$$h^{\ell} = W h^{\ell-1}$$

$$|h^{\ell}| \times 1 \quad |h^{\ell}| \times |h^{\ell-1}| \quad |h^{\ell-1}| \times 1$$

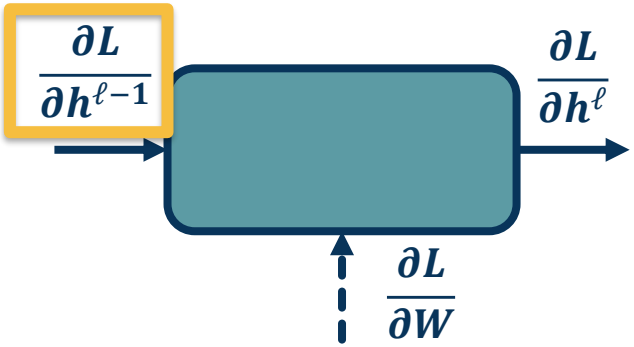
Fully Connected (FC) Layer: Forward Function

$$h^{\ell} = W h^{\ell-1}$$

$$\frac{\partial h^{\ell}}{\partial h^{\ell-1}} = W$$

Define:

$$h_i^{\ell} = w_i^T h^{\ell-1}$$



$$\frac{\partial L}{\partial h^{\ell-1}} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial h^{\ell-1}}$$

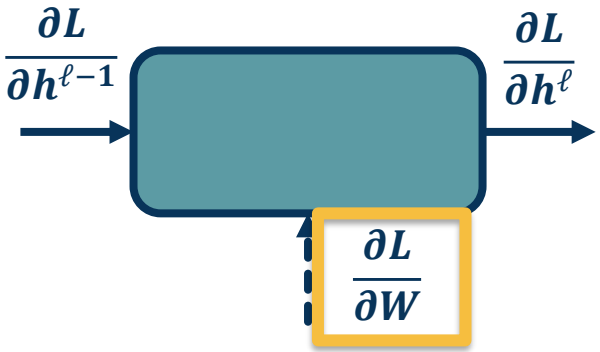
$$1 \times |h^{\ell-1}| \quad 1 \times |h^{\ell}| \quad |h^{\ell}| \times |h^{\ell-1}|$$

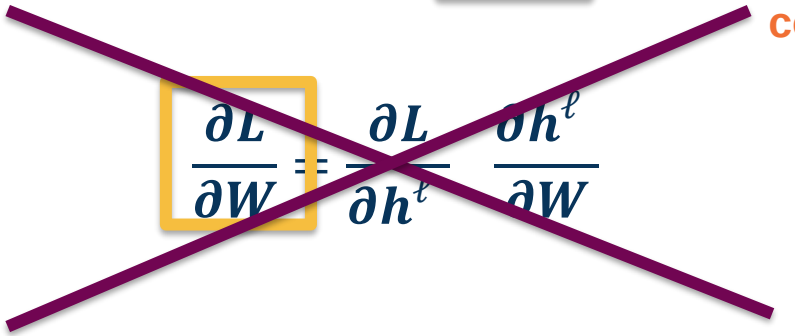
$$h^{\ell} = W h^{\ell-1}$$

$$\frac{\partial h^{\ell}}{\partial h^{\ell-1}} = W$$

Define:

$$h_i^{\ell} = w_i^T h^{\ell-1}$$





Note doing this on full W matrix would result in Jacobian tensor!

But it is *sparse* – each output only affected by corresponding weight row

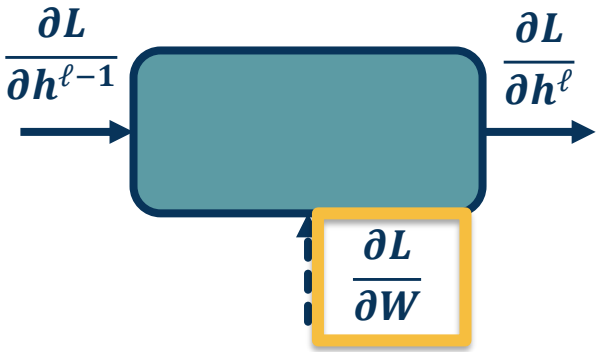
$$h^{\ell} = W h^{\ell-1}$$

$$\frac{\partial h^{\ell}}{\partial h^{\ell-1}} = W$$

Define:

$$h_i^{\ell} = w_i^T h^{\ell-1}$$

$$\frac{\partial h_i^{\ell}}{\partial w_i^T} = h^{(\ell-1),T}$$



Note doing this on full W matrix would result in Jacobian tensor!

But it is *sparse* – each output only affected by corresponding weight row

$$\frac{\partial L}{\partial w_i^T} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial w_i^T}$$

$$\begin{bmatrix} 0 \\ \frac{\partial h_i^{\ell}}{\partial w_i^T} \\ 0 \end{bmatrix}$$

$$\frac{\partial L}{\partial W}$$

$1 \times |h^{\ell-1}|$ $1 \times |h^{\ell}|$ $|h^{\ell}| \times |h^{\ell-1}|$

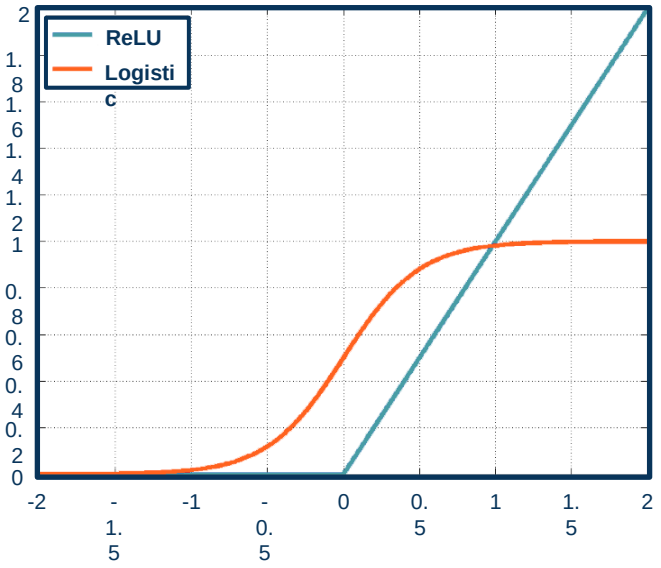
Iterate and populate
Note can simplify/vectorize!

We can employ **any differentiable (or piecewise differentiable) function**

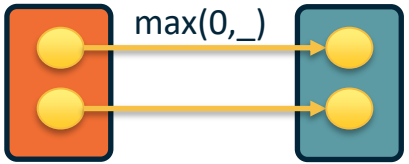
A common choice is the **Rectified Linear Unit**

- Provides non-linearity but better gradient flow than sigmoid
- Performed **element-wise**

How many parameters for this layer?



$$h^{\ell} = \max\left(0, h^{\ell-1}\right)$$

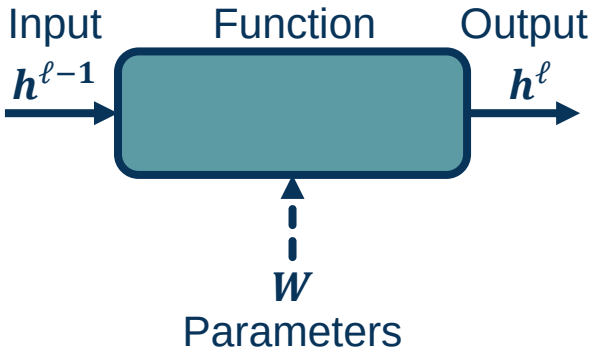


Full Jacobian of ReLU layer is **large** (output dim x input dim)

- But again it is **sparse**
- Only **diagonal values non-zero** because it is element-wise
- An output value affected only by **corresponding input value**

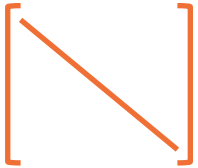
Max function **funnels gradients through selected max**

- Gradient will be **zero** if input **<= 0**



Forward: $h^{\ell} = \max(0, h^{\ell-1})$

Backward: $\frac{\partial L}{\partial h^{\ell-1}} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial h^{\ell-1}}$



$|h^{\ell} \times h^{\ell-1}|$

For diagonal

$$\frac{\partial h^{\ell}}{\partial h^{\ell-1}} = \begin{cases} 1 & if\ h^{\ell-1} > 0 \\ 0 & otherwise \end{cases}$$

Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

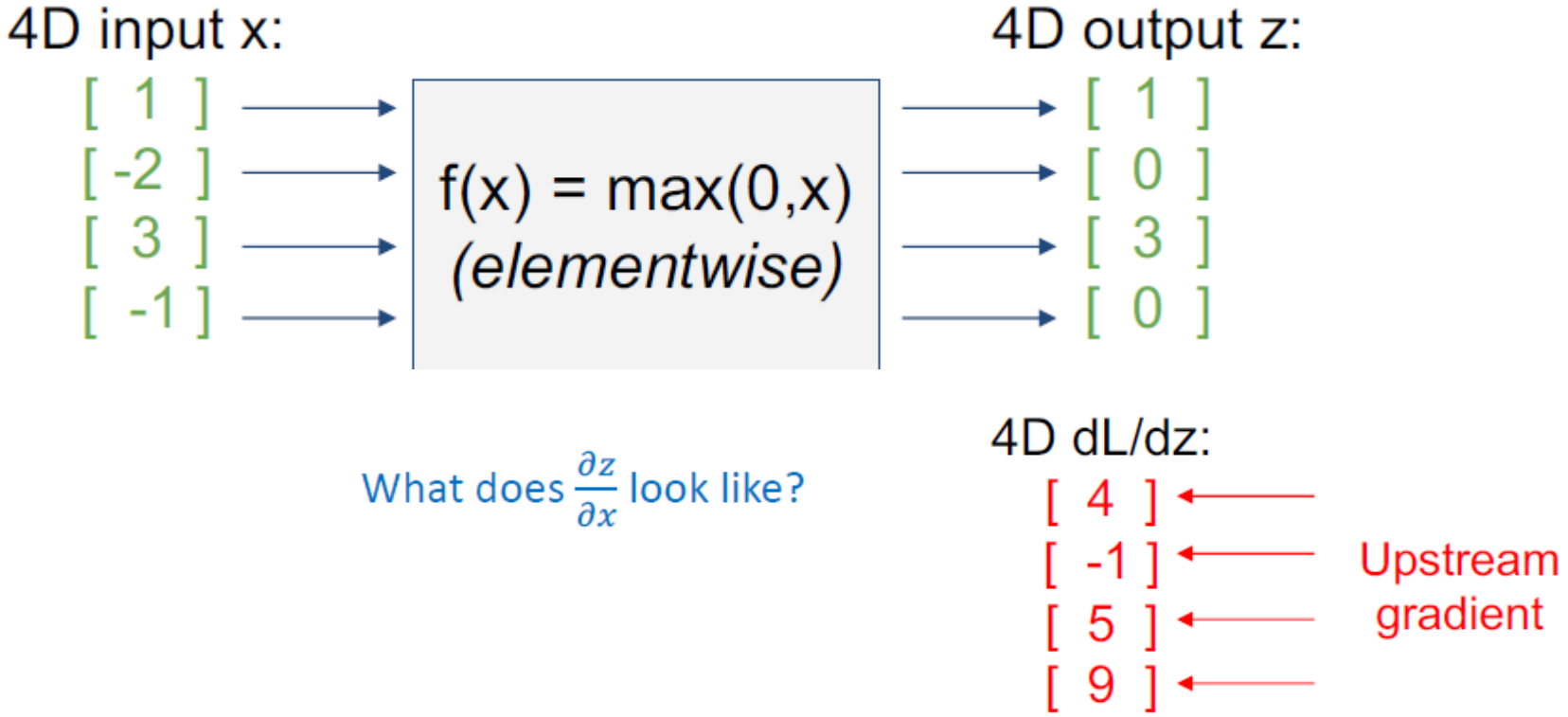
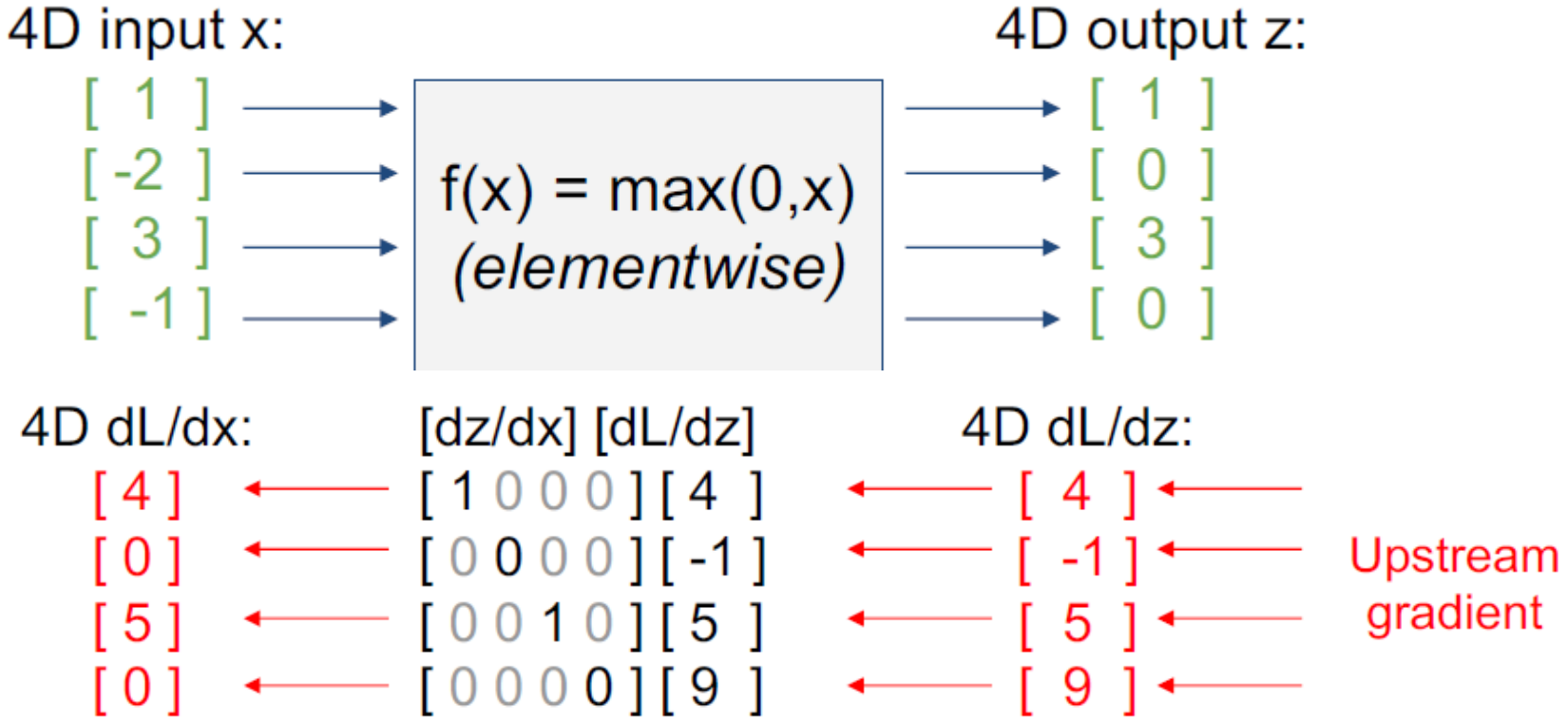


Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



For element-wise ops, jacobian is **sparse**: off-diagonal entries always zero!
Never **explicitly** form Jacobian -- instead use elementwise multiplication

- Neural networks involves composing simple functions into a **computation graph**

- Optimization (updating weights) of this graph is through backpropagation
  - Recursive algorithm: Gradient descent (partial derivatives) plus chain rule

- Remaining questions:
  - How does this work with vectors, matrices, tensors?
    - Across a composed function? **Next!**
  - How can we implement this algorithmically to make these calculations automatic? **Automatic Differentiation**

Summary

Composition of Functions: $f(g(x)) = (f \circ g)(x)$

A complex function (e.g. defined by a neural network):

$$f(x) = g_\ell\left(g_{\ell-1}(\dots g_1(x))\right)$$

$$f(x) = g_\ell \circ g_{\ell-1} \dots \circ g_1(x)$$

(Many of these will be parameterized)

(Note you might find the opposite notation as well!)

$$\mathbf{x} \in \mathbb{R}^1 \xrightarrow[g_1]{} z \in \mathbb{R}^1 \xrightarrow[g_2]{} y \in \mathbb{R}^1$$

$$y = g_2\big(g_1(x)\big)$$

$$= \frac{\partial y}{\partial z} * \frac{\partial z}{\partial x}$$

Scalar Multiplication

Scalar Case

Georgia Tech

$$\begin{bmatrix} \frac{\partial \vec{y}}{\partial \vec{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \vec{y}}{\partial \vec{z}} \end{bmatrix} \begin{bmatrix} \frac{\partial \vec{z}}{\partial \vec{x}} \end{bmatrix}$$

$J_{g_1 \circ g_2}$ J_{g_1} J_{g_2}

Matrix Multiplication

Vector Case

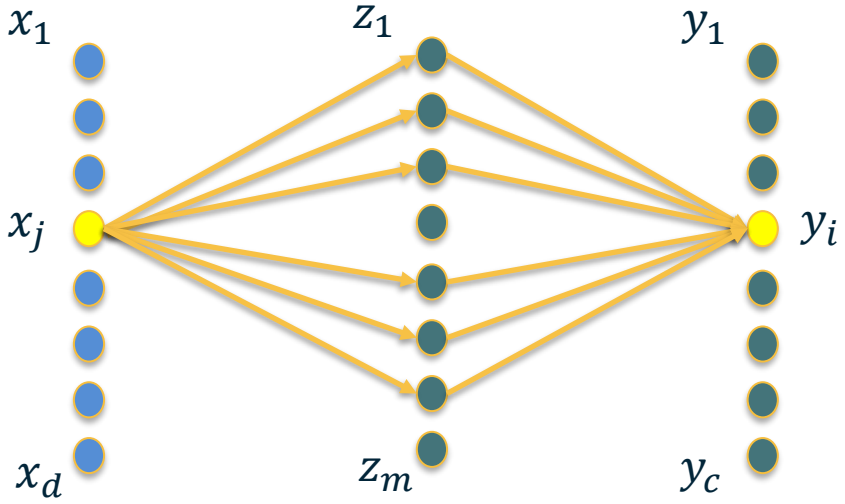
Georgia Tech

$$\begin{bmatrix} \frac{\partial y_i}{\partial x_j} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_i}{\partial z_k} \end{bmatrix} \begin{bmatrix} \frac{\partial z_k}{\partial x_j} \end{bmatrix}$$

$$\frac{\partial y_i}{\partial x_j} = \sum_k \frac{\partial y_i}{\partial z_k} * \frac{\partial z_k}{\partial x_j}$$

Jacobian View of Chain Rule

Georgia Tech



$$\frac{\partial y_i}{\partial x_j} = \sum_k \frac{\partial y_i}{\partial z_k} * \frac{\partial z_k}{\partial x_j}$$

k paths

Graphical View of Chain Rule

Georgia Tech

$$h^0 \in \mathbb{R}^d \longrightarrow \mathrm{h}^1 \in \mathbb{R}^{\boldsymbol{d}} \longrightarrow \quad \ldots \quad \longrightarrow \mathrm{h}^l \in \mathbb{R}^{\boldsymbol{d}}$$

$$\frac{\partial h^l}{\partial h^1} = \frac{\partial h^l}{\partial h^{l-1}} \frac{\partial h^{l-1}}{\partial h^{l-2}} \ldots \frac{\partial h^2}{\partial h^1}$$

$$\begin{bmatrix} \quad \end{bmatrix} = \begin{bmatrix} \quad \end{bmatrix} \begin{bmatrix} \quad \end{bmatrix} \begin{bmatrix} \quad \end{bmatrix}$$

Chain Rule: Cascaded

Georgia Tech

$$h^0 \in \mathbb{R}^d \longrightarrow h^1 \in \mathbb{R}^d \longrightarrow \cdots \longrightarrow h^l \in \mathbb{R}^d \longrightarrow L \in \mathbb{R}^1$$

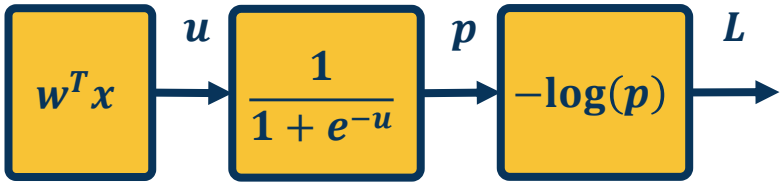
$$\frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^l} \frac{\partial h^l}{\partial h^{l-1}} \frac{\partial h^{l-1}}{\partial h^{l-2}} \cdots \frac{\partial h^2}{\partial h^1}$$

$$\begin{bmatrix} \end{bmatrix} = \begin{bmatrix} \end{bmatrix} \begin{bmatrix} \end{bmatrix} \begin{bmatrix} \end{bmatrix} \begin{bmatrix} \end{bmatrix}$$

Which directions is more efficient to multiply?

Chain Rule: Cascaded

Georgia Tech



$$\bar{L} = 1$$

$$\bar{p} = \frac{\partial L}{\partial p} = -\frac{1}{p}$$

where $p = \sigma(w^T x)$ and $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\bar{u} = \frac{\partial L}{\partial u} = \frac{\partial L}{\partial p}\ \frac{\partial p}{\partial u} = \bar{p}\ \sigma(1-\sigma)$$

$$\bar{w} = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial u}\ \frac{\partial u}{\partial w} = \bar{u}x^T$$

We can do this in a combined way to see all terms together:

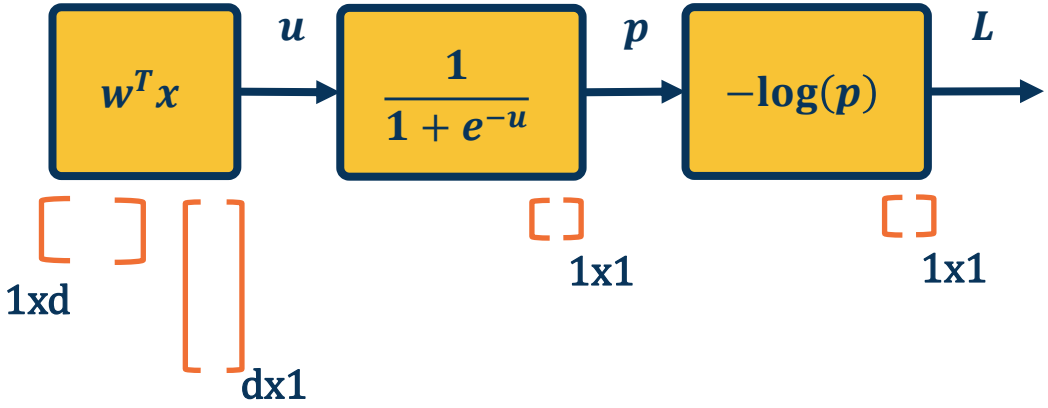
$$\bar{w} = \frac{\partial L}{\partial p}\ \frac{\partial p}{\partial u}\ \frac{\partial u}{\partial w} = -\frac{1}{\sigma(w^T x)}\sigma(w^T x)(1-\sigma\ (w^T x))x^T$$
$$= -\left(1-\sigma(w^T x)\right)x^T$$

This effectively shows gradient flow along path from L to w

Example Gradient Computations

Georgia Tech

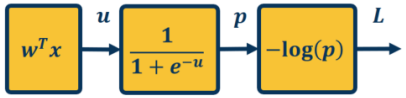
The chain rule can be computed as a **series of scalar, vector, and matrix linear algebra operations**



Extremely efficient in graphics processing units (GPUs)

$$\bar{w} = -\frac{1}{\sigma(w^T x)}\sigma(w^T x)(1-\sigma(w^T x))x^T$$

1x1 1x1 1x1 1xd



$L = 1$

$\bar{p} = \frac{\partial L}{\partial p} = -\frac{1}{p}$

where $p = \sigma(w^T x)$ and $\sigma(x) = \frac{1}{1+e^{-x}}$

$\bar{u} = \frac{\partial L}{\partial u} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial u} = \bar{p}\,\sigma(1-\sigma)$

$\bar{w} = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial u}\frac{\partial u}{\partial w} = \bar{u}x^T$

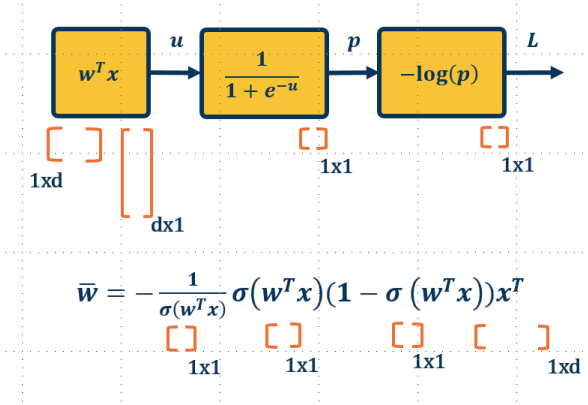
We can do this in a combined way to see all terms together:

$\bar{w} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial u}\frac{\partial u}{\partial w} = -\frac{1}{\sigma(w^T x)}\sigma(w^T x)(1-\sigma(w^T x))x^T$

$= -\left(1-\sigma(w^T x)\right)x^T$

This effectively shows gradient flow along path from L to w

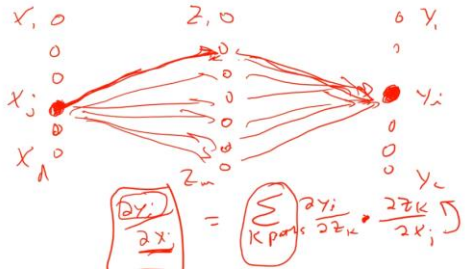
Computation Graph / Global View of Chain Rule



$$\bar{w} = -\frac{1}{\sigma(w^T x)}\sigma(w^T x)(1-\sigma(w^T x))x^T$$

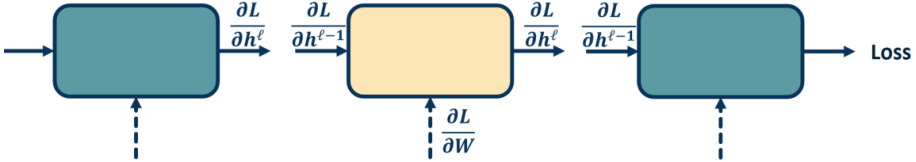
1x1 1x1 1x1 1xd

Computational / Tensor View



Graph View

- We want to to compute: $\left\{\frac{\partial L}{\partial h^{\ell-1}}, \frac{\partial L}{\partial W}\right\}$



Backpropagation View (Recursive Algorithm)

Different Views of Equivalent Ideas

Georgia Tech

- **Backpropagation:** Recursive, modular algorithm for chain rule + gradient descent

- **When we move to vectors and matrices:**
  - Composition of functions (scalar)
  - Composition of functions (vectors/matrices)
  - Jacobian view of chain rule
  - Can view entire set of calculations as linear algebra operations (matrix-vector or matrix-matrix multiplication)

- **Automatic differentiation**:
  - Reduction of modules to simple operations we know (simple multiplication, etc.)
  - Automatically build computation graph in background as write code
  - Automatically compute gradients via backward pass

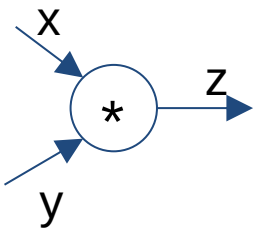
Automatic Differentiation

Deep Learning = Differentiable Programming

- Computation = Graph
  - Input = Data + Parameters
  - Output = Loss
  - Scheduling = Topological ordering

- What do we need to do?
  - Generic code for representing the graph of modules
  - Specify modules (both forward and backward function)

Modularized implementation: forward / backward API



(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$\frac{\partial L}{\partial z}$

$\frac{\partial L}{\partial x}$

Backpropagation does not really spell out how to **efficiently** carry out the necessary computations

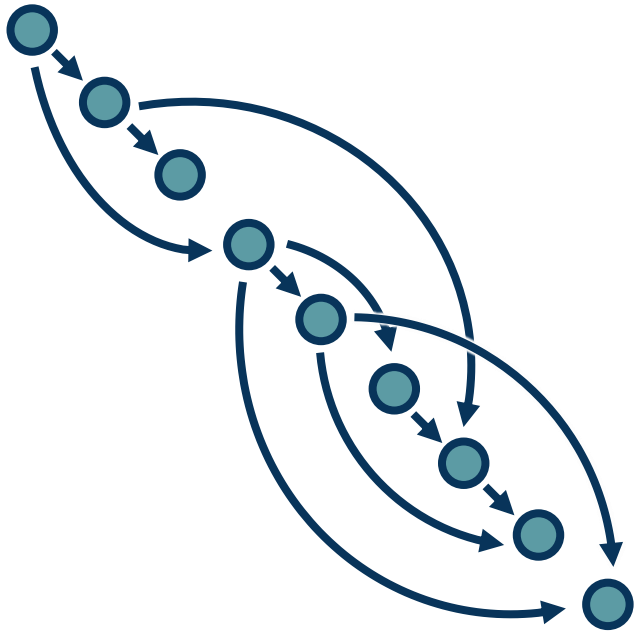
But the idea can be applied to **any directed acyclic graph (DAG)**

- Graph represents an **ordering constraining** which paths must be calculated first

Given an ordering, we can then iterate from the last module backwards, **applying the chain rule**

- We will store, for each node, its **local gradient function/computation for efficiency**
- We will do this **automatically** by computing backwards function for primitives and as you write code, express the function with them

This is called reverse-mode **automatic differentiation**



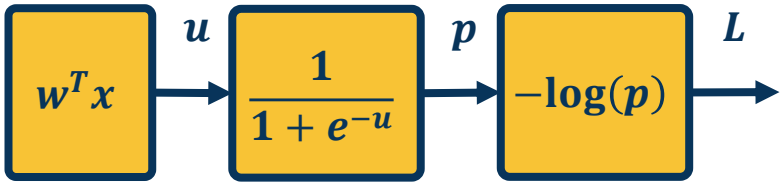
A General Framework

Computation = Graph

- Input = Data + Parameters
- Output = Loss
- Scheduling = Topological ordering

Auto-Diff

- A family of algorithms for implementing chain-rule on computation graphs



Automatic differentiation:

- Carries out this procedure for us on arbitrary graphs
- Knows derivatives of primitive functions
- As a result, we just define these (forward) functions **and don't even need to specify the gradient (backward) functions!**

$$\bar{L} = 1$$

$$\bar{p} = \frac{\partial L}{\partial p} = -\frac{1}{p}$$

where $p = \sigma(w^T x)$ and $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\bar{u} = \frac{\partial L}{\partial u} = \frac{\partial L}{\partial p}\ \frac{\partial p}{\partial u} = \bar{p}\ \sigma(1-\sigma)$$

$$\bar{w} = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial u}\ \frac{\partial u}{\partial w} = \bar{u} x^T$$

We can do this in a combined way to see all terms together:

$$\bar{w} = \frac{\partial L}{\partial p}\ \frac{\partial p}{\partial u}\ \frac{\partial u}{\partial w} = -\frac{1}{\sigma(w^T x)}\sigma(w^T x)(1-\sigma(w^T x))x^T$$
$$= -\left(1-\sigma(w^T x)\right)x^T$$

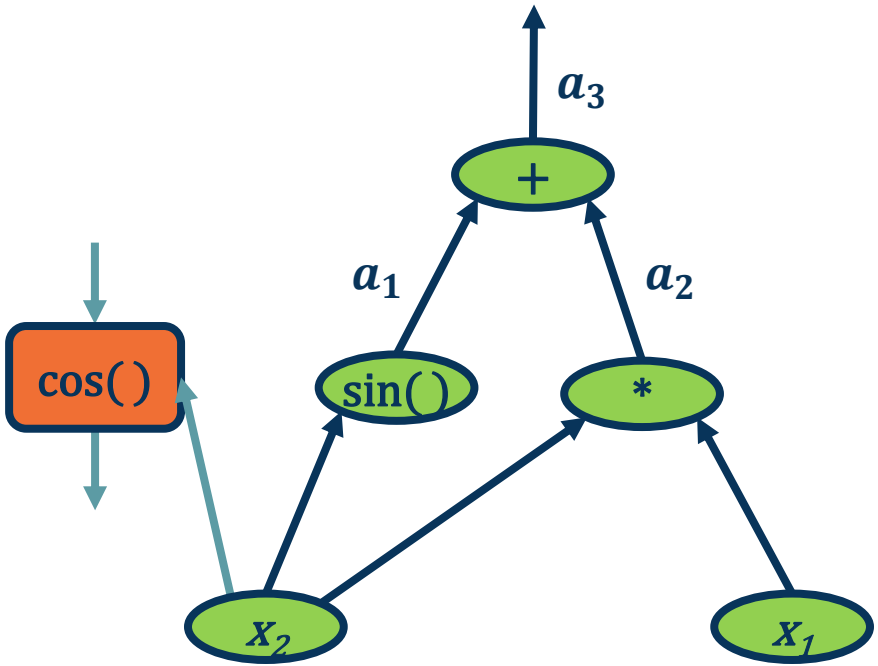
This effectively shows gradient flow along path from L to w

Example Gradient Computations

- Key idea is to **explicitly store computation graph** in memory and **corresponding gradient functions**

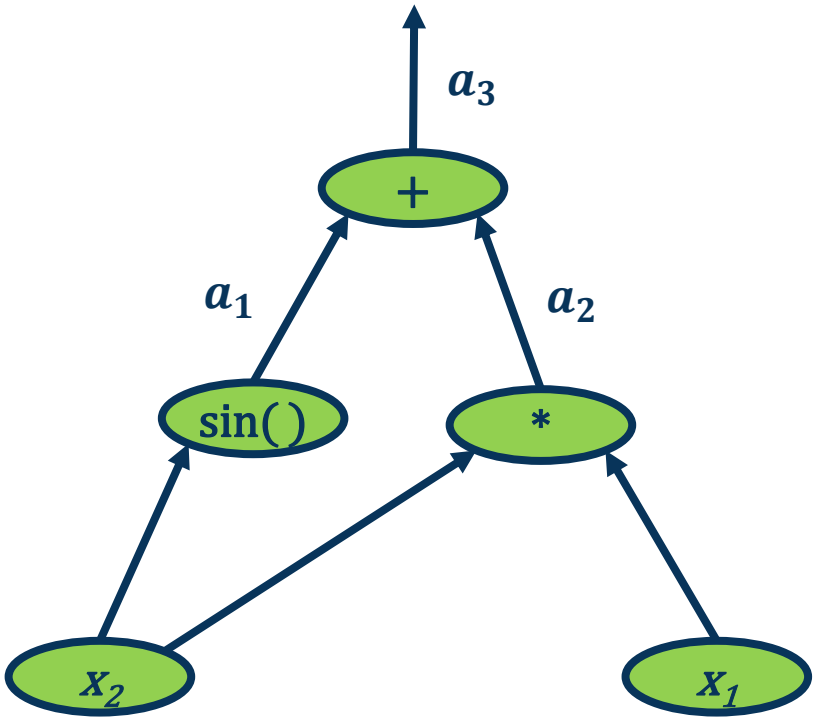
- **Nodes** broken down to **basic primitive computations** (addition, multiplication, log, etc.) for which **corresponding derivative is known**

$$\overline{x_2} = \frac{\partial f}{\partial a_1}\frac{\partial a_1}{\partial x_2} = \overline{a_1}\ \cos(x_2)$$



Example

$$f(x_1, x_2) = x_1 x_2 + \sin(x_2)$$



We want to find the **partial derivative of output f** (output) with respect to **all intermediate variables**

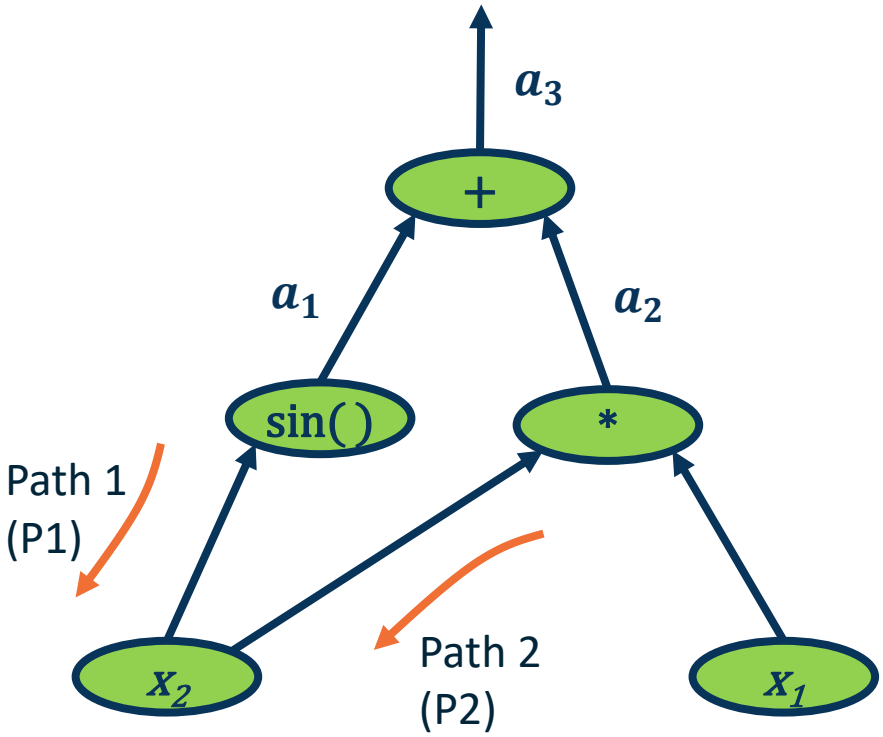
- Assign intermediate variables

  Simplify notation:

  Denote bar as: $\overline{a_3} = \frac{\partial f}{\partial a_3}$

- Start at **end** and move **backward**

$$f(x_1, x_2) = x_1 x_2 + \sin(x_2)$$



$$\overline{a_3} = \frac{\partial f}{\partial a_3} = 1$$

$$\overline{a_1} = \frac{\partial f}{\partial a_1} = \frac{\partial f}{\partial a_3}\frac{\partial a_3}{\partial a_1} = \frac{\partial f}{\partial a_3}\frac{\partial (a_1+a_2)}{\partial a_1} = \frac{\partial f}{\partial a_3}\,1 = \overline{a_3}$$

$$\overline{a_2} = \frac{\partial f}{\partial a_2} = \frac{\partial f}{\partial a_3}\frac{\partial a_3}{\partial a_2} = \overline{a_3}$$

$$\overline{x_2^{P1}} = \frac{\partial f}{\partial a_1}\frac{\partial a_1}{\partial x_2} = \overline{a_1}\,\cos(x_2)$$

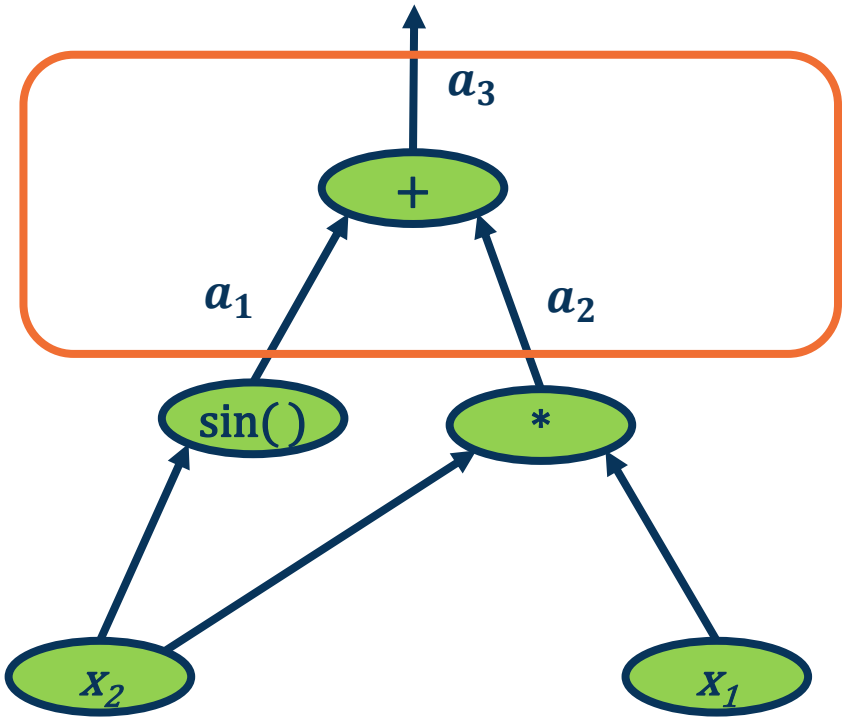
$$\overline{x_2^{P2}} = \frac{\partial f}{\partial a_2}\frac{\partial a_2}{\partial x_2} = \frac{\partial f}{\partial a_2}\frac{\partial (x1x2)}{\partial x_2} = \overline{a_2} x_1$$

Gradients from multiple paths **summed**

$$\overline{x_1} = \frac{\partial f}{\partial a_2}\frac{\partial a_2}{\partial x_1} = \overline{a_2} x_2$$

Example

$$f(x_1, x_2) = x_1 x_2 + \sin(x_2)$$



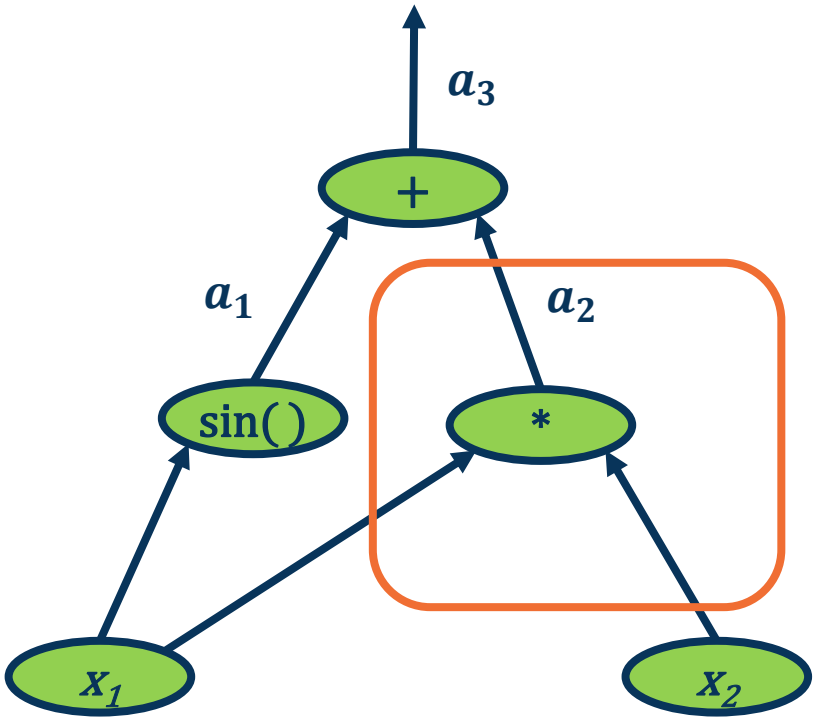
$$\overline{a_1} = \frac{\partial f}{\partial a_1} = \frac{\partial f}{\partial a_3}\frac{\partial a_3}{\partial a_1} = \frac{\partial f}{\partial a_3}\frac{\partial (a_1 + a_2)}{\partial a_1} = \frac{\partial f}{\partial a_3}\,1 = \overline{a_3}$$

$$\overline{a_2} = \frac{\partial f}{\partial a_2} = \frac{\partial f}{\partial a_3}\frac{\partial a_3}{\partial a_2} = \overline{a_3}$$

Addition operation distributes gradients along all paths!

Patterns of Gradient Flow: Addition

$$f(x_1, x_2) = x_1 x_2 + \sin(x_2)$$



Multiplication operation is a gradient switcher (multiplies it by the values of the other term)

$$\overline{x_2} = \frac{\partial f}{\partial a_2}\frac{\partial a_2}{\partial x_2} = \frac{\partial f}{\partial a_2}\frac{\partial (x1x2)}{\partial x_2} = \overline{a_2} x_1$$

$$\overline{x_1} = \frac{\partial f}{\partial a_2}\frac{\partial a_2}{\partial x_1} = \overline{a_2} x_2$$

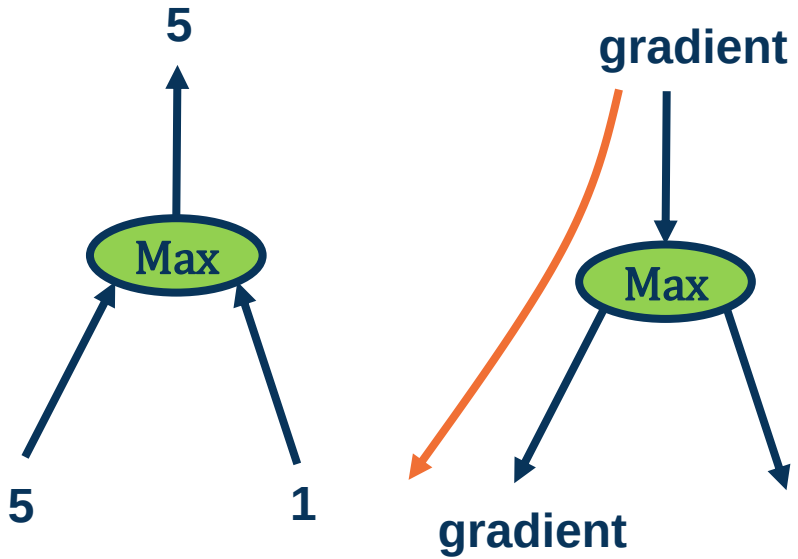
Patterns of Gradient Flow: Multiplication

Georgia Tech

Several other patterns as well, e.g.:

Max operation **selects** which path to push the gradients through

- Gradient flows along the path that was “selected” to be max
- This information must be recorded in the forward pass



The flow of gradients is one of the **most important aspects** in deep neural networks

- If gradients **do not flow backwards properly,** learning slows or stops!

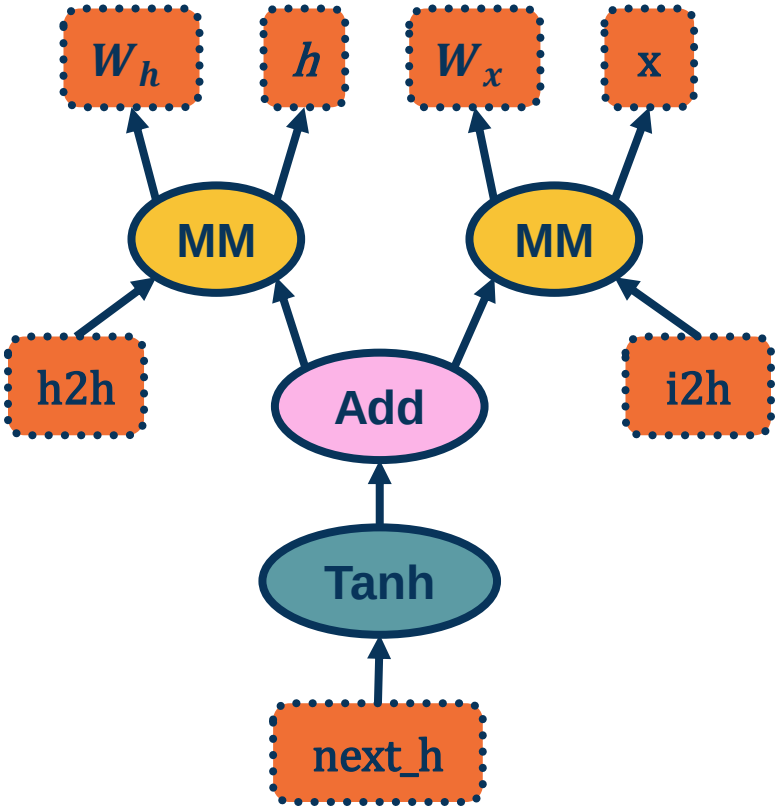
A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 20))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 20))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
```

(Note above)

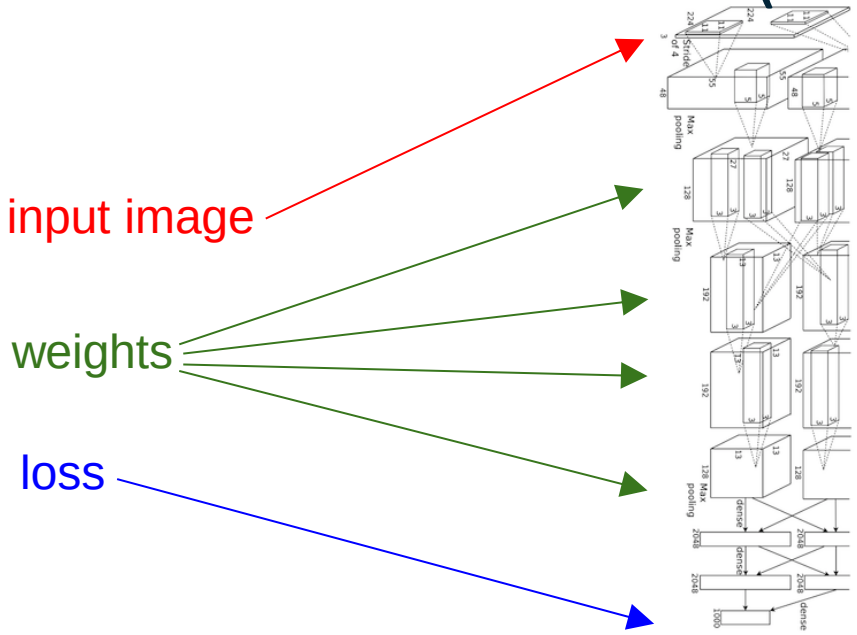
Back-propagation uses the dynamically built graph

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 20))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 20))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()

next_h.backward(torch.ones(1, 20))
```



From pytorch.org

Convolutional network (AlexNet)



Neural Turing Machine

input image

loss

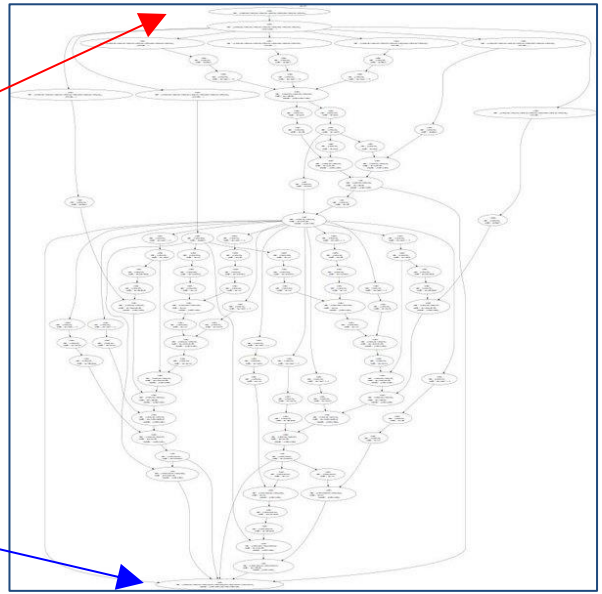
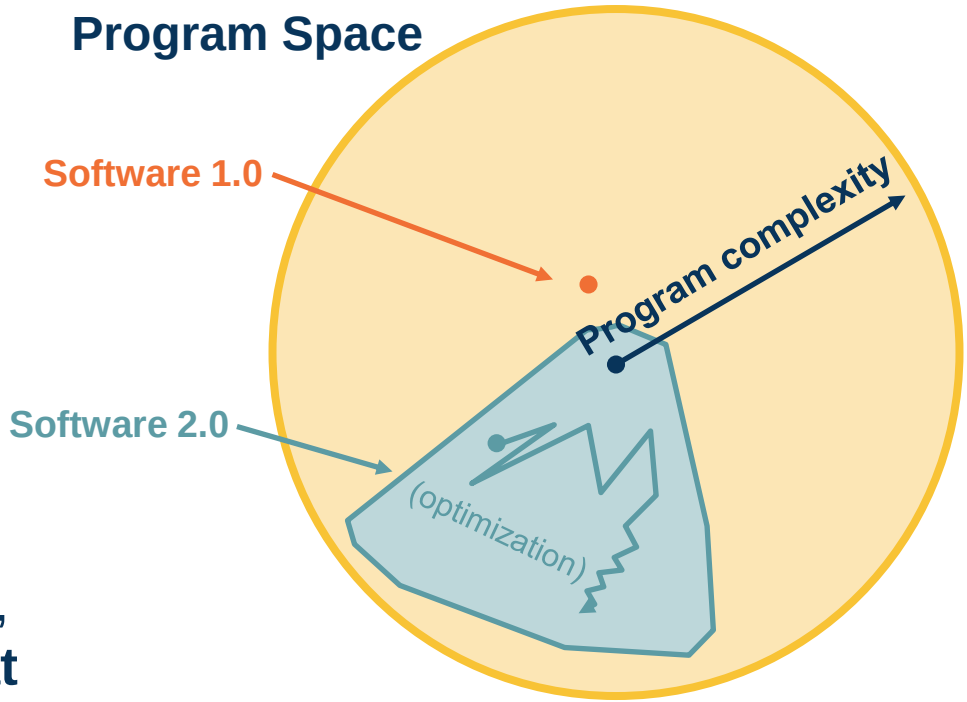


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

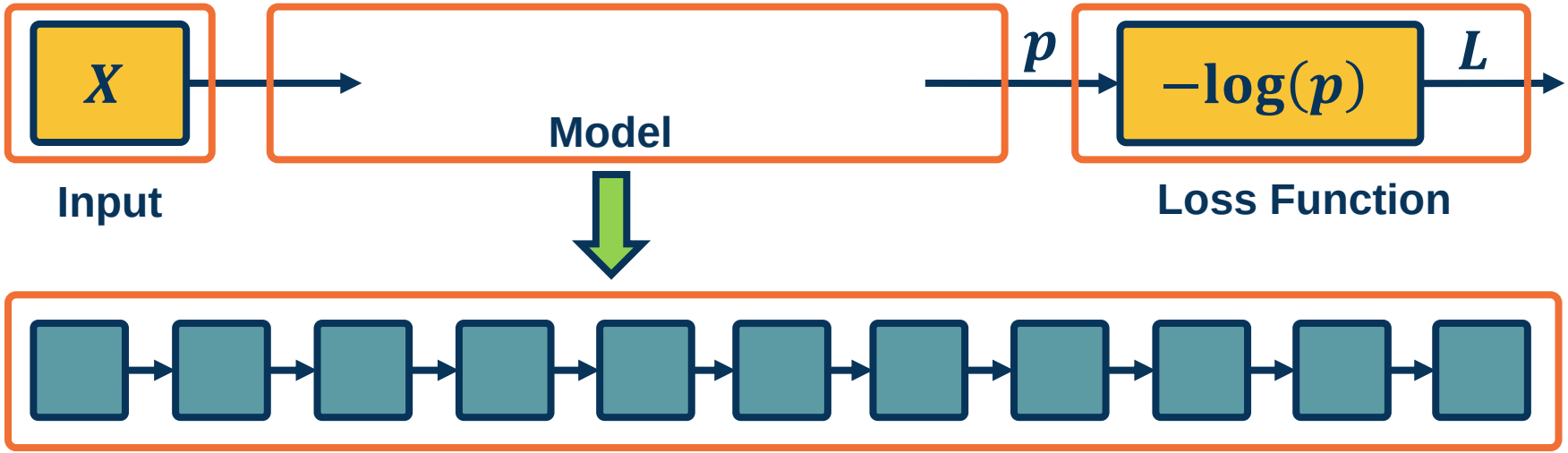
- Computation graphs are **not limited to mathematical functions!**
- Can have **control flows** (if statements, loops) and **backpropagate** through **algorithms**!
- Can be done **dynamically** so that **gradients are computed**, then **nodes are added, repeat**
- **Differentiable programming**



Adapted from figure by Andrej Karpathy

Backpropagation, and automatic differentiation, allows us to optimize **any** function composed of differentiable blocks

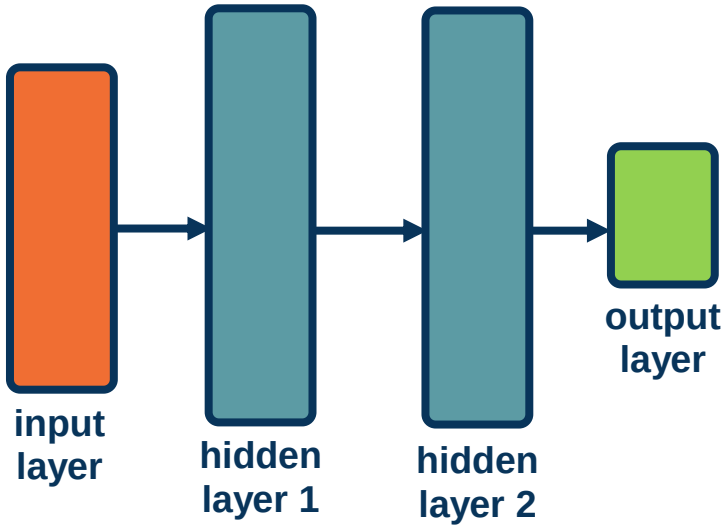
- **No need to modify** the learning algorithm!
- The complexity of the function is only limited by **computation and memory**



A network with two or more hidden layers is often considered a **deep** model

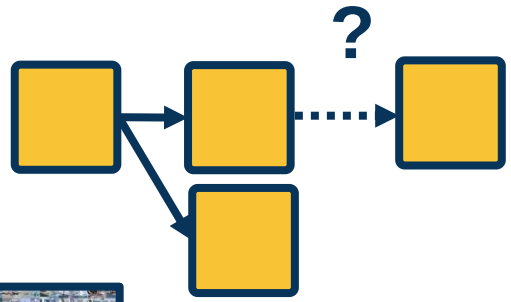
Depth is important:

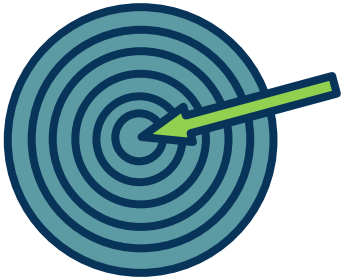
- Structure the model to represent an inherently compositional world
- Theoretical evidence that it leads to parameter efficiency
- Gentle dimensionality reduction (if done right)



There are still many design decisions that must be made:

- **Architecture**
- **Data Considerations**
- **Training and Optimization**
- **Machine Learning Considerations**





Local Minima